



debian

Debian-Pakete bauen
mit
Git-Buildpackage

Dipl.-Ing. Mechtilde Stehmann

und

Dr. Michael Stehmann

13. Oktober 2024

Deutsche Version:



<https://ddp-team.pages.debian.net/dpb/BuildWithGBP.pdf>

Englische Version:



https://ddp-team.pages.debian.net/dpb/en_US/BuildWithGBP.pdf

Inhalt

I. Überblick	1
1. Lizenz	3
2. Zum Titel des Buches	5
3. Wer sollte dieses Buch lesen?	7
4. Wie entsteht dieses Buch?	9
4.1. Motivation	9
4.2. Baustellen	10
4.3. Werkzeuge	10
5. Konventionen	13
5.1. System	13
5.2. Terminologie	13
5.3. Typographie	13
5.4. Darstellung des Quellcodes	13
6. Kurzanleitung	15
6.1. Vorbereitung der Build-Umgebung	15
6.2. Nutzung des Programmskriptes	16
II. Grundlagen	17
7. Lektüre	21
7.1. Debian Free Software Guidelines	21
7.2. Debian Policy-Handbuch	21
7.3. Debian Entwicklerreferenz	21
7.4. Referenz für <i>Git-Buildpackage</i>	22
7.5. Handbuch für Debian-Betreuer	22
7.6. Debian Leitfaden für Neue Paketbetreuer	22
7.7. Weitere Informationen	22
8. Was ist ein Debian-Paket?	23
9. Auswahl der zu paketierenden Software	25

10. Prüfung der Quellen	27
10.1. Lizenzprüfung	27
10.1.1. <i>debmake</i>	27
10.1.2. <i>licensecheck</i>	28
10.1.3. <i>scan-copyrights</i>	28
10.1.4. <i>licensing</i>	28
10.1.5. <i>cme</i>	29
10.1.6. Manuell	29
10.1.7. Stolperfallen	29
10.2. Feststellung der Programmiersprache	30
10.3. Prüfung der Abhängigkeiten	30
10.3.1. Abhängigkeiten ermitteln mit <i>packages.debian.org</i>	30
10.3.2. Abhängigkeiten ermitteln auf <i>codesearch.debian.net</i>	30
10.3.3. Suche nach Abhängigkeiten auf der Konsole	31
10.4. Änderungen am Quellcode	31
10.4.1. Ganze Dateien ausschließen	31
10.4.1.1. Auflistung der auszuschließenden Dateien	31
10.4.1.2. Fallunterscheidungen	32
10.4.1.3. Benamung der Pakete beim Ausschluss von Dateien	32
10.4.2. Änderungen in einzelnen Quellcode-Dateien (Patches)	33
10.4.2.1. Patches mit <i>quilt</i>	33
10.4.2.2. Patches in einem <i>Patch-Queue</i> -Zweig	33
11. Versionierung der Pakete	35
11.1. Paketname	35
11.2. Versionierungsschema	35
11.3. <i>apt</i> und <i>dpkg</i>	36
11.4. <i>uscan</i> und die Datei <i>debian/watch</i>	36
12. <i>dh_make</i>	39
13. Java-Pakete bauen	41
13.1. Herausforderungen	41
13.2. Anwendungen und Bibliotheken	41
13.2.1. Java-Programme paketieren	42
13.2.2. Java-Bibliotheken paketieren	42
13.2.3. Name des Java-Paketes	42
13.3. Abhängigkeiten bei Java-Paketen	42
13.3.1. Weitere Abhängigkeiten feststellen	42
13.3.2. Abhängigkeiten ermitteln	43
13.4. Build-Systeme für Java-Pakete	43
13.4.1. Das Build-System <i>maven</i>	43
13.4.2. Paketieren mit <i>maven</i>	44
13.4.3. Paketieren mit <i>ant</i>	47
13.4.4. Paketieren mit <i>gradle</i>	47
13.5. Java-Pakete bauen ohne Build-System	47

14. Mozilla-Erweiterungen bauen	49
14.1. Quellen der Erweiterungen	49
14.2. Integration ins Dateisystem	50
15. Python-Pakete bauen	51
16. Dokumentation paketieren	53
16.1. Dokumentation für Debian bauen	53
16.2. Upstream-Dokumentation bauen	53
17. Metapakete bauen	55
17.1. Kein Upstream-Quellcode	55
17.2. Natives Debian-Paket	55
17.2.1. <i>debian/source/format</i>	55
17.2.2. <i>debian/control</i>	55
17.2.3. <i>debian/rules</i>	55
17.2.4. <i>debian/changelog</i>	55
18. Konfiguration zur Installation	57
18.1. <i>debconf</i>	57
18.2. <i>dbconfig-common</i>	57
19. System einrichten	59
19.1. Abhängigkeiten für das Programmskript	59
19.1.1. Generelle Abhängigkeiten	59
19.1.2. Abhängigkeiten für das Bauen von Java-Paketen	60
19.1.3. Abhängigkeiten für Erweiterungen, die <i>Zip</i> -Archive sind	60
19.2. Verzeichnisse und Dateien	60
19.2.1. Pfade zu den Projekten	60
19.2.2. Konfigurationsdateien	61
19.2.2.1. Für jedes Projekt	61
19.2.2.2. Für viele Projekte	62
19.2.2.3. Fingerprint des Maintainer-Schlüssels	62
19.2.3. <i>.bashrc</i>	62
19.3. <i>PBuilder</i> einrichten	62
19.3.1. <i>Chroot</i>	63
19.3.2. Konfiguration des <i>Pbuilders</i>	63
19.3.3. Hooks einrichten	67
19.3.4. Hooks - Beispiele	68
19.3.4.1. Hook A	68
19.3.4.2. Hook B	68
19.3.4.3. Hook C	69
19.3.4.4. Hook D	69
19.3.4.5. Hook E	69
19.3.4.6. Hook F	70
19.3.4.7. Hook G	70
19.3.4.8. Hook H	70

19.3.4.9. Hook I	70
19.3.5. Alternative <i>Chroot</i> -Umgebungen	71
19.4. Konfiguration von <i>sbuild</i>	71
19.5. Weitere <i>Chroot</i> -Systeme	72
19.6. <i>Quilt</i> fürs Patchen einrichten	72
20. Git einrichten	75
20.1. Branches	75
20.2. Mergen	75
20.3. <i>gbp.conf</i>	75
20.3.1. Reihenfolge	76
20.3.2. Abschnitte in der <i>gbp.conf</i>	76
20.3.3. Syntax der Optionen	76
20.3.4. Beispiel	77
20.4. Git-Repositoryn auf eigener Infrastruktur	78
20.4.1. Lokales Git-Repository	78
20.4.2. Eigener Git-Server	78
21. Salsa-Repositoryn	79
21.1. <i>Salsa</i> -Account anlegen	79
21.2. Anlage eines <i>Salsa</i> -Repositorys	79
21.3. <i>Salsa</i> -Repository für das Java-Team	80
21.3.1. Quelle des Skripts	80
21.3.2. Abhängigkeiten	80
21.3.3. Zugangstoken beschaffen	80
21.3.4. Token eintragen	81
21.3.5. Skript aufrufen	81
21.4. Aufgaben auf <i>salsa.debian.org</i>	82
21.4.1. Merge Request	82
22. Paketieren jenseits vom Zweig <i>Unstable</i>	83
22.1. Security-Updates	84
22.2. (Old-)Stable-Proposal	84
22.2.1. Fehlerbericht	85
22.2.2. Anforderungen an einen Patch	86
22.2.3. Abhängigkeiten zu <i>Mozilla</i> -Paketen	86
22.3. Stable-Backports	86
22.4. Backports-Repository	87
22.5. Experimental	87
22.6. Backporten fremder Pakete	87
22.7. Versionierung	87
23. Zum Start eine E-Mail	89
23.1. ITP - Intent To Package	89
23.2. RFP - Request For Package	90
23.3. ITA - Intent To Adoption	91

23.4. RFA - Request for Adoption	91
23.5. RFH - Request For Help	91
23.6. O - Orphaned	91
23.7. RFS - Request For Sponsor	91
24. Änderungen am Fehlerbericht	93
24.1. Anpassung des Schweregrades	93
24.2. Anpassung des Titels	93
24.3. Änderung des Maintainers	93
24.4. Öffnen eines geschlossenen Fehlerberichtes	94
24.5. Schließen eines Fehlerberichtes	94
24.6. <i>usertags</i> hinzufügen	94
25. Reportbug einrichten	95
26. Schwierigkeiten überwinden	97
26.1. Ein Paket loseisen	97
26.1.1. Beantragung einer Entsperrung	97
26.2. Releasekritische Fehler beheben	98
26.3. Paket aus Repositorien entfernen	98
26.3.1. Feststellung der Rückwärtsabhängigkeit	98
26.3.2. Fehlerbericht	99
27. Autopkgtest	101
28. Lintian-Meldungen	103
29. Reproduzierbare Builds	105
29.1. reprotect	105
30. piuparts	107
III. Wie ein Shell-Skript hilft, ein Debian-Paket zu bauen	109
31. Erste Schritte im Programmskript	111
31.1. Der Anfang steht am Schluss	112
31.2. Und das sieht der Nutzer als Erstes	112
31.3. Projektname abfragen	114
31.4. Weiterer Ablauf	116
32. Anlegen eines neuen Projektes	117
32.1. Konfigurationsdatei erstellen	117
32.1.1. Abfrage allgemeiner Variablen für die Konfigurationsdatei	119
32.1.2. Abfrage spezieller Variablen für die Konfigurationsdatei	130
32.1.2.1. Ermittlung der Plugin-Pfade	130
32.1.2.2. Variablenabfrage für Java-Pakete	131
32.1.2.3. Variablenabfrage für Mozilla-Erweiterungen	133

32.1.2.4. Variablenabfrage für Python3-Pakete	134
32.1.3. Speichern der Konfiguration	136
32.1.4. Beispiel einer Konfigurationsdatei	138
32.2. Anlegen der Infrastruktur	138
32.2.1. Anlegen der notwendigen Verzeichnisse	138
32.2.2. Definition der Pfade	139
32.2.3. Anlegen der Log-Datei	139
32.3. Git-Repositoryn	140
32.3.1. Gibt es bereits ein Git-Repository?	140
32.3.2. Auswahldialog	141
32.4. Neuanlage eines lokalen Git-Repositorys	144
32.4.1. Git-Repository anlegen	144
32.4.2. Name und E-Mail-Adresse ins Git-Repository einfügen	145
32.4.3. Repository auf <i>salsa.debian.org</i>	158
32.4.3.1. Manuell	158
32.4.3.2. Innerhalb des Java-Teams	159
32.4.4. Remoteserver anzeigen	159
32.5. Klonen von <i>salsa.debian.org</i>	160
32.5.1. Bestimmung der Git-Zweige	161
32.5.2. Git-Zweige ermitteln	164
32.5.3. Git-Zweig Distribution zuordnen	165
32.5.4. Name und E-Mail-Adresse hinzufügen	166
32.6. Import eines Debian-Quellcode-Paketes	168
32.7. Import für das Sponsoring	170
32.8. GnuPG-Schlüssel verfügbar?	172
32.9. Fingerprint nutzen	173
32.10 Start des Paketierens	176
33. Arbeiten in einem angelegten Projekt	177
33.1. Konfigurationsdatei laden und editieren	177
33.2. Ändern von Zeilen in der Konfigurationsdatei	181
33.3. Zeile in Konfigurationsdatei einfügen	182
33.4. Auswahl eines Git-Zweiges	182
33.4.1. Prüfung mit <i>git status</i>	183
33.4.2. Fehlermeldung und -behebung	184
33.4.3. Auswahl der Debian-Banches	185
33.4.4. Dialog zur Auswahl eines Branches	186
33.4.5. Eintrag ändern	187
33.4.6. Konfiguration einlesen	189
33.4.7. Kein oder nur ein Branch existiert	189
33.5. Aufgabenauswahl	191
34. Bauen einer neuen Version	195
34.1. Änderungen von <i>Salsa</i> herunterladen	195
34.2. Importieren einer vorhandenen <i>Patch-Queue</i>	197
34.2.1. Erster Importversuch	198

34.2.2. Erneuter Importversuch	199
34.2.3. Import im PQ-Branch erfolgreich	200
34.3. Werkzeuge zum Herunterladen der Upstream-Sourcen	202
34.4. Herunterladen auf klassische Weise	206
34.4.1. Archiv-Formate	206
34.4.2. Herunterladen des Quellcodes	206
34.4.2.1. Herunterladen	208
34.4.2.2. Kopieren des Quellarchivs	210
34.4.3. Komprimierung ermitteln	212
34.4.4. Upstream-Version ermitteln	214
34.4.5. Dateien aus Upstream-Archiv ausschließen	217
34.4.6. Debian-Quellcode-Datei erzeugen	226
34.4.7. Signatur prüfen	229
34.4.7.1. Signatur-Datei herunterladen	229
34.4.7.2. Prüfung der Signatur	230
34.4.8. Link durch Kopie ersetzen	231
34.4.9. <i>gbp</i> -Konfigurationsdatei	232
34.4.10. Prüfung des Git-Repositorys	241
34.4.11. Import nach Git	243
34.5. Herunterladen und Importieren mit <i>uscan</i>	248
35. Bauen einer neuen Revision	253
35.1. Anlegen des Debian-Verzeichnisses	254
35.2. Abfrage: Bauen mit <i>mh-make</i> ?	255
35.3. Sollen die Debian-Dateien angezeigt werden?	256
35.4. Dateien im Verzeichnis <i>debian/</i>	257
35.4.1. Anzeigen der Debian-Dateien	257
35.4.2. <i>debian/source/format</i>	260
35.4.3. <i>debian/source/include.binaries</i>	262
35.4.4. <i>debian/upstream/metadata</i>	262
35.4.5. <i>debian/copyright</i>	264
35.4.6. <i>debian/control</i>	265
35.4.6.1. Grundlegender Aufbau	265
35.4.6.2. Anpassungen für Java-Pakete	267
35.4.6.3. Web-Extension-Plugin	268
35.4.6.4. Python-Plugin	268
35.4.7. <i>debian/watch</i>	269
35.4.8. <i>debian/rules</i> - Grundlegender Aufbau	273
35.4.8.1. Erstellen der Datei	274
35.4.8.2. Export von Variablen	274
35.4.8.3. Aufruf der Debhelper	275
35.4.8.4. <i>debian/rules</i> - overrides	276
35.4.8.5. Schluss der Funktion	276
35.4.9. <i>salsa-ci.yml</i>	277
35.4.10. <i>debian/javabuild</i>	278
35.4.11. <i><Paketname>.install</i>	278

35.4.12.<Paketname>.dirs	279
35.4.13.<Paketname>.docs	279
35.4.14.<Paketname>.links	280
35.4.15.Paketname.desktop	280
35.4.16.<Paketname>.manpages	280
35.4.17.<Paketname>.examples	281
35.4.18.README.Debian	282
35.4.19.README.source	282
35.4.20.source/lintian-overrides	282
35.5. Überprüfung der Dateien in <i>debian/</i> mit CmeFix	282
36. Änderungen am Upstream-Code vornehmen	285
36.1. Arbeiten mit <i>gbp pq</i>	288
36.1.1. Erstellen eines Patch-Queue-Zweiges	288
36.1.2. Manuelle Bearbeitung	292
36.1.3. Hinweise zur Fehlerbehebung	292
36.1.4. Aktualisieren des Patch-Queue-Zweiges	292
36.1.5. Hinweise zur Bereinigung der Patch-Queue	294
36.1.6. Import vorhandener Patches	294
36.1.7. Bearbeiten des Quellcodes	296
36.1.8. Export der Patches	298
36.2. Auswahl der Patches	300
36.3. Nutzung von Quilt	302
36.3.1. Patch erstellen	305
36.4. Neuen Patch erstellen	307
36.5. Datei zum Patchen auswählen	310
36.5.1. Patch löschen	313
36.5.2. Ausgangszustand wiederherstellen	316
36.6. Patch editieren	318
36.7. Patch bearbeiten	319
37. Bauen	321
37.1. <i>debian/changelog</i>	321
37.1.1. Versionsbezeichnung einfügen	323
37.2. Verschieben der <i>gbp</i> -Konfigurationsdatei	331
37.3. Parameter für <i>gbp buildpackage</i> festlegen	331
37.3.1. Git-Zweig und Distribution ermitteln	332
37.3.2. Git-Zweig und Distribution prüfen	335
37.3.3. Git-Zweig anpassen	337
37.3.4. Distribution ermitteln	337
37.3.5. Auswahl des Build-Systems	338
37.3.6. Überprüfung der Parameter	339
37.3.7. Letzte Ausstiegsmöglichkeit	339
37.4. Was macht <i>sbuild</i> ?	341
37.5. In der <i>Pbuilder-Chroot</i> bauen	343
37.5.1. <i>base.cow</i> erstellen	343

37.5.2. git-pbuilder update	345
37.5.3. Aufnahme des *.orig-Archives in *.changes	346
37.5.4. Bauen mit <i>gbp buildpackage</i>	351
38. Wenn das Bauen fehlschlägt	353
39. Bauen jenseits von <i>Unstable (sid)</i>	355
39.1. Bauen für bereits offiziell freigegebene Distributionen	355
39.2. Proposed-Updates – Besonderheiten	356
40. Überprüfungen	357
40.1. Auswahl der Changes-Datei	359
40.2. Yamllint	360
40.3. Prüfung mit Lintian	360
40.4. Uscaan	362
40.5. Überprüfen der Datei <i>debian/copyright</i>	365
40.6. Überprüfen mit <i>debdiff</i> und <i>diffoscope</i>	365
40.6.1. <i>debdiff</i>	365
IV. Veröffentlichen	371
41. Vorbereitungen zum Hochladen	373
41.1. Existiert <i>debian/changelog</i> ?	373
41.2. <i>debian/changelog</i> fertigstellen	375
41.3. Nochmaliges Bauen?	379
41.4. Soll ein <i>Debdiff</i> erstellt werden?	380
42. Hochladen auf Git-Repositoryen	383
42.1. Hochladen nach salsa.debian.org	383
42.2. Hochladen auf eigenen Git-Server	386
43. Paket(e) hochladen	387
43.1. Auswahl des Zielrepositoriums	388
43.2. Signatur erzeugen	391
43.3. Hochladen mit <i>dput</i>	392
43.4. Nach FTP-Master hochladen	393
43.4.1. Zurückweisung eines Paketes	398
43.5. Nach mentors.debian.net hochladen	398
43.6. Als <i>Non-Maintainer-Upload</i> hochladen	399
43.7. Hochladen nach <i>people.debian.org</i>	400
43.8. Lokales Repositoryum	403

V. Weitere Bestandteile des Skriptes	405
44. Weitere Aufgaben	407
44.1. Neuen Branch erstellen	407
44.2. Eingabe des Namens oder der IP eines eigenen Git-Servers	408
44.3. AddGitServer	409
45. Kopf des Skriptes	411
45.1. Shebang	411
45.2. Copyright-Vermerk	411
45.3. Abhängigkeiten für das Programmskript	412
45.4. <i>set -e</i> und Funktionsheader	412
45.5. Funktion zur Fehlersuche	413
VI. Plugins und Skripte	415
46. Java-Plugin	417
46.1. Anpassungen für Java-Pakete	417
47. Maven-Plugin	419
47.1. Kopf des Maven-Plugins	419
47.2. Mitteilung	420
47.3. Bauen mit Maven	420
47.4. Maven-Dateien bearbeiten	429
47.4.1. <i>maven.rules</i>	429
47.4.2. <i>maven.ignoreRules</i>	429
47.4.3. <i>maven.properties</i>	430
47.4.4. <i>Paketname.poms</i>	431
47.4.5. <i>README.source</i>	432
47.5. <i>debian/rules</i> - Ergänzungen für Java-Pakete mit <i>Maven</i>	432
48. Web-Extension-Plugin	433
48.1. Kopf des Webext-Plugins	433
48.2. Erstellen der <i>webext*.*</i> -Dateien in <i>debian/</i>	434
48.2.1. Ermittlung des Namens der <i>*.xpi</i> -Datei	434
48.2.2. <i>debian/rules</i> - Ergänzungen für Mozilla-AddOns	435
48.2.3. <i>debian/control</i> - Ergänzungen für Mozilla-AddOns	437
48.2.4. <i>debian/webext-*.install</i>	437
48.2.5. <i>debian/webext-*.docs</i>	437
48.2.6. <i>debian/webext-links-tb</i>	438
48.2.7. Meldung: Webext-Plugin geladen	438
49. Python-Plugin	439
49.0.1. Meldung: Python-Plugin geladen	439
49.1. Anpassungen für Python-Pakete	440
49.2. <i>debian/control</i> - Ergänzung für Python-Pakete	441

50. Skripte	443
50.1. Anlage eines Projektes im Java-Team	443
50.2. Skript zum Extrahieren der Dokumentation im PDF- und Epub-Format .	446
50.2.1. Abhängigkeiten	446
50.2.2. Ablauf	446
50.3. Skript zum Extrahieren der Skripte	449
50.4. <i>gitlab-ci.yml</i> für die Salsa-CI	451
VII. Anhang	A–i
Abbildungsverzeichnis	A–iii
Literaturverzeichnis	A–viii
Stichwortverzeichnis	A–xiii

Teil I.
Überblick

1. Lizenz

Der Text des Buches „Debian-Pakete bauen mit *Git-Buildpackage*“ von Mechtilde und Michael Stehmann steht unter der **Creative Commons Namensnennung - Weitergabe unter gleichen Bedingungen 4.0 International Lizenz (CC BY-SA 4.0)**[1].

Das beschriebene Programmskript steht unter der **GNU General Public License Version 3** oder nach Ihrer Wahl einer späteren Version[2].

Copyright: © 2012-2024 Mechtilde Stehmann (E-Mail: mechtilde@debian.org),
Michael Stehmann (E-Mail: michael@canzeley.de)

2. Zum Titel des Buches

Was ein **Debian-Paket** ist, wird in Kapitel 8 (Seite 23) beschrieben. Vorab sei verraten: Ein **Debian-Paket** ist nicht nur ein Archiv im *deb*-Format, das Binär-Dateien enthält.

git-buildpackage ist ein **Debian-Paket**, welches nützliche Werkzeuge enthält, **Debian-Pakete** aus einem **Git**-Repository zu bauen. Einige dieser Werkzeuge werden vom Programmskript, das in diesem Buch beschrieben wird (Kapitel 31, Seite 111), verwandt (s. Kapitel 19, Seite 59).

Die Namen der Werkzeuge in diesem Paket fangen mit *gbp* an. Ein wichtiges Werkzeug ist *gbp buildpackage*.

3. Wer sollte dieses Buch lesen?

Die Ausführungen in diesem Buch sind besonders für die Nutzer des Skriptes interessant. Aber auch Menschen, die sich allgemein für das Paketieren für eine Distribution interessieren, werden in diesem Buch Informationen finden können.

Dieses Buch will kein „Lehrbuch“ des **Debian**-Paketbaus sein. Es ist eher ein Erfahrungsbericht, wobei die Erfahrungen „in Code gegossen“ worden sind.

Das Buch beschreibt, wie **Debian**-Pakete auf der Basis eines **Git**-Repositoriums mit den Programmen aus dem Paket *git-buildpackage* [3] und anderen nützlichen Befehlen erstellt werden. Am Ende sollte der Leser in der Lage sein, mit der Hilfe des dargestellten Programmskripts und der Beschreibungen der einzelnen Schritte „veröffentlichungsreife“ **Debian**-Pakete zu bauen.

Das Programmskript selbst baut **keine Debian**-Pakete, sondern unterstützt den Nutzer beim Bauen derselben. Es ist nur ein Assistenzprogramm.

Dieses Buch kann auch dazu dienen, Probleme zu verstehen, die beim Paketieren von **Debian**-Paketen auftreten können.

Paketieren ist im Grunde nicht schwierig. Es gibt aber immer wieder neue Herausforderungen. Paketieren macht daher Spaß.

4. Wie entsteht dieses Buch?

4.1. Motivation

Was treibt uns, ein solches Buch zu schreiben?

Dazu muss man folgendes wissen:

Beim Paketieren werden viele Befehle in einer sinnvollen Reihenfolge in der Shell ausgeführt. Außerdem müssen viele kleine Dateien gepflegt und eingebunden werden. Kleinste Fehler und Ungenauigkeiten führen meist dazu, dass das Paket nicht korrekt gebaut werden kann. Auch ist es aufwändig und fehlerträchtig, immer wieder die korrekten Optionen einzusetzen.

Um diese Fehlerquellen möglichst klein zu halten, wurden diese Schritte in einem Shell-Skript zusammengefasst. Im Laufe der Zeit und mit jedem weiteren Paket wächst dieses Skript und wird auch immer weiter verfeinert. Daraus ist bisher schon ein umfangreiches Programmskript geworden.

Als Mechtilde anfang, sich mit dem Bauen von **Debian**-Paketen zu beschäftigen, stand die Frage im Raum, wie sich diese vielen Schritte dokumentieren und automatisieren lassen. Zur Automatisierung ist dann dieses Shell-Skript entstanden. Der Bedarf an Dokumentation konnte von Anfang an nicht mit Kommentaren – weder in den einzelnen Dateien, noch in diesem Programmskript – gedeckt werden.

Deshalb haben wir auch schon früh angefangen, unsere Schritte beim Paketieren ausführlich festzuhalten. Ein besonderes Augenmerk haben wir auf Beschreibungen gelegt, welche getroffene Entscheidungen nachvollziehbar und überprüfbar machen. Dies erleichtert notwendige Veränderungen.

Daher haben wir auch soweit wie möglich bei der Angabe von Optionen die Langform gewählt. Dies erleichtert die Lesbarkeit.

Die Dokumentation soll also sowohl das eigentliche Paketieren beschreiben, als auch das Skript erläutern.

Die **Debian**-Distribution ist das Werk vieler Menschen. Sie besteht aus mehreren zehntausenden Paketen. Das Bauen der Pakete ist eine wesentliche Aufgabe der Paketbetreuer. Viele Paketbetreuer nutzen hierzu eigene Skripte. Die Veröffentlichung eines solchen Skriptes ist daher ein Wagnis. Wenn unser Skript einigen Paketbetreuern das Leben erleichtert und Neulinge an das Paketbauen heranzuführt, hat sich dieses Wagnis gelohnt.

Das beschriebene Programmskript bezieht sich nicht auf ein bestimmtes **Debian**-Paket. Vielmehr sollen damit generell einfache **Debian**-Pakete gebaut werden können.

Es werden die Schritte beschrieben, die wir für das Paketieren der von Mechtilde betreuten Pakete benötigen. Das Programmskript erhebt nicht den Anspruch, man könne damit aus jedem Quellcode ein **Debian**-Paket bauen.

An vielen Stellen kann und muss man auch manuell eingreifen. Hierzu soll die Beschreibung der Vorgänge beim Paketieren eine Hilfe sein.

Dass das Programmskript die Möglichkeit manueller Eingriffe voraussetzt, macht es erforderlich, dass das Programmskript immer wieder prüft, ob die Voraussetzungen, von denen die Verfasser ausgegangen sind, auch tatsächlich vorliegen. Diese Notwendigkeit erhöht leider den Umfang und die Komplexität des Skriptes und damit auch den Umfang des Buches.

4.2. Baustellen

Das Buch und das Skript sind immer noch „Baustellen“, weil immer wieder neue Erfahrungen einfließen.

Das Buch ist in deutscher, die Oberfläche des Programmskripts in englischer Sprache verfasst. Lokalisierungen sind willkommen. Als „Proof-of-Concept“ wurde bereits ein Teil des Buches in die englische Sprache übersetzt.¹

Die Veröffentlichung des Quelltextes erfolgt im vom Debian-Projekt bereitgestellten Git-Repositorium². Dort ist die CI/CD (Kapitel 50.4, Seite 451) aktiviert. Daher steht das Buch als *.pdf³ und *.epub⁴ zur Verfügung. Dort können auch die Programmskripte heruntergeladen werden⁵

4.3. Werkzeuge

„Schöner Leben mit Dokumentation“ ist ein geflügeltes Wort in unserer *Peergroup*.

Mit welchen Werkzeugen lässt sich eine solche Dokumentation erstellen? Stehen diese Werkzeuge auch als Debian-Pakete zur Verfügung?

Einen wichtigen Hinweis dazu hat Mechtilde auf einer Veranstaltung zum Software Freedom Day 2012⁶ in Köln erhalten. Dort hat sie die Möglichkeiten von noweb [4]⁷ kennengelernt. Dies bedeutete auch, sich mit L^AT_EX näher zu beschäftigen.

In dieser Kombination ist es möglich, Code und dessen Beschreibung in *einem* Dokument zu pflegen. Donald Knuth bezeichnet dies als „Literate Programming“⁸

Weiter haben wir uns damit beschäftigt, dass sich mit L^AT_EX neben PDF-Dokumenten auch Dokumente im EPUB-Format erstellen lassen. Diese können auch mit einem E-Book-Reader gelesen werden. (Kapitel 50.2, Seite 446)

Dieses Buch in eine andere Sprache zu übersetzen, stellt eine besondere Herausforderung da. Unsere Tests haben ergeben, dass OmegaT⁹ insoweit ein nützliches und komforta-

¹https://ddp-team.pages.debian.net/dpb/en_US/BuildWithGBP.pdf

²<https://salsa.debian.org/ddp-team/dpb>

³<https://ddp-team.pages.debian.net/dpb/BuildWithGBP.pdf>

⁴<https://ddp-team.pages.debian.net/dpb/BuildWithGBP.epub>

⁵<https://ddp-team.pages.debian.net/dpb/build-gbp.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-maven-plugin.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-webext-plugin.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-python-plugin.sh>

<https://ddp-team.pages.debian.net/dpb/build-gbp-java-plugin.sh>

⁶https://sfd.koelnerlinuxtreffen.de/SFD2012/2012Robert_Stanowsky.html

⁷s.a. <https://de.wikipedia.org/wiki/Noweb>

⁸<http://www.literateprogramming.com/>

⁹<https://packages.debian.org/sid/omegat>

bles Werkzeug darstellt. Der entsprechende Prozess wird in einem separaten Büchlein dokumentiert ¹⁰.

Das Literaturverzeichnis wird mit *jabref* erstellt und gepflegt. Die so erstellte Datei kann in das L^AT_EX-Dokument eingebunden werden.

Als Editor wird *geany* mit dem Plugin *geany-plugin-latex* verwendet.

Git ist genial. Das Bauen erfolgt daher mit den Werkzeugen aus dem Paket *git-buildpackage*¹¹ (s. Kapitel 19.1, Seite 59).

Die Menschen bei **Debian** haben viele nützliche Programme geschaffen, die das Bauen von **Debian**-Paketen erleichtern und vereinheitlichen. Das dargestellte Programmskript wurde daher „auf den Schultern von Riesen“ geschaffen.

Es dient dazu, die eingesetzten Hilfsprogramme in zweckmäßiger Reihenfolge aufzurufen und mit sinnvollen Optionen zu versehen. Es soll seinen Nutzern den Weg weisen und ihnen die Arbeit erleichtern. Um seine Anpassung an die Bedürfnisse seiner Nutzer zu erleichtern, ist es ein Shellskript.

¹⁰<https://oldmike.pages.debian.net/omegatbooklet/omegat.pdf>

¹¹<https://packages.debian.org/sid/git-buildpackage>

5. Konventionen

Einige Hinweise zum besseren Verständnis des Buches:

5.1. System

Das Buch und besonders das Programmskript wurden auf einer *64-Bit-PC*-Architektur erstellt. Diese wird bei **Debian** als *amd64* bezeichnet. Eine weitere Bezeichnung für dieses System ist *x86-64*.

5.2. Terminologie

Ein **neues Paket** ist ein Paket, welches das Programmskript noch nicht kennt. D.h. zu diesem Paket existiert noch keine Konfigurationsdatei.

Eine **neue Version** ist eine neue Upstream-Version. Dem Bauen einer neuen Version folgt der Bau einer neuen Revision.

Eine **neue Revision** bezeichnet ein neu hochzuladenes **Debian**-Paket.

5.3. Typographie

Alle Programmnamen sind in *kursiv* gesetzt. Alle Eigennamen sind in **nicht-proportionaler** Schrift gesetzt. Hochgestellte Zahlen weisen auf die Fußnoten auf der gleichen Seite hin. Zitate auf ein Gesamtdokument weisen in eckigen Klammern [] direkt auf das Literaturverzeichnis.

Alle Optionen der Shell-Kommandos werden in der Langform angegeben, soweit dies möglich ist. Dies erhöht die Lesbarkeit.

Für die verwendeten Abkürzungen wird auf die Einträge im Glossar ¹ verwiesen.

5.4. Darstellung des Quellcodes

Die Darstellung des Quellcodes erfolgt in Teilstücken (sogenannten Code Chunks). Die Reihenfolge dieser Code-Teile im Buch entspricht oft nicht der Reihenfolge in den Skripten. Dass die Reihenfolge in Buch und Skript nicht übereinstimmen muss, ist ein Vorzug von **noweb**.

¹<https://wiki.debian.org/Glossary>

6. Kurzanleitung

Dieser Abschnitt soll dem Nutzer helfen, die benötigten Schritte nachzuvollziehen und deren Beschreibung schneller im Buch zu finden.

6.1. Vorbereitung der Build-Umgebung

Bis aus dem Buch und den Skripten ein Debian-Paket entsteht, sind die folgenden Schritte notwendig,

Das Programmskript selbst baut **keine** Debian-Pakete, sondern unterstützt den Nutzer beim Bauen derselben. Es ist nur ein Assistenzprogramm.

1. Herunterladen von *salsa.debian.org* Der Quelltext dieses Buches und die folgenden beiden Skripte finden sich unter `https://salsa.debian.org/ddp-team/dpb`.
2. Installieren der Abhängigkeiten, um das Buch und die Build-Skripte lokal zu generieren. (Kapitel 50.2.1, Seite 446)
3. Generieren des PDF mit `./create-book.sh` (Kapitel 50.2, Seite 446)
4. Mit `./create-buildscript.sh` wird das Programmskript generiert (Kapitel 50.3, Seite 449).
5. **Alternativ:** Herunterladen der Dateien von
 - `https://ddp-team.pages.debian.net/dpb/BuildWithGBP.pdf`
 - `https://ddp-team.pages.debian.net/dpb/create-book.sh`
 - `https://ddp-team.pages.debian.net/dpb/create-buildscript.sh`
6. Erstellen von Symlinks auf die generierten Skripte unter `/usr/local/bin` Damit liegen die jeweils aktuellen Skripte auch in einem Programmpfad (*PATH*) und können überall ohne Pfadangabe aufgerufen werden.
7. Installation aller Abhängigkeiten, die das Programmskript benötigt. (Kapitel 45.3, Seite 412)
8. Anlegen der benötigten Verzeichnisse und Dateien (Kapitel 19.2, Seite 60)
 - Verzeichnis für die Konfigurationsdateien `~/.debian_project/` (Kapitel 19.2.2, Seite 61)
 - Anlegen des Projektverzeichnisses und der Unterverzeichnissen (Kapitel 19.2.1, Seite 60)
 - Eintragungen in die Datei `~/.bashrc` (Kapitel 19.2.3, Seite 62)
 - Anlegen der Datei `gpb.conf` (Kapitel 20.3, Seite 75)
 - Bei Nutzung des *pbuilder* folgt die Konfiguration (Kapitel 19.3.2, Seite 63) und Einrichtung der Hooks (Kapitel 19.3.3, Seite 67)

6.2. Nutzung des Programmskriptes

1. Bereitstellen des GPG-Keys, um Git-Tags zu signieren
2. Ausführen von *build-gbp.sh* (Kapitel 31.1, Seite 112)
3. Mit dem Projektnamen die Konfigurationsdatei erstellen bzw. laden (Kapitel 31.3, Seite 114).
4. Beschaffen des Upstream-Quellcodes (Kapitel 34, Seite 195)
5. Anlegen bzw. Aktualisieren der Dateien im Verzeichnis *debian/* (Kapitel 35, Seite 253)
6. Änderungen am Quellcode vornehmen (Kapitel 36, Seite 285)
7. Bauen des **Debian**-Paketes (Kapitel 37, Seite 321)
8. Überprüfen der Qualität des gebauten **Debian**-Paketes (Kapitel 40, Seite 357)
9. Veröffentlichen des **Debian**-Paketes (Kapitel 41, Seite 373)

Das Programmskript ist modular aufgebaut, sodass man an vielen Stellen „aussteigen“ und später wieder „einsteigen“ kann. Es ermöglicht auch manuelle Eingriffe.

Vor dem Arbeiten mit dem Programmskript sollte auch in folgenden Teil hineingeschaut werden.

Teil II.

Grundlagen

13. Oktober 2024

In diesem Teil des Buches gibt es zunächst ein wenig Theorie. Sodann wird die Einrichtung des Systems einschließlich des Git-Repositorys, welches für *git-buildpackage* essentiell ist, beschrieben.

7. Lektüre

Manche fragen, was sie gelesen haben sollten, bevor sie mit dem Bauen von Debian-Paketen beginnen. Andere wollen wissen, wo sie hilfreiche Informationen finden können. Daher gibt es hier Leseempfehlungen für alle, die Debian-Pakete bauen (wollen).

Eine kurze Einführung in das Thema „Paketieren für Debian“ enthält das **Simple Packaging Tutorial**[5].

Die folgenden drei Dokumente gehören zur „Pflichtlektüre“ eines jeden Paket-Betreuers.

7.1. Debian Free Software Guidelines

Allen, die bei Debian mitwirken wollen, wird die Lektüre des Gesellschaftsvertrages empfohlen. Die **Debian Free Software Guidelines (DFSG)**[6] sind wesentlicher Bestandteil des Gesellschaftsvertrages. Sie enthalten die Bedingungen, die eine Lizenz erfüllen muss, um als „frei“ zu gelten. Diese Bedingungen sind bereits bei der Auswahl der zu paketierenden Software bedeutsam (Kapitel 10, Seite 27).

Die **Debian Free Software Guidelines (DFSG)** gelten nicht nur für Software-Lizenzen, sondern gemäß Ziffer 1 der Version 1.1 des Gesellschaftsvertrages („alle Komponenten“) auch für die Lizenzen von Bildern, Tönen, Texten etc.

7.2. Debian Policy-Handbuch

Das **Debian Policy-Handbuch**[7] beschreibt die Richtlinien für die Distribution **Debian GNU/Linux**. Beschrieben werden die Struktur und der Inhalt des **Debian-Archivs**, verschiedene Design-Entscheidungen des Betriebssystems und technische Anforderungen, die jedes Paket erfüllen muss, um in die Distribution aufgenommen zu werden. Dieses Dokument steht nur in englischer Sprache zur Verfügung.

Es steht auch als Debian-Paket *debian-policy* bereit.

Der **Filesystem Hierarchy Standard**[8] ist eine wichtige Ergänzung zum Policy-Handbuch.

Daneben gibt es ergänzend noch weitere Regelwerke.¹

7.3. Debian Entwicklerreferenz

In der **Entwicklerreferenz**[9] werden die Verfahren und Ressourcen für **Debian-Entwickler** aufgeführt. Das Dokument beschreibt, wie man ein neuer Entwickler für das **Debian-Projekt** wird, die Upload-Prozedur, wie die Fehlerdatenbank (Bug-Tracking-System), die Mailinglisten, Internet-Server etc. bedient werden.

¹<https://www.debian.org/devel/index.en.html>

Dieses Dokument ist als Referenzhandbuch für alle Debian-Entwickler gedacht. Es steht als Debian-Paket *developers-reference-de* auch in deutscher Sprache zur Verfügung. [9]

7.4. Referenz für *Git-Buildpackage*

Es gibt verschiedene Verfahren und Werkzeuge für das Bauen von Debian-Paketen. In diesem Buch wird *git-buildpackage* verwendet.

Zusätzlich empfehlen wir daher noch die Referenz für das von uns gewählte Build-System *git-buildpackage*[3].

7.5. Handbuch für Debian-Betreuer

Dieses Handbuch für Debian-Betreuer[10] beschreibt den Bau eines Debian-Paketes mittels des *debmake*-Befehls. Es ist für normale Benutzer und angehende Paketbetreuer gedacht.

Der Fokus liegt auf dem modernen Paketierungsstil. Es enthält viele einfache Beispiele.

Dieses „Handbuch für Debian-Betreuer“ soll als Erbe des „Debian-Leitfaden für Neue Paketbetreuer“ betrachtet werden.

Es steht auch in deutscher Sprache und als Debian-Paket *debmake-doc* zur Verfügung. [10]

7.6. Debian Leitfaden für Neue Paketbetreuer

Dieses reife Werk[11] versucht, das Erstellen von Debian-Paketen für normale Anwender und zukünftige Entwickler verständlich mit funktionierenden Beispielen zu beschreiben.

Anders als frühere Dokumente baut dieses auf *debhelper* und weiteren Werkzeugen, die einem Entwickler zur Verfügung stehen, auf.[11]

Auch dieses Dokument steht in deutscher Sprache und als Debian-Paket zur Verfügung.

7.7. Weitere Informationen

Weitere nützliche Literatur findet man unter <https://www.debian.org/doc>.

Zum Thema des Bauens von Debian-Paketen kann man im Internet an weiteren Stellen Dokumente finden. Es gilt jedoch zu beachten: „Das Internet vergisst nichts, und irgendwo ist immer noch eine veraltete Dokumentation verlinkt, deren Hinfälligkeit mangels Verfallsdatums auch nicht zu erkennen ist.“²[12]

²Über dieses Buch, Kapitel 2.9 im Buch Debian-Paketmanagement

8. Was ist ein Debian-Paket?

Um den Weg zu verstehen, sollte man das Ziel kennen. Das Ziel des Paket-Bauens ist ein **Debian-Paket**[5].

Was aber ist ein **Debian-Paket**?

Eine formale Antwort lautet: Ein **Debian-Paket** ist ein Software-Paket, welches vom **Debian-Projekt** veröffentlicht wurde.

Damit ein Paket vom **Debian-Projekt** veröffentlicht wird, muss es Anforderungen genügen, die vom **Debian-Projekt** aufgestellt, schriftlich niedergelegt und zwecks Transparenz veröffentlicht worden sind.

Ein **Debian-Paket** ist ein Software-Paket, welches vor allem den Anforderungen genügt, die in der **Debian-Policy**[7] beschrieben werden. Es gehört zur vom **Debian-Projekt** gepflegten Transparenz, dass die genaue Version der **Debian-Policy**, die beim Bauen des Paketes beachtet wurde, in der Datei *debian/control* angegeben wird. (Kapitel 35.4.6, Seite 265)

Ein **Debian-Paket** ist eine Sammlung von Dateien, die es ermöglichen, Anwendungen oder Bibliotheken über das **Debian-Paketverwaltungssystem** zu verteilen. Das Ziel der Paketierung ist es, die Automatisierung der Installation, der Aktualisierung, der Konfiguration und des Entfernens von Software für **Debian** auf eine konsistente Weise zu ermöglichen.[5][13]

Ein **Debian-Paket** ist mehr als nur eine Archiv-Datei mit ausführbarem Code mit der Endung *.deb*. Ein **Debian-Paket** besteht aus insgesamt **vier** Dateien; davon sind drei Archive.

1. Eine Archiv-Datei mit der Endung *.orig.tar.gz* bzw. *.orig.tar.xz* enthält den Quellcode, aus dem das Paket gebaut wird.
2. Eine weitere Archiv-Datei mit der Endung *.debian.tar.xz* enthält die debian-spezifischen Dateien, die den Bauvorgang und die Installation steuern oder zusätzliche Informationen enthalten.
3. Eine Datei enthält neben beschreibenden Angaben zum Paket die Prüfsummen der beiden vorgenannten Archiv-Dateien. Diese Datei hat die Endung *.dsc*. Diese Datei ist signiert.
4. Den ausführbaren Code enthält schließlich das Archiv mit der Endung *.deb*. Auch diese Datei ist signiert.

In der Regel werden nur die erstgenannten drei Quellcodedateien vom Maintainer hochgeladen. Die Datei, die den ausführbaren Code enthält, wird aus den Quellcodedateien auf der Projekt-Infrastruktur gebaut.

Das **Debian-Paket-System** ermöglicht die Nachvollziehbarkeit vom Upstream-Quellcode bis zum binären **Debian-Paket**. Angestrebt und auch oft erreicht wird eine bitgenaue Reproduzierbarkeit des Bauprozesses[14].

Wie dies nachgeprüft werden kann, wird in Kapitel 29 (Seite 105) beschrieben.

Diese Transparenz gibt dem Anwender Sicherheit, dass das Binär-Paket auch nachvollziehbar aus dem veröffentlichten Quellcode gebaut wurde.

Wer Software aus seinem Build-System in irgendein Archiv im *deb*-Format speichert, ohne sich um Standards und Regeln zu kümmern, packt kein **Debian**-Paket.

Vom **Debian**-Projekt werden zu jedem Zeitpunkt mehrere Varianten (Releases) angeboten, nämlich in erster Linie *stable*, *testing* und *unstable*. Nach der Veröffentlichung jeder neuen *Stable*-Version wird die bisherige *Stable*-Version noch einige Zeit lang als *Oldstable* gepflegt. Desweiteren gibt es noch *Oldoldstable* und einen Zweig *Experimental*. Dort werden Änderungen ausprobiert, die gravierende Auswirkungen auf das Gesamtsystem haben könnten.

Der Zweig *experimental* ist allerdings keine komplette Distribution, sondern funktioniert nur als Erweiterung von *unstable* [15].

Der Zweig *unstable* (*sid*) ist der erste Anlaufpunkt für neue Programme und neue Versionen von Programmen, bevor sie in *testing* integriert werden. Die Entwicklung eines **Debian**-Paketes beginnt somit in der Regel im Zweig *unstable*.¹

Jedes **Debian**-Paket gehört zu einem definierten Entwicklungsstadium der **Debian**-Distribution, also zu einer bestimmten, veröffentlichten Version. Diese werden als Releases bezeichnet.²

Jedes Paket muss auf diese Veröffentlichung abgestimmt sein. Es darf keine Abhängigkeiten zu einer anderen Veröffentlichung haben. Bibliotheken dürfen in einer Veröffentlichung grundsätzlich nur in einer Version vorhanden sein, damit Sicherheitsaktualisierungen für den Anwender einfach möglich sind. Ein **Debian**-Paket darf also keine eigene Version einer solchen Bibliothek enthalten.³

¹DPMB, I.Konzepte, Kapitel 2.10.1[12]

²DPMB,Kapitel 2.10 in [12]

³Kapitel 7.1 [16]

9. Auswahl der zu paketierenden Software

Oft wird die Frage an das Projekt gestellt, welches Paket sich denn zum Starten eigne. Zum Erlernen des Paketierens gibt es kein eigens dafür erstelltes „Schulungs-“Paket. Vielmehr erlernt man das Paketieren durch das Packen von **Debian**-Paketen („Learning by doing“).

Es wird von Anfang an darauf Wert gelegt, dass die Motivation mitgebracht wird, sich Schritt für Schritt in die Thematik des Paketierens einzuarbeiten. Eine gute Voraussetzung dafür ist auch, dass es eine Software ist, die man gerne selber als **Debian**-Paket nutzen und anderen zur Verfügung stellen möchte.

Oft gibt es auch Pakete, für die die Maintainer Hilfe benötigen oder die verwaist sind. Dies kann auch auf selbstgenutzte Pakete zutreffen. Um herauszufinden, ob und welche der installierten Pakete betroffen sind, gibt es ein Werkzeug, das man sich zusätzlich als Paket *how-can-i-help* [17] installieren kann.

Diese Installation führt dazu, dass *apt* am Schluss *how-can-i-help --apt* aufruft. Damit werden dann hilfebedürftige Pakete aufgelistet, die gerade in der lokalen Installation aktualisiert wurden.

Mit *how-can-i-help --old* kann man sich anzeigen lassen, welche der installierten Pakete Hilfe benötigen.

Mit *how-can-i-help --all* werden alle Pakete angezeigt, für die Hilfe benötigt wird¹.

Diese Informationen findet man auch unter <https://www.debian.org/devel/wnpp/>. Das Akronym *wnpp* kürzt *Work-Needing and Prospective Packages* ab und steht sinngemäß für Pakete, an denen Arbeit erforderlich ist und die im Entstehen sind (angehende Pakete).

Motivierend und hilfreich ist es auch, wenn man im Upstream-Projekt aktiv ist. In diesen Fällen ist eine gewisse Nachhaltigkeit der Betreuung des Paketes gewährleistet.

Für viele Kategorien von Paketen haben sich Betreuer-Teams gebildet². Wählt man ein Paket, welches in das Portfolio eines solchen Teams passt, hat man bessere Chancen, Unterstützung und Sponsoren zu finden. Unterstützung findet man auch auf den Mailinglisten der jeweiligen Teams.

Bevor man mit dem Paketieren anfängt, sollte man natürlich prüfen, ob die Software bereits paketiert ist. Mit *apt search <name>* kann festgestellt werden, ob es bereits ein entsprechendes **Debian**-Paket gibt. Auf salsa.debian.org kann nachgeforscht werden, ob schon Paketerversuche im Gange sind.

Für alle Fragen zum Paketieren für **Debian** gibt es auch eine spezielle Mailingliste³. Schon das Mitlesen dieser Liste ist sinnvoll.

¹Weitere Einzelheiten zu *how-can-i-help* werden in *Debian-Paketmanagement*[12] in Kapitel 37.3.6 beschrieben.

²Eine Liste der Teams findet sich im Wiki[18]

³<https://lists.debian.org/debian-mentors/>

10. Prüfung der Quellen

Nach der Auswahl der für Debian zu paketierenden Software muss nun der Quellcode näher untersucht werden. Ziel dieser Untersuchung ist die Feststellung, dass die ausgewählte Software vom Paketbetreuer für Debian `main` paketierte werden kann. Dazu ist zu prüfen, ob **alle** Teile des Quellcodes konform zu den `Debian Free Software Guidelines` (DFSG)[6] und mit der `Debian-Policy`[7] vereinbar sind.

Dies erfordert eine intensive Prüfung **aller** Quellcode-Dateien.

10.1. Lizenzprüfung

In Debian darf im Zweig `main` ausschließlich Freie Software im Sinne der `Debian Free Software Guidelines` [6] veröffentlicht werden. Dabei ist zu beachten, dass die `Debian Free Software Guidelines` (DFSG) nicht nur für Software-Lizenzen gelten, sondern gemäß Ziffer 1 der Version 1.1 des Gesellschaftsvertrages („alle Komponenten“) auch für die Lizenzen von Bildern, Tönen, Texten etc.

Die Prüfung dieser Anforderung hat einen großen Anteil an der Arbeit eines Paket-Betreuers, besonders um ein neues Paket in Debian zu veröffentlichen.

Bei der Erstellung und Pflege der `debian/copyright`-Datei wird dann diese Arbeit verwendet. Damit ist diese Datei auch eine der wichtigsten Dateien eines Debian-Paketes. Sie beschreibt die urheberrechtliche Situation.

Diese Datei muss das Copyright und die Lizenzen aller Dateien eines Quellpakets genau unter Verwendung einer spezifischen Syntax beschreiben (DEP-5).[19]

Die `debian/copyright`-Datei wird von den FTP-Mastern[20] überprüft, wenn ein neues Paket im Debian-Projekt angenommen werden soll.

Um zu vermeiden, dass hierzu Einträge übersehen werden, gilt mindestens ein Vier-Augen-Prinzip (Peer-Review)[21].

Der Inhalt der Copyright-Datei muss also die Lizenzen aller Dateien genau wiedergeben. Die Lizenz wird oft in den Kommentaren einer Quelldatei angegeben.

Der gesamte Quellcode ist unter diesem Aspekt zu prüfen. Es gibt einige Hilfsmittel, die dem Paket-Betreuer helfen, diese Tests durchzuführen. ¹

10.1.1. `debmake`

Der Befehl `debmake -cc` wird auch vom Programmskript zur Erstellung der Datei `debian/copyright` genutzt (Kapitel 35.4.5, Seite 264).

In der Praxis zeigt sich, dass dies manchmal nicht ausreicht. Dies liegt daran, dass die Informationen zu den Lizenzen und den Urhebern nicht einem Standard folgend abgelegt sind. Daher findet das Programm `debmake` nicht alle Einträge bezüglich des benötigten `Copyrights`. Es werden also weitere Tests benötigt.

¹<https://wiki.debian.org/CopyrightReviewTools>

10.1.2. *licensecheck*

Das Programm *licensecheck* (aus dem gleichnamigen Paket) ist in der Lage, Quelldateien zu scannen und meldet das Copyright und die darin angegebenen Lizenzen. Er fasst diese Informationen jedoch nicht zusammen: Für jede Datei eines Pakets wird eine Copyright-Zeile generiert.

Der Test wurde mit der Befehlszeile

```
licensecheck --check '*' --recursive \  
--deb-machine --lines 0 -- *
```

wie im Wiki² ausgeführt.

Leider ist die Handbuchseite (Manpage) von *licensecheck* eher unergiebig. Mehr Informationen liefert der Befehl:

```
licensecheck --help
```

Die Ausgabe muss manuell ausgewertet werden, da *licensecheck* auch versucht, das Copyright für sogenannte Binärdateien (z. B. Bilder, Fonts) anzuzeigen.

10.1.3. *scan-copyrights*

Mit dem Programm *scan-copyrights* aus dem Paket *libconfig-model-dpkg-perl* kann eine vorhandene Copyright-Datei durch erneutes Scannen der Quelle aktualisiert werden.

Die Befehlszeile dazu lautet:

```
scan-copyrights
```

Das Programm kann auch eine solche Datei von Grund auf neu erstellen. Dies erfolgt im DEP-5-Format. [19]

Dieses Programm ist in Perl geschrieben und nutzt *licensecheck*.

10.1.4. *licensing*

Der Befehl *licensing* aus dem Paket *licenseutils* kann den Quellcode scannen und gefundene Lizenzen mit dem Befehl

```
licensing detect *
```

melden. Es kann auch Lizenz-Vorlagen zu neuem Code hinzufügen.

²<https://wiki.debian.org/CopyrightReviewTools#licensecheck>

10.1.5. *cme*

Ein weiteres Tool ist *cme*. *cme* prüft und/oder editiert die Daten in den Konfigurationsdateien. Damit kann man unter anderem prüfen, ob die vom Original-Autor bereitgestellte Copyright-Datei alle notwendigen Informationen enthält.

Der Befehl *cme fix dpkg* prüft die *dpkg*-Dateien, aktualisiert veraltete Parameter und wendet alle Korrekturen an (Kapitel 35.5, Seite 282).

Mit

```
cme update dpkg-copyright
```

werden die in den Headern der Quellcodedateien aufgeführten Lizenzen geprüft und aufgelistet.

Mit

```
cme check dpkg-copyright -file <Pfad/Dateiname>
```

können Daten aus einer beliebigen Datei gelesen werden.

Zu diesem Paket gibt es mit *libcomcnfig-model-tkwi-perl* auch eine graphische Oberfläche.

10.1.6. Manuell

Der Gebrauch dieser Werkzeuge ist manchmal nicht ausreichend. Es kann vorkommen, dass irgendwo im Code weitere Autoren genannt werden. Hiernach kann mit folgenden Befehlen gesucht werden:

1. `grep --recursive --ignore-case "(c)" .`
2. `grep --recursive --ignore-case "copyright" .`
3. `grep --recursive --ignore-case "author" .`

10.1.7. Stolperfallen

Es gibt folgende Stolperfallen:

1. mitgelieferte Build-Abhängigkeiten
2. Microcode
3. mitgelieferte unfreie Dokumente
4. vom Upstream-Autor unvollständig beachtete Lizenzen

Solche Dateien müssen aus dem zu veröffentlichenden **.orig.tar.xz* ausgeschlossen werden, sofern dies sinnvoll möglich ist. Dabei ist auch eine entsprechende Versionsbezeichnung zu wählen. [22]. In diesem Programmskript ist vorgesehen, die Dateien mithilfe der Datei *debian/copyright* auszuschließen (Kapitel 34.4.5, Seite 217)

Wenn im Quellcode-Paket unterschiedlich lizenzierte Dateien enthalten sind, ist neben der Prüfung der Lizenzen der einzelnen Quellcode-Dateien auch das Quellcode-Paket insgesamt zu untersuchen. Es ist zu prüfen, ob die verwendeten Lizenzen miteinander kompatibel sind. Hierbei ist *FINOS Open Source License Compliance Handbook*[23] hilfreich.

10.2. Feststellung der Programmiersprache

Software wird in unterschiedlichen Programmiersprachen geschrieben. Es gibt auch Programme, die in mehreren Programmiersprachen geschrieben sind. Je nach Programmiersprache sind verschiedene Compiler- und Build-Systeme zu verwenden.

Für das Bauen der Java-Pakete gibt es mehrere Build-Systeme. Dies wird in einem eigenen Kapitel ausführlich beschrieben (Kapitel 13, Seite 41).

10.3. Prüfung der Abhängigkeiten

Es wird zwischen zwei Arten von Abhängigkeiten unterschieden:

- Software, die für das Bauen des Paketes notwendig ist (build dependencies),
- Software, die für die Installation und/oder den Ablauf des paketierte Programms erforderlich ist (runtime dependencies)

Um ein Paket in `Debian main` veröffentlichen zu können, müssen **alle** Abhängigkeiten (inklusive Build-Abhängigkeiten) bereits in `Debian main` verfügbar sein[7]. Dies bedeutet, dass Abhängigkeiten, die noch nicht verfügbar sind, zunächst zu paketieren sind.

Wie die Abhängigkeiten ermittelt werden können, hängt von der Programmiersprache ab. Hinweise auf unerfüllte Build-Abhängigkeiten findet man auch in den Fehlermeldungen beim Bauen eines Paketes. Angaben dazu, warum das Bauen fehlgeschlagen ist findet man in der Datei `<Paketname>_<Version>.build`.

Für Java-Programme gibt es verschiedene Orte, an denen diese Informationen gefunden werden können. Dies wird im entsprechenden Kapitel beschrieben (Kapitel 13, Seite 41)

Die dann gefundenen Build- und Runtime-Abhängigkeiten müssen in der Datei `debian/control` im passenden Abschnitt aufgeführt werden (Kapitel 35.4.6, Seite 265).

10.3.1. Abhängigkeiten ermitteln mit `packages.debian.org`

Zur Ermittlung, ob die benötigte Abhängigkeit bereits paketierte ist, kann die Webseite `packages.debian.org` eine Anlaufstelle sein.

Im Abschnitt *Durchsuchen der Paket-Verzeichnisse* sind die Optionen *Nur Paketnamen* und *Quellpaket-Namen* interessant.

Im Abschnitt *Durchsuchen des Inhalts von Paketen* ist es dann die Option *Pakete mit Dateien, deren Namen das Suchwort enthalten*.

Als Distribution sollte dann *alle* bzw. *Unstable* ausgewählt werden.

10.3.2. Abhängigkeiten ermitteln auf `codesearch.debian.net`

Manchmal kann es hilfreich sein, nach Quellcode zu suchen, in dem ein ähnliches Problem schon mal gelöst wurde.

Ein Beispiel ist, dass in einem Paket, das mit `Maven` gebaut wurde, schon einmal ein bestimmter Eintrag in der Datei `debian/maven.rules` erfolgte.

Eine solche Suche kann – wie folgt – durchgeführt werden:

```
https://codesearch.debian.net/search?q=<SearchString>
```

10.3.3. Suche nach Abhängigkeiten auf der Konsole

Mit `apt-file search`³ (Alias: `apt-file find`) kann festgestellt werden, ob benötigte Abhängigkeiten bereits in Debian paketiert sind. Mit `apt-file list <Paketname>` (Alias: `apt-file show`) kann der Inhalt eines Paketes aufgelistet werden. Hier kann dann nach Programmnamen gesucht werden. Die gleichen Informationen erhält man mit `dpkg -L <Paketname>`.

10.4. Änderungen am Quellcode

Der Quellcode ist sorgfältig darauf zu prüfen, ob Änderungen an ihm für das Debian-Paket vorgenommen werden müssen. Eine solche Notwendigkeit kann verschiedene Ursachen haben.

Das Upstream-Release kann Teile enthalten, die nicht der Debian-Policy[7] oder den Debian-Free-Software-Guidelines[6] entsprechen.

Änderungen am Quellcode speziell für das Debian-Paket können wie folgt vorgenommen werden:

- Es können ganze Dateien ausgeschlossen werden. Dies geschieht beim Bauen einer neuen Version (Kapitel 34.4.5, Seite 217).
- Es können Änderungen an Upstream-Dateien durch *Patches* (Kapitel 10.4.2, Seite 33) vorgenommen werden. Dies erfolgt durch das Programmskript beim Bauen einer neuen Revision (Kapitel 36, Seite 285).

10.4.1. Ganze Dateien ausschließen

Wenn ganze Dateien ausgeschlossen werden müssen, muss ein Quellcode-Paket erstellt werden, das diese Dateien nicht mehr enthält.

Bevor `mk-origtar.gz` aufgerufen wird, ermöglicht das Programmskript einzelne Quellcode-Dateien von der Aufnahme in das *orig*-Archiv auszuschließen.

Die Dateien, die nicht der Debian-Policy[7] oder den Debian-Free-Software-Guidelines (DFSG)[6] entsprechen, sind zu entfernen.

Dabei ist immer eine neue Version zu bauen, auch dann, wenn kein neues Upstream-Quellcode-Archiv verwendet wird.

Die Ausschlüsse sollen mit ihrer Begründung in der Datei `debian/README.source` (Kapitel 35.4.19, Seite 282) dokumentiert werden.

10.4.1.1. Auflistung der auszuschließenden Dateien

Die auszuschließenden Dateien können entweder in einer separaten Datei oder in der Datei `debian/copyright` im *DEP-5-Format* [19] aufgelistet werden.

Beispiel:

```
Format: https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/
Upstream-Name: <UpstreamName>
Source: <URL to upstream source>
Files-Excluded: <all files to be excluded>
Comment: <description, why the files are excluded>
```

³<https://manpages.debian.org/unstable/apt-file/apt-file.1.en.html>

In der Aufzählung werden die Dateien, die auszuschließen sind, durch Leerzeichen oder Zeilenumbruch mit Einrückung um ein Zeichen separiert.

10.4.1.2. Fallunterscheidungen

Oft steht schon vor der Einreichung des Paketes zur *New Queue* fest, dass Dateien auszuschließen sind. Da es zu diesem Zeitpunkt noch keine Datei *debian/copyright* gibt, bietet es sich an, die auszuschließenden Dateien in einer separaten Datei aufzulisten.

Auf die entsprechende Frage im Programmskript ist anzugeben, dass Dateien auszuschließen sind. Dann ermittelt das Programmskript, wo die Informationen zum Ausschluss von Dateien hinterlegt werden sollen. Dies ist vor der Einreichung des ersten Paketes eines neuen Projektes eine separate Datei.

Dazu wird eine Datei *<SourceName>-excluded* im Verzeichnis *PrjPath* (Kapitel 32.2.2, Seite 139) erstellt.

Die Informationen in dieser Datei sind bei der Erstellung der Datei *debian/copyright* dorthin zu übertragen (Kapitel 35.4.5, Seite 264).

Nach einer Zurückweisung aus der *New Queue* muss das Paket neu für eine Veröffentlichung bereit gestellt werden. Auch hierbei kann es vorkommen, dass Dateien, deren Lizenz nicht DFSG-konform ist, ausgeschlossen werden müssen. Der Ausschluss der Dateien gilt bereits für das zu veröffentlichende *.orig.tar.(gz | bz2 | xz). Meist steht zu diesem Zeitpunkt aber keine weitere (neuere) Upstream-Version zur Verfügung. Die zu erstellende Version muss jedoch größer sein als die bisherige Version aber kleiner als die nächste zu erwartende Upstream-Version.

Im Falle des Entfernens von Dateien, die nicht DFSG-konform sind, ist dies der Anhang *+dfsg*, der sich direkt an die Upstream-Version anschließt. Ist dieser Anhang bereits vorhanden, so wird er nach *+dfsg1* hochgezählt (s. Kapitel 10.4.1.3, Seite 32). So kann mit *mk-origtargz* ein neuer Tarball mit einer größeren Versionsbezeichnung erstellt werden. (Kapitel 11.2, Seite 35)

Die Namen der auszuschließenden Dateien werden der dann schon vorhandenen Datei *debian/copyright* im DEP-5-Format[19] hinzugefügt. Nach dem Commit dieser Anpassungen im Verzeichnis *debian/* kann eine neue Version gebaut werden. Dazu wird das bisherige Upstream-Archiv verwandt.

Auf diese Weise wird ein neuer Orig-Tarball mit *mk-origtargz* ohne die auszuschließenden Dateien aus dem bisherigen *.tar.gz erstellt und dessen Inhalt mit *gbp import-orig* in das vorhandene Git-Repository eingefügt (Kapitel 34.4.11, Seite 243).

Wird es bei einer neuen Upstream-Version notwendig, dass Dateien auszuschließen sind, oder ändern sich die auszuschließenden Dateien, ist genauso zu verfahren, wie soeben dargelegt. Es muss die Copyright-Datei geändert und ein entsprechender Commit ausgeführt werden. Wiederum ist beim Bauen der neuen Version anzugeben, dass Dateien auszuschließen sind.

10.4.1.3. Benamung der Pakete beim Ausschluss von Dateien

Damit die Versionsbezeichnung dieser neuen Version größer als die der bisherigen, aber kleiner als die zu erwartenden nächsten Upstream-Version ist, wird die bisherige Versionsbezeichnung mit einem Anhang versehen. Dieser macht zugleich deutlich, dass und warum der Upstream-Code verändert wurde.

Übliche Ergänzungen der Versionsbezeichnung sind *+dfsg* und *+ds*⁴.

Wenn Lizenzen einzelner Dateien des Upstream-Quellcodes nicht den **Debian Free Software Guidelines** (DFSG) genügen und daher nicht als Quellen durch Debian veröffentlicht werden sollen, wird als Ergänzung *+dfsg* benutzt. Erfordern andere Gründe den Ausschluss, nimmt man *+ds* („Debian Source“).[22]

Es kann nämlich passieren, dass zunächst nur Dateien ausgeschlossen wurden, die nicht der **Debian**-Policy entsprechen.

Wird später festgestellt, dass das hochgeladene Original-Archiv noch unfreie Dokumente enthält, führt dies dann zu einer Versionsbezeichnung nach dem Muster *<Versionsnummer>+ds+dfsg*⁵

In manchen Fällen kann ein Pluszeichen (+) Probleme vor allem beim Bauen von Java-Paketen verursachen.

Die dargestellte Benamung der **Debian**-Pakete hat auch Konsequenzen für den Inhalt der Datei *debian/watch* (Kapitel 11.4 Seite 36 und Kapitel 40.4 Seite 362).

10.4.2. Änderungen in einzelnen Quellcode-Dateien (Patchen)

Änderungen an Quellcode-Dateien sind dann notwendig, wenn aufgrund unterschiedlicher Build-Umgebungen Anpassungen vorzunehmen sind. Andere Fälle sind Fehlerbehebungen, vor allem *Security-Patches*.

Die klassische Methode ist die im Folgenden beschriebene mit *quilt*.

Daneben kann dies auch in einem *Patch-Queue*-Zweig mit *gbp pq* erfolgen (Kapitel 10.4.2.2, Seite 33)

10.4.2.1. Patchen mit *quilt*

Änderungen an Quellcode-Dateien erfolgen durch sogenannten Patch-Dateien. Diese dokumentieren den ursprünglichen Quellcode und die jeweiligen Änderungen. Eine weitere Datei (*debian/patches/series*) steuert die Reihenfolge der Anwendung der Patches. Herkömmlicherweise geschieht dies mit *quilt*. *Dquilt* ist eine **debian**-spezifische Anpassung für *quilt*.

Hierzu gibt es eine Beschreibung im *Debian-Leitfaden für Neue Paketbetreuer*⁶ und im *Debian-Wiki*[24]. Daneben gibt es eine allgemeine Einführung in die Nutzung von *Quilt* [25].

In Kapitel 19.6 (Seite 72) wird beschrieben, wie die **debian**-spezifische Anpassung erzeugt wird.

10.4.2.2. Patchen in einem *Patch-Queue*-Zweig

Da ein Arbeitsablauf mit **Git** bzw. *git-buildpackage* [3] zum Bauen der **Debian**-Pakete genutzt wird, bietet es sich an, *gbp pq* statt *quilt*⁷ einzusetzen.

Die Änderungen erfolgen dann in einem eigenen **Git**-Zweig

⁴<https://wiki.debian.org/DebianMentorsFaq> in section *What does „dfsg“ or „ds“ in the version string mean*

⁵s. Mentors-FAQ 2.6[22]

⁶<https://www.debian.org/doc/manuals/maint-guide/modify.de.html>

⁷section *gbp.patches.html* [3]

Bei der Nutzung von *gbp pq* erfolgen in diesem Zweig dann *alle* Codeänderungen. Diese Änderungen sollten thematisch und kleingliedrig sein. Dann können die Patches einfacher nach Upstream fließen oder von anderen Distributionen übernommen werden. Dies erleichtert auch die Anpassung an eine neue Upstream-Version.

Die Grundidee ist, dass Patches vom **Debian**-Zweig in den Patch-Queue-Zweig so importiert werden, dass jeweils eine Patch-Datei in *debian/patches* jeweils einem Commit auf dem Patch-Queue-Zweig entspricht.

Die Anlage des Patch-Queue-Zweiges erfolgt mit *gbp pq import*. (s. a. Kapitel 34.2 Seite 197)

Der erstellte Zweig wird nach dem Zweig benannt, aus dem er importiert wurde. Zur Unterscheidung wird *patch-queue/* vorangestellt. Wird also die **Debian**-Paketierung auf *debian/sid* gemacht und ein *gbp pq import* ausgeführt, wird der neu erstellte Zweig *patch-queue/debian/sid* genannt.

Das Programmskript ermöglicht diesen Import vor dem Herunterladen einer neuen Upstream-Version (Kapitel 34.2, Seite 197) und als Auswahlmöglichkeit zum Patchen (Kapitel 36.1, Seite 288).

Im Patch-Queue-Zweig kann mit den bekannten **Git**-Befehlen (*rebase*, *commit* und *--amend*, etc.) an den Commits gearbeitet werden. Wenn dies erledigt ist, wird *gbp pq export* verwendet, um die Commits auf dem *Patch-Queue-Zweig* wieder in Patches in *debian/patches/*-Dateien umzuwandeln (Kapitel 36.1, Seite 288).

Dieser beschriebene Workflow erleichtert z. B. das Cherry-Picking von Patches für stabile Releases, die Weiterleitung von Patches an neue Upstream-Versionen durch die Verwendung von *git rebase* auf den Patch-Queue-Zweig (bereits angewandte Patches werden automatisch erkannt) sowie das Neuordnen, Ablegen und Umbenennen von Patches, ohne auf *quilt* zurückgreifen zu müssen. Die erzeugten Patches in *debian/patches/* haben alle notwendigen Informationen, um sie nach Upstream weiterzuleiten, da sie ein Format ähnlich dem *git-format-patch* verwenden.

Der Hauptnachteil dieses Arbeitsablaufes ist der Mangel an Historie im Patch-Queue-Zweig, da er häufig fallen gelassen und neu erstellt wird. Aber es gibt natürlich eine vollständige Historie auf dem **Debian**-Zweig im Verzeichnis *debian/patches/*.

Zu beachten ist, dass *gbp pq* derzeit keine vollständige Unterstützung für DEP3-Header[26] bietet.

Zunächst wird versucht, mit *git-mailinfo*[27] zu parsen, was nur die Felder *From* und *Subject* unterstützt. Wenn keines dieser beiden vorhanden ist, wird *gbp pq* versuchen, den Patch vom DEP3-Format in ein *git-mailinfo(1)*-kompatibles Format zu konvertieren. Dabei wird zuerst aus den Feldern "Autor" und "Betreff" über die erste Zeile des Feldes Beschreibung geladen. Anschließend werden alle zusätzlichen Felder (wie "Origin" und "Forwarded") und der Rest der Beschreibung (falls vorhanden) an den *mail body* angehängt.

11. Versionierung der Pakete

Sowohl die Software, die für Debian gebaut werden soll, als auch Debian-Pakete haben Versionsbezeichnungen. Dabei sollte die Versionsbezeichnung der Upstream-Software in der des Debian-Paketes enthalten sein.

Die Namen aller Debian-Binärpaketdateien sind folgendermaßen aufgebaut:

`<foo>_<Versionsnummer >-<Debian-Revisionsnummer>_<Debian-Architektur>.deb.` ¹

11.1. Paketname

Vor der Versionsbezeichnung kommt der Paketname.

Dieser muss gänzlich in Kleinbuchstaben geschrieben werden. Die Versionsbezeichnung kann auch Ziffern enthalten. Zusätzlich können `+`, `-`, `~` enthalten sein.

Die Versionsbezeichnung muss mit einer Ziffer beginnen.

11.2. Versionierungsschema

Die Versionierung der Pakete folgt einem fest definierten Schema. Damit wird sichergestellt, dass alle Werkzeuge, die diese Versionierung verwenden, auf die gleiche Systematik zugreifen können. Dabei muss sichergestellt werden, dass das System die neue Versionsbezeichnung auch als größer als die vorherige Versionsbezeichnung erkennt. (s.a Kapitel 34.4.4, Seite 214)

Wie muss also die Versionsbezeichnung des zukünftigen Paketes zusammengesetzt sein? Nach der Versionsbezeichnung des Upstream-Paketes können debian-spezifische Zusätze erfolgen. Diese können anzeigen, ob und warum Teile der Upstream-Software entfernt wurden oder ob das Paket auf ein älteres Paket zurückportiert wurde. Außerdem ist in der Versionsbezeichnung des Paketes eine Debian-Revisionsnummer enthalten. Diese Revisionsnummer wird insbesondere dann erhöht, wenn das Paket wegen Korrekturen in Dateien im Verzeichnis *debian/* neu gebaut wurde.

Mögliche Varianten können mit dem folgenden Befehl auf ihre Tauglichkeit überprüft werden:

```
dpkg --compare-versions Version1 Operant Version2 && \  
echo "OK"
```

Als Operanden sind diese zu nutzen: `lt` `le` `eq` `ne` `ge` `gt`

lt kleiner als (`<`)

le kleiner oder gleich (`<=`)

eq gleich (`=`)

¹DebianFAQ2019, Kapitel 7.3[16]

ne ungleich (\neq)

ge größer oder gleich (\geq)

gt größer als ($>$)

Die Versionsvergleichsregeln können wie folgt zusammengefasst werden:²

- Zeichenketten werden von Anfang bis Ende verglichen.
- Buchstaben sind größer als Ziffern.
- Ziffern werden als Ganzzahlen verglichen.
- Buchstaben werden in ASCII-Sortierreihenfolge verglichen.
- Es gibt besondere Regeln für Punkt ($.$), Plus- ($+$) und Tilde- (\sim) Zeichen, wie folgt:
Beispiel: $0.0 < 0.5 < 0.10 < 0.99 < 1 < 1.0\sim rc1 < 1.0 < 1.0+b1 < 1.0+nmul < 1.1 < 2.0$

In manchen Fällen kann ein Pluszeichen ($+$) Probleme vor allem beim Bauen von Java-Paketen verursachen.

Die "richtige" Versionierung spielt eine Rolle, wenn diese von Programmen gelesen und genutzt werden soll. Dies sind vor allem die Programme *dpkg*, *apt* und *uscan*.

11.3. *apt* und *dpkg*

Die korrekte Versionierung des Debian-Paketes ist dann nicht ganz einfach, wenn nicht für Debian-Sid gebaut wird (Kapitel 22.7, Seite 87).

dpkg bzw. *apt* müssen zutreffend ermitteln können, ob ein neueres Paket im jeweiligen Release-Zweig zur Verfügung steht.

11.4. *uscan* und die Datei *debian/watch*

uscan ist ein nützliches Werkzeug im Debian-Projekt. Mittels *uscan* kann geprüft werden, ob es eine Upstream-Version gibt, die aktueller (also höher versioniert) ist, als das aktuelle Debian-Paket.

Das Programm *uscan* wird mit dem Paket *devscripts* zur Verfügung gestellt.

Anhand der in *debian/watch* enthaltenen Daten prüft *uscan*, ob neuere Versionen in der Originalquelle vorhanden sind (Kapitel 40.4, Seite 362), und kann sie gegebenenfalls herunterladen (Kapitel 34.5, Seite 248).

Die durch *uscan* gewonnene Informationen werden auch auf der Seite des jeweiligen Maintainers

<https://qa.debian.org/developer.php?<Maintainer>@debian.org>
dargestellt.

Dafür benötigt *uscan* vor allem eine taugliche Datei *debian/watch*. Die Erstellung dieser Datei, die auch beim Herunterladen des Quellcodes mit *uscan* (Kapitel 34.5, Seite 248) benötigt wird, wird vom Programmskript unterstützt (Kapitel 35.4.7, Seite 269).

Wenn Dateien ausgeschlossen werden und die **.orig.tar.xz*-Datei entsprechend benannt wurde (Kapitel 10.4.1.3, Seite 32), sind entsprechende Optionen in die Datei *debian/watch* einzufügen. Diese Optionen sind *repack*, *compression*, *repacksuffix* und *dversionmangle*. Dann kann bei der nächsten neuen Upstream-Version das Herunterladen mit *uscan*

²New Maintainer Guide, Kapitel 2.6 [11, Kapitel 2.6] s.a. Debmake-doc, Kapitel 5.2

durchgeführt werden. Dies gilt auch für die Prüfung mittels *uscan*, ob es eine neue Upstream-Version gibt.

Beispiel:

```
opts=repack,\
compression=xz,\
repacksuffix=dfsg,\
dversionmangle=s/\+dfsg// \
```


12. *dh_make*

Es gibt erfreulicherweise viele nützliche Programme (Hilfsprogramme), die das Bauen von **Debian**-Paketen erleichtern und vereinheitlichen. Wer sich mit dem Bauen von **Debian**-Paketen befasst, wird immer wieder auf das Werkzeug *dh_make* stoßen. Das Paket heißt *dh-make* (mit Bindestrich).

dh_make zeigt auf, wie ein **Debian**-Paket potenziell aussehen kann. Dies ermöglicht das Erlernen des Paketierens an konkreten Beispielen. Lehrreich sind vor allem die im Programmpaket enthaltenen Vorlagen (Templates).

dh_make erstellt viele Dateien in dem Verzeichnis *debian/*. Für die meisten Pakete wird nur eine Teilmenge davon benötigt.

Die Praxis zeigt, dass jedes Mal geprüft werden muss, welche Dateien für das konkrete Paket nicht benötigt werden. Auch sind die erzeugten Dateien unvollständig und erfordern einiges an Handarbeit.

Um *dh_make* nutzen zu können, sind einige Vorbereitungen zu treffen. Dazu wird ein entsprechendes „Arbeitsverzeichnis“ angelegt. Darin wird das heruntergeladene Quellcode-Archiv abgelegt. Bevorzugt wird ein hierfür vorgesehenes Tar-Archiv.

Dieses Tar-Archiv wird dort entpackt. Dies kann auf der Konsole durchgeführt werden mit

```
tar --extract --auto-compress --file=<Tar-Archiv>.tar.gz
```

dh_make muss in dem Verzeichnis aufgerufen werden, das den Quellcode enthält. Dabei kann es vorkommen, dass der Name des Verzeichnisses mit dem Quellcode außer Kleinbuchstaben noch weitere Zeichen enthält. *dh_make* benötigt jedoch einen Namen, der Einschränkungen unterliegt und dem Schema <paketname>-<version> entspricht. *dh_make* akzeptiert bei Paketnamen nur Kleinbuchstaben.

Damit *dh_make* die Vorlagen korrekt füllen kann, wird in solchen Fällen an *dh_make* die zusätzliche Option *--packagename* hinzugefügt. Hiermit kann die Verwendung des nachfolgenden Namens als Name des Quellcodepaketes erzwungen werden. Dieser Verzeichnisname muss die Versionsbezeichnung enthalten. Zwischen Paketnamen und Versionsbezeichnung muss ein Bindestrich stehen.

So kann nach dem Wechsel in dieses Verzeichnis dort mit

```
dh_make --createorig --packagename <SourceName>
```

der Source-Tarball in der von **Debian** gewünschten Form *<SourceName>.orig.tar.gz/xz* erzeugt werden. Daneben wird ein Entwurf der Dateien im Verzeichnis *debian/* erstellt.

Dabei werden folgende Informationen abgefragt.

Type of package = Paketklassen single, indep, library, python mit folgender Bedeutung
single Es wird ein einzelnes Binary-Paket (*.deb) erstellt. Dies ist der Normalfall
indep Es wird ein Binary-Paket erstellt, das von der Architektur unabhängig ist.

library Es werden mindestens zwei Binärdateien erstellt. Ein Bibliothekspaket, das nur die lib in /usr/lib enthält, und ein weiteres *-dev_*.deb-Paket, das die Dokumentation und C-Header enthält.

python

Einzelheiten zum Paket Es werden die ermittelten Werte für *Maintainer Name*, *Email-Address*, *Date*, *Package Name*, *Version*, *License* und *Package Type* zur Bestätigung angezeigt.

Es werden folgende Dateien erzeugt. Dabei werden zunächst die Dateien gelistet, die auf jeden Fall benötigt werden, jeweils in alphabetischer Reihenfolge. Die Dateien *README.Debian* und *README.source* können zu Informationszwecke genutzt werden. Zum Schluss folgen solche Vorlagen, die nur in sehr wenigen Paketen genutzt werden.

- changelog
- control
- copyright
- rules
- salsa-ci.yml.ex
- source/format
- watch.ex
- README.Debian
- README.source
- manpage.1.ex
- manpage.sgml.ex
- manpage.xml.ex
- postinst.ex
- postrm.ex
- preinst.ex
- prerm.ex
- <packagename>.cron.d.ex
- <packagename>.doc-base.EX
- <packagename>-docs.docs

Die nicht benötigten Dateien werden aus dem Verzeichnis *debian/* entfernt.

Dass der Verzeichnisname die jeweilige Versionsbezeichnung enthalten muss, ist bei der Verwendung von *git* ungünstig. Es wird bisher davon abgesehen, dieses Hilfsprogramm im Programmskript einzusetzen.

13. Java-Pakete bauen

Das Bauen von Java-Paketen weist besondere Herausforderungen auf. Diese Besonderheiten werden im Folgenden ausführlich beschrieben.

Für das Packen von Java-Paketen gibt es eine speziellen Richtlinie, die *Debian-Java-Policy*[28].

Ferner gibt es noch eine Beschreibung über das Paketieren mit den *Javatools*[29]. Darin werden die *Java-Helper* beschrieben. Dieses Tutorial ist auch Bestandteil des *Debian-Paketes javahelper*.

Weitere Informationen erhält man auch aus der *Java-FAQ*[30], vor allem im Abschnitt 2.4¹

13.1. Herausforderungen

In einem Blog-Artikel[31] beschreibt Hans-Christoph Steiner zutreffend die Herausforderungen für ein *Debian-Paket* in Bezug auf *Java-Software*.

Das *Debian-Java-Team* muss demnach konsequent gegen die *Java-Standardpraxis* ankämpfen, alle Abhängigkeiten in einer einzigen **.jar*-Datei zu bündeln. Dies bedeutet, dass es keine gemeinsamen Sicherheitsaktualisierungen gibt. Jeder Entwickler muss jede Abhängigkeit in diesem Modell selbst aktualisieren. Das funktioniert hervorragend für große Unternehmen mit Mitarbeitern, die sich dieser Aufgabe widmen.

Für die Mehrheit der *Debian-Anwendungsfälle* funktioniert das schlecht. *Debian* hält das Versprechen ein, dass die Leute *foo* einfach passend installieren können, damit es funktioniert, und Sicherheitsaktualisierungen erhalten. Der Benutzer muss nicht einmal wissen, in welcher Sprache das Programm geschrieben ist, es funktioniert einfach.

Die Hoffnung, dass die *Java-Entwickler-Gemeinschaft* sich den Wert dieser Anwendungsfälle zu eigen macht und *Debian* hilft, indem sie es einfacher macht, *Java-Projekte* in der Standard-Distributionsmethode zu paketieren, mit gemeinsamen Abhängigkeiten, die unabhängig aktualisiert werden, hat sich bislang nur rudimentär erfüllt.

13.2. Anwendungen und Bibliotheken

Es gibt zwei Kategorien von *Java-Paketen*: Anwendungen und Bibliotheken. Sowohl *Java-Anwendungen* als auch *Java-Bibliotheken* können im Quellcode (weitere) Bibliotheken enthalten, die vorkompiliert sind.

Sollen in *Java* geschriebene Anwendungen paketiert werden, müssen in der Regel auch *Java-Bibliotheken* als *Debian-Pakete* bereitgestellt werden.

Sowohl Anwendungen als auch Bibliotheken werden in **.jar*-Dateien als *Zip-Archiv* bereitgestellt.

¹<https://www.debian.org/doc/manuals/debian-java-faq/ch2.en.html#s2.4>

Beide Arten müssen grundsätzlich als **Java-Bytecode** (*.class-Dateien, verpackt in einem *.jar-Archiv) und mit einer „*Architecture: all*“ (Kapitel 35.4.8, Seite 273) ausgeliefert werden.

Hat man von einer zu paketierenden Anwendung oder Bibliothek zunächst nur ein (kompiliertes) *.jar-Archiv, kann ein Blick in die Datei *MANIFEST.MF* des jeweiligen .jar-Archivs helfen, diejenige *URL* zu ermitteln, unter der der Quellcode veröffentlicht worden ist.

13.2.1. Java-Programme paketieren

Java-Programme sind dazu bestimmt, von Endbenutzern ausgeführt zu werden. Diese benötigen auch eine Manpage, soweit es sich um selbstständig ausführbare Programme handelt.²

Das Paket muss auch sicherstellen, dass die richtige Klasse als Hauptklasse verwendet wird.

Zusätzliche Klassen im Paket müssen in ein oder mehrere **Java-Archiv(e)** (*.jar-Datei(en)) gepackt werden, die in */usr/share/java* (wenn sie für die Verwendung durch andere Programme vorgesehen sind) oder in */usr/share/<package>* als ein „privates“ Verzeichnis abgelegt werden können.

13.2.2. Java-Bibliotheken paketieren

Java-Bibliotheken dienen der Erfüllung von Abhängigkeiten von Programmen. Dies können Build- und/oder Laufzeitabhängigkeiten sein.

13.2.3. Name des Java-Paketes

Eine besondere Herausforderung ist die Bildung des korrekten Namen des **Java-Paketes**.

13.3. Abhängigkeiten bei Java-Paketen

Java-Anwendungen hängen immer auch von **Java-Bibliotheken** ab. Außerdem wird für die **Java-Pakete** immer *default-jdk* als Abhängigkeit benötigt. Dahinter verbirgt sich in der Regel die aktuell verfügbare **OpenJDK-Version**, eine freie **Java Implementierung** mit Langzeitunterstützung (**JDK = Java Development Kit**).

Daneben gibt es spezielle Abhängigkeiten, die sich aus dem **Build-System** ergeben. Diese werden bei den einzelnen **Build-Systemen** im Folgenden besprochen.

13.3.1. Weitere Abhängigkeiten feststellen

Java-Entwickler fügen ihren Quellcode-Paketen die benötigten **Build-Abhängigkeiten** - auch Bibliotheken von Drittanbietern - in kompilierter Form als *.jar-Archive hinzu.

Gerade unter **Java-Entwicklern** ist es leider sehr verbreitet, auf irgendwo im **World Wide Web** veröffentlichte Bibliotheken zurückzugreifen und diese dann vorkompiliert mit

²Kapitel 2.3 der *Java-Policy*[28]

auszuliefern. Diese Bibliotheken müssen durch in **Debian** veröffentlichte Pakete ersetzt werden.

Zur Feststellung von Abhängigkeiten ist also im Quellcode nach **.jar*-Archiven zu suchen.

In diesem Falle ist für den **Debian**-Maintainer ein Repacking zum Entfernen dieser Dateien aus dem Upstream-Tarball notwendig. (Kapitel 36, Seite 285)

Die ausgeschlossenen Bibliotheken müssen durch eigenständige Pakete verfügbar gemacht werden.

13.3.2. Abhängigkeiten ermitteln

Hat man Abhängigkeiten festgestellt, so ist zu ermitteln, wie diese Abhängigkeiten erfüllt werden können. Hierzu ist als erstes zu prüfen, ob bereits entsprechende **Debian**-Pakete existieren. Um zu ermitteln, ob eine Abhängigkeit bereits in **Debian** paketierte wurde, gibt es mehrere Wege.

Debian-Wiki (für Maven-Pakete):

<https://wiki.debian.org/Java/MavenPkgs>

13.4. Build-Systeme für Java-Pakete

Zum Bauen von **Java**-Paketen werden verschiedene Build-Systeme genutzt. Hierbei handelt es sich um *maven*, *ant* und *gradle*. Daraus ergeben sich Eigenheiten, die beim Paketieren für **Debian** beachtet werden müssen. Es gibt auch **Java**-Pakete, die ohne eines dieser Build-Systeme gebaut werden. Die folgenden Beschreibungen orientieren sich an den gemachten Erfahrungen der Autoren.

13.4.1. Das Build-System *maven*

Apache **Maven** ist ein Werkzeug zur Verwaltung und zum Verstehen von Softwareprojekten.

Ein wesentliches Erkennungsmerkmal ist die Datei *pom.xml* für das **Maven**-Build-System. Diese enthält alle Informationen zum jeweiligen Softwareprojekt und folgt einem standardisierten *XML*-Format.

Wesentlich ist auch die Standard-Verzeichnisstruktur eines **Maven**-Projektes, die im Folgenden dargestellt wird.[32]

src/main/java Anwendungs- / Bibliotheksquellen
src/main/resources Anwendungs- / Bibliotheksressourcen
src/main/filters Ressourcenfilterdateien
src/main/webapp Webanwendungsquellen
src/test/java Testquellen
src/test/resources Ressourcen testen
src/test/filters Testen Sie Ressourcenfilterdateien
src/it Integrationstests (hauptsächlich für Plugins)
src/assembly Assembly-Deskriptoren
src/site Website
LICENSE.txt Lizenz des Projektes

NOTICE.txt Hinweise und Zuordnungen für Bibliotheken, von denen das Projekt abhängt

README.txt Readme des Projektes

pom.xml Beschreibung des Projektes und der Konfiguration

In den meisten Fällen reicht es aus, den Zweig *src/main/* als **Debian**-Paket zu bauen. Dies gilt besonders dann, wenn Bibliotheken gebaut werden, die als Abhängigkeiten für Anwendungen benötigt werden.

Das Upstream-Paket muss bestimmte Voraussetzungen erfüllen, um mit dem Build-System **Maven** gebaut werden zu können.

Dazu gehört auf jeden Fall mindestens eine *pom.xml*-Datei. Diese Datei kann an verschiedenen Stellen innerhalb des Quellcodes liegen. Manchmal wird sie - besonders im **Maven**-Repositorium (<https://mvnrepository.com/>) - nur zusätzlich bereitgestellt.

Wenn vorhanden, ist es jedoch oft besser, **GitHub** oder andere **Git**-Repositorien als Quelle zu wählen. Dabei ist auf die Version zu achten! Für manche Pakete wird der Quellcode auch unter der Domain <https://gitbox.apache.org/repos/asf> veröffentlicht.

13.4.2. Paketieren mit *maven*

Für *maven*-basierte Pakete wird die Verwendung von *maven-debian-helper*³ empfohlen.

In der Datei *debian/rules* (Kapitel 47.5, Seite 432) wird dazu `--buildsystem=maven` hinzugefügt.

Zur Unterstützung des Bauens mit *maven* enthält dieses Paket folgende Befehle:

mh_genrules(1) generiert, zumindest teilweise, die Datei *debian/rules*

mh_ lspoms(1) sucht nach allen **POM**-Dateien, die im Quellcode des Projektes angegeben sind.

mh_make(1) generiert durch Lesen der Informationen aus der **Maven-POM** das **Debian**-Paket.

mh_resolve_dependencies(1) löst die Abhängigkeiten auf und erzeugt die Datei *<Paketname>.substvars*, die die Liste der abhängigen Pakete enthält, zur Nutzung von *debian/control*.

Zu diesen Befehlen gibt es auch entsprechende *Man-Pages*.

Hiervon wird zunächst nur der Befehl *mh_make*⁴ im Plugin *build-gbp-maven-plugin.sh* genutzt. (Kapitel 47, Seite 419)

Bei der Nutzung von *mh_make* ist zu beachten, dass dort, wo *mh_make* gestartet wird, alle Abhängigkeit des zu bauenden Paketes vorhanden sein sollten. Andernfalls sind etwaige Einträge in den einschlägigen Dateien (z.B. *debian/control*) manuell nachzuholen. (Kapitel 47.4, Seite 429)

Dazu empfiehlt es sich, diese Operationen in einer eigens dafür eingerichteten *Chroot* durchzuführen, in der alle Abhängigkeiten installiert werden können, ohne das eigentliche Host-System zu belasten. Die Einrichtung derselben ist in Kapitel 19.5 (Seite 72) beschrieben.

Von *mh_make* werden folgende Dateien im Verzeichnis *debian/* angelegt:

- *maven.cleanIgnoreRules*
- *maven.rules*
- *maven.ignoreRules*

³<https://manpages.debian.org/unstable/maven-debian-helper/index.html>

⁴[urlhttps://manpages.debian.org/unstable/maven-debian-helper/mh_make.1.en.html](https://manpages.debian.org/unstable/maven-debian-helper/mh_make.1.en.html)

- maven.properties
- <Paketname>.poms
- maven.publishedRules

Zusätzlich werden von *mh_make* auch die (Standard-)Dateien

- changelog
- control
- copyright
- rules
- README.source

erstellt. Dies ist bei den weiteren Arbeiten an diesen Dateien zu berücksichtigen (Kapitel 35.4, Seite 257).

Mit *apt-file find*⁵ kann festgestellt werden, ob es zu einer *artifactID* ein Debian-Paket gibt. Diese Funktion wird auch von *mh_make* genutzt. Die *artifactID* befindet sich in der *pom.xml*-Datei aus dem Upstream-Code.

Unter wiki.debian.org/Java/MavenPkgs/Unstable befindet sich eine Liste aller in Debian bereits paketierte *Maven*-Pakete.⁶

mh_make erstellt aus der *pom.xml* auch die Datei *debian/<Paketname>.poms*.

In dieser Datei werden alle im Paket vorkommende *pom.xml*-Dateien mit ihrem relativen Pfad aufgelistet. Diese können dann dort mit den aufgelisteten Optionen versehen werden.

```
45 <debian/JavaPackage.poms 45>≡
# List of POM files for the package
# Format of this file is:
# <path to pom file> [option]*
# where option can be:
# --ignore: ignore this POM and its artifact if any
# --ignore-pom: don't install the POM. To use on POM files that are created
# temporarily for certain artifacts such as Javadoc jars.
# [mh_install, mh_installpoms]
# --no-parent: remove the <parent> tag from the POM
# --package=<package>: an alternative package to use when installing
# this POM and its artifact
# --has-package-version: to indicate that the original version of the
# POM is the same as the upstream part of the version for the package.
# --keep-elements=<elem1,elem2>: a list of XML elements to keep in the POM
# during a clean operation with mh_cleanpom or mh_installpom
# --artifact=<path>: path to the build artifact associated with this POM,
# it will be installed when using the command mh_install. [mh_install]
# --java-lib: install the jar into /usr/share/java to comply with Debian
# packaging guidelines
# --usj-name=<name>: name to use when installing the library in
# /usr/share/java
# --usj-version=<version>: version to use when installing the library
# in /usr/share/java
# --no-usj-versionless: don't install the versionless link in
# /usr/share/java
```

⁵<https://manpages.debian.org/unstable/apt-file/apt-file.1.en.html>

⁶Dank an Thorsten Glaser und der Firma Tarent <https://www.tarent.de/> für diese Unterstützung

```

# --dest-jar=<path>: the destination for the real jar.
#   It will be installed with mh_install. [mh_install]
# --classifier=<classifier>: Optional, the classifier for the jar.
#   Empty by default.
# --site-xml=<location>: Optional, the location for site.xml if it
#   needs to be installed. Empty by default. [mh_install]
#

```

Eine häufig verwendete Zeile ist *pom.xml* `-no-parent -has-package-version`. Dies bedeutet, dass das `<parent>` Tag beim Bauen aus der POM entfernt wird. Die Option `-has-package-version` gibt an, dass die Originalversion der POM dieselbe sei, wie der Upstream-Teil der Paketversion.

Diese Datei kann innerhalb des Bauprozesses einer neuen Debian-Revision angepasst werden. (Kapitel 47.4, Seite 429).

Liegt die *pom.xml* im Wurzelverzeichnis des Quellcodes, werden durch *mh_make* die weiteren Dateien für das Verzeichnis *debian/* erstellt.

Im Maven-Repositorium wird die Datei *pom.xml* oft nur separat bereitgestellt. Diese Datei wird dann im *debian/* Verzeichnis abgelegt.

Daher sind hier einige Einträge in *debian/rules* (Kapitel 35.4.8, Seite 273 erforderlich:

Nach dem Aufruf aller *debhelper* (*dh \$@*) wird das *dh_auto_build* zunächst mit einem *override* außer Kraft gesetzt, um noch einige vorbereitenden Konfigurationen durchzuführen.

Dazu gehört auch das Kopieren von Dateien an den Ort, wo das Build-System diese Dateien benötigt, um das gewünschte Programm zu bauen.

```

46 <maven-build 46>≡
    override_dh_auto_build:
        cp debian/pom.xml .
        dh_auto_build

```

Weitere Literatur

1. <https://wiki.debian.org/Java/MavenBuilder>
2. <https://wiki.debian.org/Java/MavenRepoSpec>
3. <https://wiki.ubuntu.com/JavaTeam/Specs/MavenSupportSpec>

Mit *apt-file find*⁷ kann festgestellt werden, ob es zu einer *artifactID* ein Debian-Paket gibt. Diese Funktion wird auch von *mh_make* genutzt. Die *artifactID* befindet sich in der *pom.xml*-Datei aus dem Upstream-Code.

Unter wiki.debian.org/Java/MavenPkgs/Unstable befindet sich eine Liste aller in Debian bereits paketierte *Maven*-Pakete.⁸

13.4.3. Paketieren mit *ant*

Das Build-System *ant* erkennt man daran, dass es eine Datei *build.xml* gibt.⁹

13.4.4. Paketieren mit *gradle*

13.5. Java-Pakete bauen ohne Build-System

Es ist auch möglich, einfach eine Java-Bibliothek aus einem Verzeichnis mit *.java-Dateien zu bauen. Dies ist dann sinnvoll, wenn aus einem größeren Programmpaket nur eine oder wenige einzelne Java-Klasse benötigt wird, um Buildabhängigkeiten anderer Pakete zu erfüllen.

Dabei gibt es für ein solches Java-Verzeichnis kein Build-System wie die noch zu beschreibenden *ant* (Kapitel 13.4.3, Seite 47 bzw. *maven* (Kapitel 13.4.1, Seite 43.

```
47 <java-builds 47>≡
    export DH_VERBOSE=1
    export DH_OPTIONS=-v

    Class-Path: src/net/numericalchameleon/util/phoneticalphabets.jar
    export CLASSPATH=/usr/share/java/sugar.jar

    %:
        dh $@ --with javahelper --sourcedirectory=src/net/numericalchameleon/util/

    override_jh_build:

        javac -encoding UTF-8 src/net/numericalchameleon/util/phoneticalphabets/*
        jh_build
        jar cvf src/phoneticalphabets.jar \
        src/net/numericalchameleon/util/phoneticalphabets/*.class

        javac -encoding UTF-8 src/net/numericalchameleon/util/spokennumbers/*
        jh_build
```

⁷<https://manpages.debian.org/unstable/apt-file/apt-file.1.en.html>

⁸Dank an Thorsten Glaser und der Firma Tarent <https://www.tarent.de/> für diese Unterstützung

⁹<https://wiki.debian.org/Java/Packaging/Ant>

13. Oktober 2024

Das *javahelper*-Paket bietet Hilfestellung beim Bauen mit *dh*. Diese wird für die Java-Paketierung dringend empfohlen.

Ferner wird die Datei *debian/javabuild* benötigt. Diese enthält zwei Spalten.

Anzugeben ist in der ersten Spalte die zu bauende **.jar*-Datei und in der zweiten Spalte das Verzeichnis des Quellcodes, in dem die **.java*-Dateien liegen.

Beispiel:

```
#NameOfJarFile SourceDirToPackage
```

Das Skript ermöglicht das Erstellen dieser Datei (Kapitel 35.4.10, Seite 278).

14. Mozilla-Erweiterungen bauen

Das Paketieren von Mozilla-Erweiterungen als Debian-Pakete bietet für Nutzer und Administratoren den Vorteil, dass Erweiterungen zusammen mit der Hauptanwendung aktualisiert werden können.

Debian-Pakete werden zentral für alle Systemnutzer installiert. Die Aktualisierungen erfolgen auch mit dem Paketmanagementsystem der Distribution. Es bedarf dann nur eines Werkzeuges für diesen Zweck.

Bei Sicherheitsaktualisierungen oder neuen Versionen der Anwendungen sind diese Aspekte besonders relevant.

Dies hat den weiteren Vorteil, dass alle Nutzer mit den gleichen Versionsständen arbeiten.

14.1. Quellen der Erweiterungen

Unter addons.mozilla.org findet man Erweiterungen für den Firefox veröffentlicht von Mozilla.

Für den Thunderbird werden unter addons.thunderbird.net die Erweiterungen veröffentlicht.

Diese liegen dort als *.xpi*-Archive vor. Sofern dies Quellcode-Pakete sind, gibt es nicht zwingend auch separate Quellcode-Archive.

Im Falle einer *.xpi*-Datei bedarf es spezieller Einträge in der Datei *debian/watch* zum Umpaketieren. Das *.xpi*-Format ist ein besonders spezifiziertes *.zip*-Archiv. Nachstehend ist die *debian/watch*-Datei für die Thunderbird-Erweiterung Mailmindr dargestellt.

```
49 <watch4xpi 49>≡
    version=4
    opts=\
    repack,compression=xz,\
    uversionmangle=s/-?([\d.]+)/~$1/;tr/A-Z/a-z/,\
    filenamemangle=s/.\+\/v?(\d\S+)\.*/$1/,
    https://addons.thunderbird.net/en-US/thunderbird/addon/mailmindr/versions/ \
    (\d.\d.\d)
```

Manchmal findet man auch Quellcode-Veröffentlichungen von Personen und Projekten beispielsweise auf Github (github.com) oder auf selbstgehosteten Seiten. Diese können auch als *.tar.gz*- oder *.zip*-Archive vorliegen.

Erfahrungsgemäß sind dort die Versionen, die bereits im *addons*-Archiv veröffentlicht worden sind, nicht immer zeitnah als neue Version gekennzeichnet.

14.2. Integration ins Dateisystem

Um alle gewünschten **Mozilla**-Erweiterungen einheitlich zu bauen, werden die benötigten Teile des Skriptes als *Webext-Plugin* realisiert (Kapitel 48, Seite 433). Damit wird auch eine einheitliche Integration in das System gewährleistet.

Ab der Version (≥ 78.2) des **Thunderbirds** können ausschließlich *Mail=Extensions* genutzt werden. Die alten (für TB $\leq 68.x$) Erweiterungen funktionieren nicht mehr.

Die Erweiterung muss nun im Verzeichnis */usr/share/webext* mit der Bezeichnung (id) aus der Datei *manifest.json* ergänzt und mit der Endung (*.xpi*) als Zip-Archiv abgelegt werden (Kapitel 48.2.4, Seite 437).

Dazu wird die Datei *debian/rules* entsprechend erweitert (Kapitel 48.2.2, Seite 435). Zusätzlich sind immer die Dateien *<Paketname>.install* (Kapitel 48.2.4, Seite 437) und *<Paketname>.links* (Kapitel 48.2.6, Seite 438) so zu erstellen, dass **Thunderbird** die Erweiterung auch findet.

15. Python-Pakete bauen

16. Dokumentation paketieren

16.1. Dokumentation für Debian bauen

Diese Dokumentation kann als Beispiel dafür dienen, wie man Dokumentationen als Debian-Paket bereitstellt.

Diese Dokumentation ist bereits auf salsa.debian.org¹ veröffentlicht. Dies bietet die Möglichkeit, das Buch als **natives Debian**-Paket zu bauen.

Ein natives Quellpaket ist nach der Debian-Policy² ein Paket, das nicht zwischen Debian-Paketveröffentlichungen und Upstream-Veröffentlichungen unterscheidet. Ein natives Quellpaket enthält eine einzelne *tar*-Datei mit Quellmaterial, und die Versionierung hat keine Debian-spezifische Komponente. Native Pakete werden normalerweise (aber nicht ausschließlich) für Software verwendet, die keine unabhängige Existenz außerhalb von Debian hat, wie z. B. Software, die speziell für ein Debian-Paket geschrieben wurde. Gleiches gilt für Dokumentation, die speziell für Debian geschrieben wurde. Ein nicht-natives Quellpaket trennt die Upstream-Veröffentlichung von der Debian-Paketierung und allen Debian-spezifischen Änderungen.

Die Information, dass es sich um ein natives Paket handelt, wird in der Datei *debian/source/format* als *3.0 (native)* abgelegt (Kapitel 35.4.2, Seite 260).

Native Pakete benötigen keine Datei *debian/upstream/metadata* (Kapitel 35.4.4, Seite 262). und keine Datei *debian/watch* (Kapitel 35.4.7, Seite 269). Die Versionsbezeichnung enthält keine Revisionsbezeichnung (Kapitel 37.1.1, Seite 323 und Kapitel 37.5.3, Seite 346).

Das Bauen *nativer* Pakete wird vom Programmskript unterstützt (Kapitel 37.5.3, Seite 346).

16.2. Upstream-Dokumentation bauen

Wird vom Upstream-Projekt eine umfangreiche oder mehrsprachige Dokumentation bereitgestellt, empfiehlt es sich, diese Dokumentation in gesonderten Debian-Paketen zu veröffentlichen.

¹<https://salsa.debian.org/ddp-team/dpb>

²Kapitel 4 der Debian-Policy[7]

17. Metapakete bauen

Metapakete sind solche Debian-Pakete, die von einer Gruppe von Paketen abhängen. Damit werden Gruppen von Paketen zusammengefasst, bei denen es sinnvoll ist, sie gemeinsam zu installieren. Beispiele sind Erweiterungen für eine bestimmte Software.

Metapakete weisen einige Besonderheiten auf.

17.1. Kein Upstream-Quellcode

Da es keinen Upstream-Quellcode gibt, benötigt ein Metapaket auch keine Datei *debian/watch*

17.2. Natives Debian-Paket

17.2.1. *debian/source/format*

17.2.2. *debian/control*

17.2.3. *debian/rules*

17.2.4. *debian/changelog*

18. Konfiguration zur Installation

18.1. *debconf*

Debconf ist ein Konfigurationsmanagementsystem für Debian. Dies dient dazu, Informationen für die Installation vom Nutzer abzufragen und diese dann bei der Installation zu berücksichtigen.¹

Mit `sudo dpkg-reconfigure debconf` kann *debconf* selbst konfiguriert werden. Dies beinhaltet auch die grafische Oberfläche.

first step: <https://wiki.debian.org/debconf>
<https://wiki.debian.org/debconf>

18.2. *dbconfig-common*

Dbconfig-common stellt eine einfache, zuverlässige und konsistente Methode zur Verwaltung von Datenbanken bereit, die von Debian-Paketen verwendet werden.

¹<https://wiki.debian.org/debconf>

19. System einrichten

Es wird davon ausgegangen, dass bereits ein installiertes und lauffähiges **Debian**-System existiert. Dabei ist es unerheblich, welcher **Zweig** (*unstable*, *testing* oder *stable*) darauf läuft.

Zum Signieren der vom Programmskripts erzeugten Dateien muss ein **GPG**-Schlüssel des Nutzers auf dem System zur Verfügung stehen. Dieser sollte sinnvollerweise der für das **Debian**-Repositorium vorgesehene Schlüssel sein.

Dies gilt auch, wenn für das Bauen eine virtuelle Maschine eingerichtet und genutzt wird.

19.1. Abhängigkeiten für das Programmskript

Die Pakete, die zum Einsatz des Programmskriptes erforderlich sind, sind in den Kopfzeilen desselben aufgeführt (Kapitel 45.3, Seite 412).

Einige der darin enthaltenen Programme werden im folgenden beschrieben:

19.1.1. Generelle Abhängigkeiten

Die nachstehend aufgeführten, hilfreichen Programme werden regelmäßig vom Programmskript zum Bauen eines **Debian**-Paketes eingesetzt:

mk-origtargz aus dem Paket *devscripts* benennt den Tarball der Originalautoren um, ändert wahlweise die Komprimierung und entfernt unerwünschte Dateien. Dazu stehen unterschiedliche Optionen zur Verfügung[33] (s. Verwendung in Kapitel 34.4.6, Seite 226).

gbp import-orig aus dem Paket *git-buildpackage* importiert eine neue Upstream-Version in ein **Git**-Repositorium (s. Verwendung in Kapitel 34.4.11, Seite 243).

gbp import-dsc aus dem Paket *git-buildpackage* importiert ein existierendes **Debian**-Quellpaket in ein **Git**-Repositorium (s. Verwendung in Kapitel 32.6, Seite 168). Auf diesem Weg kann ein Mentor ein Paket „sponsorn“ (Kapitel 32.7, Seite 170).

dh_make aus dem Paket *dh-make* kann die benötigten **Debian**-Dateien erstellen. Die Praxis zeigt, dass dies sehr unvollständig ist und einiges an Handarbeit erfordert. Daher wird bisher davon abgesehen, dieses Hilfsprogramm im Programmskript einzusetzen. (s. Kapitel 12, Seite 39)

debmake -cc aus dem gleichnamigen Paket *debmake* durchsucht die Quelldateien nach Copyright- und Lizenztexten (s. Kapitel 10.1.1, Seite 27). Dieser Befehl wird vom Programmskript zur Erstellung der Datei *debian/copyright* verwandt. (s. Verwendung in Kapitel 35.4.5, Seite 264).

gbp dch ist auch im Paket *git-buildpackage* enthalten. Im Programmskript wird *dch* über *gbp dch* verwandt (s. Verwendung in Kapitel 37.1, Seite 321). Weitere Optionen für *dch* können an *gbp dch* durch *--dch-opt=<dch-Optionen>* übergeben werden.

gbp buildpackage ist ebenfalls im Paket *git-buildpackage* enthalten. Mit diesem Befehl wird ein Paket aus dem Git-Repository erstellt (s. Verwendung in Kapitel 37.5.4, Seite 351)

lintian aus dem Paket *lintian* ist ein Analysewerkzeug für Debian-Pakete. Es berichtet Fehler und Verletzungen der Debian-Policy [7]. Jeder Debian-Betreuer sollte mit diesem Werkzeug das Paket vor dem Hochladen ins Archiv prüfen. Das Programmskript erwendet *lintian* mit den Optionen *--pedantic* und *--display-experimental*. (s. Verwendung in Kapitel 40.3, Seite 360).

Während des Bauens wird *lintian* durch *dh_lintian* aus dem Paket *devscripts* aufgerufen.

uscan in *devscripts*

debsign in *devscripts* auch zum Signieren beim Sponsoring

dput in *dput* bzw *dput-ng*

mh_make (Plugin)

19.1.2. Abhängigkeiten für das Bauen von Java-Paketen

Die folgenden Abhängigkeiten werden für das Bauen mit Build-Systemen für Java-Pakete benötigt.

```
gradle-debian-helper, maven-debian-helper, libmaven-bundle-plugin-java,
```

19.1.3. Abhängigkeiten für Erweiterungen, die Zip-Archive sind

Manche Erweiterungen (Plugins) werden als *Zip-Archive* (z. B. für Libreoffice *.oxt*, für Mozilla *.xpi*) verteilt, die den Quellcode enthalten.

Für diese Erweiterungen muss in der Datei *debian/control* (Kapitel 35.4.6, Seite 265) *zip* als Buildabhängigkeit aufgeführt werden.

Für Mozilla-Erweiterungen wird dies vom *Web-Extension-Plugin* erledigt (Kapitel 48.2.3, Seite 437).

19.2. Verzeichnisse und Dateien

19.2.1. Pfade zu den Projekten

Es ist sinnvoll, die Paketierprojekte jeweils als Unterverzeichnisse eines Projektverzeichnisses anzulegen. Ein Beispiel für ein solches Projektverzeichnis ist:

```
~/Projekte/Git/01_Salsa
```

Dieser Pfad wird in jeder Konfigurationsdatei in der Variablen *ProjectPath* hinterlegt.

Ferner kann dieser Pfad auch in der Variablen *DefaultProjectPath* für viele Projekte als Standardwert in der Datei *~/debian_project/DefaultValues* aufgenommen werden (Kapitel 19.2.2.2, Seite 62).

PrjPath ist das Unterverzeichnis des Projektverzeichnisses (*ProjectPath*), in dem das einzelne Projekt liegt. Der Name dieses Unterverzeichnisses ist der Name des entsprechenden Projektes (*OrigName*). In dieses Unterverzeichnis werden die fertigen Pakete abgelegt.

In diesem Unterverzeichnis befindet sich ein weiteres Unterverzeichnis (*GitPath*), welches das Git-Repository enthält.

Diese Unterverzeichnisse werden, wenn nötig, vom Programmskript angelegt (s. a. Kapitel 32.2.1, Seite 138)

19.2.2. Konfigurationsdateien

19.2.2.1. Für jedes Projekt

Für jedes Projekt wird eine eigene Konfigurationsdatei erstellt. Diese wird im Homeverzeichnis des Nutzers im Verzeichnis *.debian_project/* als Datei *<Projektname>* abgelegt. Darin werden projektspezifische Informationen hinterlegt.

Die Konfigurationsdatei ist technisch ein Shell-Skript, welches von dem Programmskript erstellt und geladen wird. Dieses Shell-Skript enthält Variablen, denen dort Werte zugewiesen werden. Die in der Konfigurationsdatei mit Werten versehenen Variablen können durch das Laden vom Programmskript verwendet werden.

Ferner enthält die Datei Kommentare. Diese können vom Nutzer beliebig erstellt werden. Ein Teil der Kommentare wird jedoch vom Programmskript erstellt. Diese enthalten ein Eigenschafts-Werte-Paar, welches wie die Zuweisung eines Wertes an eine Variable aufgebaut ist. Technischer Hintergrund ist, dass im Eigenschaftsnamen sinnvollerweise Zeichen vorkommen, die im Bezeichner einer Variablen nicht verwendet werden dürfen.

Existiert die Konfigurationsdatei, so wird sie zunächst geladen. (Kapitel 33.1, Seite 177). Man kann sie dann, wenn nötig, editieren. Andernfalls wird diese Datei durch das Skript erstellt. (Kapitel 32.1, Seite 117)

Folgende Informationen werden dort hinterlegt:

```
# !/usr/bin/bash
# ConfigFile for <OrigName>
## General parameters
SourceName
PackName
ProjectPath = ~/Projekte/Git/01_Salsa
SalsaName
Java-Package
    SalsaName = java-team/<SourceName>.git
JavaFlag
MavenPluginFlag
Maintainer =Maintainer=Debian_Java_Maintainers
    _@lt@pkg-java-maintainers@lists.aliioth.debian.org@gt@
Uploader =Mechtilde_Stehmann_@lt@mechtilde@debian.org@gt@
Web-Extension-Packages
    SalsaName = webext-team/
WebextFlag
Maintainer
Uploader
Python-Packages SalsaName = python-team/packages/
    PythonFlag
Maintainer
Uploader
DefaultBranch
```

```
RecentBranch = debian/sid
RecentUpstreamSuffix = .tar.gz
RecentRepackSuffix = +dfsg | +ds
master_Dist
DownloadUrl
DownloadZip
Maintainer Mechtilde Stehmann <mechtilde@debian.org>
ExcludeFile
```

19.2.2.2. Für viele Projekte

In der Datei `~/.debian_project/DefaultValues` werden Variablen Werte zugewiesen, die für viele Projekte gelten.

```
62 <DefaultValues 62>≡
    #!/usr/bin/bash

    HOME=/home/<user>
    DefaultProjectPath=${HOME}/Projekte/Git/01_Salsa
```

19.2.2.3. Fingerprint des Maintainer-Schlüssels

Im Verzeichnis `~/.debian_project/` befindet sich auch eine Datei *fingerprint*, die den Fingerprint des Maintainer-Schlüssels enthält. Mit diesem Schlüssel werden die Pakete signiert . (Kapitel 32.9, Seite 173)

19.2.3. *.bashrc*

In die Datei *.bashrc* sind folgende Einträge aufzunehmen:

```
DEBFULLNAME="<Vor- und Nachname des Maintainer>"
DEBEMAIL="<E-Mail-Adresse des Maintainers>"
export DEBEMAIL DEBFULLNAME
```

Verschiedene Debian-Werkzeuge erkennen Ihre E-Mail-Adresse und Ihren Namen anhand der Shell-Umgebungsvariablen `$DEBEMAIL` und `$DEBFULLNAME`. ¹

Diese Einträge werden auch von der Funktion *DEBValues* des Programmskripts ausgelesen. (Kapitel 32.4.2, Seite 145)

Auch für *quilt* ist die Datei *.bashrc* zu ergänzen. (Kapitel 19.6, Seite 72)

19.3. *PBuilder* einrichten

Gbp buildpackage verwendet *cowbuilder* . Der Befehl *cowbuilder* ist ein Wrapper für *pbuilder*, welcher die Nutzung eines *pbuilder*-ähnlichen Interfaces in einer *cowdancer*-Umgebung ermöglicht.

¹Kapitel 3.1 in [11]

19.3.1. Chroot

Das Paket *pbuilder* enthält Programme, um eine *Chroot*-Umgebung aufzubauen und zu betreiben.

Chroot bedeutet *Change root*

Es entspricht guter Praxis, *Debian*-Pakete in einer *Chroot* zu bauen. Damit soll gewährleistet werden, dass das Paket mit den Ressourcen der jeweilige Distribution gebaut werden kann.

Beim Bauen eines *Debian*-Paketes in dieser *Chroot*-Umgebung wird nämlich geprüft, ob alle erforderlichen Build-Abhängigkeiten erfüllbar sind. Hierdurch können FTBFS-Fehler (FTBFS = Failed To Build From Source) vermieden werden. Unter Umständen kann es jedoch noch zu solchen Fehlermeldungen kommen.

19.3.2. Konfiguration des Pbuilders

Zunächst muss das Verzeichnis */var/cache/pbuilder/result* angelegt werden. Dieses Verzeichnis muss für den Nutzer beschreibbar sein. Danach wird die Konfiguration des *pbuilder* durchgeführt.

Die Konfiguration des *pbuilder* ist nicht völlig trivial, was gegebenenfalls die Fehlersuche erschwert. Daher gehen wir hier sehr ausführlich darauf ein.

Die Standard-Konfiguration wird aus der Datei */usr/share/pbuilder/pbuilderrc* genommen.

```
# pbuilder defaults; edit /etc/pbuilderrc to override these and see
# pbuilderrc.5 for documentation

# Set how much output you want from pbuilder, valid values are
# E => errors only
# W => errors and warnings
# I => errors, warnings and informational
# D => all of the above and debug messages
LOGLEVEL=I
# if positive, some log messages (errors, warnings, debugs) will be colored
# auto => try automatic detection
# yes => always use colors
# no  => never use colors
USECOLORS=auto

BASETGZ=/var/cache/pbuilder/base.tgz
#EXTRAPACKAGES=""
#export DEBIAN_BUILDARCH=athlon
BUILDPLACE=/var/cache/pbuilder/build
# directory inside the chroot where the build happens. See #789404
BUILDDIR=/build
# what be used as value for HOME during builds. See #441052
# The default value prevents builds to write on HOME, which is prevented on
# Debian buildds too. You can set it to $BUILDDIR to get a working HOME, if
# you need to.
BUILD_HOME=/nonexistent
MIRROR SITE=http://deb.debian.org/debian
#OTHERMIRROR="deb http://www.home.com/updates/ ./"
#export http_proxy=http://your-proxy:8080/
USESHM=yes
USEPROC=yes
USEDEVFS=no
USEDEVPTS=yes
USESYSFS=yes
```

13. Oktober 2024

```
USENETWORK=no
USECGROUP=yes
BUILDRESULT=/var/cache/pbuilder/result/

# specifying the distribution forces the distribution on "pbuilder update"
#DISTRIBUTION=sid
# specifying the architecture passes --arch= to debootstrap; the default is
# to use the architecture of the host
#ARCHITECTURE=$(dpkg --print-architecture)
# specifying the components of the distribution, for instance to enable all
# components on Debian use "main contrib non-free" and on Ubuntu "main
# restricted universe multiverse"
COMPONENTS="main"
#specify the cache for APT
APTCACHE="/var/cache/pbuilder/aptcache/"
APTCACHEHARDLINK="yes"
REMOVEPACKAGES=""
#HOOKDIR="/usr/lib/pbuilder/hooks"
HOOKDIR=""
EATMYDATA=no
# NB: this var is private to pbuilder; ccache uses "CCACHE_DIR" instead
# CCACHEDIR="/var/cache/pbuilder/ccache"
CCACHEDIR=""

# make debconf not interact with user
export DEBIAN_FRONTEND="noninteractive"

#for pbuilder debuild
BUILDSOURCEROOTCMD="fakeroot"
PBUILDERROOTCMD="sudo -E"
# use cowbuilder for pdebuild
#PDEBUILD_PBUILER="cowbuilder"

# Whether to generate an additional .changes file for a source-only upload,
# whilst still producing a full .changes file for any binary packages built.
SOURCE_ONLY_CHANGES=no

# additional build results to copy out of the package build area
#ADDITIONAL_BUILDRESULTS=(xunit.xml .coverage)

# command to satisfy build-dependencies; the default is an internal shell
# implementation which is relatively slow; there are two alternate
# implementations, the "experimental" implementation,
# "pbuilder-satisfydepends-experimental", which might be useful to pull
# packages from experimental or from repositories with a low APT Pin Priority,
# and the "aptitude" implementation, which will resolve build-dependencies and
# build-conflicts with aptitude which helps dealing with complex cases but does
# not support unsigned APT repositories
PBUILDERSATISFYDEPENDSCMD="/usr/lib/pbuilder/pbuilder-satisfydepends"

# Arguments for $PBUILDERSATISFYDEPENDSCMD.
# PBUILDERSATISFYDEPENDSOPT=()

# You can optionally make pbuilder accept untrusted repositories by setting
# this option to yes, but this may allow remote attackers to compromise the
# system. Better set a valid key for the signed (local) repository with
# $APTKEYRINGS (see below).
ALLOWUNTRUSTED=no

# Option to pass to apt-get always.
export APTGETOPT=()
# Option to pass to aptitude always.
export APTITUDEOPT=()

# Whether to use debdelta or not. If "yes" debdelta will be installed in the
```

```

# chroot
DEBDELTA=no

#Command-line option passed on to dpkg-buildpackage.
#DEBBUILDPOPTS="-IXXX -iXXX"
DEBBUILDPOPTS=""

#APT configuration files directory
APTCONFDIR=""

# the username and ID used by pbuilder, inside chroot. Needs fakeroot, really
BUILDUSERID=1234
BUILDUSERNAME=pbuilder

# BINDMOUNTS is a space separated list of things to mount
# inside the chroot.
BINDMOUNTS=""

# Set the debootstrap variant to 'buildd' type.
DEBOOTSTRAPOPTS=(
  '--variant=buildd'
  '--force-check-gpg'
)
# or unset it to make it not a buildd type.
# unset DEBOOTSTRAPOPTS

# Keyrings to use for package verification with apt, not used for debootstrap
# (use DEBOOTSTRAPOPTS). By default the debian-archive-keyring package inside
# the chroot is used.
APTKEYRINGS=()

# Set the PATH I am going to use inside pbuilder: default is
# "/usr/sbin:/usr/bin:/sbin:/bin"
export PATH="/usr/sbin:/usr/bin:/sbin:/bin"

# SHELL variable is used inside pbuilder by commands like 'su';
# and they need sane values
export SHELL=/usr/bin/bash

# The name of debootstrap command, you might want "cdebootstrap".
DEBOOTSTRAP="debootstrap"

# default file extension for pkgname-logfile
PKGNAME_LOGFILE_EXTENSION="_$(dpkg --print-architecture).build"

# default PKGNAME_LOGFILE
PKGNAME_LOGFILE=""

# default AUTOCLEANAPTCACHE
AUTOCLEANAPTCACHE=""

#default COMPRESSPROG
COMPRESSPROG="gzip"

# pbuilder copies some configuration files (like /etc/hosts or
# /etc/hostname)
# from the host system into the chroot. If the directory specified here
# exists and contains one of the copied files (without the leading /etc) that
# file will be copied from here instead of the system one
CONFDIR="/etc/pbuilder/conf_files"

```

Diese Werte können gegebenenfalls von Werten in der Datei */etc/pbuilderrc* überschrieben werden.

Nach einer Neuinstallation sieht die */etc/pbuilderrc* wie folgt aus:

13. Oktober 2024

```
# this is your configuration file for pbuilder.  
# the file in /usr/share/pbuilder/pbuilderrc is the default template.  
# /etc/pbuilderrc is the one meant for overwriting defaults in  
# the default template  
#  
# read pbuilderrc.5 document for notes on specific options.  
MIRRORSITE=http://ftp.de.debian.org/debian/
```

Diese habe ich wie folgt eingerichtet:

```
# this is your configuration file for pbuilder.  
# the file in /usr/share/pbuilder/pbuilderrc is the default template.  
# /etc/pbuilderrc is the one meant for overwriting defaults in  
# the default template  
#  
# read pbuilderrc.5 document for notes on specific options.  
# adapt from Mechtilde - 2019-09-07 (analog wiki)  
MIRRORSITE=http://ftp.de.debian.org/debian/  
AUTO_DEBSIGN=${AUTO_DEBSIGN:-no}  
HOOKDIR=/var/cache/pbuilder/hooks  
# Codenames for Debian suites according to their alias.  
# Update these when needed.  
# EXPERIMENTAL_CODENAME ="experimental"  
UNSTABLE_CODENAME="sid"  
TESTING_CODENAME="bookworm"  
STABLE_CODENAME="bullseye"  
STABLE_BACKPORTS_SUITE="${STABLE_CODENAME}-backports"
```

Neben der globalen Konfigurationsdatei */etc/pbuilderrc* kann auch eine nutzerspezifische Datei *~/.pbuilderrc* angelegt werden. Der Inhalt dieser Datei überschreibt die systemweiten Einstellungen.

```
# BINDMOUNTS is a space separated list of things to mount  
# inside the chroot.  
BINDMOUNTS="/var/local/repository" # lokales Verzeichnis einbinden (mounten)  
OTHERMIRROR="deb http://deb.debian.org/debian/ buster-backports main \  
| deb [trusted=yes] file:///var/local/repository ./"  
# OTHERMIRROR="${OTHERMIRROR} | deb file:///var/local/repository ./"  
# Fertige Pakete im lokalen Repository ablegen  
# BUILDRESULT=..  
  
# Added after reading:  
# https://lists.debian.org/debian-backports/2018/09/msg00021.html  
  
# List of Debian suites.  
DEBIAN_SUITES=($UNSTABLE_CODENAME $TESTING_CODENAME \  
$STABLE_CODENAME $STABLE_BACKPORTS_SUITE  
"experimental" "unstable" "testing" "stable")  
  
# Mirrors to use. Update these to your preferred mirror.  
DEBIAN_MIRROR="ftp.de.debian.org"  
  
# Added after reading https://wiki.debian.org/cowbuilder  
BASEPATH="/var/cache/pbuilder/base.cow/"
```

Neben den systemweiten und nutzerspezifischen Konfigurationen werden auch paketspezifische Konfigurationen des Pbuilders benötigt. Diese Dateien können im Projektverzeichnis abgelegt und mit *--configfile <Konfigurationsdatei>* eingelesen werden. Mit dieser Konfiguration werden eventuell schon vorhandene Werte überschrieben. Hier können

dann die Informationen zu den Projekten abgelegt werden, wenn für diese Veröffentlichungen im Backports-Zweig benötigt werden. Diese Datei wird nach allen anderen Konfigurationsdateien eingelesen.

Siehe dazu auf Seite

<https://wiki.debian.org/BuildingFormalBackports>

den Abschnitt *#Advanced: `_Building_multi-dependency_packages`*

Dies kann auch in der Datei `~/pbuilderrc` hinzugefügt werden, wenn es diese Datei gibt. Als MIRROR wird ein gängiger gut erreichbarer Debian-Mirror angegeben. Falls vorhanden, kann hier auch die Adresse eines vorhandenen lokalen Spiegel eingetragen werden.

Für die *Hook*-Skripte ist ein Verzeichnis `~/pbuilder/` anzulegen.

19.3.3. Hooks einrichten

Hooks sind Skripte, die an bestimmten, vordefinierten Punkten während des Bauprozesses Dinge erledigen. Mit den sogenannten *Hooks* (Haken) kann der Prozess in der Build-Chroot auch an vordefinierten Positionen unterbrochen werden, um noch manuell in den Prozess eingreifen zu können.

Die *Hook*-Skripte befinden sich im Verzeichnis `~/pbuilder/`

Der Name des Hook-Skriptes bestimmt, an welcher Stelle im Bauprozess der *Hook* ausgeführt wird.

Dabei gilt folgende Konversation:

```
X<digit><digit><whatever-else-you-want-as-name>
```

Das „X“ (A bis G) bestimmt die Hook-Klasse, die folgenden 2 Ziffern die Reihenfolge, in der die Hooks einer Klasse ausgeführt werden. Der Rest dient als Beschreibung.

Leider entspricht die Reihenfolge, in der die Klassen im Build-Prozess ausgeführt werden, nicht der alphabetischen Reihenfolge.

- A** Ist für das `--build` Ziel. Es wird vor dem Baubeginn ausgeführt. D.h. nach dem Entpacken des Bausystems, des Quelltextes und nach der Erfüllung der Bau-Abhängigkeit.
- B** Wird ausgeführt, nachdem das Bausystem den Bau erfolgreich abgeschlossen hat, bevor das Bauergebnis zurückkopiert wird. - Unterbrechung nach erfolgreichem Bauen
- C** Wird nach einem Build-Fehler, vor der Bereinigung ausgeführt. - Unterbrechung nach gescheitertem Build
- D** Wird vor dem Entpacken der Quelle innerhalb der Chroot-Umgebung ausgeführt, nachdem die Chroot-Umgebung eingerichtet wurde. Erstellt `$TMP` und `$TMPDIR` wenn erforderlich. Dies wird aufgerufen, bevor die Build-Abhängigkeit befriedigt ist. Auch nützlich für den Aufruf von `apt update`. – Möglichkeit zum Editieren der `Sources.list`.²
- E** Wird ausgeführt, nachdem `pbuilder --update` und `pbuilder --create` die Arbeit von `apt-get` mit dem Chroot beendet hat, bevor das Kernel-Dateisystem (`/proc`) umountet und der Tarball aus dem Chroot erzeugt wird.

²weiteres s.a. https://wiki.debian.org/PbuilderTricks#How_to_include_local_packages_in_the_build

- F** Is executed just before user logs in, or program starts executing, after chroot is created in --login or --execute target.
- G** Is executed just after debootstrap finishes, and configuration is loaded, and pbuilder starts mounting /proc and invoking apt install in --create target.
- H** Wird unmittelbar nach dem Auspacken von chroot, mounting proc und jedem in BIND-MOUNTS angegebenen bind mount ausgeführt. Es wird für jedes Ziel ausgeführt, das die entpackte Chroot benötigt. Es ist nützlich, wenn Sie die Chroot-Einbauten dynamisch ändern wollen, bevor irgendetwas anfängt, sie zu benutzen.
- I** Wird ausgeführt, nachdem das Bausystem den Bau erfolgreich abgeschlossen hat, nach dem Zurückkopieren der Build-Ergebnisse.

19.3.4. Hooks - Beispiele

Diese stehen alle im Verzeichnis: `~/pbuilder/`

19.3.4.1. Hook A

Dieser Hook könnte z.B. *A10shell* heißen.

```
68a <Hook-A 68a>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell before build starts.

BUILDDIR="${BUILDDIR:-/tmp/buildd}"

apt-get install -y "${APTGETOPT[@]}" nano less
cd "$BUILDDIR"/*/debian/..
echo "Hook A - the dependencies are installed. Now the build can start."
echo "Please use CTRL-D to continue"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.2. Hook B

Dieser Hook könnte z.B. *B20shell* heißen.

```
68b <Hook-B 68b>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell if build fails.

BUILDDIR="${BUILDDIR:-/tmp/buildd}"

# apt-get install -y "${APTGETOPT[@]}" vim less
cd "$BUILDDIR"/*/debian/..
echo "Hook B - The build was built successfully"
echo "You can check it with ls -la ../"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.3. Hook C

Hier können dann noch Pakete hinzu installiert werden, die zwingend benötigt werden, wenn der Build fehlschlägt.

Dieser Hook könnte z.B. *C10shell* heißen.

```
69a <Hook-C 69a>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell if build fails.

BUILDDIR="${BUILDDIR:-/tmp/builddd}"

apt-get install -y "${APTGETOPT[@]}" vim less mc unzip locate
cd "$BUILDDIR"/*/debian/..
echo "Hook C - The build wasn't built successfully"
echo "After analysing the errors you can continue with using CTRL-D"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.4. Hook D

```
69b <Hook-D 69b>≡
#!/usr/bin/bash
# example file to be used with --hookdir
#
# invoke shell before unpacking the source
# inside the chroot

BUILDDIR="${BUILDDIR:-/tmp/builddd}"

apt-get install -y "${APTGETOPT[@]}" less nano
#cd "$BUILDDIR"/*/debian/..
echo "Hook D -"
echo "After unpacking the sources the dependencies"
echo "can be downloaded and unpacked."
echo "Please use CTRL-D to continue"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.5. Hook E

```
69c <Hook-E 69c>≡
echo "Hook E"
echo "Please use CTRL-D to continue"
/usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.6. Hook F

```
70a <Hook-F 70a>≡
    echo "Hook F"
    echo "Please use CTRL-D to continue"
    /usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.7. Hook G

```
70b <Hook-G 70b>≡
    echo "Hook G"
    echo "Please use CTRL-D to continue"
    /usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.8. Hook H

```
70c <Hook-H 70c>≡
    #!/usr/bin/bash
    # example file to be used with --hookdir
    #
    # invoke shell if build fails.

    BUILDDIR="${BUILDDIR:-/tmp/buildd}"

    echo "Hook H"
    echo "Executed after preparing the chroot \n and before installing the dependencies"
    echo "Here you can include dependency from e.g a local repo for testing."
    echo "Next the source code of the package to be built is unpacked."
    echo "Please use CTRL-D to continue"
    /usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.4.9. Hook I

```
70d <Hook-I 70d>≡
    #!/usr/bin/bash
    # example file to be used with --hookdir
    #
    # invoke shell if build fails.

    BUILDDIR="${BUILDDIR:-/tmp/buildd}"

    #apt-get install -y "${APTGETOPT[@]}" vim less
    #cd "$BUILDDIR"/*/debian/..
    echo "Hook I"
    echo "Please use CTRL-D to continue"
    /usr/bin/bash < /dev/tty > /dev/tty 2> /dev/tty
```

19.3.5. Alternative *Chroot*-Umgebungen

Es hat sich bewährt, für die verschiedenen *Debian*-Zweige eigene *Chroot*-Umgebungen bereitzustellen.

Dazu wird eine Kopie von dem Verzeichnis `/var/cache/pbuilder/base.cow` angelegt. Darin kann dann z.B. Die Datei `/etc/apt/sources.lists` entsprechend angepasst werden.

Dabei kann es bei einer Aktualisierung der Pakete notwendig sein, dass zunächst auch Abhängigkeiten dieser Pakete aktualisiert werden müssen. Diese stehen jedoch nur mit Zeitverzögerung im Repo zur Verfügung. Hier kann es helfen für die weiteren Tests schon einmal die Zeile

```
deb http://incoming.debian.org/debian-buildd buildd-unstable main
```

in der `/etc/apt/sources.lists` zu aktivieren. Damit stehen die dortigen Pakete für das Bauen in der *Pbuilder*-*Chroot* zur Verfügung.

19.4. Konfiguration von *sbuild*

sbuild kann ohne Root-Rechte genutzt werden. In diesem Fall ist der Paketbetreuer in die Gruppe *sbuild* aufzunehmen.

Für diese Nutzung sind folgende Pakete zu installieren.[34]

```
sudo apt install sbuild mmdebstrap uidmap
```

Die benötigten Verzeichnisse sind wie folgt zu installieren:

```
mkdir -p ~/.cache/sbuild
```

Nun wird der Tarball der *Chroot* erstellt.

```
mmdebstrap --variant=buildd unstable ~/.cache/sbuild/unstable-amd64.tar.zst
```

Danach wird eine neue Konfigurationsdatei `~/.sbuildrc` erstellt. Eine bereits bestehende Datei wird überschrieben. Sollten Sie bereits eine `~/.sbuildrc` haben, erstellen Sie bei Bedarf eine Sicherungskopie.

```
71 <.sbuild.rc 71>≡
    $chroot_mode = 'unshare';
    $distribution = 'unstable';
    #run_autopkgtest = 1;
    $autopkgtest_root_args = '';
    $autopkgtest_opts = [ '--apt-upgrade', '--', 'unshare', '--release', '%r', '--arch', '%a' ];
```

19.5. Weitere *Chroot*-Systeme

Neben dem Bauen in einer *pbuilder*-Chroot gibt es noch weitere Situationen, in denen die Nutzung eines separates Systems nützlich sein kann. Dies gilt besonders für das Ausführen von *mh_make*. (Kapitel 47.3, Seite 420)

Diese Einrichtung einer *Maven-Chroot* wird als Beispiel beschrieben:

Zunächst ist das Paket *debootstrap*, sofern noch nicht vorhanden, zu installieren. Es wird mit

```
sudo mkdir --parents /srv/maven-chroot
```

ein entsprechendes Verzeichnis angelegt. Die Chroot selber wird mit

```
sudo /usr/sbin/debootstrap --arch amd64 sid \  
/srv/maven-chroot http://ftp.de.debian.org/debian
```

erstellt.^[35]

Danach kann der Nutzer *root* mit dem Befehl *chroot* ein neues Wurzelverzeichnis starten.

Die konkreten Befehle lauten (als *root*):

```
# mount --options bind /proc /srv/maven-chroot/proc  
# mount devpts /dev/pts --types devpts  
# LANG=C chroot /srv/maven-chroot /usr/bin/bash
```

Die letzte Zeile startet die angelegte Chroot.

Dieser User *root* kann dann nicht mehr auf Dateien außerhalb des neuen Wurzelzeichnisses zugreifen.

Die Chroot-Umgebung kann wie folgt wieder beseitigt werden:

```
sudo umount /srv/maven-chroot/proc # Unmount first!  
sudo rm -rf /srv/maven-chroot/
```

19.6. Quilt fürs Patchen einrichten

Mit diesem Skript wird für das Patchen unter anderem *dquilt* verwendet. Dies ist eine *debian*-spezifische Anpassung für *quilt*.

Um diese Anpassung zu erzeugen, bedarf es der Datei *.quiltrc-dpkg* mit folgendem Inhalt ³:

```
72 <DQuilt 72>≡  
d=. ; while [ ! -d $d/debian -a 'readlink -ev $d' != / ]; do d=$d/..; done  
if [ -d $d/debian ] && [ -z $QUILT_PATCHES ]; then  
    # falls in Debian-Paketbaum mit ungesetztem $QUILT_PATCHES  
    QUILT_PATCHES="debian/patches"  
    QUILT_PATCH_OPTS="--reject-format=unified"  
    QUILT_DIFF_ARGS="-p ab --no-timestamps --no-index --color=auto"
```

³<https://www.debian.org/doc/manuals/maint-guide/modify.html>

```

QUILT_REFRESH_ARGS="-p ab --no-timestamps --no-index"
QUILT_COLORS="diff_hdr=1;32:diff_add=1;34:diff_rem=1;\
31:diff_hunk=1;33:diff_ctx=35:diff_cctx=33"
if ! [ -d $d/debian/patches ]; then mkdir $d/debian/patches; fi
fi

```

Dazu gehört für den manuellen Betrieb auch der Eintrag in der Datei `~/.bashrc`. Dieser Eintrag sieht wie folgt aus:

```

# wird fuer die Patch-Verfolgung mit Quilt benoetigt
alias dqilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
complete -F -quilt-completion $_quilt_complete_opt dqilt

```

Zu beachten ist, dass Einstellungen in der Datei `~/.bashrc` bei der Ausführung dieses Programmskriptes nicht beachtet werden. Es müssen also alle notwendigen Einstellungen im Programmskript komplett abgebildet werden.

Die Verwendung von *quilt* bzw. *dqilt* wird in Kapitel *Nutzung von Quilt* (Kapitel 36.3, Seite 302) beschrieben.

20. Git einrichten

20.1. Branches

Das Git-Repository eines Debian-Paketes hat in der Regel mindestens folgende Zweige:

- *debian/sid*
- *upstream*
- *pristine-tar*

Der Branch *debian/sid* kann auch *master* oder *main* genannt werden.

Es kann einen zusätzlichen Branch für *experimental* geben. Außerdem können Branches für *backports* und *update-proposal* angelegt werden.

Solche weiteren Branches können auch vom Programmskript angelegt werden. (Kapitel 44.1, Seite 407)

20.2. Mergen

Wird von *debian/experimental* nach *debian/sid* gemergt, wird in der Regel ein Fast-Forward-Merge durchgeführt.

Dies ist dann der Fall, wenn - wie hier - zunächst in *debian/experimental* weitere Aktualisierungen zum Testen eingepflegt wurden, dann aber die Entwicklung in *debian/sid* fortgeführt werden soll.

Wurden jedoch in der Zwischenzeit auch im Branch *debian/sid* Änderungen hinzugefügt, kann nur ein Recursive-Merge durchgeführt werden.

Gegebenenfalls müssen einige Anpassungen, z.B. im Verzeichnis *debian/* einzeln ausgewählt und dem jeweiligen Zweig hinzugefügt werden (*git cherry-pick*).

Eine dazu bewährte Befehlszeile ist:

```
git cherry-pick --edit -x <commit>
```

Sollen mehrere Commits auf einmal dem Zweig zugeführt werden, gilt folgende Befehlszeile:

```
git cherry-pick --no-commit <commit> <commit> \dots
```

20.3. *gbp.conf*

Das Programmskript nutzt die Anwendungen aus dem Debian-Paket *git-buildpackage*.

git-buildpackage (*gbp*) kann und sollte konfiguriert werden.

Die Konfigurationsdatei *gbp.conf* wird zur Steuerung dieser Anwendung verwandt. Sie kann an verschiedenen Stellen im Dateisystem platziert werden.

20.3.1. Reihenfolge

Die Konfigurationsdateien für gbp werden in folgender Reihenfolge eingelesen:

1. /etc/git-buildpackage/gbp.conf, die systemweite Konfigurationsdatei
2. ~/.gbp.conf, die nutzerspezifische Konfigurationsdatei
3. debian/gbp.conf, Konfiguration für das Repositorium oder den Branch
4. git/gbp.conf, Konfiguration für das lokale Repositorium

Alle Konfigurationsdateien haben das gleiche Format. ¹

Durch Setzen der Umgebungsvariablen `GBP_CONF_FILES` kann diese Reihenfolge überschrieben werden. Der Inhalt dieser Variable kann mit `echo $GBP_CONF_FILES` ermittelt werden. ²

20.3.2. Abschnitte in der *gbp.conf*

Es gibt verschiedene Abschnitte in der *gbp.conf*. Diese Abschnitte sind alle optional.

Für jeden *gbp*-Befehl³ kann ein eigener Abschnitt erstellt werden. Zusätzlich gibt es einen Abschnitt, der für alle Befehle gilt

Einige wichtige Abschnitte werden im Folgenden aufgeführt.

[DEFAULT] In diesem Abschnitt angegebenen Optionen werden auf alle *gbp*-Befehle angewandt. ⁴

[import-orig] Die Optionen dieses Abschnittes überschreiben die entsprechenden Abschnitte unter *[DEFAULT]*. Sie werden auf den Befehl *gbp import-orig* angewandt.

[pq] Die Optionen dieses Abschnittes werden auf den Befehl *gbp pq* angewandt und überschreiben die des Abschnittes *[DEFAULT]*.

[dch] Die Optionen dieses Abschnittes werden auf den Befehl *gbp dch* angewandt und überschreiben die Optionen des Abschnittes *[DEFAULT]*.

[buildpackage] Die Optionen dieses Abschnittes werden auf den Befehl *gbp buildpackage* angewandt und überschreiben die Optionen des Abschnittes *[DEFAULT]*.

20.3.3. Syntax der Optionen

Die Optionen in den Abschnitten der *gbp.conf* werden aus den Befehlszeilenoptionen gebildet. Die möglichen Optionen zu den einzelnen *gbp*-Befehlen können der jeweiligen Manpage entnommen werden.

Diese werden ohne das einleitende doppelte Minuszeichen angegeben. Aus Befehlszeilenoption `--patch-num-format=%02d_` wird dann `patch-num-format=%02d_`.

Im Falle von *gbp buildpackage* ist zusätzlich auch *git*- wegzulassen⁵.

Der Eintrag `pbuilder-options=PBUILDER_OPTION` in der *gbp.conf* entspricht also `--git-pbuilder-options=PBUILDER_OPTION`.

¹[3], Abschnitt *Configuration Files* und die Manpage zu *gbp-conf*

²[3] Abschnitt *Configuration Files/Overriding Parsing Order*

³[3] Manpage zu *gbp-conf*

⁴[3] Manpage zu *gbp-conf*

⁵Manpage zu *gbp-conf*[3]

20.3.4. Beispiel

Im Homeverzeichnis des Nutzers kann eine Datei `~/.gbp.conf` beispielsweise wie folgt angelegt werden:

```
77 <gbp.conf 77>≡

[DEFAULT]
sign-tags = True
# keyid for signing the package
keyid = 0x<keyid>
pristine-tar = True
# generate gz compressed orig file
# compression = xz
# If you want to use normally sbuild
# builder = sbuild

[buildpackage]
postbuild = lintian $GBP_CHANGES_FILE
cleaner = /bin/true
# If you want to use normally pbuilder
# pbuilder = True
pbuilder-options = --source-only-changes --hookdir /home/mechtilde/.pbuilder

#[buildpackage]
# use a build area relative to the git repository
# export-dir=../build-area
# to use the same build area for all packages use an absolute path:
#export-dir=/home/debian-packages/build-area

[dch]
id-length = 7

# Options only affecting gbp pq
[pq]
#patch-numbers = False
# The format specifier for patch number prefixes
#patch-num-format = '%04d-'
patch-num-format = '%02d_'
# Whether to renumber patches when exporting patch queues
#renumber = False
renumber = True
# Whether to drop patch queue after export
#drop = False
```

In der Datei `/etc/git-buildpackage/gbp.conf` werden die Konfigurationsmöglichkeiten aufgeführt.

20.4. Git-Repositoryn auf eigener Infrastruktur

20.4.1. Lokales Git-Repository

Das lokale Git-Repository wird entweder vom Skript angelegt (Kapitel 32.4, Seite 144) oder durch Klonen erzeugt (Kapitel 32.5, Seite 160). Außerdem kann es mit `gbp import-dsc` erzeugt werden (Kapitel 32.6, Seite 168).

Ihm können weitere Branches hinzugefügt werden (Kapitel 44.1, Seite 407).

20.4.2. Eigener Git-Server

Das lokale Git-Repository kann auch auf einem eigenen Git-Server „gespiegelt“ werden.

Die Einrichtung des Servers erfolgt manuell. Die Verwendung von `cgit` oder `gitweb` erleichtern den Zugriff.

Im Programmskript können der Name oder die IP des eigenen Git-Servers eingegeben werden (Kapitel 44.2, Seite 408).

Der „Workflow“ ist dann wie folgt: Nach der Anlage der Konfigurationsdatei wird zunächst der Name oder die IP des eigenen Git-Servers eingegeben (Kapitel 44.2, Seite 408), bevor mit dem Bauen eines neuen Paketes begonnen wird (Kapitel 32, Seite 117).

Das „fertige“ Paket kann dann auf den eigenen Git-Server hochgeladen werden (Kapitel 42.2, Seite 386).

21. *Salsa*-Repositorien

Salsa ist der Name eines gemeinschaftlichen Entwicklungsserver für *Debian*, der auf der *GitLab*-Software basiert. *Salsa* soll die notwendigen Werkzeuge für die kollaborative Entwicklung für Paketbetreuer, Paketierungsteams und andere *debian*-bezogene Einzelpersonen und Gruppen bereitstellen.[36]

Salsa bietet alle Funktionen von *GitLab*. Der Dienst steht unter <https://salsa.debian.org> zur Verfügung. Es gibt eine Dokumentation der *debian*-spezifischen Eigenheiten [37], mit denen man sich zunächst vertraut machen sollte.

21.1. *Salsa*-Account anlegen

Das Anlegen eines Accounts auf *Salsa* ist sehr einfach. Man ruft die Seite https://salsa.debian.org/users/sign_up auf, welche selbsterklärend ist.

21.2. Anlage eines *Salsa*-Repositoriums

Ein *Git*-Repositorium auf *salsa.debian.org* wird nicht vom Programmskript eingerichtet.

Nach dem Login und der Authentifizierung auf <https://salsa.debian.org> kann dort im jeweiligen Team ein neues Repositorium angelegt werden.

Für die Anlage von neuen Projekten auf <https://salsa.debian.org> sind entsprechende Rechte erforderlich. Dies können die Rechte eines *Debian*-Developers sein. Für die team-betreuten Projekte können diese Rechte auch an weitere Personen von den Betreuenden vergeben werden.

Eine Beschreibung der Anlage eines Projektes im Java-Team erfolgt in Kapitel 21.3, Seite 80.

Nach dem Login auf der Seite <https://salsa.debian.org> und der Auswahl des passenden Projektes für das neue Paket, wird mit dem Klick auf den Button *Neu erstellen* und der Auswahl der Funktion *Neues Projekt/Repository* die entsprechende Seite aufgerufen. Dort ist meist *Ein leeres Projekt* auszuwählen. Dann gibt man den Projektnamen ein. Dies ist meist der Name des Quellcodepaketes. Als Sichtbarkeitsgrad (Visibility Level) wird *Öffentlich* ausgewählt. Dann folgt der Klick auf den Button *Projekt anlegen*.

Auf der folgenden Web-Seite gibt es noch einige Erläuterungen. Vieles davon wird zunächst lokal mit dem beschriebenen Programm angelegt.

In der linken Navigationsleiste werden nun im Bereich *Einstellungen* weitere Konfigurationen zum Projekt vorgenommen.

Hier ist es wichtig unter *CI/CD* den Pfad und den verwendeten Dateinamen anzugeben.

CI steht für Continuous Integration

CD steht für Continuous Delivery

Die hier verwendete Datei heißt *salsa-ci.yml* (Kapitel 35.4.9, Seite 277) im Verzeichnis *debian/*.

Das Build-Skript trägt das Salsa-Repositorium als „Remote-Repositorium“ ein und erinnert den Nutzer an die Anlage auf *salsa.debian.org* (Kapitel 32.4.3, Seite 158)

21.3. Salsa-Repositorium für das Java-Team

Salsa-Repositorien, die einem speziellen Projekt (wie beispielsweise dem Java-Team) zugeordnet sind, sollten möglichst einheitlich angelegt werden. Häufig wird hierzu ein Skript vom Projekt bereitgestellt, welches hierzu verwandt werden soll.

Dazu wechselt man in das gewünschte Team-Verzeichnis.



Abbildung 21.1.: Information zum Java-Team^a

^aQuelle: <https://salsa.debian.org/java-team/>

21.3.1. Quelle des Skripts

Unter

<https://salsa.debian.org/java-team/pkg-java-scripts/blob/master/setup-salsa-repository> kann das Skript des *Java-Teams* heruntergeladen werden. Es ist auch im Anhang beigefügt (Kapitel 50.1, Seite 443).

21.3.2. Abhängigkeiten

Dieses *Setup-Skript* nutzt *jq*. *jq* ist ein leichtgewichtiger und flexibler *JSON*-Prozessor für die Kommandozeile. Er hat nur minimale Laufzeitabhängigkeiten. Es gibt ein gleichnamiges *Debian*-Paket. Dieses Paket muss lokal installiert sein (Kapitel 19.1.2, Seite 60).

21.3.3. Zugangstoken beschaffen

Für die Anpassung wird noch ein projektspezifischer Token benötigt.

Nach dem Login auf *salsa.debian.org* wird im Dropdown-Menü des Nutzers *Einstellungen* gewählt. Damit wird die Seite *Benutzereinstellungen* des eingeloggten Nutzers aufgerufen.

In der linken Leiste befindet sich nun der Eintrag *Zugangs-Token*. Dort wird die Seite https://salsa.debian.org/profile/personal_access_tokens zur Erstellung eines solchen Token aufgerufen. Er wird für jedes Projekt neu generiert.

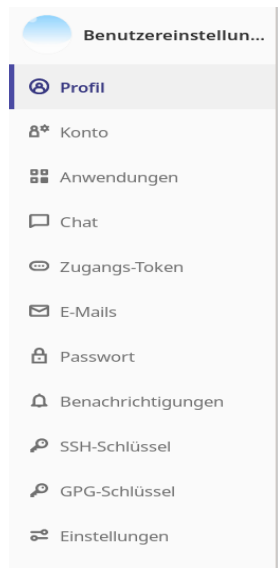


Abbildung 21.2.: Zugangstoken erstellen^a

^aQuelle:<https://salsa.debian.org>

Dort wird der Name des zu erstellenden Projektes eingegeben. Ferner wird ein Ablaufdatum für das Token eingegeben. Danach wird der Gültigkeitsbereich des Tokens festgelegt. Als *Scope* ist hier *api* auszuwählen. Mit dem Klick auf den Button *Create personal access token* erscheint oben auf der Seite der Zugangstoken.

21.3.4. Token eintragen

Das generierte Token ist als *SALSA_TOKEN* in das Skript einzutragen. Das Kommentarzeichen ist zu entfernen

Es erscheint sinnvoll, dieses Skript jeweils im Projektverzeichnis abzulegen.

21.3.5. Skript aufrufen

Das Skript wird nun mit dem Namen des neuen Projektes (Paketes) als Parameter aufgerufen:

```
./setup-salsa-repository.sh <packagename>
```

Danach werden folgende Meldungen ausgegeben (anhand des Beispiels BeanValidationApi):

```
./setup-salsa-repository.sh beanvalidation-api
Creating the beanvalidation-api repository\dots
Configuring the BTS tag pending hook\dots
```

13. Oktober 2024

```
Configuring the KGB hook\dots
Configuring email notification on push\dots
```

```
Done! The repository is located at
  https://salsa.debian.org/java-team/beanvalidation-api
```

Erscheint lediglich die erste Zeile, sind das verwendete Skript (s. Kapitel 50.1, Seite 443) und Token zu prüfen.

21.4. Aufgaben auf *salsa.debian.org*

21.4.1. Merge Request

Manchmal gibt es auch sogenannte Merge Request, in dem andere Patches bereitstellen.

Auf *salsa.debian.org* gibt es in der linken Navigationsleiste dann einen Eintrag „Merge-Requests“. Dort kann dieser dann bearbeitet werden.

Dies erfolgt durch Klicken auf die Commit-Message.

22. Paketieren jenseits vom Zweig *Unstable*

Es gibt verschiedene Gründe, warum von dem generellen Weg, neue Upstream-Versionen ausschließlich für *Unstable = sid* zu paketieren, abgewichen werden darf und wird. Auch das Programmskript ermöglicht dies (Kapitel 39, Seite 355).

Die Entwicklerreferenz [9] bezeichnet das Hochladen in die Distribution *Stable* und *Oldstable* als „Sonderfall“.¹

Ein wesentlicher Grund ist das Vorhandensein eines schwerwiegenden Fehlers oder eines Sicherheitsproblems. Schwerwiegende Fehler in einem Paket werden in der Regel durch einen entsprechend eingestuften Fehlerbericht gemeldet.

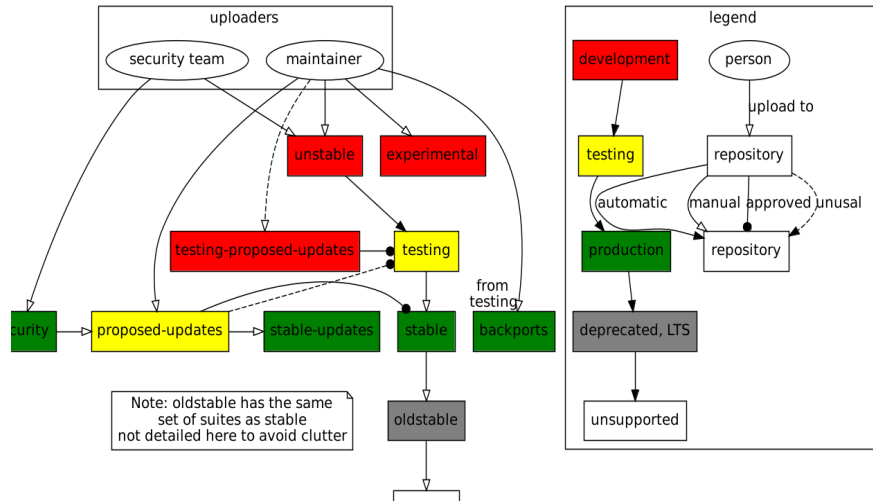
Eine weitere Abweichung von der generellen Regel gibt es bezüglich der Pakete der **Mozilla-Suite**, **Firefox** und **Thunderbird**. Diese Pakete werden zeitnah nach ihrem Upstream-Release für die *ESR = Extended Support Release* Version auch als *Security-Updates* (Kapitel 22.1, Seite 84) für das *Stable*-Release bereitgestellt.

Daraus können sich z.B. für die Erweiterungen, hier als Webextensionen (Kapitel 14, Seite 49) beschrieben, Inkompatibilitäten mit der dann aktuellen Version im *Stable*-Release ergeben. Dies ist ebenfalls ein Grund, eine Aktualisierung im *Stable*-Release vorzunehmen (s.a. Kapitel 22.2.3, Seite 86).

Daneben ist es bei Desktop-Anwendungen oft wünschenswert, aktuellere Versionen derselben in einer stabilen Betriebssystemumgebung zu nutzen. Dazu werden die gewünschten Pakete als sogenannte *Backports* (d.h. Rückportierungen) zur Verfügung gestellt. (s. Kapitel 22.3, Seite 86)

Es gibt auch gute Gründe, Pakete zunächst nach *Experimental* (auch *rc-buggy* genannt) hochzuladen. Dies gilt besonders für neue Pakete. In dem Zeitraum, in dem die bestehenden Versionen „eingefroren“ sind und nur Fehlerbehebungen für das nächste Release erlaubt sind, werden neue Versionen oft vorab nach *Experimental* hochgeladen. (s. Kapitel 22.5, Seite 87)

¹Kapitel 5.5.1 in der Developer-Reference[9]

Abbildung 22.1.: Arbeitsabläufe [38] ^a

^a©2016 Antoine Beaupré anarcat@debian.org, CC-BY-SA 4.0

22.1. Security-Updates

Wichtig ist die Rückportierung von Änderungen an Quellcode-Dateien wegen Sicherheitsproblemen.

Immer, wenn ein Sicherheitsproblem bekannt wird, soll der Maintainer mit dem Sicherheits-Team zusammenarbeiten, um für das *Stable*- bzw. *Oldstable*-Release eine korrigierte Version bereitzustellen. Die Fehlerbehebung sollte gezielt erfolgen. Patches sollten so klein wie möglich gehalten werden. Weitere Informationen befinden sich in den Kapiteln 3.1.2 und 5.8.5 der *Debian-Entwicklerreferenz*[9].

22.2. (Old-)Stable-Proposal

Sollen Pakete mit schwerwiegenden Fehlern, die nicht die IT-Sicherheit betreffen, aktualisiert werden, kann dies unter bestimmten Voraussetzungen in *Proposed-Updates* erfolgen. Gleiches gilt hinsichtlich der Erweiterungen für *Firefox* und *Thunderbird*.

Als Kriterien für das Hinzufügen von Paketen zu *stable-updates* sind folgende genannt worden. ²:

- Die Aktualisierung ist dringend und nicht sicherheitsrelevant. Die Sicherheitsaktualisierungen erfolgen im oben genannten Wege (Kapitel 22.1, Seite 84).
- Das fragliche Paket ist ein Datenpaket, und die Daten müssen zeitnah aktualisiert werden (z.B. tzdata).
- Korrekturen an Paketen, die durch externe Änderungen beeinträchtigt sind und von denen kein anderes oder nur wenige andere Pakete abhängig sind.
- Pakete, die aktuell sein müssen, um nützlich zu sein (z.B. clamav).

²<https://lists.debian.org/debian-devel-announce/2011/03/msg00010.html>

Wer glaubt, dass die angestrebte Aktualisierung diese Kriterien erfüllt, sollte sich vertrauensvoll an das Release Team über die Mailingliste `debian-release@lists.debian.org`, und erläutern, dass diese Aktualisierung auch über *stable-updates* erfolgen sollte.

Das für die Rückportierung auf diesem Wege vorgesehene Paket sollte bereits in *Unstable* oder besser noch in *Testing* veröffentlicht worden sein.

Während Backports in einem eigenen Repositorium gesammelt werden, werden *Proposed-Updates* in das *Stable-Proposed*-Repositorium eingefügt. Diese Pakete werden dazu von den Debian-Entwicklern nach *Stable* hochgeladen. Dies gilt für Oldstable-Proposed-Updates entsprechend. Die Veröffentlichung erfolgt im nächsten Point-Release.

22.2.1. Fehlerbericht

Eine notwendige Voraussetzung ist auch hier zunächst ein aussagekräftiger Fehlerbericht (Bug-Report) (s.a. Kapitel 23, Seite 89). Dieser wird an `submit@bugs.debian.org` gesandt. Vorzugsweise sollte dafür das Programm `Reportbug` genutzt werden.

Dieser Fehlerbericht benötigt folgende Parameter:

```
Package: release.debian.org
Severity: normal
Tags: <ReleaseName>
User: release.debian.org@packages.debian.org
Ustags: pu
X-Debbugs-Cc: <SourceName>@packages.debian.org, <Maintainer>@debian.org
Control: affects -1 + src:<SourceName>
```

[Reason]

This package is ... # Why should it be updated?

[Impact]

Otherwise ... # What happens instead?

[Risks]

Which packages are affected?

[Checklist]

- [X] *all* changes are documented in the `d/changelog`
- [X] I reviewed all changes and I approve them
- [X] attach `debdiff` against the package in `(old)stable`
- [X] the issue is verified as fixed in `unstable`

[Changes]

What are the changes?

[Other info]

Do you have further infos?

Dieser Fehlerbericht wird gegen das Pseudopaket *release.debian.org* erstellt. Der Schweregrad eines solchen Fehlerberichtes ist im Allgemeinen maximal *normal*. Im Betreff wird der Name des Paketes aufgeführt, das für *Proposed-Updates* gebaut wird.

Proposed-Updates werden, wie dargelegt, nur unter besonderen Umständen zugelassen. Das *Release-Team* entscheidet, ob die Veröffentlichung erfolgt.

Daher muss der Fehlerbericht eine Begründung enthalten, warum diese Version in *stable* aktualisiert werden soll. Sofern es bereits eine dazugehörige Fehlernummer gibt, ist diese anzugeben. Dieser Fehlerbericht **muss** einen Schweregrad von *important* oder höher aufweisen.

Ein entsprechender Fehlerbericht muss gegebenenfalls auch für *Old-Stable* erstellt werden.

Weist dieser Fehlerbericht den notwendigen Schweregrad nicht auf, ist er zu berichtigen. Wie dies erfolgt, wird in Kapitel 24 (Seite 93)

22.2.2. Anforderungen an einen Patch

Ein Fehler sollte mit einem möglichst kleinen und passgenauen Patch behoben werden können. Damit soll verhindert werden, dass neue Fehler entstehen. Es dürfen auch keine neuen Abhängigkeiten hinzukommen. Auch muss berücksichtigt werden, welche anderen Pakete von diesem Paket abhängen.³

Diese passgenauen Patches werden mit *debdiff <BisherigesPaket>.dsc <NeuesPaket>.dsc > <Dateiname>.txt* dokumentiert. Es wird die Differenz angegeben, die zwischen der aktuellen Paketversion in *Stable/Oldstable* (bisheriges Paket) und der Version besteht, die hochgeladen werden soll (neues Paket). Wie diese Dokumentation mit dem Programmskript erstellt werden kann, wird in Kapitel 40.6.1 (Seite 365) beschrieben.

22.2.3. Abhängigkeiten zu Mozilla-Paketen

Eine hinreichende Begründung für *Webeextensions* kann auch die oben (s. Kapitel 22, Seite 83) erwähnte Ausnahmeregelung für die Pakete der *Mozilla-Suite* sein.

Für diese Erweiterungen können sich nämlich Inkompatibilitäten mit der dann aktuellen Version von *Firefox* oder *Thunderbird* im *Stable*-Release ergeben. Dann sind die bisherigen Versionen unbrauchbar, was vor allem im produktiven Betrieb ein beachtliches Problem darstellt. (Kapitel 39, Seite 355)

22.3. Stable-Backports

Um Pakete für Backports bauen zu können, sind einige Vorbereitungen zu treffen.

Oft lässt sich ein solches Paket nicht ohne Weiteres in einer *Stable*-Umgebung bauen.

³Kapitel 5.5.1 in der Developer-Reference[9]

Dann werden weitere Pakete aus Backports benötigt. Pakete aus Backports werden aber nicht automatisch auch in einer solchen *Chroot* installiert.

Daher ist es notwendig, in */etc/apt/preferences* eine solche Installation zu ermöglichen. Dazu werden im *pbuilder Hooks* verwendet (Kapitel 19.3.3, Seite 67).
https://wiki.debianforum.de/Pbuilder_-_personal_package_builder

22.4. Backports-Repositoryum

Regelmäßig werden die Pakete für den Unstable-Zweig gebaut. Von dort gelangen sie dann - sofern keine Fehler gefunden werden - in den Testing-Zweig. Beim Release wird der Testing-Zweig zunächst eingefroren und dann zum neuen Stable-Zweig. Es wird dann ein neuer Testing-Zweig eröffnet.

Werden Programmversionen aus *testing* oder (ausnahmsweise ⁴) auch aus *unstable* in ein früheres Release übertragen, nennt man dies *backporting*.

Durch die an der Praxis orientierten Releasezyklen der stabilen Debian-Version ist es manchmal wünschenswert, Softwarepakete oder neuere Versionen aus Testing auch unter Stable verfügbar zu haben.

Hierzu dient das Backports-Repositoryum.

Versionierung (z.B. +deb9u1)

Backports gibt es für das Stable- und Oldstable-Release.

22.5. Experimental

22.6. Backporten fremder Pakete

pristine-tar NMU Abhängigkeiten aus Backports

Verweis auf (Kapitel ??, Seite ??) ⁵ für die Veröffentlichung nach *Proposed-Updates* für *Stable* bzw. *Old-Stable*.

22.7. Versionierung

Die Ursache bzw. Herkunft der Aktualisierung hat auch Einfluss auf die Versionierung. Dabei ist sicherzustellen, dass *dpkg* neuere Versionen korrekt als Upgrades interpretieren kann. Ein Blick auf die nächste zu erwartende Versionsnummer kann dabei helfen.

Beim Bauen für *experimental* wird in der Datei *debian/changelog* folgender Versions-
 eintrag empfohlen:

<Versionsnummer>-<Revisionsnummer>~exp<Laufende Nummer>

Die Revisionsnummer ist in der Regel: 1

Beim Bauen für *Proposed-Updates* sind zwei Fälle zu unterscheiden.

Fall A Eine im *Stable*-Release vorhandene Version soll unter Beibehaltung dieser Versionsnummer korrigiert werden.

⁴in der Regel nur Sicherheits-Updates

⁵[9] Abschnitt *upload-stable*

13. Oktober 2024

Fall B Ausnahmsweise soll eine neue Version in das Release aufgenommen werden.

Für den Versionseintrag in *debian/changelog* ist zwingend folgende Nomenklatur zu verwenden:

Fall A <Ursprüngliche Versions- und Revisionsnummer> +deb<Debian-Release-Nummer>u<Revisionsnummer des Updates>

Fall B <Versions- und Revisionsnummer aus Unstable/Testing>~deb<Debian-Release-Nummer>u<Revisionsnummer des Updates>

Die Verwendung der ~bewirkt, dass die Version in Stable kleiner als die in Testing (für das nächste Release) ist. Dies sichert den Aktualisierungspfad.

23. Zum Start eine E-Mail

Bevor das Paketieren mit dem Ziel der Veröffentlichung im Debian-Repository beginnen kann, bedarf es einer E-Mail. Diese löst einen „Bugreport“ aus, dessen Nummer in der Datei *debian/changelog* zu vermerken ist. Durch die Veröffentlichung des Paketes soll nämlich dann dieser Bugreport geschlossen werden.

Weitere Informationen dazu gibt es unter <https://www.debian.org/devel/wnpp/>

23.1. ITP - Intent To Package

ITP bedeutet „Intent to Package“. Dies bedeutet, dass an diesem Paket gearbeitet wird.

Dies ist die Bezeichnung eines Bug-Reportes, der gegen das Paket *WNPP* erstellt wird, was *Work-Needing and Prospective Package* bedeutet. Dies kann mit *Arbeitsbedürftige und voraussichtliche Pakete* übersetzt werden.

Das Paketieren von neuer Software sollte rechtzeitig angekündigt werden. Vielleicht gibt es ja noch Hinweise, was bei dem geplanten Paket zu beachten ist. Auch wird so verhindert, dass verschiedene Gruppen versuchen, dieselbe Software für Debian zu paketieren.

Eine Möglichkeit ist, dies mit dem auf jedem Debian-System installierten Tool *reportbug* durchzuführen. Es gibt aber auch die Möglichkeit einfach eine E-Mail-Vorlage zu nutzen. Die folgende Vorlage basiert auf dem *itp_template* mit dem *reportbug* diese E-Mail erstellt.

```
To: submit@bugs.debian.org
Subject: ITP: <Source Name> - <Short Description>
Package: wnpp
Severity: wishlist
```

```
* Package name : <Package Name>
  Version : x.y.z
  Upstream Author : Name <somebody@example.org>
* URL : http://www.example.org/
* License : (GPL, LGPL, BSD, MIT/X, etc.)
  Programming Lang: (C, C++, C#, Perl, Python, etc.)
  Description : <Short Description>
```

(Include the long description here.)

<And answer following questions:>

* Why is this package useful/relevant?

13. Oktober 2024

- * Is it a dependency for another package?
- * Do you use it yourself?
- * If there are other packages providing similar functionality, how does it compare?
- * How do you plan to maintain it? Do you plan to maintain it inside a packaging team?
(check list at <https://wiki.debian.org/Teams>)
- * Are you looking for co-maintainers? Do you need a sponsor?

Der Text in der ersten Zeile hinter dem *To:* kommt in die Adresszeile. Der Text in der zweiten Zeile hinter dem *Subject:* kommt in die Betreffzeile, wobei die Platzhalter durch den Namen des Quellcodes und einer kurzen Beschreibung ersetzt werden.

Wird das Paket hochgeladen, muss diese Nummer aus dem Bug-Tracking-System im *debian/changelog* vermerkt werden (Closes:#XXXXXX)

23.2. RFP - Request For Package

RFP bedeutet *Request for Package*. Dies bedeutet, dass ein solches Paket in Debian erwünscht ist.

Auch hierzu habe ich einmal eine E-Mail-Vorlage erstellt.

```
To: submit@bugs.debian.org
Subject: RFP: <Sourcecode Name> - <Short Description>
Package: wnpp
Severity: wishlist
```

```
* Package name : <Package Name>
  Version : x.y.z
  Upstream Author : Name <somebody@example.org>
* URL : http://www.example.org/
* License : (GPL, LGPL, BSD, MIT/X, etc.)
  Programming Lang: (C, C++, C#, Perl, Python, etc.)
  Description : <Short Description>
```

(Include the long description here.)

<And answer following questions:>

- * Why is this package useful/relevant?
Is it a dependency for another package?
- * Do you use it yourself?
- * If there are other packages providing similar functionality, how does it compare?
- * How do you plan to maintain it? Do you plan to maintain it inside a packaging team?

(check list at <https://wiki.debian.org/Teams>)

* Are you looking for co-maintainers? Do you need a sponsor?

Auch hier gilt: Der Text in der ersten Zeile hinter dem *To:* kommt in die Adresszeile. Der Text in der zweiten Zeile hinter dem *Subject:* kommt in die Betreffzeile, wobei die Platzhalter durch den Namen des Quellcodes und einer kurzen Beschreibung ersetzt werden.

23.3. ITA - Intent To Adoption

Dies wird verwendet, wenn ein Paket, dass mit „O“ oder „RFA“ gekennzeichnet ist, übernommen werden soll.

Dazu muss der vorherige Fehlerbericht umbenannt und „O“ bzw. „RFA“ durch „ITA“ ersetzt werden.

Dabei tragen Sie sich als Besitzer ein.

Soll ein Fehlerbericht umbenannt oder der Besitzer geändert werden, muss dies per E-Mail an control@bugs.debian.org oder direkt an den Fehlerbericht über die Nummer (xxxxxxx@bugs.debian.org) erfolgen¹.

Dabei ist ein strukturierter *Pseudo-Header* zu nutzen.²

23.4. RFA - Request for Adoption

23.5. RFH - Request For Help

23.6. O - Orphaned

23.7. RFS - Request For Sponsor

Wie auf <https://mentors.debian.net/sponsor/rfs-howto> beschrieben, wurde die Vorlage angepasst.

```
To: submit@bugs.debian.org
Subject: RFS: <Package Name> - <Short Description>
Package: sponsorship-request
Severity: normal
      [important for RC bugs, wishlist for new packages]
```

Dear mentors,

I am looking for a sponsor for my package "<Source Name>":

```
* Package name : <Source Name>
  Version : x.y.z
```

¹<https://www.debian.org/devel/wnpp/>

²<https://www.debian.org/Bugs/Reporting#control>

13. Oktober 2024

```
Upstream Author : Name <somebody@example.org>
* URL : http://www.example.org/
* License : (GPL, LGPL, BSD, MIT/X, etc.)
  Programming Lang: (C, C++, C#, Perl, Python, etc.)
  Description : <Short Description>
```

It builds those binary packages:

<Name of the Binaries>

To access further information about this package, please visit the following URL:

<https://mentors.debian.net/package/<package name>>

Alternatively, one can download the package with dget using this command:

```
dget -x https://mentors.debian.net/debian/pool/main/<p> \
/<package name>/<package name>_x.y.z.dsc
```

Changes since the last upload:

[your most recent changelog entry]

Regards,

24. Änderungen am Fehlerbericht

Im Laufe eines solchen Prozesses kann es vorkommen, dass Änderungen am Fehlerbericht erfolgen müssen. Eine solche Änderung kann eine Änderung des Maintainers, des Titels oder auch Anderes sein.

Dazu werden die Befehle des *Kontroll-E-Mail-Server* genutzt. Unter <https://www.debian.org/Bugs/server-control> findet sich die Beschreibung dazu.

Dies kann mit einer strukturierten E-Mail erfolgen. Diese wird an die Adresse *control@bugs.debian.org* gesandt.

Befehle, die eigentlich an *control@bugs.debian.org* zu senden sind, funktionieren auch, wenn sie an *submit@bugs.debian.org* oder an *<bug number>@bugs.debian.org* gesandt werden ¹

24.1. Anpassung des Schweregrades

Im Betreff kann hier *Change severity* angegeben werden.

```
severity <bug number> important  
thank you
```

Für die automatische Auswertung der relevanten Informationen in der E-Mail muss sie mit *thanks*, *thankyou*, *thank you* oder einer anderen Endemarkierung wie *quit* oder *stop* abgeschlossen werden.

24.2. Anpassung des Titels

Für die Änderung des Titels wird dann zusätzlich eine Zeile wie folgt

```
retitle <bug number> -1 <neuer Titel>
```

eingefügt.

24.3. Änderung des Maintainers

Für die Änderung des Maintainers wird die folgende Zeile hinzugefügt:

```
Control: owner -1 <Neuer Maintainer oder wnpp@debian.org>
```

¹<https://www.debian.org/Bugs/Reporting#control>

24.4. Öffnen eines geschlossenen Fehlerberichtes

Gegebenenfalls muss ein fälschlich geschlossener Bericht wieder geöffnet werden. Dies erfolgt mit

```
reopen Fehlernummer [ <Urheber-Adresse> | = | ! ]
```

24.5. Schließen eines Fehlerberichtes

```
close Fehlernummer [ Urheber-Adresse | = | ! ]
```

24.6. *usertags* hinzufügen

Im Laufe eines Maintainer-Lebens kommt es immer wieder vor, dass Fehlermeldungen mit tags versehen werden müssen oder sollten.

Z. B. ist es hilfreich für sogenannten *Bug Squashing Parties*, die geplanten und durchgeführten Fehlerbehebungen auch zu kennzeichnen.

Dafür wird eine E-Mail an den Fehlerbericht geschrieben. Adresse lautet dann <Bugnummer>@debian.org. Gleichzeitig soll diese E-Mail auch *CC* an *control@bugs.debian.org* gehen.

Am Anfang einer solchen E-Mail steht dann:

```
user debian-release@lists.debian.org
usertags -1 + <Titel der BSP>
thank you
```

25. Reportbug einrichten

Das Comand-Line-Interface ist bereits Bestandteil der Basisinstallation. Zusätzlich kann mit dem Paket *reportbug-gtk* noch ein graphisches Nutzerinterface installiert werden.

26. Schwierigkeiten überwinden

26.1. Ein Paket loseisen

Eine Schwierigkeit ergibt sich daraus, dass es vor einem geplanten Release eine Zeitspanne gibt, in der die Pakete nicht mehr automatisch von *unstable* nach *testing* migrieren.

Im vollständigen „Freeze“ benötigen alle Pakete, die noch von *unstable* nach *testing* migrieren sollen, eine Entsperrung durch das Release-Team.[39]. Diese muss mit einem *Unblock Bugreport* beantragt werden.

26.1.1. Beantragung einer Entsperrung

Dazu wird zunächst eine Datei mit *debdiff* erstellt (s.a. Kapitel 22.2, Seite 84).

Damit wird die Differenz zwischen der Version in *testing* (alte Version) und *Unstable* (neue Version) mit der jeweiligen *dsc* erstellt.

Diese Differenz-Datei ist darauf zu prüfen, dass sie keine unwichtigen Änderungen für die gewünschte Fehlerbehebung hat.

Anschließend wird mit dem Werkzeug *reportbug* ein Fehlerbericht gegen das Paket *release.debian.org* erstellt und die Differenz-Datei angehängt. Dieser Fehlerbericht enthält eine ausführliche Begründung für die Änderungen und Verweise auf Fehlernummern. Ebenso enthält sie eine prägnante Beschreibung des Problems, das behoben wurde.

Dabei sollte auf folgende Fragen eingegangen werden.

```
Package: release.debian.org
User: release.debian.org@packages.debian.org
Usertags: unblock
Severity: normal
```

```
Please unblock package <source name>
```

```
(Please provide enough (but not too much) information to help
the release team to judge the request efficiently. E.g. by
filling in the sections below.)
```

```
[ Reason ]
(Explain what the reason for the unblock request is.)
```

```
[ Impact ]
(What is the impact for the user if the unblock isn't granted?)
```

```
[ Tests ]
```

(What automated or manual tests cover the affected code?)

[Risks]

(Discussion of the risks involved. E.g. code is trivial or complex, key package vs leaf package, alternatives available.)

[Checklist]

- [] all changes are documented in the d/changelog
- [] I reviewed all changes and I approve them
- [] attach debdiff against the package in testing

[Other info]

(Anything else the release team should know.)

26.2. Releasekritische Fehler beheben

Vor einer neue Veröffentlichung ist es oft notwendig die Maintainer dabei zu unterstützen, Fehler zu beheben, die einer Veröffentlichung des Paketes entgegenstehen.

Dies geschieht häufig auf dafür organisierten Veranstaltungen ¹.

Unter <https://www.debian.org/doc/manuals/developers-reference/pkggs.html#non-maintainer-uploads-nmus> ist das gewünschte Vorgehen beschrieben.

Wesentlich ist dabei, dass in der Datei *debian/changelog* (Kapitel 37.1, Seite 321) ein entsprechender Eintrag in der zweiten Zeile erfolgt.

```
Non-maintainer upload
```

26.3. Paket aus Repositorien entfernen

In der Developer-Reference ² wird auch beschrieben, wie ein Paket entfernt werden kann.

Dazu muss zunächst festgestellt werden, dass kein weiteres Paket dieses als Abhängigkeit benötigt.

26.3.1. Feststellung der Rückwärtsabhängigkeit

Zu prüfen ist, ob andere Pakete das zu entfernende Paket benötigen.

Überflüssig sind vor allem Bibliotheken, die von keiner Anwendung (mehr) benötigt werden.

Die Prüfung kann mit

```
apt-cache rdepends <Paketname>
```

¹<https://wiki.debian.org/BSP>

²[9], Abschnitt 5.9-Pakete entfernen

26.3.2. Fehlerbericht

Für die Durchführung ist nun ein Fehlerbericht zu erstellen. Dieser wird gegen das Pseudopaket *ftp.debian.org* erstellt.³

³s. a.[9], Abschnitt 5.9-Pakete entfernen

27. Autopkgtest

28. Lintian-Meldungen

Das Ergebnis des Bauens eines Debian-Paketes wird mit *lintian* überprüft (Kapitel 40.3, Seite 360).

Es gibt ein User's Manual zu Lintian, das auch im Paket *lintian* als Datei *lintian.rst* (in englischer Sprache) vorliegt[40].

Hierbei wird der Maintainer oft mit Fehlermeldungen konfrontiert.

Ein vollständiges Verzeichnis der möglichen Meldungen befindet sich unter <https://udd.debian.org/lintian-tag.cgi>.

Eine Liste aller Fehlermeldungen erhält man auch im Terminal:

```
lintian-explain-tags --list-tags | less
```

Die Erläuterung zu einer einzelnen Fehlermeldung erhält man mit

```
lintian-explain-tags <Lintian message>
```

Nachstehend einige Beispiele:

bad-jar-name Der Name entspricht nicht den Richtlinien der Java-Policy.¹

codeless-jar Die *.jar Datei enthält keinen kompilierten Java-Code.

empty-binary-package Das gebaute Paket ist leer.

javali-but-no-public-jars Im Verzeichnis */usr/share/java/* gibt es kein *.jar-Archiv. Dann fehlt in der Regel der Eintrag *-java-lib* in der entsprechenden Datei *debian/<paketname.poms*.

new-package-should-close-its-bug In der Datei *debian/changelog* ist der ITP-Bug zu schließen (Closes: #nnnnnn)

rules-requires-root-missing In die Datei *debian/control* wird im ersten Abschnitt der Eintrag *Rules-Requires-Root: no* benötigt.

wildcard-matches-nothing-in-dep5-copyright

backports-changes-missing

out-of-date-standard

testsuite-autopkgtest-missing

Ein Werkzeug zur automatischen Fehlerbehebung ist *lintian-brush*. Dieses Programm befindet sich im gleichnamigen Paket. Allerdings enthält es Skripte für nur etwas mehr als 10 % der möglichen Meldungen.

Mit *lintian-brush --list-tags* kann ermittelt werden, welche Korrekturskripte zur Verfügung stehen.

¹Kapitel 2.4 der Java-Policy[28]

29. Reproduzierbare Builds

29.1. reprotest

30. piuparts

Teil III.

Wie ein Shell-Skript hilft, ein Debian-Paket zu bauen

31. Erste Schritte im Programmskript

Nun geht es mit dem Programmskript los. Dieses hilft beim Bauen eines **Debian**-Paketes und unterstützt das Hochladen desselben. Dabei gibt der Programmablauf eine zweckmäßige Reihenfolge der notwendigen Schritte vor. Das Programmskript baut also keine **Debian**-Pakete, sondern unterstützt als Assistent den Maintainer. Hierauf wird im Eingangsdialog auch ausdrücklich hingewiesen. (Kapitel 31.2, Seite 112)

Voraussetzung ist, dass alle benötigten Pakete installiert und das System entsprechend eingerichtet ist (Kapitel 19, Seite 59).

Das Programmskript ist modular aufgebaut, sodass man an vielen Stellen „aussteigen“ und später wieder „einsteigen“ kann. Man muss also den Bauprozess nicht immer bis zum Abschluss „in einem Zug“ durchführen.

Der für den Nutzer sichtbare Programmablauf beginnt in jedem Fall mit einem Startdialog (Kapitel 31.2, Seite 112).

Das Bauen eines neuen **Debian**-Paketes erfordert zunächst die Anlage eines neuen Projektes (Kapitel 32, Seite 117).

Alle nachfolgende Punkte werden zumindest im Wesentlichen vom Programmskript erledigt.

Konfigurationsdatei Das Programm benötigt eine Konfigurationsdatei (Kapitel 32.1, Seite 117), die zunächst vom Programmskript angelegt wird. Diese kann jederzeit geändert werden.

Systemeinrichtung Um ein **Debian**-Paket zu bauen, müssen unter anderem folgende vorbereitende Aufgaben erfüllt werden:

Bereitstellung der benötigten Verzeichnisse Das Erstellen der Verzeichnisse wird in Kapitel 32.2.1, Seite 138 beschrieben.

Einrichten eines Git-Repositories Die Einrichtung des lokalen Git-Repositories (Kapitel 32.4, Seite 144) muss vor der Ausführung von *gbp import-orig* (Kapitel 34.4.11, Seite 243) erfolgen.

Bereitstellung des Quellcodes Dies wird in Kapitel 34.3, Seite 202 beschrieben.

Bereitstellung der benötigten Dateien im Verzeichnis *debian/* Das Erzeugen der Dateien im Verzeichnis *debian/* erfolgt im Rahmen des Bauens der **Debian**-Revision (Kapitel 35, Seite 253).

Bauen Dies wird in Kapitel 37, Seite 321 beschrieben

Testen – soweit wie möglich Dies wird in Kapitel 40, Seite 357 beschrieben.

Hochladen Dies wird in Kapitel 43, Seite 387 beschrieben.

Bis zum Hochladen ist es ein langer Weg. Doch auch der längste Weg beginnt bekanntlich mit dem ersten Schritt. Dabei ist noch zu beachten, dass das Programmskript tastaturbetont (mit der TAB-Taste) bedient wird. Eine Steuerung mit der Maus kann zu unerwarteten Effekten führen.

31.1. Der Anfang steht am Schluss

Das Programmskript enthält viele Funktionen.

Das Hauptprogramm ruft lediglich die Funktion *BuildApp* auf. Es steht am Ende des Programmskriptes.

```
112a  <MainProgram 112a>≡ (176b)
      #####
      # Here it starts
      BuildApp

      #####
      # This is the end, my friend
```

31.2. Und das sieht der Nutzer als Erstes

Die Funktion *BuildApp* steuert den Programmablauf im Wesentlichen durch den Aufruf weiterer Funktionen.

Sie stellt dem Nutzer als Erstes das Programm vor.

```
112b  <BuildApp 112b>≡ (115)
      function BuildApp {
          # Called by main program

          #####

          # Intro

          #####

          intro="Assistent to build simple Debian packages\nusing git-buildpackage\n
Authors: Mechtilde Stehmann\n
          Michael Stehmann\n
Version: 0.9.0\n
License: GPL v3+\n
This program does not build Debian packages itself.
It is only an assistant for the package maintainer."

          whiptail --title "Introduction" --msgbox "$intro" 20 60
```

```
<BuildApp3 113>
```

Dazu erscheint der folgende Begrüßungsdialog.

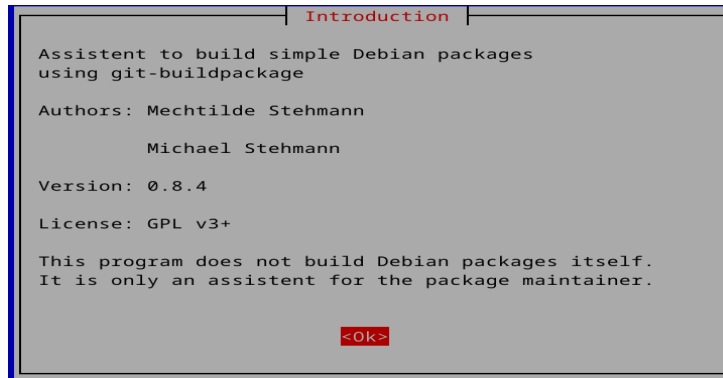


Abbildung 31.1.: Startbildschirm

Danach fragt diese Funktion den Namen des entsprechenden Projektes mit der Funktion *AskOrigName* ab (Kapitel 31.3, Seite 114).

```

113 <BuildApp3 113>≡ (112b)
    # Definitions of Project
    AskOrigName
    CreateDirsAndLogFile

    # Flag for additional gbp buildpackage options
    OptFlag=0

    #####

    # End of intro

<BuildApp7 140a>

```

Schließlich prüft diese Funktion *BuildApp* das Vorhandensein der notwendigen lokalen Infrastruktur (Konfigurationsdatei (Kapitel 32.1, Seite 117), Verzeichnisse (Kapitel 32.2.1, Seite 138), Git-Repositorium (Kapitel 33.4, Seite 182)) und sorgt gegebenenfalls für deren Anlage.

31.3. Projektname abfragen

Es wird nun der Projektname eines existierenden Projektes abgefragt oder für ein neues Projekt festgelegt. Das Einfügen des Projektnamens kann auch per *Copy & Paste* erfolgen. Diese Möglichkeit ist eine Besonderheit von *whiptail*.

```
114a  <AskOrigName 114a>≡ (191)
      function AskOrigName {
          # Called by BuildApp ConfigFileLEC and itself

          # Name of the project (without this name the app cannot work)
          OrigName=$(whiptail --title "This name is required!" \
              --inputbox "Name of the project:" \
              --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
      }
      <AskOrigName1 114b>
```

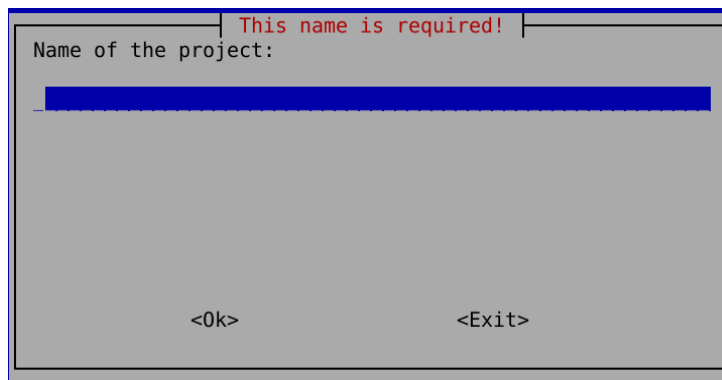


Abbildung 31.2.: Angabe des Projektnamens.

Eine *whiptail --inputbox* erfordert Umleitungen der Ausgabe unter Verwendung von Dateibezeichnern (*3>&2 2>&1 1>&3*).

```
114b  <AskOrigName1 114b>≡ (114a)
      if [ $? -ne 0 ]
      then
          whiptail --title "Bye" --msgbox "Bye" 15 60
          exit
      fi

      <AskOrigName2 115>
```

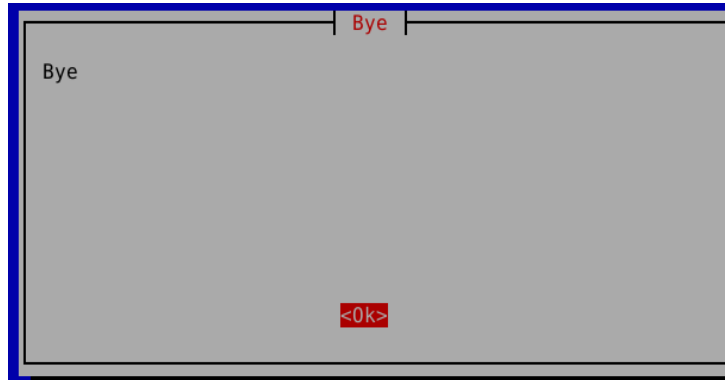



Abbildung 31.3.: Bye

Nur die ersten drei Dateibezeichner (beginnend mit 0) haben eine standardisierte Bedeutung:

- 0 - stdin - Standard-Eingabe
- 1 - stdout - Standard-Ausgabe
- 2 - stderr - Standard-Fehleranzeige

Wenn das Projekt bereits existiert, geht es mit der Anzeige der Konfigurationsdatei (Kapitel 33.1, Seite 177), der Auswahl eines Git-Zweiges (Kapitel 33.4, Seite 182) und dann der Aufgabenauswahl (Kapitel 33.5, Seite 191) weiter.

Wird **kein** Projektname eingegeben, erfolgt ein Hinweis. Die Eingabe eines Projektname ist zwingend erforderlich. Mit *Exit* wird das Programm abgebrochen.

```

115 <AskOrigName2 115>≡ (114b)
    if [ -z "${OrigName}" ]
    then
        whiptail --title "No project name" \
        --msgbox "You have to identify a project name\n \
        even it is a new project!" 15 60
        AskOrigName
    fi
    ConfigFileLEC
}

```

<BuildApp 112b>

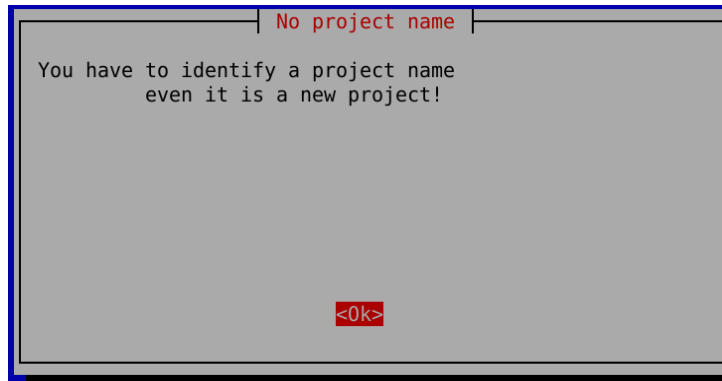


Abbildung 31.4.: Kein Projektname angegeben.

Nach der Bestätigung von *OK* wird erneut der Eingabedialog angezeigt.

31.4. Weiterer Ablauf

Der weitere Ablauf hängt nun davon ab, ob schon eine Konfigurationsdatei für das Projekt, wie in Kapitel 19.2.2.1, Seite 61 beschrieben, existiert.

In diesem Fall wird diese „geladen“ (Kapitel 33.1, Seite 177).

Andernfalls wird die Konfigurationsdatei neu angelegt.

32. Anlegen eines neuen Projektes

Zur Anlage eines neuen Projektes werden zunächst die Konfigurationsdatei erstellt und die notwendige Infrastruktur angelegt.

32.1. Konfigurationsdatei erstellen

Die Konfigurationsdatei wird im Homeverzeichnis des Nutzers im Verzeichnis *.debian_project/* als Datei *<Projektname>* gespeichert.

```
117a <ConfigFileLEC 117a>≡ (180)
function ConfigFileLEC {
    # Called by AskOrigName CreateNewBranch TaskSelect OwnServer

    ## Load, edit or create config file - using AskConfig

    # Path to config files directory
    ConfigPath=~/.debian_project/
    changeflag=0
```

<ConfigFileLEC1 177>

Die Funktion *ConfigFileLEC* prüft zunächst, ob eine Konfigurationsdatei mit dem Projektnamen vorhanden ist. Ist das Ergebnis der Prüfung negativ, erfolgt die Meldung, dass keine Konfigurationsdatei mit diesem Namen gefunden werden konnte.

```
117b <ConfigFileLEC4 117b>≡ (179)
else
    if whiptail --title "Config file not found" \
    --yesno "There is no config file for ${OrigName}\n \
    which you can edit.\n \
    Do you want to create a new project?" \
    --yes-button "Yes" --no-button "No" 15 60
    then
        changeflag=1
<ConfigFileLEC5 118>
```

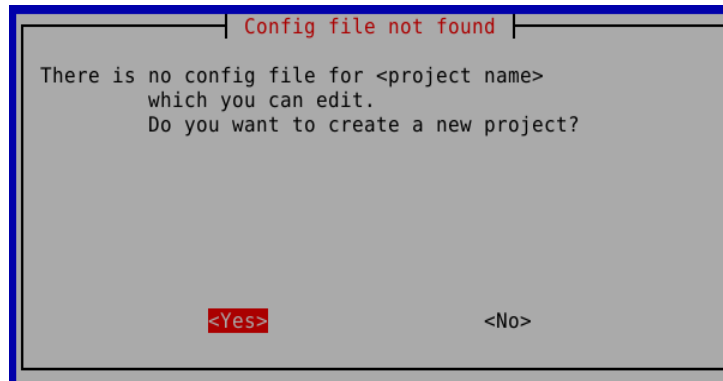


Abbildung 32.1.: Keine Konfigurationsdatei gefunden.

Ein *whiptail -yesno* gibt 0 bzw true zurück, wenn die Frage bejaht wird, und 1 bzw false, wenn sie verneint wird.

Wird diese Frage bejaht, wird vorsorglich auch das Verzeichnis *.debian_project* als *ConfigPath*, sofern noch nicht vorhanden, angelegt. Dann wird die Funktion *AskConfig* aufgerufen, mit der die Konfigurationsdatei erstellt wird.

```

118 <ConfigFileLEC5 118>≡ (117b)
      mkdir --parents ${ConfigPath}
      AskConfig
    else
      AskOrigName
    fi
  fi
  set -e
}

<CreateDirsAndLogfile 138>

```

Die Abfrage, ob ein neues Projekt erstellt werden soll, ermöglicht dem Nutzer, Tippfehler bei der Eingabe des Projektnamens zu korrigieren, wenn er sie verneint.

In den folgenden Abschnitten werden die in der Konfigurationsdatei enthaltenen Variablen besprochen. Zunächst folgen die Variablen, die für alle Pakete erforderlich sind. Danach folgen die Variablen, die nur für jeweils eine Gruppe von Paketen benötigt werden (für Java-Pakete s. Kapitel 32.1.2.2, Seite 131, für *webext*-Pakete s. Kapitel 32.1.2.3, Seite 133).

32.1.1. Abfrage allgemeiner Variablen für die Konfigurationsdatei

Die Funktion *AskConfig* ordnet in ihrem ersten Teil den Variablen, die in der Konfigurationsdatei enthalten sein sollen, Werte zu. Dies geschieht durch die Abfrage der einzelnen Variablen. Das Speichern der Konfigurationsdatei wird dann in Kapitel 32.1.3, Seite 136 beschrieben.

Es wird geprüft, ob der jeweiligen Variablen ein Wert zugewiesen wurde.

Dies geschieht als Erstes für den Namen des Quellpaketes, welcher als Wert der Variablen *SourceName* zugewiesen wird.

Der Name des Quellpaketes ist der Name, den das Upstream-Projekt seiner Software gegeben hat.

```

119 <AskConfig 119>≡ (131a)
    function AskConfig {
        # Called by ConfigFileLEC

        if [ -z "${SourceName}" ]
        then
            SourceName=$(whiptail --title "Source Package Name" \
                --inputbox "Name of the source package:" \
                --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

            if [ $? -ne 0 ] # Cancel-Button was pressed
            then
                exit
            else
                changeflag=1
            fi
        fi

        <AskConfig1 120>

```

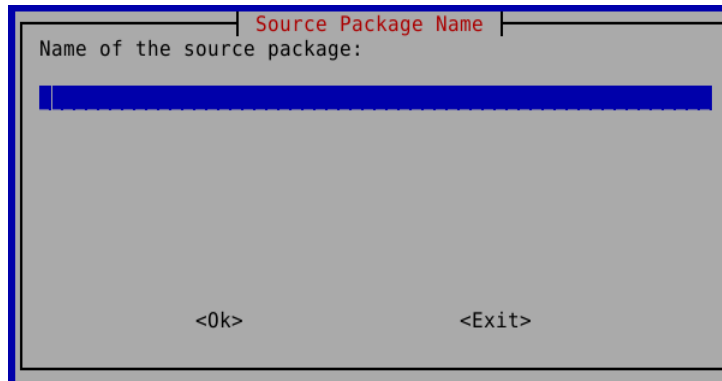


Abbildung 32.2.: Name des Quellcode-Paketes

Enthält die Variable einen Wert, wird gefragt, ob dies der richtige Wert sei. Dies dient zum Einen der Kontrolle der vorherigen Eingabe und zum Anderen eröffnet dies die Möglichkeit des Editierens (Kapitel 33.1, Seite 177) der Konfigurationsdatei mittels der Funktion *AskConfig*.

```
120 <AskConfig1 120>≡ (119)
    if ! whiptail --title "Source Package Name" \
        --yesno "The name of the source package is ${SourceName}" \
        --yes-button "Yes" --no-button "No" 15 60
    <AskConfig2 121>
```

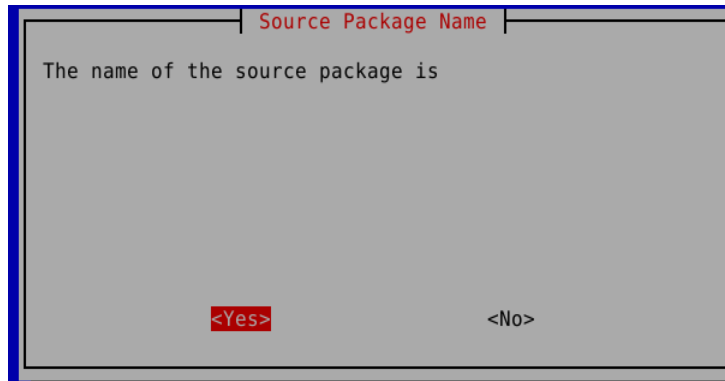


Abbildung 32.3.: Namen des Quellpaketes angeben

```

121  <AskConfig2 121>≡ (120)
      then
        SourceName=$(whiptail --title "Name of the source package" \
          --inputbox "Real name of the source package:" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

        if [ ${#SourceName} -eq 0 -o $? -ne 0 ]
        then
          exit
        else
          changeflag=1
        fi
      fi

  <AskConfig3 122>

```

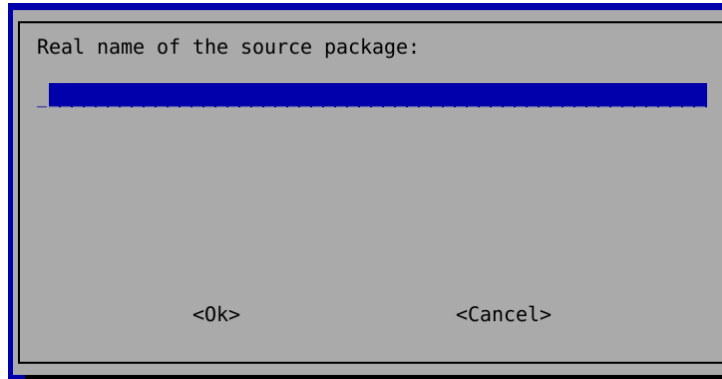


Abbildung 32.4.: Korrekten Namen des Quellpaketes angeben

Nun wird der Name des zu bauenden Paketes ermittelt. Dafür wird der Variablen *PackName*, wenn sie noch nicht existiert, der Wert der Variablen *SourceName* als Standardwert zugewiesen. Auch hierzu wird eine Bestätigung abgefragt und eine Korrekturmöglichkeit eröffnet.

```

122 <AskConfig3 122>≡ (121)
    if [ -z "${PackName}" ]
    then
        PackName=${SourceName}
        tadd=", too?"
    else
        tadd=""
    fi

    if ! whiptail --title "PackName" \
        --yesno "The name of the package is ${PackName}${tadd}" \
        --yes-button "Yes" --no-button "No" 15 60
    <AskConfig3-1 123>

```


Wenn die Variable *PackName* noch nicht existierte:

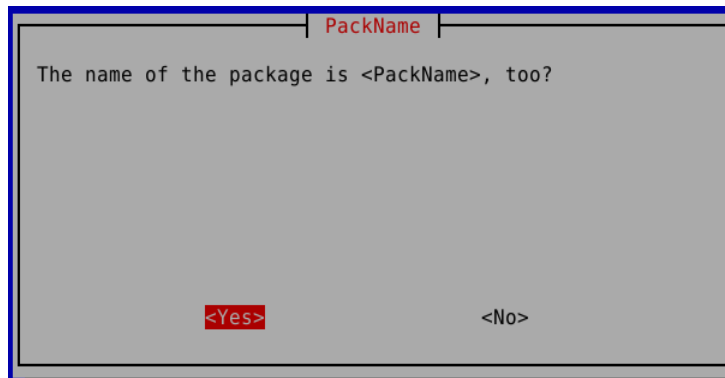


Abbildung 32.5.: Ist der Paketname korrekt?

Wenn die Variable *PackName* bereits existierte:

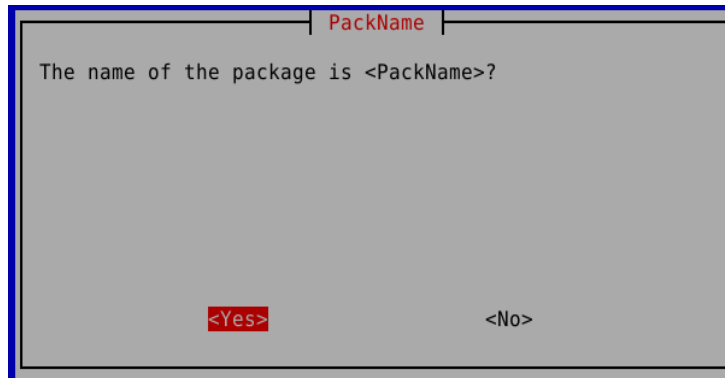


Abbildung 32.6.: Ist der Paketname korrekt?

```

123  <AskConfig3-1 123>≡ (122)
      then
        PackName=$(whiptail --title "Name of the Debian Package" \
          --inputbox "Real name of the package,\nwhich should be built:" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

        if [ ${#PackName} -eq 0 -o $? -ne 0 ]
        then
          exit
        else
          changeflag=1
        fi
      fi

```

<AskConfig4 124>

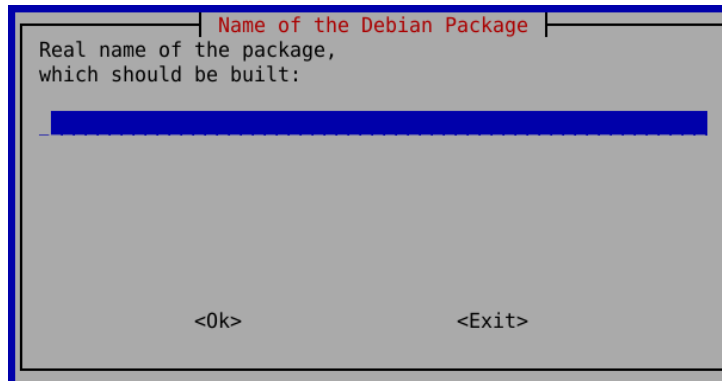


Abbildung 32.7.: Korrekten Name des Paketes angeben.

Nun wird der Name der Gruppe abgefragt, unter welcher das Git-Repository auf *salsa.debian.org* angelegt werden soll.

```
124 <AskConfig4 124>≡ (123)
    if [ -z "${SalsaName}" ]
    then
        SalsaName=$(whiptail --title "Group at Salsa" \
            --inputbox "Group on salsa.debian.org:" \
            --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
        if [ $? -ne 0 ] # Cancel-Button was pressed
        then
            exit
        else
            SalsaName="${SalsaName}/${SourceName}.git"
        fi
    fi
```

<AskConfig5 125a>

Abbildung 32.8.: Name der Gruppe auf *salsa.debian.org* angeben.

Es gibt verschiedene Paketier-Teams (beispielsweise für Java- oder Python-Pakete)¹. Beim Python-Team muss zusätzlich das Verzeichnis *packages* angegeben werden, also *python-team/packages*.

Diese Teams verfügen über eigene Gruppen auf *salsa.debian.org*. Mit <https://salsa.debian.org/explore/groups> können alle öffentlichen Gruppen angezeigt werden.

Soll ein Paket unabhängig von einem Paketier-Team betreut werden, ist als Gruppe *Debian* einzutragen.

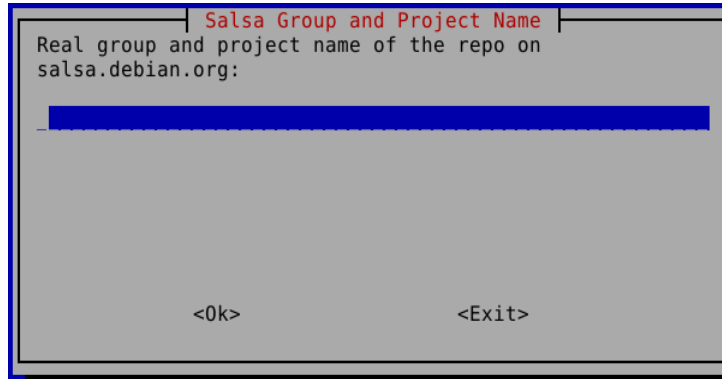
```
125a <AskConfig5 125a>≡ (124)
      if ! whiptail --title "Salsa Name" \
        --yesno "Group and project name of the repo on salsa.debian.org is $SalsaName" \
        --yes-button "Yes" --no-button "No" 15 60
<AskConfig5-1 125b>
```

Abbildung 32.9.: Name der Gruppe auf *salsa.debian.org* angegeben.

```
125b <AskConfig5-1 125b>≡ (125a)
      then
        SalsaName=$(whiptail --title "Salsa Group and Project Name" \
          --inputbox "Real group and project name of the repo on salsa.debian.org:" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)

<AskConfig5-2 126a>
```

¹https://wiki.debian.org/Teams#Packaging_teams

Abbildung 32.10.: Name der Gruppe auf *salsa.debian.org* angeben.

```
126a  <AskConfig5-2 126a>≡ (125b)
      if [ $#SalsaName -eq 0 -o $? -ne 0 ]
      then
          exit
      else
          changeflag=1
      fi
  fi
```

<AskConfig6 126b>

Es wird nun geprüft, ob im Verzeichnis *~/debian_project* eine Datei *DefaultValues* existiert. In dieser Datei werden Variablen Werte zugewiesen, die für viele Projekte gelten (Kapitel 19.2.2.2, Seite 62). Das Skript *DefaultValues* wird dann ausgeführt.

```
126b  <AskConfig6 126b>≡ (126a)
      if [ -f ${ConfigPath}/DefaultValues ]
      then
          . ${ConfigPath}/DefaultValues
      fi
```

<AskConfig7 126c>

Der Variablen *DefaultProjectPath* ist als Wert der Pfad zugewiesen, welcher zu dem Verzeichnis führt, das die einzelnen Projektverzeichnisse als Unterverzeichnisse enthält.

```
126c  <AskConfig7 126c>≡ (126b)
      if [ -n "${DefaultProjectPath}" ]
      then
          ProjectPath=${DefaultProjectPath}
      fi

      if [ -z "${ProjectPath}" ]
      then
          ProjectPath=$(whiptail --title "Path to Project Directory" \
            --inputbox "Path to the project directory on your local machine\n \
            (without '/${OrigName}':" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
```

<AskConfig8 127>

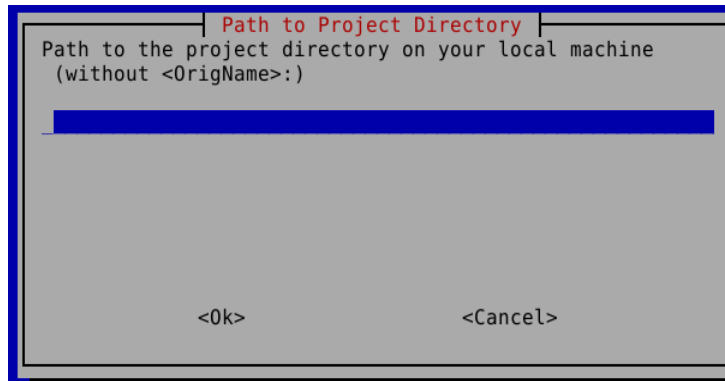


Abbildung 32.11.: Pfad zum Projektverzeichnis auf der lokalen Maschine

127 *<AskConfig8 127>*≡

(126c)

```

if [ -z "${ProjectPath}" ]
then
    echo -e "Path to the project directory on your local machine\n \
    (without '${OrigName}':)"
    read ProjectPath
fi
changeflag=1
fi

if ! whiptail --title "ProjectPath" \
--yesno "Path to the project directory on your local machine \
is ${ProjectPath}/${OrigName}" \
--yes-button "Yes" --no-button "No" 15 60

```

<AskConfig9 128>

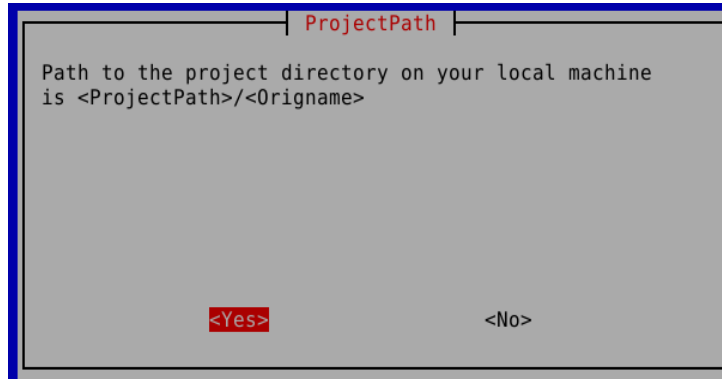


Abbildung 32.12.: Pfad zum Projektverzeichnis auf der lokalen Maschine mit OrigName

```
128 <AskConfig9 128>≡ (127)
    then
      ProjectPath=$(whiptail --title "Path to Project Directory" \
        --inputbox "Real path to the project directory on your local machine\n \
        (without '/${OrigName}':" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    <AskConfig9-1 129>
```

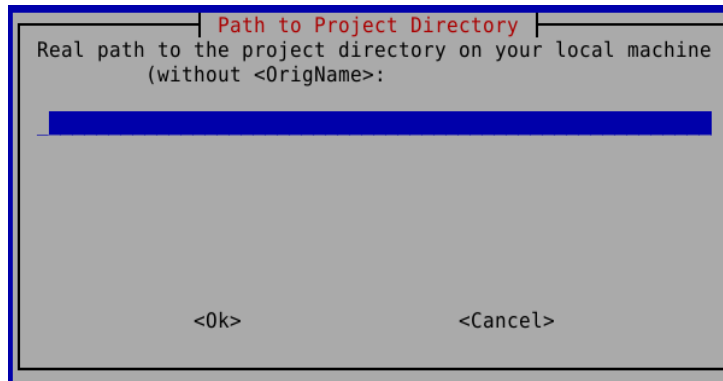


Abbildung 32.13.: Pfad zum Projektverzeichnis auf der lokalen Maschine (real)

```

129  <AskConfig9-1 129>≡ (128)
      if [ -z "${ProjectPath}" ]
      then
          echo -e "Path to the project directory on your local machine\n \
(w without '/${OrigName}':)"
          read ProjectPath
      fi
      if [ -z "${ProjectPath}" ]
      then
          exit
      else
          changeflag=1
      fi
  fi
<AskConfig10 131b>

```

32.1.2. Abfrage spezieller Variablen für die Konfigurationsdatei

Nun werden die Variablen ermittelt, die nur für spezielle Arten von Paketen benötigt werden. Begonnen wird mit den Variablen für Java-Paketen (Kapitel 32.1.2.2, Seite 131)

Außerdem wird gegebenenfalls in die Konfigurationsdatei der Pfad zu einem benötigten Plugin eingetragen.

32.1.2.1. Ermittlung der Plugin-Pfade

Bestimmte Arten von Paketen werden unter Verwendung von Plugins zum (Haupt-)Skript gebaut. Es handelt sich hierbei um Pakete, die mit Java, *maven* gebaut werden, um Mozilla-Erweiterungen und Python-Programme.

Die Funktion *DetectPlugins* ermittelt die Pfade zu Plugin-Skriptdateien. Sie kann für weitere Plugins verwendet werden. Sie wird derzeit nicht benutzt.

Beim Aufruf von *DetectPlugins* müssen zwei Optionen übergeben werden. Dabei wird die erste Option (*\$1*) an *PluginName* und die zweite (*\$2*) an *PluginFile* übergeben.

```

130 <DetectPlugins 130>≡ (172)
function DetectPlugins {
    # Not Used.
    # Funktion to find Plugins

    # This function needs two options: the name of the plugin
    # and the name of the plugin script file
    PluginName=$1
    PluginFile=$2

    # Determine path to the (needed) plugin script
    PluginPathL=$(locate ${PluginFile})
    PluginPathA=(${PluginPathL})

    # If there is only one result
    if [ $#PluginPathA[@] -eq 1 ]
    then
        PluginPath=${PluginPathA[0]}
    # If there are some results
    elif [ $#PluginPathA[@] -gt 1 ]
    then
        i=0; slct=''
        for element in ${PluginPathA[*]}
        do
            slct=$slct' '$i' '${element}' off '
            i=$(expr $i + 1)
        done

        PluginNr=$(whiptail --title "${PluginName} plugin found" \
            --radiolist "Select:" 15 60 5 $slct \
            --cancel-button "Cancel" 3>&2 2>&1 1>&3)
    }

<DetectPlugins3 131a>

```



```

131a  <DetectPlugins3 131a>≡ (130)
        if [ $? -eq 1 ]
        then
            return 24
        fi

        PluginPath=${PluginPathA[${PluginNr}]}
# If there is no result
else
    PluginFlag=0
    whiptail --title "File not found" \
--msgbox "'${PluginFile}' was not located!" \
    15 60
    return 24
fi
}

<AskConfig 119>

```

32.1.2.2. Variablenabfrage für Java-Pakete

Bestimmte Arten von Paketen erfordern eine spezielle Behandlung. Diese erfolgt unter anderem durch separate Skripte. Diese Behandlung wird durch „flags“ gesteuert. Damit wird auch ein unnötiges Laden der Plugin-Skripte vermieden.

```

131b  <AskConfig10 131b>≡ (129)
        # Java Flag
        # This is needed to trigger special entries
        # in debian/control and debian/rules
        if [ -z "${JavaFlag}" ]
        then
            if whiptail --title "Java" --defaultno \
--yesno "Do you want to build a Java package?" \
--yes-button "Yes" --no-button "No" --defaultno 15 60

```

<AskConfig10-1 132a>



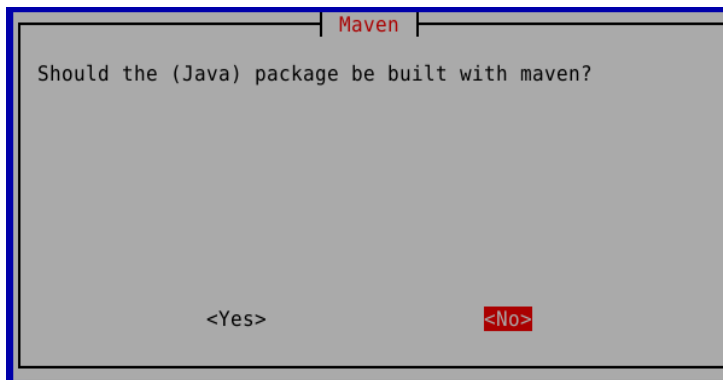
Abbildung 32.14.: Soll ein Java-Paket gebaut werden?

```
132a  <AskConfig10-1 132a>≡ (131b)
      then
        JavaFlag=1
      else
        JavaFlag=0
      fi
      changeflag=1
    fi
```

<AskConfig11 132b>

Abgefragt wird auch, ob das Maven-Plugin (Kapitel 47, Seite 419) verwendet werden soll. Dieses wird dann im System lokalisiert und der Pfad zum Skript in der Konfigurationsdatei hinterlegt.

```
132b  <AskConfig11 132b>≡ (132a)
      # Maven Plugin
      if [ ${JavaFlag} -eq 1 ]
      then
        if [ -z "${MavenPluginFlag}" ]
        then
          if whiptail --title "Maven" --defaultno \
            --yesno "Should the (Java) package be built with maven?" \
            --yes-button "Yes" --no-button "No" --defaultno 15 60
          <AskConfig12 133a>
```

Abbildung 32.15.: Soll ein Java-Paket mit *maven* gebaut werden?

133a `<AskConfig12 133a>`≡ (132b)

```

then
    MavenPluginFlag=1
else
    MavenPluginFlag=0
fi
changeflag=1
fi
fi

```

`<AskConfig15 133b>`

Beim Aufruf von *DetectPlugins* müssen zwei Optionen übergeben werden. Dabei wird 'Maven' als *\$1* an die Variable *PluginName* und *build-gbp-maven-plugin.sh* als *\$2* an die Variable *PluginFile* übergeben (s. Kapitel 32.1.2.1, Seite 130).

32.1.2.3. Variablenabfrage für Mozilla-Erweiterungen

133b `<AskConfig15 133b>`≡ (133a)

```

# Webext Flag
# This is needed to trigger special entries
# in debian/control and debian/rules
if [ -z "${WebextFlag}" ]
then
    if whiptail --title "Mozilla AddOns" --defaultno \
        --yesno "Do you want to build a Mozilla AddOn package?" \
        --yes-button "Yes" --no-button "No" --defaultno 15 60

```

`<AskConfig15-1 134a>`

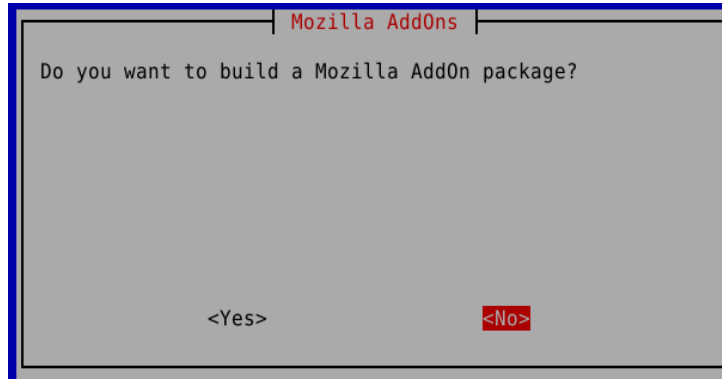


Abbildung 32.16.: Soll eine Erweiterung für Mozilla paketiert werden?.

```
134a  <AskConfig15-1 134a>≡ (133b)
      then
        WebextFlag=1
      else
        WebextFlag=0
      fi
      changeflag=1
    fi

    <AskConfig18 134b>
```

32.1.2.4. Variablenabfrage für Python3-Pakete

Auch für das Paketieren von Python-Paketen kann auf ein entsprechendes Plugin zurückgegriffen werden. Dafür werden zunächst die benötigten Informationen abgefragt und in die Konfigurationsdatei eingetragen.

```
134b  <AskConfig18 134b>≡ (134a)
      # Python Flag
      # This is needed to trigger special entries
      # in debian/control and debian/rules
      if [ -z "${PythonFlag}" ]
      then
        if whiptail --title "Python3 programs" --defaultno \
          --yesno "Do you want to build a Python3 package?" \
          --yes-button "Yes" --no-button "No" --defaultno 15 60
        <AskConfig19 135>
```



Abbildung 32.17.: Soll ein Python3-Pakte gebaut werden?.

```
135  <AskConfig19 135>≡ (134b)
      then
        PythonFlag=1
      else
        PythonFlag=0
      fi
      changeflag=1
    fi
<AskConfig20 136>
```

32.1.3. Speichern der Konfiguration

Das (Neu-)Erstellen der Konfigurationsdatei geschieht im folgenden Teil der Funktion *AskConfig*.

Es wird zunächst geprüft, ob bereits eine Konfigurationsdatei für das Projekt existiert. Sofern vorhanden wird sie gelöscht.

Die Konfigurationsdatei ist ein Shell-Skript. Sie beginnt daher mit einer entsprechenden *Shebang*.

```
136 <AskConfig20 136>≡ (135)
    if [ $changeflag -eq 1 ]
    then
        if [ -f ${ConfigPath}${OrigName} ]
        then
            rm ${ConfigPath}${OrigName}
        fi
        touch ${ConfigPath}${OrigName}

        # Shebang of the config file
        SB='#!/usr/bin/bash'

        # The config file is a shell script
        echo ${SB} >> ${ConfigPath}${OrigName}
        echo '# ConfigFile for '${OrigName}' >> ${ConfigPath}${OrigName}
        echo '## General parameters' >> ${ConfigPath}${OrigName}
        echo 'SourceName='${SourceName}' >> ${ConfigPath}${OrigName}
        echo 'PackName='${PackName}' >> ${ConfigPath}${OrigName}
        echo 'ProjectPath='${ProjectPath}' >> ${ConfigPath}${OrigName}
        echo 'SalsaName='${SalsaName}' >> ${ConfigPath}${OrigName}
<AskConfig22 137a>
```

Die folgenden Einträge in der Konfigurationsdatei werden nur erstellt, wenn entsprechende Flags gesetzt wurden. Dies gilt für Java-Pakete. Ein zusätzlicher Eintrag erfolgt, wenn sie mit *maven* gebaut werden. Dies gilt auch für die Erweiterungen für Firefox und Thunderbird und Programme in der Programmiersprache Python.

```
137a <AskConfig22 137a>≡ (136)
    echo '## Parameters for Java packages'>> ${ConfigPath}${OrigName}
    echo 'JavaFlag=${JavaFlag} >> ${ConfigPath}${OrigName}
    if [ ${JavaFlag} -eq 1 ]
    then
        echo 'MavenPluginFlag=${MavenPluginFlag} >> ${ConfigPath}${OrigName}
    fi

    echo '## Parameters for Webext packages'>> ${ConfigPath}${OrigName}
    echo 'WebextFlag=${WebextFlag} >> ${ConfigPath}${OrigName}

    echo '## Parameters for Python3 packages'>> ${ConfigPath}${OrigName}
    echo 'PythonFlag=${PythonFlag} >> ${ConfigPath}${OrigName}s
    changeflag=0
    fi
}
```

<ReplaceTilde 137b>

Da beim Ablauf des Skriptes die Tilde (~) im Pfad nicht automatisch durch */home/<username>* ersetzt wird, muss dies durch eine Funktion *ReplaceTilde* im Programmskript geschehen. Ferner wird ein eventueller Slash (/) am Ende des Pfades entfernt.

```
137b <ReplaceTilde 137b>≡ (137a)
function ReplaceTilde {
    # Called by ConfigFileLEC GbpConfIntegration
    RecentUser=$(whoami)
    set +e
    tp=$(echo ${SuspectPath} | grep --count '~')
    if [ $tp -ge 1 ]
    then
        CleanPath=$(echo ${SuspectPath} | \
            sed --expression="s/~~/\home\/${RecentUser}/g")
    else
        CleanPath=${SuspectPath}
    fi

    # Replace / at the end
    CleanPath=$(echo ${CleanPath} | sed --expression="s/\$///")
    set -e
}
```

<SetDefaults 180>

32.1.4. Beispiel einer Konfigurationsdatei

```
#!/usr/bin/bash
# ConfigFile for <OrigName>
## General parameters
SourceName=<SourceName>
PackName=<PackName>
ProjectPath=/home/mechtilde/Projekte/Git/01_Salsa
SalsaName=<Name of the team>/<SourceName>.git
## Parameters for Java packages
JavaFlag=0
## Parameters for Webext packages
WebextFlag=0
## Parameters for Python3 packages
PythonFlag=0
RecentBranch=debian/sid
## Maintainer and Uploaders
Maintainer=<Name and E-Mail-Address of the Maintainer>
Uploaders=<Name and E-Mail-Address of the Uploaders>
## Download from upstream
DownloadURL=<Upstream URL for download>
RecentUpstreamSuffix=.xpi
# debian/sid_Dist=sid
```

32.2. Anlegen der Infrastruktur

Zur Infrastruktur jedes Projektes gehören Verzeichnisse, eine Log-Datei und ein Git-Repository. Sofern diese nicht bereits vorhanden sind, werden die notwendigen Verzeichnisse und die Log-Datei angelegt.

Zur Infrastruktur gehören auch ein *Chroot*-Verzeichnis *base.cow* oder ein *Sbuild-Chroot*-Verzeichnis. Diese werden nur angelegt, wenn sie für die Distribution, für die das Paket gebaut werden soll, noch nicht existieren. (Kapitel ??, Seite ??)

32.2.1. Anlegen der notwendigen Verzeichnisse

Zunächst werden die zuvor definierten Pfade (Kapitel 32.2.2, Seite 139) angelegt.

```
138 <CreateDirsAndLogfile 138>≡ (118)
    function CreateDirsAndLogFile {
        # Called by BuildApp ConfigFileLEC

        # Replace tilde if necessary
        SuspectPath=${ProjectPath}
        ReplaceTilde
        ProjectPath=${CleanPath}

    <CreateDirsAndLogfile1 139a>
```


32.2.2. Definition der Pfade

Am Ende der Funktion *ConfigFileLEC* werden noch zwei zusammengesetzte Pfade definiert. Die Beschreibung dieser Pfade findet sich in Kapitel 19.2 (Seite 60).

```
139a <CreateDirsAndLogfile1 139a>≡ (138)
      # Set paths
      PrjPath=${ProjectPath}/${OrigName}
      GitPath=${PrjPath}/${SourceName}

      # Create directories if necessary
      mkdir --parents ${PrjPath} # redundantly?
      mkdir --parents ${GitPath}

<CreateLogFile 139b>
```

32.2.3. Anlegen der Log-Datei

In die Log-Datei werden bei jedem Programmstart zunächst Datum und Uhrzeit eingetragen.

```
139b <CreateLogFile 139b>≡ (139a)
      # Create Log-File
      cd ${PrjPath}
      log=${PrjPath}/${OrigName}.log.txt
      touch ${log}
      echo -e "\n\n===\n=== $(date) ===\n===\n">>${log}
      echo "ConfigFile OK!" >> ${log}
    }

<InsertDebName 155a>
```

32.3. Git-Repositories

Für die Arbeit mit *git-buildpackage* ist ein lokales Git-Repository mit Arbeitsverzeichnis wesentlich.

Das Programmskript geht davon, dass sowohl ein lokales Repository (Kapitel 20.4, Seite 78) als auch ein Repository auf *salsa.debian.org* (Kapitel 21, Seite 79) angelegt werden soll. Daneben berücksichtigt das Programmskript auch ein Git-Repository auf einem eigenen Git-Server (Kapitel 20.4.2, Seite 78).

Die Neuanlage eines Git-Repositories ist der erste Schritt zum Bauen eines neuen Debian-Paketes. Danach wird der Quellcode für die (neue) Version heruntergeladen (Kapitel 34.3, Seite 202), eine Revision gebaut (Kapitel 35, Seite 253) und schließlich hochgeladen. (Kapitel 43, Seite 387). Daher wird der Nutzer gefragt, ob er ein neues Paket bauen will.

32.3.1. Gibt es bereits ein Git-Repository?

Vorsorglich wird geprüft, ob bereits ein lokales Git-Repository für das anzulegende Projekt existiert. Es wird berücksichtigt, wenn in einem übergeordneten Verzeichnis ein Git-Repository existiert.

Existiert ein lokales Git-Repository, geht es mit der Auswahl eines Git-Zweiges weiter (s. Kapitel 33.4, Seite 182).

```
140a <BuildApp7 140a>≡ (113)
    ## Checks whether there is a git repo
    cd ${GitPath}
    set +e
    git status 1>/dev/null 2>&1

    # '==' does the same as '-eq'
    if [ $? == 0 ]
    then
        if [ -d .git ]
        then
            SelectBranch
        else
<BuildApp8 140b>
```

Wenn ein Git-Repository in einem übergeordneten Verzeichnis existiert, wird ein entsprechender Hinweis gegeben.

```
140b <BuildApp8 140b>≡ (140a)
    SupOrdMsg="Is there a git repository in a superordinate directory?"
    echo ${SupOrdMsg} >> ${log}
    whiptail --title="Attention!" --msgbox "${SupOrdMsg}" 15 60
    StartTasks
    fi
    else
    StartTasks
    fi

<BuildApp10 176b>
```

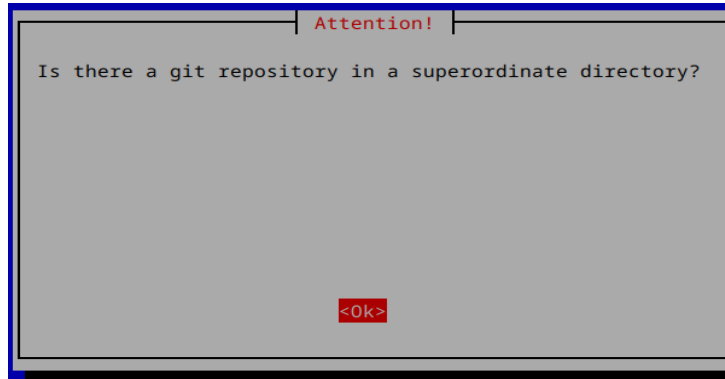


Abbildung 32.18.: Gibt es ein übergeordnetes Git-Repository?

Existiert kein Git-Repository im entsprechenden Verzeichnis, wird die Funktion *StartTasks* aufgerufen.

32.3.2. Auswahldialog

Existiert noch kein Git-Repository, stellt das Skript vier Möglichkeiten bereit, ein solches Repository zu erstellen. Dies sind die Neuanlage mittels *git init* und das Klonen eines bestehenden (Git-)Repositories von *salsa.debian.org* (Kapitel 32.5, Seite 160). Letzteres setzt vor allem voraus, dass dort ein Zweig *pristine-tar* vorhanden ist.

Andernfalls kann ein Repository durch *gbp import-dsc* angelegt werden (Kapitel 32.6, Seite 168).

Schließlich wird auch der Sonderfall des Sponsorings berücksichtigt (Kapitel 32.7, Seite 170)

```
141 <StartTasks 141>≡
    function StartTasks {
        # Called by BuildApp

        Task=$(whiptail --title "Tasks for building a new package:" \
        --radiolist "What do you like to do to build a new package?" 17 60 9 \
        "0" "Create a git repo and download upstream code" on \
        "11" "Clone an existing repo from Salsa" off \
        "12" "Importing already existing Debian packages" off \
        "13" "Importing from mentors.debian.net for sponsoring" off \
        --cancel-button "Exit" 3>&2 2>&1 1>&3)
```

<StartTasks1 142a>

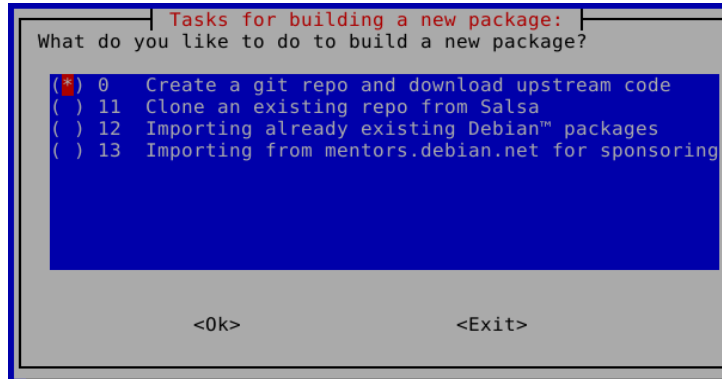


Abbildung 32.19.: Ein neues Paket erstellen

142a `<StartTasks1 142a>≡` (141)

```

if [ -z "${Task}" ]
then
    exit
fi
TaskSelect
}

```

`<DebDiffCheckList 366b>`

Die Punkte des Menüs befassen sich mit verschiedenen Wegen, ein neues Projekt aufzusetzen. Es geht vor allem um die Beschaffung des Quellcodes des Upstream-Projektes.

Der Aufruf der entsprechenden Funktionen erfolgt durch die Funktion *TaskSelect*.

Bei *Create a git repo and download upstream code* wird durch die Funktion *BuildNewPackage* zunächst das lokale Git-Repository angelegt (Kapitel 32.4, Seite 144). Sodann wird eine neue Version heruntergeladen (Kapitel 34.3, Seite 202).

142b `<TaskSelect 142b>≡`

```

function TaskSelect {
    # Called by StartTasks CommonTasks

    ## The lines below serve also the sequence control. A called function
    ## changes finally the variable 'Task', so one of the following
    ## if-clauses matches. That is why the following lines can not be
    ## replaced by a case statement.

    NoPull=0 # Flag for PullFromSalsa

    ## Start tasks

    # Building a new package from archive
    if [ $Task -eq 0 ]
    then
        BuildNewPackage
    fi
}

```

`<TaskSelect0 143a>`

Bei *Clone an existing repo from Salsa* wird ein bereits vorhandenes Repository von *salsa.debian.org* heruntergeladen. Die Funktion *CloneFromSalsa* kloniert dieses Repository mittels des Befehles *gbg clone*. (Kapitel 32.5, Seite 160)

143a `<TaskSelect0 143a>`≡ (142b)

```
# Clone an existing repo from Salsa
if [ $Task -eq 11 ]
then
    CloneFromSalsa
fi
<TaskSelect1 143b>
```

Bei *Importing already existing Debian™ package* wird eine **.dsc*-Datei heruntergeladen und durch das Programm *gbp import-dsc* ein Git-Repository angelegt (Kapitel 32.6, Seite 168).

143b `<TaskSelect1 143b>`≡ (143a)

```
# Importing already existing Debian package
if [ $Task -eq 12 ]
then
    ImportDebianPackage
fi
<TaskSelect2 143c>
```

Ein spezieller Fall für die Erstellung eines Git-Repositories mit *gbp import-dsc* ist das Sponsoring (Kapitel 32.7, Seite 170). Dabei unterstützt ein erfahrenes Mitglied des Debian-Projektes einen Maintainer ohne Upload-Rechte, indem das Mitglied das Paket signiert und hochlädt.

143c `<TaskSelect2 143c>`≡ (143b)

```
# Importing from mentors.debian.net for sponsoring
if [ $Task -eq 13 ]
then
    Import4Sponsoring
fi

<TaskSelect3 192>
```

32.4. Neuanlage eines lokalen Git-Repositorys

Eine Möglichkeit der Erstellung eines Git-Repositorys ist die Neuanlage mittels *git init*. Dies erfolgt in der Funktion *BuildNewPackage*. Zuvor erfolgt noch ein entsprechender Eintrag in der Log-Datei.

```

144a <BuildNewPackage 144a>≡ (159b)
function BuildNewPackage {
    # Called by TaskSelect
    cd ${GitPath}
    if [ -d .git ]
    then
        echo "There seems already to be a git repository in ${GitPath}." >> ${log}
        if ! whiptail --title "Warning" \
            --yesno "There seems already to be a git repository in ${GitPath}.\n \
            However continue?" --yes-button "Yes" --no-button "No" 15 60
        then
            echo "Exit" >> ${log}
            exit
        fi
    fi
}
<BuildNewPackage1 144b>

```

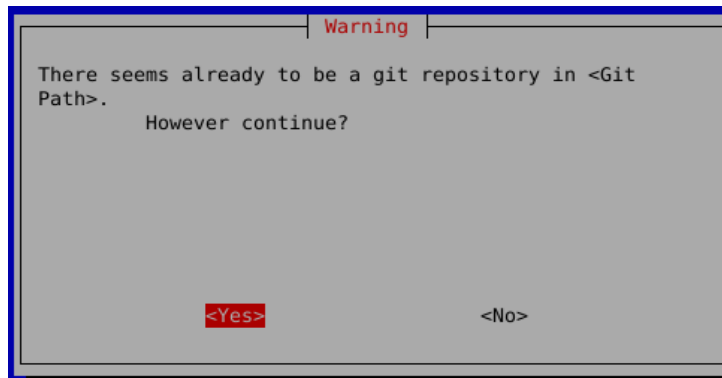


Abbildung 32.20.: Das Git-Repository existiert bereits.

32.4.1. Git-Repository anlegen

```

144b <BuildNewPackage1 144b>≡ (144a)
    else
        echo "In ${GitPath} a new git repository will be created." >> ${log}
        git init
    <BuildNewPackage2 158a>

```

32.4.2. Name und E-Mail-Adresse ins Git-Repository einfügen

Der Name und die E-Mail-Adresse des Maintainers können in die Konfigurationsdatei des Git-Repositorys eingetragen werden, das für das Projekt angelegt worden ist. Diese Daten werden beispielsweise jeder Commit-Mitteilung hinzugefügt.

Zunächst wird abgefragt, ob dies erwünscht ist. Diese Information kann bereits höher-rangig hinterlegt worden sein.

```
145a <BuildNewPackage3 145a>≡ (158a)
    if whiptail --title "Name and email" \
        --yesno "Do you like to add your name and email address \n \
        to the local git config file?" --yes-button "Yes" \
        --no-button "No" 15 60
    then
        AddNameAndEmail
    fi
<BuildNewPackage5 159a>
```

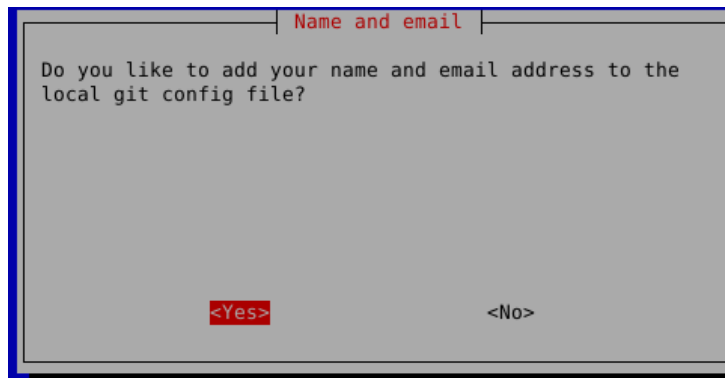


Abbildung 32.21.: Name und E-Mail.

Die folgende Funktion wird immer dann aufgerufen, wenn bei der Anlage eines neuen Git-Repositorys die Frage nach der Eintragung in die lokale Konfigurationsdatei bejaht wird.

```
145b <AddNameAndEmail 145b>≡ (157)
function AddNameAndEmail {
    # Called by BuildNewPackage CloneFromSalsa
    # ImportDebianPackage and itself

    DEBValues
    GCName=${DEBFULLNAME}
    GCEmail=${DEBEMAIL}

<AddNameAndEmail1 154a>
```

Damit der Paketierer nicht immer seinen vollen Namen und seine E-Mail-Adresse schreiben muss, sucht das Skript diese Daten zunächst in der Konfigurationsdatei und sodann in der `~/.bashrc`. Gegebenenfalls wird Gefundenes oder Erfragtes in die Konfigurationsdatei eingetragen.

```

146a  <Bashrc 146a>≡
      DEBEMAIL="your.email.address@example.org"
      DEBFULLNAME="Firstname Lastname"
      export DEBEMAIL DEBFULLNAME

146b  <DEBValues 146b>≡ (147)
      function DEBValues {
          # Called by DebianControlTemplate PatchHeader AddNameAndEmail
          MaintainerF=0
          set +e
          # if Maintainer is stored in config file
          if [ "${Maintainer}" ]
          then
              Maintainer=$(echo ${Maintainer} | sed --expression='s/_/ /g')
              Maintainer=$(echo ${Maintainer} | sed --expression='s/@lt@/</g')
              Maintainer=$(echo ${Maintainer} | sed --expression='s/@gt@/>/g')
              DEBFULLNAME=$(echo ${Maintainer} | sed --expression='s/<.*//')
              DEBEMAIL=$(echo ${Maintainer} | sed --expression='s/^.*</' | sed 's/>/'')
              MaintainerF=1
          fi

          # if Uploaders is stored in config file
          if [ "${Uploaders}" ]
          then
              Uploaders=$(echo ${Uploaders} | sed --expression='s/_/ /g')
              Uploaders=$(echo ${Uploaders} | sed --expression='s/@lt@/</g')
              Uploaders=$(echo ${Uploaders} | sed --expression='s/@gt@/>/g')
              DEBFULLNAME=$(echo ${Uploaders} | sed --expression='s/<.*//')
              DEBEMAIL=$(echo ${Uploaders} | sed --expression='s/^.*</' | sed 's/>/'')
          fi

          # Looking for a team as maintainer
          if [ ${MaintainerF} -eq 0 ]
          then
              TeamMaintainer
          fi
      }

<DEBValues3 148>

```


Es gibt Pakete, die von einem Team[18] betreut werden. Meist werden viele gleichartige Pakete von einem solchen Team betreut. In diesen Fällen tritt das Debian-Projektmitglied als *Uploader* auf und das Team als *Maintainer*. Dies wird entsprechend in die Datei *debian/control* (Kapitel 35.4.6, Seite 265) eingetragen. Im Programmskript werden bisher das *Java packaging team*, das *Mozilla WebExtensions packaging team* und das *Debian Python Team* berücksichtigt.

147 `<TeamMaintainer 147>≡` (264b)

```
function TeamMaintainer {
    # Called by DEBValues

    # Makes sure that variable exists
    if [ -z '${JavaFlag}' ]
    then
        JavaFlag = 0
    fi

    if [ ${JavaFlag} -eq 1 ]
    then
        Maintainer="Debian Java Maintainers \
        <pkg-java-maintainers@lists.alioth.debian.org>"
        MaintainerF=1
    fi

    if [ -z '${WebextFlag}' ]
    then
        WebextFlag = 0
    fi

    if [ ${WebextFlag} -eq 1 ]
    then
        Maintainer="Debian Mozilla Extension Maintainers \
        <pkg-mozext-maintainers@alioth-lists.debian.org>"
        MaintainerF=1
    fi

    if [ -z '${PythonFlag}' ]
    then
        PythonFlag = 0
    fi

    if [ ${PythonFlag} -eq 1 ]
    then
        Maintainer="Debian Python Team <team+python@tracker.debian.org>"
        MaintainerF=1
    fi
}
```

`<DEBValues 146b>`

```

148  <DEBValues3 148>≡ (146b)
      # Extracts DEBFULLNAME and DEBEMAIL from ~/.bashrc (if exist)
      if [ ${MaintainerF} -eq 0 ]
      then
        if grep --quiet 'DEBFULLNAME' ~/.bashrc
        then
          dfnb=$(grep DEBFULLNAME ~/.bashrc)
          dfnb=$(echo ${dfnb} | sed --expression='s/export .*//')
          dfnb=$(echo ${dfnb} | sed --expression='s/DEBFULLNAME=//')
          dfnb=$(echo ${dfnb} | sed --expression='s/"//g')
          dfnb=$(echo ${dfnb} | sed --expression="s/'//g")
          DEBFULLNAME=${dfnb}
        fi
        if grep --quiet 'DEBEMAIL' ~/.bashrc
        then
          demb=$(grep 'DEBEMAIL' ~/.bashrc)
          demb=$(echo ${demb} | sed --expression='s/export .*//')
          demb=$(echo ${demb} | sed --expression='s/DEBEMAIL=//')
          demb=$(echo ${demb} | sed --expression='s/"//g')
          demb=$(echo ${demb} | sed --expression="s/'//g")
          DEBEMAIL=${demb}
        fi
        Maintainer=${DEBFULLNAME}" <${DEBEMAIL}">"
        MaintainerF=1
      fi

      # Insert name and email address
      if [ ${MaintainerF} -eq 0 ]
      then
        DEBFULLNAME=$(whiptail --title "Name of the maintainer" \
          --inputbox "Please insert full name of the maintainer" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
      fi
    <DEBValues4-1 149a>

```

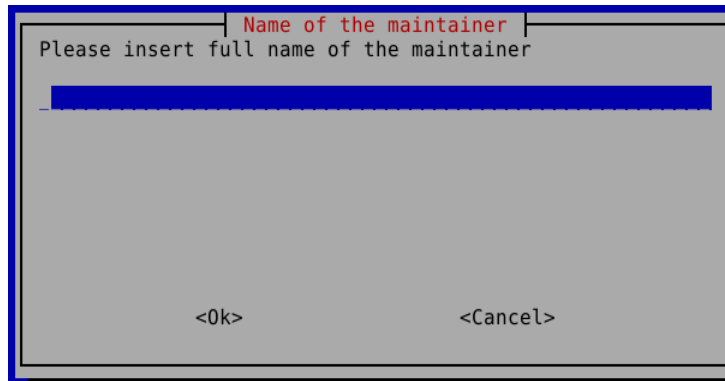


Abbildung 32.22.: Name des Maintainer

```
149a <DEBValues4-1 149a>≡ (148)
      DEBEMAIL=$(whiptail --title "Email of the maintainer" \
        --inputbox "Please insert email address of the maintainer" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
<DEBValues4-2 149b>
```

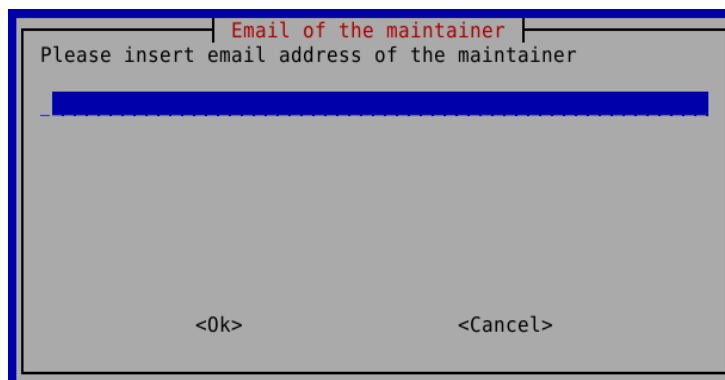


Abbildung 32.23.: E-Mail des Maintainers

```
149b <DEBValues4-2 149b>≡ (149a)
      Maintainer=${DEBFULLNAME}" <${DEBEMAIL}>"
      changeflag=1
      MaintainerF=1
      fi

      if ! whiptail --title "Maintainer" \
        --yesno "The full name and email address of the maintainer(s):\n \
          ${Maintainer}" --yes-button "OK" --no-button "Insert other" 15 60
<DEBValues5 150a>
```

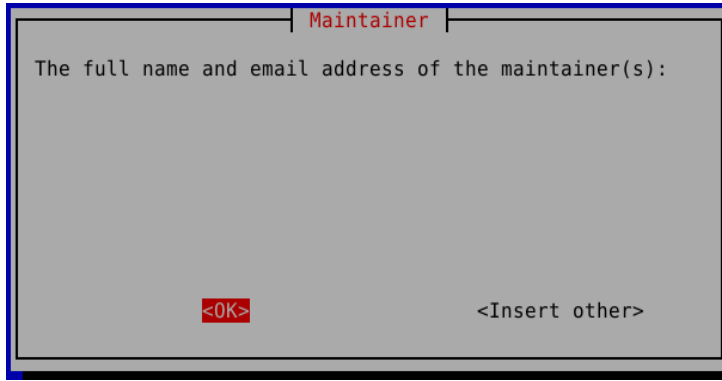


Abbildung 32.24.: Debian-Maintainer OK?

```
150a <DEBValues5 150a>≡ (149b)
      then
        DEBFULLNAME=$(whiptail --title "Name of the maintainer" \
          --inputbox "Please insert full name of the maintainer" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
      <DEBValues5-1 150b>
```

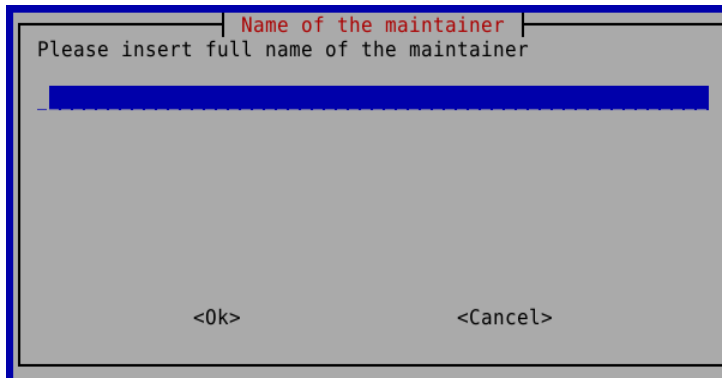


Abbildung 32.25.: Name des Debian-Maintainer

```
150b <DEBValues5-1 150b>≡ (150a)
      DEBEMAIL=$(whiptail --title "Email of the maintainer" \
        --inputbox "Please insert email address of the maintainer" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
      <DEBValues5-2 151a>
```

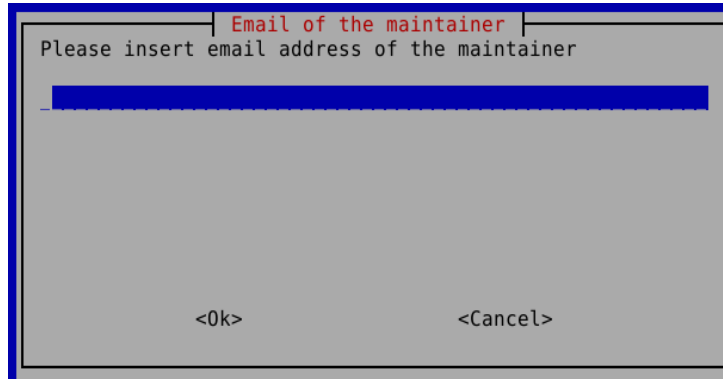


Abbildung 32.26.: E-Mail des Debian-Maintainer

151a $\langle DEBValues5-2\ 151a \rangle \equiv$ (150b)

```
Maintainer=${DEBFULLNAME}" <${DEBEMAIL}>"
changeflag=1
MaintainerF=1
```

fi

$\langle DEBValues6\ 151b \rangle$

151b $\langle DEBValues6\ 151b \rangle \equiv$ (151a)

```
# Insert maintainer data into config file if necessary
if [ $changeflag -eq 1 ]
then
  # Because Maintainer contains blanks
  MaintainerCF=$(echo ${Maintainer} | sed --expression='s/ /_/g')
  # Remove < and >
  MaintainerCF=$(echo ${MaintainerCF} | sed --expression='s/</@lt@/g')
  MaintainerCF=$(echo ${MaintainerCF} | sed --expression='s/>/@gt@/g')
  echo '## Maintainer and Uploaders' >> ${ConfigPath}${OrigName}
  echo 'Maintainer=${MaintainerCF}' >> ${ConfigPath}${OrigName}
  changeflag=0
```

fi

$\langle DEBValues7\ 152a \rangle$

```

152a  <DEBValues7 152a>≡ (151b)
      # Insert uploaders data into config file if necessary
      if [ ${JavaFlag} -eq 1 ]
      then
        if [ -z "${Uploaders}" ]
        then
          if grep --quiet 'DEBFULLNAME' ~/.bashrc
          then
            dfnb=$(grep DEBFULLNAME ~/.bashrc)
            dfnb=$(echo ${dfnb} | sed --expression='s/export .*//')
            dfnb=$(echo ${dfnb} | sed --expression='s/DEBFULLNAME=//')
            dfnb=$(echo ${dfnb} | sed --expression='s"/"/g')
            dfnb=$(echo ${dfnb} | sed --expression="s'/'/g")
            DEBFULLNAME=${dfnb}
          else
            DEBEMAIL=$(whiptail --title "Email of the uploader" \
              --inputbox "Please insert email address of the uploader" \
              --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
          fi
        fi
      fi
    <DEBValues8 152b>

```

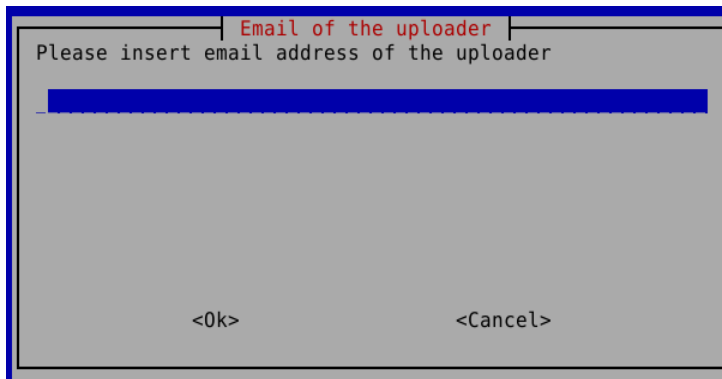


Abbildung 32.27.: E-Mail des Debian-Uploaders

```

152b  <DEBValues8 152b>≡ (152a)
      if grep --quiet 'DEBEMAIL' ~/.bashrc
      then
        demb=$(grep 'DEBEMAIL' ~/.bashrc)
        demb=$(echo ${demb} | sed --expression='s/export .*//')
        demb=$(echo ${demb} | sed --expression='s/DEBEMAIL=//')
        demb=$(echo ${demb} | sed --expression='s"/"/g')
        demb=$(echo ${demb} | sed --expression="s'/'/g")
        DEBEMAIL=${demb}
      else
        DEBEMAIL=$(whiptail --title "Email of the uploader" \
          --inputbox "Please insert email address of the uploader" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
      fi
    <DEBValues9 153>

```

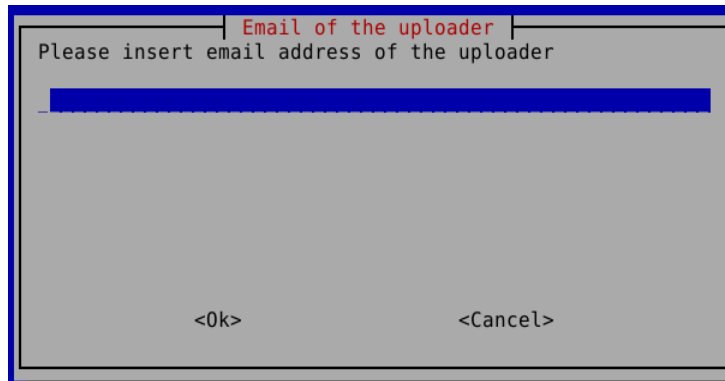


Abbildung 32.28.: E-Mail des Debian-Uploaders

```

153  <DEBValues9 153>≡ (152b)
      Uploaders=${DEBFULLNAME}" <${DEBEMAIL}>"
      # Because Uploaders contains blanks
      UploadersCF=$(echo ${Uploaders} | sed --expression='s/ /_/g')
      # Remove < and >
      UploadersCF=$(echo ${UploadersCF} | sed --expression='s/</@lt@/g')
      UploadersCF=$(echo ${UploadersCF} | sed --expression='s/>/@gt@/g')
      echo 'Uploaders=${UploadersCF}' >> ${ConfigPath}${OrigName}
      changeflag=0
      fi
    fi
    set -e
}

```

<DebianControlTemplate 265a>

```

154a  <AddNameAndEmail1 154a>≡ (145b)
      if [ -z "${GCName}" ]
      then
          InsertDebName
      fi
      # if [ -n "${GCName}" ]
      # then
          git config user.name ${GCName}
      # fi
      if [ -z "${GCEmail}" ]
      then
          InsertDebEmail
      fi
      # if [ -n "${GCEmail}" ]
      # then
          git config user.email ${GCEmail}
      # fi

      set +e
      configStr=$(git config --list | grep 'user')
      set -e

      if ! whiptail --title "Result" \
          --yesno "${configStr}\nAllright?" \
          --yes-button "Yes" --no-button "No" 15 60
      <AddNameAndEmail2 154b>

```

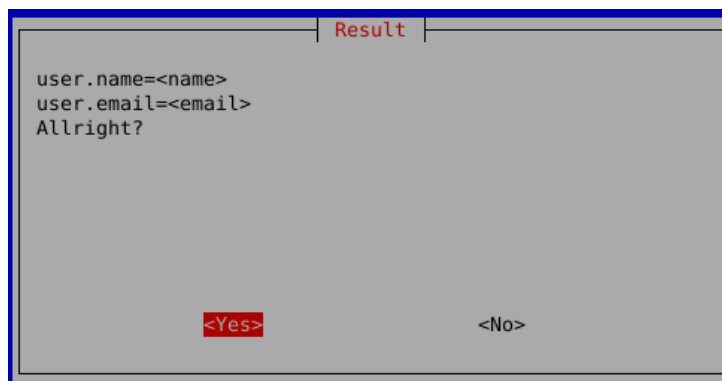


Abbildung 32.29.: Name und E-Mail des Maintainers korrekt?

```

154b  <AddNameAndEmail2 154b>≡ (154a)
      then
          AddNameAndEmail
      fi
  }

  <AddHomeServer 159b>

```


Die folgende Funktion dient zur Eingabe des Vor- und Nachnamens des Maintainers.

```
155a <InsertDebName 155a>≡ (139b)
function InsertDebName {

    # Called by AddNameAndEmail and itself
    DEBFULLNAME=$(whiptail --title "Name of the maintainer" \
        --inputbox "Please insert full name of the maintainer" \
        --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
<InsertDebName1 155b>
```

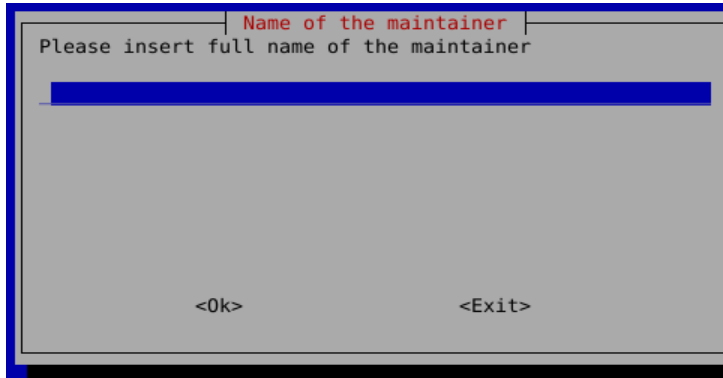


Abbildung 32.30.: Name des Maintainers eingeben

```
155b <InsertDebName1 155b>≡ (155a)
    if [ $? -ne 0 ]
    then
        exit
    fi

    # Test Name
    if [ -z "${DEBFULLNAME}" ]
    then
        whiptail --title "Your Name" \
            --msgbox "Your name is necessary." 15 60
        InsertDebName
    fi
}
<InsertDebEmail 156>
```

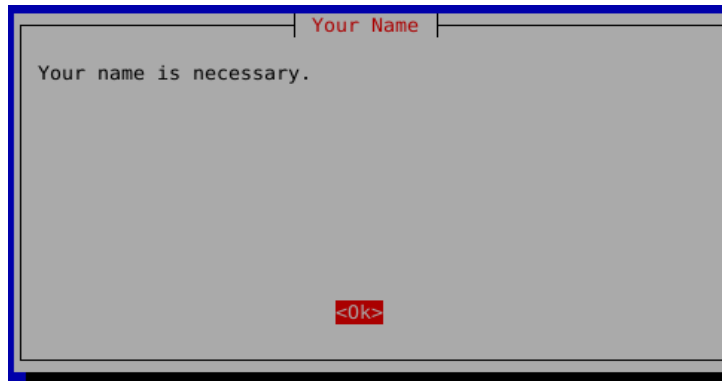


Abbildung 32.31.: Name des Maintainers erforderlich

Die folgende Funktion dient zur Eingabe des E-Mail-Adresse des Maintainers.

```
156 <InsertDebEmail 156>≡ (155b)
    function InsertDebEmail {
        # Called by AddNameAndEmail and itself
        DEBEMAIL=$(whiptail --title "Email of the maintainer" \
            --inputbox "Please insert email address of the maintainer" \
            --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
    }
    <InsertDebEmail1 157>
```

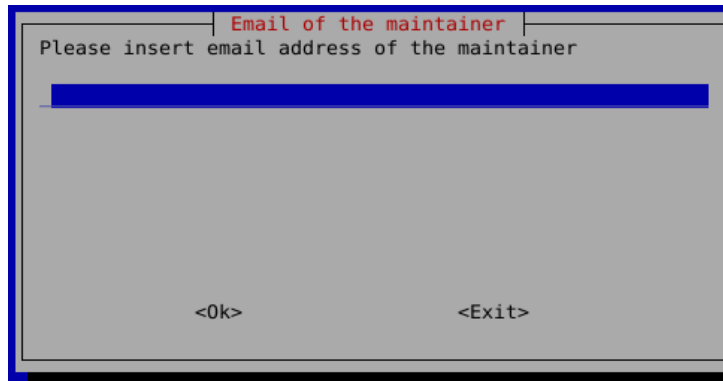


Abbildung 32.32.: E-Mail-Adresse des Maintainers eingeben

```

157  <InsertDebEmail1 157>≡ (156)
      if [ $? -ne 0 ]
      then
          exit
      fi

      # Regex string
      EmailR="\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}\b"

      # Test email address
      set +e
      if ! echo ${DEBEMAIL} | grep -E ${EmailR} > /dev/null
      then
          whiptail --title "Bad!" --msgbox "That's no email address." 15 60
          InsertDebEmail
      fi
      set -e
    }
    <AddNameAndEmail 145b>

```

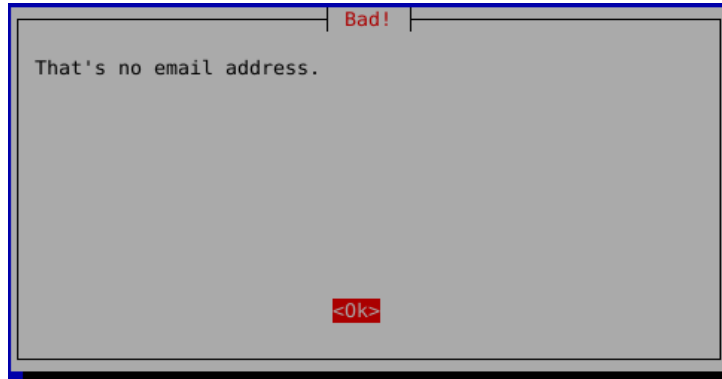


Abbildung 32.33.: Dies ist keine E-Mail-Adresse

32.4.3. Repositorium auf *salsa.debian.org*

Das Salsa-Repositorium wird als „Remote-Repository“ vom Programmskript eingetragen.

158a `<BuildNewPackage2 158a>≡ (144b)`
`git remote add salsa git@salsa.debian.org:${SalsaName}`
`<BuildNewPackage3 145a>`

32.4.3.1. Manuell

Das entsprechende Repositorium auf *salsa.debian.org* wird dann manuell (Kapitel 21.2, Seite 79) angelegt (s. Kapitel 50.1, Seite 443).

Hieran erinnert das Programmskript auch den Nutzer.

158b `<BuildNewPackage6 158b>≡ (159a)`
`whiptail --title "If not happened yet:" \
--msgbox "Please create a git repo ${SalsaName} \n \
on salsa.debian.org!" 15 60`
`<BuildNewPackage6-1 159c>`

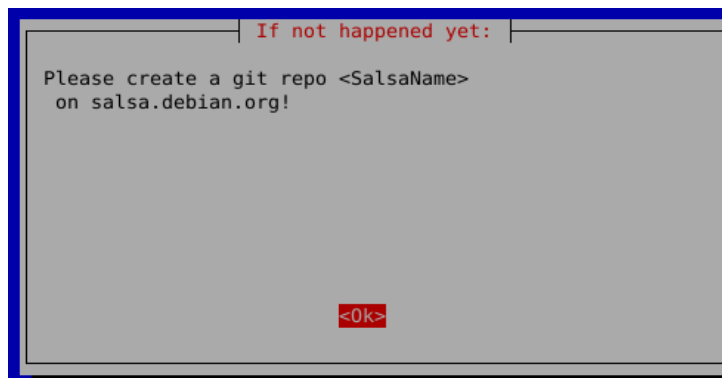


Abbildung 32.34.: Ein Git-Repositorium auf salsa anlegen

32.4.3.2. Innerhalb des Java-Teams

Innerhalb des Java-Teams (Kapitel 21.3, Seite 80) sollte ein neues Repository mit dem vom Team bereitgestellten *setup-salsa-repository*-Skript (Kapitel 50.1, Seite 443) angelegt werden.

32.4.4. Remoteserver anzeigen

Es werden die im Git-Repository eingetragenen Remoteserver angezeigt. Ein eigener Server wird nur angezeigt, wenn er zuvor eingetragen wurde.

```
159a <BuildNewPackage5 159a>≡ (145a)
      AddHomeServer
```

```
      fi
      <BuildNewPackage6 158b>
```

```
159b <AddHomeServer 159b>≡ (154b)
```

```
function AddHomeServer {
    # Called by BuildNewPackage AddGitServer ImportDebianPackage
    if [ -n "$ServerName" ]
    then
        git remote add home $(whoami)@${ServerName}:/srv/git/${SourceName}.git

        whiptail --title "Remoteserver" \
        --msgbox "New server added:\n$(git remote --verbose)" 15 60
        echo -e "New server added:\n $(git remote --verbose)" >> ${log}
    fi
}
```

```
<BuildNewPackage 144a>
```



```
Remoteserver
New server added:
home
mechtilde@pontian:/srv/git/buildscript-nw.git (fetch)
home
mechtilde@pontian:/srv/git/buildscript-nw.git (push)
salsa      git@salsa.debian.org:ddp-team/dpb.git
(fetch)
salsa      git@salsa.debian.org:ddp-team/dpb.git
(push)
<0k>
```

Abbildung 32.35.: Remoteserver ergänzen

Sodann wird eine neue Version heruntergeladen (Kapitel 34.3, Seite 202).

```
159c <BuildNewPackage6-1 159c>≡ (158b)
```

```
    NoPull=1 # Do not ask for pulling from salsa.debian.org
    Task=2 # Go to BuildNewVersion
}
```

```
<IdentifyBranches 164b>
```

NoPull=1 verhindert, dass vor dem Herunterladen der neuen Version gefragt wird, ob zuvor eventuelle Änderungen von *salsa.debian.org* heruntergeladen werden sollen.

32.5. Klonen von *salsa.debian.org*

In diesem Fall entfallen teilweise Schritte, die in den anderen Kapiteln genannt sind. Das Klonen erfolgt mit dem Befehl *gbp clone*. Danach wird beim erfolgreichen Klonen das Remote-Repository von *origin* nach *salsa* umbenannt. Damit ist eine aussagekräftigere Benennung der Repositorien möglich.

Andernfalls erfolgt eine Fehlermeldung und das Programm endet.

```
160a  <CloneFromSalsa 160a>≡ (196)
      function CloneFromSalsa {
          # Called by TaskSelect
          echo "Clone an existing repo from salsa.debian.org" >> ${log}
          cd ${PrjPath}
          gbp clone git@salsa.debian.org:${SalsaName} --aliases ${SourceName}
          if [ $? -eq 0 ]
          then
              cd ${GitPath}
              git remote rename origin salsa
              echo "${SalsaName} was cloned" >> ${log}
          else
              echo "${SalsaName} could not be cloned"
              exit
          fi
      }
```

<CloneFromSalsa2 160b>

Bei einem erfolgreichen Herunterladen werden mittels der Funktion *DebianBranchName* die heruntergeladenen Zweige (Branches) angezeigt und der aktive Zweig gesondert angezeigt.

```
160b  <CloneFromSalsa2 160b>≡ (160a)
      # Identify branches and choose one
      DebianBranchName
```

<CloneFromSalsa3 166b>

32.5.1. Bestimmung der Git-Zweige

Mit der Funktion *DebianBranchName* werden die Git-Zweige des geklonten Salsa-Repositoriums ermittelt.

Hierzu wird zunächst die Funktion *IdentifyBranches* (Kapitel 32.5.2, Seite 164) ausgeführt.

```
161a <DebianBranchName 161a>≡ (166a)
function DebianBranchName {
    # Called by CloneFromSalsa

    ## Identify and show branches
    IdentifyBranches
    ba=($bl)

    whiptail --title "Branches in repo ${OrigName}:" --msgbox "${bl}" 15 60

<DebianBranchName2 161b>
```



Abbildung 32.36.: Zeigt Liste der Git-Zweige

Sodann wird der aktive Git-Zweig ermittelt.

```
161b <DebianBranchName2 161b>≡ (161a)
    set +e
    for element in ${ba[*]}
    do
        # Find the default branch
        if echo ${element} | grep --quiet '^x_'
        then
            DefaultBranch=$(echo ${element} | sed --expression='s/^x_//')
            whiptail --title "Recent branch found" \
                --msgbox "Found: ${DefaultBranch}" 15 60
        fi
    fi
<DebianBranchName3 162>
```

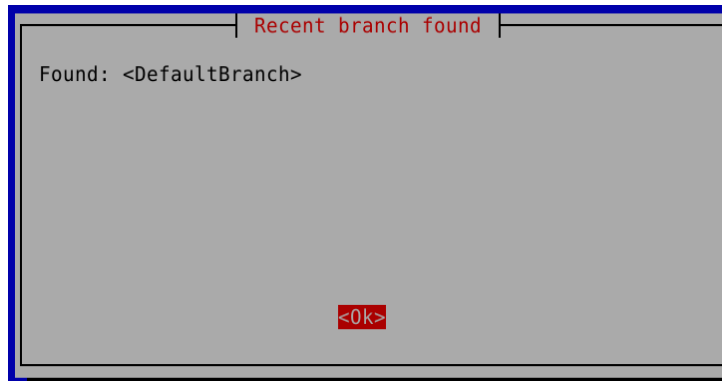


Abbildung 32.37.: Zeigt aktuellen Git-Zweig

```
162  <DebianBranchName3 162>≡ (161b)
      # Ignore HEAD
      if echo ${element} | grep --quiet 'HEAD'
      then continue
      fi
      # Checkout all branches
      if echo ${element} | grep --quiet '^remotes/salsa/'
      then
          NewBranchName=$(echo ${element} | \
              sed --expression='s/^remotes\/salsa\/\///')
          git checkout ${NewBranchName}
          whiptail --title "Checkout branch" \
              --msgbox "Checkout of ${NewBranchName}" 15 60
      fi
      done
      set -e
      <DebianBranchName4 163a>
```

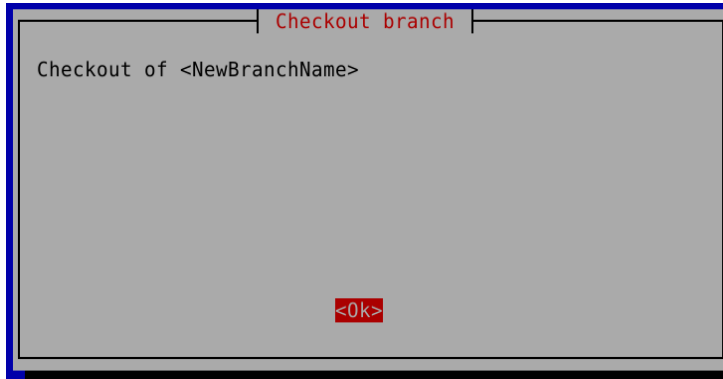



Abbildung 32.38.: Aktueller Git-Zweig

163a $\langle DebianBranchName4 \ 163a \rangle \equiv$

(162)

```
# Finally checkout the default branch (again)
git checkout ${DefaultBranch}
whiptail --title "Checkout branch" \
--msgbox "Checkout of ${DefaultBranch}" 15 60
 $\langle DebianBranchName5 \ 163b \rangle$ 
```

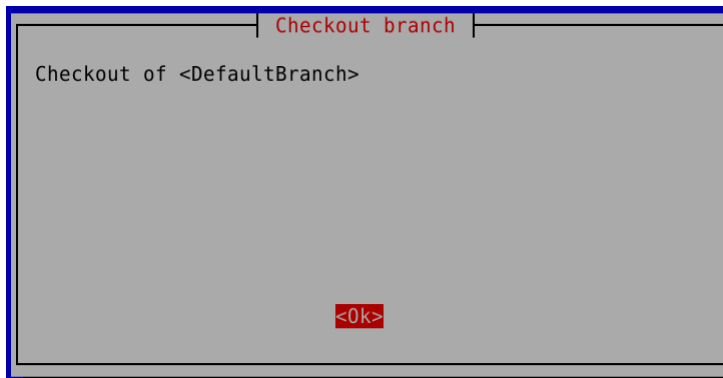


Abbildung 32.39.: Vorgegebener Git-Zweig

163b $\langle DebianBranchName5 \ 163b \rangle \equiv$

(163a)

```
# Insert default branch into the config file and write into logfile
echo 'DefaultBranch=${DefaultBranch} >> ${ConfigPath}${OrigName}'
echo "Branches: "${bl} >> ${log}
echo "DefaultBranch: "${DefaultBranch} >> ${log}
RecentBranch=${DefaultBranch}
whiptail --title "Please check! (1)" \
--msgbox "The branch is ${RecentBranch}" 15 60
 $\langle DebianBranchName6 \ 164a \rangle$ 
```

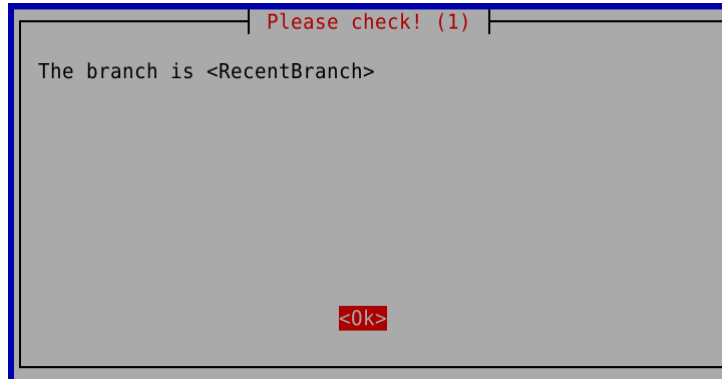


Abbildung 32.40.: Aktueller Git-Zweig

```
164a  <DebianBranchName6 164a>≡ (163b)
      echo 'RecentBranch=${RecentBranch} >> ${ConfigPath}${OrigName}
      echo "RecentBranch: "${RecentBranch} >> ${log}
      Distro4Branch
    }
```

<FailureNotice 184>

Durch den Aufruf der Funktion *Distro4Branch* wird schließlich dem Git-Zweig eine Distribution zugeordnet (Kapitel 32.5.3, Seite 165)

32.5.2. Git-Zweige ermitteln

Diese Funktion ermittelt alle vorhandenen Zweige. Dies geschieht mittels des Befehles *git branch --all*. Bei der Kennzeichnung des aktiven Zweiges werden Sternchen (*) und Leerzeichen durch kleines X (x) und Unterstrich(_) ersetzt.

Diese Funktion wird an verschiedenen Stellen im Ablauf des Programmskriptes aufgerufen.

```
164b  <IdentifyBranches 164b>≡ (159c)
      function IdentifyBranches {
          # Called by DebianBranchName AskDist DebianBranches
          cd ${GitPath}
          # sed is used to kill the asterisk
          bl=$(git branch --all | sed --expression='s/* /x_/')
      }
```

<CreateNewCow 343>

32.5.3. Git-Zweig Distribution zuordnen

An verschiedenen Stellen im Programmablauf ist es erforderlich, einem Git-Zweig eine Debian-Veröffentlichung zuzuordnen.

```

165 <Distro4Branch 165>≡ (344)
function Distro4Branch {
    # Called by DebianBranchName CreateNewBranch AskDist BuildNewRevision

    set +e
    # Set Debian distribution for branch
    if [ ${RecentBranch} != "debian/sid" -o -z "${RecentBranchD}" ]
    then
        CowL=$(ls /var/cache/pbuilder/ | grep .cow | grep '-' | \
            sed 's/base-//' | sed 's/.cow//')
        CowA=($CowL)

        i=1; cowE="0 sid on "
        for element in ${CowA[*]}
        do
            cowE=$cowE' '$i' '${element}' off '
            i=$(expr $i + 1)
        done

        RecentBranchDNr=$(whiptail --title "Debian release" \
            --radiolist "Select pbuilder cow:" \
            --cancel-button "Other" 15 60 8 \
            $cowE 3>&2 2>&1 1>&3)
    <Distro4Branch2 166a>

```

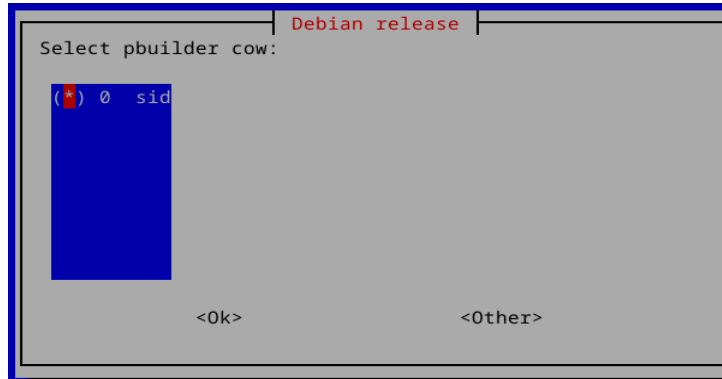


Abbildung 32.41.: Auswahl des Debian Release.

```

166a  <Distro4Branch2 166a>≡ (165)
      if [ $? -eq 1 ]
      then
          CreateNewCow
      fi

      if [ ${RecentBranchDNr} -eq 0 ]
      then
          bDist="sid"
      else
          RecentBranchDNr=$(expr ${RecentBranchDNr} - 1)
          bDist=${CowA[${RecentBranchDNr}]}
      fi

      echo "# ${RecentBranch}_Dist=${bDist} >> ${ConfigPath}${OrigName}
RecentBranchD=${bDist}
      echo "Notice from Distro4Branch: The distribution is " \
${RecentBranchD} >> ${log}

      fi
      set -e
  }

  <DebianBranchName 161a>

```

32.5.4. Name und E-Mail-Adresse hinzufügen

Besonders für den Fall, dass es sich um kein „eigenes“ Salsa-Repository handelt, wird die Möglichkeit eröffnet, den Namen und die E-Mail-Adresse des Maintainers in die lokale git-Konfigurationsdatei einzutragen. Dies erfolgt mit der Funktion *AddNameAndEmail* (s. a, Kapitel 32.4.2, Seite 145

```

166b  <CloneFromSalsa3 166b>≡ (160b)
      if whiptail --title "Name and email" \
      --yesno "Do you like to add your name and email address \n \
to the local git config file?" --yes-button "Yes" \
      --no-button "No" 15 60
      <CloneFromSalsa5 167a>

```

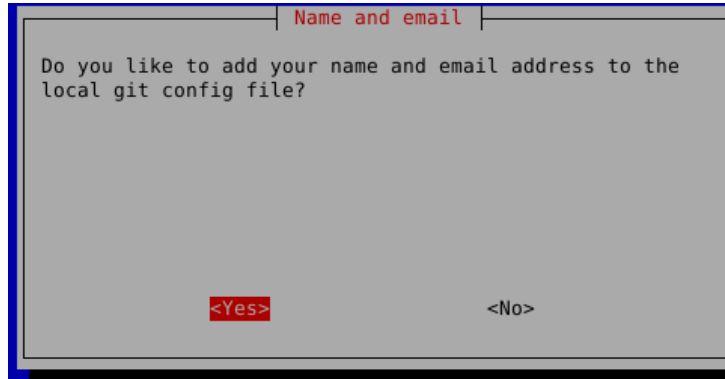


Abbildung 32.42.: Name und E-Mail.

```

167a  <CloneFromSalsa5 167a>≡                               (166b)
      then
        AddNameAndEmail
      fi

      <CloneFromSalsa7 167b>

167b  <CloneFromSalsa7 167b>≡                               (167a)
      AddHomeServer
      PQImport 1
      CommonTasks
    }

      <CutSuffix 212b>

```

32.6. Import eines Debian-Quellcode-Paketes

Nicht alle Pakete des Debian-Projektes liegen auf *salsa.debian.org*, einer Gitlab-Instanz. Von den Paketen auf *salsa.debian.org* werden nicht alle mit *git-buildpackage* gebaut. Manchmal fehlt der Zweig *pristine-tar*. In diesem Fall ist ein Klonen nicht sinnvoll.

Auch diese Pakete können jedoch mithilfe dieses Programmskriptes gebaut werden. In der Datei */etc/apt/sources.list* oder in einer Datei im Verzeichnis */etc/apt/sources.list.d* muss dazu der Eintrag

```
deb-src http://deb.debian.org/debian/ sid main
```

aktiviert vorhanden sein. Dies bedeutet, dass nach der Aktivierung auch *sudo apt update* ausgeführt werden muss.

Für seltene Sonderfälle ist diese Zeile entsprechend anzupassen.

Dieser Eintrag ermöglicht das Herunterladen von Quellpaketen.

```
168a <ImportDebianPackage 168a>≡ (369b)
function ImportDebianPackage {
    # Called by TaskSelect

    echo "Import an existing package without pristine-tar" >> ${log}
    cd ${PrjPath}
    echo "Download ${SourceName} with apt source" >> ${log}
    apt source --download-only ${SourceName} >> ${log}
}
```

```
<ImportDebianPackage2 168b>
```

Mit *apt source --download-only* wird nur das Quellcodepaket heruntergeladen.

Das Erstellen des Git-Verzeichnisses erfolgt mit *gbp import-dsc* und der heruntergeladenen **.dsc*-Datei.

Da die Funktion *gbp import-dsc* ein Signieren ausführt, wird durch die Funktion *GpgKeyAvailable* (Kapitel 32.8, Seite 172) zunächst abgefragt, ob der GnuPG-Schlüssel verfügbar ist.

```
168b <ImportDebianPackage2 168b>≡ (168a)
GpgKeyAvailable

# GitPath has to be deleted because gbp import-dsc will create it.
rmdir --verbose ${GitPath} >> ${log}

echo "gbp import-dsc" >> ${log}
gbp import-dsc --verbose ${SourceName}*.dsc >> ${log}

cd ${GitPath}
git remote add salsa git@salsa.debian.org:${SalsaName}
<ImportDebianPackage3 169a>
```

Danach erfolgt die Anzeige der vorhandenen Git-Zweige und die Bestimmung des aktiven Zweiges durch Aufruf der Funktion *DebianBranchName* (Kapitel 32.5.1, Seite 161).

Dann wird die Möglichkeit eröffnet, den Namen und die E-Mail-Adresse des Maintainers in die lokale git-Konfigurationsdatei einzutragen. Dies erfolgt mit der Funktion *AddNameAndEmail* (s, a, Kapitel 32.4.2, Seite 145

```
169a <ImportDebianPackage3 169a>≡ (168b)
      # Identify branches and choose one
      DebianBranchName

      if whiptail --title "Name and email" \
        --yesno "Do you like to add your name and email address \n \
        to the local git config file?" --yes-button "Yes" \
        --no-button "No" 15 60
    <ImportDebianPackage3a 169b>
```

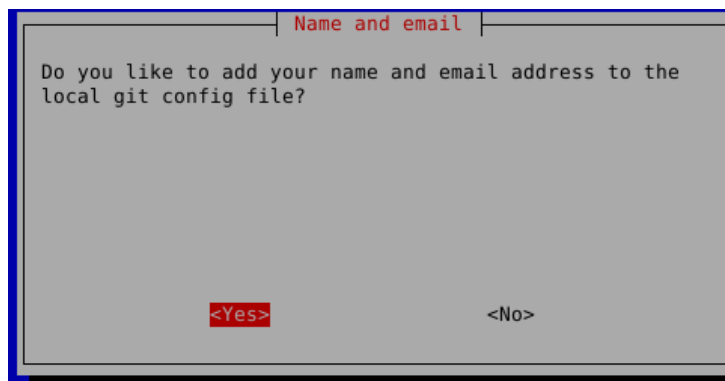


Abbildung 32.43.: Name und E-Mail.

```
169b <ImportDebianPackage3a 169b>≡ (169a)
      then
        AddNameAndEmail
      fi

    <ImportDebianPackage4 169c>

169c <ImportDebianPackage4 169c>≡ (169b)
      AddHomeServer
      PQImport
      CommonTasks

    }

    <Import4Sponsoring 170a>
```

32.7. Import für das Sponsoring

Ein spezieller Fall für die Erstellung eines Git-Repositoriums mit *gpb import-dsc* ist das Sponsoring. Das Herunterladen der Quellcodepakete erfolgt hier mit *dget* (in der Regel von *mentors.debian.net*).

```
170a <Import4Sponsoring 170a>≡ (169c)
function Import4Sponsoring {
    # Called by StartTasks

    cd ${PrjPath}

    if whiptail --title "Should the Debian sources be downloaded?" \
        --yesno "Should the Debian sources (*.orig.tar.gz, *.debian.tar.gz\n \
        and *.dsc) be downloaded?" \
        --defaultno --yes-button "Yes" --no-button "No" 15 60
    <Import4Sponsoring1 170b>
```

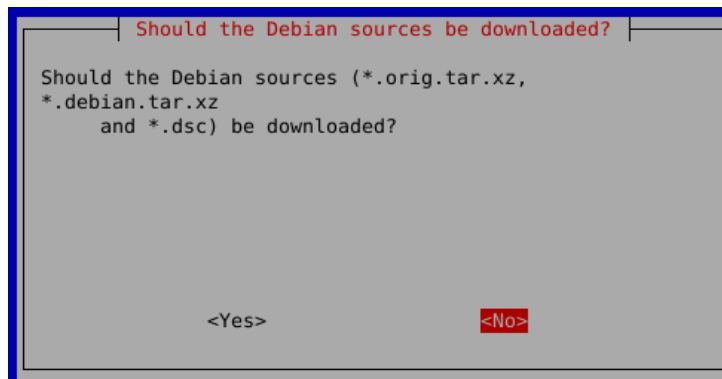


Abbildung 32.44.: Herunterladen der Debian Sourcen

Sollen die Quellcodepakete von *mentors.debian.net* heruntergeladen werden, hat die URL folgendes Format:

```
https://mentors.debian.net/debian/pool/main/<p> \
/<package name>/<package name>_x.y.z.dsc
```

```
170b <Import4Sponsoring1 170b>≡ (170a)
    then
        DownloadUrl=$(whiptail --title "Insert URL for download" \
            --inputbox "Please insert the complete\n \
            URL to download ${UpstreamSourceName}\n(with 'https://'\n \
            e.g. from mentors.debian.net):" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)

    <Import4Sponsoring2 171>
```

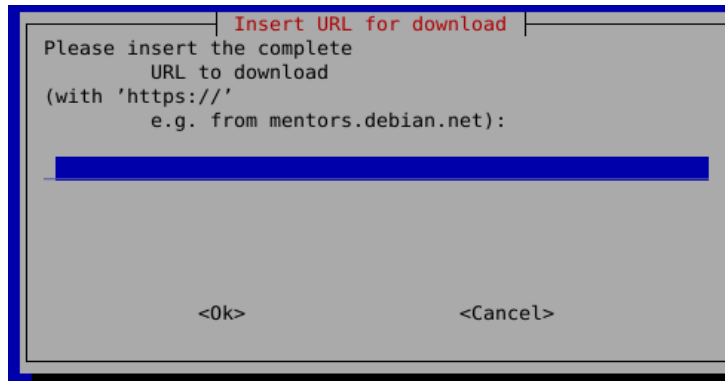



Abbildung 32.45.: Eingabe der Url der Debian Sourcen

```

171  <Import4Sponsoring2 171>≡ (170b)
      dget --download-only ${DownloadUrl}
      ls -la
      echo -e "\n Press RETURN to continue!"
      read a
    fi

    # GitPath has to be deleted because gbp import-dsc will create it.
    echo $(pwd) >> ${log}
    rmdir ${GitPath}

    GpgKeyAvailable
    gbp import-dsc --verbose $(ls *.dsc) >> ${log} &&

    ParseConfig
    cd ${SourceName}
    echo $(pwd) >> ${log}
    SBuildOrPBuilder 0
    Task=5 # Go to RunningTests
}

<CommonTasks 191>

```

Nach dem Bauen des Paketes werden die Tests aufgerufen (Kapitel 40, Seite 357).

32.8. GnuPG-Schlüssel verfügbar?

Die Funktion *GpgKeyAvailable* fordert den Nutzer auf zu prüfen, ob der GnuPG-Schlüssel zum Signieren verfügbar ist.

Sie wird an verschiedenen Stellen im Programmskript aufgerufen. Wird die Frage nach der Verfügbarkeit des GnuPG-Schlüssels verneint, wird das Programmskript beendet.

```
172 <GpgKeyAvailable 172>≡ (181b)
    function GpgKeyAvailable {
        # Called by BuildWithUscan Import2Git CreateSignature ImportDebianPackage
        # PrepareUploading

        if ! whiptail --title "GPG-Key available?" \
            --yesno "GPG-Key must be available for signing." \
            --yes-button "Yes, is available" --no-button "Exit" 15 60
        then
            exit
        fi
        GettingFingerprint
    }

<DetectPlugins 130>
```

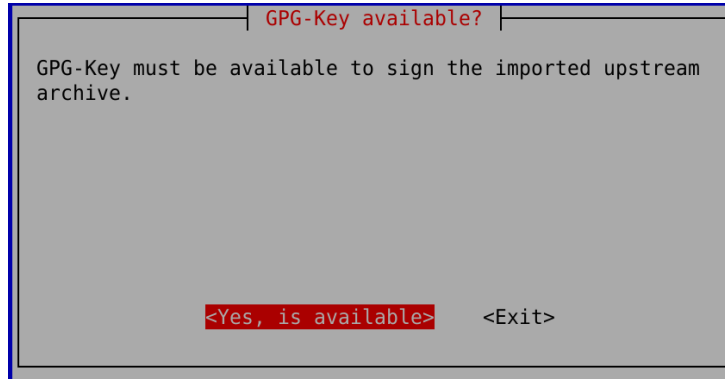


Abbildung 32.46.: GPG-Schlüssel verfügbar

32.9. Fingerprint nutzen

Zum Signieren wird der Fingerprint des Maintainer-Schlüssels benötigt. Dies ist der Fingerprint des Schlüssels, der auch im `Debian-Keyring` hinterlegt ist.

Befindet sich keine entsprechende Datei im Verzeichnis der Konfigurationsdateien, wird eine entsprechende Datei erstellt.

```
173 <GettingFingerprint 173>≡
function GettingFingerprint {
    # Called by GpgKeyAvailable

    finchflag=0

    # getting the fingerprint of the key to sign
    if [ -f ${ConfigPath}/fingerprint ]
    then
        . ${ConfigPath}/fingerprint
    else
        mkdir --parents ${ConfigPath}
        finchflag=1
    fi

    if [ -z "${fipr}" ]
    then
        fipr=$(whiptail --title "Your fingerprint" \
            --inputbox "Please insert fingerprint of your key for signing!" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    <GettingFingerprint1 174a>
```

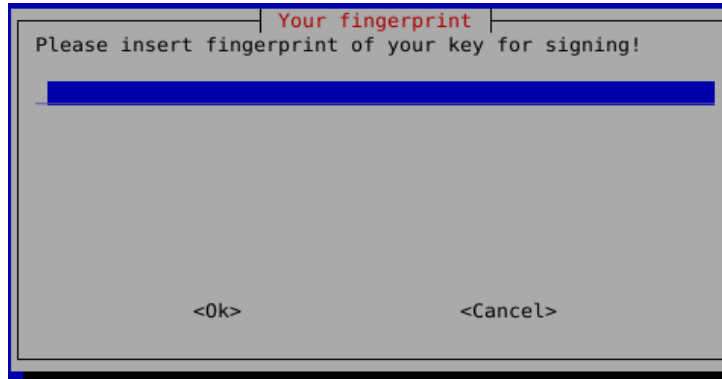


Abbildung 32.47.: Fingerprint eingeben

```
174a  <GettingFingerprint1 174a>≡ (173)
      if [ -z "${fipr}" ]
      then
        echo "Please insert fingerprint of your key for signing!"
        read fipr
      fi
      finchflag=1
    fi

    if ! whiptail --title "Fingerprint" \
      --yesno "Is ${fipr} the right fingerprint of the key for signing?" \
      --yes-button "Yes" --no-button "No" 15 60
    <GettingFingerprint2 174b>
```

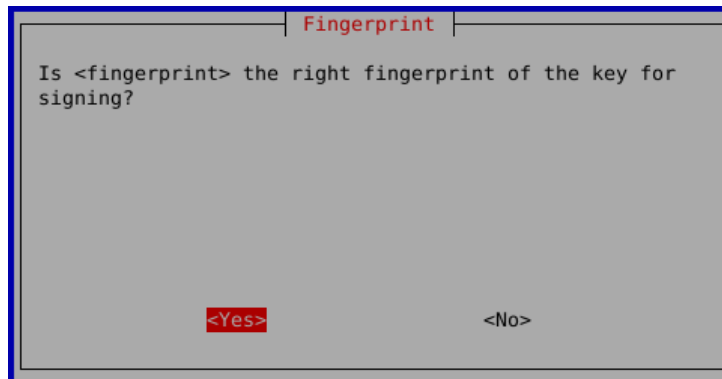


Abbildung 32.48.: Ist der Fingerprint korrekt?

```
174b  <GettingFingerprint2 174b>≡ (174a)
      then
        fipr=$(whiptail --title "Key for signing" \
          --inputbox "Real fingerprint of the key for signing:" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    <GettingFingerprint3 175>
```

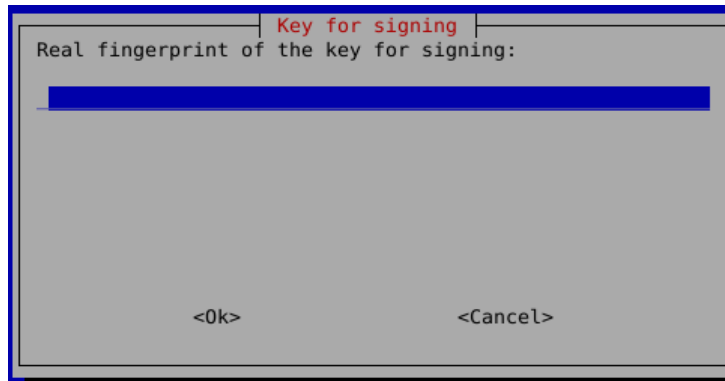


Abbildung 32.49.: Korrekten Fingerprint eingeben

```

175  <GettingFingerprint3 175>≡ (174b)
      if [ -z "${fipr}" ]
      then
          echo "Please insert fingerprint of your key for signing!"
          read fipr
      fi
      finchflag=1
  fi

  if [ $finchflag -eq 1 ]
  then
      if [ -f ${ConfigPath}/fingerprint ]
      then
          mv ${ConfigPath}/fingerprint ${ConfigPath}/fingerprint.backup
          whiptail --title "Fingerprint file" \
            --msgbox "You can find the old fingerprint file as\n \
            ${ConfigPath}/fingerprint.backup" 15 60
      fi
  fi
<GettingFingerprint4 176a>

```

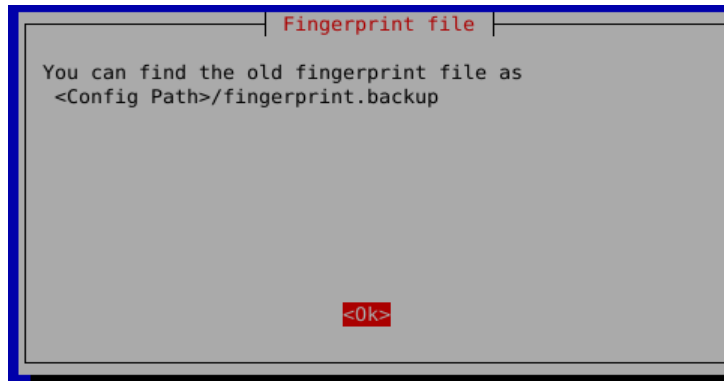


Abbildung 32.50.: Backup des Fingerprints anlegen

```
176a  <GettingFingerprint4 176a>≡ (175)
      touch ${ConfigPath}/fingerprint
      echo "#!/usr/bin/bash" >> ${ConfigPath}/fingerprint
      echo "fipr=${fipr}" >> ${ConfigPath}/fingerprint
      . ${ConfigPath}/fingerprint
      fi
    }

    <CreateSignature (nicht definiert)>
```

32.10. Start des Paketierens

Die Funktion *BuildApp* ruft am Ende die Aufgabenauswahl für die einzelnen Schritte des Paketierens auf.

```
176b  <BuildApp10 176b>≡ (140b)
      set -e
      # Start of Packaging
      CommonTasks
    }

    <MainProgram 112a>
```

33. Arbeiten in einem angelegten Projekt

33.1. Konfigurationsdatei laden und editieren

Wird ein Projektname eingegeben, versucht das Programmskript die Konfigurationsdatei dieses Projektes zu laden und mit *less* anzuzeigen.

Die Funktion *ConfigFileLEC* zeigt die geladene Konfigurationsdatei an und fragt dann ab, ob sie korrekt sei. Dann kann entschieden werden, ob diese Datei editiert werden soll.

```
177 <ConfigFileLEC1 177>≡ (117a)
    set +e
    if [ -f ${ConfigPath}${OrigName} ]
    then
        . ${ConfigPath}${OrigName} # executes config script
        less --LINE-NUMBERS ${ConfigPath}${OrigName}
        if ! whiptail --title "Check config file" \
            --yesno "Is the config file OK?" \
            --yes-button "Yes" --no-button "No" 15 60
    <ConfigFileLEC2 178>
```

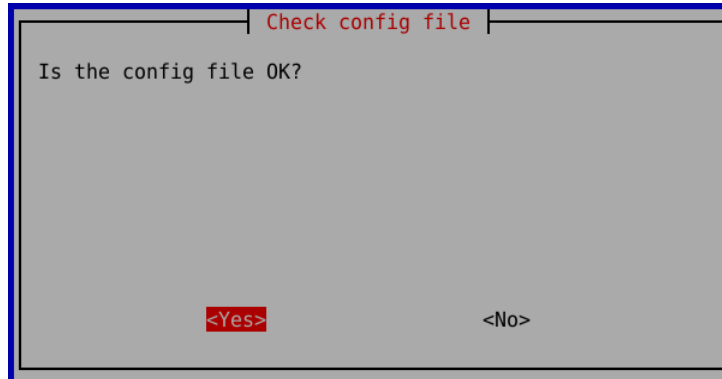


Abbildung 33.1.: Abfrage - Konfigurationsdatei.

Wird die Frage, ob die Konfigurationsdatei korrekt ist, verneint, kann sie editiert werden. Dies kann mit dem Editor *nano* geschehen oder durch die Beantwortung von Fragen.

Ansonsten geht es weiter mit Kapitel 33.4, Seite 182. Wenn jedoch nur ein oder kein Branch existiert, geht es mit Kapitel 33.4.7, Seite 189 weiter.

```

178 <ConfigFileLEC2 178>≡ (177)
    then
        Edit=$(whiptail --title "Edit Config File" \
            --radiolist "How do you like to edit the config file?" \
            15 60 2 "0" "Using Nano" on \
            "1" "Answering questions" off \
            --cancel-button "Exit" 3>&2 2>&1 1>&3)
    <ConfigFileLEC3 179>
  
```

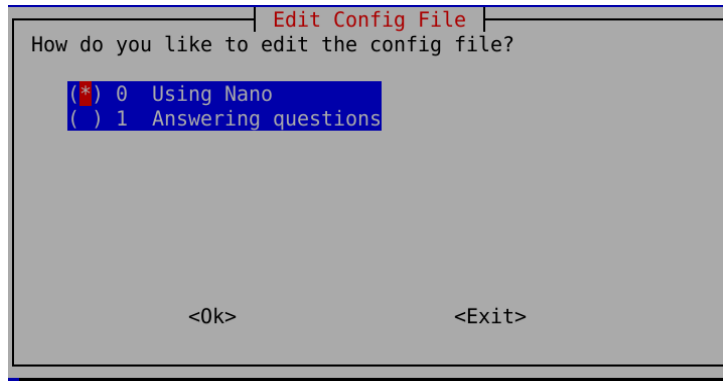



Abbildung 33.2.: Abfrage - Konfigurationsdatei bearbeiten.

Wird keine Editier-Methode angegeben, beendet sich das Programm.

179 $\langle \text{ConfigFileLEC3 } 179 \rangle \equiv$

(178)

```

if [ -z "${Edit}" ]
then
    whiptail --title "Bye" --msgbox "Bye" 15 60
    exit
fi
if [ ${Edit} -eq 0 ]
then
    nano --linenumbers --mouse \
        --softwrap ${ConfigPath}${OrigName}
        . ${ConfigPath}${OrigName}
else
    AskConfig
fi
fi
. ${ConfigPath}${OrigName} # executes config script (another time)
SetDefaults

```

$\langle \text{ConfigFileLEC4 } 117b \rangle$

Soll die Konfigurationsdatei mit dem Editor *nano* bearbeitet werden, wird dieser aufgerufen und öffnet sich. Nach der Beendigung desselben (mit *STRG-X*) läuft das Programmskript weiter (Kapitel 33.4, Seite 182).

Soll die Anpassung mittels Abfragen erfolgen, wird die Funktion *AskConfig* (s. Kapitel 32.1.1, Seite 119) aufgerufen, Danach wird die Datei neu geschrieben.

Enthält die Konfigurationsdatei Daten hinsichtlich der Plugins nicht, so werden mit der folgenden Funktion Standardwerte gesetzt.

```

180 <SetDefaults 180>≡
function SetDefaults {
    # Called by ConfigfileLEC

    if [ -z "${WebextFlag}" ]
    then
        WebextFlag=0
    fi

    if [ -z "${PythonFlag}" ]
    then
        PythonFlag=0
    fi

    if [ -z "${JavaFlag}" ]
    then
        JavaFlag=0
    fi

    if [ -z "${MavenPluginFlag}" ]
    then
        MavenPluginFlag=0
    fi
}
<ConfigFileLEC 117a>
(137b)

```

33.2. Ändern von Zeilen in der Konfigurationsdatei

Im Skript ist auch eine Funktion enthalten, die es ermöglicht, einzelne Werte in der Konfigurationsdatei zu ändern. Hierzu werden der Funktion zwei Parameter übergeben, nämlich als Erstes der Bezeichner der Variablen und zweitens der neue Wert derselben.

Diese Funktion wird von den Funktionen *BuildNewVersion* (Kapitel 34.4.2, Seite 206), *PQMigration* (Kapitel 36.1, Seite 288), *ChangeEntry* (Kapitel 33.4.5, Seite 187) und *OwnServer* (Kapitel 44.2, Seite 408) genutzt.

```
181a <ReplaceConfigLines 181a>≡ (182a)
function ReplaceConfigLines {
    # Called by BuildNewVersion PQMigration ChangeEntry OwnServer CurSuffix

    # This function needs two parameters:
    # First the name of the variable
    # and second it's new value
    ConfProp=$1
    ConfVal=$2
```

<ReplaceConfigLines1 181b>

Die Funktion prüft, ob die zu ändernde Zeile in der Konfigurationsdatei vorhanden ist. Andernfalls wird sie mit der Funktion *InsertConfigLines* erstellt.

Vor dem Ersetzen mit *sed* sind (ebenfalls mit *sed*) eventuell vorhandene Schrägstriche in der Variablen *ConfVal* zu maskieren.

```
181b <ReplaceConfigLines1 181b>≡ (181a)
    set +e
    cprop=$(grep --count ${ConfProp} ${ConfigPath}${OrigName})
    set -e
    if [ ${cprop} -ge 1 ]
    then
        # Masquerade slashes
        ConfVal=$(echo ${ConfVal} | sed 's/\\/\\\\/g')

        sed --in-place --expression="s/${ConfProp}=.*/${ConfProp}=${ConfVal}/g" \
        ${ConfigPath}${OrigName}
    else
        # InsertConfigLine needs two parameters:
        # name of the variable and new value
        InsertConfigLine ${ConfProp} ${ConfVal}
    fi
}
```

<GpgKeyAvailable 172>

33.3. Zeile in Konfigurationsdatei einfügen

Die Funktion *InsertConfigLine* fügt eine Zeile in die Konfigurationsdatei ein. Ihr sind die gleichen Parameter wie der Funktion *ReplaceConfigLines* zu übergeben.

```
182a <InsertConfigLine 182a>≡
function InsertConfigLine {
    # Called by ReplaceConfigLines

    # This function needs two parameters:
    # First the name of the variable
    # and second it's value

    ConfProp=$1
    ConfVal=$2

    echo "${ConfProp}"="`${ConfVal}` >> ${ConfigPath}${OrigName}
}

<ReplaceConfigLines 181a>
```

33.4. Auswahl eines Git-Zweiges

Damit ein Wechsel des Git-Zweiges möglich ist, wird geprüft, ob im aktuellen Zweig unversionierte Änderungen vorhanden sind. Diese Prüfung erfolgt mit der Funktion *CheckGitStatus* (Kapitel 33.4.1, Seite 183)

Wenn kein Wechsel des Git-Zweiges erforderlich ist, müssen Änderungen im Verzeichnis *debian/* nicht zwingend versioniert werden.

Bereits erfolgte Änderungen am Upstream-Code sind mittels *git restore* zurückzusetzen.

Existieren mehrere Branches, kann ein Branch ausgewählt werden. Existiert nur ein Branch, so wird dieser genannt. Anderfalls wird die Funktion *StartTasks* aufgerufen (Kapitel 32.3, Seite 140).

Diese Funktion kann von der Aufgabenauswahl (Kapitel 33.5, Seite 191) aus aufgerufen werden.

Die Funktion *SelectBranch* wird gegebenenfalls auch beim Bauen einer neuen Revision aufgerufen (Kapitel 37.3.1, Seite 332).

```
182b <SelectBranch 182b>≡ (189a)
function SelectBranch {
    # Called by BuildApp TaskSelect BuildNewRevision

    CheckGitStatus
    DebianBranches
<SelectBranch1 186>
```

Nach der Funktion *CheckGitStatus*, die im folgenden Abschnitt beschrieben wird, wird die Funktion *DebianBranches* zur Ermittlung der lokalen Debian-Branches im Git-Repositorium aufgerufen (Kapitel 33.4.3, Seite 185).

33.4.1. Prüfung mit *git status*

Die Funktion *CheckGitStatus*, die auch noch an anderen Stelle im Programm-Skriptes aufgerufen wird, prüft, ob im aktuellen Git-Zweig neue oder geänderte Dateien vorhanden sind. Ist dies der Fall, können nämlich manche Git-Operationen nicht vorgenommen werden.

Wird dies mittels *git status* festgestellt, wird die Funktion *FailureNotice* aufgerufen und eine Möglichkeit zur Fehlerbehebung in einem weiteren Terminal eröffnet (Kapitel 33.4.2, Seite 184).

Gibt es keine Probleme, geht es weiter mit der Anzeige der vorhandenen Git-Zweige (Kapitel 33.4.4, Seite 186).

```

183 <CheckGitStatus 183>≡ (231)
function CheckGitStatus {
    # Called by Import2Git BuildWithUscan PQImport
    # PatchesTreatment SelectBranch and itself

    set +e
    # Checks git status
    echo "Notice from CheckGitStatus:" >> ${log}
    echo "$(git status) >> ${log}

    if [ ! -z "$(git status --short)" ]
    then
        git status
        FailureNotice "'git status' shows problems\n\
        Please clean up 'git status'"
        if whiptail --title "Check another time?" \
        --yesno "Do you want to check the git status another time?" \
        --yes-button "Yes" --no-button "No" 15 60
        then
            CheckGitStatus
        fi
    fi
    set -e
}

<CheckTags 242a>

```



Abbildung 33.3.: Abfrage - Weiterer Check?.

33.4.2. Fehlermeldung und -behebung

Wird die Funktion *FailureNotice* aufgerufen, erfolgt eine Meldung auf dem Terminal und es wird die Möglichkeit eröffnet, Fehler in einem anderen Terminal zu beheben.

Dieser Funktion kann beim Aufruf ein spezieller Text als Parameter mitgegeben werden. Geschieht dies nicht, wird ein Standardtext ausgegeben.

```

184 <FailureNotice 184>≡ (164a)
function FailureNotice {
    # Called by PQImport CheckGitStatus RebasePQBranch PQMigration
    # You can call this function with a text as parameter (optional)

    if [ ! "$1" ]
    then
        echo "Failure"
        echo "Something went wrong!"
    else
        for i in $*
        do
            String=${String}" "$i
        done

        echo -e ${String}
    fi
    echo
    echo "Break for fixing it in another terminal"
    echo "After fixing press RETURN to go on!"
    read a
}

<PQImport 197>

```

33.4.3. Auswahl der Debian-Banches

Die Funktion *DebianBranches* ermittelt, welche einschlägigen Branches im Git-Repositorium vorhanden sind.

Hierzu wird zunächst die Funktion *IdentifyBranches* aufgerufen (Kapitel 32.5.2, Seite 164).

```
185a <DebianBranches 185a>≡ (187)
function DebianBranches {
    # Called by CreateNewBranch SelectBranch
    # selects the Debian branches
    IdentifyBranches
```

<DebianBranches1 185b>

Im Folgenden wird die Auswahl auf die lokalen Debian-Banches begrenzt.

```
185b <DebianBranches1 185b>≡ (185a)
    ## Trim branchlist
    bl=$(echo $bl | sed 's/pristine-tar/ /')
    bl=$(echo $bl | sed 's/upstream/ /')
    bl=$(echo $bl | sed 's/HEAD/ /')
    # bl=$(echo $bl | sed 's/remotes\./origin\./ /g')
    # bl=$(echo $bl | sed 's/remotes\./salsa\./ /g')
    # bl=$(echo $bl | sed 's/remotes\./home\./ /g')
    bl=$(echo $bl | sed 's/remotes\./ /g')
}
```

<CreateNewBranch (nicht definiert)>

33.4.4. Dialog zur Auswahl eines Branches

Die Anzeige dieses Dialoges erfolgt nur, wenn es mehrere Debian-Branches gibt. Der aktuelle Branch wird vorausgewählt.

Existiert nur ein oder kein Branch, geht es im Kapitel 33.4.7 (Seite 189) weiter.

```

186 <SelectBranch1 186>≡ (182b)
    ## Create a radiolist with the branch names
    ba=($bl)
    i=1; slct=''
    set +e
    for element in ${ba[*]}
    do
        echo ${element} | grep 'x_' > /dev/null
        if [ $? -eq 0 ]
        then
            if [ "${element}" = "x_" ]
            then
                continue
            else
                ostr="on"
            fi
        else
            ostr="off"
        fi
        slct=${slct}' '$i'' "${element}'' '${ostr}' '
        i=$(expr $i + 1)
    done
    set -e

    if [ $#ba[@] -gt 1 ]
    then
        ## select branch
        branch=$(whiptail --title "Branch" --radiolist "Select:" \
        15 60 8 ${slct} --cancel-button "Task selection" 3>&2 2>&1 1>&3)
        if [ ! -z "${branch}" ]
        then
            set +e
            branch=$(expr ${branch} - 1)
            set -e
            bName=${ba[$branch]}
            bName=$(echo ${bName} | sed --expression='s/^x_//')
            ## checkout branch
            git checkout ${bName}
            ## Change config file -
            ## make selected branch to recent one
            ChangeEntry
        <SelectBranch3 188>

```

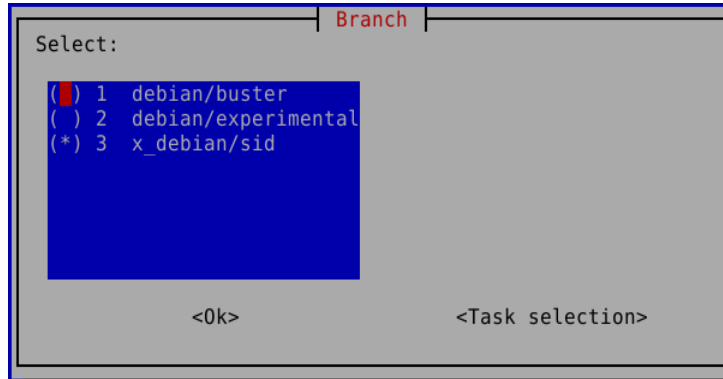



Abbildung 33.4.: Auswahl des Debian Zweiges.

Statt *debian-stable* wird der für diesen Git-Zweig gewählte Name angezeigt.

Ist dies nicht der Fall, wird mit dem Klick auf den Button *Task selection* die Aufgabenwahl ausgewählt (Kapitel 33.5, Seite 191). Dort kann mit *Exit* auch das Programm verlassen werden. Mit *Create new branch* kann dort auch ein neuer Git-Zweig angelegt werden.

Mit *Ok* wird die Konfigurationsdatei nochmals mit *less* angezeigt.

33.4.5. Eintrag ändern

187

```

<ChangeEntry 187>≡
function ChangeEntry {
    # Called by CreateNewBranch SelectBranch
    set +e
    RecentBranchEntry=$(grep --count 'RecentBranch=' ${ConfigPath}${OrigName})
    set -e

    ## Change RecentBranch entry in config file
    if [ ${RecentBranchEntry} -eq 0 ]
    then
        echo "RecentBranch=${bName} >> ${ConfigPath}${OrigName}"
    else
        # ReplaceConfigLines needs two parameters:
        # name of the variable and new value
        ReplaceConfigLines 'RecentBranch' ${bName}
        # bName1=$(echo ${bName} | sed --expression='s/\\/\\/\\/\\/g')
        # sed --in-place --expression=\
        # "s/RecentBranch=.*\/RecentBranch=${bName1}/g" \
        # ${ConfigPath}${OrigName}
    fi
    less --LINE-NUMBERS ${ConfigPath}${OrigName}

    ## Set variable
    RecentBranch=${bName}
    echo "Notice from ChangeEntry: The branch is "${RecentBranch} >> ${log}
}

```

13. Oktober 2024

<DebianBranches 185a>

```
188  <SelectBranch3 188>≡ (186)
      whiptail --title "This branch was selected" \
      --msgbox "${bName} was selected" 15 60
      echo "${bName} was selected" >> ${log}
      ParseConfig
      fi
<SelectBranch4 189b>
```

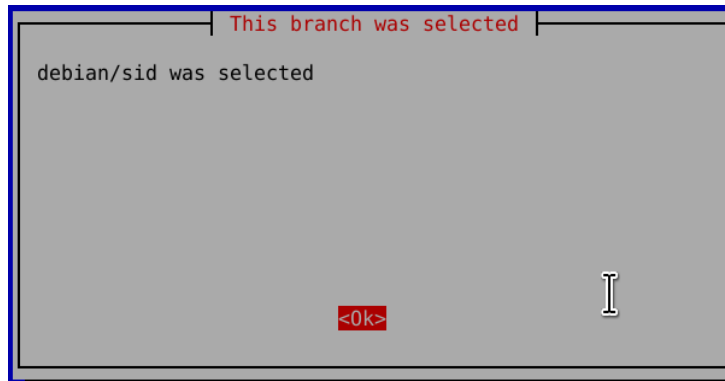


Abbildung 33.5.: Ausgewählter Debian Zweig.

In der Nutzeroberfläche geht es weiter mit der Aufgabenauswahl (Kapitel 33.5, Seite 191).

33.4.6. Konfiguration einlesen

Diese Funktion ordnet dem ausgewählten Git-Zweig die Debian-Distribution zu, für die das Paket gebaut wird. (Kapitel 37.3.4, Seite 337)

```
189a <ParseConfig 189a>≡
function ParseConfig {
    # Called by SelectBranch TaskSelect

    # Parse config file for Debian distribution of branch
    set +e
    vc=$(grep --count ${bName}_Dist ${ConfigPath}${OrigName})
    set -e
    if [ $vc -ge 1 ]
    then
        Search4Dist
    else
        va="sid"
    fi
    RecentBranchD=${va}
    echo "Notice from ParseConfig: The distribution is "${RecentBranchD} >> ${log}
}

<SelectBranch 182b>
```

33.4.7. Kein oder nur ein Branch existiert

```
189b <SelectBranch4 189b>≡ (188)
    elif [ ${#ba[0]} -eq 1 ]
    then
        whiptail --title "Only one branch" \
        --msgbox "There is only one Debian branch: ${ba[0]}" 15 60
<SelectBranch6 190>
```

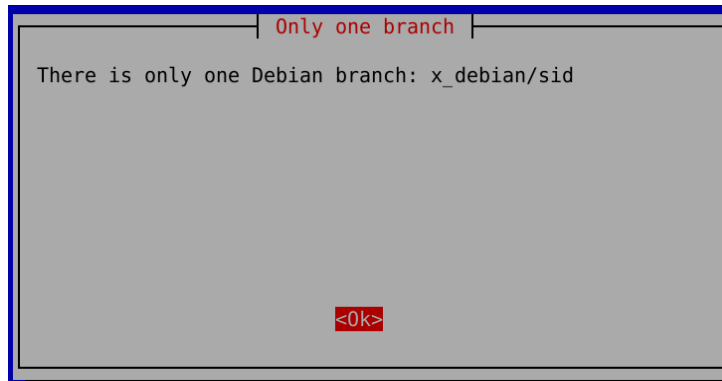


Abbildung 33.6.: Es gibt nur einen Git-Zweig

Gibt es einen Git-Zweig geht es mit der Aufgabenauswahl weiter (Kapitel 33.5, Seite 191). Andernfalls erfolgt ein Hinweis.

```
190 <SelectBranch6 190>≡ (189b)
    else
        whiptail --title "There is no branch" \
            --msgbox "There is no branch created.\nPlease build a new version." 15 60
    fi
}

<AddGitServer (nicht definiert)>
```

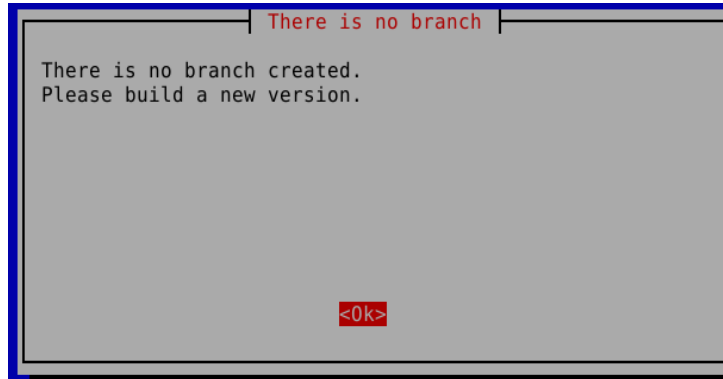


Abbildung 33.7.: Kein Zweig erstellt

33.5. Aufgabenauswahl

Nachdem die Konfigurationsdatei geladen und geprüft oder erstellt wurde, erscheint ein Menü zur Aufgabenauswahl.

Da das Programm modular aufgebaut ist, können die Aufgaben auch einzeln ausgewählt werden. Der Bauprozess kann abgebrochen und später wieder aufgenommen werden.

```

191 <CommonTasks 191>≡ (171)
    function CommonTasks {
        # Called by BuildApp TaskSelect

        Task=$(whiptail --title "Tasks:" \
            --radiolist "What do you like to do? " 17 60 9 \
                "2" "Build a new version of a package" off \
                "3" "Build a new debian revision" on \
                "4" "Rebuilding a revision" off \
                "5" "Running lintian and uscan" off \
                "6" "Uploading only (build again if necessary)" off \
                "7" "Create new branch" off \
                "8" "Select branch" off \
                "9" "Set name or IP of own git server" off \
                "10" "Create a debdiff" off \
            --cancel-button "Exit" 3>&2 2>&1 1>&3)

        if [ -z "${Task}" ]
        then
            exit
        fi
        TaskSelect
    }

    <AskOrigName 114a>

```

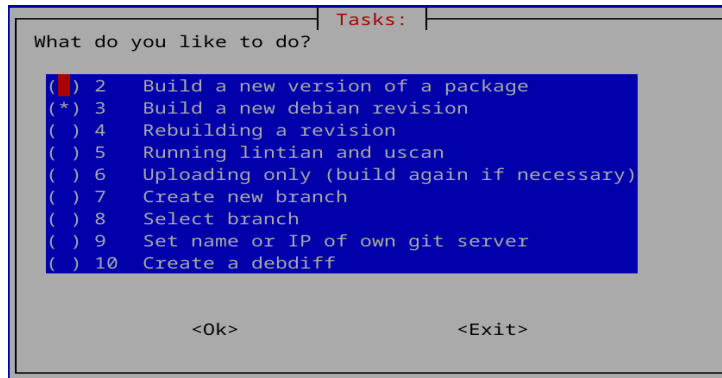


Abbildung 33.8.: Aufgabenauswahl.

Zur Auswahl stehen zunächst der Bau einer neuen Version (Kapitel 34, Seite 195) und einer neuen Revision. Als *default* geht es mit dem Bauen einer neuen **Debian**-Revision (Kapitel 35, Seite 253) weiter.

Soll eine neue Version gebaut werden, wird gefragt, ob zunächst Änderungen von *salsa.debian.org* heruntergeladen werden sollen (Kapitel 34.1, Seite 195). Danach prüft das Programmskript mit dem Aufruf der Funktion *ClassicalOrUscan*, ob es bereits Patches gibt. In einem solchen Fall können diese Patches in einen separaten Git-Branch als *patch-queue* importiert werden (s. Kapitel 34.2, Seite 197).

In der Funktion *ClassicalOrUscan* (Kapitel 34.3, Seite 202) wird abgefragt, auf welche Weise das Herunterladen des Upstream-Quellcodes erfolgen.

Nach dem Bauen können die Pakete geprüft werden (Kapitel 40, Seite 357).

Der nächste Menüpunkt betrifft das Hochladen (Kapitel 41, Seite 373).

Ferner kann auch ein neuer **Git**-Zweig angelegt werden, beispielsweise auch für Backports, gearbeitet wird (Kapitel 44.1, Seite 407).

Sind mehrere **Git**-Zweige vorhanden, kann einer von Ihnen ausgewählt werden (Kapitel 33.4, Seite 182).

Schließlich kann man auch einen eigenen **Git**-Server in den Arbeitsablauf einbinden (Kapitel 44.2, Seite 408).

Die folgenden Zeilen des Programmskripts dienen der Ablaufsteuerung. Eine aufgerufene Funktion ändert an ihrem Ende die Variable *Task*, so dass eine der folgenden *if*-Klauseln zutrifft. Aus diesem Grund können die folgenden Zeilen nicht durch eine *Case*-Anweisung ersetzt werden.

Die Kette von *If*-Anweisungen gibt die Möglichkeit, eine Bedingung nach der anderen abzuarbeiten, da die aufgerufene Funktion an das Ende der *If*-Anweisung zurückkehrt, von der sie aufgerufen wurde. Eine *Case*-Anweisung wird mit dem Aufruf der abzuarbeitenden Funktion beendet.

Der Aufruf der entsprechenden Funktionen erfolgt in der genannten Weise wiederum durch die Funktion *TaskSelect*.

Mit dem folgenden Aufruf wird das Bauen einer neuen Version aufgerufen (Kapitel 34.1, Seite 195).

```
192 <TaskSelect3 192>≡ (143c)
    ## Common tasks
    rcts=0 # ReCall TaskSelect flag
```

```

# Building a new version
if [ $Task -eq 2 ]
then
    if [ ${NoPull} -eq 0 ]
    then
        PullFromSalsa
    fi
    ClassicalOrUscan # and then download new version
fi

```

⟨TaskSelect4 193a⟩

Oder es geht weiter mit dem Bauen einer neuen Debian-Revision (Kapitel 35, Seite 253). Dies entspricht dem voreingestellten Wert.

193a ⟨TaskSelect4 193a⟩≡ (192)

```

# Building a new revision
if [ $Task -eq 3 ]
then
    BuildNewRevision
fi

# Rebuilding a revision
if [ $Task -eq 4 ]
then
    ParseConfig
    AskDist
    SBuildOrPBuilder 0
    Task=5 # Running Tests
fi

```

⟨TaskSelect5 193b⟩

Nach dem Bauen einer neuen Revision ist diese zu testen. (Kapitel 40, Seite 357) weiter. Dies geschieht mittels *lintian* (Kapitel 40.3, Seite 360) und *uscan* (Kapitel 40.4, Seite 362).

193b ⟨TaskSelect5 193b⟩≡ (193a)

```

# Running lintian and uscan
if [ $Task -eq 5 ]
then
    RunningTests
fi

```

⟨TaskSelect6 194a⟩

Mit dem nächsten Schritt wird die Funktion *PrepareUploading* aufgerufen (Kapitel 41, Seite 373). Darin wird zunächst überprüft, ob die Datei *debian/changelog* schon für eine Veröffentlichung vorbereitet ist. Andernfalls wird dies noch durchgeführt.

```
194a <TaskSelect6 194a>≡ (193b)
    # Uploading the package
    if [ $Task -eq 6 ]
    then
        PrepareUploading
    fi
```

<TaskSelect7 194b>

Der folgende Abschnitt ist notwendig, um für eine abweichende Distribution zu bauen. Dies bezieht sich auf Debian-Backports, Debian-Proposed-Updates für Stable und eventuell für Oldstable. Gegebenenfalls wird dies auch benötigt, um für einen Ubuntu-Zweig ein Paket bereitzustellen. (Siehe auch Kapitel 39, Seite 355)

```
194b <TaskSelect7 194b>≡ (194a)
    # Create new branch
    # (e.g. for backports or proposed-updates)
    if [ $Task -eq 7 ]
    then
        CreateNewBranch
        rcts=1
    fi

    # Select branch
    if [ $Task -eq 8 ]
    then
        SelectBranch
        rcts=1
    fi

    # Set name or IP of own git server
    if [ $Task -eq 9 ]
    then
        OwnServer
        rcts=1
    fi

    # Create a debdiff
    if [ $Task -eq 10 ]
    then
        DebDiff 0
        rcts=1
    fi
```

<TaskSelect9 (nicht definiert)>

34. Bauen einer neuen Version

In diesem Kapitel werden die Schritte beschrieben, die zur Erstellung des Debian-Quellcode-Paketes (**.orig.tar.(g/x)z*) (Kapitel 8, Seite 23) führen. Dies erfolgt nach dem Start des Programmskriptes (Kapitel 31.2, Seite 112).

Das Programmskript ermöglicht auf verschiedenen Weisen, den Upstream-Quellcode zu erlangen. Zwei Möglichkeiten wurden bereits beschrieben, nämlich das Klonen von *salsa.debian.org* (Kapitel 32.5, Seite 160) sowie der Import eines Debian-Quellcode-Paketes (Kapitel 32.6, Seite 168)

Zwei weitere Möglichkeiten werden im Folgenden beschrieben (Kapitel 34.3, Seite 202). Nach der Ausführung von *mk-origtargz* zum Erstellen des Debian-Quellcode-Paketes folgt die Integration des Quellcodes in das lokale Git-Repository (Kapitel 34.4.11, Seite 243).

Zunächst wird jedoch die Möglichkeit eröffnet, eventuelle Änderungen im Salsa-Repository des Paketes herunterzuladen, was manchmal sinnvoll ist. Ferner kann aus einer vorhandenen Patch-Queue ein Patch-Queue-Zweig erstellt werden. (Kapitel 34.2, Seite 197).

34.1. Änderungen von *Salsa* herunterladen

Es kann besonders bei team-betreuten Paketen vorkommen, dass andere Team-Mitglieder Änderungen im Git-Repository auf *salsa.debian.org* bereitstellen. Für die weiteren Arbeiten an diesem Repository ist es nun zwingend erforderlich, diese Änderungen auch ins lokale Repository zu übernehmen.

Dies kann erfolgen mit

```
gbp pull --all salsa
```

Wenn dies zu Problemen führt, können die einzelnen Branches auch mit

```
git checkout <BranchName>
git pull salsa <BranchName>
```

aktualisiert werden.

```
195 <PullFromSalsa 195>≡ (205)
function PullFromSalsa {
    # Called by TaskSelect

    cd ${GitPath}
    set +e
    if git remote --verbose | grep --quiet 'salsa'
    then
        if whiptail --title "Pull from Salsa?" \
```

```
--yesno "Do you like to pull possible changes from salsa?" \  
--yes-button "Yes" --no-button "No" 15 60  
<PullFromSalsa1 196>
```

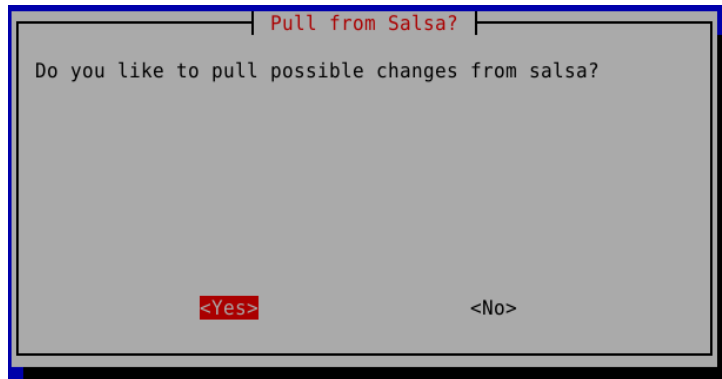


Abbildung 34.1.: Herunterladen von salsa.debian.org

Dazu wird dann noch das Passwort für den SSH-Schlüssel abgefragt, mit dem auf *salsa.debian.org* zugegriffen werden kann.

```
196 <PullFromSalsa1 196>≡ (195)  
    then  
        echo "RecentBranch: "$(git branch) >> ${log}  
        git pull --all  
    fi  
fi  
set -e  
}  
  
<CloneFromSalsa 160a>
```

Danach wird die Funktion *ClassicalOrUscan* (Kapitel 34.3, Seite 202) aufgerufen, welche als Erstes die Funktion *PQImport* zum Importieren einer vorhandenen *Patch-Queue* aufruft.

34.2. Importieren einer vorhandenen *Patch-Queue*

Vor dem Herunterladen der neuen Upstream-Version wird geprüft, ob es für die bisher gepackte Version bereits eine *Patch-Queue* gibt. Mit „*Patch-Queue*“ sind die in der Datei *debian/patches/series* aufgeführten Patches in der dortigen Reihenfolge gemeint.

In diesem Falle wird die Möglichkeit eröffnet, diese mittels *gbp pq import* in einen eigenen *Patch-Queue-Branch* zu importieren, sofern dieser nicht bereits existiert. Dies erlaubt es, die Patches später – nach dem Import der neuen Version – in diese wieder aufzunehmen.

Beim Import in den Patch-Queue Branch werden die Patches auf den Upstream-Quellcode angewandt.

Voraussetzung dafür ist, dass es im aktuellen Git-Zweig keine unversionierten Dateien gibt und sich alle in der Datei *debian/patches/series* aufgeführten Patches anwenden lassen. Andernfalls schlägt *gbp pq import* fehl. Dann wird kein *patch-queue branch* erzeugt. Die Patches manuell anwendbar zu machen, kann aufwändig sein.

Die genannten Bedingungen sollten aber in der Regel vor dem Import einer neuen Version erfüllt sein.

```

197 <PQImport 197>≡ (184)
function PQImport {
    # Called by ClassicalOrUscan PQMigration CloneFromSalsa and itself
    returnflag=$1
    if [ ! ${returnflag} ]
    then
        returnflag=1
    fi
    cd ${GitPath}
    set +e
    if echo $(git branch) | grep --quiet 'patch-queue/'${RecentBranch}
    # patch-queue branch already exists
    then
        set -e
        return
    fi
    set -e

    if [ -f debian/patches/series ]
    # debian/patches/series exists
    then
        if whiptail --title "There are patches" \
        --yesno "Do you like to import the current patches\n\
        onto the patch-queue branch? (recommended)" \
        --yes-button "Yes" --no-button "No" 15 60
        <PQImport1 198a>

```



Abbildung 34.2.: Es gibt Patches

Wird diese Frage verneint, geht es mit dem Herunterladen des Quellcodes weiter (Kapitel 34.3, Seite 202).

Ansonsten folgt zur Vorbereitung des Importes in einen *Patch-Queue*-Zweig die Prüfung des Git-Status wie in Kapitel 33.4.1 (Seite 183) beschrieben.

```
198a <PQImport1 198a>≡ (197)
      then
          CheckGitStatus
      <PQImport1-1 198b>
```

34.2.1. Erster Importversuch

Sofern kein Patch-Queue-Branch existiert, was der Normalfall sein sollte, wird ein solcher erstellt. Danach erfolgt der Import in diesen Patch-Queue-Zweig. Nach erfolgtem Import wird in den ursprünglichen Zweig (meist *debian/sid*) (zurück-)gewechselt. (Kapitel 34.2.3, Seite 200).

Alle in der Datei *debian/patches/series* aufgeführten Patches müssen anwendbar sein.

```
198b <PQImport1-1 198b>≡ (198a)
      echo "Notice from gbp pq import: " >> ${log}
      gbp pq --verbose import >> ${log} 2>&1
      if [ $? -eq 1 ]
      then
          Notice="All patches listed in debian/patches/series\n\
          have to be applicable"'\n\
          For Details, look into the log file of the project.\n"
          FailureNotice ${Notice}
      <PQImport2 199a>
```

Schlägt der Import fehl, wird eine Möglichkeit zur Fehlerbehebung eröffnet (Kapitel 33.4.2, Seite 184). Danach kann ein erneuter Versuch unternommen werden. Der erfolgreiche Import wird vom Programmskript mitgeteilt (Kapitel 34.2.3, Seite 200).

34.2.2. Erneuter Importversuch

Nach der Fehlerbehebung kann ein erneuter Importversuch unternommen werden. Wird der Versuch der Fehlerbehebung für misslungen erachtet, kann der Import abgebrochen werden.

```
199a <PQImport2 199a>≡ (198b)
      if whiptail --title "Fixed?" --yesno "Retry?" \
        --yes-button "Yes" --no-button "No import" 15 60
        <PQImport3 199b>
```

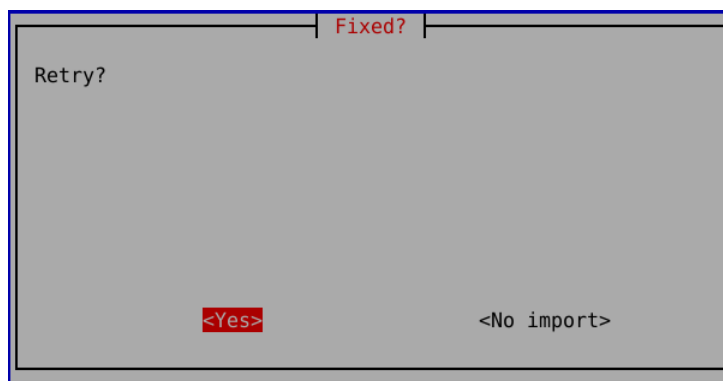


Abbildung 34.3.: Fixed? Retry?

```
199b <PQImport3 199b>≡ (199a)
      then
        PQImport ${returnflag}
      else
        whiptail --title "No import onto a patch-queue branch" \
          --msgbox "Let's go on without the import\n\
            of the current patches onto a patch-queue branch" \
            15 60
      fi
    <PQImport4 200>
```

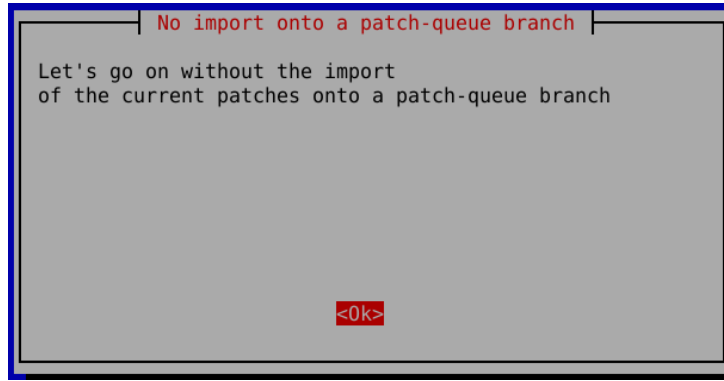


Abbildung 34.4.: Kein Import in Patch-Queue

34.2.3. Import im PQ-Branch erfolgreich

Der erfolgreiche Import in den PQ-Branch wird mit einem Dialog angezeigt.

```
200 <PQImport4 200>≡ (199b)
      else
        whiptail --title "Done" \
          --msgbox "Imported the current patches onto the patch-queue branch" \
            15 60
      <PQImport5 201>
```

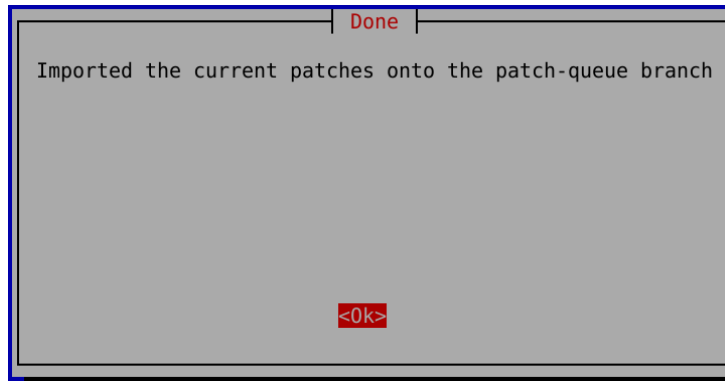


Abbildung 34.5.: PQ-Import erfolgreich

Sodann kehrt das Programmskript in den ursprünglichen Git-Zweig (Debian-Branch) zurück.

```

201  <PQImport5 201>≡ (200)
        if [ ${returnflag} -eq 1 ]
        then
            # Back to the previous branch
            git checkout ${RecentBranch}
        fi
    fi
fi
}

<ClassicalOrUscan 202a>

```

34.3. Werkzeuge zum Herunterladen der Upstream-Sourcen

Zum Bauen eines Debian-Paketes wird zunächst der Quellcode des Upstream-Projektes benötigt. Dieser kann auf verschiedene Arten erlangt werden.

Das Herunterladen des Quellcodes mit *wget* ist die klassische Methode (Kapitel 34.4, Seite 206). Dieser Weg ist beim ersten Bauen eines neuen Paketes zu wählen.

Es kommt vor, dass Upstream kein Quellcode-Archiv zur Verfügung stellt. Stattdessen findet man den Quellcode bei *Github* oder ähnlichen Hostern. Dies stellt besondere Anforderungen an den Betreuer eines Debian-Paketes.

Alleine schon, um *reproducible* bauen zu können, muss ein Tar-Archiv des Upstream-Quellcodes bereitgestellt werden. Dies muss ohne Netzzugriff auf Externe Ressourcen möglich sein.

Diese Methode ist auch zu wählen, wenn ein bestimmter Stand des Upstream-Codes aus einem Git-Repository (z.B. Github o.ä.) verwendet werden soll.

Existieren jedoch bereits eine Datei *debian/watch* und andere Dateien im Verzeichnis *debian/*, kann hierfür auch *uscan* (Kapitel 34.5, Seite 248) eingesetzt werden. Ein heruntergeladen mit *uscan* empfiehlt sich dann, wenn die Datei *debian/watch* einen Eintrag *wversionmangle=* enthält.

Kann *uscan* die neue Version nicht identifizieren, muss die neue Version manuell oder per *wget* bereitgestellt werden. Gleiches gilt, wenn *uscan* die neue Version zwar identifizieren kann, ein Herunterladen aber am Aufbau der Adresse zum Herunterladen scheitert.

Diese Möglichkeiten werden vom Programmskript alternativ angeboten.

Außerdem gibt es noch die Möglichkeit *get-orig-source* in *debian/rules* zu verwenden.

```
202a <ClassicalOrUscan 202a>≡ (201)
function ClassicalOrUscan {
    # Called by PullFromSalsa

    # Before importing a new version, check whether there is a patch-queue,
    # which can be exported onto a patch-queue branch first
    PQImport 1
}
```

<ClassicalOrUscan1 202b>

Die Funktion *PQimport* wird im Kapitel 34.2 (Seite 197) beschrieben. Sie ermöglicht die Erstellung eines Patch-Queue-Zweiges vor dem Herunterladen.

```
202b <ClassicalOrUscan1 202b>≡ (202a)
    if [ ! -f ${GitPath}/debian/watch ]
    then
        BuildNewVersion
    return
<ClassicalOrUscan1-1 203a>
```


Es wird geprüft, ob in der Datei `debian/watch addons.thunderbird.net` oder `addons.mozilla.org` als Quelle aufgeführt werden. Von dort ist nämlich kein Herunterladen mit `uscan` möglich. Denn die Adresse zum Herunterladen enthält dort zusätzlich zur Versionsbezeichnung eine nicht vorhersehbare ID.

203a `<ClassicalOrUscan1-1 203a>`≡ (202b)

```
else
  set +e
  # Download from Mozilla repos is not possible with uscan
  if grep --quiet "addons.thunderbird.net" ${GitPath}/debian/watch
  then
    whiptail --title "Thunderbird Repository" \
      --msgbox "From addons.thunderbird.net \nyou can't download with uscan" \
      15 60
    BuildNewVersion
    set -e
    return
  fi
```

`<ClassicalOrUscan2 203b>`

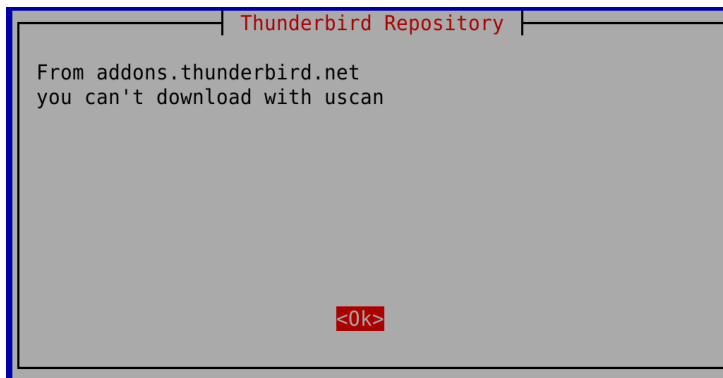


Abbildung 34.6.: Kein Herunterladen per `uscan` von `thunderbird.net`

203b `<ClassicalOrUscan2 203b>`≡ (203a)

```
if grep --quiet "addons.mozilla.org" ${GitPath}/debian/watch
then
  whiptail --title "Mozilla Repository" \
    --msgbox "From addons.mozilla.org \nyou can't download with uscan" \
    15 60
  BuildNewVersion
  set -e
  return
fi
set -e
```

`<ClassicalOrUscan3 204>`



Abbildung 34.7.: Kein Herunterladen per uscan von mozilla.org

Ist kein Herunterladen mit *uscan* möglich, wird die Funktion *BuildNewVersion* zum Herunterladen auf klassische Weise aufgerufen (Kapitel 34.4.2, Seite 206)

Erscheint ein Herunterladen mit *uscan* möglich, wird der Benutzer gefragt, ob er auf „klassische“ Weise oder mit *uscan* die neue Version herunterladen möchte.

```
204 <ClassicalOrUscan3 204>≡ (203b)
    NVTask=$(whiptail --title "Classical download or uscan" \
    --radiolist "How do you want to download the new version? " 17 60 9 \
    "0" "using the classical way" on \
    "1" "using uscan" off --cancel-button "Exit" 3>&2 2>&1 1>&3)
```

```
<ClassicalOrUscan5 205>
```

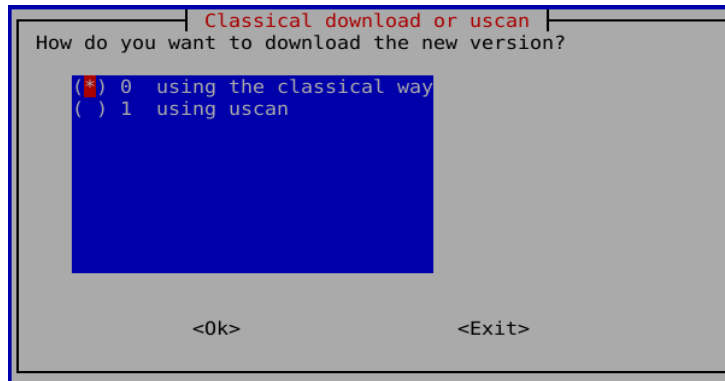


Abbildung 34.8.: Download - klassisch oder mit uscan

```

205  <ClassicalOrUscan5 205>≡ (204)
      if [ -z "${NVTask}" ]
      then
          exit
      fi
      case "$NVTask" in
          0) BuildNewVersion;;
          1) BuildWithUscan;;
      esac
      fi
}

<PullFromSalsa 195>

```

Existiert die Datei *debian/watch* nicht, muss der Benutzer die neue Version auf klassische Weise herunterladen.

Wird statt des klassischen Weges das Herunterladen mit *uscan* ausgewählt, geht es mit Kapitel 34.5 (Seite 248) weiter.

34.4. Herunterladen auf klassische Weise

In der Regel wird der Quellcode einer Software als Archiv bereitgestellt. Dazu stehen verschiedene Formate zur Verfügung. Die Verwendung derselben wird im Folgenden beschrieben.

34.4.1. Archiv-Formate

Für die Nutzung mit *git-buildpackage* wird zwingend ein Orig-Tar-Archiv als Quelle benötigt. Dieses darf nur die Formate **.tar.gz* oder **.tar.xz* haben. Das Orig-Tar-Archiv wird auch in das **Debian**-Archiv hochgeladen.

Zur Nutzung unter Linux wird meist ein **.tar.gz* bereitgestellt. Manchmal ist dies auch ein **.tar.xz*.

Für betriebssystemübergreifende Software wird der Quellcode gerne als Zip-Archiv bereitgestellt. Ein Zip-Archiv wird deshalb in ein *.tar.xz* ungepackt. Dazu gibt es das Tool *mk-origtargz* (Kapitel 34.4.6, Seite 226).

Zusätzlich kann in der zum Projekt gehörigen Datei *debian/gbp.conf* angegeben werden, welches Archiv-Format verwendet werden soll. Wird hier als Kompression *compression = xz* angegeben, muss auch ein **.tar.gz* in ein **.tar.xz* umgewandelt werden. Dies wird dann in der Datei *debian/README.source* dokumentiert (Kapitel 35.4.19, Seite 282).

Mit *mk-origtargz* können der Tarball der Originalautoren umbenannt, optional die Komprimierung geändert und unerwünschte Dateien entfernt werden.

34.4.2. Herunterladen des Quellcodes

Zunächst wird also der Quellcode heruntergeladen.

```
206 <BuildNewVersion 206>≡ (247)
    function BuildNewVersion {
        # Called by ClassicalOrUscan

        echo "Building a new version" >> ${log}

        UpstreamSourceName=$(whiptail --title "Name of the source" \
        --inputbox "Please insert the file name of the upstream source version\n \
        to be downloaded or copied (including version and suffix):\n" \
        --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
```

<BuildNewVersion3 207a>

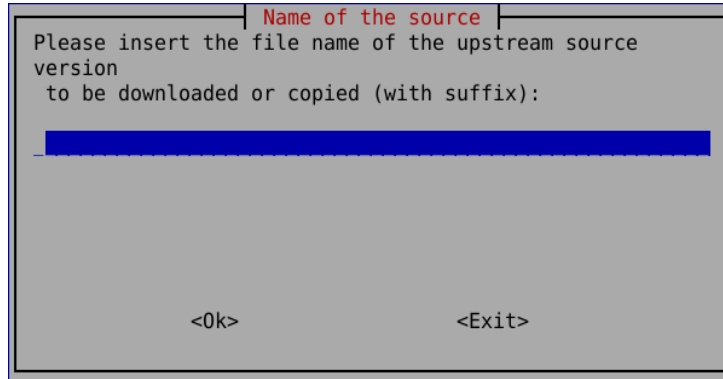


Abbildung 34.9.: Name des Quellcode-Paketes

Es ist der Name des Quellcode-Paketes einzugeben.

```
207a <BuildNewVersion3 207a>≡ (206)
      if [ -z "${UpstreamSourceName}" ]
      then
        exit
      fi
```

<BuildNewVersion4 207b>

Das Programm erledigt das Herunterladen der Upstream-Version. Ist diese bereits in den Projektpfad heruntergeladen worden, kann das Programm auch damit weiter arbeiten.

```
207b <BuildNewVersion4 207b>≡ (207a)
      cd ${PrjPath}
      if whiptail --title "Should the source be downloaded?" \
        --yesno "Should $UpstreamSourceName\n \
        be downloaded from the upstream page?" \
        --defaultno --yes-button "Yes" --no-button "No" 15 60
      <BuildNewVersion4-1 208a>
```

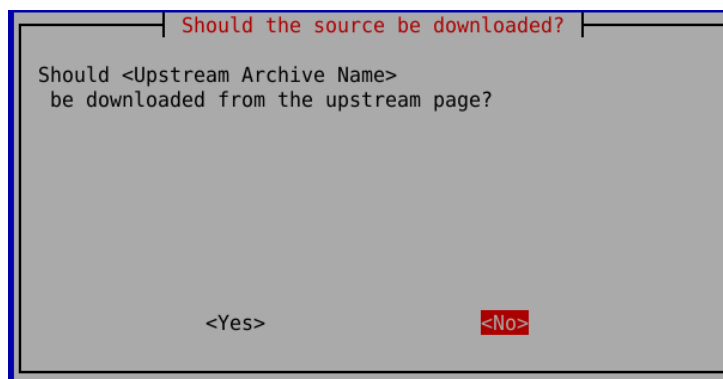


Abbildung 34.10.: Herunterladen (oder kopieren)?

Wird die Frage verneint, geht es mit Kopieren (Kapitel 34.4.2.2, Seite 210) weiter.

34.4.2.1. Herunterladen

```
208a <BuildNewVersion4-1 208a>≡ (207b)
    then
        if [ -z "${DownloadUrl}" ]
        then
            DownloadUrl=$(whiptail --title "Insert URL for download" \
                --inputbox "Please insert the complete\n \
                URL to download ${UpstreamSourceName}\n(with 'https://'\n \
                or so and the name of the archive):" \
                --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
        fi
```

<BuildNewVersion4-2 208b>

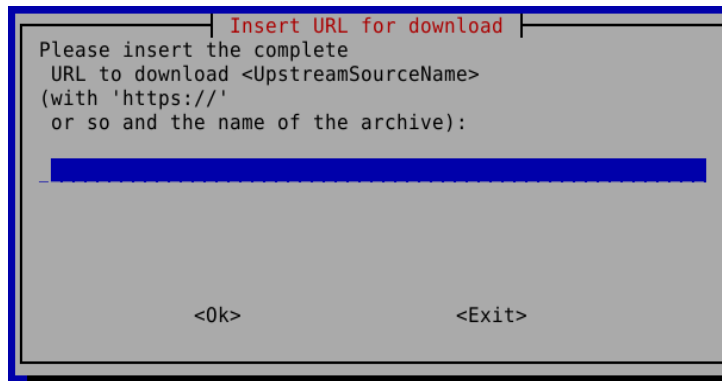


Abbildung 34.11.: Link für Download eingeben

```
208b <BuildNewVersion4-2 208b>≡ (208a)
    if [ -z "${DownloadUrl}" ]
    then
        exit
    else
        changeflag=1
    fi
```

<BuildNewVersion5 208c>

Vorsorglich wird nachgefragt, ob die URL zum Herunterladen des Quellcodes korrekt ist.

```
208c <BuildNewVersion5 208c>≡ (208b)
    if ! whiptail --title "DownloadUrl" \
        --yesno "The complete URL to download ${UpstreamSourceName} is\n \
        ${DownloadUrl}" --yes-button "Yes" --no-button "No" 15 60
    <BuildNewVersion5-1 209a>
```

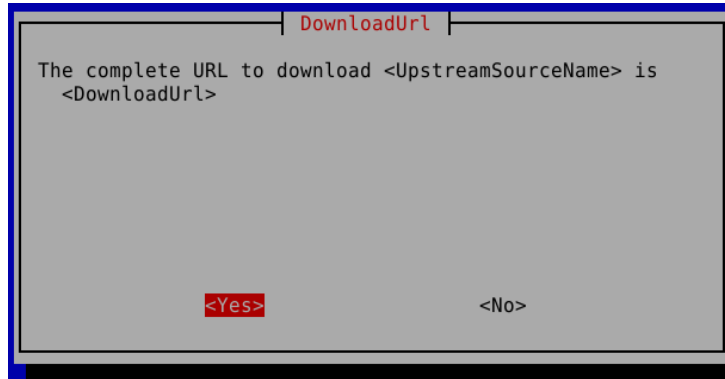


Abbildung 34.12.: Download-URL korrekt?

```
209a  <BuildNewVersion5-1 209a>≡ (208c)
      then
        DownloadUrl=$(whiptail --title "Complete URL" \
          --inputbox "Real complete URL to download ${UpstreamSourceName}\n \
            (with 'https://' or so and the name of the archive):" \
          --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
      fi
```

<BuildNewVersion5-2 209b>

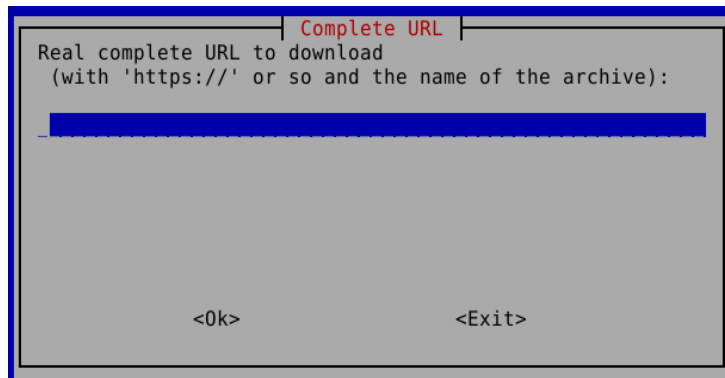


Abbildung 34.13.: Korrekte Download-URL

```
209b  <BuildNewVersion5-2 209b>≡ (209a)
      if [ -z "${DownloadUrl}" ]
      then
        exit
      else
        changeflag=1
      fi
    <BuildNewVersion6 210a>
```

Die neue URL zum Herunterladen des Quellcodes wird in die Konfigurationsdatei eingetragen. Danach erfolgt das Herunterladen mittels *wget*.

Dann wird die Möglichkeit eröffnet, auch die Signaturdatei herunterzuladen und zu prüfen (Kapitel 34.4.7, Seite 229).

```
210a <BuildNewVersion6 210a>≡ (209b)
    # Write download URL into config file
    if [ $changeflag -eq 1 ]
    then
        ReplaceConfigLines 'DownloadUrl' ${DownloadUrl}
        changeflag=0
    fi

    # getting sources using wget
    wget --verbose $DownloadUrl &&
    echo -e "The sources were pulled from\n${DownloadUrl}\n \
    by wget." >> ${log}

    if whiptail --title ".asc file?" \
    --yesno "Do you want to download an .asc file, too?" \
    --yes-button "Yes" --no-button "No" 15 60
    then
        DownloadAscFile
    fi
<BuildNewVersion6-1 210b>
```

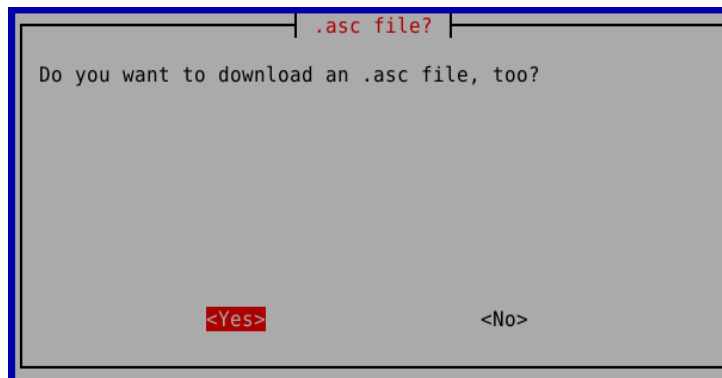


Abbildung 34.14.: *asc Datei herunterladen

34.4.2.2. Kopieren des Quellarchivs

Das Programmskript teilt mit, wohin die Upstream-Version kopiert werden soll.

```
210b <BuildNewVersion6-1 210b>≡ (210a)
    else
        whiptail --title "Please copy the source code now" \
        --msgbox "Please copy ${UpstreamSourceName} to ${PrjPath}!" 15 60
    <BuildNewVersion6-2 211a>
```

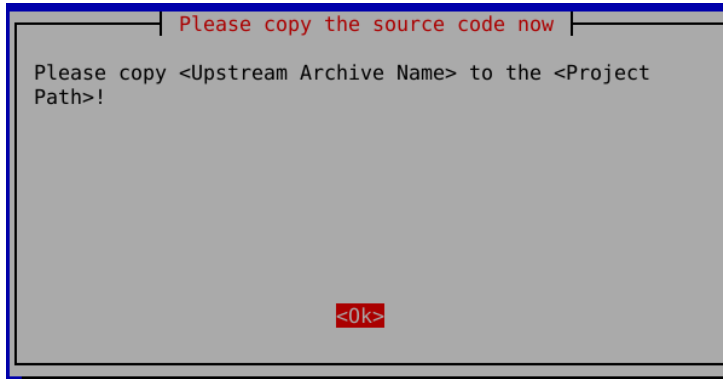



Abbildung 34.15.: Aufforderung zum Kopieren

Das Programmskript fragt, ob das Kopieren erledigt wurde. Wird diese Frage verneint, beendet sich das Programmskript.

```
211a <BuildNewVersion6-2 211a>≡ (210b)
      if ! whiptail --title "Copy finished?" \
        --yesno "Was ${UpstreamSourceName} copied to ${PrjPath}?" \
        --yes-button "Yes" --no-button "No" 15 60
<BuildNewVersion6-3 211b>
```

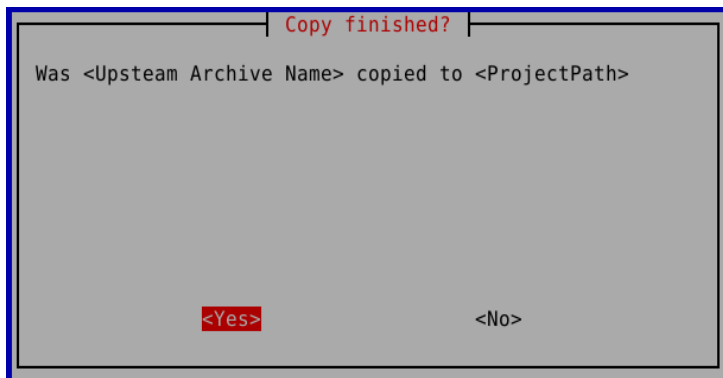


Abbildung 34.16.: Kopieren erledigt?

```
211b <BuildNewVersion6-3 211b>≡ (211a)
      then
        echo "Exit" >> ${log}
        whiptail --title "Bye" --msgbox "Bye" 15 60
        exit
      fi
<BuildNewVersion6-4 212a>
```

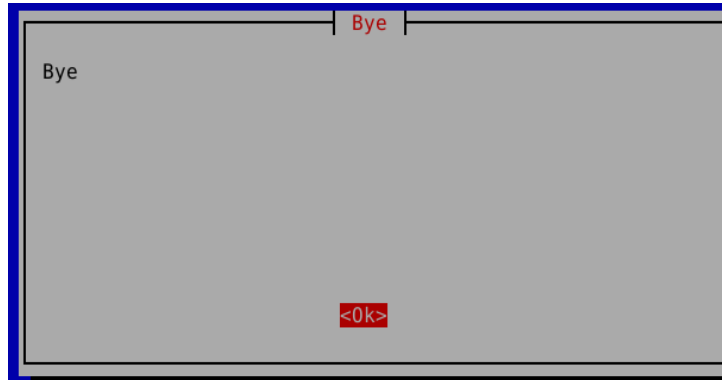


Abbildung 34.17.: Programm beenden

212a `<BuildNewVersion6-4 212a>≡` (211b)
`fi`

```
# Identify the type of the upstream archive by suffix
CutSuffix
```

`<BuildNewVersion7 214a>`

Nun wird die Komprimierung und die Versionsnummer (s. Kapitel 34.4.4 (Seite 214) ermittelt. Dies wird auch insgesamt als *Suffix* bezeichnet. Ist hier alles in Ordnung geht es im Kapitel 34.4.5 (Seite 217) weiter.

34.4.3. Komprimierung ermitteln

Welche Komprimierung *mk-origtargz* automatisch wählt, hängt vom Dateityp des Upstream-Archivs ab. Dabei wird die Dateinamenserweiterung mit einer Liste sinnvoller Dateitypen verglichen.

212b `<CutSuffix 212b>≡` (167b)

```
function CutSuffix {
    # Called by BuildNewVersion

    # List of reasonable suffixes
    typea=( '.tar.gz' '.tar.xz' '.tgz' '.zip' '.oxt' '.xpi' '.jar' )
```

`<CutSuffix1 213>`

Die Dateitypen *.oxt*, *.xpi* und *.jar* beschreiben allesamt *.zip*-Archive.

```

213 <CutSuffix1 213>≡ (212b)
    KnownTyp=0
    set +e
    for element in ${typea[*]}
    do
        if echo ${UpstreamSourceName} | grep ${element} > /dev/null
        then
            echo "Notice from CutSuffix: The suffix of the upstream \
            file is "${element} >> ${log}
            UpstreamSuffix=${element}
            RecentUpstreamSuffix=$(echo ${UpstreamSuffix} | sed --expression s/^./)
            ReplaceConfigLines 'RecentUpstreamSuffix' ${UpstreamSuffix}
            KnownTyp=1
        fi
    done
    set -e

    if [ ${KnownTyp} -ne 1 ]
    then
        echo "Notice from CutSuffix: Unknown suffix" >> ${log}
        if ! whiptail --title "Unknown suffix" \
        --yesno "The suffix of ${UpstreamSourceName} is not listed.\n \
        Continue anyway?" --defaultno --yes-button "Yes" \
        --no-button "No" 15 60
        then
            exit
        fi
    fi
}

```

<Name2Version 214b>

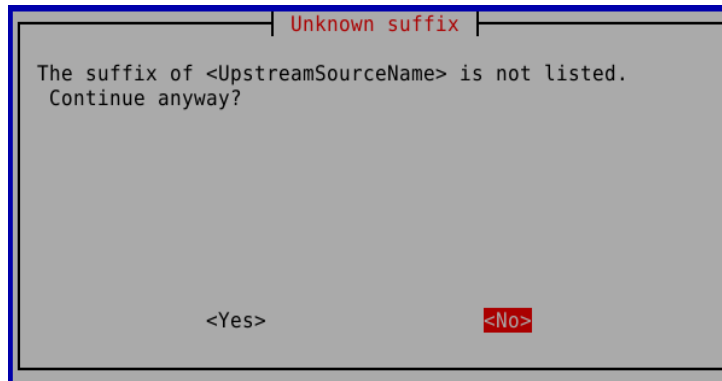


Abbildung 34.18.: Unbekannte Endung

34.4.4. Upstream-Version ermitteln

```
214a <BuildNewVersion7 214a>≡ (212a)
      # Identify the upstream version number
      Name2Version
```

```
<BuildNewVersion8 214c>
```

Die Funktion *Name2Version* versucht aus dem Namen des Upstream-Paketes die Versionsbezeichnung zu extrahieren.

```
214b <Name2Version 214b>≡ (213)
function Name2Version {
    # Called by BuildNewVersion
    # Extracts version from upstream archive name
    Suffix=$(echo ${UpstreamSuffix} | sed --expression='s/\./\\\./g')
    Version1=$(echo ${UpstreamSourceName} | sed --expression="s/${Suffix}$//" | \
    sed --expression="s/.*${SourceName}//gI" | \
    sed --expression='s/-//' | sed --expression='s/v//')
    if [ -z ${Version1} ]
    then
        Version1="0.0.0" # Default value
    fi
}
```

```
<GbpConfIntegration 232c>
```

Misslingt dies, wird die Versionsbezeichnung auf *0.0.0* gesetzt.

Die ermittelte (oder nicht ermittelte) Versionsbezeichnung wird dem Nutzer angezeigt. Der Nutzer hat zu entscheiden, ob die angezeigte Versionsbezeichnung die korrekte ist. Zu beachten ist das Versionierungsschema (Kapitel 11.2, Seite 35).

```
214c <BuildNewVersion8 214c>≡ (214a)
      if ! whiptail --title "Version" \
      --yesno "You want to build version ${Version1}.\n\n \
      It must be a correct version number - not more!" \
      --yes-button "Yes" --no-button "No" 15 60
      <BuildNewVersion9 215>
```

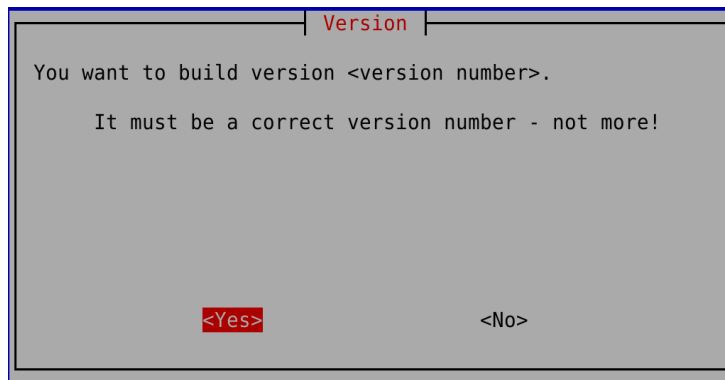


Abbildung 34.19.: Ist dies die korrekte Version?

Ist die Versionsbezeichnung korrekt, wird die Funktion *ExcludeFiles* aufgerufen (s. Kapitel 34.4.5 Seite 217)

Es kann vorkommen, dass das Programmskript die korrekte Versionsbezeichnung nicht ermitteln kann. Dann kann in der folgenden Dialogbox die Versionsbezeichnung eingegeben werden. Damit sie auch weiterverarbeitet werden kann, darf die Versionsbezeichnung neben den Ziffern nur Punkte enthalten. Sie darf ferner die Angabe enthalten, ob es sich um einen *Release-Kandidaten*, eine *Beta-* oder *Alpha-Version* handelt. Zulässig ist auch die Angabe, ob es sich um einen bestimmten *Commit* aus dem *Git*-Repository handelt.

215 `<BuildNewVersion9 215>≡` (214c)

```

then
  Version1=$(whiptail --title "Version" \
    --inputbox "Name of the upstream version: ${UpstreamSourceName}\n \
    Which version (without repack identifiers and without revision)\n \
    of the package ${SourceName} should be built?" \
    --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)

```

`<BuildNewVersion10 216a>`

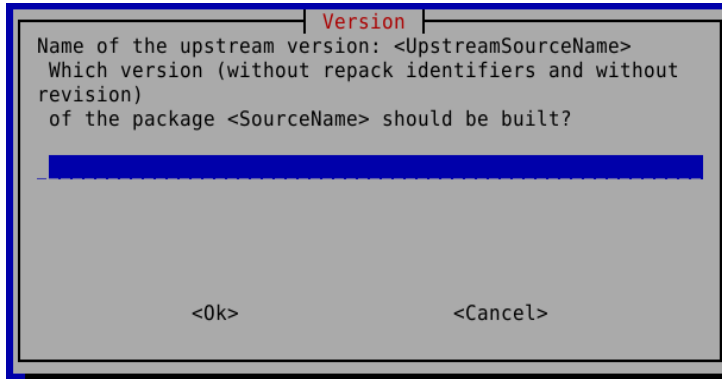


Abbildung 34.20.: Welche Version soll gebaut werden?

```
216a  <BuildNewVersion10 216a>≡ (215)
      if ! whiptail --title "Version" \
        --yesno "Do you really want to build version ${Version1}." \
        --yes-button "Yes" --no-button "Exit" 15 60
      <BuildNewVersion11 216b>
```

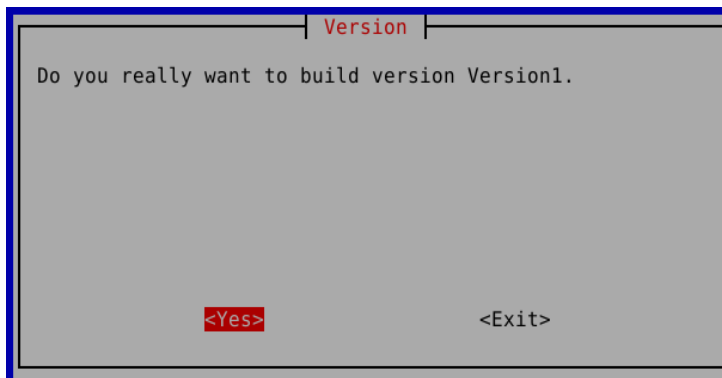


Abbildung 34.21.: Wird korrekte Version gebaut?

```
216b  <BuildNewVersion11 216b>≡ (216a)
      then
        echo "Exit" >> ${log}
        whiptail --title "Bye" --msgbox "Bye" 15 60
        exit
      fi
      <BuildNewVersion11-1 217a>
```

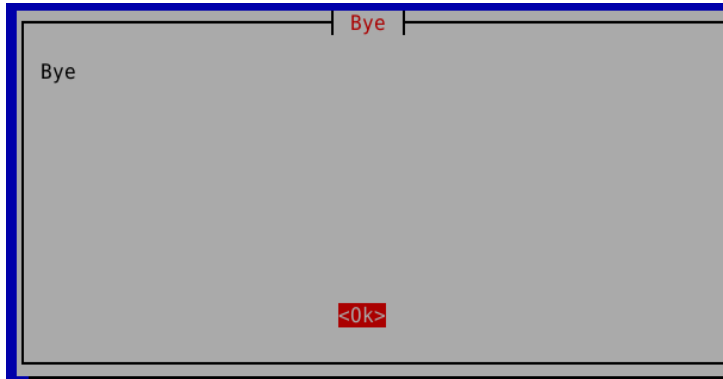


Abbildung 34.22.: Programm beenden

Die Funktion *ExcludeFiles* dient dazu, Dateien aus dem Upstream-Archiv von der Übernahme in die Datei **.orig.tar.xz* ausschließen.

Sollen keine Dateien ausgeschlossen werden, geht es mit Kapitel 34.4.6 (Seite 226)

```
217a <BuildNewVersion11-1 217a>≡ (216b)
      fi
```

```
      ExcludeFiles
```

```
      echo "Version "${Version1}${ESuffix}" of the package "${PackName}" \
      should be built." >> ${log}
```

```
<BuildNewVersion12 226b>
```

34.4.5. Dateien aus Upstream-Archiv ausschließen

Bevor *mk-origtar.gz* aufgerufen wird, ermöglicht das Programmskript einzelne Quellcode-Dateien von der Aufnahme in das *orig*-Archiv auszuschließen.

```
217b <ExcludeFiles 217b>≡ (221a)
      function ExcludeFiles {
          # Called by BuildNewVersion
```

```
<ExcludeFiles1 217c>
```

Anzugeben ist, dass Dateien auszuschließen sind. Dazu ermittelt das Programmskript, woher die Informationen zum Ausschluss von Dateien erfolgen sollen. Dies ist hier die Datei *debian/copyright*.

Nun wird geprüft, ob eine Datei *debian/copyright* existiert.

```
217c <ExcludeFiles1 217c>≡ (217b)
      # Checks whether debian/copyright contains a section Files-Excluded
      gitflag=0
      exflag=0
      crflag=0
      if [ -f ${GitPath}/debian/copyright ]
      then
      <ExcludeFiles2 218a>
```

Sodann wird geprüft, ob sie den Ausdruck *Files-Excluded* enthält. In diesem Fall wird abgefragt, ob die Datei *debian/copyright* editiert werden soll.

```
218a  <ExcludeFiles2 218a>≡ (217c)
      crflag=1
      set +e
      grep 'Files-Excluded' ${GitPath}/debian/copyright > /dev/null
      if [ $? -eq 0 ]
      then
          exflag=1
          whiptail --title "Copyright file contains Files-Excluded" \
          --msgbox "debian/copyright contains section Files-Excluded." 15 60
          less --LINE-NUMBERS ${GitPath}/debian/copyright
      fi
      set -e
  fi
```

<ExcludeFiles3 218b>

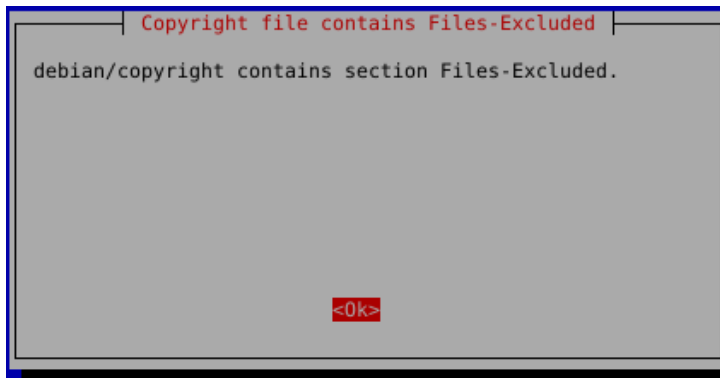


Abbildung 34.23.: Datei *debian/copyright* enthält Abschnitt *Files-Excluded*

```
218b  <ExcludeFiles3 218b>≡ (218a)
      if whiptail --title "Exclude files from upstream source" \
      --yesno "Do you want to exclude files from upstream source?" \
      --defaultno --yes-button "Yes" --no-button "No" 15 60
      <ExcludeFiles3-1 219>
```

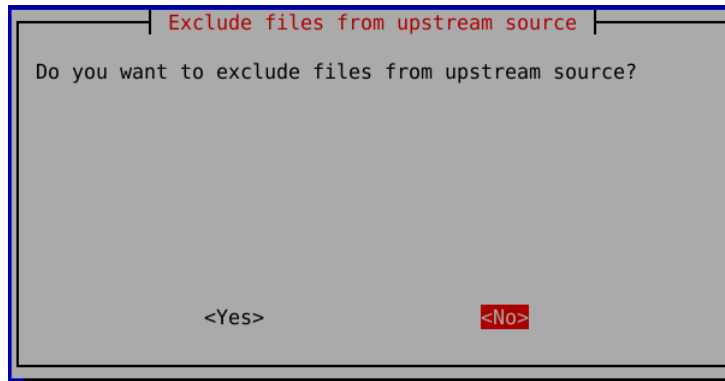



Abbildung 34.24.: Dateien ausschließen

Wird kein Ausschluss von Dateien benötigt, wird im Hintergrund *mk-origtargz* ausgeführt. (Kapitel 34.4.6, Seite 226). Dann geht es in Kapitel 34.4.8, (Seite 231) weiter.

```
219 <ExcludeFiles3-1 219>≡ (218b)
    then
      if [ $crflag -eq 1 ]
      then
        if whiptail --title "Copyright file existst" \
          --yesno "debian/copyright exists.\nDo you want to edit it?" \
          --yes-button "Yes" --no-button "No" 15 60
        then
          gitflag=1
          nano --linenumbers --mouse --softwrap ${GitPath}/debian/copyright
        fi
        AddOpt=" --copyright-file "${SourceName}"/debian/copyright"
```

<ExcludeFiles4 220>



Abbildung 34.25.: Soll debian/copyright editiert werden?

Andernfalls wird die Funktion *SpecialExcludeFile* aufgerufen. In dieser Funktion wird, sofern nicht - wie im vorliegenden Fall - die Angaben in der Datei *debian/copyright* verwandt werden, nach einer speziellen Datei gefragt, die die Namen der auszuschließenden Dateien im Format DEP-5¹ enthält.

```
220 <ExcludeFiles4 220>≡ (219)
      else
        SpecialExcludeFile
      fi
<ExcludeFiles5 221b>
```

¹DEP-5[19]

Steht schon vor der Einreichung des Paketes zur **New Queue** fest, dass Dateien auszuschließen sind, existiert zu diesem Zeitpunkt noch keine Datei *debian/copyright*. Es bietet sich dann an, die auszuschließenden Dateien in einer separaten Datei aufzulisten.

Die Funktion *ExcludeFiles* ruft hierfür die folgende Funktion auf. In dieser Funktion wird nach einer speziellen Datei gefragt, die die Namen der auszuschließenden Dateien im Format DEP-5 enthält.

221a *<SpecialExcludeFile 221a>*≡ (228)

```
function SpecialExcludeFile {
    # Called by ExcludeFiles
    if [ -z "${ExcludeFile}" ]
    then
        ExcludeFile=$(whiptail --title "Name of exclude file" \
            --inputbox "Please insert name of the exclude file:" \
            --cancel-button "Exit" 15 60 3>&2 2>&1 1>&3)
        if [ -z "${ExcludeFile}" ]
        then
            exit
        fi
        echo 'ExcludeFile='${ExcludeFile} >> ${ConfigPath}${OrigName}
    else
        whiptail --title "Exclude file name" \
            --msgbox "The name of the exlude file is ${ExcludeFile}" \
            15 60
    fi
    AddOpt=" --copyright-file "${ExcludeFile}
}
```

<ExcludeFiles 217b>

Dann erfragt das Programmskript den Anhang zur Ergänzung der Upstream-Versionsbezeichnung.

Oft ist hier als Anhang *+dfsg* zu wählen, um den Grund des Ausschlusses zu dokumentieren (s.a Kapitel 10.4.1.3, Seite 32)

221b *<ExcludeFiles5 221b>*≡ (220)

```
ESuffixN=$(whiptail --title "Suffix:" \
    --radiolist "Please choose the suffix: " \
    --cancel-button "Cancel" 15 60 4 \
    "0" "+ds" off \
    "1" "+dfsg" on \
    "2" "other" off 3>&2 2>&1 1>&3)
if [ ${ESuffixN} -eq 1 ]
then
    ESuffix="+dfsg"
elif [ ${ESuffixN} -eq 0 ]
then
    ESuffix="+ds"
```

<ExcludeFile6 222>

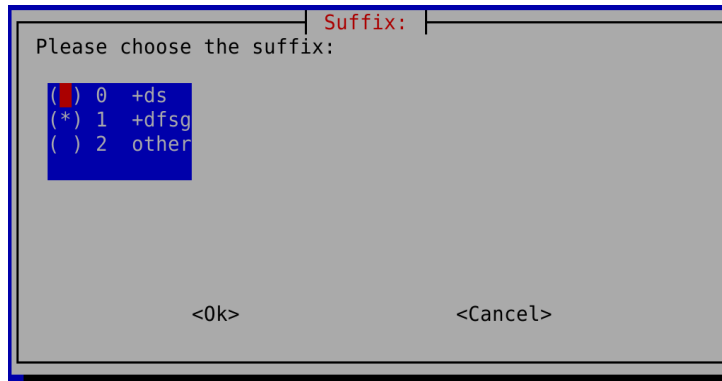


Abbildung 34.26.: Suffix für den Ausschluss von Dateien

Wenn ein anderes Suffix als die beiden vorgeschlagenen verwendet werden soll, ist dies manuell hinzuzufügen. Dieser Eintrag muss auch entsprechend in der Datei *debian/watch* ergänzt werden. (s.a Kapitel 35.4.7, Seite 269)

```

222 <ExcludeFile6 222>≡ (221b)
      else
        ESuffix=$(whiptail --title "Enter suffix" \
          --inputbox "Please insert the suffix:" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
      fi
<ExcludeFile7 223>

```

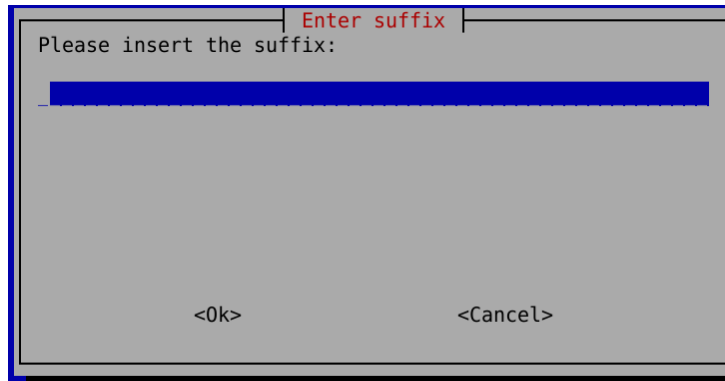


Abbildung 34.27.: Eigener Suffix für den Ausschluss von Dateien

```

223  <ExcludeFile7 223>≡ (222)
      if [ -z "${ESuffix}" ]
      then
        whiptail --title "Warning!" \
          --msgbox "You repacked the upstream source and\n\
          do not want to use a repack suffix." 15 60
        if [ -n "${RecentRepackSuffix}" ]
        then
          # Remove suffix from config file
          sed --in-place \
            --expression="s/RecentRepackSuffix=.*//g" \
            ${ConfigPath}${OrigName}
        fi
      else
        <ExcludeFile8 224a>

```

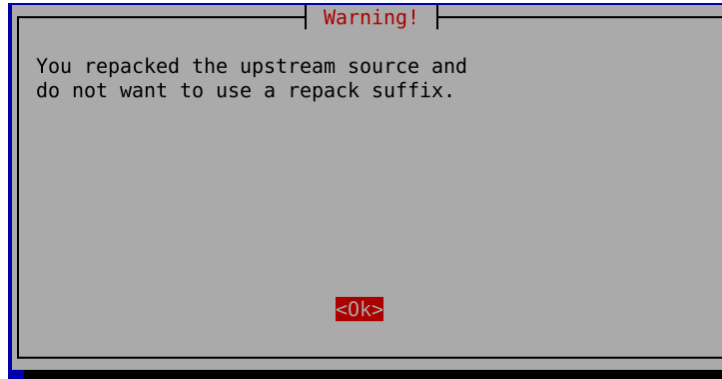


Abbildung 34.28.: Warnung! - Kein Suffix angegeben

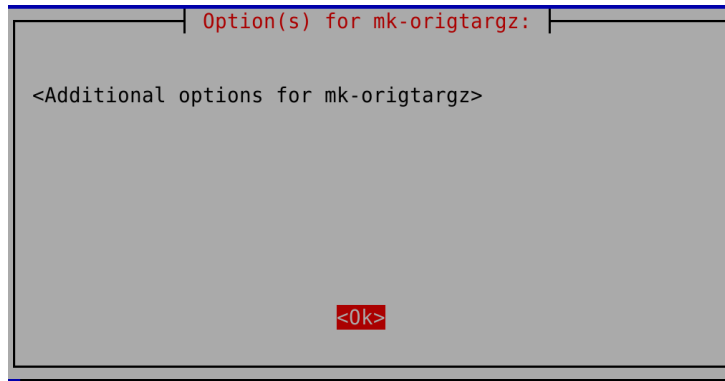
In manchen Fällen kann ein Pluszeichen (+) Probleme vor allem beim Bauen von Java-Paketen verursachen. Manchmal ist es auch notwendig, dass kein Suffix hinzugefügt werden kann. Die Dokumentation solcher Vorgänge kann dann in der Datei *README.source* (Kapitel 35.4.19, Seite 282) erfolgen.

```
224a <ExcludeFile8 224a>≡ (223)
# Insert suffix into config file
if [ -z "${RecentRepackSuffix}" ]
then
    echo "RecentRepackSuffix=${ESuffix} >> ${ConfigPath}${OrigName}
else
    sed --in-place \
    --expression="s/RecentRepackSuffix=./RecentRepackSuffix=${ESuffix}/g" \
    ${ConfigPath}${OrigName}
fi

RecentRepackSuffix=${ESuffix}
AddOpt=${AddOpt}" --repack-suffix "${ESuffix}
fi
<ExcludeFiles10 224b>
```

Die Kompression des **.orig.tar.**-Archives wird in der Variable *suffix* hinterlegt. Er wird auf *.tar.xz* festgelegt.

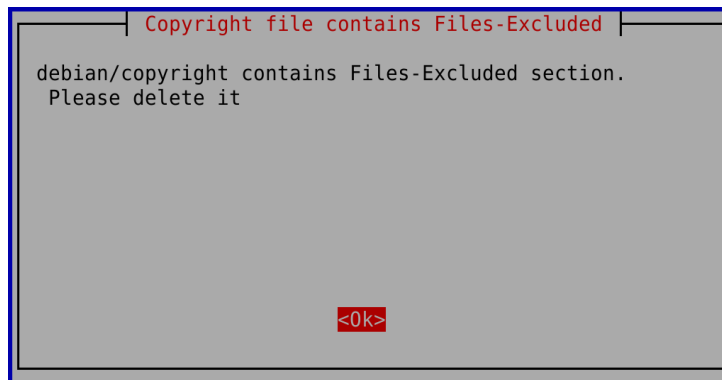
```
224b <ExcludeFiles10 224b>≡ (224a)
whiptail --title "Option(s) for mk-origtargz:" \
--msgbox "\n${AddOpt}" 15 60
else
    AddOpt=""
    ESuffix=""
<ExcludeFiles12 225>
```

Abbildung 34.29.: Zusätzliche Optionen für *mk-origtargz*

Sollen keine Dateien ausgeschlossen werden, enthält aber die Datei *debian/copyright* einen Abschnitt *Files-Excluded:*, so ist dieser (manuell) zu entfernen.

```
225 <ExcludeFiles12 225>≡ (224b)
    if [ $exflag -eq 1 ]
    then
        gitflag=1
        whiptail --title "Copyright file contains Files-Excluded" \
        --msgbox "debian/copyright contains Files-Excluded section.\n \
        Please delete it" 15 60
        nano --linenumbers --mouse \
        --softwrap ${GitPath}/debian/copyright
    fi
fi
```

```
<ExcludeFiles15 226a>
```

Abbildung 34.30.: *debian/copyright* enthält auszuschließende Dateien

Wurde im Zusammenhang mit dem Ausschluss von Dateien die Datei *debian/copyright* bearbeitet, erfolgt ein entsprechender *commit*.

```
226a <ExcludeFiles15 226a>≡ (225)
    if [ $gitflag -eq 1 ]
    then
        git add debian/copyright
        git commit -am "Changed debian/copyright"
    fi
}

<CheckSignature 230>
```

34.4.6. Debian-Quellcode-Datei erzeugen

Das Skript führt dann die Funktion *BuildNewVersion* weiter aus und übergibt dem Programm *mk-origtargz* die zum Ausschluss notwendigen Parameter. (Referenz auf diese Stelle in der anderen Funktion)

Auf diese Weise wird ein neuer Orig-Tarball mit *mk-origtargz* ohne die auszuschließenden Dateien aus dem bisherige **.tar.gz* erstellt und dessen Inhalt mit *gbp import-orig* in das vorhandene Git-Repositorium eingefügt.

```
226b <BuildNewVersion12 226b>≡ (217a)
    # Creating orig file using mk-origtargz
    if [ -z ${Version1} ]
    then
        whiptail --title "No version number!" \
        --msgbox "No version - no *.orig.tar.gz! Bye!" \
        --ok-button "Exit" 15 60
        exit
    fi
    echo "mk-origtargz --package "${SourceName}" \
    --version "${Version1}${AddOpt}" "${UpstreamSourceName} >> ${log}
    mk-origtargz --package ${SourceName} \
    --version ${Version1}${AddOpt} ${UpstreamSourceName} 2>> ${log}
    <BuildNewVersion13 227a>
```

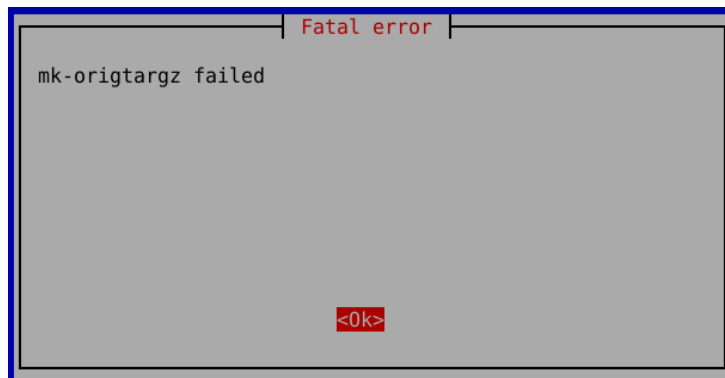

Abbildung 34.31.: Keine Versionsnummer – kein *mk-origtargz*

```

227a  <BuildNewVersion13 227a>≡ (226b)
      if [ $? -eq 0 ]
      then
          echo "orig file was created by mk-origtargz" >> ${log}
          Version1=${Version1}${ESuffix}
      else
          echo "mk-origtargz failed" >> ${log}
          whiptail --title "Fatal error" \
          --msgbox "mk-origtargz failed" 15 60
          exit
      fi

```

<BuildNewVersion14 227b>

Abbildung 34.32.: *mk-origtargz* gescheitert

```

227b  <BuildNewVersion14 227b>≡ (227a)
      Link2File
      SearchGbpConf

      cd ${GitPath}

      DebianBranch4Import
<BuildNewVersion15 243b>

```

13. Oktober 2024

Wenn die Variable *RecentBranch* nicht existiert oder leer ist, wird ihr der Wert *debian/sid* zugeordnet und ein entsprechender Eintrag in der Konfigurationsdatei vorgenommen.

```
228 <DebianBranch4Import 228>≡ (331b)
function DebianBranch4Import {
    # Called by BuildNewVersion # Makes sure RecentBranch contains value
    set +e
    RecentBranch=$(grep 'RecentBranch=' ${ConfigPath}${OrigName})
    RecentBranch=$(echo ${RecentBranch} | sed --expression='s/RecentBranch=//')
    set -e
    if [ -z "${RecentBranch}" ]
    then
        RecentBranch="debian/sid"
        changeflag=1
        whiptail --title "Set RecentBranch" \
            --msgbox "Set RecentBranch to ${RecentBranch}" 15 60
    fi

    if [ $changeflag -eq 1 ]
    then
        echo 'RecentBranch=${RecentBranch}' >> ${ConfigPath}${OrigName}
    fi
    echo "Notice from DebianBranch4Import: \
The branch is ${RecentBranch} >> ${log}"
    changeflag=0
}
```

<SpecialExcludeFile 221a>

34.4.7. Signatur prüfen

Manche Projekte veröffentlichen neben dem Quellcodepaket auch eine Signaturdatei. Das Skript kann diese herunterladen und eine kryptografische Prüfung durchführen. Erwartet wird eine Signaturdatei im *asc*-Format.

34.4.7.1. Signatur-Datei herunterladen

```

229 <DownloadAscFile 229>≡ (230)
function DownloadAscFile {
    # Called by BuildNewVersion and itself

    cd ${PrjPath}

    AscFileURL=$(whiptail --title "URL of .asc file" \
        --inputbox "URL and name (with suffix)\nof the .asc file:" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)

    if [ $? -eq 1 ]
    then
        return 1
    fi

    if [ -z "${AscFileURL}" ]
    then
        echo -e "URL and name (with suffix)\nof the .asc file:"
        read AscFileURL
    fi

    if [ -n "${AscFileURL}" ]
    then
        # getting -asc file using wget
        wget --version ${AscFileURL} &>> ${log}
        if [ $? -eq 0 ]
        then
            echo -e "The .asc file was pulled from\n${AscFileURL}\n \
                by wget." >> ${log}
            whiptail --title "Download successful" \
                --msgbox "${AscFileURL} was downloaded." 15 60
            if [ -r ${UpstreamSourceName}.asc ]
            then
                CheckSignature
            else
                whiptail --title "There is something wrong!" \
                    --msgbox "Maybe you has downloaded the wrong .asc file." 15 60
                DownloadAscFile
            fi
        fi
        else
            DownloadAscFile
        fi
    fi
}

```

13. Oktober 2024

<Link2File 231>

34.4.7.2. Prüfung der Signatur

```
230 <CheckSignature 230>≡ (226a)
    function CheckSignature {
        # Called by DownloadAscFile
        gpg --verify {UpstreamSourceName}.asc >> {log}

        if [ {?} -ne 0  ]
        then
            tail --lines=5 {log}
            read x
        else
            whiptail --title "Check successfull!" --msgbox "gpg \
                --verify has been successfull" 15 60
        fi
    }
```

<DownloadAscFile 229>

34.4.8. Link durch Kopie ersetzen

Das Standardverhalten von *mk-origtargz* (Kapitel 34.4.6, Seite 226) ist einen symbolischen Verweis auf die Originaldatei zu erzeugen, wenn diese unverändert übernommen wird.

Das Programmskript ersetzt diesen Verweis gegebenenfalls durch eine entsprechende Datei.

```

231 <Link2File 231>≡ (229)
function Link2File {
    # Called by BuildNewVersion
    echo "Version: "${Version1} >> ${log}
    set +e
    OrigLinkNr=$(ls -la ${PrjPath} | grep ${Version1} | \
    grep --count -e '.orig.tar.[gx]z -> ')
    if [ $OrigLinkNr -ge 1 ]
    then
        OrigLink=$(ls -la ${PrjPath} | grep --regexp='.orig.tar.[gx]z -> ')
        OrigLink=$(echo ${OrigLink} | \
        sed --expression='s/^.*:.. //' | sed --expression='s/ //g')
        echo "${OrigLink}" will be transformed into a file" >> ${log}

        LinkTarget=$(echo $OrigLink | sed --expression='s/^.*->//')
        LinkName=$(echo $OrigLink | sed --expression='s/->.*$//')
        rm ${PrjPath}/${LinkName}
        cp -a ${PrjPath}/${LinkTarget} ${PrjPath}/${LinkName}
        whiptail --title "Result of transformation link to file:" \
        --msgbox "$(ls -la ${PrjPath})" --scrolltext 15 60
        echo "Result of transformation link to file: \
        "$(ls -la ${PrjPath}) >> ${log}
    fi
    set -e
}

<CheckGitStatus 183>

```

34.4.9. *gbp*-Konfigurationsdatei

gbp import-orig (Kapitel 34.4.11, Seite 243) fügt den heruntergeladenen Quellcode dem Git-Repository hinzu.

Zur Steuerung dieses Prozesses wird die Einfügung einer vorbereiteten Datei *gbp.conf* in das Verzeichnis *.git* ermöglicht (Kapitel 34.4.11, Seite 243)

```
232a <SearchGbpConf 232a>≡ (240)
function SearchGbpConf {
    # Called by BuildNewVersion BuildWithUscan

    # Neither .git/gbp.conf nor debian/gbp.conf exist
    if [ ! -f ${GitPath}/.git/gbp.conf -a ! -f ${GitPath}/debian/gbp.conf ]
    then
        if whiptail --title "gbp.conf needed?" \
            --yesno "Do you want to integrate a special gbp.conf for this project?" \
            --defaultno --yes-button "Yes" --no-button "No" 15 60
        <SearchGbpConf1 232b>
```

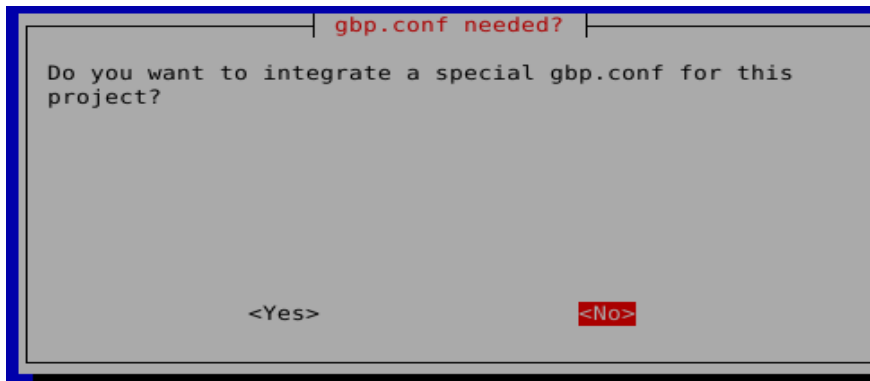


Abbildung 34.33.: Spezielle *gbp.conf*

Wird diese Frage verneint, geht es mit dem Import in das Git-Repository (Kapitel 34.4.11, Seite 243) weiter.

```
232b <SearchGbpConf1 232b>≡ (232a)
    then
        GbpConfIntegration
    fi
fi
<SearchGbpConf2 234>

232c <GbpConfIntegration 232c>≡ (214b)
function GbpConfIntegration {
    # Called by SearchGbpConf and itself
    GbpConfPath=$(whiptail --title "gbp.conf" \
        --inputbox "Please insert the path to the your special\n \
        gbp.conf for this project:" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    <GbpConfIntegration1 233>
```

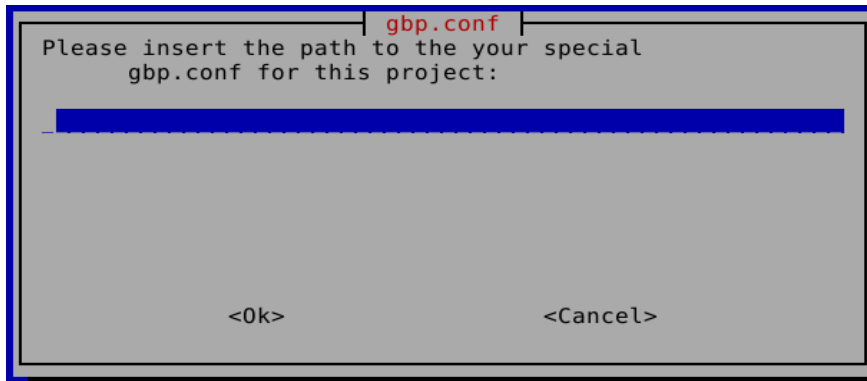


Abbildung 34.34.: Abfrage: Pfad zur gbp.conf

```

233  <GbpConfIntegration1 233>≡ (232c)
      if [ -z "${GbpConfPath}" ]
      then
          echo "Please insert the path to the your special gbp.conf for this project:"
          read GbpConfPath
      fi

      # Replace tilde if necessary
      SuspectPath=${GbpConfPath}
      ReplaceTilde
      GbpConfPath=${CleanPath}

      if [ -f ${GbpConfPath}/gbp.conf ]
      then
          cp -av ${GbpConfPath}/gbp.conf ${GitPath}/.git/
      else
          if whiptail --title "File not found!" \
          --yesno "There was no gbp.conf found at ${GbpConfPath}! Retry?" \
          --yes-button "Yes" --no-button "No" 15 60
          then
              GbpConfIntegration
          fi
      fi
  }

  <TwoConfFilesFound 237b>

```

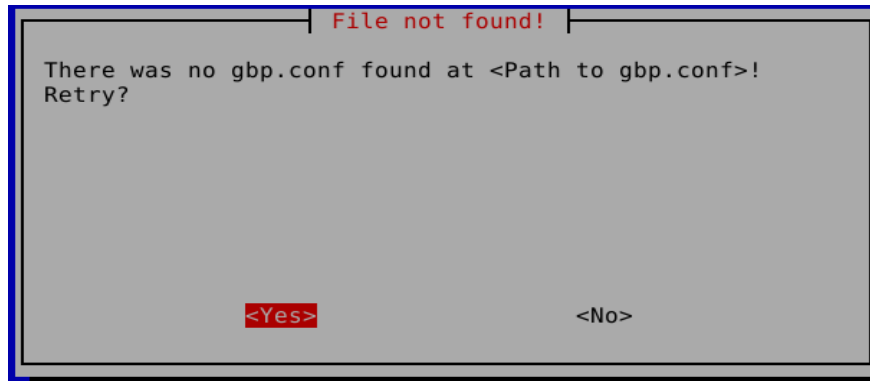


Abbildung 34.35.: gbp.conf nicht gefunden

Wird eine Datei *gbp.conf* gefunden, wird diese zwecks Überprüfung und gegebenenfalls Anpassung im Editor angezeigt. Zu prüfen ist besonders der Wert der Variablen *compression* – vor allem dann, wenn ein entsprechender Wechsel des Archivformats erfolgen soll.

```
234 <SearchGbpConf2 234>≡ (232b)
    # debian/gbp.conf exists, but not .git/gbp.conf
    if [ ! -f ${GitPath}/.git/gbp.conf -a -f ${GitPath}/debian/gbp.conf ]
    then
        whiptail --title "Found gbp.conf" \
        --msgbox "Please check and edit your gbp.conf (if necessary)" 15 60
    <SearchGbpConf3 235>
```

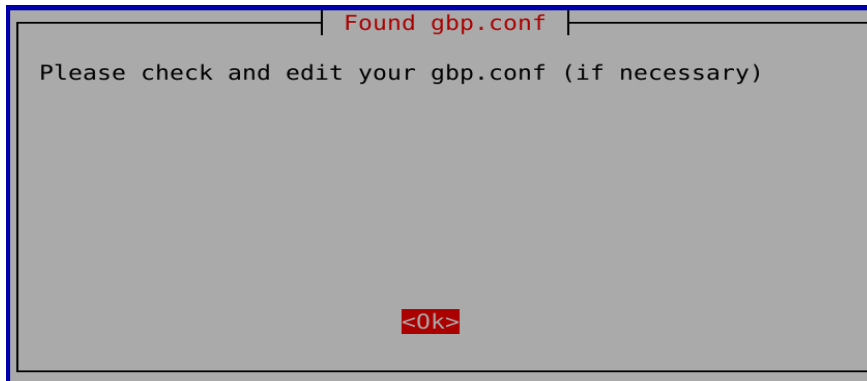



Abbildung 34.36.: Check gbp.conf

235 `<SearchGbpConf3 235>≡``(234)`

```

    nano --linenumbers --mouse --softwrap ${GitPath}/debian/gbp.conf
fi
# .git/gbp.conf exists, but not debian/gbp.conf
if [ -f ${GitPath}/.git/gbp.conf -a ! -f ${GitPath}/debian/gbp.conf ]
then
    whiptail --title "Found gbp.conf" \
    --msgbox "Please check and edit your gbp.conf (if necessary)" 15 60

```

`<SearchGbpConf4 236>`

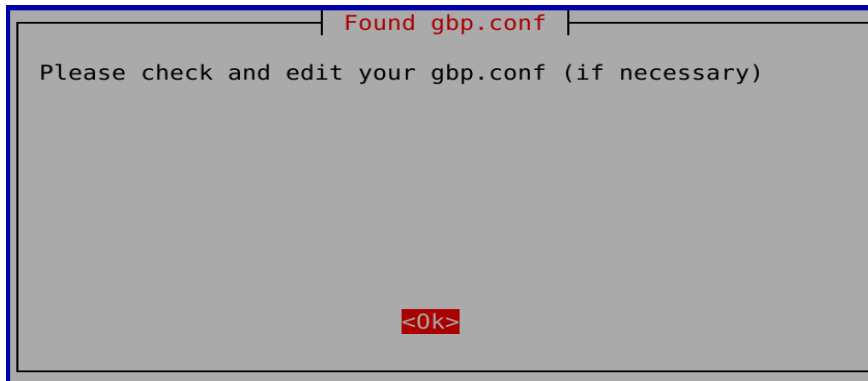


Abbildung 34.37.: Check gbp.conf

236 *<SearchGbpConf4 236>*≡

(235)

```
    nano --linenumbers --mouse --softwrap ${GitPath}/.git/gbp.conf
fi
# There is a gbp.conf in both directories
if [ -f ${GitPath}/.git/gbp.conf -a -f ${GitPath}/debian/gbp.conf ]
then
    TwoConfFilesFound
fi
}
```

<MovingGbpConfFile 331b>

Im folgenden wird die Datei mit den Informationen abgebildet, die für viele Debian-Pakete verwendet werden können.

```
237a <debian/gbp.conf 237a>≡
# Configuration file for git-buildpackage and friends

[DEFAULT]
# use pristine-tar:
pristine-tar = True
# generate xz compressed orig file
compression = xz
debian-branch = debian/experimental
upstream-branch = upstream

[pq]
patch-numbers = False

[dch]
id-length = 7
debian-branch = debian/experimental

[import-orig]
# filter out unwanted files/dirs from upstream
filter = [ '.cvsignore', '.gitignore', '.hgtags', '.hgignore', '*.orig', *.rej' ]
# filter the files out of the tarball passed to pristine-tar
filter-pristine-tar = True
```

Nach der Überprüfung und eventuellen Verbesserung der Datei *gbp.conf* geht es mit weiteren Vorbereitungen für den Import ins lokale *Git*-Repository weiter (Kapitel 34.4.10, Seite 241). Verläuft die Überprüfung des *Git*-Repositories befundlos, geht es direkt mit dem Import ins lokale *Git*-Repository weiter (Kapitel 34.4.11, Seite 243).

Die Funktion *TwoConfFilesFound* behandelt den Fall, dass jeweils eine Datei *gbp.conf* sowohl im Verzeichnis *.git/* als auch im Verzeichnis *debian/* vorkommt.

```
237b <TwoConfFilesFound 237b>≡ (233)
function TwoConfFilesFound {
    # Called by SearchGbpConf MovingGbpConfFile

    whiptail --title "Information" \
        --msgbox "There are a gbp.conf in debian/ and a gbp.conf in .git/" 15 60
<TwoConfFilesFound0-1 238>
```

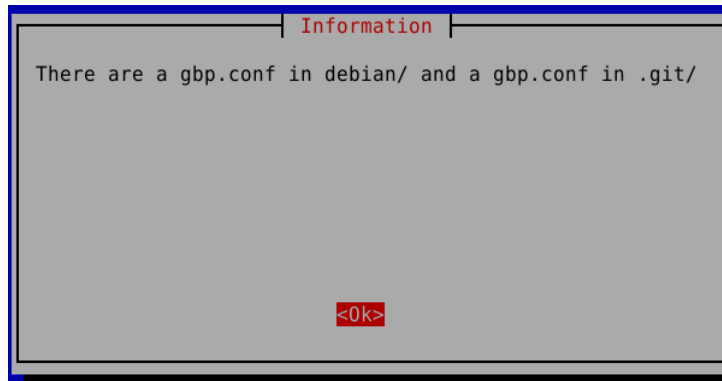


Abbildung 34.38.: *gbp.conf* zweimal gefunden.

```
238  <TwoConfFilesFound0-1 238>≡ (237b)
      # Are they different?
      GitConfFile=$(cat ${GitPath}/.git/gbp.conf)
      DebianConfFile=$(cat ${GitPath}/debian/gbp.conf)
      if [ "${GitConfFile}" != "${DebianConfFile}" ]
      then
        whiptail --title "Warning!" \
          --msgbox "There are a gbp.conf in debian/ and a gbp.conf in .git/\n \
            But they are different!\n\nThe left column is ${GitPath}/.git/gbp.conf\n \
            The right column is ${GitPath}/debian/gbp.conf\n \
            After studying the diff press RETURN!" 15 60
      fi
    <TwoConfFilesFound1 239>
```

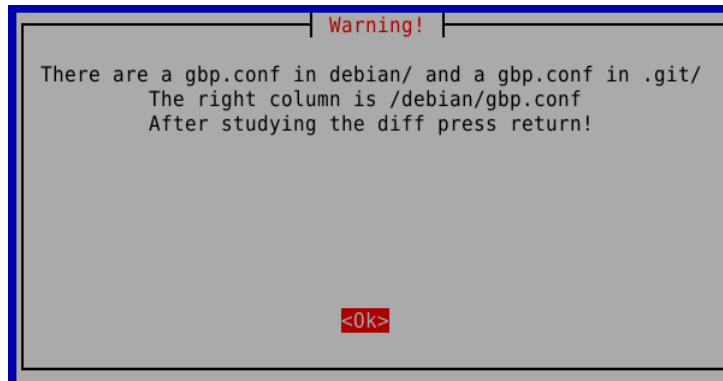


Abbildung 34.39.: Unterschiedliche Konfigurationsdateien

Mit `diff --side-by-side (-y)` wird die Differenz in zwei Spalten angezeigt.

```
239 <TwoConfFilesFound1 239>≡ (238)
    diff --side-by-side ${GitPath}/.git/gbp.conf ${GitPath}/debian/gbp.conf
    read a
    fi
    # Editing
    if whiptail --title "debian/gbp.conf" \
    --yesno "Do you want to edit ${GitPath}/debian/gbp.conf?" \
    --yes-button "Yes" --no-button "No" 15 60
    <TwoConfFilesFound2 240>
```



Abbildung 34.40.: *gbp.conf* im Verzeichnis *debian/* bearbeiten?

```
240  <TwoConfFilesFound2 240>≡ (239)
      then
        nano --linenumbers --mouse --softwrap ${GitPath}/debian/gbp.conf
      fi
      if whiptail --title ".git/gbp.conf" \
        --yesno "Do you want to edit ${GitPath}/.git/gbp.conf (too)?" \
        --yes-button "Yes" --no-button "No" 15 60
      then
        nano --linenumbers --mouse --softwrap ${GitPath}/.git/gbp.conf
      fi
    }

    <SearchGbpConf 232a>
```

Abbildung 34.41.: *gbp.conf* im Verzeichnis *.git/* bearbeiten?

34.4.10. Prüfung des Git-Repositories

Zur weiteren Vorbereitung des Importes in das lokale Git-Repositories finden noch Prüfungen desselben statt.

```

241 <Import2Git 241>≡ (243a)
    function Import2Git {
        # Called by BuildNewVersion and itself

        CheckGitStatus # to exercise caution
        CheckTags

    <Import2Git1 244>
  
```

Es wird geprüft, ob noch eine lokale Änderung nicht zum Commit vorgemerkt wurde. Dies erfolgt mit der Funktion *CheckGitStatus* (Kapitel 33.4.1, Seite 183). Solche Änderungen müssen vor dem Import einer neuen Version abgearbeitet werden.

Eine weitere Prüfung erfolgt mit der Funktion *CheckTags*. Diese listet die vorhandenen *Tags* auf. Für den nachfolgenden Commit mit *gbp import-orig* darf der Tag der zu importierenden Version noch nicht existieren (Tag-Kollision).

Bleiben die Überprüfungen befundlos, erfolgt der Import der neuen Version in das Git-Repository (Kapitel 34.4.11, Seite 243).

```
242a <CheckTags 242a>≡ (183)
function CheckTags {
    # Called by BuildWithUscan Import2Git and itself
    # checks git tags before executing gbp import-orig
    echo $(git tag) >> ${log}
    set +e
    cTags=$(git tag | grep --fixed-strings ${Version1})
    set -e
    if [ ${#cTags} -gt 0 ]
    then
        cTags1=$(echo ${cTags} | sed --expression='s/ /\n/g')
        if ! whiptail --title "List of dubious tags:" \
            --yesno "${cTags1}\n\nDo you want to continue regardless?" --defaultno \
            --yes-button "Yes" --no-button "No" 15 60
        <CheckTags2 242b>
```

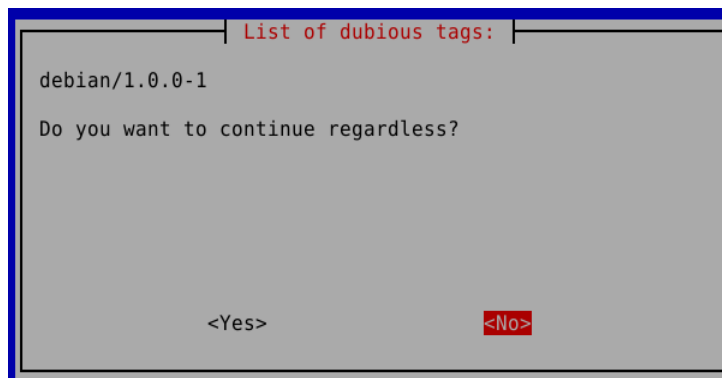


Abbildung 34.42.: Zweifelhafter Git-Tag

Wird hier ein Git-Tag aufgeführt, der vor dem Import einer neuen Version noch entfernt werden soll, lautet hier die Antwort „No“. Dann erfolgt im nächsten Dialog die Aufforderung, diesen Git-Tag zu entfernen.

```
242b <CheckTags2 242b>≡ (242a)
    then
        whiptail --title "Delete tags!" \
            --msgbox "Please delete tags in another terminal\n \
            and then press ok" 15 60
        <CheckTags3 243a>
```

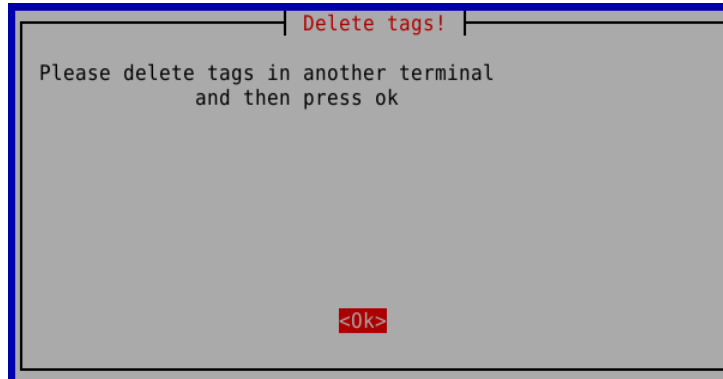



Abbildung 34.43.: Git Tags entfernen

```

243a  <CheckTags3 243a>≡ (242b)
        # git tag -d
        CheckTags
        fi
    fi
}

<Import2Git 241>

```

34.4.11. Import nach Git

Hier wird zum ersten Mal *gbp* eingesetzt. Als erster Schritt wird mit *gbp import-orig* der heruntergeladene Quellcode dem Git-Repository hinzugefügt. Der Option *-debian-branch* wird der Inhalt der Variable *RecentBranch* zugewiesen.

Existieren mehrere mögliche Branches, kann der *Debian-Branch*, in den importiert werden soll, zuvor ausgewählt werden (Kapitel 33.4, Seite 182).

```

243b  <BuildNewVersion15 243b>≡ (227b)
        Import2Git # Contains import to the git repo using gbp import
        Task=3 # Go to BuildNewRevision
    }

    <DebianFormatTemplate 260>

```

Die Funktion *Import2Git* (s. Kapitel 34.4.10, Seite 241) wird von der Funktion *BuildNewVersion* aufgerufen. Nach der Überprüfung des Git-Repositoriums wird der Zweig ermittelt, in den importiert werden soll.

```

244 <Import2Git1 244>≡ (241)
    # Check branch for import
    bl=$(git branch --list | sed --expression='s/* /x_/')
    ba=$(bl)
    set +e
    for element in ${ba[*]}
    do
        if echo ${element} | grep --quiet '^x_'
        then
            ActiveBranch=$(echo ${element} | sed --expression 's/\x_//')
        fi
    done
    set -e

    # After git init $ActiveBranch is always empty
    if [ ${ActiveBranch} ] && [ "${ActiveBranch}" != "${RecentBranch}" ]
    then
        whiptail --title "Check Branch!" \
        --msgbox "The active branch is ${ActiveBranch}.\n\
        In ${ConfigPath}${OrigName} 'RecentBranch' is ${RecentBranch}." \
        15 60
        echo -e "The active branch is "${ActiveBranch}".$n"\
        "In "${ConfigPath}${OrigName}" 'RecentBranch' is "${RecentBranch}".$n"
        FailureNotice
    fi

    # Import to the git repo using gbp import

    echo "Notice from BuildNewVersion: The branch is \
    ${RecentBranch}" >> ${log}
    whiptail --title "Notice from BuildNewVersion:" \
    --msgbox "The branch is ${RecentBranch}" 15 60
<Import2Git2 245>

```

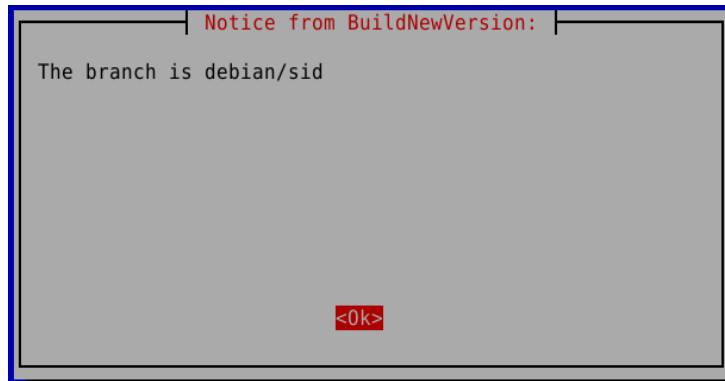


Abbildung 34.44.: Der Zweig is ...

```
245 <Import2Git2 245>≡ (244)  
    OrigFile=$(ls ${PrjPath}/${SourceName}_${Version1}.orig.tar.?z)  
    whiptail --title "Notice from BuildNewVersion:" \  
      --msgbox "The orig file to be imported is ${OrigFile}" 15 60  
<Import2Git3 246>
```

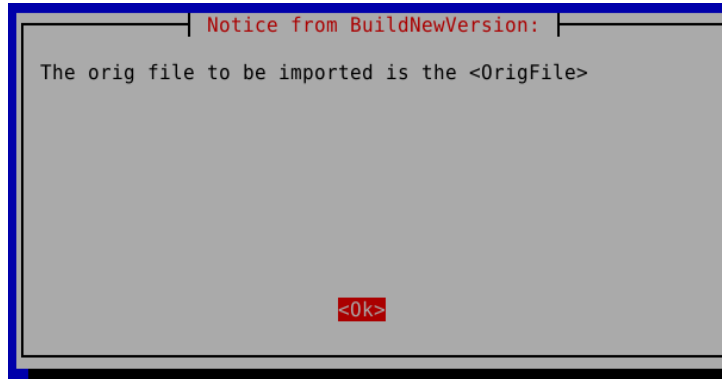


Abbildung 34.45.: Importiert wurde der Upstream-Code

Hier wird nun die Eingabe der Passphrase für den GPG-Schlüssel angefordert. Wird er nicht zeitnah eingegeben, wird der Import zurückgerollt. Es wird daher zunächst abgefragt, ob der GnuPG-Schlüssel zur Verfügung steht (Kapitel 32.8, Seite 172). Wird die Frage verneint, wird das Programm beendet.

Damit wird der zu erzeugende Git-Tag signiert. Dies entspricht guter Praxis. Nach *salsa.debian.org* sollten nur signierte Tags hochgeladen werden.

Das Signieren der Tags ist das Standardverhalten von *gbp import-orig*. Diese Option wurde gleichwohl in die Befehlszeile aufgenommen, denn „explizit ist besser als implizit“.

Will man ausnahmsweise mal nicht signieren, ist die Option *-no-sign-tags* zu verwenden.

246 `<Import2Git3 246>≡` (245)

```
GpgKeyAvailable
echo -e "\n Starting gbp import-orig - Please wait"'\n"
# Signing tags is default
gbp import-orig --verbose --debian-branch=${RecentBranch} \
--sign-tags ${OrigFile}
```

`<Import2Git4 247>`

Nun zeigt *gbp buildpackage* ausführlich alle Schritte, die nun ausgeführt werden. Dabei muss die ermittelte Paket-Version bestätigt oder angepasst werden.

```

247 <Import2Git4 247>≡ (246)
    if [ $? -eq 0 ]
    then
        echo "${OrigFile}" was imported by gbp import-orig" >> ${log}
    else
        whiptail --title "Something went wrong!" \
        --msgbox "gbp import-orig -v --debian-branch=${RecentBranch} \
        --sign-tags ${OrigFile} failed!" 15 60
        echo "gbp import-orig -v --debian-branch=${RecentBranch}" \
        --sign-tags "${OrigFile}" failed!" >> ${log}
        FailureNotice
        Import2Git
    fi
}

<BuildNewVersion 206>

```

Am Ende ruft die Funktion *BuildNewVersion* die Funktion *BuildNewRevision* auf (Kapitel 35, Seite 253)

34.5. Herunterladen und Importieren mit *uscan*

Die folgenden Schritte werden durch den Befehl *gbp import-orig --uscan ...* ausgeführt.

Anhand des ersten Eintrages in der Datei *debian/changelog* (Kapitel 37.1, Seite 321) ermittelt *uscan* die Versionsbezeichnung des zuletzt gebauten Paketes. *uscan* lädt dann die Web-Seite von der in der Datei *debian/watch* (Kapitel 35.4.7, Seite 269) angegebenen *URL*. Dann sucht *uscan* unter Verwendung des in *debian/watch* angegebenen Suchmusters nach Hyperlinks (*href*), die auf Upstream-Archive verweisen.

uscan lädt das Upstream-Archiv mit der neuesten Version herunter, wenn diese neuer als die in *debian/changelog* zuletzt angegebene Version ist. Das heruntergeladene Archiv wird im übergeordnete Verzeichnis gespeichert. Schließlich wird *mk-origtar.gz* (s. Kapitel 19.1.1, Seite 59) aufgerufen.

Im Programmskript wird zunächst geprüft, ob ein Herunterladen mit *uscan* möglich und sinnvoll ist.

Ferner wird eine vorhandene Datei *gbp.conf* (mit der Funktion *SearchGbpConf* (Kapitel 34.4.9, Seite 232)) zum Zwecke der Prüfung im Editor geöffnet. Es ist besonders auf die angegebene Kompression (*compression*) zu achten.

```
248 <BuildWithUscan 248>≡ (250a)
function BuildWithUscan {
    # Called by ClassicalOrUscan

    cd ${GitPath}

    echo "Try gbp import-orig --uscan" >> ${log}
    set +e
    uscaninfo=$(uscan --no-download --verbose)
    if [ ${#uscaninfo} -gt 0 ]
    then
        SearchGbpConf
        whiptail --title "uscan" --msgbox "${uscaninfo}" \
            --scrolltext 15 60
        echo -e "Result of uscan:\n"${uscaninfo} >> ${log}
        set +e
        echo ${uscaninfo} | grep '=> Package is up to date' > /dev/null
        if [ $? -eq 0 ]
        then
            whiptail --title "uscan" \
                --msgbox "Package seems to be up to date.\n \
                Nothing to do!" 15 60
            echo "Package seems to be up to date. Nothing to do!" \
                >> ${log}
        fi
    fi
}
<BuildWithUscan4 249a>
```

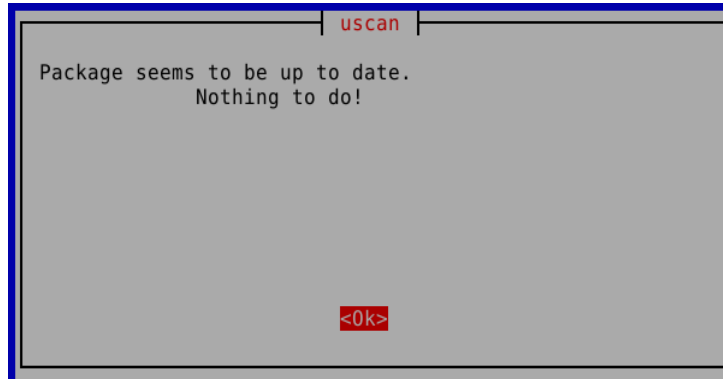


Abbildung 34.46.: Up to date

249a \langle BuildWithUscan4 249a $\rangle \equiv$ (248)

```
fi
echo ${uscaninfo} | grep '=> Newer package available from' \
> /dev/null
if [ $? -eq 0 ]
then
  if ! whiptail --title "Newer package available" \
  --yesno "All well? Continue?" --yes-button "Yes" \
  --no-button "Exit" 15 60
  then
    exit
  fi
fi
```

\langle BuildWithUscan5 249b \rangle

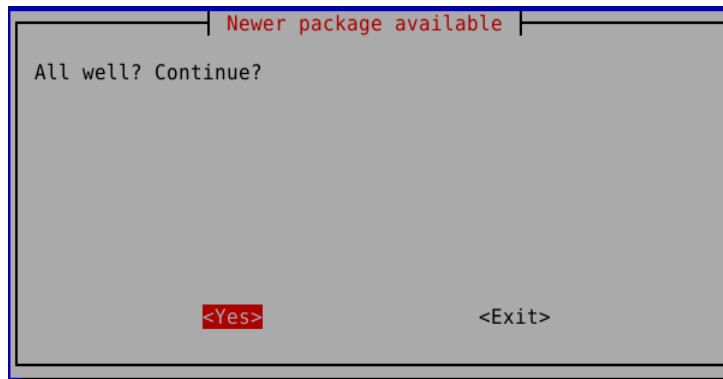


Abbildung 34.47.: Neue Version verfügbar

Es wird nun die Funktion *CheckRepackSuffix* ausgeführt.

249b \langle BuildWithUscan5 249b $\rangle \equiv$ (249a)

```
set -e
CheckRepackSuffix
set +e
```

\langle BuildWithUscan5-1 250b \rangle

Hier wird geprüft, ob in der Konfigurationsdatei der Eintrag *RecentRepackSuffix* vorhanden ist, aber kein entsprechender Eintrag in *debian/watch*.

In diesem Fall erfolgt ein Hinweis und die Datei *debian/watch* wird zum Editieren präsentiert.

```
250a <CheckRepackSuffix 250a>≡ (358)
function CheckRepackSuffix {
    # Called by BuildWithUscan
    if [ -n "${RecentRepackSuffix}" ]
    then
        if ! [ cat ${GitPath}/debian/watch | grep "repacksuffix=" > /dev/null ]
        then
            whiptail --title "debian/watch!" \
                --msgbox "No repacksuffix in debian/watch." 15 60
            nano --linenumbers --mouse --softwrap ${GitPath}/debian/watch
        fi
    fi
}

<BuildWithUscan 248>
```



Abbildung 34.48.: Kein Repack Suffix in debian/watch

Die Versionsbezeichnung der neuen Version wird durch einen Testlauf von *uscan* ermittelt.

Sodann wird die Funktion *CheckGitStatus* aufgerufen (Kapitel 33.4.1, Seite 183), um sicherzustellen, dass alle lokalen Änderungen committet wurden.

```
250b <BuildWithUscan5-1 250b>≡ (249b)
    Version1=$(uscan --no-download --verbose | \
    grep newversion | sed --expression 's/ $newversion = //' )
    set -e
    CheckGitStatus
<BuildWithUscan6 251a>
```


Zum Signieren des Archivs des heruntergeladenen Quellcodes wird der GPG-Schlüssel des Maintainers benötigt. Dieser muss daher zur Verfügung stehen. Sonst kann es nicht weitergehen. (Kapitel 32.8, Seite 172

```
251a <BuildWithUscan6 251a>≡ (250b)
      GpgKeyAvailable
      <BuildWithUscan7 251b>
```

Die eingangs beschriebenen Schritte werden durch *gbp import-orig --uscan* ausgeführt. Für Einzelheiten kann auf die Handbuchseite von *gbp import-orig* [41] verwiesen werden.

```
251b <BuildWithUscan7 251b>≡ (251a)
      CheckTags
      set +e
      # Downloads with uscan and imports with gbp import-orig
      gbp import-orig --uscan --verbose \
      --debian-branch=${RecentBranch} --sign-tags ${OrigFile}
      if [ $? -ne 0 ]
      then
          echo "Import with gbp import-orig --uscan failed!" \
          >> ${log}
          exit
      fi
      echo "Imported with gbp import-orig --uscan" >> ${log}
  else
      exit
  fi
  set -e
  else
      whiptail --title "Uscan failed!" \
      --msgbox "Please check the watch file with uscan." 15 60
      echo "Uscan failed! Please check the watch file with uscan." >> ${log}
      exit
  fi

  Task=3 # Go to BuildNewRevision
}

<PrepareUploading (nicht definiert)>
```

13. Oktober 2024

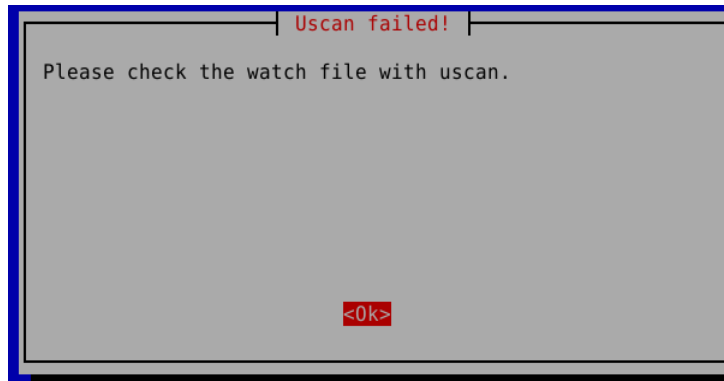


Abbildung 34.49.: Uscan kann Watch-Datei nicht parsen

Am Ende ruft die Funktion *BuildWithUscan* die Funktion *BuildNewRevision* auf.

35. Bauen einer neuen Revision

Die Funktion *BuildNewRevision* wird automatisch von den Funktionen *BuildNewVersion* und *BuildWithUsca*n aufgerufen.

Um das Bauen einer neuen Revision auf einen späteren Zeitpunkt verschieben zu können, wird die Möglichkeit gegeben, das Programm zuvor abzubrechen.

```
253a <BuildNewRevision2 253a>≡ (254)
      # Intro
      if ! whiptail --title "New Debian revision" \
        --yesno "A new Debian revision will be built." --yes-button "Yes" \
        --no-button "Exit" 15 60
      <BuildNewRevision3 253b>
```

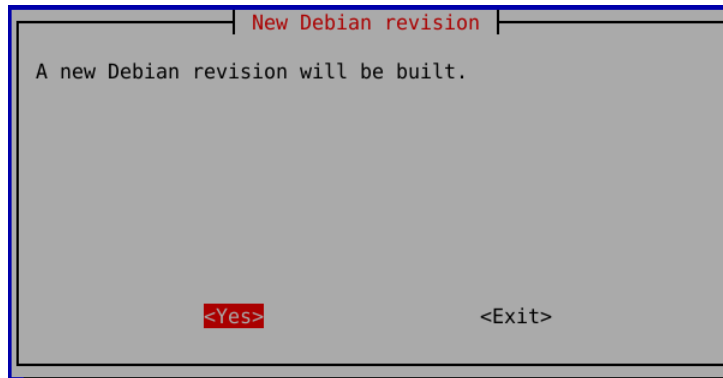


Abbildung 35.1.: Neue Revision bauen

```
253b <BuildNewRevision3 253b>≡ (253a)
      then
        exit
      fi

      echo "A new Debian revision will be built." >> ${log}
      <BuildNewRevision4 255>
```

Existiert das Verzeichnis *debian/source* und handelt es sich nicht um ein **Maven**-Paket geht es mit der Frage weiter, ob die Dateien im Verzeichnis *debian/* zum Editieren angezeigt werden sollen (Kapitel 35.3, Seite 256)

35.1. Anlegen des Debian-Verzeichnisses

Sofern das Verzeichnis *debian/source* (noch) nicht existiert, wird es vom Programmskript angelegt. Damit wird zugleich sichergestellt, dass auch das Verzeichnis *debian/* existiert (Kapitel 35.4, Seite 257).

Dies ist in der Regel nur dann relevant, wenn ein neues Paket für **Debian** gepackt werden soll.

```
254 <BuildNewRevision 254>≡ (338b)
function BuildNewRevision {
    # Called by TaskSelect
    cd ${GitPath}

    ## Generate directory if necessary
    echo $(pwd) >> ${log}
    if [ -d debian/source ]
    then
        echo "The directory debian/source in ${GitPath} \
        already exists." >> ${log}
        dfe=1
    else
        mkdir --parents debian/source
        echo "Directory debian/source was created" >> ${log}
        dfe=0
    fi
}

<BuildNewRevision2 253a>
```

Das Ergebnis der Ausführung des Programmskriptes wird in der Log-Datei vermerkt.

35.2. Abfrage: Bauen mit *mh-make*?

Sofern das Maven-Plugin (Kapitel 47, Seite 419) installiert ist und Maven als Build-System ausgewählt wurde, wird gefragt, ob bestimmte Dateien für dieses Build-System erstellt werden sollen. In der Regel kann diese Frage verneint werden.

255 *<BuildNewRevision4 255>*≡ (253b)

```
# For building java packages with maven

# To avoid an error, if 'MavenPluginFlag' is empty
if [ ! -z ${MavenPluginFlag} ] && [ ${MavenPluginFlag} -eq 1 ]
then
    if whiptail --title "Maven" \
    --yesno "Should mh_make create the ${PackName}.poms\n \
    file and some maven.* files?\n\n \
    Normally you only need it at the first run" --yes-button "Yes" \
    --no-button "No" --defaultno 15 60
    then
        . build-gbp-maven-plugin
        MakeMaven
    fi
fi
```

<BuildNewRevision5 256>

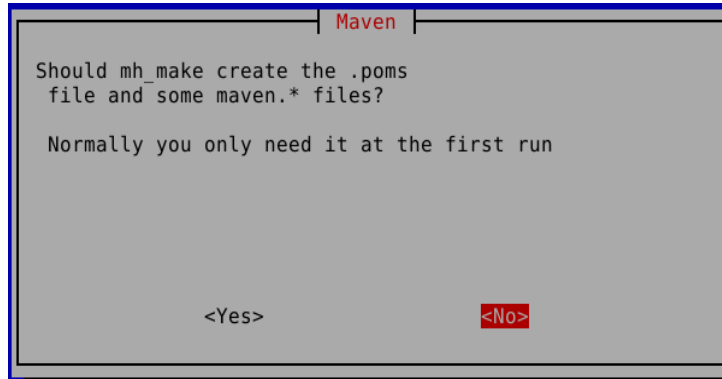


Abbildung 35.2.: Daten für Maven erstellen?

Wird sie bejaht, wird das `Maven`-Plugin geladen (Kapitel 47, Seite 419) und die Funktion `MakeMaven` aufgerufen. In Kapitel 47.3 (Seite 420) werden die weiteren Schritte beschrieben.

35.3. Sollen die Debian-Dateien angezeigt werden?

Hier wird nun die Möglichkeit eröffnet, die Dateien im Verzeichnis `debian/` zu erstellen und zu editieren.

```

256 <BuildNewRevision5 256>≡ (255)
    # Displaying files in debian/ for editing
    if [ ${dfe} -ne 1 ]
    then
        DisplayDebianFiles
    else
        if whiptail --title "Showing debian files for editing" \
        --yesno "Should the files of debian/ be displayed\n \
        to check, edit or create them?" --yes-button "Yes" \
        --no-button "No" 15 60
        then
            DisplayDebianFiles
        fi
    fi
    dfe=0

```

<BuildNewRevision5-1 285a>

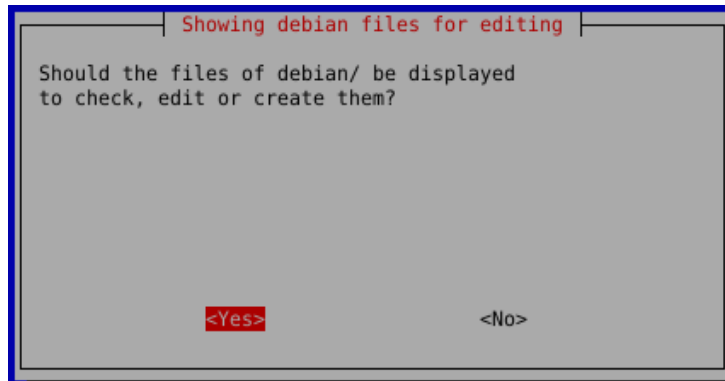


Abbildung 35.3.: Debian-Dateien anzeigen

Wird die Frage, ob die Dateien im Verzeichnis *debian/* angezeigt werden sollen, bejaht, werden diese Dateien, soweit notwendig, erstellt und zum möglichen Editieren angezeigt.

Andernfalls geht es mit Änderungen am Upstream-Code weiter (Kapitel 36, Seite 285) und es wird lediglich noch die Datei *debian/changelog* angezeigt (Kapitel 37.1, Seite 321).

35.4. Dateien im Verzeichnis *debian/*

Die Dateien im Verzeichnis *debian/* dienen der Steuerung und der Dokumentation des Build-Prozesses. Die Dateien in diesem Verzeichnis werden in der Archiv-Datei mit der Endung *<Paketname>.debian.tar.xz* veröffentlicht (s. Kapitel 8, Seite 23).

Das Verzeichnis *debian/* wird, wenn nötig, vom Programmskript erstellt (Kapitel 35.1, Seite 254).

Das Skript hat die Aufgabe, die notwendigen oder häufiger verwendeten Dateien im Verzeichnis *debian/* zu erstellen und - soweit möglich - Vorschläge für ihren Inhalt zu machen.

Existieren die Debian-Dateien bereits, kann der Nutzer sie prüfen und verbessern.

35.4.1. Anzeigen der Debian-Dateien

Sollen die Dateien im Verzeichnis *debian/* angezeigt werden, um sie zu prüfen, zu editieren oder auch zu erzeugen, wird die folgende Funktion ausgeführt:

```
257 <DisplayDebianFiles 257>≡ (283)
function DisplayDebianFiles {
    # Called by BuildNewRevision

    # Add Debian files

    # If the Debian files already exists, you can review and improve them now
    # If not, you have to write them

    ## There is default content for Debian files
    ## Change the default values, if you know, what you are doing

<DisplayDebianFiles0 258a>
```

Ist in der Konfigurationsdatei ein entsprechender Eintrag vorhanden, wird das Plugin-Script für *Webext*-Pakete geladen (Kapitel 48, Seite 433)

Das Plugin-Skript zeigt an, dass es geladen wurde (s. Kapitel 48.2.7, Seite 438).

```
258a <DisplayDebianFiles0 258a>≡ (257)
      # Loading Webext plugin or Python3 Plugin if needed

      if [ ${WebextFlag} -eq 1 ]
      then
        . build-gbp-webext-plugin
      fi
```

<DisplayDebianFiles01 258b>

Ist in der Konfigurationsdatei ein entsprechender Eintrag vorhanden, wird das Plugin-Script für *Python*-Pakete geladen (Kapitel 48, Seite 433)

Das Plugin-Skript zeigt an, dass es geladen wurde (s. Kapitel 49.0.1, Seite 439).

```
258b <DisplayDebianFiles01 258b>≡ (258a)
      if [ ${PythonFlag} -eq 1 ]
      then
        . build-gbp-python-plugin
      fi
```

<DisplayDebianFiles02 258c>

Es werden dann Funktionen aufgerufen, die gegebenenfalls Vorlagen für Dateien im Verzeichnis *debian/* erstellen.

```
258c <DisplayDebianFiles02 258c>≡ (258b)

      DebianFormatTemplate
      <DisplayDebianFiles1 258d>
```

Die Funktion *DebianFormatTemplate* erstellt eine Vorlage für die Datei *debian/source/format* (Kapitel 35.4.2, Seite 260).

```
258d <DisplayDebianFiles1 258d>≡ (258c)
      DebianUpstreamMetadataTemplate
      <DisplayDebianFiles2 258e>
```

Kapitel 35.4.4, Seite 262

```
258e <DisplayDebianFiles2 258e>≡ (258d)
      DebianCopyrightTemplate
      <DisplayDebianFiles3 258f>
```

```
258f <DisplayDebianFiles3 258f>≡ (258e)
      DebianControlTemplate
      <DisplayDebianFiles4 258g>
```

```
258g <DisplayDebianFiles4 258g>≡ (258f)
      DebianWatchTemplate
      <DisplayDebianFiles4-1 259a>
```


259a $\langle DisplayDebianFiles4-1 259a \rangle \equiv$ (258g)
 DebianRulesTemplate
 $\langle DisplayDebianFiles4-2 259b \rangle$

259b $\langle DisplayDebianFiles4-2 259b \rangle \equiv$ (259a)
 DebianSalsaCiTemplate
 $\langle DisplayDebianFiles5 278b \rangle$

Wenn ein Java-Paket ohne Build-System gebaut werden soll, kann es notwendig sein, eine Datei *debian/javabuild* zu erstellen.

Wenn das Maven-Plugin verwendet wird, werden die Maven-Dateien zum Editieren angezeigt (Kapitel 47.4, Seite 429).

259c $\langle DisplayDebianFiles10 259c \rangle \equiv$ (281)

```

if [ ${JavaFlag} -eq 1 ]
then
  if [ ${MavenPluginFlag} -eq 0 ]
  then
    DebianJavabuildTemplate
  else
    ls debian/ | grep 'maven'
    if [ $? -eq 0 ]
    then
      ShowMaven
    fi
  fi
fi
CmeFix
}

 $\langle ForceOrig 346b \rangle$ 

```

35.4.2. debian/source/format

Diese Datei enthält das Format des Quellpaketes. In der Datei *debian/source/format* wird der Eintrag *3.0 (quilt)* erstellt. Dies bedeutet, dass es sich um *kein* natives Paket handelt. Bei einem nativen Paket muss *3.0 (native)* eingetragen werden.

Was ein natives Paket ist, wird in Kapitel 16.1 (Seite 53) und in Kapitel 4 der Debian-Policy [7] beschrieben.

Eine ausführliche Beschreibung zur Datei *debian/source/format* gibt es im *Debian-Leitfaden für Neue Paketbetreuer*[11] in Kapitel 5.22 ¹

```
260 <DebianFormatTemplate 260>≡ (243b)
function DebianFormatTemplate {
    # Called by DisplayDebianFiles

    if ! [ -f ${GitPath}/debian/source/format ]
    then
        # String for debian/source/format
        if whiptail --title "Kind of package" \
            --defaultno --yesno "Is it a native debian package?" \
            --yes-button "Yes" --no-button "No" 15 60

    <DebianFormatTemplate1 261>
```

¹<https://3ws.debian.org/doc/manuals/maint-guide/dother.de.html#sourcef>

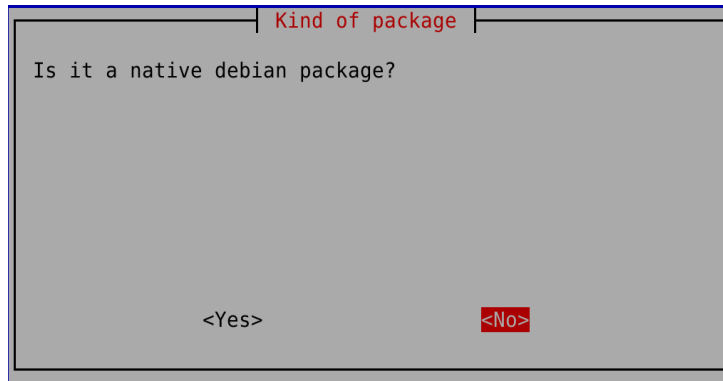


Abbildung 35.4.: Art des Quellpaketes (Nativ? J/N)

```

261  <DebianFormatTemplate1 261>≡ (260)
      then
          str4format="3.0 (native)"
      else
          str4format="3.0 (quilt)"
      fi

      touch ${GitPath}/debian/source/format
      echo ${str4format} >> ${GitPath}/debian/source/format
      echo "/debian/source/format was created." >> ${log}
  fi
  nano --linenumbers --mouse --softwrap ${GitPath}/debian/source/format
}

<DebianUpstreamMetadataTemplate 262>

```

35.4.3. debian/source/include.binaries

Grundsätzlich sollen Binär-Dateien keine Aufnahme in das Debian-Paket finden. Sie sind daher regelmäßig von der Aufnahme in das *.orig-Archiv auszuschließen (Kapitel 34.4.5, Seite 217).

Es gibt jedoch Ausnahmen. Hierzu können Medien-Dateien und komprimierte Dokumentationen zählen. Diese Dateien sind zwecks Dokumentation in der Datei *debian/source/include-binaries* unter Angabe ihres Pfades aufzuführen.

35.4.4. debian/upstream/metadata

Es gibt eine ausführliche Beschreibung^[42] dieser *YAML*-Datei in englischer Sprache. Dort wird auch auf DEP-12² hingewiesen.

Dort wird auch erläutert, welche Informationen in den einzelnen Zeilen vom Maintainer eingetragen werden sollen, sofern diese Informationen vorhanden sind.

Es wird empfohlen, beim Editieren der Datei die Kommentare zu entfernen.

```
262 <DebianUpstreamMetadataTemplate 262>≡ (261)
function DebianUpstreamMetadataTemplate {
    # Called by DisplayDebianFiles

    # Treatment for native packages
    set +e
    cat debian/source/format | grep "native" > /dev/null
    if [ $? -eq 0 ]
    then
        set -e
        return
    fi
    set -e

    # Strings for debian/upstream/metadata
    if ! [ -f ${GitPath}/debian/upstream/metadata ]
    then
        mkdir --parents debian/upstream
        # creating a template for debian/upstream/metadata
        echo -e "# You can find a description at\n#\n\
https://wiki.debian.org/UpstreamMetadata" \
        >> debian/upstream/metadata
        echo "# Archive: " >> debian/upstream/metadata
        echo "# ASCL-id: " >> debian/upstream/metadata
        echo "Bug-Database: " >> debian/upstream/metadata
        echo "Bug-Submit: " >> debian/upstream/metadata
        echo "# Cite-As: " >> debian/upstream/metadata
        echo "Changelog: " >> debian/upstream/metadata
        echo "# CPE: " >> debian/upstream/metadata
        echo "Documentation: " >> debian/upstream/metadata
        echo "# Donation: " >> debian/upstream/metadata
        echo "# FAQ: " >> debian/upstream/metadata
        echo "# Funding: " >> debian/upstream/metadata
```

²Quelle:[43]

```

echo "# Gallery: " >> debian/upstream/metadata
echo "# Other-References: " >> debian/upstream/metadata
echo "# Reference: " >> debian/upstream/metadata
echo "#     Author: " >> debian/upstream/metadata
echo "#     Booktitle: " >> debian/upstream/metadata
echo "#     DOI: " >> debian/upstream/metadata
echo "#     Editor: " >> debian/upstream/metadata
echo "#     Eprint: " >> debian/upstream/metadata
echo "#     ISBN: " >> debian/upstream/metadata
echo "#     ISSN: " >> debian/upstream/metadata
echo "#     Journal: " >> debian/upstream/metadata
echo "#     Number: " >> debian/upstream/metadata
echo "#     Pages: " >> debian/upstream/metadata
echo "#     PMID: " >> debian/upstream/metadata
echo "#     Publisher: " >> debian/upstream/metadata
echo "#     Title: " >> debian/upstream/metadata
echo "#     Type: " >> debian/upstream/metadata
echo "#     URL: " >> debian/upstream/metadata
echo "#     Volume: " >> debian/upstream/metadata
echo "#     Year: " >> debian/upstream/metadata
echo "#     Debian-package: " >> debian/upstream/metadata
echo "# Registration: " >> debian/upstream/metadata
echo "# Registry: " >> debian/upstream/metadata
echo "# Repository: " >> debian/upstream/metadata
echo "# Repository-Browse: " >> debian/upstream/metadata
echo "# Screenshots: " >> debian/upstream/metadata
echo "# Security-Contact: " >> debian/upstream/metadata
echo "# Webservice: " >> debian/upstream/metadata
fi
nano --linenumbers --mouse --softwrap ${GitPath}/debian/upstream/metadata
}

```

<DebianCopyrightTemplate 264a>

Folgende Felder sind für viele Pakete relevant. Ihr Vorhandensein wird teilweise von *lintian* geprüft.

Bug-Database URL zur Liste der bekannten Fehler

Bug-Submit Adresse, an die Fehlermeldungen gesandt werden können.

Changelog URL des Upstream Changelogs

Documentation Upstream Dokumentation

Repository URL zum Upstream-Repositorium

Repository-Browse Durchsuchbares Repositorium von Upstream

35.4.5. debian/copyright

Diese Datei enthält Informationen über das Copyright und die Lizenzen der Quellen der Originalautoren.

Diese Datei kann mit *debmake -cc* erzeugt werden und im DEP-5-Format [19] abgelegt

264a `<DebianCopyrightTemplate 264a>≡ (262)`

```
function DebianCopyrightTemplate {
    # Called by DisplayDebianFiles

    if ! [ -f ${GitPath}/debian/copyright ]
    then
        # creating debian/copyright using debmake
        debmake -cc > debian/copyright
    <DebianCopyrightTemplate2 264b>
}
```

Auszug aus der Manpage für *debmake*

```
-c, --copyright
    scan source for copyright+license text and exit.
```

- `-c`: simple output style
- `-cc`: normal output style (similar to the `debian/copyright` file)
- `-ccc`: debug output style

Danach muss die Datei noch bearbeitet werden. Dateien mit gleichem Autor und gleicher Lizenz können zusammengefasst werden. Stehen Dateien unter mehreren Lizenzen, werden diese Lizenzen mit *or* verbunden.

264b `<DebianCopyrightTemplate2 264b>≡ (264a)`

```
fi
    nano --linenumbers --mouse --softwrap ${GitPath}/debian/copyright
}

<TeamMaintainer 147>
```

Jeder Abschnitt *Files* in der maschinenlesbaren Copyright-Datei muss auf eine Lizenz verweisen, für die jeweils ein eigenständiger Lizenzabsatz existiert. Diese Absätze müssen **nach** allen *Files*-Absätzen erscheinen.

Eigenständige Lizenzabsätze können verwendet werden, um den vollständigen Lizenztext für eine bestimmte Lizenz nur einmal bereitzustellen, anstatt ihn in jedem *Files*-Abschnitt zu wiederholen, der auf sie verweist.[44]

Steht der Lizenztext unter `/usr/share/common-licenses/` zur Verfügung, ist statt des kompletten Lizenztextes eine Kurzfassung und der Dateiname nebst Pfad der Datei, die den Lizenztext enthält, (beispielsweise `/usr/share/common-licenses/GPL-3`) aufzuführen.

35.4.6. *debian/control*

Diese Datei enthält essentielle Werte, die durch die Paketverwaltungswerkzeuge verwendet werden.

Das Paketverwaltungssystem verarbeitet Daten, die in einem gemeinsamen Format, den sogenannten Kontrolldaten, in der *control*-Datei gespeichert ist. Diese Daten werden für Quellpakete, Binärpakete und die **.changes*-Dateien verwendet, die die Installation der hochgeladenen Dateien steuern.

Einzelheiten werden in der *Debian-Policy*[7] beschrieben.

35.4.6.1. Grundlegender Aufbau

Das Programmskript erzeugt ein „Grundgerüst“ der *control*-Datei für das Quellpaket. Die *control*-Datei des Binärpaketes und der *.changes*-Datei werden durch den Build-Prozess aus den Informationen des Abschnittes für die Binär-Datei(en) erstellt.

```
265a <DebianControlTemplate 265a>≡ (153)
function DebianControlTemplate {
    # Called by DisplayDebianFiles

    # Strings for debian/control
    str4versiondebhelpers="(=13)"
<DebianControlTemplate1 265b>
```

Ab der Version *debhelper* ≥ 12 wird die *Kompatibilitätsversion* nicht mehr in der Datei *debian/compat* zusätzlich gepflegt. Stattdessen wird in der Datei *debian/control* der Eintrag *debhelper* durch *debhelper-compat* mit der Version (= 13) ersetzt.³ Dies gilt auch für alle darauf folgenden Versionen.

Von der Verwendung der Version 11 wird bereits abgeraten.

```
265b <DebianControlTemplate1 265b>≡ (265a)
    str4standardsversion="4.7.0"

<DebianControlTemplate2 266a>
```

³https://release.debian.org/bookworm/freeze_policy.html (2023)

In der Datei *debian/control* muss es einen Eintrag „Standard-Version: “ geben, der die Konformität zur Version der Debian-Policy (s. Kapitel 7.2, Seite 21) angibt. Hier wird die jeweils aktuelle Version (*jetzt: 4.6.1*) vorgegeben.

An dieser Stelle im Programmskript wird eine Vorlage für die Datei *debian/control* erstellt, wenn die Datei noch nicht existiert.

```
266a  <DebianControlTemplate2 266a>≡ (265b)
      if ! [ -f ${GitPath}/debian/control ]
      then
          # creating a template for debian/control
          echo -e "Source: "${SourceName} > debian/control
          echo -e "Priority: optional" >> debian/control
```

<DebianControlTemplate3 267b>

Name und E-Mail-Adresse von **Maintainer** und gegebenenfalls **Uploaders** werden durch die Funktion *DEBValues* ermittelt. (Kapitel 32.4.2, Seite 145)

```
266b  <DebianControlTemplate4 266b>≡ (267b)
      DEBValues

      if [ -n "${Maintainer}" ]
      then
          echo -e "Maintainer: "${Maintainer} >> debian/control
      else
          echo "Maintainer: " >> debian/control
      fi

      if [ -n "${Uploaders}" ]
      then
          echo -e "Uploaders: "${Uploaders} >> debian/control
      fi
```

<DebianControlTemplate5 266c>

```
266c  <DebianControlTemplate5 266c>≡ (266b)
      echo -e "Build-Depends: debhelper-compat" \
          ${str4versiondebhelpers} >> debian/control
```

<DebianControlTemplate6 267c>

```
266d  <DebianControlTemplate7 266d>≡ (267c)
      echo -e "Standards-Version: "${str4standardsversion} \
          >> debian/control
      echo -e "Rules-Requires-Root: no" >> debian/control
      echo -e "Vcs-Git: https://salsa.debian.org/"${SalsaName} \
          >> debian/control
      BrowserName=$(echo ${SalsaName} | sed --expression='s/.git$//g')
      echo -e "Vcs-Browser: https://salsa.debian.org/"${BrowserName} \
          >> debian/control
      echo -e "Homepage: \n" >> debian/control
```

<DebianControlTemplate8 267a>

Nun folgt in der Datei *debian/control* die Informationen über das Binär-Paket.

```
267a <DebianControlTemplate8 267a>≡ (266d)
      echo -e "Package: "${PackName} >> debian/control
      echo -e "Architecture: all" >> debian/control
      echo -e "Depends: \${misc:Depends}" >> debian/control
      echo -e "Description: " >> debian/control
      echo "A template for debian/control was created." >> ${log}
<DebianControlTemplate9 268a>
```

35.4.6.2. Anpassungen für Java-Pakete

Informationen aus dieser Datei werden bei der Erstellung der *control*-Datei des Binär-Paketes verwendet.

Für das Paketieren von Java-Paketen können schon folgende Einträge vorgenommen werden.

```
267b <DebianControlTemplate3 267b>≡ (266a)
      if [ ${JavaFlag} -eq 1 ]
      then
          echo -e "Section: java" >> debian/control
      else
          echo -e "Section:" >> debian/control
      fi
<DebianControlTemplate4 266b>
```

Bei Paketen, die im Team betreut werden, wird hier die Adresse des Teams angegeben. Dies ist in der Regel die E-Mail-Adresse der Mailingliste. In diesen Fällen ist auch das Feld **Uploaders** mit den Namen der Paketbetreuer zu füllen.

Maintainer für das Java-Team ist beispielsweise *Debian Java maintainers* <pkg-java-maintainers@lists.aliases.debian.org>.

Damit der Paketierer nicht immer seinen vollen Namen und seine E-Mail-Adresse schreiben muss, sucht das Skript diese Daten zunächst in der Konfigurationsdatei (Kapitel 32.4.2, Seite 145).

```
267c <DebianControlTemplate6 267c>≡ (266c)
      if [ ${JavaFlag} -eq 1 ]
      then
          echo " , default-jdk" >> debian/control

          if [ ${MavenPluginFlag} -eq 1 ]
          then
              echo " , maven-debian-helper" >> debian/control
          fi
      fi
<DebianControlTemplate7 266d>
```

35.4.6.3. Web-Extension-Plugin

Aufruf der Funktion zur Anpassung der Datei *debian/control* für Mozilla-AddOns. Diese Funktion befindet sich im *Webext*-Plugin (Kapitel 48.2.3, Seite 437)

```
268a  <DebianControlTemplate9 268a>≡ (267a)
        if [ ${WebextFlag} -eq 1 ]
        then
            WebextControl
        fi
    <DebianControlTemplate10 268b>
```

35.4.6.4. Python-Plugin

Aufruf der Funktion zur Anpassung der Datei *debian/control* für Python-Pakete und -Bibliotheken. Diese Funktion befindet sich im *Python*-Plugin (Kapitel 49.2, Seite 441)

```
268b  <DebianControlTemplate10 268b>≡ (268a)
        if [ ${PythonFlag} -eq 1 ]
        then
            PythonControl
        fi
    <DebianControlTemplate11 268c>
```

```
268c  <DebianControlTemplate11 268c>≡ (268b)
        fi
        nano --linenumbers --mouse --softwrap ${GitPath}/debian/control
    }

    <OptionsWatchFile 270a>
```

35.4.7. debian/watch

Die Datei *watch* im Debian-Verzeichnis enthält Daten für das Programm *uscan* (Kapitel 34.5, Seite 248 und Kapitel 40.4, Seite 362).

Zur Bestimmung des Namens des Quellcodepaketes liest *uscan* den ersten Eintrag in der Datei *debian/changelog* (Kapitel 37.1, Seite 321). Anhand dieses Eintrages ermittelt *uscan* auch die Versionsbezeichnung des zuletzt gebauten Paketes [45].

Dann verarbeitet *uscan* die Zeilen der Datei *debian/watch* in einem Zuge von oben nach unten. Einzelheiten zur Datei *debian/watch* werden im entsprechenden Artikel im Debian-Wiki beschrieben⁴.

In dieser Datei können reguläre Ausdrücke im Perl-Format⁵ verwendet werden. Zeilen, die mit einem *#* beginnen, werden als Kommentarzeilen ignoriert.

Eingangs dieser Datei steht die Version des verwendeten Formats. Diese Angabe ist erforderlich. Die empfohlene Versionsnummer ist die 4 und wird vom Skript schon bei der Erstellung der Datei eingetragen.

```
269 <DebianWatchTemplate 269>≡ (272b)
function DebianWatchTemplate {
    # Called by DisplayDebianFiles

    # Treatment for native packages
    set +e
    cat debian/source/format | grep "native" > /dev/null
    if [ $? -eq 0 ]
    then
        set -e
        return
    fi
    set -e

    # String for debian/watch
    str4watch="version=4"

    if ! [ -f ${GitPath}/debian/watch ]
    then
        # creating a template for debian/watch
        echo ${str4watch} > debian/watch
        OptionsWatchFile
    <DebianWatchTemplate3 273>
```

⁴<https://wiki.debian.org/debian/watch>

⁵siehe hierzu:

- https://de.wikibooks.org/wiki/Perl-Programmierung:_Reguläre_Ausdrücke
- <http://www.mathe2.uni-bayreuth.de/perl/GK/regExp.htm>
- http://perl-seiten.privat.t-online.de/html/perl_reg.html

Nun werden die Optionen definiert, wie *uscan* überprüfen kann, ob auch die aktuelle Version gebaut wird.

Die Auswertung der Versionierung folgt der Darstellung in Kapitel 11 (Seite 35).

Die Optionen geben Regeln für die Auswahl möglicher Upstream.Archive vor. Sie werden in der Handbuchseite (Manpage) von *uscan* erläutert ⁶.

Das Programmskript stellt die Optionen auf der Basis der bisher vorhandenen Informationen in die Datei *debian/watch* ein.

Das Programmskript vermeidet Leerzeichen in der Liste der Optionen. Andernfalls muss die Optionenliste von doppelten Anführungszeichen (") eingerahmt werden.

```
270a  <OptionsWatchFile 270a>≡ (268c)
      function OptionsWatchFile {
          # Called by DebianWatchfile

          olf='\\n'
          WOpt='opts='
```

<OptionsWatchFile1 270b>

Bei die Erstellung der **.orig.tar.**-Datei (Kapitel 34.4.1, Seite 206) wurde bereits das ursprüngliche Archiv-Format festgestellt. Diese Information wird nun für die Datei *debian/watch* verwendet.

Die folgende Option legt beim Bauen einer neuen Version mittels *uscan* fest, dass die **.orig.tar.**-Datei in einem anderen Kompressionsformat archiviert wird, als das herunterzuladende Upstream-Archiv. Angegeben wird hier als Kompressionsformat des orig-Archives *xz*.

Wenn das Upstream-Archiv in einem Zip-Format (einschließlich *.xpi*, *.jar* oder *.oxz*) vorliegt, muss nämlich eine Neupaketierung vorgenommen werden. Die Option *compression=xz* legt fest, dass ein **.orig.tar.xz* gebildet wird. Die Option *repack* ist in diesem Fall entbehrlich; aber explizit ist besser als implizit.

```
270b  <OptionsWatchFile1 270b>≡ (270a)
      # Repacked <UpstreamPackage>.zip
      RepackFlag=0
      ZipSuffix=(.zip .oxz .xpi .jar)
      if [[ ${ZipSuffix} =~ ${RecentUpstreamSuffix} ]]
      then
          WOpt=${WOpt}${olf}'repack,compression=xz,'
          RepackFlag=1
      fi
```

<OptionsWatchFile2 271a>

⁶<https://people.debian.org/~osamu/uscan.html#WATCH-FILE-OPTIONS>

Muss eine Neupaketierung vorgenommen werden, weil aus dem Quellcode-Archiv Dateien zu entfernen sind, sollte dies im Namen der **.orig.tar.**-Datei ersichtlich sein. Hierzu dient die Option *repacksuffix*. Die auszuschließenden Dateien ergeben sich aus der entsprechenden Liste (*Files-Excluded*) in der Datei *debian/copyright*.

```
271a <OptionsWatchFile2 271a>≡ (270b)
      # Excluded files
      if [ -z ${RecentRepackSuffix} ]
      then
        if [ ${RepackFlag} -eq 1 ]
        then
          WOpt=${WOpt}${olf}'repacksuffix='${RecentRepackSuffix}',\\n'
        else
          WOpt=${WOpt}${olf}'repack,compression=xz,\\n\
repacksuffix='${RecentRepackSuffix}',\\n'
        fi
```

<OptionsWatchFile3 271b>

Als Nächstes folgt die Option *dversionmangle*. Hiermit wird die letzte gefundene Upstream-Versionsbezeichnung in der Datei *debian/changelog* normalisiert, um sie mit der Version des verfügbaren Upstream-Archivs zu vergleichen. Dazu werden die Debian-spezifischen Suffixes wie *+dfsg* oder *+ds* im Wege der Ersetzung entfernt.

```
271b <OptionsWatchFile3 271b>≡ (271a)
      WOpt=${WOpt}${olf}'dversionmangle=s/'${RecentRepackSuffix}'//,'
      fi
```

<OptionsWatchFile4 271c>

Mit der Option *uversionmangle* werden die Zeichenketten der Dateien der Upstream-Versionen normalisiert, die aus den Links auf diese Dateien im Quellcode der Webseite extrahiert werden. In der Versionsbezeichnung werden die nicht-numerischen Zeichen (außer dem Punkt) zwecks Vereinheitlichung sinnvoll ersetzt. Damit wird die Versionsbezeichnung des Upstream-Archives versionierungsschemakonform (Kapitel 11.2, Seite 35) gestaltet. Dies wird als Versionsortierungsindex bei der Auswahl der neuesten Upstream-Version verwendet.

```
271c <OptionsWatchFile4 271c>≡ (271b)
      # For beta-, rc- etc. releases

      WOpt=${WOpt}${olf}'uversionmangle=s/-?([\d.]+)/~$1/;tr/A-Z/a-z/,'
<OptionsWatchFile5 272a>
```

filenamemangle generiert den Upstream-Tarball-Dateinamen aus der ausgewählten *href*-Zeichenkette, wenn die Vergleichsmuster die neueste Upstream-Version aus der ausgewählten *href*-Zeichenkette extrahieren können. Andernfalls wird der Upstream-Tarball-Dateiname aus seiner vollständigen URL-Zeichenkette erzeugt und die fehlende Upstream-Version aus dem erzeugten Upstream-Tarball-Dateinamen eingesetzt.

Ohne diese Option wird der standardmäßige Upstream-Tarball-Dateiname generiert, indem die letzte Komponente der URL genommen und alles nach einem '?' oder '#' entfernt wird.

```
272a <OptionsWatchFile5 272a>≡ (271c)
    # For packages from Github
    set +e
    if echo ${DownloadUrl} | grep "github" > /dev/null
    then
        WOpt=${WOpt}${olf}'filenamemangle=s/.+\/v?(\d\S+)\.*/$1/,'
    fi
    set -e
```

<OptionsWatchFile8 272b>

```
272b <OptionsWatchFile8 272b>≡ (272a)
    # If there are no options
    if [ $#WOpt -eq 6 ]
    then
        WOpt='# '${WOpt}
        WOpt=$(echo ${WOpt} | sed 's/\\/\\/\'')
    fi

    echo -e ${WOpt} >> debian/watch
}
```

<DebianWatchTemplate 269>

uscan lädt die Web-Seite von der in *debian/watch* angegebenen *URL*. Dann sucht *uscan* unter Verwendung des in *debian/watch* angegebenen Suchmusters nach Hyperlinks (*hrefs*), die auf Upstream-Archive verweisen.

Ausgehend von der *URL*, mit der der Quellcode heruntergeladen wurde, wird im Programmskript definiert, wie nach einer neuen Version gesucht werden kann. Die *URL* wurde der Variablen *DownloadURL* als Wert zugewiesen (Kapitel 34.4.1, Seite 206).

```
273 <DebianWatchTemplate3 273>≡ (269)
    set +e
    if echo ${DownloadUrl} | grep "github" > /dev/null
    then
        DownloadUrl=$(echo ${DownloadUrl} | \
            sed --expression 's/archive.*$/releases/')
        DownloadUrl=${DownloadUrl}' ./v?(\d\S+)\.tar\.gz'
        echo -e ${DownloadUrl} >> debian/watch
    fi
    set -e
    echo "A template for debian/watch was created." >> ${log}
    whiptail --title "Edit debian/watch!" \
        --msgbox "Please insert reasonable regular expressions\n \
            into debian/watch!" 15 60
    fi
    nano --linenumbers --mouse --softwrap ${GitPath}/debian/watch
}
```

<DebianRulesTemplate 274a>

Ein Beispiel:

```
opts=repack,compression=xz,dversionmangle=s/\+dfsg$//,\
uversionmangle=s/-Beta/~beta/;s/-rc/~rc/,\
filenamemangle=s/.*\v?(\d+\.\d+\.\d+(?:-(Beta|rc)\d+)?)\.tar\
.gz/jax-maven-plugin-$1.tar.gz/ \
https://github.com/davidmoten/jax-maven-plugin/releases .*v?(\d\S+)\.tar\.gz
```

Die Datei *debian/watch* kann, wenn man in dem Verzeichnis ist, in dem sich das Git-Repository befindet, mit dem Befehl *uscan -no-download -debug* geprüft werden. Die Option *-no-download* bewirkt, dass ein gefundenes, neueres Upstream-Archiv nicht heruntergeladen wird. Die Option *-debug* erzeugt einen für Menschen lesbaren Bericht, der auch den Status der internen Variablen anzeigt.

35.4.8. *debian/rules* - Grundlegender Aufbau

Diese Datei steuert den Ablauf des Buildprozesses.

Im Unterschied zu den anderen Dateien im Verzeichnis *debian* ist die Datei *debian/rules* als ausführbar zu kennzeichnen.

Wie jedes andere *Makefile* ist die Datei *debian/rules* aus mehreren Regeln zusammengesetzt, welche das Ziel definieren und wie diese Regeln ausgeführt werden. In der Debian-Policy, Kapitel 4.9[7] *Main building script: debian/rules* werden die Details erklärt.

35.4.8.1. Erstellen der Datei

Existiert die Datei *debian/rules* noch nicht, wird sie angelegt.

Sie ist ein Makefile und hat daher eine entsprechende *Shebang* (*#!/usr/bin/make -f*).

```
274a  <DebianRulesTemplate 274a>≡ (273)
function DebianRulesTemplate {
    # Called by DisplayDebianFiles

    # Strings for debian/rules
    str4rules="#!/usr/bin/make -f\n# -*- makefile -*-\n"\
    str4rulesdh="%\n\tdh \@%\n\n"

    if ! [ -f ${GitPath}/debian/rules ]
    then
        touch ${GitPath}/debian/rules
        echo -e ${str4rules} >> ${GitPath}/debian/rules
    <DebianRulesTemplate1 274b>
```

35.4.8.2. Export von Variablen

Es wird durch den Export der Variablen *DH_VERBOSE* und *DH_OPTIONS*, sowie der Zuweisung entsprechender Werte die umfassendere Ausgabe der Meldungen eingeschaltet.

```
274b  <DebianRulesTemplate1 274b>≡ (274a)
    echo -e "# Uncomment this to turn on verbose mode.\n" \
    >> ${GitPath}/debian/rules
    echo -e "export DH_VERBOSE=1\nexport DH_OPTIONS=-v\n" \
    >> ${GitPath}/debian/rules
```

<DebianRulesTemplate2 274c>

Zusätzlich werden für verschiedene Pakettypen ergänzende Variablen exportiert. Diese werden in den jeweiligen Plugins definiert. Die Beschreibung für Java-Pakete findet sich in Kapitel 46.1 (Seite 417). Die Anpassung für Java-Pakete erfolgt durch die Funktion *Rules4Java*.

```
274c  <DebianRulesTemplate2 274c>≡ (274b)
    if [ ${JavaFlag} -eq 1 ]
    then
        Rules4Java
    fi
```

<DebianRulesTemplate3 275a>

Für das Bauen der **Mozilla**-Erweiterungen werden auch zusätzliche Einträge benötigt. Die Beschreibung der Besonderheiten für die **Mozilla**-AddOns findet sich in Kapitel 48.2.2 (Seite 435).

Die Anpassung für *Webext*-Pakete erfolgt durch die Funktion *WebextRules* im entsprechenden Plugin (Kapitel 48 Seite 433).

```
275a <DebianRulesTemplate3 275a>≡ (274c)
      if [ ${WebextFlag} -eq 1 ]
      then
          WebextRules
      fi
```

<DebianRulesTemplate4 275b>

Entsprechendes Gilt auch für die **Python**-Pakete (Kapitel 49.1 Seite 440).

```
275b <DebianRulesTemplate4 275b>≡ (275a)
      if [ ${PythonFlag} -eq 1 ]
      then
          PythonRules
      fi
```

<DebianRulesTemplate5 275c>

35.4.8.3. Aufruf der Debhelper

```
275c <DebianRulesTemplate5 275c>≡ (275b)
      echo -e ${str4rulesdh} >> ${GitPath}/debian/rules
```

<DebianRulesTemplate6 275d>

Für **Maven**-Pakete muss der Aufruf der *debhelper* ergänzt werden. Dies erfolgt durch die Funktion *Rules3MavenDH* des **Maven**-Plugins (Kapitel 47.5 Seite 432).

```
275d <DebianRulesTemplate6 275d>≡ (275c)
      if [ ${MavenPluginFlag} -eq 1 ]
      then
          Rules4MavenDH
      fi
```

<DebianRulesTemplate7 275e>

Auch für die **Mozilla**-Erweiterungen muss der Aufruf der *debhelper* ergänzt werden. Dies erfolgt durch das *Web-Extension-Plugin* (Kapitel 48, Seite 433).

```
275e <DebianRulesTemplate7 275e>≡ (275d)
      if [ ${WebextFlag} -eq 1 ]
      then
          WebextRulesDH
      fi
```

<DebianRulesTemplate10 276a>

Auch beim Paketieren von Python-Paketen ist der Aufruf der Debhelper in der Datei `debian/rules` zu ergänzen.

```
276a <DebianRulesTemplate10 276a>≡ (275e)
      if [ ${PythonFlag} -eq 1 ]
      then
          PythonRulesDH
      fi

<DebianRulesTemplate11 276b>
```

35.4.8.4. `debian/rules` - overrides

Manchmal ist es notwendig, vor oder nach der Ausführung der jeweiligen `debhelper`-Skripte, weitere Schritte auszuführen. Dazu wird das jeweilige Skript übersteuert.

Beispiel:

```
override_dh_auto_build:
    dh_auto_build -- -f org/xmlunit/pom.xml package -DskipTests
```

In der ersten Zeile des Beispiels wird das Ziel genannt, welches modifiziert werden soll. Die zweite Zeile muss mit einem *Tabulatorzeichen* in einer Breite von 8 Zeichen beginnen.

Das doppelte Minuszeichen bedeutet, dass zunächst die Parameter ausgeführt werden, die `dh_auto_build` normalerweise übergibt. Danach können weitere Parameter aufgeführt werden, die an das Programm übergeben werden.

`package` bezeichnet das sogenannte *goal*.

35.4.8.5. Schluss der Funktion

Am Ende der Funktion erfolgt ein Eintrag in die Log-Datei. Die Datei `debian/rules` wird zum Editieren angezeigt. Schließlich wird sie als *Make*-Datei ausführbar gemacht.

```
276b <DebianRulesTemplate11 276b>≡ (276a)
      echo "debian/rules was created " >${log}

      fi
      nano --linenumbers --mouse --softwrap ${GitPath}/debian/rules

      if ! [ -x ${GitPath}/debian/rules ]
      then
          chmod ugo+x ${GitPath}/debian/rules
          echo "${GitPath}/debian/rules is now executable" >> ${log}
      fi
  }

<DebianSalsaCiTemplate 277>
```

35.4.9. salsa-ci.yml

Auf *salsa.debian.org* wird im jeweiligen Projekt unter *Einstellungen - CI/CD - Allgemeine Pipelines - CI/CD configuration file debian/salsa-ci.yml* eingetragen, oder der entsprechende Dateiname, der sich im Verzeichnis *debian/* befindet.

Damit wird der automatische Bauprozess auch für die *Reproducible Builds* ausgelöst.

```

277 <DebianSalsaCiTemplate 277>≡ (276b)
function DebianSalsaCiTemplate {
    # Called by DisplayDebianFiles

    # String for debian/salsa-ci.yml
    str4salsa="include:\n\
- https://salsa.debian.org/salsa-ci-team/pipeline/raw/master/salsa-ci.yml\n\
- https://salsa.debian.org/salsa-ci-team/pipeline/raw/master/pipeline-jobs.yml"

    if ! [ -f ${GitPath}/debian/salsa-ci.yml ]
    then
        touch ${GitPath}/debian/salsa-ci.yml
        echo -e ${str4salsa} >> ${GitPath}/debian/salsa-ci.yml
        echo "debian/salsa-ci.yml was created." >> ${log}
    fi
    nano --linenumbers --mouse --softwrap ${GitPath}/debian/salsa-ci.yml
}

<SelectChangesFile 359>
```

35.4.10. debian/javabuild

Die folgende Funktion ermöglicht die Erstellung einer Datei *debian/javabuild*, welche zum Bauen eines Java-Paketes ohne Build-System dienen kann.

Die Datei *debian/javabuild* enthält in jeder Zeile den Namen einer *.jar*-Datei gefolgt von einer Liste von Quellcode-Dateien oder -Verzeichnissen. Diese Datei wird von dem *jahelper*-Programm *jh_build* eingelesen.

```
278a  <DebianJavabuildTemplate 278a>≡ (326)
function DebianJavabuildTemplate {
    # Called by DisplayDebianFiles
    # For building a java package without build system (like maven)
    if [ -f debian/javabuild ]
    then
        if whiptail --title "Creating debian/javabuild?" \
        --defaultno --yesno "Should debian/javabuild be created?" \
        --yes-button "Yes" --no-button "No" 15 60
        then
            echo "# NameOfJarFile SourceDirToPackage" >>debian/javabuild
            echo "debian/javabuild was created" >> ${log}
            nano --linenumbers --mouse --softwrap debian/javabuild
        fi
    fi
}

<DisplayDebianChangelog 321b>
```

35.4.11. <Paketname>.install

Diese Datei wird benötigt, um anzugeben, wohin Dateien installiert werden sollen. Dabei ist zwingend darauf zu achten, dass als Paketname wirklich der Name des zu bauenden Binary verwendet wird.

```
278b  <DisplayDebianFiles5 278b>≡ (259b)
    if [ ${WebextFlag} -eq 1 ] && [ ! -f ${GitPath}/debian/${PackName}.install ]
    then
        WebextInstall
    fi

    nano --linenumbers --mouse \
    --softwrap ${GitPath}/debian/${PackName}.install
<DisplayDebianFiles6 279a>
```

Beispiel:

```
nc.jar usr/share/java
```

Damit werden die jeweiligen Dateien in das zukünftige Verzeichnis kopiert. Standardmäßig ist es nicht möglich, eine Datei umzubenennen, damit der Dateiname zum Beispiel der Benennung von Java-Bibliotheken in der *Debian Policy für Java* [28] entspricht.

Wie im Handbuch zu *dh_install* ⁷ beschrieben. Dazu sind dann auch weitere Schritte notwendig.

- Das Paket muss eine Bauabhängigkeit zu *dh-exec* haben. Das Paket, das in der Datei *debian/control* angegeben werden muss, ist *dh-exec*
- Die Installationsdatei muss als ausführbar markiert sein.

35.4.12. <Paketname>.dirs

Diese Datei muss nicht zwingend den Namen des Binaries führen. Der Paketname kann auch komplett weggelassen werden.

```
279a <DisplayDebianFiles6 279a>≡ (278b)
      nano --linenumbers --mouse \
        --softwrap ${GitPath}/debian/${PackName}.dirs
<DisplayDebianFiles7 279b>
```

Beispiel:

```
usr/share/java
```

35.4.13. <Paketname>.docs

Die hier aufgeführten Dateien werden im Buildprozess vom entsprechenden *debhelper dh_installdocs* in ein dazu erstelltes Verzeichnis */usr/share/docs/<Paketname>* installiert.

Die Datei *LICENSE* muss nicht aufgenommen werden. Diese wird automatisch nach */usr/share/docs/<Paketname>* installiert.

```
279b <DisplayDebianFiles7 279b>≡ (279a)
      nano --linenumbers --mouse \
        --softwrap ${GitPath}/debian/${PackName}.docs
<DisplayDebianFiles8 280a>
```

⁷https://manpages.debian.org/unstable/debhelper/dh_install.1.de.html

35.4.14. <Paketname>.links

Diese Datei wird vom *dh_link* aufgerufen.

```
280a <DisplayDebianFiles8 280a>≡ (279b)
      if [ ${WebextFlag} -eq 1 ] && [ ! -f ${GitPath}/debian/${PackName}.links ]
      then
          WebextLinksTB
      fi

      nano --linenumbers --mouse \
          --softwrap ${GitPath}/debian/${PackName}.links
<DisplayDebianFiles9 281>
```

35.4.15. Paketname.desktop

```
280b <PaketnameDesktop 280b>≡
      [Desktop Entry]
      X-AppInstall-Package=JVerein
      X-AppInstall-Popcon=1
      X-AppInstall-Section=main

      Version=1.0
      Name=JVerein
      Comment=Administration of an Association
      Comment[de]=Vereinssoftware

      Exec=jameica
      Icon=jameica-icon
      Terminal=false
      Type=Application
      Categories=Office
      Keywords=Association;Verein
      StartupNotify=true
```

35.4.16. <Paketname>.manpages

Nach der Debian-Policy[7] sollte jedes Programm und jede Funktion eine entsprechende Handbuchseite haben. Diese sollte im selben Paket oder zumindest in einer Abhängigkeit enthalten sein.

Gibt es im Quellcode keine solchen Handbuchseiten, muss diese vom Maintainer erstellt und im Verzeichnis *debian/man* abgelegt werden.

Dieser Ort wird dann in die Datei *<Paketname>.manpages* eingetragen. Damit kann sie von *dh_installman* aufgerufen werden.

Beispiel:

```
debian/man/build-gbp.1
```

35.4.17. <Paketname>.examples

Diese Datei wird vom *dh_installexamples* aufgerufen.

```
281 <DisplayDebianFiles9 281>≡ (280a)
    nano --linenumbers --mouse \
        --softwrap ${GitPath}/debian/${PackName}.examples
```

```
<DisplayDebianFiles10 259c>
```

35.4.18. README.Debian

35.4.19. README.source

In dieser Datei können die Ausschlüsse (Kapitel 10.4.1.3, Seite 32) und ihre Begründungen dokumentiert werden⁸. Diese Datei erhält auf jeden Fall einen Eintrag, wenn mit *maven* gebaut wird. (Kapitel 47.4.5, Seite 432)

35.4.20. source/lintian-overrides

Manchmal erlauben die Debian-Richtlinien^[7] Ausnahmen von einer Regel. Dann erzeugt *lintian* eine falsche Meldung. Gleiches gilt in den seltenen Fällen, in denen *lintian* selbst fehlerhaft ist.

In diesen Fällen kann die Datei *debian/source/lintian-overrides* genutzt werden, um eine solche Fehlermeldung zu unterdrücken. Damit wird dann das Programm fehlerfrei beendet.

Beispiel für eine solche Datei:

```
#/usr/share/lintian/overrides/<PackageName>
<PackageName>: <Lintian message>
```

Näheres zu den Lintian-Meldungen findet sich in Kapitel 28 (Seite 103).

35.5. Überprüfung der Dateien in *debian/* mit CmeFix

Der Befehl *cme fix dpkg* prüft die *dpkg*-Dateien, aktualisiert veraltete Parameter und wendet alle Korrekturen an.

Mit dem Parameter *-verbose* erhält man mehr Informationen darüber, was passiert. Der Parameter *-backup* erzeugt vor der Speicherung der Änderungen Backup-Dateien. Diese werden durch die Endung *.old* gekennzeichnet. Zu beachten ist, dass in diesem Fall die Langform der Optionen wirklich nur mit einem Bindestrich eingeleitet werden⁹.

282 `<CmeFix 282>≡` (330)

```
function CmeFix {
    # Called by DisplayDebianFiles

    if whiptail --title "Check and fix with cme?" \
        --yesno "Should debian files be checked and fixed using 'cme fix'?" \
        --yes-button "Yes" --no-button "No" 15 60
    then
        set +e
        if whiptail --title "Backup?" \
            --yesno "Should the recent files be backed up (recommended)?" \
            --yes-button "Yes" --no-button "No" 15 60
        then
            cme fix --verbose --backup dpkg
        fi
    fi
}
```

⁸Debian-Policy, Kapitel 4.14 [7]

⁹<https://manpages.debian.org/unstable/cme/cme.1p.en.html>


```

else
    cme fix --verbose dpkg
fi
set -e
⟨cme fix 283⟩

```

Die Ausführung des Programmskriptes wird hier angehalten, damit die Ausgaben des Befehls *cme fix dpkg* analysiert werden können. In einem anderen Terminal können – sofern die Backupoption gewählt wurde – die ursprünglichen Dateien mit den neu erstellten Dateien verglichen, Backupdateien gelöscht und gegebenenfalls Korrekturen vorgenommen werden.

```

283 ⟨cme fix 283⟩≡ (282)
    echo "Please check the result of cme fix!"
    echo "You can check and fix it in another terminal."
    echo "Please press RETURN to go on."
    read a
    fi
}

```

⟨DisplayDebianFiles 257⟩

36. Änderungen am Upstream-Code vornehmen

Weiter geht es mit möglichen Änderungen am Quellcode. Werden keine Änderungen am Quellcode vorgenommen, folgt direkt die Behandlung der Datei *debian/changelog* (Kapitel 37.1, Seite 321).

```
285a <BuildNewRevision5-1 285a>≡ (256)
      # Patches treatment
      PatchesTreatment
```

<BuildNewRevision6 321a>

Wenn eine neue Revision erstellt wird, wird der Nutzer gefragt, ob Veränderungen am Quellcode vorgenommen werden sollen. Dazu prüft das Programm jedoch zuvor nochmals (s. Kapitel 34.2, Seite 197), ob ein Verzeichnis *debian/patches* bereits vorhanden ist, und teilt dem Nutzer das Ergebnis dieser Prüfung mit.

```
285b <PatchesTreatment 285b>≡ (350)
      function PatchesTreatment {
          # Called by BuildNewRevision

          # Patches treatment
          cd ${GitPath}
          if [ -d debian/patches ]
          then
              whiptail --title "Info" \
                  --msgbox "There is a directory debian/patches" 15 60
          fi
      }
      <PatchesTreatment1 286a>
```

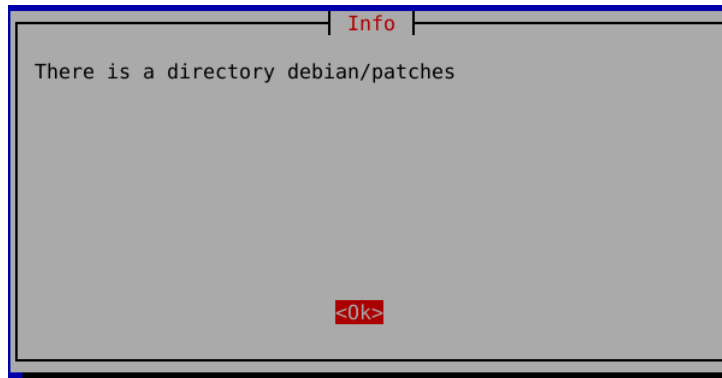


Abbildung 36.1.: Es gibt ein Verzeichnis debian/patches.

```
286a  <PatchesTreatment1 286a>≡ (285b)
      else
        whiptail --title "Info" \
          --msgbox "There is no directory debian/patches" 15 60
      fi
```

<PatchesTreatment2 286b>

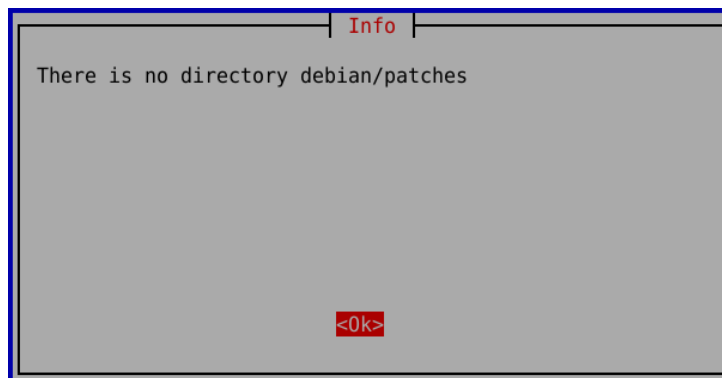


Abbildung 36.2.: Es gibt kein Verzeichnis debian/patches.

Nun erfolgt die Abfrage, ob ein Patch erstellt, bearbeitet oder gelöscht werden soll. Außerdem kann zwischen den beiden Methoden *quilt* (Kapitel 36.3, Seite 302) und *gbp pq* (Kapitel 36.1, Seite 288) gewählt werden.

```
286b  <PatchesTreatment2 286b>≡ (286a)
      PMTask=$(whiptail --title "Tasks:" \
        --radiolist "Do you want to create, edit or delete patches?" 15 60 4 \
        "0" "By using quilt" off \
        "1" "By using gbp pq" off \
        "2" "No" on --cancel-button "Exit" 3>&2 2>&1 1>&3)
```

<PatchesTreatment3 287a>

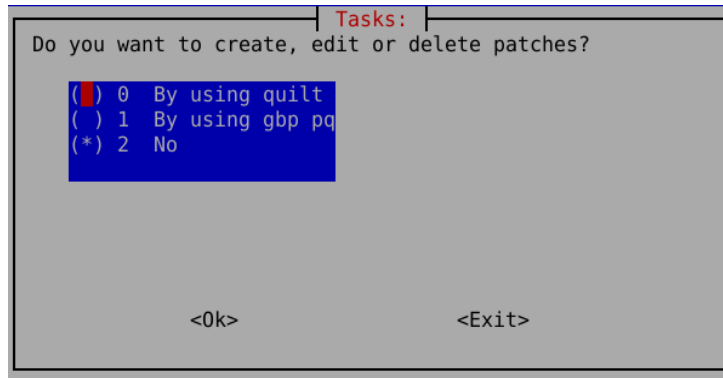


Abbildung 36.3.: Patches für Debian erstellen

Natürlich kann der Bauprozess auch ohne *patches* fortgesetzt werden (Kapitel 37.1, Seite 321).

Wird die Option „No“ ausgewählt, geht es mit dem Editieren der Datei *debian/changelog* weiter (Kapitel 37.1, Seite 321).

Mit *Exit* wird das Programmskript verlassen.

```
287a <PatchesTreatment3 287a>≡ (286b)
      if [ -z "${PMTask}" ]
      then
          exit
      fi
```

<PatchesTreatment4 287b>

Hat man sich zum Patchen und zu einer Methode entschieden, ruft das Programmskript nun die jeweils weiterführenden Funktionen auf.

```
287b <PatchesTreatment4 287b>≡ (287a)
      if [ ${PMTask} -eq 0 ]
      then
          PatchRunNr=0
          PatchTasks
```

<PatchesTreatment5 287c>

Entweder man arbeitet mit *gbp pq* oder mit *quilt* (Kapitel 36.3, Seite 302).

Hat man sich für die Arbeit mit *quilt* entschieden, wird die Variable *PatchRunNr* auf 0 gesetzt und die Funktion *PatchTasks* (Kapitel 36.3, Seite 302) aufgerufen.

```
287c <PatchesTreatment5 287c>≡ (287b)
      elif [ $PMTask -eq 1 ]
      then
          echo -e "\n\n\033[31mEVERYTHING MUST BE COMMITTED! \033[0m\n"
          CheckGitStatus
          PQMigration
      fi
  }
```

<LastQuestionsBeforeBuild 339a>

Andernfalls wird die Funktion *PQMigration* (Kapitel 36.1.1, Seite 288) des Programmskriptes aufgerufen. Diese Methode sollte bei der Nutzung von *git-buildpackage* bevorzugt werden.

Vor dem Aufruf der Funktion *PQMigration* (und vorsichtshalber später nochmals) wird zur Prüfung die Funktion *CheckGitStatus* (Kapitel 33.4.1, Seite 183) aufgerufen.

36.1. Arbeiten mit *gbp pq*

Um Patches zu verwalten, kann *gbp pq* verwandt werden. Dies ist hauptsächlich für Quellpakete im Format 3.0 (*quilt*) gedacht (s. Kapitel 35.4.2, Seite 260). Die Änderung des Upstream-Quellcodes erfolgt durch Dateien im Verzeichnis *debian/patches/*.

Die Funktion *PQMigration* kann unter Verwendung dieser Patches einen Patch-Queue-Zweig erstellen, falls er noch nicht existiert. Ferner kann diese Funktion einen existierenden Patch-Queue-Zweig aktualisieren.

36.1.1. Erstellen eines Patch-Queue-Zweiges

Es wird nochmals geprüft, ob es eine Datei *debian/patches/series* gibt oder nicht.

Auch wird geprüft, ob es einen passenden *Patch-Queue-Branch* gibt.

Es sind vier Fälle zu unterscheiden:

1. Es gibt weder eine Datei *debian/patches/series* noch einen passenden *Patch-Queue Branch*. Dann wird mit *git checkout -b* ein neuer *Patch-Queue Branch* erzeugt und in diesen gewechselt.
2. Die Datei *debian/patches/series* existiert nicht, aber schon ein passender *Patch-Queue Branch*. Dann wird auf dessen Vorhandensein hingewiesen und in diesen gewechselt.
3. Die Datei *debian/patches/series* existiert, aber kein passender *Patch-Queue Branch*. Dann wird mit *gbp pq import* ein passender *Patch-Queue Branch* erzeugt. Allerdings ist dies mit erheblichen Risiken verbunden, wenn nicht alle Patches der Queue anwendbar sind (Kapitel 34.2, Seite 197). Daher wird **dringend** empfohlen, den *Patch-Queue Branch* gegebenenfalls vor dem Herunterladen einer neuen Version anzulegen. Ist dies nicht geschehen, sollte die Option gewählt werden, mit *quilt* zu arbeiten.
4. Sowohl die Datei *debian/patches/series* als auch der passende *Patch-Queue Branch* existieren. Dies wird häufig der Fall sein. Dann wird zunächst *gbp pq rebase* angewandt.

Ein *Patch-Queue-Zweig* wird lediglich in den Fällen 1 und 3 erstellt. Andernfalls wird der *Patch-Queue Branch* auf die neue Version rebased.

```
288 <PQMigration 288>≡ (293c)
function PQMigration {
    # Called by PatchesTreatment and itself
    # Transfers patches into patch-queue branch
    npqf=0
    cd ${GitPath}

    if [ ! -f debian/patches/series ] # debian/patches/series does not exists
```

```

then
  # Case 2
  # Anything is easy
  npqf=1
  set +e
  if echo $(git branch) | grep --quiet 'patch-queue/'${RecentBranch}
  then
    whiptail --title "PQ-branch exists" \
      --msgbox "Branch 'patch-queue/${RecentBranch}' exists." 15 60
    git checkout patch-queue/${RecentBranch}
  }
  <PQMigration0 289>

```

Wenn die Datei *debian/patches/series* nicht existiert, aber schon ein passender *Patch-Queue Branch*, erscheint als Hinweis folgende Nachricht.

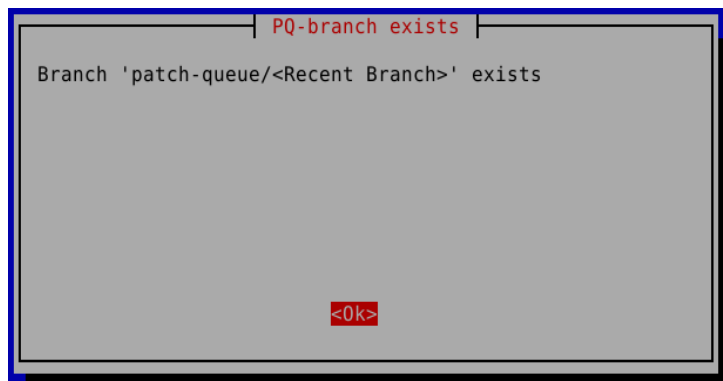


Abbildung 36.4.: Es existiert ein PQ-Zweig.

Existiert die Datei *debian/patches/series* nicht und auch kein passender Patch-Queue-Zweig, wird er hier neu erstellt und in diesen gewechselt.

```

289 <PQMigration0 289>≡ (288)
    # Case 1
    else # creating a new patch-queue branch
      git checkout -b patch-queue/${RecentBranch}
    fi
    set -e
  }
  <PQMigration1 290>

```

Es kann dann sofort mit der Bearbeitung des Quellcodes begonnen werden (Kapitel 36.1.7, Seite 296).

Sowohl *gbp pq rebase* als auch *gbp pq import* setzen voraus, dass es im aktuellen Git-Zweig keine unversionierten Dateien gibt und sich alle bisherigen Patches anwenden lassen.

```
290 <PQMmigration1 290>≡ (289)
    else # debian/patches/series exists
        Notice="All patches listed in 'debian/patches/series' \n\
        have to be applicable"'\n\
        Otherwise you have to solve 'merge conflicts''\
        if whiptail --title "Attention please"'\ \
        --yesno "${Notice} Do you want to check the situation?" \
        --yes-button "Yes" --no-button "No" 15 60
        then
            ShowPatches "check"
<PQMmigration1-1 291>
```

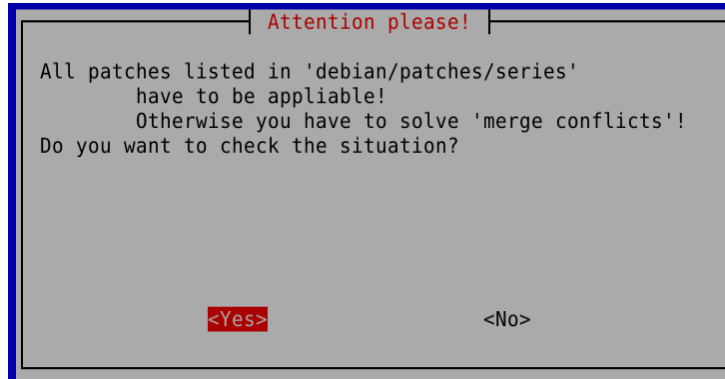



Abbildung 36.5.: Alle Patches müssen anwendbar sein.

Andernfalls sind Merge-Konflikte zu lösen.

Hierauf wird der Nutzer hingewiesen.

Wird die Frage, ob die Situation geprüft werden soll, bejaht, wird die Funktion *ShowPatches* (mit dem Parameter *check*) aufgerufen (s. Kapitel 36.2, Seite 300).

Die einzelnen Patches können angepasst werden.

Danach wird gefragt, ob die Datei *debian/patches/series* editiert werden soll.

Werden Patches oder die Datei *debian/patches/series* verändert, sind diese Änderungen zu committen. Daher wird auch nochmals der *Git*-Status geprüft.

Es kann die Patch-Queue geprüft (Kapitel 36.1.2, Seite 292) und eventuell mittels *dquilt* bearbeitet werden. Wer die Lösung von Merge-Konflikten nicht scheut, kann sich diesen „Umweg“ sparen.

Was dann durchgeführt werden kann, steht in Kapitel 36.1.5 (Seite 294).

```

291 <PQMigration1-1 291>≡ (290)
    if whiptail --title "Attention please"!' \
      --yesno "Do you want to edit 'debian/patches/series'?" \
      --yes-button "Yes" --no-button "No" 15 60
    then
      nano --linenumbers --mouse --softwrap debian/patches/series
    fi
  fi

  echo -e ${Notice}
  CheckGitStatus
fi
<PQMigration2 292>

```

Abbildung 36.6.: Soll *debian/patches/series* editiert werden?.

Als nächstes wird vorsorglich nochmals der `Git`-Status geprüft. Dann geht es wie in Kapitel 36.1.4 (Seite 292) beschrieben weiter.

36.1.2. Manuelle Bearbeitung

In der Datei *debian/patches/series* werden alle Patches in der Reihenfolge aufgelistet, in der diese bisher angewandt wurden.

Nun müssen sie darauf geprüft werden, ob die Probleme durch Upstream bereits verbessert wurden und diese Patches damit entfallen.

Dann sind diese Patches aus dem Verzeichnis *debian/patches* und ihre Einträge aus der Datei *debian/patches/series* zu entfernen. Diese Anpassungen erfolgen im Hauptzweig.

36.1.3. Hinweise zur Fehlerbehebung

Zur Fehlerbehebung nach einem Fehlschlag können nur allgemeine Hinweise gegeben werden.

Liegen bereits Änderungen an Quellcode-Dateien vor, sind diese mit `git restore <Pfad/Dateiname>` rückgängig zu machen. Ein eventuell vorhandenes Verzeichnis *.pc/* ist zu löschen.

Änderungen an der Datei *debian/changelog* sind zu committen.

36.1.4. Aktualisieren des Patch-Queue-Zweiges

Das Programmskript prüft, ob bereits ein entsprechender Patch-Queue-Zweig existiert.

Existiert ein passender Patch-Queue-Zweig ruft das Programmskript zunächst die Funktion `RebasePQBranch` auf.

Andernfalls wird Patch-Queue-Zweig durch `gbp pq import` aus dem aktuellen `Git`-Zweig erstellt (Kapitel 36.1.6, Seite 294).

292 `<PQMigration2 292>≡` (291)

```
if [ ${npqf} -eq 0 ]
then
  set +e
  if echo $(git branch) | grep --quiet 'patch-queue/'${RecentBranch}
```

```

    # patch-queue branch exists
    then
        # Case 4
        RebaseCounter=0
        RebasePQBranch
    <PQMigration3 295>

```

Die Funktion *RebasePQBranch* führt ein *gbp pq rebase* aus. Die Commits des *debian/-* Zweiges werden auf den Patch-Queue-Zweig angewandt.

Mit *gbp pq rebase* wird in den Patch-Queue-Zweig gewechselt, der mit dem aktuellen Zweig verbunden ist. Alle Neuerungen des aktuellen Zweiges werden durch ein *rebase* in den Patch-Queue-Zweig übernommen.[46]

```

293a <RebasePQBranch 293a>≡ (314)
function RebasePQBranch {
    # Called by PQMigration and itself
    if [ ${RebaseCounter} == 0 ]
    then
        gbp pq rebase --verbose
    else
        git rebase --continue
    fi

```

```

<RebasePQBranch1 293b>

```

Misslingt die Ausführung von *gbp pq rebase*, kann und muss händisch eingegriffen werden, um die Situation zu bereinigen

Hierzu werden von *Git* dem Nutzer Hinweise gegeben, die sorgfältig studiert und meist befolgt werden sollten.

```

293b <RebasePQBranch1 293b>≡ (293a)
    if [ $? -ne 0 ]
    then
        git rebase --show-current-patch | cat
    <RebasePQBranch2 293c>

```

An dieser Stelle wird der Patch angezeigt, der nicht (komplett) angewandt werden kann. Mit dieser Hilfe ist es dann möglich, nach der Unterbrechung manuell die Merge-Konflikte aufzulösen.

```

293c <RebasePQBranch2 293c>≡ (293b)
    Notice="gbp pq rebase failed"'\n\
    All changes must be committed"'\n\
    All patches have to be applicable"'\n"
    FailureNotice ${Notice}
    RebaseCounter=$(expr ${RebaseCounter} + 1)
    RebasePQBranch
fi
}

```

```

<PQMigration 288>

```

Wie die Konflikte aufgelöst werden können, wird in Kapitel 36.1.5 (Seite 294) beschrieben. Danach kann ein erneuter Versuch erfolgen.

36.1.5. Hinweise zur Bereinigung der Patch-Queue

In sehr vielen Fällen schlägt der Befehl *pq rebase* fehl, da es auch immer wieder Änderungen von Upstream an bereits gepatchten Dateien gibt. Patches, die bereits vom Upstream-Projekt übernommen wurden, können entfernt werden.

Das Programmskript gibt in solchen Fällen den Hinweis, auf ein weiteres Terminal zu wechseln. Dort kann mit *git status* überprüft werden, dass ein unvollständiger Rebase vorliegt.

Mit dem nachstehenden Befehl wird in dem weiteren Terminal der fehlgeschlagene Patch angezeigt.

```
git rebase --show-current-patch
```

In noch einem weiteren Terminal kann das (noch) Vorhandensein der zu patchenden Datei mit folgendem Befehl geprüft werden.

```
tar --list --file ../<UpstreamPackageName>.orig.tar.xz | \  
grep <Filename>
```

Ist die Datei nicht (mehr) vorhanden, kann der Patch mit

```
git rebase --skip
```

entfernt werden. Sodann ist im (ursprünglichen) Terminal, in dem das Programmskript läuft, mit *RETURN* fortzufahren. Damit wird zunächst ein *git rebase -continue -verbose* ausgeführt.

Wenn die zu patchende Datei existiert, muss sie in einem Editor bearbeitet werden. Damit werden die Patches so geändert, dass diese auf die neue Version angewandt werden können.

Die Datei ist dann zu speichern und ein *git add* auszuführen. Im ursprüngliche Terminal ist mit *RETURN* fortzufahren.

Der Vorgang ist gegebenenfalls sooft zu wiederholen, bis das Programmskript mit seiner Ausführung fortfährt.

Eine Möglichkeit ist es auch, durch den Befehl

```
gbp pq import --time-maschine=<n>
```

solange Commit für Commit durchzugehen, bis die Patch-Queue angewandt werden kann.

n gibt dabei die maximale Anzahl der Rückschritte an.

36.1.6. Import vorhandener Patches

Der Inhalt von *debian/patches* wird mit *gbp pq* in den Patch-Queue-Branch importiert und in diesen gewechselt¹.

Beim Import wird eine Ausgabe wie folgt erzeugt:

¹s. Abschnitt *gbp.patches.html* und *man.gbp.pq.html*[3]

```
gbp:info: Versuchen, Patches \
unter 'aaa1011bfd5aa74fea43620aae94709de05f80be' \
anzuwenden.
```

```
gbp:info: 18 Patches, die in 'debian/patches/series' \
aufgelistet sind, importiert unter 'patch-queue/debian/sid'.
```

Es wurde mit *gbp pq* jede Patchdatei importiert und in den neu erstellten Patch-Queue-Zweig *patch-queue/debian/sid* gewechselt.

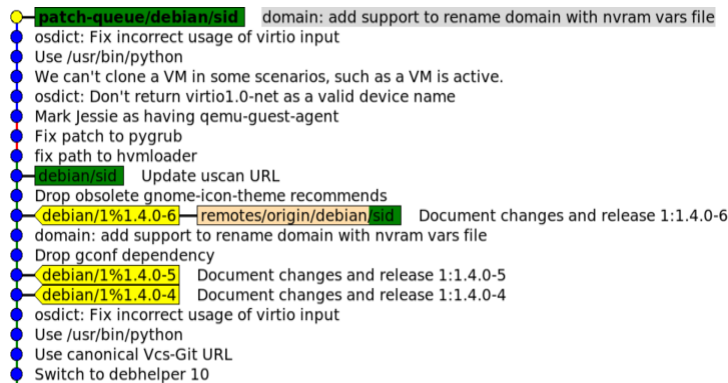


Abbildung 36.7.: Der Patch-Queue-Zweig mit Patches von *debian/patches* wurde angewendet.[3]

Existiert die Datei *debian/patches/series*, aber noch kein entsprechender Patch-Queue-Zweig, wird *gbp pq import* ausgeführt. Dies geschieht in der Funktion *PQImport* (Kapitel 34.2, Seite 197).

gbp pq import erzeugt einen neuen Patch-Queue-Zweig aus den *quilt patches* im Verzeichnis *debian/patches*, die in der Datei *debian/patches/series* aufgeführt sind. Diese Patches müssen ohne Unschärfe anwendbar sein.

```
295 <PQMigration3 295>≡ (292)
    else # patch-queue branch does not exist
      # Case 3
      if whiptail --title "Attention"'' \
        --yesno "Do you really want to use 'gbp pq import'?" \
        --yes-button "Yes" --no-button "Working with Quilt" 15 60
      then
        PQImport 0
      else
        PatchTasks # Working with Quilt
      fi
    fi
  fi
  set -e
fi
<PQMigration4 296>
```

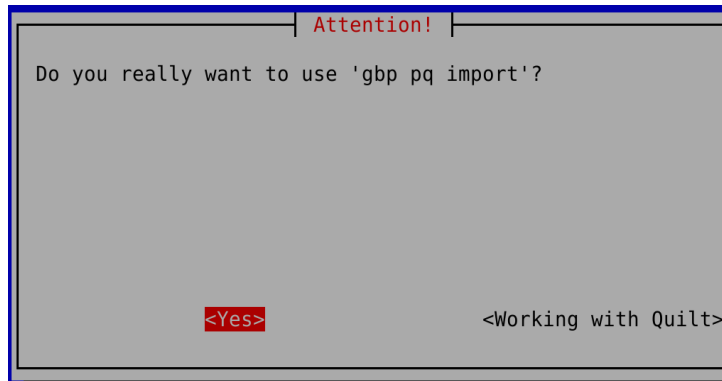


Abbildung 36.8.: Soll *gbp pq* genutzt werden?

Auch hier gelten die obigen Hinweise zur Fehlerbehebung (Kapitel 36.1.3, Seite 292).

36.1.7. Bearbeiten des Quellcodes

296 $\langle PQMigration4\ 296 \rangle \equiv$

(295)

```
# Starting the work in the patch-queue branch
echo
echo "-- Starting the work in the patch-queue branch --"
echo
echo "Break for patching in another terminal"
echo "Do not forget to commit the changes!"
echo
echo "After finishing press RETURN to go on!"
read a
```

$\langle PQMigration5\ 298a \rangle$

```
-- Starting the work in the patch-queue branch --

Break for patching in another terminal
Do not forget to commit the changes!

After finishing press RETURN to go on!
```

Abbildung 36.9.: Unterbrechung zum Patchen

Im separaten Terminal sollte mit `git branch -v` festgestellt werden, dass der richtige Git-Zweig (`patch-queue/-Zweig`) aktiv ist und dass dessen Stand dem des bisherigen Git-Zweiges entspricht. Sodann können Dateien im Patch-Queue-Zweig bearbeitet werden.

Code kann hinzugefügt, verändert oder entfernt werden. So können die Patches erstellt oder „in Form gebracht“ werden.

Dabei sind die Änderungen möglichst kleinteilig zu committen. Was später ein einzelner Patch in `debian/patches/` werden wird, wird einfach durch einen Commit hinzugefügt.

Die erste Zeile der Commit-Nachricht wird später Teil des Patch-Namens. Die folgenden Zeilen enthalten die Details zu den Funktionen des Patches. Daher ist es sinnvoll mehrzeilige Commit-Nachrichten zu schreiben.

Die mehrzeilige Commit-Nachricht sollte dann im Patch der *Patch tagging guideline* entsprechen² Der Header der Patch-Datei sieht dann wie folgt aus:

Der Autor und das Datum werden automatisch gesetzt. Der unter *Gbp-PQ* anzugebene Name darf keine Leerzeichen enthalten. Diese sind gegebenenfalls durch *Unterstriche* (`_`) zu ersetzen. Dies wird der Name der Patch-Datei im Verzeichnis `debian/patches`.

```
From: Autor <E-Mail-Adresse>
Date: <Erstellungsdatum und -zeit>
Subject: <Commit-Nachricht>
```

```
Gbp-Pq: Name <name of the patch>
Forwarded: Yes/No <if necessary>
```

```
* posix/regcomp.c (re_compile_fastmap_iter): Rewrite COMPLEX_BRACKET
  handling.
```

```
Origin: upstream, http://sourceware.org/git/?p=glibc.git;\
a=commitdiff;h=bdb56bac
```

```
Bug: http://sourceware.org/bugzilla/show_bug.cgi?id=9697
```

```
Bug-Debian: http://bugs.debian.org/510219
```

Diese kann mit `git commit --amend -m "Neue Message"` korrigiert werden.

²[https://dep-team.pages.debian.net/deps/dep3/\[26\]](https://dep-team.pages.debian.net/deps/dep3/[26])

36.1.8. Export der Patches

Sobald wir mit den Commits zufrieden sind, lassen Sie uns die Patches in *debian/patches/* unter Verwendung von *gbp pq* neu generieren. Dadurch wechseln Sie zurück zum Zweig *debian/sid* und regenerieren die Patches mit einer Methode ähnlich dem *git-format-patch*:

Das Ergebnis könnte nun wie folgt hinzugefügt werden:

```
gbp pq export
git add debian/patches
git commit
```

gbp pq export bedeutet:

Exportiert die Patches auf dem Patch-Queue-Branch, der mit dem aktuellen Zweig verbunden ist, in eine Quilt-Reihe von Patches nach *debian/patches/* und aktualisierte die Datei *series* im aktuellen Zweig (z.B. *debian/sid*)³[3].

Damit das Ergebnis nicht jedes Mal von Hand übertragen werden muss, kann auch *-commit* an den *gbp-Export*-befehl übergeben werden.

```
298a <PQMigration5 298a>≡ (296)
# Export
if whiptail --title "Use gbp pq export?" \
--yesno "Do you like to use 'gbp pq export --commit'?" \
--yes-button "Yes" --no-button "No" 15 60
<PQMigration7 298b>
```



Abbildung 36.10.: Soll *gbp pq export* angewandt werden?

```
298b <PQMigration7 298b>≡ (298a)
then
CheckGitStatus
gbp pq export --commit --verbose >> ${log} 2>&1
echo "Check branch."
git branch --verbose
echo "Press RETURN to continue!"
read a
<PQMigration8 299>
```

³<https://honk.sigxcpu.org/projects/git-buildpackage/manual-html/gbp.patches.html>

Nach dem Export der Patches wird das Ergebnis im Terminal angezeigt.

Nach dem *RETURN* kann die Datei *debian/changelog* aktualisiert werden (z. B. durch Ausführen von *gbp dch -S -a*, was dasselbe ist wie *gbp dch --snapshot --auto*. (Kapitel 37.1, Seite 321)

Das Paket kann dann– wie gewohnt – erstellt werden.

Wird die Frage verneint, kann manuell eingegriffen werden.

```

299 <PQMigration8 299>≡ (298b)
    else
        git log
        git checkout ${HistoricBranch}
        ReplaceConfigLines 'RecentBranch' ${HistoricBranch}
        git branch
        echo "-- You have cancelled the export from patch-queue branch --"
        echo
        echo "You are still in the patch-queue branch!"
        echo "Break for fixing in another terminal"
        echo "After finishing press RETURN to go on!"
        read a
    fi
}
<PatchTasks 303b>

```

36.2. Auswahl der Patches

Die folgende Funktion *ShowPatches* kann sowohl beim Arbeiten mit *gbp pq* als auch beim Arbeiten mit *quilt* Anwendung finden.

```

300 <ShowPatches 300>≡ (306)
    function ShowPatches {
        # Called by EditPatch DeletePatch PQMigration and itself
        actionstr=${1}

        patchfilesa=$(ls debian/patches)
        i=0; slct=''
        for element in ${patchfilesa[*]}
        do
            if [ ${element} != 'series' ]
            then
                slct=${slct} '$i' '${element}' off '
                newPFA[$i]=${element}
                i=$(expr $i + 1)
            fi
        done

        set +e
        PatchFileNo=$(whiptail --title "Select patch" \
            --radiolist "Select one of these patches:" \
            --cancel-button "Cancel" 15 60 8 \
            $slct 3>&2 2>&1 1>&3)

    <ShowPatches1 301>

```

```

301  <ShowPatches1 301>≡ (300)
      if [ $? -eq 1 ]
      then
          return
      fi
      set -e

      if [ "${actionstr}" = "check" ]
      then
          # Code for PQMigration
          if [ -z "${PatchFileNo}" ]
          then
              return
          fi
          PatchFileName=${newPFA[${PatchFileNo}]}
          nano --linenumbers --mouse --softwrap debian/patches/${PatchFileName}
          ShowPatches "check"
      else
          if [ -z "${PatchFileNo}" ]
          then
              PatchFileName=""
          else
              PatchFileName=${newPFA[${PatchFileNo}]}
              if ! whiptail --title "${PatchFileNo}" \
                  --yesno "Do you want to ${actionstr} ${PatchFileName}?" \
                  --yes-button "Yes" --no-button "No" 15 60
              then
                  ShowPatches
              fi
          fi
      fi
  }

  <EditPatch 318>

```

36.3. Nutzung von Quilt

Die Änderungen am Upstream können auch mittels Quilt erfolgen. Wie zuvor beschrieben, ist *dquilt* zunächst einzurichten (Kapitel 19.6, Seite 72).

Die Einrichtung in der *.bashrc* lässt sich bei der Ausführung dieses Programms nicht nutzen (Kapitel 19.6, Seite 72).

Das Programm prüft daher zunächst, ob *quilt* zur Verfügung steht. Da das Skript den Alias der *.bashrc* unbeachtet lässt, wird *dquilt* in einer Variablen definiert.

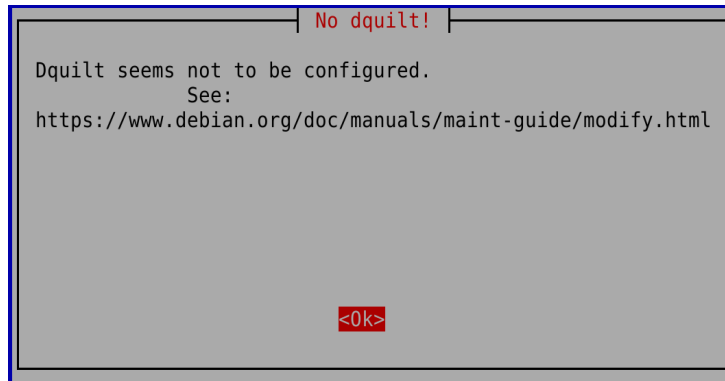
```
302a  <CreateDquilt 302a>≡ (313b)
      function CreateDquilt {
          # Called by PatchTasks

          # Check whether quilt is available
          if [ ${PatchRunNr} -eq 0 ]
          then
              if [ ! -f ~/.quiltrc-dpkg ]
              <CreateDquilt1 302b>
```

Die Zeile prüft, ob es im Home-Verzeichnis des Nutzers eine Datei *.quiltrc-dpkg* existiert. Wenn diese Datei nicht existiert, erfolgt die Meldung, dass *dquilt* nicht konfiguriert ist, weil dann der entsprechende Alias für *quilt -quiltrc=\$HOME/.quiltrc-dpkg* nicht erstellbar ist.

```
302b  <CreateDquilt1 302b>≡ (302a)
      then
          whiptail --title "No dquilt!" \
          --msgbox "Dquilt seems not to be configured.\n \
          See: https://www.debian.org/doc/manuals/maint-guide/modify.html" \
          15 62
          exit
      fi

      <CreateDquilt2 303a>
```

Abbildung 36.11.: *dquilt* ist nicht konfiguriert.

Wenn die Datei *.quiltrc-dpkg* nicht gefunden wird, gibt das Skript eine entsprechende Meldung aus und wird beendet.

Nur wenn die Datei */.quiltrc-dpkg* existiert, kann die folgende Definition erfolgen.

```
303a <CreateDquilt2 303a>≡ (302b)
      # Definition of dquilt
      dquilt="quilt --quiltrc=${HOME}/.quiltrc-dpkg"
      fi
    }
```

<MakePatches 306>

Es wird zur Auswahl gestellt, ob Patches erzeugt, editiert oder entfernt werden sollen. Diese Aufgaben können in beliebiger Reihenfolge und auch mehrmals ausgeführt werden.

```
303b <PatchTasks 303b>≡ (299)
      function PatchTasks {
        # Called by PatchesTreatment and itself

        cd ${GitPath}
        if [ ! -d debian/patches ]
        then
          # Create patch
          if whiptail --title "Patches" --yesno "Is a patch necessary?" \
            --yes-button "Yes" --no-button "No" 15 60
          then
            CreateDquilt
            MakePatches
          else
            return
          fi
        fi
      }
```

<PatchTasks0 304>

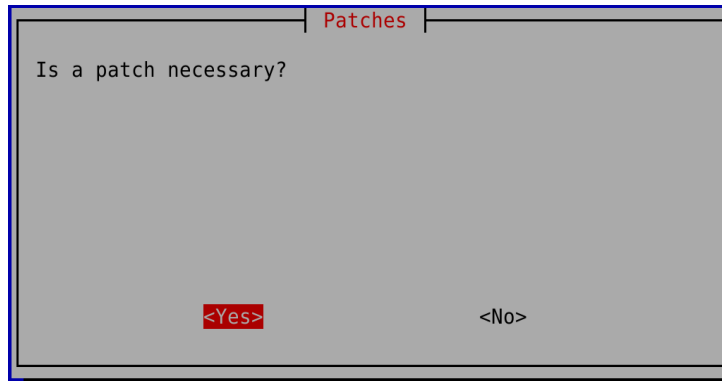


Abbildung 36.12.: Wird ein Patch benötigt?

```
304  <PatchTasks0 304>≡ (303b)
      else
        CreateDquilt
        PTask=$(whiptail --title "Tasks:" \
          --radiolist "What do you like to do? " 15 60 8 \
            "0" "Display patch files to check or edit them" off \
            "1" "Create additional patch" off \
            "2" "Add another patch to existing patch file" off \
            "3" "Show patch files for deleting" off \
            "4" "Edit debian/patches/series" off \
            "5" "Exit to go on" on --cancel-button "Cancel" 3>&2 2>&1 1>&3)
        <PatchTasks1 305a>
```

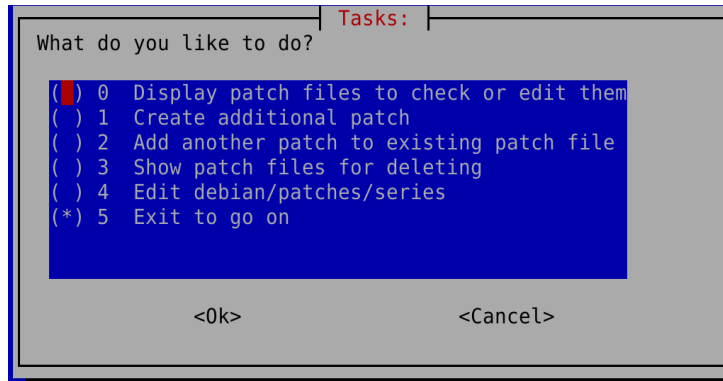


Abbildung 36.13.: Aufgaben für Quilt

Wird der Button *Cancel* gedrückt, so wird ebenfalls die Aufgabe *Exit to go on* ausgeführt. Hierbei wird der Rückgabewert *1*.

305a $\langle PatchTasks1\ 305a \rangle \equiv$ (304)

```
if [ $? -eq 1 ]
then
    PTask=6
fi
```

$\langle PatchTasks2\ 305b \rangle$

Im folgenden Abschnitt wird den einzelnen Aufgaben Befehle zugeordnet. Dabei enthält die Variable *PTask* eine Ziffer von 0 bis 4.

Wenn beispielsweise dieser Variablen der Wert *0 = Display patch files to check or edit them* zugewiesen wurde, dann wird die Funktion *ChangePatches* ausgeführt.

In der folgenden *case-Anweisung* ist *0*) ein Wert aus der Liste von 0 bis 4, gegen den der Inhalt der Variable *PTask* verglichen wird.

305b $\langle PatchTasks2\ 305b \rangle \equiv$ (305a)

```
# Patches treatment
case "$PTask" in
    0) ChangePatches;; # Edit patches
    )
```

$\langle PatchTasks3\ 305c \rangle$

Anhand des Paketes *JaxWS* wird beschrieben, wie vorhandene Patches bearbeitet werden können.

36.3.1. Patch erstellen

305c $\langle PatchTasks3\ 305c \rangle \equiv$ (305b)

```
1) MakePatches;; # If (more) patches are necessary
    )
```

$\langle PatchTasks4\ 313c \rangle$

Nun beginnt das Erstellen eines Patches.

```
306 <MakePatches 306>≡ (303a)
function MakePatches {
    # Called by PatchTasks and itself

    cd ${GitPath}
    cnpr=0
    CreateNewPatch
    if [ $cnpr -ne 0 ]
    then
        return
    fi

    PatchRunNr=1

    if whiptail --title "Another patch?" \
    --yesno "Do you want to apply another patch?" --yes-button "Yes" \
    --no-button "No" 15 60
    then
        MakePatches
    fi
}

<ShowPatches 300>
```

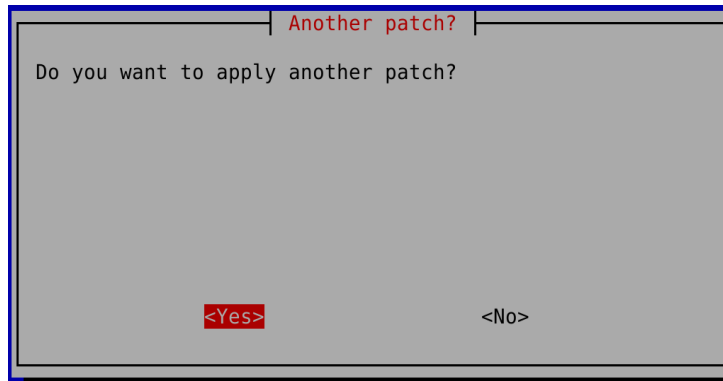



Abbildung 36.14.: Ein weiterer Patch?

36.4. Neuen Patch erstellen

```

307 <CreateNewPatch 307>≡ (310)
    function CreateNewPatch {
        # Called by MakePatches

        PatchFileName=$(whiptail --title "Patch name" \
            --inputbox "Name of the patchfile:" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
        if [ $? -ne 0 ]
        then
            cnpr=1
            return 1
        fi

        <CreateNewPatch0 308a>
    }

```

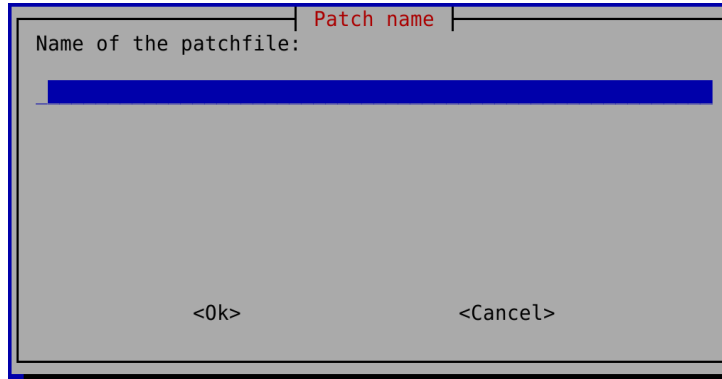


Abbildung 36.15.: Name des Patches

308a `<CreateNewPatch0 308a>≡` (307)

```
# Because a user might use blanks in a filename
PatchFileName=$(echo ${PatchFileName} | sed --expression='s/ /-/g')
if [ -z "${PatchFileName}" ]
then
    cnpr=1
    return 1
fi
```

`<CreateNewPatch1 308b>`

Es wird geprüft, ob ein gleichnamiger Patch bereits existiert.

308b `<CreateNewPatch1 308b>≡` (308a)

```
if [ -f debian/patches/${PatchFileName} ]
then
    if ! whiptail --title "Patch exists" \
        --yesno "${PatchFileName} exists already.\nContinue?" \
        --yes-button "Yes" --no-button "No" 15 60
    then
        return 1
    fi
fi
```

`<CreateNewPatch2 309>`



Abbildung 36.16.: Patch existiert bereits!

Der Befehl `dquilt new` mit dem Parameter des Patch-Namens erzeugt die erste "Patch-Datei".

Hierbei werden folgende Dateien erzeugt:

```

debian/patches/series
    enthält den Namen des Patches DescriptionNoRegistration.patch
.pc/applied-patches
    enthält den Namen des Patches DescriptionNoRegistration.patch
.pc/DescriptionNoRegistration.patch/
.pc/.quilt_patches
    enthält debian/patches
.pc/.quilt_series
    enthält series
.pc/.version
    enthält 2

```

Wenn schon Patches existieren, ist darauf zu achten, dass diese in der Datei `debian/patches/series` in der korrekten Reihenfolge genannt werden.

```

309 <CreateNewPatch2 309>≡ (308b)
    # Create a new patch file
    $dquilt new ${PatchFileName}

    # Patch
    FileToPatch
<CreateNewPatch4 313b>

```

36.5. Datei zum Patchen auswählen

```
310 <FileToPatch 310>≡ (313a)
function FileToPatch {
    # Called by CreateNewPatch PatchTasks and itself

    FileSelector ${GitPath} # Select the file to be patched
    File2Patch=${selected}

    echo "Patch ${PatchFileName} because of ${File2Patch}" >> ${log}

    # quilt add must get only the filename without
    # the path of the file to be patched
    File2pName=$(basename ${File2Patch})
    $dquilt add -P ${PatchFileName} ${File2pName}

    nano --linenumbers --mouse --softwrap ${File2Patch}

    $dquilt refresh ${PatchFileName}

    if whiptail --title "Patch another" \
        --yesno "Do you want to patch another file in ${PatchFileName}?" \
        --yes-button "Yes" --no-button "No" --defaultno 15 60
    then
        FileToPatch
    fi
}

<CreateNewPatch 307>
```

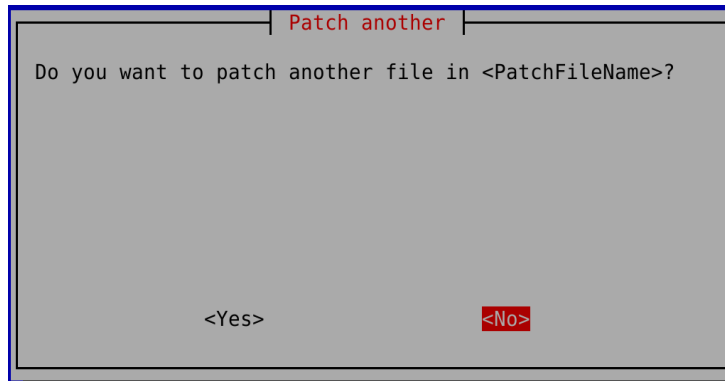


Abbildung 36.17.: Eine weitere Datei patchen?

Die Auswahl der zu korrigierenden Datei erfolgt in einem Dateiauswahldialog.

311 \langle FileSelector 311 \equiv (317)

```
function FileSelector {
    # Called by CreateNewPatch and itself
    # Dialog to select a file using whiptail

    StartPath=$1
    cd $StartPath
    txta=$(ls -a)

    i=0
    flist=''
    for element in ${txta[*]}
    do
        if [ $element == '.' ]
        then
            i=$(expr $i + 1)
            continue
        fi
        flist=$flist' '$i' '${element}
        i=$(expr $i + 1)
    done

    sel=$(whiptail --title "Filepicker" \
        --menu "Select:" 15 60 6 $flist 3>&2 2>&1 1>&3)
```

\langle FileSelector1 312 \rangle

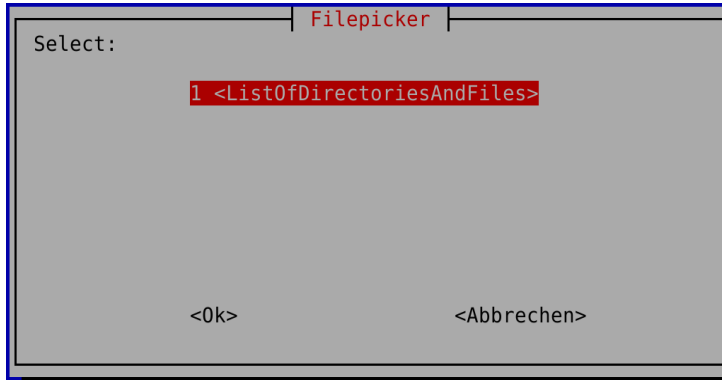


Abbildung 36.18.: Dateiauswahldialog

```

312  <FileSelector1 312>≡
      if [ $? -ne 0 ]
      then
        return
      fi

      # Go back
      if [ ${txta[$sel]} = '..' ]
      then
        cd ..
        StartPath=$(pwd)
        selected=${StartPath}
      else
        selected=${StartPath}/${txta[$sel]}
      fi

      # The order of the following if-clauses is important
      if [ -f ${selected} ]
      then
        if ! whiptail --title "Your choice;" \
          --yesno "${selected}\nContinue?" --yes-button "Yes" \
          --no-button "No" 15 60
        then
          FileSelector ${StartPath}
        fi
      fi

<FileSelector2 313a>

```

(311)

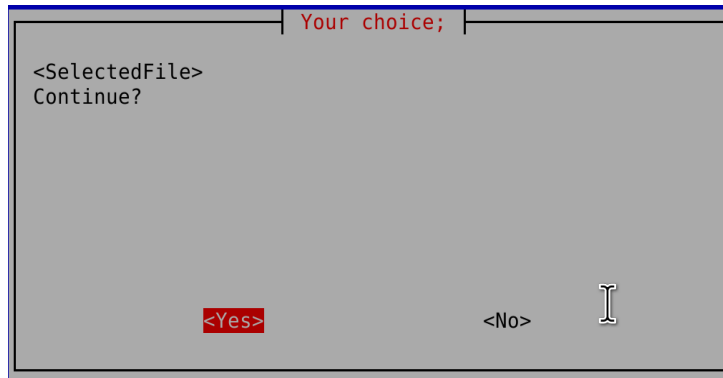


Abbildung 36.19.: Dateiauswahl bestätigen

```
313a  <FileSelector2 313a>≡ (312)
      if [ -d ${selected} ]
      then
          FileSelector ${selected}
      fi
  }
```

<FileToPatch 310>

Sind in diesem Verzeichnis alle Dateien vorhanden, wird mit *dquilt add* mit dem Parameter der zu bearbeitenden Datei diese in *quilt* eingebunden.

Damit wird in *.pc/DescriptionNoRegistration.patch/* die genannte Datei hinzugefügt. Dort steht sie dann für den ersten Abgleich zur Verfügung.

```
313b  <CreateNewPatch4 313b>≡ (309)
      PatchHeader
  }
```

<CreateDquilt 302a>

```
313c  <PatchTasks4 313c>≡ (305c)
      2) FileToPatch # Add patch to patch file
         PatchRunNr=1
         $dquilt refresh;;
  <PatchTasks6 313d>
```

36.5.1. Patch löschen

Mit der folgenden Auswahl werden bestehende Patches gelöscht. Dies ist notwendig, wenn ein bisheriger Patch nicht mehr benötigt wird, weil die Korrekturen inzwischen vom Upstream eingebaut wurden.

```
313d  <PatchTasks6 313d>≡ (313c)
      3) DeletePatches;; # Delete patches
  <PatchTasks7 315b>
```

Es kann aber auch sein, dass der bisherige Patch für die aktuelle Upstream-Version angepasst werden muss. Ein Weg hierzu ist, den alten Patch zu löschen und einen neuen zu erstellen (s. Kapitel 36.3.1, Seite 305).

```
314 <DeletePatches 314>≡ (315a)
function DeletePatches {
    # Called by PatchTasks and itself

    cd ${GitPath}
    DeletePatch
    PatchRunNr=1

    if whiptail --title "Another patch?" \
        --yesno "Do you want to delete another patch?" --yes-button "Yes" \
        --no-button "No" 15 60
    then
        DeletePatches
    fi
}

<RebasePQBranch 293a>
```


Wird die Frage mit *No* beantwortet, springt das Programm zur Auswahl der Patchaufgaben (Kapitel 36.3, Seite 302) zurück.

```
315a <DeletePatch 315a>≡ (319)
function DeletePatch {
    # Called by DeletePatches

    ShowPatches "delete" # String will be found in ${1}

    if [ -z "${PatchFileName}" ]
    then
        PatchTasks
    fi

    less --LINE-NUMBERS ${GitPath}/debian/patches/${PatchFileName}

    if whiptail --title "Delete this patch?" \
    --yesno "Do you really want to delete ${PatchFileName}?" \
    --yes-button "Yes" --no-button "No" 15 60
    then
        $dquilt delete -r --backup ${PatchFileName}

        if whiptail --title "Delete backup file?" \
        --yesno "Do you want to delete the backup file, too?" \
        --yes-button "Yes" --no-button "No" 15 60
        then
            rm ${GitPath}/debian/patches/${PatchFileName}~
        fi
    fi
}
```

<DeletePatches 314>

Damit die Funktion *gbp pq* genutzt werden, muss diese vorher eingerichtet werden (Kapitel 10.4.2.2, Seite 33).

```
315b <PatchTasks7 315b>≡ (313d)
    # Edit series
    4) nano --linenumbers --mouse --softwrap debian/patches/series;;
<PatchTasks8 316>
```

36.5.2. Ausgangszustand wiederherstellen

Wenn ein Patch mit *quilt* erstellt, bearbeitet oder gelöscht wurde, werden beim Verlassen der Funktion die Patches aus dem Upstream-Code entfernt und dieser in den ursprünglichen Zustand zurückversetzt. Dies geschieht mit *quilt pop -a*. Dies bedeutet, dass alle angewandten Patches entfernt werden.

```

316 <PatchTasks8 316>≡ (315b)
      5) if [ ${PatchRunNr} -eq 1 ]
          then
              # remove all patches and return the source
              # to its original state
              ${dquilt} pop -a
              PatchRunNr=0
          fi
          # If debian/patches/series is empty,
          # delete directory debian/patches
          if ! [ -s debian/patches/series ]
          then
              rm debian/patches/series
              rmdir debian/patches
          fi
          return;;
      esac
  fi

  PatchTasks
}

<GitBranch2RecentBranch 337a>

```

So ist es für Quilt möglich immer das Diff zwischen der Vorversion und der aktuellen Version zu erstellen. Danach erfolgt die Änderung. Die Informationen zur Registrierung werden entfernt. Mit *dquilt refresh* erzeugt man das Diff zur Dokumentation der Änderung. Mit *dquilt header -e* wird nun die Beschreibung der Änderung im \$EDITOR erstellt.

```
317 <PatchHeader 317>≡ (360a)
function PatchHeader {
    # Called by CreateNewPatch EditPatch

    PatchDescription=$(whiptail --title "Describe patch!" \
        --inputbox "Description:\n\n" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    if [ -z "${PatchDescription}" ]
    then
        echo "Please insert the description of the patch!"
        read PatchDescription
    fi
    if whiptail --title "Describe patch!" --yesno "Forwarded:?" \
        --yes-button "Yes" --no-button "No" 15 60
    then
        PatchForwarded="Yes"
    else
        PatchForwarded="No"
    fi

    DEBValues
    if [ -z ${Uploaders} ]
    then
        PatchAuthor=${Maintainer}
    else
        PatchAuthor=${Uploaders}
    fi

    PatchUpdate=$(date +%Y'-'%m'-'%d)

    touch ${PatchFileName}.header
    echo "Description: ${PatchDescription} >> ${PatchFileName}.header
    echo "Forwarded:    ${PatchForwarded} >> ${PatchFileName}.header
    echo "Author:       ${PatchAuthor} >> ${PatchFileName}.header
    echo "Last-Update:  ${PatchUpdate} >> ${PatchFileName}.header

    $dquilt header -a < ${PatchFileName}.header
    rm ${PatchFileName}.header
}
```

<FileSelector 311>

Hier ein Beispiel dazu:

```
Description: removed registration of license
Forwarded: No
Author: Mechtilde Stehmann <ooo@mechtilde.de>
Last-Update: 2014-05-18
```

36.6. Patch editieren

```
318  <EditPatch 318>≡ (301)
      function EditPatch {
          # Called by ChangePatches

          ShowPatches "edit" # String will be found in ${1}

          if [ -z "${PatchFileName}" ]
          then
              return 1
          else
              PatchRunNr=1
          fi

          $dquilt pop ${PatchFileName}
          nano --linenumbers --mouse --softwrap debian/patches/${PatchFileName}
          $dquilt refresh

          if whiptail --title "New Patch Header" \
          --yesno "Do you want to create a new patch header?" --yes-button "Yes" \
          --no-button "No" 15 60
          then
              PatchHeader
          fi

          while $dquilt push
          do
              $dquilt refresh
          done
      }

      <ChangePatches 319>
```

36.7. Patch bearbeiten

```
319 <ChangePatches 319>≡ (318)
    function ChangePatches {
        # Called by PatchTasks and itself

        cd ${GitPath}
        EditPatch

        if whiptail --title "Another patch?" \
            --yesno "Do you want to edit another patch?" --yes-button "Yes" \
            --no-button "No" 15 60
        then
            ChangePatches
        fi
    }

    <DeletePatch 315a>
```


37. Bauen

Das eigentliche Bauen der Binärpakete erfolgt in einer *chroot*. Hierbei wird hauptsächlich geprüft, ob alle benötigten Buildabhängigkeiten in der Datei *debian/control* (Kapitel 35.4.6, Seite 265) aufgeführt sind und bereits als Debian-Pakete zur Verfügung stehen.

Damit wird gewährleistet, dass das Paket auch von den FTP-Mastern reproduzierbar ohne Zugriff auf weitere Netzressourcen gebaut werden kann.

37.1. debian/changelog

Vor dem eigentlichen Bauen wird die Datei *debian/changelog* zur Anpassung angezeigt.

```
321a <BuildNewRevision6 321a>≡ (285a)
      # Check debian/changelog
      # - includes creating changelog using gbp dch
      DisplayDebianChangelog
```

```
<MovingGbpConf 331a>
```

Zunächst wird im Skript geprüft, ob eine Datei *debian/changelog* bereits existiert. Wenn dies der Fall ist, wird diese Datei angezeigt und abgefragt, ob sie korrekt ist.

Andernfalls wird sie mit *gbp dch* erzeugt.

```
321b <DisplayDebianChangelog 321b>≡ (278a)
      function DisplayDebianChangelog {
          # Called by BuildNewRevision

          newChangelog=0
          if [ -f debian/changelog ]
          then
              less --LINE-NUMBERS debian/changelog
              if ! whiptail --title "Changelog ok?" --defaultno \
                --yesno "Is debian/changelog ok?" --yes-button "Yes" \
                --no-button "No" 15 60
              then
                  newChangelog=1
              fi
          else
              newChangelog=1
          fi
      }
```

```
<DisplayDebianChangelog1 322>
```

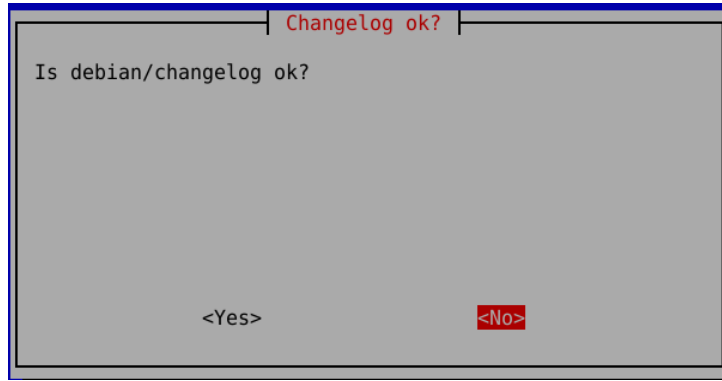


Abbildung 37.1.: Debian-Changelog OK?

Wird die Frage nach der Korrektheit des Changelogs bejaht, geht es mit einem eventuellen Verschieben der *gbp*-Konfigurationsdatei weiter (Kapitel 37.2, Seite 331).

Ist ein solches Verschieben der *gbp*-Konfigurationsdatei nicht erforderlich, werden die Parameter für *gbp buildpackage* festgelegt (Kapitel 37.3, Seite 331).

Andernfalls wird diese Datei im Editor angezeigt.

Zu achten ist besonders auf die Versions- und Revisionsbezeichnung in der ersten Zeile der Datei *debian/changelog*. Dies gilt hauptsächlich, wenn Dateien aus dem Upstream-Quellcode ausgeschlossen wurden (Kapitel 10.4.1.3, Seite 32).

Für einen Non-Maintainer-Upload (NMU) wird als erster Eintrag nach den Informationen zu der Version der Eintrag

```
* Non-maintainer upload
```

erwartet.

Existiert die Datei *debian/changelog* noch nicht, wird sie mit *gbp dch* erzeugt.

```
322 <DisplayDebianChangelog1 322>≡ (321b)
    # Check whether d/control exists without a comment in line 1
    if [ ${newChangelog} -eq 1 ]
    then
        if ! [ -f debian/control ]
        then
            DebianControlTemplate
        else
            set +e
            cat --number debian/control | grep '^1 ' | grep '#' > /dev/null
            if [ $? -eq 0 ]
            then
                set -e
                DebianControlTemplate
            fi
            set -e
        fi
        # creating changelog using gbp dch

        AddVersionNumber
    <DisplayDebianChangelog3 329>
```


37.1.1. Versionsbezeichnung einfügen

gbp dch benötigt die Angabe der aktuellen Versionsbezeichnung. Enthält die entsprechende Variable nicht diesen Wert, wird zur (manuellen) Eingabe dieser Versionsbezeichnung aufgefordert.

Dagegen kann *dch* die nächsthöhere Versionsbezeichnung mit *dch -increment* erstellen. Dies kann auch explizit mit *dch --newversion <Version>* erfolgen.¹

Das Programmskript nutzt die zweite Möglichkeit.

```
323a <AddVersionNumber 323a>≡ (324b)
function AddVersionNumber {
    # Called by DisplayDebianChangelog
    if [ -z "${Version1}" ]
    then
        RecentIdentifier
    fi
}
```

<AddVersionNumber1 325a>

Zunächst wird geprüft, welche Versionsbezeichnung in der ersten Zeile der Datei *debian/changelog* eingetragen ist. Es wird abgefragt, ob eine dort gefundene Versionsbezeichnung übernommen werden soll. Existiert die Datei *debian/changelog* nicht, wird die Versionsnummer abgefragt (s. Seite 327).

```
323b <RecentIdentifier 323b>≡ (328)
function RecentIdentifier {
    # Called by AddVersionNumber ForceOrig
    # Takes version number from debian/changelog, if it exists

    if [ -f ${GitPath}/debian/changelog ]
    then
        set +e
        firstLine=$(grep --line-number 'urgency=' ${GitPath}/debian/changelog | grep '^1:')
        set -e
        whiptail --title "First line:" \
        --msgbox "First line of debian/changelog;\n${firstLine}" 15 60
        recentId=$(echo ${firstLine} | sed --expression='s/^.*(//' | \
        sed --expression='s/).*//')
    fi
}
```

<RecentIdentifier2 324a>

¹New Maintainer Guide, Kap. 8.1[11]

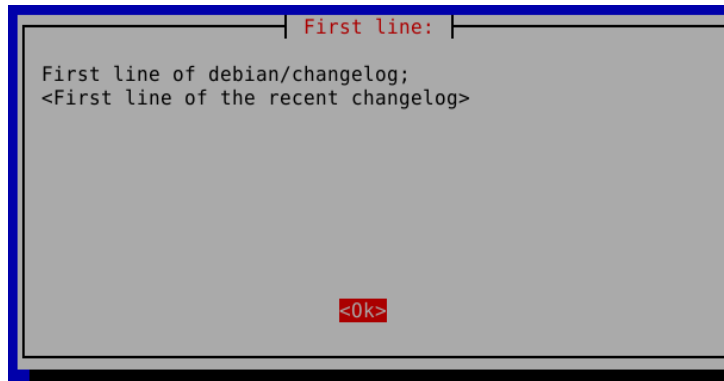


Abbildung 37.2.: Anzeige der ersten Zeile der Datei *debian/changelog*

324a $\langle \text{RecentIdentifier2 } 324a \rangle \equiv$ (323b)
`whiptail --title "Recent identifier" \
--msgbox "Recent identifier is ${recentId}" 15 60`

$\langle \text{RecentIdentifier3 } 324b \rangle$

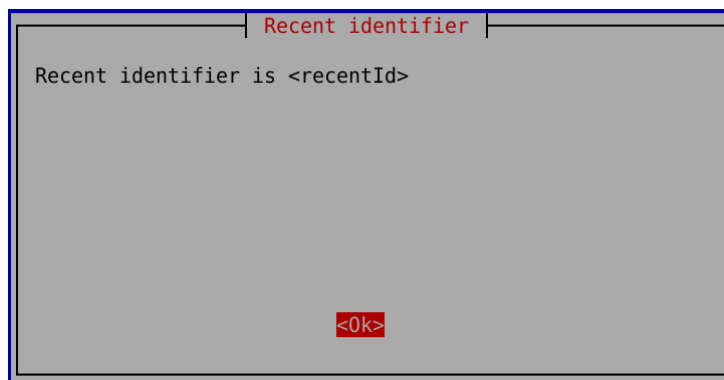


Abbildung 37.3.: Aktuelle Version

324b $\langle \text{RecentIdentifier3 } 324b \rangle \equiv$ (324a)
`if [-n "${recentId}"]
then
 Version1=${recentId}
fi
else
 InsertIdentifier
fi
}`

$\langle \text{AddVersionNumber } 323a \rangle$

Danach geht es mit Kontrollfragen weiter.

```
325a <AddVersionNumber1 325a>≡ (323a)
    set +e
    revisionflag=$(echo ${Version1} | grep --count '\-[0-9]')
    set -e
    if [ ${revisionflag} -eq 0 ]
    then
        if ! whiptail --title "Identifier of the version:" \
        --defaultno --yesno "${Version1} contains no revision number.\n \
        Is it a native package?" --yes-button "Yes" --no-button "No" 15 60
```

```
<AddVersionNumber2 325b>
```

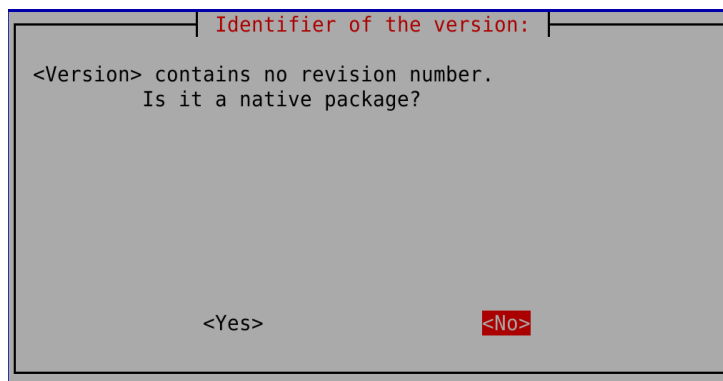


Abbildung 37.4.: Keine Revisionsnummer angegeben

```
325b <AddVersionNumber2 325b>≡ (325a)
    then
        InsertIdentifier
        if ! whiptail --title "Identifier of the version:" \
        --defaultno --yesno "Is ${Version1} the right identifier?" \
        --yes-button "Yes" --no-button "No" 15 60
```

```
<AddVersionNumber3 326>
```

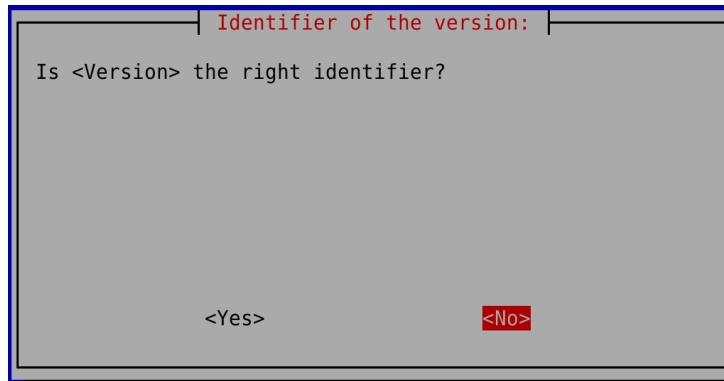


Abbildung 37.5.: Korrekte Versionsbezeichnung angeben?

```
326  <AddVersionNumber3 326>≡ (325b)
      then
        InsertIdentifier
      fi
    fi
  else
    if ! whiptail --title "Identifier of the version:" \
      --defaultno --yesno "Is ${Version1} the right identifier?" \
      --yes-button "Yes" --no-button "No" 15 60
    then
      InsertIdentifier
    fi
  fi
  echo "Message from AddVersionNumber: identifier=${Version1} >> ${log}"
}

<Debian.JavabuildTemplate 278a>
```

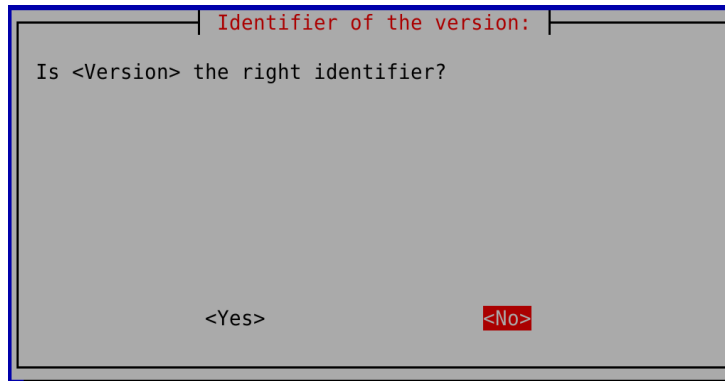


Abbildung 37.6.: Korrekte Versionsbezeichnung angeben?

In der Funktion *InsertIdentifier* wird zunächst der komplette Bezeichner des Paketes einschließlich der Versionsbezeichnung der Revision abgefragt.

327 \langle *InsertIdentifier* 327 $\rangle \equiv$ (335a)

```
function InsertIdentifier {
    # Called by AddVersionNumber Recentidentifier
    RIdentifier=${Version1}
    set +e
    cat debian/source/format | grep "native" > /dev/null
    if [ $? -eq 0 ]
    then
        Version1=$(whiptail --title "Identifier" \
            --inputbox "Recent identifier: ${RIdentifier}\n \
            Please insert the identifier of the package\n \
            (without revision version because it is a native package):" \
            --nocancel 15 60 3>&2 2>&1 1>&3)
```

\langle *InsertIdentifier1* 328 \rangle

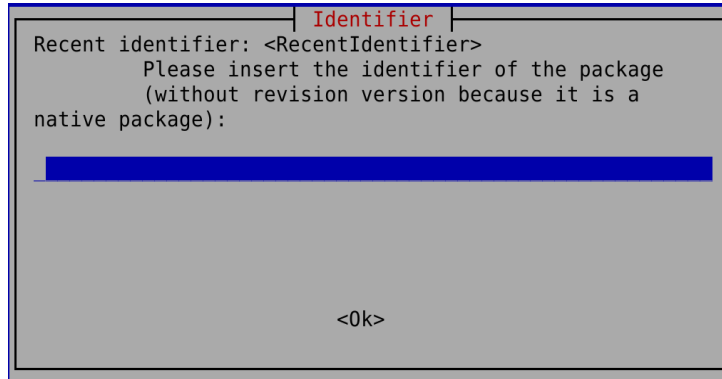


Abbildung 37.7.: Abfrage des Bezeichners eines nativen Paketes

```
328  <InsertIdentifier1 328>≡ (327)
      else
        Version1=$(whiptail --title "Identifier" \
          --inputbox "Recent identifier: ${RIdentifier}\n \
            Please insert the whole identifier of the package\n \
              (including revision version):" \
                --nocancel 15 60 3>&2 2>&1 1>&3)
        fi
        set -e
    }

    <RecentIdentifier 323b>
```

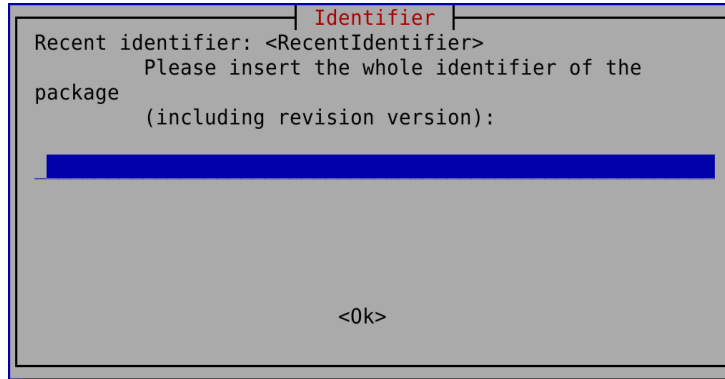


Abbildung 37.8.: Abfrage des Bezeichners

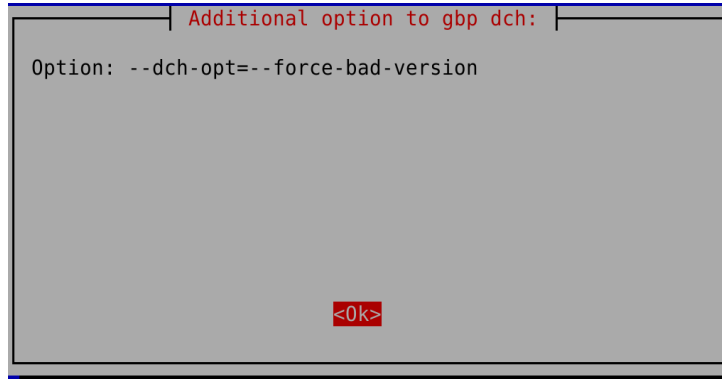
Anhand des Vorhandenseins einer Tilde wird geprüft, ob es sich um eine Schnappschuss-Version handelt. Ist dies der Fall, wird dem Befehl *gbp dch* auch die Option *--dch-opt=--force-bad-version* mitgegeben und dies angezeigt.

Dies bewirkt, dass das Programm *debchange* nicht stoppen wird, falls die neue Version kleiner als die aktuelle ist. Dies ist vor allem beim Rückportieren sinnvoll

```

329 <DisplayDebianChangelog3 329>≡ (322)
    set +e
    SnapshotFlag=$(echo ${Version1} | grep --count '~')
    if [ ${SnapshotFlag} -eq 0 ]
    then
        DchAdd=''
    else
        DchAdd=' --dch-opt=--force-bad-version'
        whiptail --title "Additional option to gbp dch:" \
            --msgbox "Option: ${DchAdd}" 15 60
    fi
<DisplayDebianChangelog4 330>

```

Abbildung 37.9.: Weitere Optionen für *dch*

Die Datei *debian/changelog* wird vom Programmskript mittels des Programms *gbp dch* erstellt. Immer wird *gbp dch* mit den Optionen *--verbose*, *--debian-branch=* und *--new-version=* aufgerufen.

Im Falle eines Fehlschlages von *gbp dch* wird eine Fehlermeldung ausgegeben. Andernfalls wird die Datei *debian/changelog* zum Editieren angezeigt.

```
330 <DisplayDebianChangelog 330>≡ (329)
      gbp dch --verbose --debian-branch=${RecentBranch} \
      --new-version=${Version1}${DchAdd}
      if [ $? -ne 0 ]
      then
          set -e
          FailureNotice
      fi
      set -e
      nano --linenumbers --mouse --softwrap debian/changelog
  fi
}
```

<CmeFix 282>

Für das Paket *tbsync* ergab sich folgende Befehlszeile zum Bauen einer Fehlerkorrektur für Buster, das seinerzeit das stabile Release war.

```
gbp dch --verbose --debian-branch=debian/buster \
--new-version=1.16-1-deb10u1 --dch-opt=--force-bad-version
```

Zum Hintergrund ist folgendes auszuführen:

Mit *--dch-opt=<dch-option>* können dem Befehl *gbp dch* Optionen für *debchange* (*dch*) mitgegeben werden. Dabei ist zu beachten, dass *gbp dch* das Programm *dch* mehrfach aufruft und die Option jeweils bei allen Aufrufen übergibt. Daher sind an dieser Stelle nicht alle *dch*-Optionen sinnvoll. Außerdem kann es passieren, dass Optionen im Widerspruch stehen zu Optionen, die von *gbp dch* selbstständig übergeben werden.

Nach dem Schließen des Editors wird regelmäßig der Dialog zur Prüfung des *Git*-Zweiges angezeigt (Kapitel 37.3.2, Seite 335).

37.2. Verschieben der *gbp*-Konfigurationsdatei

Wenn für das zu bauende Paket eine spezielle Konfigurationsdatei für *git-buildpackage* verwandt werden soll (Kapitel 20.3, Seite 75), wird diese im Verzeichnis *debian/* mitveröffentlicht.

Wurde für *gbp import-orig* beim (erstmaligen) Herunterladen des Quellcodes eine solche Konfigurationsdatei erstellt (Kapitel 34.4.9, Seite 232), ist diese gegebenenfalls in das Verzeichnis *debian/* zu verschieben. Dies geschieht durch die Funktion *MovingGbpConfFile*.

```
331a <MovingGbpConf 331a>≡ (321a)
      MovingGbpConfFile
      <Preparations 331c>
```

Existiert eine Datei *gbp.conf* im Verzeichnis *.git/*, aber keine im *debian/*-Verzeichnis, so wird diese Datei nach *debian/* verschoben.

Sind in beiden Verzeichnissen entsprechende Dateien vorhanden, kann die zu veröffentliche Datei ausgewählt werden.

```
331b <MovingGbpConfFile 331b>≡ (236)
      function MovingGbpConfFile {
          # Called by BuildNewRevision

          # .git/gbp.conf exists, but not debian/gbp.conf
          # Move gbp.conf from .git/ to debian/
          if [ -f ${GitPath}/.git/gbp.conf -a ! -f ${GitPath}/debian/gbp.conf ]
          then
              mv -iv ${GitPath}/.git/gbp.conf ${GitPath}/debian
          fi
          # There is a gbp.conf in both directories
          if [ -f ${GitPath}/.git/gbp.conf -a -f ${GitPath}/debian/gbp.conf ]
          then
              TwoConfFilesFound
          fi
      }
```

<DebianBranch4Import 228>

Die Funktion *TwoConfFilesFound*, welche diese Auswahl ermöglicht, wird in (Kapitel 34.4.9, Seite 232) beschrieben.

37.3. Parameter für *gbp buildpackage* festlegen

Bevor das Bauen der Pakete beginnen kann, müssen zunächst Parameter für *gbp buildpackage* ermittelt und festgelegt werden.

```
331c <Preparations 331c>≡ (331a)
      # Preparations for gbp buildpackage
      AskDist # Ensure that RecentBranch has a value
      <Preparations1 335b>
```

37.3.1. Git-Zweig und Distribution ermitteln

Damit im richtigen Git-Zweig und mit der richtigen Distribution gebaut wird, werden diese Parameter ermittelt und angezeigt. Sie können auch noch vom Benutzer angepasst werden.

Zunächst werden mit der Funktion *IdentifyBranches* die vorhandenen Git-Zweige ermittelt (Kapitel 32.5.2, Seite 164).

```
332a <AskDist 332a>≡ (337b)
function AskDist {
    # Called by BuildNewRevision PrepareUploading LastQuestionsBeforeBuild

    IdentifyBranches
    ba=$(bl)
    set +e
    for element in ${ba[*]}
    do
        # rb=$(echo ${element} | grep --count '^x_')
        # if [ $rb -ge 1 ]
        if echo ${element} | grep --quiet '^x_'
        then
            CurrentBranch=$(echo ${element} | sed --expression='s/^x_//')
        fi
    done
    set -e
}
```

<AskDist0 332b>

Zunächst wird mit *-z* geprüft, ob die Variable *RecentBranch* leer ist. In diesem Fall wird die Funktion *GitBranch2RecentBranch* (Kapitel 37.3.3, Seite 337) aufgerufen.

Andernfalls wird geprüft, ob der Name des aktuellen Zweiges dem Wert der Variablen *RecentBranch* entspricht. Sollte dies nicht der Fall sein, erfolgt ein Hinweis und der Nutzer kann einen der beiden Zweige auswählen.

```
332b <AskDist0 332b>≡ (332a)
if [ -z "${RecentBranch}" ]
then
    GitBranch2RecentBranch
else
    if [ "${RecentBranch}" != "${CurrentBranch}" ]
    then
        Msg="Branch according to git: "${CurrentBranch}","\n \
        branch according to "${ConfigPath}${OrigName}": "${RecentBranch}
        whiptail --title "There is something wrong!" --msgbox "${Msg}" 15 60
    fi
fi
<AskDist1 333>
```

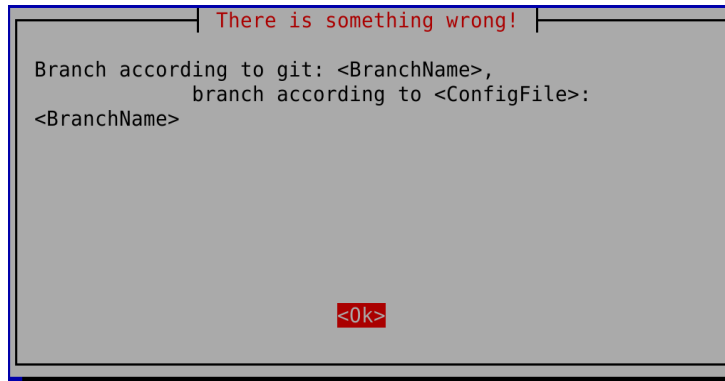


Abbildung 37.10.: Da läuft was falsch!

```

333  <AskDist1 333>≡
      WishedBranch=$(whiptail --title "Choose branch:" \
      --radiolist "Which branch do you want to work with? " \
      --cancel-button "Cancel" 15 60 2 \
      "0" "${RecentBranch}" off \
      "1" "${CurrentBranch}" off 3>&2 2>&1 1>&3)
<AskDist2 334>

```

(332b)

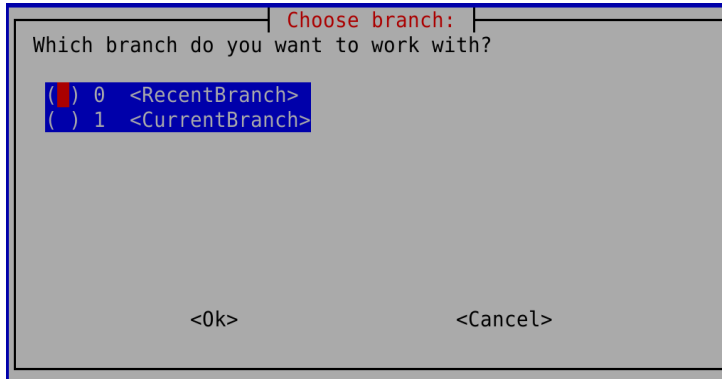


Abbildung 37.11.: Auswahl des Branches

334 `<AskDist2 334>≡` (333)

```
if [ ${WishedBranch} -eq 0 ]
then
    git checkout ${RecentBranch}
else
    GitBranch2RecentBranch
fi
fi
fi

echo "Notice from AskDist: The branch is "${RecentBranch} >> ${log}
set +e
va=$(grep --count ${RecentBranch}_Dist ${ConfigPath}${OrigName})
if [ $va -eq 1 ]
then
    bName=${RecentBranch}
    set -e
    Search4Dist
    RecentBranchD=${va}
elif [ $va -gt 1 ]
then
    nano ${ConfigPath}${OrigName}
    AskDist
```

`<AskDist5 335a>`

```

335a  <AskDist5 335a>≡ (334)
      else
        set -e
        Distro4Branch
      fi
      set -e

      if [ -z "${RecentBranchD}" ]
      then
        RecentBranchD="sid"
      fi
      echo "Notice from AskDist: The distribution is "${RecentBranchD} >> ${log}
    }

    <InsertIdentifier 327>

```

37.3.2. Git-Zweig und Distribution prüfen

Der Git-Zweig wird angezeigt, damit der Nutzer prüfen kann, ob es der richtige ist.

```

335b  <Preparations1 335b>≡ (331c)
      echo "Notice from BuildNewRevision: Branch is "${RecentBranch} >> ${log}
      whiptail --title "Please check!" \
        --yesno "The git branch is ${RecentBranch}" --yes-button "Yes" \
        --no-button "No" 15 60
      rbq=$?
    <Preparations2 335c>

```

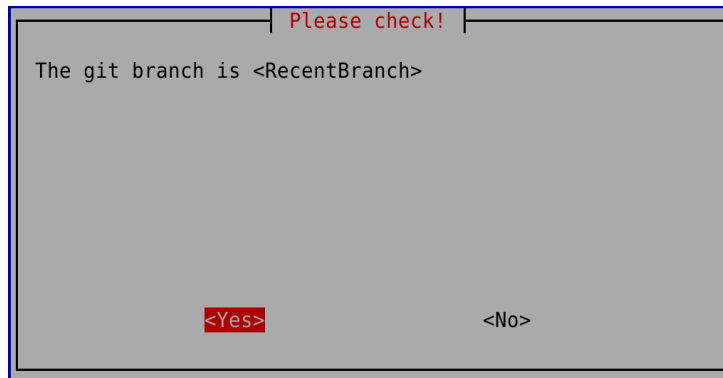


Abbildung 37.12.: Anzeige des Git-Zweiges

Wird die Frage, ob der angezeigte Git-Zweig der richtige ist, verneint, kann der zutreffende Zweig ausgewählt werden (Kapitel 33.4, Seite 182).

```

335c  <Preparations2 335c>≡ (335b)
      if [ $rbq -ne 0 ]
      then
        SelectBranch
      fi
    <Preparations3 336a>

```

Wird die Frage, ob die angezeigte Distribution die richtige ist, verneint, kann die zutreffende Distribution ausgewählt werden (Kapitel 32.5.3, Seite 165).

```
336a  <Preparation3 336a>≡ (335c)
      if ! whiptail --title "Please check!" \
        --yesno "The distribution is ${RecentBranchD}" --yes-button "Yes" \
        --no-button "No" 15 60
      <Preparation4 336b>
```

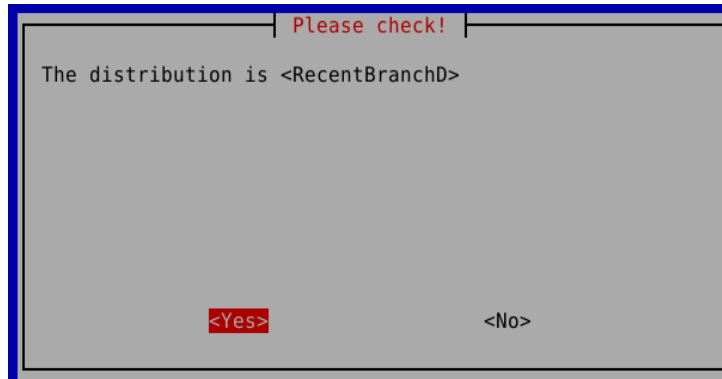


Abbildung 37.13.: Release-Branch der Distribution

```
336b  <Preparation4 336b>≡ (336a)

      then
        RecentBranchD=""
        Distro4Branch
      fi

      echo "Notice from BuildNewRevision: Distribution is "${RecentBranchD} >> ${log}

      SBuildOrPBuilder 0

      <BuildNewRevision8 352>
```

Zur Auswahl des Build-Systems wird die Funktion *SBuildOrPBuilder* aufgerufen (Kapitel 37.3.5, Seite 338).

37.3.3. Git-Zweig anpassen

Zunächst wird im Wert der Variablen *CurrentBranch* (welcher der Name des aktuellen Zweiges ist) ein "/" durch "\/" ersetzt, also ein Schrägstrich maskiert.

Dann wird der so veränderte Wert der Variablen als *RecentBranch* in die Konfigurationsdatei eingefügt.

Schließlich wird dieser Wert der Variablen *RecentBranch* zugewiesen.

```
337a <GitBranch2RecentBranch 337a>≡ (316)
function GitBranch2RecentBranch {
    # Called by AskDist

    bName1=$(echo ${CurrentBranch} | sed --expression='s/\//\\//g')
    sed --in-place --expression="s/RecentBranch=.*\/RecentBranch=${bName1}/g" \
    ${ConfigPath}${OrigName}
    RecentBranch=${CurrentBranch}
}

<Search4Dist 337b>
```

37.3.4. Distribution ermitteln

```
337b <Search4Dist 337b>≡ (337a)
function Search4Dist {
    # Called by AskDist ParseConfig
    set +e
    va=$(grep ${bName}_Dist ${ConfigPath}${OrigName})
    bName1=$(echo ${bName} | sed --expression='s/\//\\//g')
    va=$(echo $va | sed --expression="s/# ${bName1}_Dist=//g")
    va=$(echo $va | sed --expression='s/"//g')
    set -e
}

<AskDist 332a>
```

37.3.5. Auswahl des Build-Systems

Nun kann noch die Auswahl getroffen werden, ob mit *pbuilder* oder *sbuild* gebaut werden soll.

```
338a <SBuildOrPBuilder 338a>≡ (351)
function SBuildOrPBuilder {
    # Called by BuildNewRevision TaskSelect PrepareUploading Import4Sponsoring
    # Flag for tagging and signing
    tsf=$1

    Builder=$(whiptail --title "Which builder do you want to use?" \
        --radiolist "Which builder do you want to use? " 15 60 6 \
        "0" "PBuilder" off \
        "1" "SBuild" on \
        --cancel-button "Exit" 3>&2 2>&1 1>&3)
    <SBuildOrPbuilder1 338b>
```

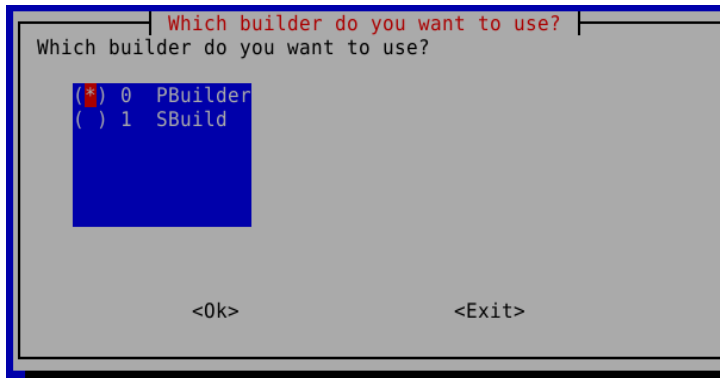


Abbildung 37.14.: Auswahl des Build-Systems

Voreingestellt ist die Auswahl von *sbuild*.

```
338b <SBuildOrPbuilder1 338b>≡ (338a)
    if [ -z "${Builder}" ]
    then
        exit
    fi
    if [ ${Builder} -eq 1 ]
    then
        echo "Using SBuild." >>${log}
        UsingSBuild
    else
        echo "Using PBuilder." >>${log}
        UsingPBuilder
    fi
}
```

<BuildNewRevision 254>

Wird *SBuild* ausgewählt, ruft das Programmskript die Funktion *UsingSBuild* auf. Wird *PBuilder* ausgewählt, wird die Funktion *UsingPBuilder* aufgerufen.

Eine Beschreibung des *PBuilders* finden Sie im Kapitel 37.5 (Seite 343).

Was *sbuild* macht, erfährt man in Kapitel 37.4 (Seite 341).

Zuvor erhält der Nutzer jedoch die Gelegenheit vor dem Bau des Paketes die Parameter letztmalig zu prüfen. Außerdem wird ihm eine letzte Ausstiegsmöglichkeit gewährt (Kapitel 37.3.7, Seite 339).

37.3.6. Überprüfung der Parameter

Bevor der Paketbau endgültig beginnt, werden die Parameter letztmalig zur Prüfung angezeigt.

```
339a <LastQuestionsBeforeBuild 339a>≡ (287c)
function LastQuestionsBeforeBuild {
    # Called by UsingSBuild UsingPBuilder

    if ! whiptail --title "Please check!" \
        --yesno "The release you want to build for in ${BuildEnv} is ${RecentBranchD}" \
        --yes-button "Yes" --no-button "No" 15 60
    then
        AskDist
    fi

<LastQuestionsBeforeBuild1 339b>
```

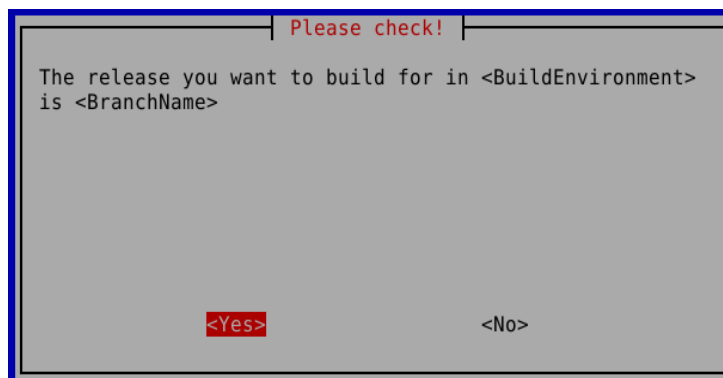


Abbildung 37.15.: Parameterüberprüfung

37.3.7. Letzte Ausstiegsmöglichkeit

Vor dem Paketbau wird schließlich eine letzte Gelegenheit zum Ausstieg gegeben.

```
339b <LastQuestionsBeforeBuild1 339b>≡ (339a)
    whiptail --title "Last opportunity to exit before building" \
        --yesno "Do you want to start the build process?" --yes-button "Yes" \
        --no-button "Exit" 15 60
<LastQuestionsBeforeBuild2 340a>
```

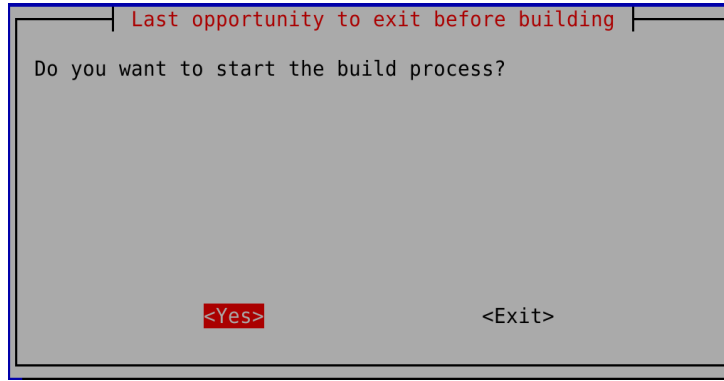


Abbildung 37.16.: Bau des Paketes starten

```
340a  <LastQuestionsBeforeBuild2 340a>≡ (339b)
      if [ $? -ne 0 ]
      then
        whiptail --title "Bye" --msgbox "Exit" 15 60
        <LastQuestionsBeforeBuild3 340b>
```

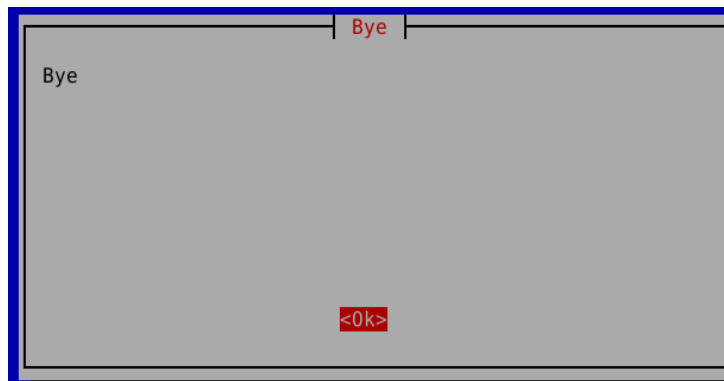


Abbildung 37.17.: Beenden

```
340b  <LastQuestionsBeforeBuild3 340b>≡ (340a)
      exit
    else
      if [ "${BuildEnv}" = "pbuilder" ]
      then
        whiptail --title "Start building" \
          --msgbox "${BuildEnv} will be updated first\n \
          This need sudo and/or root rights" 15 60
      fi
    fi
  }

  <UsingSBuild 341a>
```

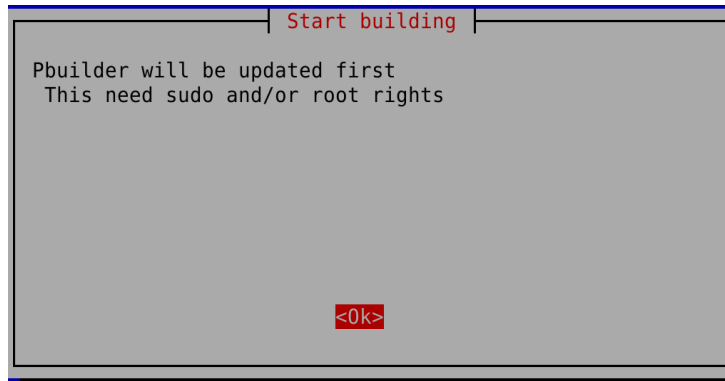


Abbildung 37.18.: Information zur Aktualisierung der Build-Umgebung

37.4. Was macht *sbuild*?

Sbuild wird im offiziellen *buildd*-Netzwerk verwendet, um Binär- und Quellpakete für alle unterstützten Architekturen zu erstellen[34].

Es wird dafür jeweils eine eigene Buildumgebung erstellt.

341a `<UsingSBuild 341a>≡` (340b)

```
function UsingSBuild {
    # Called by BuildNewRevision
```

```
    BuildEnv="sbuild"
```

`<UsingSBuild1 341b>`

Zunächst erfolgt der Aufruf der Funktion *LastQuestionsBeforeBuild* zur Festlegung der Parameter *release* und des dazugehörigen Git-Zweiges für *gbp buildpackage* (Kapitel 37.3.7, Seite 339).

Dann besteht die Gelegenheit, die Datei `~/.sbuildrc` zu prüfen und gegebenenfalls anzupassen.

341b `<UsingSBuild1 341b>≡` (341a)

```
LastQuestionsBeforeBuild
```

```
if whiptail --title ".sbuildrc." \
--yesno "Do you want to check (and edit) ~/.sbuildrc?" \
--defaultno --yes-button "Yes" --no-button "No" 15 60
then
```

```
    nano ~/.sbuildrc
```

```
fi
```

`<UsingSbuild2 342>`

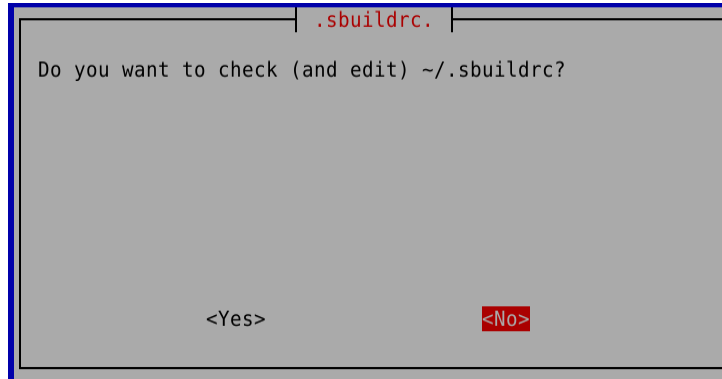


Abbildung 37.19.: .sbuildrc prüfen

Wenn die Datei `~/.sbuildrc` leer ist, ist diese mit dem beschriebenen Inhalt zu füllen (s. Kapitel 19.4, Seite 71)

Abhängig davon, zu welchem Zwecke der Paketbau erfolgen soll, wird `gbp buildpackage` mit einer unterschiedlichen Parameterkonfiguration aufgerufen.

342 `<UsingSbuild2 342>≡` (341b)

```
# Check whether chroot directory exists

if ! [ -d ~/.cache/sbuild ]
then
    mkdir -p ~/.cache/sbuild
fi

# Create/Update tarballed chroot

mmdebstrap --variant=buildd ${RecentBranchD} \
~/.cache/sbuild/${RecentBranchD}-amd64.tar.xz

set +e
if [ $tsf -eq 0 ]
then
    gbp buildpackage --git-builder=sbuild \
    --git-debian-branch=${RecentBranch} \
    --git-dist=${RecentBranchD} --git-ignore-new
    # Setting flag for success
    gbpq=$?
elif [ $tsf -eq 1 ]
then
    gbp buildpackage --git-builder=sbuild \
    --git-debian-branch=${RecentBranch} \
    --git-dist=${RecentBranchD} --git-verbose --git-tag \
    --git-sign-tags
    # Setting flag for success
    gbpq=$?
else
    gbp buildpackage --git-builder=sbuild \
```

```

        --git-debian-branch=${RecentBranch} \
        --git-dist=${RecentBranchD}
        # Setting flag for success
        gbpq=$?
    fi
    set -e
}
⟨UsingPBuilder 345⟩

```

37.5. In der *Pbuilder-Chroot* bauen

Mit *PBuilder* wird eine Umgebung geschaffen, in der Debian-Pakete erstellt werden können (s. a. Kapitel 19.3, Seite 62).

Die Pakete werden in einer *chroot* gebaut, die von *pbuilder* bereitgestellt wird. Zur besseren Kontrolle des Prozesses können an vordefinierten Stellen sogenannte *hooks* eingebaut werden (Kapitel 19.3.3, Seite 67).

Da *git pbuilder* zur Aktualisierung der Basisinstallation *root*-Rechte benötigt, erfolgt eine Passwortabfrage.

37.5.1. *base.cow* erstellen

Ist die erforderliche *base-cow* noch nicht vorhanden, wird sie mit *git-pbuilder create* angelegt. Damit kann auch eine *chroot* für eine abweichende Architektur wie zum Beispiel *i386 (32-bit)* auf einem *64-bit*-Rechner angelegt werden. Dazu wird eine *base-sid-i386.cow* mit

```
DIST=sid ARCH=i386 git-pbuilder create
```

erstellt.

```

343  ⟨CreateNewCow 343⟩≡ (164b)
    function CreateNewCow {
        # Called by Distro4Branch UsingPBuilder

        bDist="sid"
        set +e
        bDist=$(whiptail --title "Create new cow for ${RecentBranch}" \
            --inputbox "Debian distribution\n \
            for this branch ${RecentBranch}:" \
            --cancel-button "Use Sid" 15 60 3>&2 2>&1 1>&3)
    }
    ⟨CreateNewCow1 344⟩

```

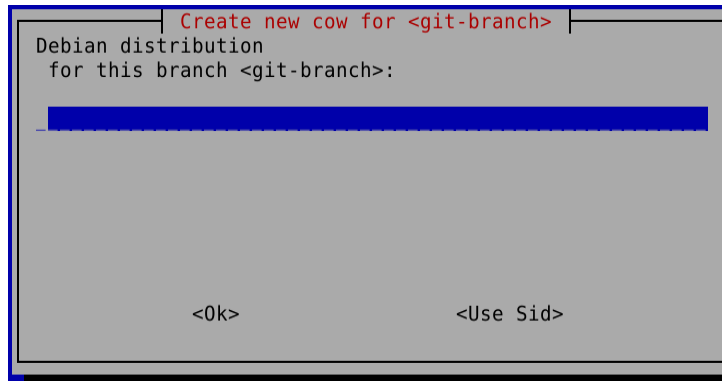


Abbildung 37.20.: Auswahl der zu erstellenden Cow.

```
344  <CreateNewCow1 344>≡ (343)
      if [ -z "${bDist}" ]
      then
          bDist="sid"
      fi
      # You must be root to create a cow
      echo -e "\nPlease enter Password for creating pbuilder cow.\n"
      sudo DIST=${bDist} git-pbuilder create
      echo "Cow for "${bDist}" created." >> ${log}
      set -e
  }

  <Distro4Branch 165>
```

37.5.2. git-pbuilder update

Vor dem Start von *gbp buildpackage* sind Vorkehrungen zu treffen, damit die Einrichtung in der *chroot* aktuell ist. Dies geschieht mittels *git-pbuilder update*. Zuvor wird noch geprüft, ob die notwendige *base-cow* (des entsprechenden Branches/Releases) vorhanden ist.

Während die *cow*-Verzeichnisse für die übrigen Veröffentlichungen die Release-Bezeichnungen im Namen tragen, heißt das entsprechende Verzeichnis für *sid* schlicht *base.cow*.

Daher wird anhand zweier Bedingungen geprüft, ob das entsprechende Verzeichnis vorhanden ist.

Fehlt das entsprechende Verzeichnis wird es mittels der Funktion *CreateNewCow* (Kapitel 37.5.1, Seite 343) erstellt.

345 `<UsingPBUILDER 345>≡` (342)

```
function UsingPBUILDER {
    # Called by BuildNewRevision

    BuildEnv="pbuilder"
    LastQuestionsBeforeBuild

    # Building package using git-pbuilder and gbp buildpackage

    # check, whether cow exists
    # if exists update cow
    # else create cow
    if [ -d /var/cache/pbuilder/base- $\{\text{RecentBranchD}\}$ .cow ]
    then
        echo -e "\nPlease enter Password for updating pbuilder cow.\n"
        DIST= $\{\text{RecentBranchD}\}$  git-pbuilder update
        echo "Notice from BuildNewRevision: Pbuilder was updated." >>  $\{\text{log}\}$ 
    elif [ -d /var/cache/pbuilder/base.cow -a  $\{\text{RecentBranchD}\}$  = "sid" ]
    then
        echo -e "\nPlease enter Password for updating pbuilder cow.\n"
        DIST= $\{\text{RecentBranchD}\}$  git-pbuilder update
        echo "Notice from BuildNewRevision: Pbuilder was updated." >>  $\{\text{log}\}$ 
    else
        CreateNewCow
    fi

    if [  $\{\text{OptFlag}\}$  -ne 1 ]
    then
        ForceOrig
    <UsingPBUILDER3 346a>

```

Für die Anzeige der bereits gesetzten Optionen für *gbp buildpackage* in der Funktion *MoreOptions* werden diese mit der Variablen *normalOpts* übergeben.

```
346a  <UsingPBuilder3 346a>≡ (345)
    if [ $tsf -eq 0 ]
    then
        normalOpts="--git-debian-branch=${RecentBranch}" \
        --git-dist=${RecentBranchD}" --git-verbose \
        --git-ignore-new${pbuilderOpt}
    elif [ $tsf -eq 1 ]
    then
        normalOpts="--git-debian-branch=${RecentBranch}" \
        --git-dist=${RecentBranchD}" --git-verbose --git-tag \
        --git-sign-tags${pbuilderOpt}
    else
        normalOpts="--git-debian-branch=${RecentBranch}" \
        --git-dist=${RecentBranchD}" --git-verbose${pbuilderOpt}
    fi
    MoreOptions
fi
<UsingPBuilder4 351>
```

37.5.3. Aufnahme des *.orig-Archives in *.changes

Grundsätzlich wird das *.orig-Archiv (**.orig.tar.gz* oder **.orig.tar.xz*) nur dann in die *.changes-Datei eingebunden und damit auch hochgeladen, wenn die Revisionsnummer 1 nicht übersteigt.

Zunächst wird also die Versionsbezeichnung durch Aufruf der Funktion *RecentIdentifier* (Kapitel 37.1.1, Seite 323) ermittelt und angezeigt.

```
346b  <ForceOrig 346b>≡ (259c)
function ForceOrig {
    # Called by BuildNewRevision PrepareUploading
    OptFlag=1
    if [ -z "${Version1}" ]
    then
        RecentIdentifier
    fi
    whiptail --title "Version" --msgbox "Version: ${Version1}" 15 60
}
<ForceOrig2 347a>
```

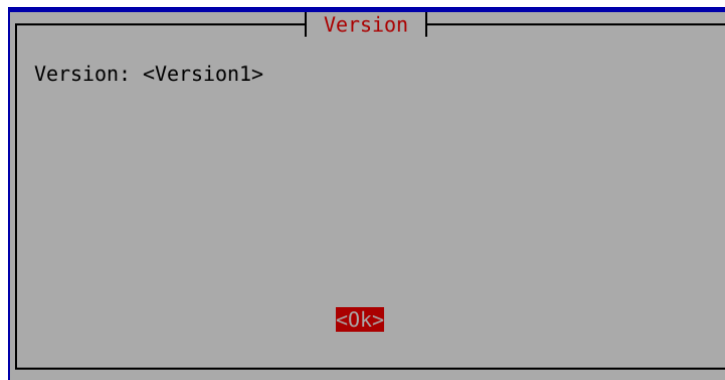



Abbildung 37.21.: Anzeige der Version mit Revisionsnummer

Es wird geprüft, ob es sich um ein *natives* Paket (Kapitel 16.1, Seite 53) handelt.

```
347a <ForceOrig2 347a>≡ (346b)
      set +e
      cat debian/source/format | grep "native" > /dev/null
      if [ $? -ne 0 ]
      <ForceOrig3 347c>
```

Handelt es sich um ein natives Paket, so enthält die Versionsbezeichnung keine Revisionsbezeichnung.

```
347b <ForceOrig7 347b>≡ (349a)
      else
          pbuilderOpt=" --git-pbuilder"
      fi
      set -e
  }
```

<MoreOptions 349b>

Bei nicht-nativen Debian-Paketen wird die Revisionsnummer extrahiert und angezeigt.

```
347c <ForceOrig3 347c>≡ (347a)
      then
          RevNr=$(echo ${Version1} | sed --expression='s/[^0-9]/#/g' | \
          sed --expression='s/^.*/#/' )
          whiptail --title "Revision number" \
          --msgbox "The number of the revision is ${RevNr}." 15 60
          pbuilderOpt=" --git-builder=git-pbuilder \
          --git-pbuilder-options='--source-only-changes'"
```

<ForceOrig5 348>

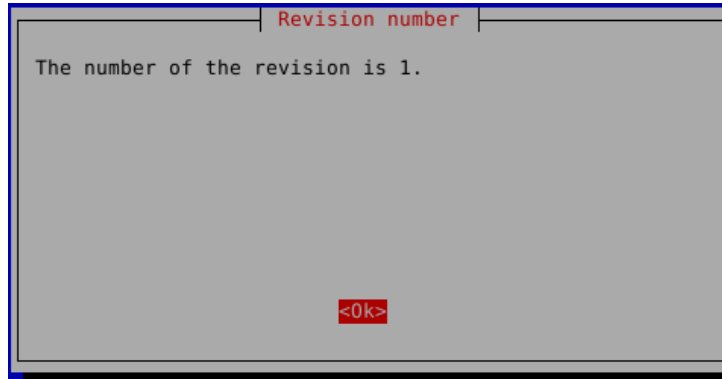


Abbildung 37.22.: Anzeige der Revisionsnummer

Man kann jedoch bei einer Revisionsnummer größer als 1 erzwingen, dass das *.orig-Archiv in die *.changes-Datei eingebunden und damit auch hochgeladen wird, indem *git-pbuilder* die Option *-sa* mitgegeben wird.

Dies ist dann sinnvoll, wenn das *orig-Archiv bislang noch nicht hochgeladen wurde oder in der *New Queue* erneut bereitgestellt werden muss.

Bei der Verwendung von *gbp buildpackage*, wie im Skript, lautet die entsprechende Option *-git-builder=git-pbuilder -sa*.

```
348 <ForceOrig5 348>≡ (347c)
    if [ ${RevNr} -gt 1 ]
    then
        if whiptail --title "orig in changes file" \
        --yesno "Do you want to insert the orig archive into the changes file?\n \
        That makes sense, if the orig archive has not been uploaded before." \
        --defaultno --yes-button "Yes" --no-button "No" 15 60
    <ForceOrig6 349a>
```

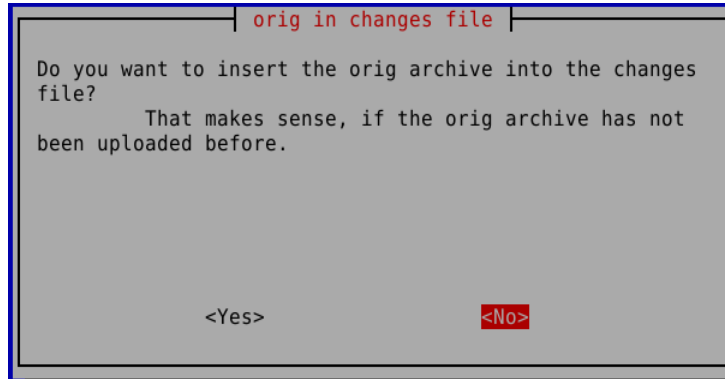


Abbildung 37.23.: Soll der Upstream-Tarball auch hochgeladen werden

Die Option `-sa` bedeutet, dass laut `dpkg-buildpackage -help` die Quelle immer *Orig* enthält.

```
349a <ForceOrig6 349a>≡ (348)
      then
          pbuilderOpt=" --git-builder=git-pbuilder -sa"
      fi
  fi
  <ForceOrig7 347b>
```

Nun wird in der folgenden Funktion die Möglichkeit eröffnet, weitere Optionen für den *pbuilder* in *gpb buildpackage* mitzugeben. Zuvor werden die bisherigen Optionen angezeigt.

```
349b <MoreOptions 349b>≡ (347b)
      function MoreOptions {
          # Called by UsingPBuilder PrepareUploading
          # Adds options to specify pbuilder in gpb buildpackage
          moreOpts=''
          intText="The options for gpb buildpackage are:\n"
          intText=${intText}${normalOpts}
          if whiptail --title "Options for gpb buildpackage" \
              --yesno "${intText}\nDo you want to add some more?" --yes-button "Yes" \
              --no-button "No" --defaultno 15 60
          <MoreOptions2 350>
```



Abbildung 37.24.: Anzeige der Optionen von *gbp buildpackage*

```
350  <MoreOptions2 350>≡ (349b)
      then
        moreOpts=$(whiptail --title "Options for gbp buildpackage" \
          --inputbox "${intText}\nPlease insert options to be added:" \
          --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
        moreOpts=" ${moreOpts}"
      fi
    }

    <PatchesTreatment 285b>
```

Die Informationen zu *pbuilderOpt* wurden aus verschiedenen Manpages zusammengetragen.

In der Manpage zu *gbp buildpackage* erhält man die Information, dass mit *-git-pbuilder-options=PBUILDER_OPTIONS* weitere Optionen des *pbuilders* hinzugefügt werden können. Welche Optionen dies sind, findet man in der Manpage von *pbuilder*

37.5.4. Bauen mit *gbp buildpackage*

Nach den Vorbereitungen erfolgt nun das Bauen des jeweiligen Paketes mit *gbp buildpackage*. Misslingt dies, beendet sich das Programm. Andernfalls geht es weiter zur Überprüfung der gebauten Pakete (Kapitel 40, Seite 357). Good luck!

Hier beginnt nun mit dem Herunterladen der Build-Abhängigkeiten das eigentliche Bauen des Debian-Paketes.

```

351 <UsingPBuilder4 351>≡ (346a)
    echo "== Options for gbp buildpackage ==" >> ${log}
    echo -e "RecentBranch contains ${RecentBranch}" >> ${log}
    echo -e "RecentBranchD contains ${RecentBranchD}" >> ${log}
    echo -e "PBuilderOpt contains ${pbuilderOpt}" >> ${log}
    echo -e "MoreOpts contains ${moreOpt} \n" >> ${log}
    echo "Starting gbp buildpackage" >> ${log}
    set +e

    if [ $tsf -eq 0 ]
    then
        gbp buildpackage --git-debian-branch=${RecentBranch} \
            --git-dist=${RecentBranchD} --git-verbose \
            --git-ignore-new${pbuilderOpt}${moreOpts}
        # Setting flag for success
        gbpq=$?
    elif [ $tsf -eq 1 ]
    then
        gbp buildpackage --git-debian-branch=${RecentBranch} \
            --git-dist=${RecentBranchD} --git-verbose --git-tag \
            --git-sign-tags${pbuilderOpt}${moreOpts}
        # Setting flag for success
        gbpq=$?
    else
        gbp buildpackage --git-debian-branch=${RecentBranch} \
            --git-dist=${RecentBranchD} --git-verbose${pbuilderOpt}${moreOpts}
        # Setting flag for success
        gbpq=$?
    fi
    set -e
}

<SBuildOrPBuilder 338a>

```

```

352  <BuildNewRevision8 352>≡ (336b)
      if [ $gbpq -eq 0 ]
      then
          echo -e "The package ${SourceName} was built with gbp buildpackage\n \
          without creating and signing tags." >> ${log}
      else
          whiptail --title "Build failed!" \
          --msgbox "Gbp buildpackage failed!" 15 60
          echo
          echo "-- Gbp buildpackage failed! --"
          echo
          echo "Please fix the problem in another terminal!"
          echo "After fixing, press RETURN to continue."
          read a
          if [ "${BuildEnv}" = "sbuild" ]
          then
              UsingSBuild
          else
              UsingPBuilder
          fi
      fi
      Task=5 # Go to RunningTests
}

```

<RunningLintian 360b>

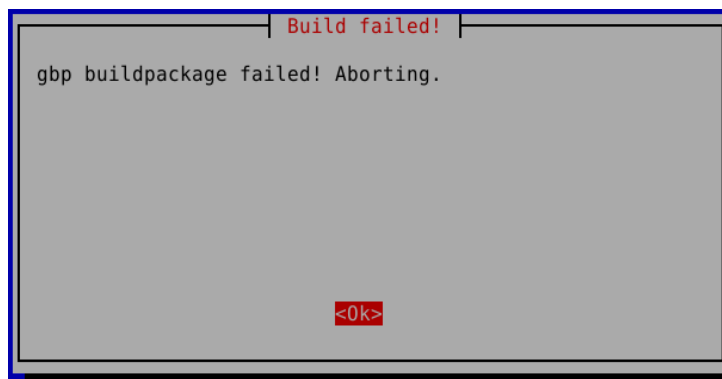


Abbildung 37.25.: Erfolgreicher Bauversuch!

38. Wenn das Bauen fehlschlägt

Wenn das Bauen fehlschlägt oder die nachfolgenden Tests (Kapitel 40, Seite 357) nicht zufriedenstellend ausfallen, kann das verschiedene Ursachen haben.

Erster Ansatzpunkt für die Ursachenforschung ist das Studium der Datei
< *SourceName* >_< *Version* >-< *Revision* >_< *Arch* >.build.

Eine Ursache, warum das Bauen fehlgeschlagen ist, kann ein unzureichendes Ermitteln der Build-Abhängigkeiten sein. Die Ermittlung, ob benötigte Abhängigkeiten bereits paketierte sind, ist in Kapitel 10.3 (Seite 30) beschrieben.

39. Bauen jenseits von *Unstable (sid)*

39.1. Bauen für bereits offiziell freigegebene Distributionen

Zum Bauen für *Backports*- und *Proposed-Updates*-Paketen (Kapitel 22, Seite 83) hat sich für uns die folgende Vorgehensweise bewährt. Die folgenden Ausführungen gelten für *Oldstable* und *Oldoldstable* entsprechend.

Zunächst ist von der Aufgabenauswahl aus (Kapitel 33.5, Seite 191) ein neuer Git-Branch anzulegen (Kapitel 44.1, Seite 407). Dieser geht in der Regel vom Git-Zweig *debian/sid* bzw. *master* oder *main* aus.

Der Name dieses neuen Zweiges sollte das Ziel-Repository des Paketes angeben (z.B. *debian/bookworm-bpo*).

Gibt es diesen Git-Zweig schon, kann dieser genutzt werden. Hierzu kann man folgende Befehle verwenden.

```
355 <Merge2Stable 355>≡
    git branch -vv
    git checkout <OlderBranch>
    git merge debian/sid # or master or main
    # Solve merge conflicts esp. d/changelog
    nano debian/changelog
    git add debian/changelog
    git commit
    # This is the merge commit
```

Die Behebung des Merge-Konfliktes erfordert in der Regel zumindest die Bearbeitung der Datei *debian/changelog*. Es wird ein neuer Eintrag in *debian/changelog* erstellt. (Kapitel 37.1, Seite 321)

Für den Versionseintrag in *debian/changelog* ist zwingend die in Kapitel 22.7 (Seite 87) beschriebene Nomenklatur zu verwenden. Diese hängt davon ab, für welchen Zweig nun gebaut wird.

Diese lauten:

stable-proposed-updates Es soll die bereits im Stable-Zweig vorhandene Paketversion bei der nächsten Zwischenveröffentlichung ersetzt werden.

stable-updates Es soll die bereits im Stable-Zweig vorhandene Paketversion dringend ersetzt werden.

stable-backports Zusätzlich zur älteren Paketversion soll eine neuere Version verfügbar gemacht werden.

39.2. Proposed-Updates – Besonderheiten

Der Changelog **darf nicht** die Nummer des Fehlerberichtes enthalten, der gegen *release.debian.org* erstellt wurde. Er sollte aber die Nummer des Fehlerberichtes enthalten, der gegen dieses Debian-Paket erstellt wurde und weswegen diese Version gebaut wird.

Danach wird das Paket für *proposed-updates* gebaut. Es ist eine neue Revision zu bauen (Kapitel 35, Seite 253). Hierzu müssen gegebenenfalls in der Datei *debian/control* die Versionen für die Abhängigkeiten angepasst werden.

Nach dem Bauen für die Veröffentlichung (Kapitel 41, Seite 373) und **vor** dem tatsächlichen Hochladen mit *dput* (Kapitel 43, Seite 387) wird mit *debdiff* (Kapitel 40.6.1, Seite 365) noch die Differenz zwischen der bisherigen Version im angestrebten Zweig und der neuen Version erstellt (Kapitel 40.6.1, Seite 365).

Wenn das Release-Team dem Antrag zustimmt, kann das Paket gebaut und hochgeladen werden. (Kapitel 42.1, Seite 383 bzw. Kapitel 43.3, Seite 392). Das Paket landet dann in der entsprechenden *New-Queue*.

Der Fehlerbericht an *release.debian.org* wird geschlossen, sobald das Paket dem nächsten Pointrelease der stabilen Version hinzugefügt wurde. Dies ist dann das Ende dieses Prozesses.

40. Überprüfungen

Eine erste Qualitätskontrolle erfolgt bereits während des Bauvorganges im *pbuilder* mit *lintian*.

Die nachfolgende Qualitätskontrolle erfolgt erneut mit *lintian* im *pedantic*-Modus (Kapitel 40.3, Seite 360).

```
357a <RunningTests 357a>≡ (364b)
function RunningTests {
    # Called by TaskSelect BuildNewRevision

    # QA using lintian and uscan

    # lintian
    RunningLintian
<RunningTests1 357b>
```

und mit *uscan* hinsichtlich des Inhaltes der Datei *debian/watch* (Kapitel 40.4, Seite 362).

```
357b <RunningTests1 357b>≡ (357a)
    # uscan
    if [ $linq -eq 0 ]
    then
        RunningUscan
    else
        usq=1
    fi
```

```
<RunningTests3 357c>
```

Wird mindestens eines der beiden Testergebnisse als mangelhaft qualifiziert, beendet sich das Programmskript nach einem entsprechenden Eintrag in die Log-Datei.

Andernfalls erfolgt die Frage, ob das Hochladen des Paketes vorbereitet werden soll.

```
357c <RunningTests3 357c>≡ (357b)
    if [ $usq -ne 0 ] || [ $linq -ne 0 ]
    then
        echo "At least one test failed!" >> ${log}
        exit
    else
        if whiptail --title "Upload?" \
        --yesno "Should the package be prepared to be uploaded now?" \
        --yes-button "Yes" --no-button "Exit" 15 60
        then
<RunningTests4 358>
```

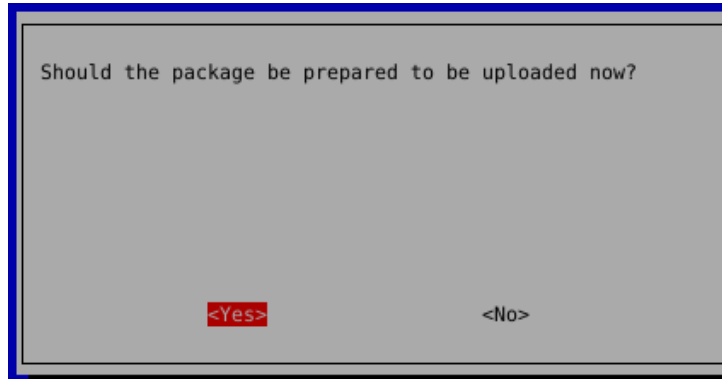


Abbildung 40.1.: Upload des Release vorbereiten

Wird die Frage bejaht, wird das Hochladen vorbereitet (Kapitel 41, Seite 373).

```
358 <RunningTests4 358>≡ (357c)
    Task=6 # Go to PrepareUploading
    else
    exit
    fi
fi
}
<CheckRepackSuffix 250a>
```

Später folgen noch weitere Überprüfungen mit *debdiff* (Kapitel 40.6.1, Seite 365).und *diffoscope*. Schließlich wird auch noch *piuparts* beschrieben.

40.1. Auswahl der Changes-Datei

Diese Funktion dient der Auswahl der Changes-Datei (**.changes*), die zur Prüfung des Paket-Bauergebnisses und zur Bestimmung der hochzuladenen Dateien genutzt wird.

```
359 <SelectChangesFile 359>≡ (277)
function SelectChangesFile {
    # Called by RunningLintian SelectUploadTarget

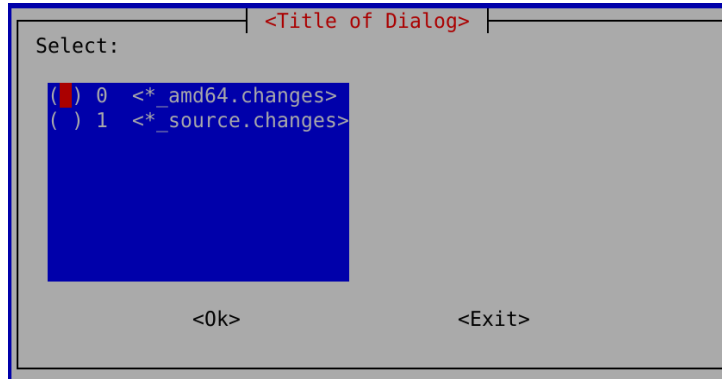
    titlestr=${1}
    cd ${PrjPath}
    changesa=$(ls ${SourceName}_${Version1}*_*.changes)

    if [ -z "${changesa}" ]
    then
        echo "File *"${SourceName}"*_ "${Version1}"*_*.changes not found!"
        exit
    fi

    i=0; slct=''
    for element in ${changesa[*]}
    do
        slct=${slct} '$i' '${element}' off '
        i=$(expr $i + 1)
    done

    paket=$(whiptail --title "${titlestr}" --radiolist "Select:" \
        --cancel-button "Exit" 15 60 8 $slct 3>&2 2>&1 1>&3)
```

<SelectChangesFile1 360a>

Abbildung 40.2.: **.changes*-Datei auswählen

Erfolgt keine Auswahl beendet sich das Programmskript.

360a `<SelectChangesFile1 360a>≡` (359)

```

if [ -z "${paket}" ]
then
    exit
fi
}

```

`<PatchHeader 317>`

40.2. Yamllint

Mit dem Kommandozeilentool *yamllint* kann die Syntax der *.yaml*-Dateien geprüft werden.

40.3. Prüfung mit Lintian

Das Ergebnis des Bauens wird mit *lintian* überprüft. Hierzu ist zunächst die **.changes*-Datei auszuwählen, auf die sich die Prüfung beziehen soll. Es wird daher die Funktion *SelectChangesFile* aufgerufen (Kapitel 40.1, Seite 359).

360b `<RunningLintian 360b>≡` (352)

```

function RunningLintian {
    # Called by RunningTests

    SelectChangesFile "lintian_check" # String will be found in ${1}
    linfile=${changesa[$paket]}
}
<RunningLintian1 361a>

```

Lintian wird mit Optionen aufgerufen, die eine ausführliche Prüfung bewirken.

361a \langle *RunningLintian1* 361a \rangle ≡ (360b)

```
set +e
lininfo=$(lintian --check --display-experimental --display-info \
--info --verbose --show-overrides --pedantic --tag-display-limit 0 \
--color auto ${linfile})
lx=$?
```

\langle *RunningLintian3* 361b \rangle

Wenn *Lintian* nichts meldet, soll der Nutzer dieses erfreuliche Ergebnis erfahren.

361b \langle *RunningLintian3* 361b \rangle ≡ (361a)

```
if [ ${lx} -eq 0 ]
then
    lininfo="Lintian does not find any Errors \
\n\n Congratulations \n\n"${lininfo}
fi
```

\langle *RunningLintian4* 361c \rangle

Die Variable *lininfo* muss noch mit *sed* bearbeitet werden, damit die *Lintian*-Meldungen in einzelnen Zeilen erscheinen.

361c \langle *RunningLintian4* 361c \rangle ≡ (361b)

```
# Make lininfo better readable
lininfo=$(echo ${lininfo} | sed --expression='s/ [EWIPNX]:/\n&/g')
```

```
set -e
echo -e "lintian("${lx}"): "${lininfo}
whiptail --title "Lintian" --msgbox "${lininfo}" --scrolltext 15 60
echo -e "Result of Lintian:\n"${lininfo} >> ${log}
```

\langle *RunningLintian5* 361d \rangle

Nach der Anzeige des Ergebnisses der Prüfung ist das Ergebnis vom Nutzer zu bewerten.

361d \langle *RunningLintian5* 361d \rangle ≡ (361c)

```
whiptail --title "All well?" \
--yesno "All well? Continue?" --yes-button "Yes" \
--no-button "Exit" 15 60
linq=$?
}
```

\langle *RunningUscan* 362 \rangle

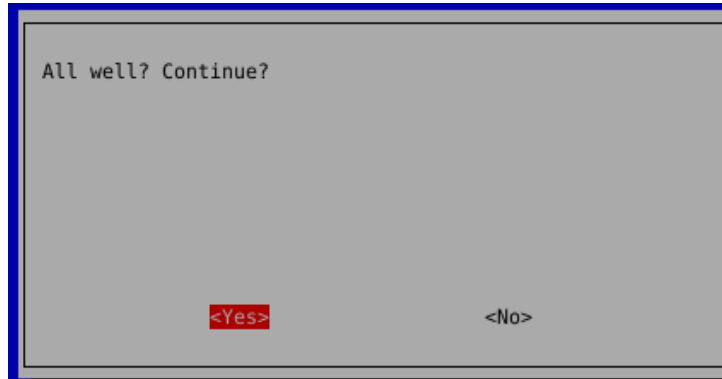


Abbildung 40.3.: Lintian: All Well?

Ist das Ergebnis in Ordnung, folgt die nächste Prüfung. Hinweise zur Fehlerbehebung sind im Kapitel 28 (Seite 103) zu finden.

40.4. Uscan

Mit der Option `-verbose` erstellt `uscan` einen für Menschen lesbaren Bericht über die Programmausführung. Mit der Option `-debug` wird zusätzlich der Status der internen Variablen angezeigt.

```

362 <RunningUscan 362>≡ (361d)
    function RunningUscan {
        # Called by RunningTests BuildWithUscan

        cd ${GitPath}

        # Check whether debian/watch exists
        if [ ! -e debian/watch ]
        then
            whiptail --title "No debian/watch file found" \
                --yesno "Is there a good reason for having no debian/watch file?" \
                --defaultno --yes-button "Yes" --no-button "No" 15 60
            if [ $? -eq 0 ]
            then
                usq=0
            else
                usq=1
            fi
            return
        fi

        set +e
        uscaninfo=$(uscan --no-download --verbose)
        if [ ${#uscaninfo} -gt 0 ]
        then
            whiptail --title "uscan" --msgbox "${uscaninfo}" --scrolltext 15 60
            echo -e "Result of uscan:\n"${uscaninfo} >> ${log}
        fi
    }

```


⟨RunningUscan1 363a⟩

Hier wird aus *uscaninfo* ausgelesen, ob die gebaute Version auch die aktuelle ist. Dies gilt für alle Builds, die in *experimental* oder *sid* veröffentlicht werden sollen. Die dazugehörige Meldung lautet: -> Package is up to date ...".

```
363a  ⟨RunningUscan1 363a⟩≡ (362)
      echo ${uscaninfo} | grep '=> Package is up to date' > /dev/null
      usc1=$?
      echo ${uscaninfo} | grep '=> Only older package available' > /dev/null
      usc2=$?
      if [ ${usc1} -eq 0 ]
      then
          whiptail --title "uscan" --msgbox "Package seems to be up to date." 15 60
          usq=0
      ⟨RunningUscan2 363b⟩
```

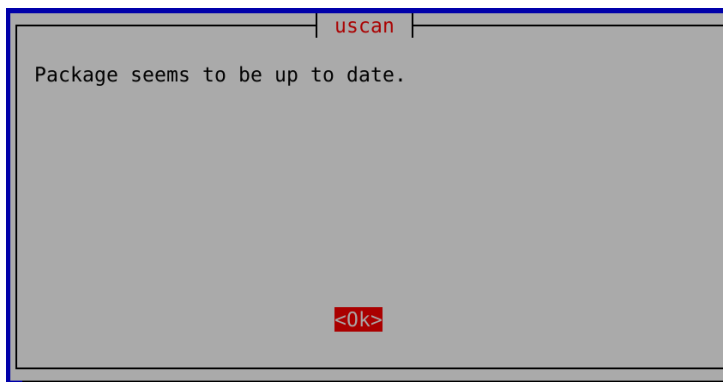


Abbildung 40.4.: Paket ist aktuell

```
363b  ⟨RunningUscan2 363b⟩≡ (363a)
      elif [ ${usc2} -eq 0 ]
      then
          whiptail --title "uscan" --msgbox "Only older package available." 15 60
          usq=0
      ⟨RunningUscan3 364a⟩
```

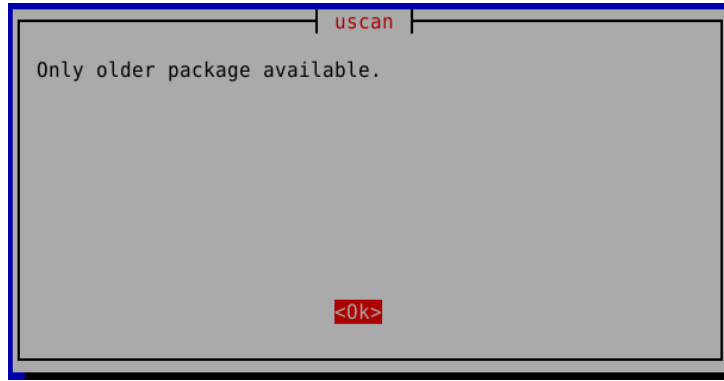


Abbildung 40.5.: Älteres Paket verfügbar

```
364a  <RunningUscan3 364a>≡ (363b)
      else
        whiptail --title "No up to date message" \
          --yesno "No up to date message.\nRegardless all well? Continue?" \
          --defaultno --yes-button "Yes" --no-button "Exit" 15 60
        usq=$?
      fi
<RunningUscan4 364b>
```

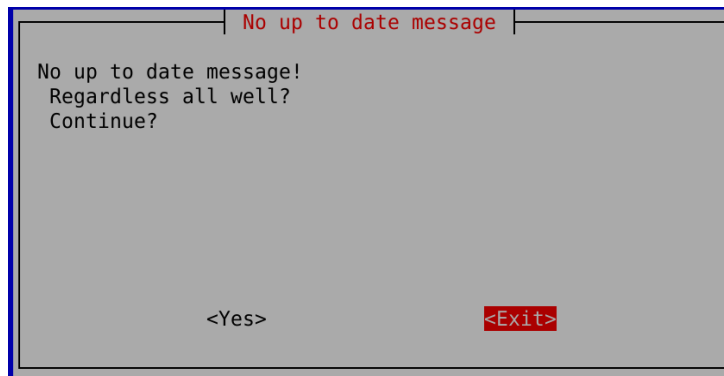


Abbildung 40.6.: Uscan - OK?

```
364b  <RunningUscan4 364b>≡ (364a)
      else
        echo "uscan failed" >> ${log}
        whiptail --title "uscan failed" --msgbox "uscan failed" 15 60
        usq=1
      fi
      set -e
    }

<RunningTests 357a>
```

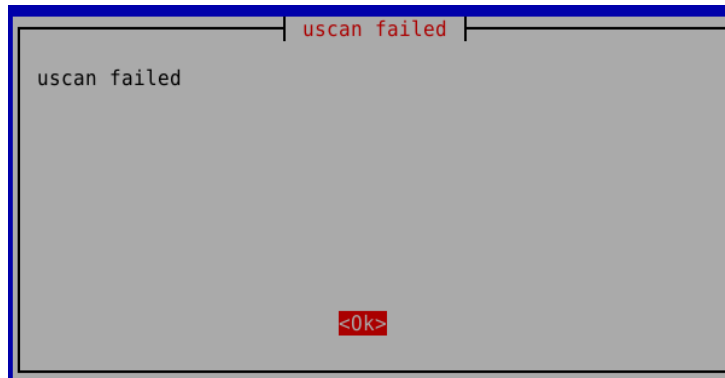


Abbildung 40.7.: Uscan schlägt fehl

40.5. Überprüfen der Datei *debian/copyright*

An dieser Stelle wird überprüft, ob auch wirklich **alle** Angaben zu den Lizenzen in der Datei *debian/copyright* erfolgt sind. Dazu gibt es mehrere Werkzeuge. Diese werden in Kapitel 10.1 (Seite 27) beschrieben.

Derzeit muss diese Prüfung noch manuell erfolgen, denn der Gebrauch dieser Werkzeuge ist manchmal nicht ausreichend.

40.6. Überprüfen mit *debdiff* und *diffoscope*

Beide Programme dienen dazu, die Differenz zwischen dem aktuellen Paket und der Vorversion darzustellen.

debdiff wird mit dem Paket *devscripts* installiert. *diffoscope* dagegen muss mit dem gleichnamigen Paket zusätzlich installiert werden.

40.6.1. *debdiff*

Mit *debdiff* können zwei Debian-Quellcodepaketen verglichen werden.

debdiff dient in diesem Falle dazu zu belegen, dass zwischen zwei Quellpaketen keine oder nur geringe Differenzen vorhanden sind.

```
365 <DebDiff 365>≡ (366b)
function DebDiff {
    # Called by TaskSelect Ask4DebDiff
    debdiffFlag=$1

    if [ ${debdiffFlag} -gt 0 ]
    then
        whiptail --title "debdiff" \
            --msgbox "Now you can detect the differences between two Debian packages.\n" \
            15 60
    fi
}
<DebDiff1 366a>
```

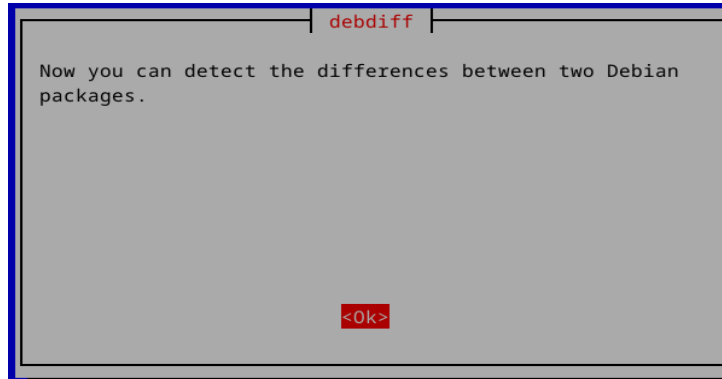


Abbildung 40.8.: Unterschiede feststellen

Der Nutzer kann zwei von ihm ausgewählte Quellcodepakete miteinander vergleichen, Hierzu werden *debdiff* die beiden Pakete in Form von *.dsc*-Dateien übergeben. Anhand dieser Dateien wird der Inhalt der Quellcodepakete verglichen.

```
366a <DebDiff1 366a>≡ (365)
      cd ${PrjPath}
      set +e
      PackageList=$(ls ${PrjPath} | grep \.dsc$ | sort --reverse --version-sort)
      set -e
      PackageArray=(${PackageList})

      i=0
      for element in ${PackageArray[*]}
      do
          packageE=${packageE}' '$i' '${element}' off '
          i=$(expr $i + 1)
      done
```

```
<DebDiff4 367a>
```

Für diese Auswahlmöglichkeit werden alle lokal verfügbaren *.dsc-Dateien* geordnet angezeigt. Davon können dann genau die zwei Dateien zum Vergleich ausgewählt werden.

```
366b <DebDiffCheckList 366b>≡ (142a)
      function DebDiffCheckList {
          # Called by DebDiff

          PackageNrL=$(whiptail --title "debdiff" \
            --checklist "Which two versions should be compared?" \
            15 60 8 \
            ${packageE} --cancel-button "Cancel" 3>&2 2>&1 1>&3)

          PackageNrA=(${PackageNrL})
      }
```

```
<DebDiff 365>
```

Das Programmskript prüft nun, ob genau zwei Einträge (*.dsc*-Dateien) ausgewählt wurden.

```
367a <DebDiff4 367a>≡ (366a)
      DebDiffCheckList

      if [ ${#PackageNrA[@]} -gt 2 ]
      then
        whiptail --title "Too much selections" \
        --msgbox "Please select only two versions" 15 60
        DebDiffCheckList

<DebDiff5 367b>
```

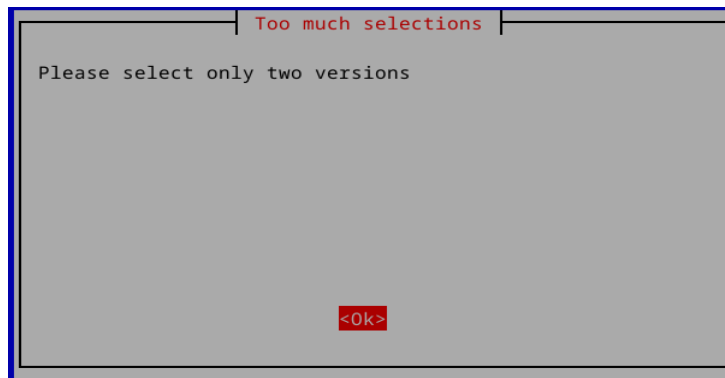


Abbildung 40.9.: Zu viele Versionen ausgewählt

```
367b <DebDiff5 367b>≡ (367a)
      elif [ ${#PackageNrA[@]} -lt 2 ]
      then
        whiptail --title "Less selections" \
        --msgbox "Please select two versions" 15 60
        DebDiffCheckList
      fi

<DebDiff6 368>
```

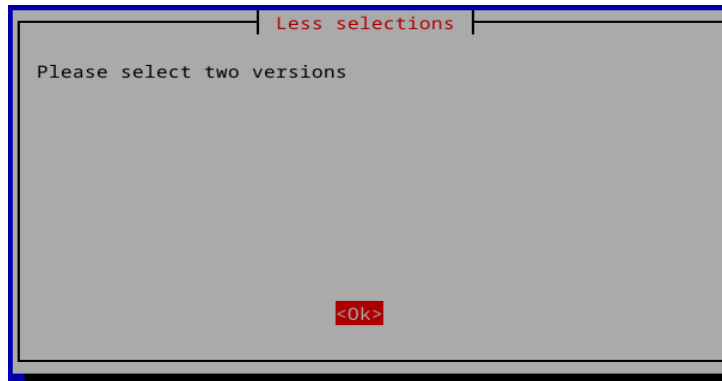


Abbildung 40.10.: Zu wenig ausgewählt

Wurden genau zwei *.dsc*-Dateien ausgewählt, erfolgt der Vergleich der Quellcodepakete mit *debdiff*.

```

368 <DebDiff6 368>≡ (367b)
    if [ -z "${PackageNrL}" ]
    then
        cd ${GitPath}
        CommonTasks
    fi

    if whiptail --title "Reverse order?" \
        --yesno "Should the order of the packages be reversed?" \
        --yes-button "Yes" --no-button "No" --defaultno 15 60
    then
        sn=${PackageNrA[1]}
        fn=${PackageNrA[0]}
    else
        fn=${PackageNrA[1]}
        sn=${PackageNrA[0]}
    fi

    <DebDiff7 369a>

```

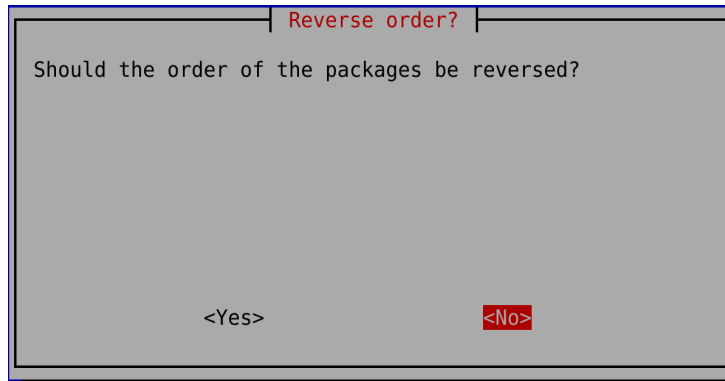


Abbildung 40.11.: Umkehr der Reihenfolge der zu vergleichenden Pakete

In bestimmten Fällen ist es sinnvoll, die Vergleichsreihenfolge umzukehren.

Regelmäßig ist das Resultat dann richtig, wenn im Dateinamen die ältere Version vor der neueren erscheint.

369a `<DebDiff7 369a>≡` (368)

```
# Killing toxic quotationmarks
fn=$(echo $fn | sed --expression="s/\\/g")
sn=$(echo $sn | sed --expression="s/\\/g")
```

`<DebDiff8 369b>`

Die Ausgabe von *debdiff* erfolgt in eine entsprechend benannten Datei im Projektverzeichnis. Der Inhalt dieser Datei wird mit *less* angezeigt.

369b `<DebDiff8 369b>≡` (369a)

```
echo -e "Compare with debdiff: \n" \
  "${PackageArray[fn]}" and "${PackageArray[sn]}"\n" >> \
  ${log}
echo -e "The result can be found in\n" \
  debdiff_${PackageArray[fn]}-${PackageArray[sn]}.diff >> ${log}
set +e
debdiff --diffstat ${PackageArray[fn]} ${PackageArray[sn]} > \
  debdiff_${PackageArray[fn]}-${PackageArray[sn]}.diff
set -e
less debdiff_${PackageArray[fn]}-${PackageArray[sn]}.diff

if [ ${debdiffFlag} -eq 0 ]
then
  cd ${GitPath}
  CommonTasks
fi
cd ${GitPath}
}
```

`<ImportDebianPackage 168a>`

Teil IV.

Veröffentlichen

41. Vorbereitungen zum Hochladen

Bisher haben wir uns damit beschäftigt, auf unserem Rechner ein **Debian**-Paket zu bauen, wie es auch vom Projekt **Debian** bereitgestellt wird.

Nun geht es darum, dass dieses Paket auch hochgeladen und damit dem **Debian**-Projekt zur Verfügung gestellt werden kann.

Die Vorbereitung erfolgt durch die Funktion *PrepareUploading*.

41.1. Existiert *debian/changelog*?

Es wird sicherheitshalber überprüft, ob eine Datei *debian/changelog* bereits existiert.

373a

```
<PrepareUploading 373a>≡
function PrepareUploading {
    # Called by TaskSelect

    cd ${GitPath}

    # Check debian/changelog
    if [ -f debian/changelog ]
    then
        less --LINE-NUMBERS debian/changelog
    else
<PrepareUploading1 373b>
```

Ist diese Datei nicht vorhanden, bricht das Programmskript an dieser Stelle mit einem entsprechenden Hinweis ab.

373b

```
<PrepareUploading1 373b>≡ (373a)
    whiptail --title "This is the end" \
    --msgbox "No changelog - no upload!" 15 60
    exit
fi

<PrepareUploading2 374>
```

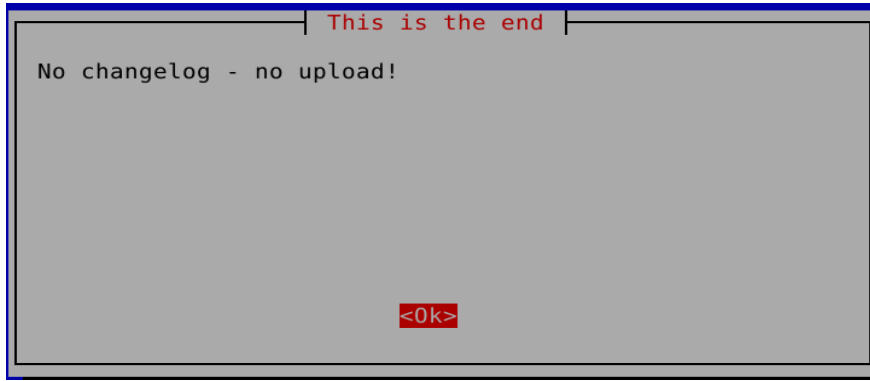


Abbildung 41.1.: Kein Changelog - kein Hochladen

Andernfalls wird der Changelog für eine Veröffentlichung vorbereitet oder verwandt. Hierzu wird abgefragt, ob der Changelog schon veröffentlichungsreif ist.

Dazu muss auf jeden Fall das „UNRELEASED“ durch die jeweilige Distributionsbezeichnung – meist „unstable“ – ersetzt werden.

```
374 <PrepareUploading2 374>≡ (373b)
    if ! whiptail --title "Changelog fit for publishing?"--defaultno \
        --yesno "Is the changelog fit for publishing?" --yes-button "Yes" \
        --no-button "No" 15 60
    <PrepareUploading3 375>
```

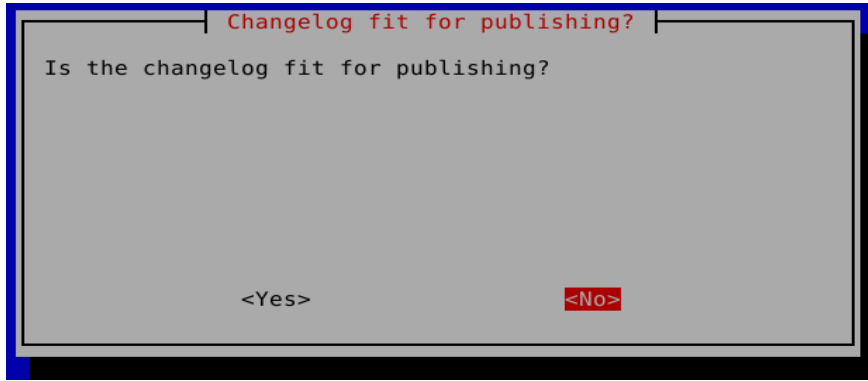


Abbildung 41.2.: Changelog veröffentlichungsreif?

Wird die Frage bejaht, wird die Option eröffnet, das Paket nochmal zu bauen (Kapitel 41.3, Seite 379).

41.2. *debian/changelog* fertigstellen

Ist die Datei *debian/changelog* nicht veröffentlichungsreif, wird sie verbessert.

Hierzu wird zunächst dazu aufgefordert zu prüfen, ob man im richtigen *Git*-Zweig ist. Mit der Funktion *AskDist* (Kapitel 37.3.1, Seite 332) wird ermittelt, welches der aktuelle *Git*-Zweig und welche Distribution ihm zugeordnet ist.

Danach wird das Ergebnis angezeigt. Gegebenenfalls kann in einem weiteren Terminal der Zweig gewechselt werden.

```

375 <PrepareUploading3 375>≡ (374)
    then
        AskDist
        echo -e "Notice from PrepareUploading: The branch is "${RecentBranch}"\n \
        The distribution is "${RecentBranchD} >> ${log}
        whiptail --title "Please check! (U)" \
        --msgbox "The branch is ${RecentBranch}" 15 60
    <PrepareUploading4 376>
  
```

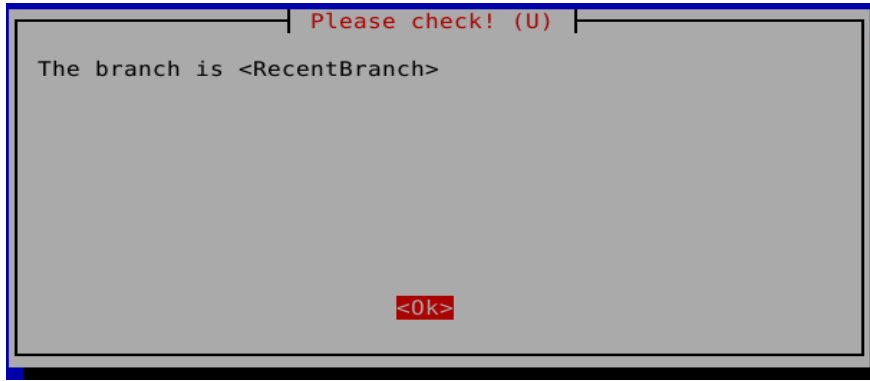


Abbildung 41.3.: Branch überprüfen

Sodann ist gegebenenfalls der Name der Distribution anzugeben, wohin das gebaute Debian-Paket hochgeladen werden soll. In der Regel ist dies die Distribution *unstable*.

376 `<PrepareUploading4 376>` ≡ (375)

```

if [ "${RecentBranchD}" = "sid" ]
then
    distName="unstable"
elif [ "${RecentBranchD}" = "experimental" ]
then
    distName="experimental"
else
    distName=$(whiptail --title "Name of the distribution" \
        --inputbox "Please insert the name of the distribution\n \
        specified in the changelog" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
fi

```

`<PrepareUploading5 377>`

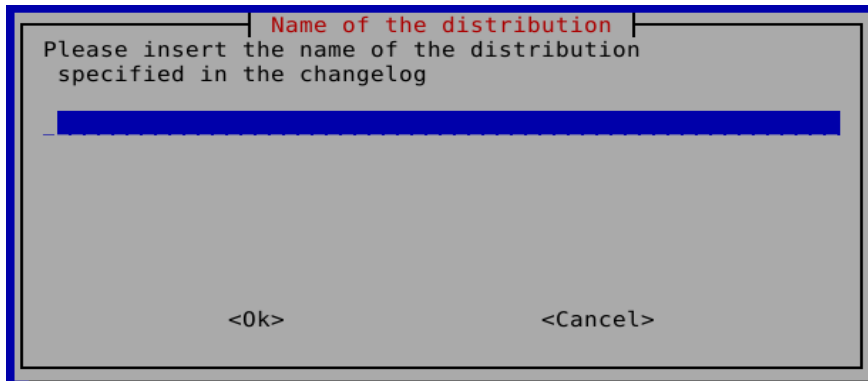


Abbildung 41.4.: Name der Distribution eingeben

Wird kein Name eingegeben, wird *unstable* als Distribution angenommen.

Dann wird nochmal der Name der Distribution mit der Aufforderung zur Prüfung angezeigt.

```
377 <PrepareUploading5 377>≡ (376)
    if [ -z "${distName}" ]
    then
        distName="unstable"
    fi

    echo -e "Another notice from PrepareUploading:\n \
The distribution is now "${distName}" >> ${log}
whiptail --title "Please check! (U)" \
--msgbox "The distribution is "${distName}" 15 60
<PrepareUploading6 378a>
```

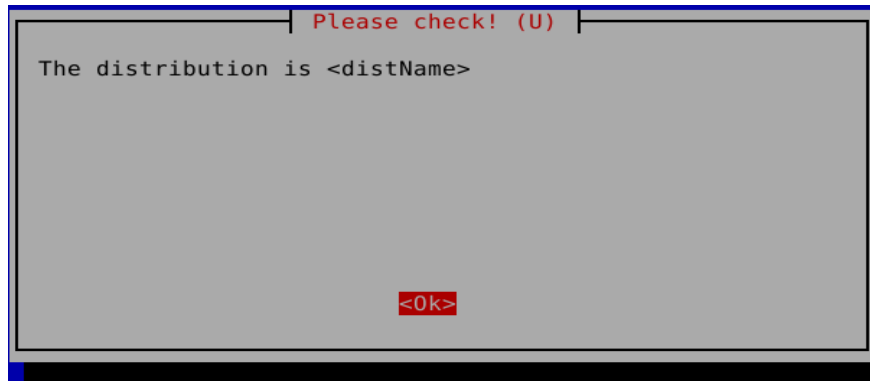


Abbildung 41.5.: Name der Distribution überprüfen

Gerade bei neuen Paketen verwenden die Autoren gerne einen Upload nach *experimental*. Für neue Pakete ist nämlich ein *Binary*-Upload notwendig.

378a `<PrepareUploading6 378a>≡` (377)

```
# making debian/changelog fit for publishing
gbp dch --release --verbose --debian-branch=${RecentBranch} \
--distribution=${distName} #--commit
```

`<PrepareUploading8 378b>`

Nach der Erstellung des Changelogs für das Release wird zur Kontrolle automatisch der Standardeditor geöffnet. Dies kann nicht vom Skript beeinflusst werden.

Diese Anzeige ist auch sinnvoll. Oft sind nämlich doppelte Commit-Einträge zu löschen.

378b `<PrepareUploading8 378b>≡` (378a)

```
git add .
git commit -a
```

```
whiptail --title "Build again" \
--msgbox "Now the release will be built another time." 15 60
```

`<PrepareUploading10 379a>`

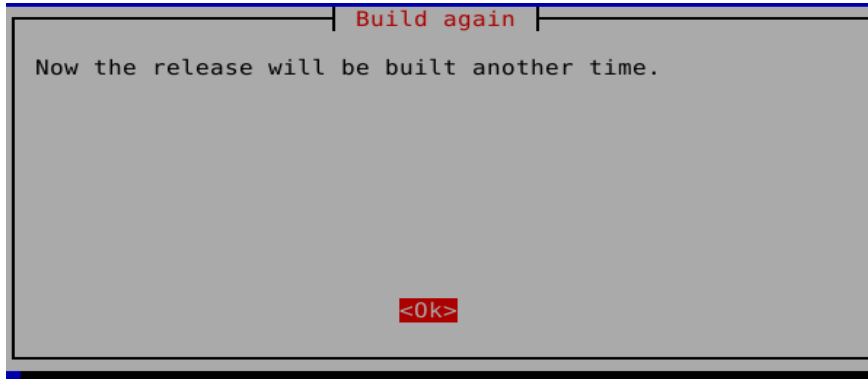


Abbildung 41.6.: Fürs Release bauen

Nun wird das Paket fürs Hochladen ins Debian-Repository gebaut.

Das nochmalige Bauen ist notwendig, da die Datei *debian/changelog* geändert wurde und daher erneut integriert werden muss.

```
379a <PrepareUploading10 379a>≡ (378b)
      # Building revision
      GpgKeyAvailable
      SBuildOrPBuilder 1
      if [ $gbpq -eq 0 ]
      then
          echo "Package ${SourceName} was built using gbp buildpackage." >> ${log}
      fi
      <PrepareUploading11 379b>
```

41.3. Nochmaliges Bauen?

Hier geht es weiter, wenn die Frage bejaht wird, ob die Datei *debian/changelog* veröffentlichungsreif ist. Dann wird die Option eröffnet, das Paket nochmals zu bauen.

Das Bauen erfolgt dann in der soeben beschriebenen Weise.

```
379b <PrepareUploading11 379b>≡ (379a)
      else
          if whiptail --title "Building another time?" \
            --yesno "Should the release be build another time?\n(Without tagging)" \
            --yes-button "Yes" --no-button "No" 15 60
          <PrepareUploading12 380a>
```

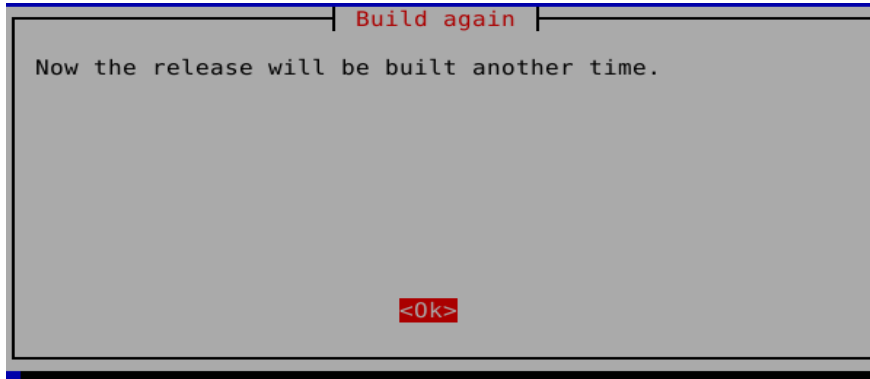


Abbildung 41.7.: Fürs Release bauen

```
380a  <PrepareUploading12 380a>≡ (379b)
      then
        # Building revision
        SBuildOrPBuilder 2
        if [ $gbpq -eq 0 ]
        then
          echo "Package ${SourceName} was built using gbp buildpackage." >> ${log}
        fi
      fi
    fi
  }

  <SelectUploadTarget 388a>
```

41.4. Soll ein Debdiff erstellt werden?

Bevor nach *salsa.debian.org* hochgeladen wird, wird gegebenenfalls abgefragt, ob ein *debdiff* (Kapitel 40.6.1, Seite 365). erstellt werden soll.

```
380b  <TaskSelect9 380b>≡
      if [ ${rcts} -eq 0 ]
      then
        Ask4DebDiff
      <TaskSelect9-1 383a>
```

Die Abfrage erfolgt nur, wenn jenseits vom Zweig *Unstable* paketiert werden soll (Kapitel 22, Seite 83).

```

381 <Ask4DebDiff 381>≡
    function Ask4DebDiff {
        # Called by TaskSelect

        # Check whether branch is sid
        if [ ${RecentBranch} == "debian/sid" ] || [ ${RecentBranch} == "master" ] || \
            [ ${RecentBranch} == "main" ]
        then
            return
        fi

        # Creating a Debdiff?
        if whiptail --title "DebDiff needed?" \
            --yesno "Do you want to create a debdiff?" \
            --yes-button "Yes" --no-button "No" 15 60
        then
            DebDiff 1
        fi
    }

    <TaskSelect (nicht definiert)>

```

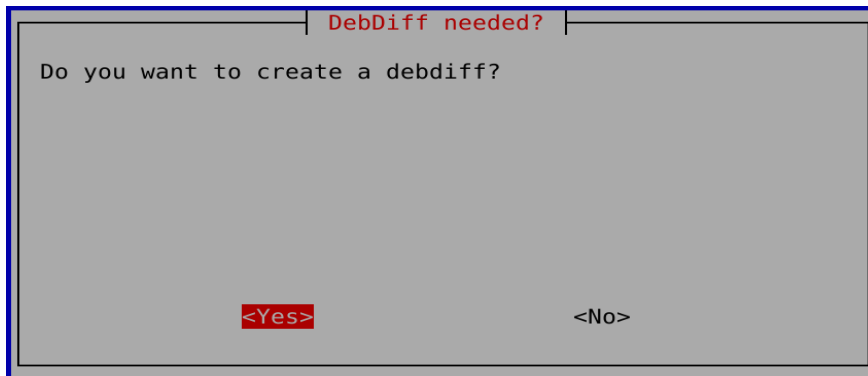


Abbildung 41.8.: DebDiff benötigt?

42. Hochladen auf Git-Repositories

Im folgenden Abschnitt aus der Aufgabenauswahl werden zunächst die Funktionen aufgerufen, um in die Git-Repositories hochzuladen.

```
383a <TaskSelect9-1 383a>≡ (380b)
      #####

      # Pushing git repo

      #####

      Upload2OwnServer
      Upload2Salsa

<TaskSelect10 387a>
```

42.1. Hochladen nach salsa.debian.org

Vor dem Hochladen nach *salsa.debian.org* überprüft das Programmskript, ob dort überhaupt schon ein entsprechendes Repository vorhanden ist.

```
383b <Upload2Salsa 383b>≡ (386)
      function Upload2Salsa {
          # Called by TaskSelect and itself

          # Uploading to Salsa

          if whiptail --title "Upload to salsa.debian.org?" \
              --yesno "Should ${SourceName} be uploaded to Salsa?" \
              --yes-button "Yes" --no-button "No" 15 60
          then
<Upload2Salsa1 384>
```

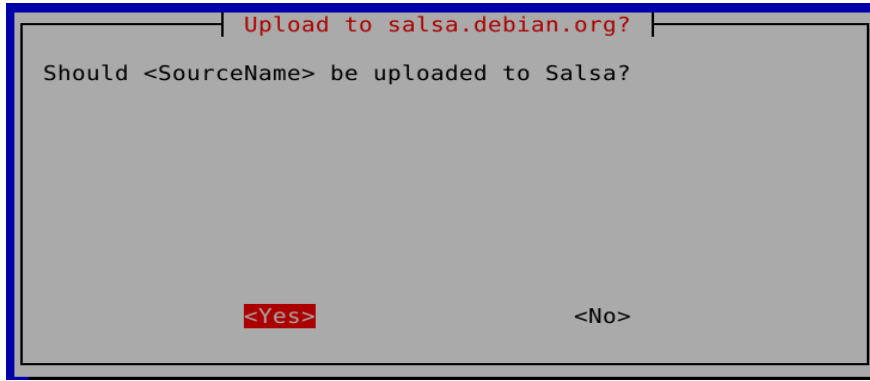


Abbildung 42.1.: Fürs Release bauen

```
384  <Upload2Salsa1 384>≡ (383b)
      BrowserName=$(echo ${SalsaName} | sed --expression='s/.git$//g')
      BrowserName="https://salsa.debian.org/${BrowserName}"
      wget --spider --verbose --max-redirect=0 \
      --append-output=${log} ${BrowserName}
      if [ $? -ne 0 ]
      then
          whiptail --title "No project found at salsa.debian.org" \
          --msgbox "Please create ${BrowserName} first" 15 60
          echo "No project "${BrowserName}" found at salsa.debian.org" \
          >> ${log}

          if whiptail --title "Done?" \
          --yesno "Created ${BrowserName} on salsa.debian.org?" \
          --yes-button "Yes" --no-button "No" 15 60
          then
              Upload2Salsa
          else
              exit
          fi
      else
          <Upload2Salsa5 385>
```

Wenn Patch-Queue-Zweige existieren, können diese vor dem Hochladen nach *salsa.debian.org* gelöscht werden.

```

385 <Upload2Salsa5 385>≡ (384)
    set +e
    if echo $(git branch) | grep --quiet 'patch-queue/'
    then
        if whiptail --title "Patch queue branches found:" \
        --yesno "$(git branch | grep 'patch-queue')\n \
        Delete all patch-queue branches?" \
        --yes-button "Yes" --no-button "No" 15 60
        then
            git branch --delete --force \
            $(git branch | grep 'patch-queue')
        fi
    fi
    set -e

    if ! whiptail --title "Last stop before upload!" \
    --yesno "Anything all right?" \
    --yes-button "Yes" --no-button "No" 15 60
    then
        exit
    fi
    set +e
    if git remote | grep 'salsa' > /dev/null
    then
        RepoName="salsa"
    else
        RepoA=$(git remote)

        i=0; slct=''
        for element in ${RepoA[*]}
        do
            slct=${slct}' '$i' '${element}' off '
            i=$(expr $i + 1)
        done

        RepoNr=$(whiptail --title "Select repository" \
        --radiolist "Select one of these repositories" \
        --cancel-button "Exit" 15 60 8 \
        $slct 3>&2 2>&1 1>&3)

        if [ -z "${RepoNr}" ]
        then
            exit
        else
            RepoName=${RepoA[${RepoNr}]}
        fi
    fi

    git push --set-upstream ${RepoName} --all >> ${log}

```

```

        git push --set-upstream ${RepoName} --tags >> ${log}
        set -e

        echo "${SourceName}" was uploaded to salsa.debian.org." >> ${log}
    fi
fi
}

<GettingFingerprint (nicht definiert)>

```

42.2. Hochladen auf eigenen Git-Server

Das Hochladen auf einen eigenen Git-Server setzt voraus, dass ein solcher eingerichtet wurde (Kapitel 20.4.2, Seite 78). Ferner ist zuvor sein Name oder seine IP-Adresse einzugeben (Kapitel 44.2, Seite 408).

```

386 <Upload2OwnServer 386>≡ (390b)
    function Upload2OwnServer {
        # Called by TaskSelect

        # Uploading to own git server
        if [ -n "$ServerName" ]
        then
            if whiptail --title "Upload to own git server?" \
                --yesno "Should ${SourceName} be uploaded to your own git server?" \
                --yes-button "Yes" --no-button "No" 15 60
            then
                git push --set-upstream home --all >> ${log}
                git push --set-upstream home --tags >> ${log}

                echo "${SourceName}" was uploaded to your git server." >> ${log}
            fi
        fi
    }

    <Upload2Salsa 383b>

```


43. Paket(e) hochladen

In der Funktion *TaskSelect* (Aufgabenauswahl) werden auch die Funktionen zum Hochladen der Pakete aufgerufen.

```
387a <TaskSelect10 387a>≡ (383a)
      #####

      # Uploading packages

      #####

      SelectUploadTarget
      CreateSignature
      UploadUsingDput
      Upload2PeopleD0
      UploadLocal
<TaskSelect11 387b>
```

Wenn die Funktionen *CreateNewBranch* (Kapitel 44.1, Seite 407), *SelectBranch* (Kapitel 33.4, Seite 182) oder *OwnServer* (Kapitel 44.2, Seite 408) aufgerufen wurden, wird die Konfigurationsdatei erneut zur Bearbeitung angezeigt (Kapitel 33.1, Seite 177 und anschließend die Aufgabenauswahl erneut aufgerufen.

```
387b <TaskSelect11 387b>≡ (387a)
      else
        ConfigFileLEC
        CommonTasks
      fi
    }

    <StartTasks (nicht definiert)>
```

43.1. Auswahl des Zielrepositoriums

Vor dem Hochladen ist das Zielrepositorium auszuwählen. Dabei wird ein *Non-Maintainer-Upload* wie ein Repository behandelt.

```
388a <SelectUploadTarget 388a>≡ (380a)
function SelectUploadTarget {
    # Called by TaskSelect Upload2FtpMaster

    # Select upload target
    Up1=$(whiptail --title "Uploading?" \
    --radiolist "Should the package be uploaded to ftp-master,\n \
    people.d.o or mentors.debian.net?" 15 60 6\
    "0" "No" off \
    "1" "ftp-master" on \
    "2" "people.d.o" off \
    "3" "Mentors" off \
    "4" "Non-Maintainer-Upload" off \
    "5" "Local repository" off --cancel-button "Exit" 3>&2 2>&1 1>&3)

    <SelectUploadTarget1 388b>
```

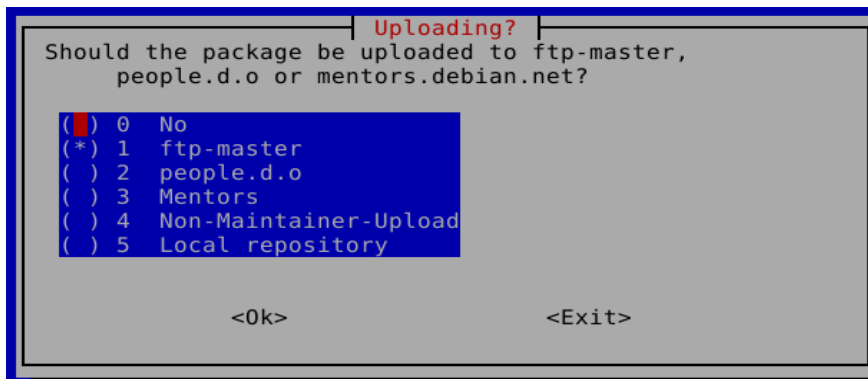


Abbildung 43.1.: Ziel des Uploads.

Wird „No“ oder „Cancel“ ausgewählt, wird das Programm beendet.

```
388b <SelectUploadTarget1 388b>≡ (388a)
    # The order of the conditions is important!
    # 'Cancel' results an empty variable
    if [ -z ${Up1} ] || [ ${Up1} -eq 0 ]
    then
        exit
    fi

    <SelectUploadTarget2 389a>
```

Über den Wert, der der Variablen *Upl* zugewiesen wird, wird der weitere Ablauf gesteuert.

```
389a <SelectUploadTarget2 389a>≡ (388b)
    case "${Upl}" in
        1) Upltext="ftp-master";;
        2) Upltext="people.d.o";;
        3) Upltext="Mentors";;
        4) Upltext="delayed";;
        5) Upltext="local repository";;
    esac
```

<SelectUploadTarget3 389b>

Im Folgenden wird nun mit der Funktion *SelectChangesFile* (Kapitel 40.1, Seite 359) die **.changes*-Datei des hochzuladenen Paketes ausgewählt.

```
389b <SelectUploadTarget3 389b>≡ (389a)

    cd ${PrjPath}

    # Select package
    SelectChangesFile "Upload" # String will be found in ${1}
    UplPaket=${changesa[$paket]}

    # Version2=$(echo ${UplPaket} | sed --expression="s/^[a-z\-\]*_//" | \
    # sed --expression="s/-.*/")
    # SourceName1=$(echo ${UplPaket} | sed --expression="s/_.*//1")
    # OrigPaket=${SourceName1}_${Version2}.orig"
```

<SelectUploadTarget4 389c>

Vor dem Hochladen erfolgt noch eine Kontrollfrage, ob das ausgewählte Paket zum ausgewählten Ziel hochgeladen werden soll. Dabei wird zum ersten Mal der Wert *Upltext* benötigt.

```
389c <SelectUploadTarget4 389c>≡ (389b)
    # Final question before uploading starts
    if [ ${Upl} -ne 4 ]
    then
        if ! whiptail --title "Upload to ${Upltext}?" \
            --yesno "Do you want to upload ${UplPaket} to ${Upltext}." \
            --yes-button "Yes" --no-button "Exit" 15 60
        <SelectUploadTarget5 390a>
```

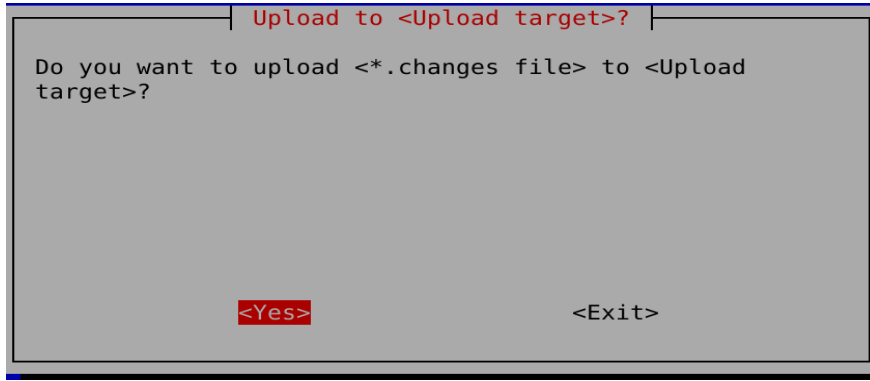


Abbildung 43.2.: Ist das angegebene Ziel des Uploads korrekt?

```
390a  <SelectUploadTarget5 390a>≡ (389c)
      then
        whiptail --title "Bye" --msgbox "Bye" 15 60
      <SelectUploadTarget6 390b>
```

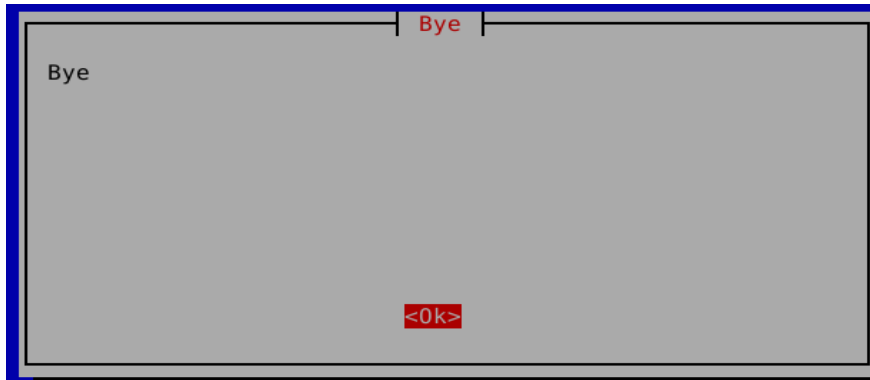


Abbildung 43.3.: Bye

```
390b  <SelectUploadTarget6 390b>≡ (390a)
      exit
    fi
  fi

  echo "${UplPaket} should be uploaded to ${Upltext}." >> ${log}
}

<Upload2OwnServer 386>
```

43.2. Signatur erzeugen

Vor dem Hochladen muss das Paket signiert werden.

Die Funktion *CreateSignature* erzeugt die erforderlichen Signaturen mittels *debsign*. *debsign* signiert ein Debian-*.changes*- und *-.dsc*-Dateipaar mittels GnuPG. Dazu muss der GnuPG-Schlüssel zur Verfügung stehen (Kapitel 32.8, Seite 172).

Wurde die Datei **.changes* bereits signiert, erscheint auf der Konsole folgende Meldung:

```
The .changes file is already signed.
Would you like to use the current signature? [Yn]
```

Im Falle des Fehlschlagens wird eine Wiederholung angeboten.

```
391 <CreateSignature 391>≡
function CreateSignature {
    # Called by TaskSelect Upload2FtpMaster and itself

    # Key available?
    GpgKeyAvailable

    # Signature using debsign
    set +e
    debsign -k${fipr} ${UplPaket}
    if [ $? -ne 0 ]
    then
        if whiptail --title "Signing failed!" \
            --yesno "Signature failed - Retry?" \
            --yes-button "Yes" --no-button "Exit" 15 60
    <CreateSignatur3 392a>
```

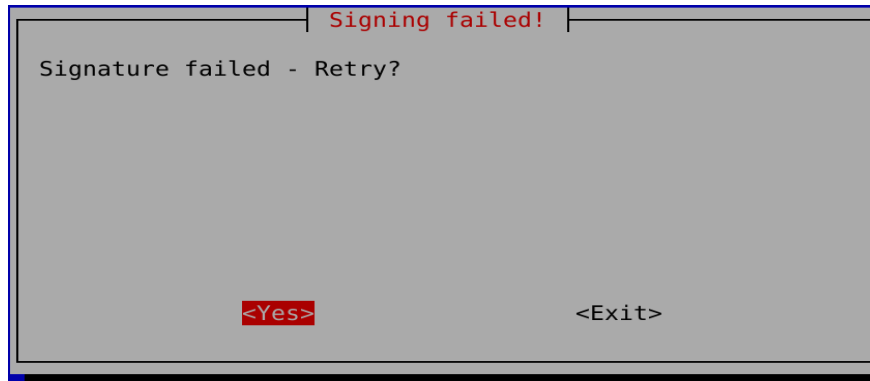


Abbildung 43.4.: Signieren erfolgreich?

```

392a  <CreateSignatur3 392a>≡ (391)
      then
        CreateSignature
      else
        exit
      fi
    fi
    set -e
    echo "${UplPaket} was signed" >> ${log}
  }

  <Upload2Mentors 398>

```

43.3. Hochladen mit dput

In der Funktion *UploadUsingDput* erfolgt nur die Weichenstellung, ob das Paket mittels *dput* zu den Ftp-Mastern (Kapitel 43.4, Seite 393) oder nach *mentors.debian.net* (Kapitel 43.5, Seite 398) hochgeladen werden soll.

```

392b  <UploadUsingDput 392b>≡ (395a)
      function UploadUsingDput {
        # Called by TaskSelect Upload2FtpMaster

        # Uploading using dput

        cd ${PrjPath}/
        if [ ${Upl} -eq 3 ]
        then
          Upload2Mentors
        elif [ ${Upl} -eq 1 ] || [ ${Upl} -eq 4 ]
        then
          Upload2FtpMaster
        fi
      }

  <UploadFilesSelect 400b>

```

43.4. Nach FTP-Master hochladen

Die Pakete werden regelmäßig nach *ftp-master* hochgeladen, um sie durch das Debian-Projekt zu veröffentlichen. Andere Veröffentlichungswege sind eher die Ausnahme.

```
393a <Upload2FtpMaster 393a>≡ (397)
function Upload2FtpMaster {
    # Called by UploadUsingDput

    # repeat question
    if whiptail --title "Last exit" \
        --yesno "Should the package be uploaded to ftp-master?" \
        --yes-button "Yes" --no-button "Exit" 15 60
    <Upload2FtpMaster1 393b>
```

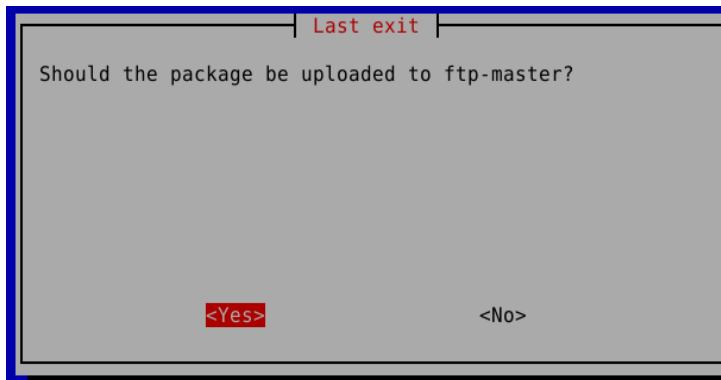


Abbildung 43.5.: Upload to FTP-Master - OK?

```
393b <Upload2FtpMaster1 393b>≡ (393a)
then
    set +e
    # Checking whether the .changes file is the right one for the upload target
    sourceFlag=$(echo ${UplPaket} | grep --count '_source.')
    expFlag=$(grep --line-number ' experimental; urgency=' ${GitPath}/debian/changelog \
        | grep '^1:')
    set -e
    echo -e "${UplPaket}:\n${expFlag}\nsourceFlag: ${sourceFlag}" >> ${log}
    # Strip line to isolate release
    expFlag=$(echo ${expFlag} | sed --expression='s/^.*' //' | \
        sed --expression='s/; .*$//')

    if [ -z "${expFlag}" ]
    then
        if [ ${sourceFlag} -eq 0 ]
        then
            if whiptail --title "Uploading?" \
                --yesno "Do you really want to upload a binary package\n \
                to ftp-master?" --yes-button "Yes" --no-button "No" 15 60

    <Upload2FtpMaster2 394>
```

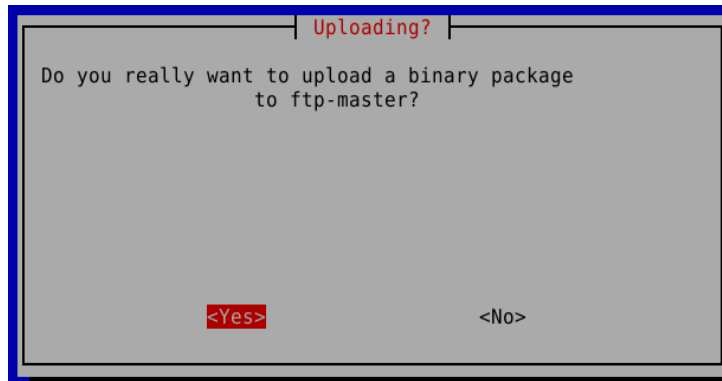


Abbildung 43.6.: Soll ein Binär-Paket auf FTP-Master hochgeladen werden?

```
394  <Upload2FtpMaster2 394>≡ (393b)
      then
        Dput2FtpMaster
      else
        echo "Next try to upload" >> ${log}
        SelectUploadTarget
      fi
    else
      Dput2FtpMaster
    fi
  else
    if [ $sourceFlag -ge 1 ]
    then
      if whiptail --title "Uploading?" \
        --yesno "Do you really want to upload a source package\n \
        to experimental?" --yes-button "Yes" --no-button "No" 15 60
    then
      <Upload2FtpMaster3 395a>
```

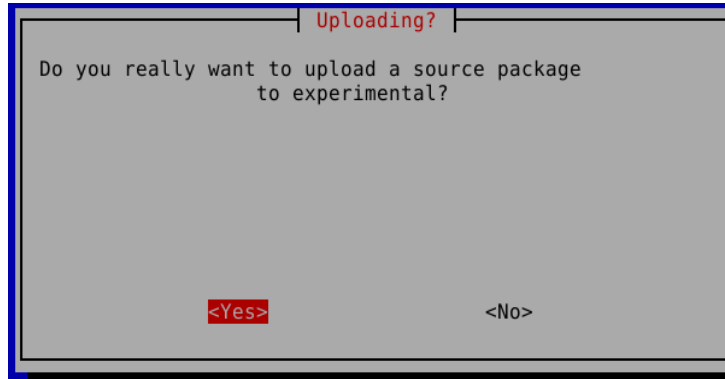



Abbildung 43.7.: Wirklich nach Experimental hochladen?

395a `<Upload2FtpMaster3 395a>`≡ (394)

```

    then
        Dput2FtpMaster
    else
        echo "Next try to upload" >> ${log}
        SelectUploadTarget
        CreateSignature
        UploadUsingDput
    fi
else
    Dput2FtpMaster
fi
fi
}

```

`<UploadUsingDput 392b>`

In der folgenden Funktion erfolgt das Hochladen nach FTP-Master.

395b `<Dput2FtpMaster 395b>`≡ (399b)

```

function Dput2FtpMaster {
    # Called by Upload2FtpMaster

    if whiptail --title "Simulate uploading?" \
        --yesno "Should the upload to ftp-master be simulated?" \
        --yes-button "Yes" --no-button "No" 15 60

```

`<Dput2FtpMaster1 396>`

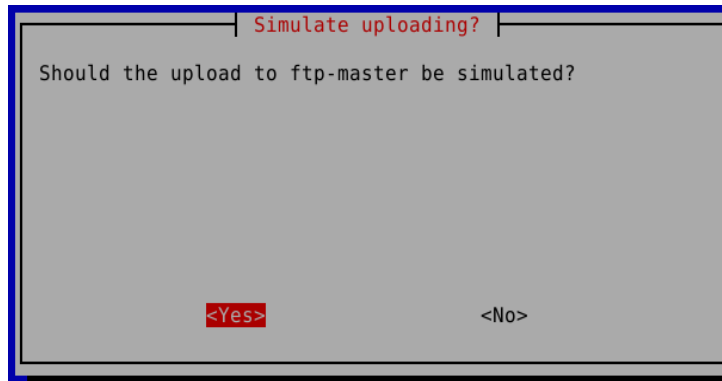


Abbildung 43.8.: Soll das Hochladen auf FTP-Master simuliert werden?

```
396  <Dput2FtpMaster1 396>≡ (395b)
      then
        dput --simulate ftp-master ${UplPaket}
        echo
        echo "After reading press return!"
        read x
      fi

      if whiptail --title "Uploading to FTP-Master?" \
        --yesno "Everything fine?\n\n \
        Should the package be uploaded to ftp-master now?" \
        --yes-button "Yes" --no-button "No" 15 60

      <Dput2FtpMaster2 397>
```

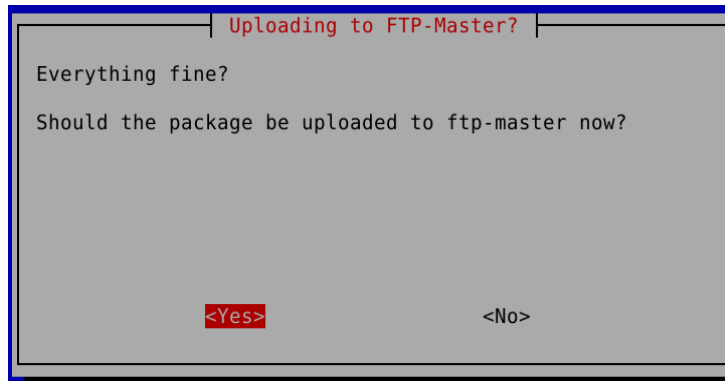


Abbildung 43.9.: Ist alles ok mit dem Hochladen?

```

397  <Dput2FtpMaster2 397>≡ (396)
      then
        if [ ${Upl} -eq 1 ]
        then
          dput ftp-master ${UplPaket}
          echo "${UplPaket} was uploaded to ${Upltext}." >> ${log}
        else
          Dput2NMU
        fi
      fi
}

<Upload2FtpMaster 393a>

```

Dieses Paket landet zunächst auf <https://incoming.debian.org/debian-build/pool/main/> bis es vom Build-Daemon (buildd) gebaut und zum Herunterladen zur Verfügung gestellt wird.

43.4.1. Zurückweisung eines Paketes

Wenn ein Paket von den FTP-Mastern zurückgewiesen wird, sollte beim anschließenden Hochladen des korrigierten Paketes die Revisionsnummer nicht erhöht werden.

Dazu muss die Datei `<PaketName>_<Version>_source.ftp-master.upload` vorher im Projektverzeichnis entfernt werden.

43.5. Nach mentors.debian.net hochladen

```
398 <Upload2Mentors 398>≡ (392a)
function Upload2Mentors {
    # Called by UploadUsingDput

    if whiptail --title "Simulate uploading?" \
        --yesno "Should the upload to Mentors be simulated?" \
        --yes-button "Yes" --no-button "No" 15 60
    then
        dput --simulate mentors ${UplPaket}
        echo
        echo "After reading press return!"
        read x
    fi

    # repeat question
    if whiptail --title "Uploading?" \
        --yesno "Should the package be uploaded to Mentors?" \
        --yes-button "Yes" --no-button "No" 15 60
    then
        dput mentors ${UplPaket}
        echo "${UplPaket} was uploade to ${Upltext}." >> ${log}
    fi
}

<Dput2NMU 399a>
```

43.6. Als *Non-Maintainer-Upload* hochladen

Im Rahmen der Ausbesserung veröffentlichungskritischer Fehler durch Andere als dem Paket-Maintainer ist es üblich, dem Paket-Maintainer noch eine gewisse Zeit zu lassen, selber das Problem zu lösen.

```
399a <Dput2NMU 399a>≡ (398)
function Dput2NMU {
    # Called by Dput2FtpMaster

    DelayDays=$(whiptail --title "Non-Maintainer-Upload" \
        --radiolist "Days for delay?" 15 60 5 \
        "0" " 5 days of delay" off \
        "1" "10 days of delay" on \
        "2" "15 days of delay" off --cancel-button "Exit" 3>&2 2>&1 1>&3)

    <Dput2NMU1 399b>
```

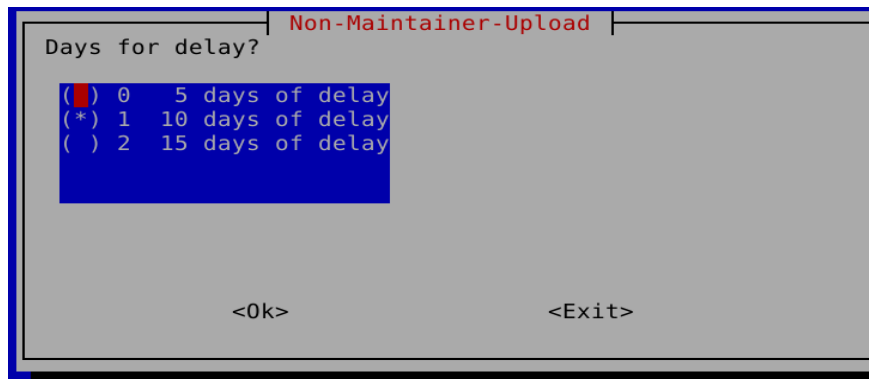


Abbildung 43.10.: Tage der Verzögerung

Es kann ausgewählt werden, wie viele Tage dem Paket-Maintainers zur eigenen Lösung des Problems verbleiben sollen.

```
399b <Dput2NMU1 399b>≡ (399a)
    if [ -z ${DelayDays} ]
    then
        exit
    fi

    case "${DelayDays}" in
        0) DelDays=5;;
        1) DelDays=10;;
        2) DelDays=15;;
    esac

    dput --delayed ${DelDays} ftp-master ${Up1Paket}
}

<Dput2FtpMaster 395b>
```

43.7. Hochladen nach *people.debian.org*

```

400a  <Upload2PeopleDO 400a>≡ (400b)
      function Upload2PeopleD0 {
          # Called by TaskSelect

          if [ ${Upl} -eq 2 ]
          then
              # For people.d.o you can not use dput
              if whiptail --title "Archive on people.d.o" \
                --yesno "Does the directory public_html/${OrigName} \
                already exist at people.d.o?\n \
                If not you have to enter the following commands\n \
                in a separate terminal:\n\n \
                ssh <user>@people.debian.org\n \
                mkdir --parents public_html/${OrigName}" \
                --yes-button "Yes" --no-button "No" 15 60
              then
                  UploadFilesSelect
          <Upload2PeopleDO3 401>
400b  <UploadFilesSelect 400b>≡ (392b)
      function UploadFilesSelect {
          # Called by Upload2PeopleD0 UploadLocal

          set +e
          UplFL=$(cat ${UplPaket} | grep --after-context=10 'Files: *')
          UplFL1=$(echo ${UplFL} | sed --expression='s/Files: //')
          i=1
          while [ $i -lt 6 ]
          do
              c=$(expr ${i} \* 5)
              UplFL2=${UplFL2}" "$(echo $UplFL1 | cut --delimiter=" " -f${c})
              i=$(expr ${i} + 1)
          done
          set -e
      }

      <Upload2PeopleDO 400a>

```

```

401  <Upload2PeopleDO3 401>≡ (400a)
    if whiptail --title "Upload file?" \
    --yesno "Should the following files\n${UplFL2}\n \
    have to be uploaded?" --yes-button "Yes" \
    --no-button "No" 15 60
    then
        if [ -z "${pdoaccount}" ]
        then
            pdoaccount=$(whiptail --title "Account at people.debian.org" \
            --inputbox "Please insert the name of your account on\n \
            people.debian.org" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
            if [ -z "${pdoaccount}" ]
            then
                echo "Please insert the name of your account on\n \
                people.debian.org"
                read pdoaccount
            fi
            changeflag=1
        fi

        if ! whiptail --title "Account name" \
        --yesno "The name of your account on people.debian.org:\n \
        ${pdoaccount}" --yes-button "Yes" --no-button "No" 15 60
        then
            pdoaccount=$(whiptail --title " Account name" \
            --inputbox "Name of your account on people.debian.org:" \
            --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
            if [ -z "${pdoaccount}" ]
            then
                echo "Please insert the name of your account on\n \
                people.debian.org"
                read pdoaccount
            fi
            changeflag=1
        fi

        if [ $changepflag -eq 1 ]
        then
            echo 'pdoaccount=${pdoaccount}' >> ${ConfigPath}/${OrigName}
            changeflag=0
        fi

        cd ${ProjectPath}/${OrigName}
        if scp -p ${UplFL2} \
        ${pdoaccount}@people.debian.org:/home/${pdoaccount}/public_html/${OrigName}
        then
            echo "${UplFL2} were uploaded to p.d.o." >> ${log}
        else
            echo "Something went wrong while uploading to p.d.o." >> ${log}
            echo "Tried to execute this command:" >> ${log}
            echo "scp -p "${UplFL2}" "${pdoaccount}@people.debian.org:/home/"\

```

```
        ${pdoaccount}"/public_html/"${OrigName} >> ${log}
    fi
    pdoarchivetext="If the archive on people.d.o should be\n \
used too, you have to enter the following commands:\n \
ssh <user>@people.debian.org\n \
cd public_html/${OrigName}\n \
apt-ftparchive packages . > Packages\n \
apt-ftparchive sources . > Sources\n \
cat Packages | gzip -c > Packages.gz\n \
cat Sources | gzip -c > Sources.gz\n \
apt-ftparchive release . > Release"

    if whiptail --title "Archive on people.d.o" \
    --yesno "${pdoarchivetext}\n\n \
Do you like to copy and paste these commands?" \
    --yes-button "Yes" --no-button "No" 15 60
    then
        echo -e $pdoarchivetext
        echo -e "\nPlease press any key to continue!"
        read x
    fi
fi
fi
fi
}
```

<UpdateLocalRepo 404>

43.8. Lokales Repositorium

Es kommt immer wieder vor, dass Pakete gebaut werden müssen, die für das eigentliche Projekt als Abhängigkeiten benötigt werden. Um die Zeit für den Gang durch die New-Queue zu überbrücken, können diese auch lokal für das Bauen in der *chroot* bereitgestellt werden.

```

403 <UploadLocal 403>≡ (404)
    function UploadLocal {
        # Called by TaskSelect
        if [ ${Upl} -eq 5 ]
        then
            # Provide for local chroot
            UploadFilesSelect
            if whiptail --title " Files Uploaded?" \
                --yesno "Should the following files\n \
                ${UplFL2}\nhave to be uploaded?" 15 60
            then
                cd ${ProjectPath}/${OrigName}
                sudo cp ${UplFL2} /var/local/repository
                UpdateLocalRepo
            fi
        fi
    }

    <ChangeEntry (nicht definiert)>

```

Die folgende Funktion gibt es bereits als Shell-Skript unter `/usr/local/bin/LocalNewRepo`.

```
404 <UpdateLocalRepo 404>≡ (401)
function UpdateLocalRepo {
    # Called by UploadLocal
    cd /var/local/repository

    # Make package archives writable
    # (not only for root)
    sudo chmod o+w Packages
    sudo chmod o+w Sources
    sudo chmod o+w Packages.gz
    sudo chmod o+w Sources.gz
    sudo chmod o+w Release

    # Use apt-ftparchive to update package archives
    sudo apt-ftparchive packages . > Packages &&
    sudo apt-ftparchive sources . > Sources &&
    sudo cat Packages | gzip -c > Packages.gz &&
    sudo cat Sources | gzip -c > Sources.gz &&
    sudo apt-ftparchive release . > Release

    # Reset rights
    sudo chmod o-w Packages
    sudo chmod o-w Sources
    sudo chmod o-w Packages.gz
    sudo chmod o-w Sources.gz
    sudo chmod o-w Release
}
```

<UploadLocal 403>

In die *Chroot-Umgebung*, hier `/var/cache/pbuilder/base.cow`, wird in die Datei `/etc/apt/sources.list` folgendes eingefügt:

```
deb [trusted=yes] file:///var/local/repository ./
deb-src [trusted=yes] file:///var/local/repository ./
```

Teil V.

Weitere Bestandteile des Skriptes

44. Weitere Aufgaben

44.1. Neuen Branch erstellen

```
407 <CreateNewBranch 407>≡
function CreateNewBranch {
    # Called by TaskSelect

    # Creates a new branch (for backports or proposed-updates)
    DebianBranches
    whiptail --title "Recent branches" \
    --msgbox "Recent branches:\n${bl}" 15 60
    bName=""
    bName=$(whiptail --inputbox "Name of the new branch:" \
    --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    if [ ${bName} != "" ]
    then
        ## Create new branch in git
        git checkout -b ${bName}

        ## Change config file - make new branch to recent one
        ChangeEntry

        whiptail --title "New branch was created" \
        --msgbox "New branch ${bName} was created" 15 60
        echo "New branch ${bName} was created" >> ${log}
        Distro4Branch
    fi
}

<ParseConfig (nicht definiert)>
```

Nun wird wieder die Aufgabenauswahl aufgerufen (Kapitel 33.5, Seite 191).

44.2. Eingabe des Namens oder der IP eines eigenen Git-Servers

Mit dieser Funktion, die von der Aufgabenauswahl (Kapitel 33.5, Seite 191) aufgerufen werden kann, werden der Name oder die IP eines eigenen Git-Servers in die Konfigurationsdatei eingetragen. Die Eingabe eines Namens setzt eine funktionierende Namensauflösung voraus.

```
408 <OwnServer 408>≡ (409)
function OwnServer {
    # Called by TaskSelect

    # Set name or IP of own git server
    ServerName=$(whiptail --inputbox "Name or IP-address of your git server:" \
        --cancel-button "Cancel" 15 60 3>&2 2>&1 1>&3)
    if [ -z "${ServerName}" ]
    then
        echo "Name or IP of your git server:"
        read ServerName
    fi
    if [ -n "$ServerName" ]
    then
        # ReplaceConfigLines needs two parameters:
        # name of the variable and new value
        ReplaceConfigLines 'ServerName' "${ServerName}"
        AddGitServer
    fi
}

<Ask4DebDiff (nicht definiert)>
```

Das Programmskript ruft die Funktion *ReplaceConfigLines* (Kapitel 33.2, Seite 181) auf, um den Git-Server in die Konfigurationsdatei einzutragen. Außerdem wird die Funktion *AddGitServer* aufgerufen.

44.3. AddGitServer

Die Funktion *AddGitServer* zeigt eine Liste der bisherigen Remote-Server an und fragt, ob der eigene Git-Server hinzugefügt werden soll.

Wird diese Frage bejaht, wird die Funktion *AddHomeServer* (Kapitel 32.4.4, Seite 159) aufgerufen.

```
409 <AddGitServer 409>≡
    function AddGitServer {
        # Called by OwnServer
        serverlist=$(git remote -v)
        if whiptail --title "Recent remote servers" \
            --yesno "${serverlist}\nAdd git remote server 'home'?" \
            --yes-button "Yes" \
            --no-button "No" 15 60
        then
            AddHomeServer
        fi
    }

    <OwnServer 408>
```


45. Kopf des Skriptes

45.1. Shebang

Am Anfang des Skriptes befinden sich die *Shebang*, Vermerke über die Autoren, die Version und die Lizenz.

```
411a <build-gbp.sh 411a>≡  
    #!/usr/bin/bash  
  
    <copyright 411b>
```

45.2. Copyright-Vermerk

Auf die *Shebang* folgt der Copyright-Vermerk.

```
411b <copyright 411b>≡ (411a)  
    # Copyright 2019-2024 Mechtilde and Michael Stehmann <mechtilde@debian.org>  
    # version 0.8.4  
  
    # This program is free software; you can redistribute it and/or modify  
    # it under the terms of the GNU General Public License as published by  
    # the Free Software Foundation; either version 3 of the License, or  
    # (at your option) any later version.  
  
    # This program is distributed in the hope that it will be useful,  
    # but WITHOUT ANY WARRANTY; without even the implied warranty of  
    # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
    # GNU General Public License for more details.  
  
    # You should have received a copy of the GNU General Public License  
    # along with this program; if not, write to the Free Software  
    # Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,  
    # MA 02110-1301, USA.  
  
    <Dependencies 412a>
```

45.3. Abhängigkeiten für das Programmskript

Sodann werden die Abhängigkeiten aufgeführt (s.a. Kapitel 19.1, Seite 59).

```
412a  <Dependencies 412a>≡ (411b)
# Dependencies: git-buildpackage, build-essential, less, pbuilder,
# pristine-tar, sudo, unzip, cowbuilder, cowdancer, debmake, quilt,
# locate, jq, lintian, devscripts, debhelper,
# sbuild, schroot, debootstrap, apt-cacher-ng,
# gradle-debian-helper, maven-debian-helper, libmaven-bundle-plugin-java

<Header 412b>
```

45.4. `set -e` und Funktionsheader

Nach der *Debian-Policy*¹ soll jedes Skript entweder `set -e` benutzen oder den Exit-Status jedes Befehls prüfen.

`set -e` veranlasst das Programmskript zum sofortigen Beenden bei Fehlern. Dies ist dann der Fall, wenn ein Befehl oder eine Pipeline (Symbol: „|“) einen *Exit*-Status ungleich 0 zurückgibt.

Dieses Verhalten ist jedoch an einigen Stellen im Programmskript unerwünscht, beispielsweise wenn der Rückgabewert vom Programmskript abgefragt und zur Ablaufsteuerung verwandt wird. In diesen Fällen wird dieses Verhalten durch `set +e` temporär ausgeschaltet.

```
412b  <Header 412b>≡ (412a)

set -e

#####

# Definitions of functions

#####

<DebugRP 413>
```

¹<https://www.debian.org/doc/debian-policy/ch-files.html#scripts> [7]

Der Funktionsheader markiert für den Leser den Beginn der technischen Beschreibungen der Funktionen.

45.5. Funktion zur Fehlersuche

Die folgende Funktion zeigt einen Pfad und ermöglicht gegebenenfalls einen Ausstieg aus dem Programm.

Sie dient dem Debugging und ist normalerweise ungenutzt.

```
413 <DebugRP 413>≡ (412b)
function DebugRP {
    # Function to show a path and give an opportunity to exit
    # It is for debugging
    descstr=${1}
    pathstr=${2}
    if ! whiptail --title "Shows the path" \
        --yesno "${descstr}= ${pathstr}?" --yes-button "Yes" \
        --no-button "No" 15 60
    then
        exit
    fi
}

<InsertConfigLine (nicht definiert)>
```


Teil VI.

Plugins und Skripte

46. Java-Plugin

```
417a <build-gbp-java-plugin.sh 417a>≡
#!/usr/bin/bash

# Copyright 2019-2023 Mechtilde and Michael Stehmann <mechtilde@debian.org>
# version 0.1.1

# java plugin for build-gbp.sh

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.

<Rules4Java 417b>
```

46.1. Anpassungen für Java-Pakete

In der Datei *debian/rules* (Kapitel 35.4.8, Seite 273) werden für das Kompilieren mit Java folgende Zeilen hinzugefügt:

```
417b <Rules4Java 417b>≡ (417a)
function Rules4Java {
    echo "export JAVA_TOOL_OPTIONS := -Dfile.encoding=UTF8" >>\
        ${GitPath}/debian/rules
    echo -e "export JAVA_HOME := /usr/lib/jvm/default-java\n" >>\
        ${GitPath}/debian/rules
}
<build-gbp-java-plugin 418>
```

Dies bedeutet, dass:

1. Immer das UTF-8 encoding genutzt wird
2. die Variable *JAVA_HOME* gesetzt ist.

Vom Skript wird lediglich eine Minimalkonfiguration der Datei *rules* bereitgestellt. Diese muss oft noch sinnvoll ergänzt werden. Für das Bauen der JAVA-Pakete werden häufig folgende Optionen benötigt.

```
JMODS := /usr/lib/jvm/default-java/jmods
JAVACMD="\$JAVA_HOME/bin/java"
```

Es kommt auch vor, dass *JDK_HOME* statt *JAVA_HOME* verwendet wird.

Diese Ergänzungen werden für die mit dem *maven*-Build-System erstellten Pakete vom *Maven-Plugin* (Kapitel 47, Seite 419) bereitgestellt.

In der Zeile mit *dh \$@* können Optionen eingefügt werden.

```
--with javahelper
--buildsystem=maven
--buildsystem=ant
--buildsystem=gradle
```

418 `<build-gbp-java-plugin 418>≡`

(417b)

```
whiptail --title "Java plugin found" \
--msgbox "build-gbp-java-plugin.sh was loaded." 15 60

echo "build-gbp-java-plugin.sh was loaded." >> ${log}
```


47. Maven-Plugin

Das Maven-Plugin unterstützt das Bauen von Java-Paketen mit dem Java-Build-System *maven* (Kapitel 13.4.1, Seite 43).

47.1. Kopf des Maven-Plugins

Der Kopfteil des Plugins enthält Angaben zu den Urhebern und der Lizenz.

Außerdem ist vermerkt, welche Debian-Pakete für den Ablauf des Programmskriptes installiert sein müssen.

```
419 <build-gbp-maven-plugin.sh 419>≡
    #!/usr/bin/bash

    # Copyright 2019-2023 Mechtilde and Michael Stehmann <mechtilde@debian.org>
    # version 0.1.1

    # maven plugin for build-gbp.sh

    # This program is free software; you can redistribute it and/or modify
    # it under the terms of the GNU General Public License as published by
    # the Free Software Foundation; either version 3 of the License, or
    # (at your option) any later version.

    # This program is distributed in the hope that it will be useful,
    # but WITHOUT ANY WARRANTY; without even the implied warranty of
    # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    # GNU General Public License for more details.

    # You should have received a copy of the GNU General Public License
    # along with this program; if not, write to the Free Software
    # Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
    # MA 02110-1301, USA.

    # Dependencies: maven-debian-helper licensecheck apt-file

    <Rules4MavenDH 432>
```

47.2. Mitteilung

Das Plugin-Skript teilt mit, dass es geladen wurde. Damit ist der Code des Plugins Teil des (Haupt-)Programmskriptes.

```
420a <MavenPlugin 420a>≡ (428)
    whiptail --title "maven plugin found" \
    --msgbox "build-gbp-maven-plugin.sh was loaded." 15 60

    echo "build-gbp-maven-plugin.sh was loaded." >> ${log}
    # Next the function MakeMaven is executed by the main script.
    # This is the end, my friend
```

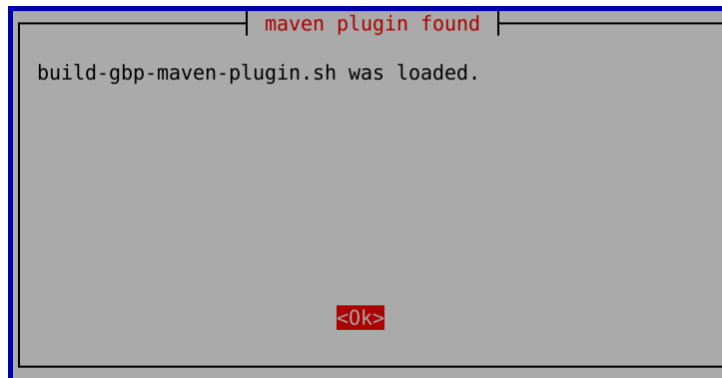


Abbildung 47.1.: Maven Plugin geladen

47.3. Bauen mit Maven

Diese Funktion wird nach dem Laden des Plugins beim Bauen einer neuen Revision von der Funktion *BuildNewVersion* des (Haupt-)Programmskriptes aufgerufen (Kapitel 35.1, Seite 254).

Die Ausführung von *mh_make* erfolgt in einer *Chroot*-Umgebung. Wenn die *chroot* noch nicht existiert, ist sie zunächst anzulegen wie in Kapitel 19.5, Seite 72 beschrieben.

```
420b <MakeMaven 420b>≡ (425)
    function MakeMaven {
        # Called by BuildNewRevision

        cd ${GitPath}
        IdentifyMavenChrootPath

        <MakeMaven1 426>
```

```

421a  <IdentifyMavenChrootPath 421a>≡ (424a)
      function IdentifyMavenChrootPath {
          # Called by MakeMaven and itself

          if [ -n "${ChrootPath}" ]
          then
              if ! whiptail --title "Maven chroot path " \
                  --yesno "Is ${ChrootPath}\nthe right path to\n\
                  maven chroot directory on the host?" \
                  --yes-button "Yes" \
                  --no-button "No" 15 60
              <IdentifyMavenChrootPath1 421b>

```

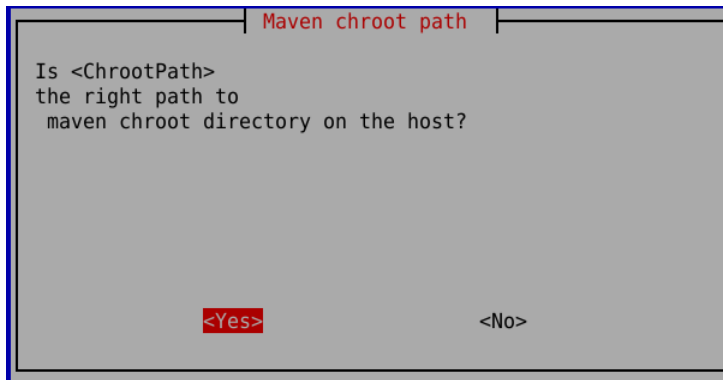


Abbildung 47.2.: Pfad zur Maven-Chroot ermitteln

```

421b  <IdentifyMavenChrootPath1 421b>≡ (421a)
      then
          OldCP=${ChrootPath}
          AskChrootPath
      fi
      else
          AskChrootPath
      fi

      <IdentifyMavenChrootPath2 422b>

```

```

421c  <AskChrootPath 421c>≡ (429b)
      function AskChrootPath {
          # Called by IdentifyMavenChrootPath

          # This is the way from GitPath to /srv/maven-chroot
          ChrootPath=$(whiptail --title "Maven chroot path" \
              --inputbox "Please insert the path\nto the maven chroot directory\n\
              on the host:" \
              --nocancel 15 60 3>&2 2>&1 1>&3)
          <AskChrootPath1 422a>

```

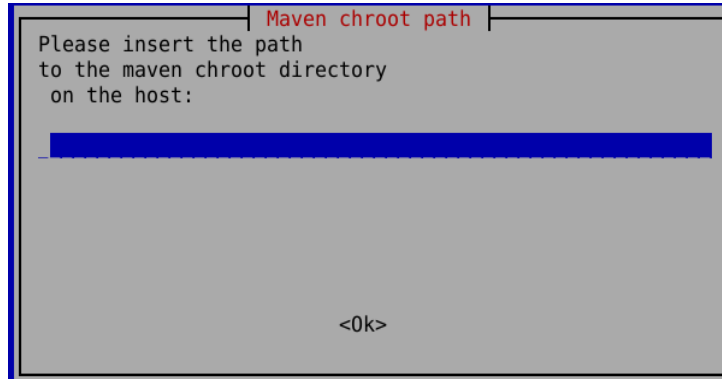


Abbildung 47.3.: Pfad zur Maven-Chroot angeben

422a $\langle AskChrootPath1 \ 422a \rangle \equiv$ (421c)

```

if [ -n "${OldCP}" ]
then
    ReplaceConfigLines "ChrootPath" ${ChrootPath}
else
    echo "### Maven chroot path"
    echo "ChrootPath=${ChrootPath} >> ${ConfigPath}${OrigName}"
fi
OldCP=""
}

```

$\langle AskWorkspacePath \ 423c \rangle$

422b $\langle IdentifyMavenChrootPath2 \ 422b \rangle \equiv$ (421b)

```

# Create Maven-Chroot if not exists
if ! [ -d ${ChrootPath} ]
then
    echo "Please enter your SUDO password!"
    sudo mkdir --parents ${ChrootPath}
fi

if ! [ -d ${ChrootPath}/usr ]
then
    echo "Please enter your SUDO password!"
    sudo /usr/sbin/debootstrap --arch amd64 sid \
    ${ChrootPath} http://ftp.de.debian.org/debian
    echo "The maven chroot has been created." >> ${log}
fi

```

$\langle IdentifyMavenChrootPath3 \ 423a \rangle$

```

423a  <IdentifyMavenChrootPath3 423a>≡ (422b)
      # The workspace is /home/user
      if [ -n "${Path2Workspace}" ]
      then
          if ! whiptail --title "Maven chroot path " \
            --yesno "Is ${Path2Workspace}\nthe workspace in the maven chroot?" \
            --yes-button "Yes" \
            --no-button "No" 15 60 # test
          <IdentifyMavenChrootPath4 423b>

```

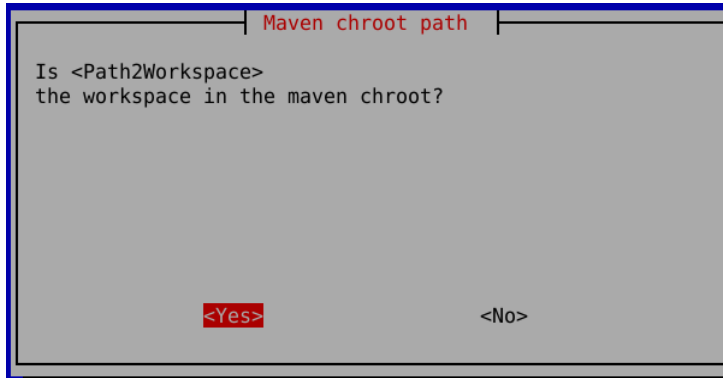


Abbildung 47.4.: Arbeitsverzeichnis in der Maven-Chroot ermitteln

```

423b  <IdentifyMavenChrootPath4 423b>≡ (423a)
      then
          OldP2W=${Path2Workspace}
          AskWorkspacePath
      fi
      else
          AskWorkspacePath
      fi
      <IdentifyMavenChrootPath5 424b>

```

```

423c  <AskWorkspacePath 423c>≡ (422a)
      function AskWorkspacePath {
          # Called by IdentifyMavenChrootPath

          # This is the way to /home/user/
          Path2Workspace=$(whiptail --title "Maven chroot path" \
            --inputbox "Please insert the path\nto the workspace directory:" \
            --nocancel 15 60 3>&2 2>&1 1>&3)
          <AskWorkspacePath2 424a>

```

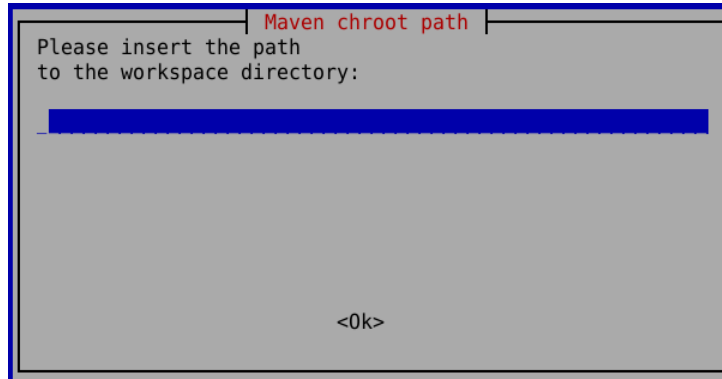


Abbildung 47.5.: Arbeitsverzeichnis in der Maven-Chroot angeben

424a $\langle AskWorkspacePath2 \ 424a \rangle \equiv$ (423c)

```

if [ -n "${OldP2W}" ]
then
    ReplaceConfigLines "Path2Workspace" "${Path2Workspace}"
else
    echo "Path2Workspace=${Path2Workspace} >> ${ConfigPath}${OrigName}"
fi
OldP2W=""
}

```

$\langle IdentifyMavenChrootPath \ 421a \rangle$

424b $\langle IdentifyMavenChrootPath5 \ 424b \rangle \equiv$ (423b)

```

# Replace / at the end
Path2Workspace=$(echo ${Path2Workspace} | sed --expression="s/\$///")
MavenChrootPath=${ChrootPath}${Path2Workspace}
if ! [ -d ${MavenChrootPath} ]
then
    whiptail --title "Error!" \
    --msgbox "${MavenChrootPath} does not exist.\n\
    It will be created now." 15 60

```

$\langle IdentifyMavenChrootPath7 \ 425 \rangle$

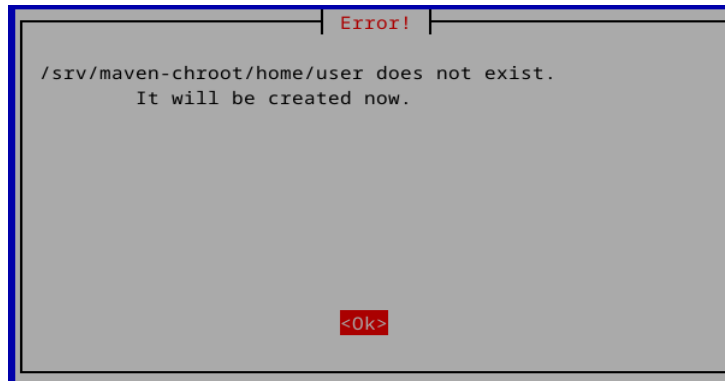


Abbildung 47.6.: Maven-Chroot existiert nicht

Existiert die *chroot* nicht, ist sie anzulegen wie in Kapitel 19.5, Seite 72 beschrieben.

```
425 <IdentifyMavenChrootPath7 425>≡ (424b)
    echo "Please enter your SUDO password!"
    sudo mkdir --parents ${MavenChrootPath}
    echo "The maven work space has been created." >> ${log}
  fi
}

<MakeMaven 420b>
```

```

# Copy workspace to maven chroot
sudo cp --archive ${GitPath} ${MavenChrootPath}

# Because mh_make has to run in the directory, where pom.xml is
echo "pom.xml is here:"
find . -name 'pom.xml' -print
echo "Please switch to the Maven chroot in another terminal."
echo "Use these commands:"
echo
echo "# sudo mount -o bind /proc /srv/maven-chroot/proc"
echo "# sudo mount devpts /dev/pts -t devpts"
echo
echo "sudo LANG=C chroot "${ChrootPath}" /usr/bin/bash"
echo "cd "${Path2Workspace}"/"${SourceName}"
echo "try: apt install maven-debian-helper"
echo "apt update && apt upgrade"
echo "mh_make --verbose "${SourceName}"
echo
echo "Then follow the questions of mh_make"
echo
echo "You can leave the chroot with 'exit'."
echo
echo "After finishing press return to go on!"
read a

#--run-tests=true --javadoc=true --verbose
#mh_make --package=${SourceName} --bin-package=${PackName} \
#--run-tests=false --javadoc=false --verbose
#if [ $? -ne 0 ]
#then
#   echo "mh_make failed!"
#   exit
#else
#   echo "mh_make has been executed for "${PackName} >> ${log}
#fi

cp --recursive --update ${MavenChrootPath}/${SourceName}/debian \
${GitPath}

```

<MakeMaven5 428>

Beim Ablauf des Programmes *mh_make* sind folgende Fragen zu beantworten. Dieses Programm selber in auch ein Shell-Skript.

Environment variable DEBLICENSE not set, using Apache-2.0 by default Die Umgebungsvariable *DEBLICENSE* ist nicht gesetzt, verwendet wird dann standardmäßig die Lizenz Apache-2.0. In der Variablen *DEBLICENSE* wird die Lizenz für die Dateien in *debian/* hinterlegt.

Checking that apt-file is installed and has been configured... [ok] Prüfung, ob das Paket *apt-file* installiert ist. Dieses Paket ist nur als *empfohlen* markiert.

Checking that licensecheck is installed... [ok] Prüfung, ob das Paket *licensecheck* installiert ist. Dieses Paket ist ebenfalls nur als *empfohlen* markiert.

Solving dependencies for package <PaketName>

Analysing pom.xml...

Resolving com.jcabi:jcabi:pom:1.21 of scope runtime...

In pom.xml: The parent POM cannot be found in the Maven repository for Debian. Ignore it? com.jcabi:jcabi:pom:1.21 [Y/n]

Dass die Abhängigkeit für das Elternelement nicht gefunden wird, ist der Regelfall. Daher wird die Frage bejaht, dass dies ignoriert werden kann. Diese Information wird dann als *-no-parent* in die Datei *<PaketName>.poms* geschrieben.

Enter the upstream version for the package. Hier wird die zu bauende Versionsnummer des Upstream-codes abgefragt und in der Regel mit *Enter* bestätigt. Andernfalls wird die korrekte Versionsnummer eingegeben.

Version of com.jcabi:jcabi-aspects is 0.22.6 Es wird nochmal die Versionsnummer angezeigt.

Choose how the version will be transformed: Nun wird eine Auswahl angezeigt, wie diese Versionsnummer umgewandelt werden soll.

0 - Replace all versions starting by 0. with 0.x Ersetze alle Versionsbezeichnungen

(1) - Change the version to the symbolic 'debian' version Die Standardangabe wird mit eckigen Klammern (Hier sind es aus satztechnischen Gründen runde Klammern) gekennzeichnet. Diese kann mit *Enter* bestätigt werden.

2 - Keep the version

3 - Custom rule

Andernfalls wird nun die gewünschte Ziffer eingegeben.

com.jcabi:jcabi-aspects is a bundle. - Inform mh_make that dependencies of type jar which may match this library should be transformed into bundles automatically? [Y/n]

Soll *mh_make* mitgeteilt werden, dass die passenden Abhängigkeiten vom Typ *jar* in Bundles umgewandelt werden. Standardmäßig wird diese Frage bejaht.

Resolving com.jcabi:jcabi-log:jar:0.17 of scope runtime... [check dependency with bundle type]

In pom.xml: This dependency cannot be found in the Debian Maven repository. Ignore this dependency? com.jcabi:jcabi-log:jar:0.17 [y/N]

Diese Meldung kommt, wenn die benötigte Abhängigkeit nicht auf der Arbeitmaschine installiert ist.

- `>dpkg -search /usr/share/maven-repo/com/jcabi/jcabi-log/*/* dpkg failed to execute successfully`
- `>apt-file search /usr/share/maven-repo/com/jcabi/jcabi-log`

```

Found /usr/share/maven-repo/com/jcabi/jcabi-log/0.18.1/jcabi-log-0.18.1.pom
in libjcabi-log-java
Found /usr/share/maven-repo/com/jcabi/jcabi-log/0.19.0/jcabi-log-0.19.0.pom
in libjcabi-log-java
Found /usr/share/maven-repo/com/jcabi/jcabi-log/debian/jcabi-log-debian.pom
in libjcabi-log-java

```

- >dpkg -search /usr/share/java/jcabi-log.jar dpkg failed to execute successfully
- >apt-file search /usr/share/java/jcabi-log.jar Found libjcabi-log-java [error] Package libjcabi-log-java does not contain Maven dependency com.jcabi:jcabi-log:jar:0.17 but there seem to be a match If the package contains already Maven artifacts but the names don't match, try to enter a substitution rule of the form s/groupId/newGroupId/ s/artifactId/newArtifactId/ jar s/version/newVersion/ here: >

Nach der Installation des fehlenden Paketes auf der Arbeitsmaschine kann die Suche wiederholt werden.

```

428 <MakeMaven5 428>≡ (426)
if [ -w debian/maven.rules ]
then
    echo "#[groupId] [artifactId] [type] [version] [classifier] [scope]" \
    >> debian/maven.rules
fi

ShowMaven

# Patch for mh_make bug
str4standardsversion="4.5.1"
cd ${GitPath}/debian/
less --LINE-NUMBERS control

sed --in-place --expression=\
"s/Standards-Version: 4.4.1/Standards-Version: ${str4standardsversion}/" \
control
# 'a' means append. The string after the 'a' will be appended
# to the sting before the 'a'.
sed --in-place \
--expression="/Standards-Version: ${str4standardsversion}/ \
a Rules-Requires-Root: no" \
control
sed --in-place --expression=\
"s/debhelper-compat (=12)/debhelper-compat ${str4versiondebhelpers}/" \
control

cd ${GitPath}
whiptail --title "Check debian/control" \
--msgbox "Please check the control file another time!" 15 60
less --LINE-NUMBERS ${GitPath}/debian/control
}

```

<MavenPlugin 420a>

47.4. Maven-Dateien bearbeiten

Pakete, die mit *Maven* (*mvn*) gebaut werden, benötigen dazu weitere Dateien im Verzeichnis *debian/*. Diese werden im Folgenden aufgelistet.

```
429a <ShowMaven 429a>≡ (432)
function ShowMaven {
    # Called by DisplayDebianFiles MakeMaven

    nano --linenumbers --mouse --softwrap debian/${PackName}.poms
```

<ShowMaven2 429b>

- maven.cleanIgnoreRules
- maven.properties
- maven.rules
- maven.ignoreRules
- maven.publishedRules

Nun können diese Dateien auch editiert werden.

```
429b <ShowMaven2 429b>≡ (429a)
mavenl=$(ls ${GitPath}/debian/ | grep 'maven')
for element in ${mavenl[*]}
do
    nano --linenumbers --mouse --softwrap debian/${element}
done

pomsl=$(ls ${GitPath}/debian/ | grep ${PackName})
for element in ${pomsl[*]}
do
    nano --linenumbers --mouse --softwrap debian/${element}
done
}
```

<AskChrootPath 421c>

47.4.1. maven.rules

Diese Datei ist wie folgt aufgebaut:

```
429c <maven-rules.header 429c>≡
#[groupID] [artifactID] [type] [version] [classifier] [scope]
```

47.4.2. maven.ignoreRules

Diese Datei ist wie die Datei *maven.rules* aufgebaut.

```
429d <maven-ignoreRules.header 429d>≡
#[groupID] [artifactID] [type] [version] [classifier] [scope]
```

Artefakte, die in der Datei *maven.rules* aufgenommen werden, sind hier gegebenenfalls zu löschen

47.4.3. maven.properties

Die Datei *maven.properties* kann verwendet werden, um Werte von Variablen innerhalb der *pom.xml*-Dateien zu überschreiben. Der häufigste Anwendungsfall ist die Deaktivierung oder Aktivierung der Tests mit *maven.test.skip=true*

Weitere Optionen sind die Änderung der Quell- (*maven.compiler.source=8*) bzw. Zielebene (*maven.compiler.target=8*). Dabei bezieht sich der Wert (hier: 8) auf die java-Mindestversion. Dieser Eintrag wird besonders dann benötigt, wenn beim Kompilieren eine Meldung wie die folgende ausgegeben wird: *use -source 8 or higher to enable ...*

Außerdem kann hier die verwendete Dateikodierung (*project.build.sourceEncoding=UTF-8*) eingestellt werden.

```
430 <maven.properties 430>≡
    # Include here properties to pass to Maven during the build.
    # For example:
    # maven.test.skip=true
    # project.build.sourceEncoding=UTF-8
    # maven.compiler.source=8
    # maven.compiler.target=8
    # To skip test - missing dependencies
    maven.test.skip=true
```

Häufig werden die Tests deaktiviert, um nicht zusätzliche Abhängigkeiten paketieren zu müssen. In diesen Fällen sind diese Abhängigkeiten auch in der Datei *pom.xml* mit einem Kommentarzeichen zu versehen oder zu entfernen.

Solche Änderungen am Upstream-Code können mit dem Programmskript vorgenommen werden (Kapitel 36, Seite 285).

47.4.4. Paketname.poms

In der Datei *<Paketname>.poms* werden die Optionen zu den jeweiligen *pom.xml*-Dateien hinterlegt.

Die Datei enthält dann 2 Spalten. In der ersten Spalte steht die *pom.xml* mit dem dazugehörigen Pfad und in der zweiten Spalte die dazugehörigen Optionen, wie z.B.

```
# List of POM files for the package
# Format of this file is:
# <path to pom file> [option]*
# where option can be:
# --ignore: ignore this POM and its artifact if any
# --ignore-pom: don't install the POM. To use on POM files that are
#   created temporarily for certain artifacts such as Javadoc jars.
#   [mh_install, mh_installpoms]
# --no-parent: remove the <parent> tag from the POM
# --package=<package>: an alternative package to use when installing
#   this POM and its artifact
# --has-package-version: to indicate that the original version of
#   the POM is the same as the upstream part of the version for the
#   package.
# --keep-elements=<elem1,elem2>: a list of XML elements to keep in the
#   POM during a clean operation with mh_cleanpom or mh_installpom
# --artifact=<path>: path to the build artifact associated with this
#   POM, it will be installed when using the command mh_install.
#   [mh_install]
# --java-lib: install the jar into /usr/share/java to comply with
#   Debian packaging guidelines
# --usj-name=<name>: name to use when installing the library in
#   /usr/share/java
# --usj-version=<version>: version to use when installing the library
#   in /usr/share/java
# --no-usj-versionless: don't install the versionless link in
#   /usr/share/java
# --dest-jar=<path>: the destination for the real jar.
#   It will be installed with mh_install. [mh_install]
# --classifier=<classifier>: Optional, the classifier for the jar.
#   Empty by default.
# --site-xml=<location>: Optional, the location for site.xml if it
#   needsto be installed.
```

13. Oktober 2024

```
# Empty by default. [mh_install]
#
pom.xml --no-parent --has-package-version
```

Dies bedeutet, dass das `<parent>`-Tag beim Bauen aus der POM entfernt wird. Die Option `-has-package-version` gibt an, dass die Originalversion der POM dieselbe sei, wie der Upstream-Teil der Paketversion.

47.4.5. README.source

Information about jcabi-aspects

This package was debianized using the `mh_make` command from the `maven-debian-helper` package.

The build system uses Maven but prevents it from downloading anything from the Internet, making the build compliant with the Debian policy.

47.5. *debian/rules* - Ergänzungen für Java-Pakete mit *Maven*

In der Funktion *Rules4MavenDH* wird die Datei *debian/rules* um die Angabe des Build-Systems ergänzt.

```
432 <Rules4MavenDH 432>≡ (419)
    function Rules4MavenDH {
        # Called by DebianRulesTemplates

        sed --in-place \
            --expression="s/dh $@/dh $@ --buildsystem=maven" \
            ${GitPath}/debian/rules

    }
    <ShowMaven 429a>
```

48. Web-Extension-Plugin

Das nachstehend dargestellte Plugin erfüllt die besonderen Anforderungen beim Bauen von Mozilla-Erweiterungen als Debian-Pakete (Kapitel 14, Seite 49).

48.1. Kopf des Webext-Plugins

Der Kopfteil des Plugins enthält Angaben zu den Urhebern und der Lizenz.

Außerdem ist vermerkt, welche Debian-Pakete für den Ablauf des Programmskriptes installiert sein müssen.

```
433 <build-gbp-webext-plugin.sh 433>≡
    #!/usr/bin/bash

    # Copyright 2020-2023 Mechtilde and Michael Stehmann <mechtilde@debian.org>
    # version 0.1.1

    # webext plugin for build-gbp.sh

    # This program is free software; you can redistribute it and/or modify
    # it under the terms of the GNU General Public License as published by
    # the Free Software Foundation; either version 3 of the License, or
    # (at your option) any later version.

    # This program is distributed in the hope that it will be useful,
    # but WITHOUT ANY WARRANTY; without even the implied warranty of
    # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    # GNU General Public License for more details.

    # You should have received a copy of the GNU General Public License
    # along with this program; if not, write to the Free Software
    # Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
    # MA 02110-1301, USA.

    <IdentifyWebextId 434>
```

48.2. Erstellen der *webext*.xpi*-Dateien in *debian/*

48.2.1. Ermittlung des Namens der **.xpi*-Datei

Die zu paketierende Erweiterung muss unter dem Namen der sich aus der Bezeichnung (id) in der Datei *manifest.json* und der Endung *.xpi* zusammensetzt, abgelegt werden.

Die Bezeichnung wird daher wie folgt ermittelt.

```
434 <IdentifyWebextId 434>≡ (433)
function IdentifyWebextId {
    # Called by WebextRulesDH WebextInstall

    if [ -z ${webextID} ] && [ -f ${GitPath}/manifest.json ]
    then
        webextID=$(grep '"id":' ${GitPath}/manifest.json)
        webextID=$(echo ${webextID} | sed --expression='s/^\\"id\": \"/')
        webextID=$(echo ${webextID} | sed --expression='s/\",\\s*//')

        echo -e "Notice from Webext-Plugin: WebextID = \"${webextID}\" >> ${log}
    fi

    if [-z ${webextID} ]
    then
        webextID="PLEASE_REPLACE_WITH_ID"
        whiptail --title "ID not found" \
            --msgbox "Didn't find ID or manifest.json in ${GitPath}" 15 60
    fi
}

<webext-rules 435a>
```


48.2.2. *debian/rules* - Ergänzungen für Mozilla-AddOns

Bei der *debian/rules*-Datei für die Webextension ist es sinnvoll, die zu installierenden Verzeichnisse bzw. Dateien dort aufzuführen und einer Variablen zuzuweisen. Dies erfolgt nach dem Muster `<Package>_FILES = <List of files to include>`

Die Liste `<Package>_FILES` wird vom Plugin-Skript automatisch erstellt. Sie ist auf Vollständigkeit und Richtigkeit zu überprüfen. Die auszuschließenden Dateien sind manuell einzupflegen.

Da die Anlage der Datei *debian/rules* einmalig erfolgt, sind beim Paketieren neuer Versionen die beiden Listen sorgfältig zu prüfen und gegebenenfalls zu anzupassen.

```
435a <webext-rules 435a>≡ (434)
function WebextRules {
    # Called by DebianRulesTemplate

    # These strings will be added to str4rules

    Package=$(echo ${SourceName} | tr "a-z" "A-Z")
    echo -e "${Package}"_FILES = "\\" >> ${GitPath}/debian/rules

    PackageL=$(ls ${GitPath})
    PackageA=(${PackageL})

    for element in ${PackageA[*]}
    do
        if [ "${element}" != "debian" ]
        then
            echo -e "${element}" "\\" >> ${GitPath}/debian/rules
        fi
    done
    echo "\$(NULL)" >> ${GitPath}/debian/rules

    <webext-rules5 435b>

435b <webext-rules5 435b>≡ (435a)
    echo -e "\n Uncomment the following lines \n"
    echo "if there are files to exclude and add them."
    echo -e "\n# "${Package}"_FILES_EXCLUDE = "\\" \
    >> ${GitPath}/debian/rules
    echo -e "# \$(NULL)\n" >> ${GitPath}/debian/rules
}

<webext-rules-dh 436>
```

Die folgenden Zeilen werden vom Programmskript der Variablen *str4rulesdh* hinzugefügt.

```

436 <webext-rules-dh 436>≡ (435b)
function WebextRulesDH {
    # Called by DebianRulesTemplate

    IdentifyWebextId

    DHCleanStr="override_dh_clean:\n\tdh_clean\n\trm -rf debian/build\n"

    DHAutoBuildIndepStr1="override_dh_auto_build-indep:"
    DHAutoBuildIndepStr2="\tmkdir \${CURDIR}/debian/build && \\"
    Str3B="\tzip --recurse-paths "
    DHAutoBuildIndepStr3="\${Str3B}"\${CURDIR}/debian/build/\${webextID}".xpi \\"
    DHAutoBuildIndepStr4="\t \${Package}_FILES) \\"
    DHAutoBuildIndepStr5="# \t --exclude \${Package}_FILES_EXCLUDE)"
    DHAutoBuildIndepStr6="\tdh_auto_build\n"

    echo -e \${DHCleanStr} >> \${GitPath}/debian/rules
    echo -e \${DHAutoBuildIndepStr1} >> \${GitPath}/debian/rules
    echo -e \${DHAutoBuildIndepStr2} >> \${GitPath}/debian/rules
    echo -e \${DHAutoBuildIndepStr3} >> \${GitPath}/debian/rules
    echo -e "\${DHAutoBuildIndepStr4}" >> \${GitPath}/debian/rules
    echo -e "\${DHAutoBuildIndepStr5}" >> \${GitPath}/debian/rules
    echo -e \${DHAutoBuildIndepStr6} >> \${GitPath}/debian/rules
}

<WebextControl 437a>

```

48.2.3. *debian/control* - Ergänzungen für Mozilla-AddOns

```
437a <WebextControl 437a>≡ (436)
function WebextControl {
    # Called by DebianControlTemplate
    # TB specific, for FF 'web'
    sed --in-place \
        --expression="s/Section: /Section: mail/" ${GitPath}/debian/control

    # 'a' means append. The string after the 'a' will be appended
    # to the sting before the 'a'.
    # @X escapes the space at the beginning of the appended line.
    # It will be removed later.
    sed --in-place \
        --expression="/Build-Depends: debhelper-compat ${str4versiondebhelpers}/ \
a @X , zip" \
        ${GitPath}/debian/control
    # TB specific, for FF 'firefox-esr (>= 91.4)'
    sed --in-place \
        --expression="/Depends: \${misc:Depends}/ \
a @X , thunderbird (>= 1:115.10)" \
        ${GitPath}/debian/control
    sed --in-place --expression="s/^@X//g" ${GitPath}/debian/control
}

<WebextInstall 437b>
```

48.2.4. *debian/webext-*.install*

Für Mozilla-Erweiterungen ist zwingend der folgende Eintrag erforderlich:

```
debian/build/<manifest-id>.xpi /usr/share/webext
```

```
437b <WebextInstall 437b>≡ (437a)
function WebextInstall {
    # Called by DisplayDebianFiles
    IdentifyWebextId
    InstallStr="debian/build/"${webextID} ".xpi\tusr/share/webext"
    echo -e ${InstallStr} >> ${GitPath}/debian/${PackName}.install
}

<WebextDocs 437c>
```

48.2.5. *debian/webext-*.docs*

```
437c <WebextDocs 437c>≡ (437b)
function WebextDocs {
    # Called by
    echo "Still empty"
}

<WebextLinks 438a>
```

48.2.6. *debian/webext-links-tb*

```
\#           Source                               Target
/usr/share/webext/<manifest.id>.xpi    /usr/lib/thunderbird/extensions/<manifest.id>.xpi
```

```
438a  <WebextLinks 438a>≡ (437c)
function WebextLinksTB {
    # Called by DisplayDebianFiles
    IdentifyWebextId
    SourceStr="/usr/share/webext/"${webextID}".xpi\t"
    TargetStr="/usr/lib/thunderbird/extensions/"${webextID}".xpi"
    echo -e ${SourceStr}${TargetStr} >> \
    ${GitPath}/debian/${PackName}.links
}

<webext-plugin-end 438b>
```

48.2.7. Meldung: Webext-Plugin geladen

Nun kommt der Schluss dieses Webext-Plugins.

```
438b  <webext-plugin-end 438b>≡ (438a)
whiptail --title "webext plugin found" \
--msgbox "build-gbp-webext-plugin.sh was loaded." 15 60
# This is the end, my friend
```

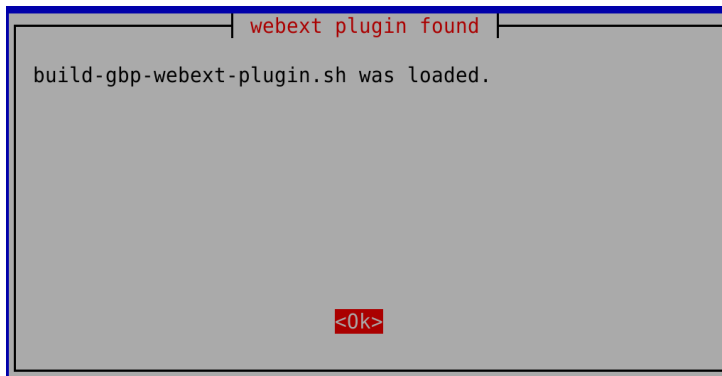


Abbildung 48.1.: Webext-Plugin wurde geladen

49. Python-Plugin

Das nachstehend dargestellte Plugin erfüllt die besonderen Anforderungen beim Bauen von Paketen, die in der Programmiersprache Python geschrieben sind. (Kapitel 15, Seite 51).

```
439a <build-gbp-python-plugin.sh 439a>≡
#!/usr/bin/bash

# Copyright 2020-2022 Mechtilde and Michael Stehmann <mechtilde@debian.org>
# version 0.1.1

# python plugin for build-gbp.sh

# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 3 of the License, or
# (at your option) any later version.

# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
# MA 02110-1301, USA.

<python-rulesDH 440b>
```

49.0.1. Meldung: Python-Plugin geladen

Nun kommt der Schluss dieses Plugin-Plugins. Es gibt bekannt, dass es geladen wurde.

```
439b <IdentifyPythonId 439b>≡ (441b)
whiptail --title "python plugin found" \
--msgbox "build-gbp-python-plugin.sh was loaded." 15 60

echo "build-gbp-python-plugin.sh was loaded." >> ${log}

<python-plugin-end 441c>
```

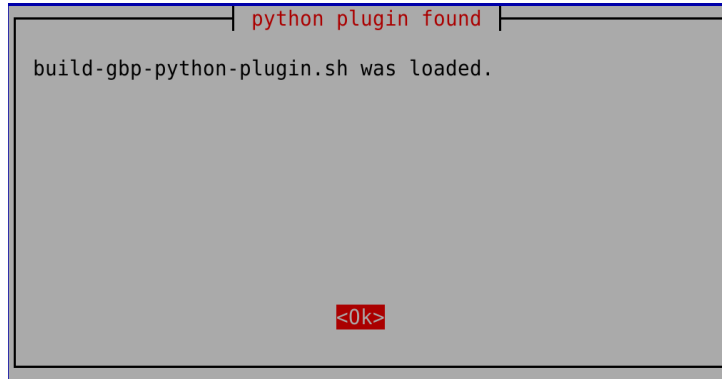


Abbildung 49.1.: Python-Plugin geladen

49.1. Anpassungen für Python-Pakete

Hier werden die Besonderheiten für die Python-Paketierung in der Datei *debian/rules* abgebildet.

```
440a <python-rules 440a>≡ (440b)
    function PythonRules {
        # Called by DebianRulesTemplate
```

```

        # These strings will be added to str4rules
        echo -e "export PYBUILD_NAME=\"${SourceName}\"\\n" >> ${GitPath}/Debian/rules
        echo -e "export PYBUILD_SYSTEM=distutils\\n" >> ${GitPath}/Debian/rules
    }
```

```
<python-control 441a>
```

```
440b <python-rulesDH 440b>≡ (439a)
    function PythonRulesDH {
        # Called by DebianRulesTemplate
```

```

        sed --in-place \
        --expression="s/dh $@/dh $@ --with python3 --buildsystem=pybuild/" \
        ${GitPath}/debian/rules
```

```
    }
    <python-rules 440a>
```

49.2. *debian/control* - Ergänzung für Python-Pakete

Hier werden die Besonderheiten für die Python-Paketierung in der Datei *debian/control* abgebildet.

```
441a <python-control 441a>≡ (440a)
function PythonControl {
    # Called by DebianControlTemplate

    # Recent Python version
    PythonVersion=$(py3versions --installed)

    sed --in-place \
    --expression="s/Section:/Section: python/" ${GitPath}/debian/control

    sed --in-place \
    --expression="/Build-Depends: debhelper-compat ${str4versiondebhelpers}/ \
    a , dh-python@X , python3-all@X , python3-setuptools@X\
    X-Python3-Version: >=${PythonVersion}" ${GitPath}/debian/control

    sed --in-place \
    --expression=""

    <python-control1 441b>
```

Für Tests werden auch die Pakete *python3-distutils* und *python3-distutils-extra* als Build-Abhängigkeit benötigt.

```
441b <python-control1 441b>≡ (441a)

    sed --in-place \
    --expression="/Depends: \${misc:Depends}/ \
    a @X , \${python3:Depends}" \
    ${GitPath}/debian/control

    sed --in-place --expression="s/@X/\n/g" ${GitPath}/debian/control
}

<IdentifyPythonId 439b>
```

```
441c <python-plugin-end 441c>≡ (439b)

# This is the end, my friend
```


50. Skripte

50.1. Anlage eines Projektes im Java-Team

Die Anlage eines Projektes im Java-Team, einschließlich der Erlangung eines Zugangstokens, welches der Variablen `SALSA_TOKEN` im nachfolgenden Skript zuzuweisen ist, wird in Kapitel 21.3 (Seite 80) beschrieben.

```
443a <setup-salsa-repository 443a>≡
#!/usr/bin/bash
#
# Setup a new Git repository on Salsa
#
# This script uses the GitLab REST API and requires an access token.
# The token is obtained from the GitLab profile page -> Access Tokens
# (https://salsa.debian.org/profile/personal_access_tokens).
# The token is in the environment variable SALSA_TOKEN
# or has to be sourced from the ~/.salsarc file and assigned
# to the SALSA_TOKEN variable:
#
# This is the Token for jollyday-java
# SALSA_TOKEN="KyuZRyWTTxfddcGcyphN"
#

set -eu
<setup-salsa-repository3 443b>
```

Das Paket `jq` muss auf der lokalen Maschine installiert werden.

```
443b <setup-salsa-repository3 443b>≡ (443a)

if ! which jq >/dev/null
then
    echo "You need to apt install jq" >&2
    exit 1
fi

<setup-salsa-repository5 444a>
```

Der Aufruf erfolgt mit der Angabe des *SourceName*.

```
444a <setup-salsa-repository5 444a>≡ (443b)
if [ -z "$1" ]; then
    echo "Usage: ./setup-salsa-repository <packagename>"
    exit 1;
fi

check_return_code() {
    if [ $? -ne 0 ]; then
        echo
        echo "Something went wrong!"
        exit 1
    fi
}

test -n "$SALSA_TOKEN" || . ~/.salsarc

PACKAGE=$1

SALSA_URL="https://salsa.debian.org/api/v4"
SALSA_GROUP=java-team
SALSA_GROUP_ID=2588
```

<setup-salsa-repository8 444b>

Nun wird das Repositorium auf *salsa.debian.org* angelegt.

```
444b <setup-salsa-repository8 444b>≡ (444a)
# -----

echo "Creating the ${PACKAGE} repository..."

RESPONSE=$(curl -s "$SALSA_URL/projects?private_token=$SALSA_TOKEN" \
  --data "path=${PACKAGE}&namespace_id=\
  ${SALSA_GROUP_ID}&visibility=public&issues_enabled=false&snippets_enabled=false&wiki_enable\
  d=false&jobs_enabled=false&printing_merge_request_link_enabled=false")

echo $RESPONSE | jq --exit-status .id > /dev/null
check_return_code

PROJECT_ID=$(echo $RESPONSE | jq '.id')
<setup-salsa-repository10 445>
```

Nun wird das ausstehende BTS-Tag-Hook konfiguriert.

```

445 <setup-salsa-repository10 445>≡ (444b)
# -----

echo "Configuring the BTS tag pending hook..."

TAGPENDING_URL="https://webhook.salsa.debian.org/tagpending/$PACKAGE"
curl --silent --output /dev/null -XPOST --header "PRIVATE-TOKEN: $SALSA_TOKEN" \
  $SALSA_URL/projects/$PROJECT_ID/hooks \
  --data "url=$TAGPENDING_URL&push_events=1&enable_ssl_verification=1"
check_return_code

# -----

echo "Configuring the KGB hook..."

KGB_URL="http://kgb.debian.net:9418/webhook/?channel=debian-java\
%26network=oftc%26private=1%26use_color=1%26use_irc_notices=1%26squash_threshold=20"
curl --silent --output /dev/null -XPOST --header "PRIVATE-TOKEN: $SALSA_TOKEN" \
  $SALSA_URL/projects/$PROJECT_ID/hooks \
  --data "url=\
  $KGB_URL&push_events=yes&issues_events=yes&\
  merge_requests_events=yes&tag_push_events=\
  yes&note_events=yes&job_events=yes&pipeline_events=yes&\
  wiki_events=yes&enable_ssl_verification=yes"
check_return_code

# -----

echo "Configuring email notification on push..."

curl --silent --output /dev/null -XPUT --header "PRIVATE-TOKEN: $SALSA_TOKEN" \
  $SALSA_URL/projects/$PROJECT_ID/services/emails-on-push \
  --data "recipients=pkg-java-commits@lists.alioth.debian.org \
  dispatch@tracker.debian.org"
check_return_code

# -----

echo
echo "Done! The repository is located at ${SALSA_URL%/api*}/$SALSA_GROUP/$PACKAGE"

```

50.2. Skript zum Extrahieren der Dokumentation im PDF- und Epub-Format

Dieses Skript dient dazu, lesbare Dokumente zu erstellen.

50.2.1. Abhängigkeiten

Folgende Pakete müssen installiert sein, damit das Skript lauffähig ist.

- noweb
- texlive
- texlive-latex-extra
- texlive-extra-utils
- biber
- texlive-lang-japanese (für die Datei ifptex.sty)
- tidy (für das EBook)
- texlive-lang-german - für die deutschsprachige Dokumentation
- texlive-lang-english - für die englische Übersetzung
- texlive-lang-french - für die französische Übersetzung

50.2.2. Ablauf

Zunächst werden mit dem Skript aus den *.nw-Dokumenten *.tex-Dokumente erzeugt.

Das folgende Skript kann mit

```
notangle -Rcreate-book.sh Part6.nw > create-book.sh &&
t=$(date +%c)
echo "#generated on $t" >> create-book.sh
chmod ugo+x create-book.sh
```

aus der Datei *Part6.nw* extrahiert werden.

```
446 <create-book.sh 446>≡
#!/usr/bin/bash

#set -e

BasePath=$(pwd)

LANGS="en_US"

noweave -index -delay BuildWithGBP.nw > BuildWithGBP.tex # contains preamble
noweave -index -delay Title.nw > GBP-Title.tex
noweave -index -delay Part1.nw > GBP-Part1.tex
noweave -index -delay Part2.nw > GBP-Part2.tex
noweave -index -delay Part3.nw > GBP-Part3.tex
noweave -index -delay Part4.nw > GBP-Part4.tex
noweave -index -delay Part5.nw > GBP-Part5.tex
noweave -index -delay Part6.nw > GBP-Part6.tex

#noweave -filter l2h -index -html BuildWithGBP.nw | htmltoc > BuildWithGBP.html
```

⟨*create-book.sh1* 447a⟩

Diese wird dann zu *.pdf- und *.epub-Dokumente umgewandelt. Das Stichwortverzeichnis darf nur einmal erstellt werden und zwar nach der ersten Ausführung von *pdflatex*. Mehrfache Ausführung erzeugt zu hohe Seitenzahlen.

```
447a ⟨create-book.sh1 447a⟩≡ (446)
  if [ -f BuildWithGBP.aux ]
  then
    rm BuildWithGBP.aux
  fi

  for ((i=4; i>0; i--))
  do
    echo $i
    pdflatex -shell-escape BuildWithGBP.tex

    # Create Index only one time
    # Otherwise you get wrong page numbers
    if [ i=4 ]
    then
      makeindex BuildWithGBP
    fi

    # Create Bibliography
    biber BuildWithGBP
  done
```

⟨*create-book.sh5* 447b⟩

Das EPUB-Format ist ein gezipptes XHTML, angereichert mit Metadaten.

```
447b ⟨create-book.sh5 447b⟩≡ (447a)
  # tex4ebook -f mobi BuildWithGBP.tex
  tex4ebook -f epub BuildWithGBP.tex ../
```

⟨*create-book.sh6* 448⟩

448 `<create-book.sh6 448>≡`

(447b)

```
# For the first translation

for lang in ${LANGS}
do
  cd translation/${lang}/target
  for ((i=4; i>0; i--))
  do
    pdflatex -shell-escape ./BuildWithGBP.tex
    if [ i=4 ]
    then
      makeindex BuildWithGBP
    fi

    # Create Bibliography
    biber BuildWithGBP
  done
  cd ${BasePath}
done

# Remove auxillary *.html files which are needed for epub
if ls | grep --quiet '\.html'
then
  rm *.html
fi
```

50.3. Skript zum Extrahieren der Skripte

Das folgende Skript kann mit

```
notangle -Rcreate-buildscript.sh Part6.nw > create-buildscript.sh &&
t=$(date +%c)
echo "#generated on $t" >> create-buildscript.sh
chmod ugo+x create-buildscript.sh
```

aus der Datei *Part6.nw* extrahiert werden.

```
449 <build-script 449>≡
#!/bin/bash

set -e

scripts="
  build-gbp.sh
  build-gbp-java-plugin.sh
  build-gbp-maven-plugin.sh
  build-gbp-webext-plugin.sh
  build-gbp-python-plugin.sh
"

cat Title.nw Part1.nw Part2.nw Part3.nw Part4.nw Part5.nw Part6.nw > BuildWithGBPg.nw
notangle -Rbuild-gbp.sh BuildWithGBPg.nw > build-gbp.sh &&
#t=$(date +%c)
# For building reproducible
t=$(date --utc --date="@${SOURCE_DATE_EPOCH:-$(date +%s)}" -R)

function build_script()
{
  notangle -R"$script" BuildWithGBPg.nw > "$script"
  sed --in-place \
  --expression='s/This is the end, my friend[[[:cntrl:]]*/This is the end, my friend/' \
  "$1"
  echo "#generated on $t" >> "$1"
  chmod ugo+x "$1"
  bash -n "$1"
}

for script in $scripts; do
  build_script "$script"
done

# Remove auxillary file BuildWithGBPg.nw
rm BuildWithGBPg.nw
```

13. Oktober 2024

Mittels des Befehls *noweave* werden die deutschsprachigen **.nw*-Dateien in **.tex*-Dateien umgewandelt.

```
450a <CreateTexFromNW 450a>≡
function CreateTexFromNW {
    # Called by TaskSelect
    # Create *.tex files from *.nw files

    cd ${SRCDIR}

    noweave -index -delay BuildWithGBP.nw > BuildWithGBP.tex # contains preamble
    noweave -index -delay Title.nw > GBP-Title.tex
    noweave -index -delay Part1.nw > GBP-Part1.tex
    noweave -index -delay Part2.nw > GBP-Part2.tex
    noweave -index -delay Part3.nw > GBP-Part3.tex
    noweave -index -delay Part4.nw > GBP-Part4.tex
    noweave -index -delay Part5.nw > GBP-Part5.tex
    noweave -index -delay Part6.nw > GBP-Part6.tex
<CreateTexFromNW1 450b>
```

Eine eventuell vorhandene Datei *BuildWithGBP_html.tex* wird entfernt.

```
450b <CreateTexFromNW1 450b>≡ (450a)

    if [ -f BuildWithGBP_html.tex ]
    then
        rm BuildWithGBP_html.tex
    fi
}
```


50.4. *gitlab-ci.yml* für die Salsa-CI

```

451 <gitlab-ci.yml 451>≡
  variables:
    DEPS: file make texlive noweb texlive-bibtex-extra texlive-lang-german
          texlive-lang-japanese texlive-latex-extra texlive-binaries
          texlive-extra-utils biber tidy texlive-lang-english

  stages:
    - build
    - deploy

  build:
    stage: build
    image: debian:sid
    before_script:
      - apt -y update
      - apt -y install $DEPS
    script:
      - make
    artifacts:
      paths:
        - '*.pdf'
        - '*.epub'
        - 'build-gbp-python-plugin.sh'
        - 'build-gbp-maven-plugin.sh'
        - 'build-gbp-webext-plugin.sh'
        - 'build-gbp.sh'
        - 'build-gbp-java-plugin.sh'
        - 'translation/en_US/target/*.pdf'

  pages:
    stage: deploy
    image: debian:sid
    needs:
      - build
    script:
      - mkdir -p public/de
      - mkdir -p public/en_US
      - cp *.pdf *.epub build-gbp*.sh public/
      - cp *.tex public/de/
      - cp translation/en_US/target/*.pdf public/en_US/

  artifacts:
    paths:
      - public
  only:
    - master

```

Teil VII.
Anhang

Abbildungsverzeichnis

21.1.	Information zum Java-Team ¹	80
21.2.	Zugangstoken erstellen ²	81
22.1.	Arbeitsabläufe [38] ³	84
31.1.	Startbildschirm	113
31.2.	Angabe des Projektnamens.	114
31.3.	Bye	115
31.4.	Kein Projektname angegeben.	116
32.1.	Keine Konfigurationsdatei gefunden.	118
32.2.	Name des Quellcode-Paketes	120
32.3.	Namen des Quellpaketes angeben	121
32.4.	Korrekten Namen des Quellpaketes angeben	122
32.5.	Ist der Paketname korrekt?	123
32.6.	Ist der Paketname korrekt?	123
32.7.	Korrekten Name des Paketes angegeben.	124
32.8.	Name der Gruppe auf <i>salsa.debian.org</i> angegeben.	125
32.9.	Name der Gruppe auf <i>salsa.debian.org</i> angegeben.	125
32.10.	Name der Gruppe auf <i>salsa.debian.org</i> angegeben.	126
32.11.	Pfad zum Projektverzeichnis auf der lokalen Maschine	127
32.12.	Pfad zum Projektverzeichnis auf der lokalen Maschine mit <i>OrigName</i>	128
32.13.	Pfad zum Projektverzeichnis auf der lokalen Maschine (<i>real</i>)	129
32.14.	Soll ein Java-Paket gebaut werden?	132
32.15.	Soll ein Java-Paket mit <i>maven</i> gebaut werden?	133
32.16.	Soll eine Erweiterung für Mozilla paketiert werden?.	134
32.17.	Soll ein Python3-Pakete gebaut werden?.	135
32.18.	Gibt es ein übergeordnetes Git-Repositoryum?	141
32.19.	Ein neues Paket erstellen	142
32.20.	Das Git-Repositoryum existiert bereits.	144
32.21.	Name und E-Mail.	145
32.22.	Name des Maintainer	149
32.23.	E-Mail des Maintainers	149
32.24.	Debian-Maintainer OK?	150
32.25.	Name des Debian-Maintainer	150
32.26.	E-Mail des Debian-Maintainer	151
32.27.	E-Mail des Debian-Uploaders	152

¹Quelle: <https://salsa.debian.org/java-team/>

²Quelle:<https://salsa.debian.org>

³©2016 Antoine Beaupré anarcat@debian.org, CC-BY-SA 4.0



32.28. E-Mail des Debian-Uploaders	153
32.29. Name und E-Mail des Maintainers korrekt?	154
32.30. Name des Maintainers eingeben	155
32.31. Name des Maintainers erforderlich	156
32.32. E-Mail-Adresse des Maintainers eingeben	157
32.33. Dies ist keine E-Mail-Adresse	158
32.34. Ein Git-Repositorium auf salsa anlegen	158
32.35. Remoteserver ergänzen	159
32.36. Zeigt Liste der Git-Zweige	161
32.37. Zeigt aktuellen Git-Zweig	162
32.38. Aktueller Git-Zweig	163
32.39. Vorgegebener Git-Zweig	163
32.40. Aktueller Git-Zweig	164
32.41. Auswahl des Debian Release.	166
32.42. Name und E-Mail.	167
32.43. Name und E-Mail.	169
32.44. Herunterladen der Debian Sourcen	170
32.45. Eingabe der Url der Debian Sourcen	171
32.46. GPG-Schlüssel verfügbar	173
32.47. Fingerprint eingeben	174
32.48. Ist der Fingerprint korrekt?	174
32.49. Korrekten Fingerprint eingeben	175
32.50. Backup des Fingerprints anlegen	176
33.1. Abfrage - Konfigurationsdatei.	178
33.2. Abfrage - Konfigurationsdatei bearbeiten.	179
33.3. Abfrage - Weiterer Check?.	184
33.4. Auswahl des Debian Zweiges.	187
33.5. Ausgewählter Debian Zweig.	189
33.6. Es gibt nur einen Git-Zweig	190
33.7. Kein Zweig erstellt	191
33.8. Aufgabenauswahl.	192
34.1. Herunterladen von salsa.debian.org	196
34.2. Es gibt Patches	198
34.3. Fixed? Retry?	199
34.4. Kein Import in Patch-Queue	200
34.5. PQ-Import erfolgreich	201
34.6. Kein Herunterladen per uscan von thunderbird.net	203
34.7. Kein Herunterladen per uscan von mozilla.org	204
34.8. Download - klassisch oder mit uscan	205
34.9. Name des Quellcode-Paketes	207
34.10. Herunterladen (oder kopieren)?	207
34.11. Link für Download eingeben	208
34.12. Download-URL korrekt?	209
34.13. Korrekte Download-URL	209

34.14.	*asc Datei herunterladen	210
34.15.	Aufforderung zum Kopieren	211
34.16.	Kopieren erledigt?	211
34.17.	Programm beenden	212
34.18.	Unbekannte Endung	214
34.19.	Ist dies die korrekte Version?	215
34.20.	Welche Version soll gebaut werden?	216
34.21.	Wird korrekte Version gebaut?	216
34.22.	Programm beenden	217
34.23.	Datei debian/copyright enthält Abschnitt Files-Excluded	218
34.24.	Dateien ausschließen	219
34.25.	Soll debian/copyright editiert werden?	220
34.26.	Suffix für den Ausschluss von Dateien	222
34.27.	Eigener Suffix für den Ausschluss von Dateien	223
34.28.	Warnung! - Kein Suffix angegeben	224
34.29.	Zusätzliche Optionen für <i>mk-origtargz</i>	225
34.30.	<i>debian/copyright</i> enthält auszuschließende Dateien	226
34.31.	Keine Versionsnummer – kein <i>mk-origtargz</i>	227
34.32.	<i>mk-origtargz</i> gescheitert	227
34.33.	Spezielle <i>gbp.conf</i>	232
34.34.	Abfrage: Pfad zur <i>gbp.conf</i>	233
34.35.	<i>gbp.conf</i> nicht gefunden	234
34.36.	Check <i>gbp.conf</i>	235
34.37.	Check <i>gbp.conf</i>	236
34.38.	<i>gbp.conf</i> zweimal gefunden.	238
34.39.	Unterschiedliche Konfigurationsdateien	239
34.40.	<i>gbp.conf</i> im Verzeichnis <i>debian/</i> bearbeiten?	240
34.41.	<i>gbp.conf</i> im Verzeichnis <i>.git/</i> bearbeiten?	241
34.42.	Zweifelhafter Git-Tag	242
34.43.	Git Tags entfernen	243
34.44.	Der Zweig is	245
34.45.	Importiert wurde der Upstream-Code	246
34.46.	Up to date	249
34.47.	Neue Version verfügbar	249
34.48.	Kein Repack Suffix in <i>debian/watch</i>	250
34.49.	Uscaan kann Watch-Datei nicht parsen	252
35.1.	Neue Revision bauen	253
35.2.	Daten für Maven erstellen?	256
35.3.	Debian-Dateien anzeigen	257
35.4.	Art des Quellpaketes (Nativ? J/N)	261
36.1.	Es gibt ein Verzeichnis <i>debian/patches</i>	286
36.2.	Es gibt kein Verzeichnis <i>debian/patches</i>	286
36.3.	Patches für Debian erstellen	287
36.4.	Es existiert ein PQ-Zweig.	289

36.5.	Alle Patches müssen anwendbar sein.	291
36.6.	Soll <i>debian/patches/series</i> editiert werden?.	292
36.7.	Der Patch-Queue-Zweig mit Patches von <i>debian/patches</i> wurde ange- wendet.[3]	295
36.8.	Soll <i>gbp pq</i> genutzt werden?	296
36.9.	Unterbrechung zum Patchen	297
36.10.	Soll <i>gbp pq export</i> angewandt werden?	298
36.11.	<i>dquilt</i> ist nicht konfiguriert.	303
36.12.	Wird ein Patch benötigt?	304
36.13.	Aufgaben für <i>Quilt</i>	305
36.14.	Ein weiterer Patch?	307
36.15.	Name des Patches	308
36.16.	Patch existiert bereits!	309
36.17.	Eine weitere Datei patchen?	311
36.18.	Dateiauswahldialog	312
36.19.	Dateiauswahl bestätigen	313
37.1.	Debian-Changelog OK?	322
37.2.	Anzeige der ersten Zeile der Datei <i>debian/changelog</i>	324
37.3.	Aktuelle Version	324
37.4.	Keine Revisionsnummer angegeben	325
37.5.	Korrekte Versionsbezeichnung angeben?	326
37.6.	Korrekte Versionsbezeichnung angeben?	327
37.7.	Abfrage des Bezeichners eines nativen Paketes	328
37.8.	Abfrage des Bezeichners	329
37.9.	Weitere Optionen für <i>dch</i>	330
37.10.	Da läuft was falsch!	333
37.11.	Auswahl des Branches	334
37.12.	Anzeige des Git-Zweiges	335
37.13.	Release-Branch der Distribution	336
37.14.	Auswahl des Build-Systems	338
37.15.	Parameterüberprüfung	339
37.16.	Bau des Paketes starten	340
37.17.	Beenden	340
37.18.	Information zur Aktualisierung der Build-Umgebung	341
37.19.	<i>.sbuildrc</i> prüfen	342
37.20.	Auswahl der zu erstellenden Cow.	344
37.21.	Anzeige der Version mit Revisionsnummer	347
37.22.	Anzeige der Revisionsnummer	348
37.23.	Soll der Upstream-Tarball auch hochgeladen werden	349
37.24.	Anzeige der Optionen von <i>gbp buildbackage</i>	350
37.25.	Erfolgloser Bauversuch!	352
40.1.	Upload des Release vorbereiten	358
40.2.	*.changes-Datei auswählen	360
40.3.	Lintian: All Well?	362

40.4.	Paket ist aktuell	363
40.5.	Älteres Paket verfügbar	364
40.6.	Uscan - OK?	364
40.7.	Uscan schlägt fehl	365
40.8.	Unterschiede feststellen	366
40.9.	Zu viele Versionen ausgewählt	367
40.10.	Zu wenig ausgewählt	368
40.11.	Umkehr der Reihenfolge der zu vergleichenden Pakete	369
41.1.	Kein Changelog - kein Hochladen	374
41.2.	Changelog veröffentlichungsreif?	375
41.3.	Branch überprüfen	376
41.4.	Name der Distribution eingeben	377
41.5.	Name der Distribution überprüfen	378
41.6.	Fürs Release bauen	379
41.7.	Fürs Release bauen	380
41.8.	DebDiff benötigt?	381
42.1.	Fürs Release bauen	384
43.1.	Ziel des Uploads.	388
43.2.	Ist das angegebene Ziel des Uploads korrekt?	390
43.3.	Bye	390
43.4.	Signieren erfolgreich?	392
43.5.	Upload to FTP-Master - OK?	393
43.6.	Soll ein Binär-Paket auf FTP-Master hochgeladen werden?	394
43.7.	Wirklich nach Experimental hochladen?	395
43.8.	Soll das Hochladen auf FTP-Master simuliert werden?	396
43.9.	Ist alles ok mit dem Hochladen?	397
43.10.	Tage der Verzögerung	399
47.1.	Maven Plugin geladen	420
47.2.	Pfad zur Maven-Chroot ermitteln	421
47.3.	Pfad zur Maven-Chroot angeben	422
47.4.	Arbeitsverzeichnis in der Maven-Chroot ermitteln	423
47.5.	Arbeitsverzeichnis in der Maven-Chroot angeben	424
47.6.	Maven-Chroot existiert nicht	425
48.1.	Webext-Plugin wurde geladen	438
49.1.	Python-Plugin geladen	440

Literaturverzeichnis

- [1] Creative Commons. „Attribution-ShareAlike 4.0 International“. In: *https://creativecommons.org/licenses/by-sa/4.0/legalcode*. (15. Okt. 2013). URL: <https://creativecommons.org/licenses/by-sa/4.0/legalcode>. (Siehe S. 3). 
- [2] Free Software Foundation, Inc. „GNU General Public License 3“. In: *https://www.gnu.org/licenses/gpl-3.0.de.html*. (29. Juni 2007). URL: <https://www.gnu.org/licenses/gpl-3.0.de.html>. (Siehe S. 3). 
- [3] Guido Günther. „Building Debian Packages with git-buildpackage“. In: *Git-Buildpackage* (2017). URL: <https://honk.sigxcpu.org/projects/git-buildpackage/manual.html>// (siehe S. 7, 22, 33, 76, 294, 295, 298).
- [4] Norman Ramsey. „Noweb — A Simple, Extensible Tool for Literate Programming“. In: *No Web* (28. Juni 2018). URL: <https://www.cs.tufts.edu/~5Ctextasciitilde%20nr/noweb/> (siehe S. 10).
- [5] *Simple Packaging Tutorial*. 26. Okt. 2019. URL: <https://wiki.debian.org/SimplePackagingTutorial> (siehe S. 21, 23).
- [6] Debian Project. „Die Debian-Richtlinien für Freie Software (DFSG)“. German. In: *Debian-Gesellschaftsvertrag* (26. Apr. 2004). URL: https://www.debian.org/social_contract.de.html (siehe S. 21, 27, 31).
- [7] Ian Jackson, Christian Schwarz und David A. Morris. „Debian Policy“. English. Version 4.6.1. In: *Debian Policy Manual* (22. Mai 2022). Hrsg. von Die Debian-Policy-Gruppe. URL: <https://www.debian.org/doc/debian-policy/>. GNU General Public License Version 2+ (siehe S. 21, 23, 27, 30, 31, 53, 60, 260, 265, 273, 280, 282, 412).
- [8] Christopher Yeoh, Paul 'Rusty' Russell, Daniel Quinlan. „Filesystem Hierarchy Standard“. English. Version 3.0. In: *Filesystem Hierarchy Standard* (19. März 2015). Hrsg. von The Linux Foundation LSB Workgroup. URL: <https://www.debian.org/doc/packaging-manuals/fhs/fhs-3.0.html>. BSD (siehe S. 21).
- [9] Ian Jackson u. a. „Debian-Entwicklerreferenz“. Deutsch. Version 13.3. In: *Debian Entwicklerreferenz* (5. Aug. 2023). Hrsg. von Hideki Nussbaum Lucas and Yamane und Holger Levsen. URL: <https://www.debian.org/doc/manuals/developers-reference/index.de.html>. GNU General Public License Version 2+ (siehe S. 21, 22, 83, 84, 86, 87, 98, 99).
- [10] Osamu Aoki. „Leitfaden für Debian-Betreuer. debmake-doc“. Deutsch. In: *Debmake-Doc* (26. März 2019). URL: <https://www.debian.org/doc/devel-manuals#debmake-doc>. Expat-Lizenz (siehe S. 22).

- [11] Josip Rodin, Osamu Aoki. „Debian-Leitfaden für Neue Paketbetreuer. New Maintainer Guide“. Deutsch. Version 1.2.43. In: *Debian-Leitfaden für Neue Paketbetreuer* (7. Nov. 2020). Hrsg. von Osamu Aoki. wird ersetzt durch Aoki, Leitfaden für Debian-Betreuer. URL: <https://www.debian.org/doc/manuals/maint-guide/>. GNU General Public License Version 2+ (siehe S. 22, 36, 62, 260, 323).
- [12] Axel Beckert und Frank Hofmann. „Debian-Paketmanagement“. In: *DPMB* (7. Feb. 2021), S. 400. URL: <https://book.dpmb.org/debian-paketmanagement.chunked/index.html> (siehe S. 22, 24, 25).
- [13] wiki.debian.org, Hrsg. *What is a "package"?* 23. Dez. 2020. URL: <https://wiki.debian.org/Packaging/Intro> (siehe S. 23).
- [14] *ReproducibleBuilds*. 6. Dez. 2020. URL: <https://reproducible-builds.org/> (siehe S. 23).
- [15] *The experimental repository*. 15. Nov. 2020. URL: <https://wiki.debian.org/DebianExperimental> (siehe S. 24).
- [16] *The Debian GNU/Linux FAQ*. 12. Aug. 2019. URL: <https://www.debian.org/doc/manuals/debian-faq/index.de.html> (siehe S. 24, 35).
- [17] *How can i help*. 7. Feb. 2021. URL: <https://wiki.debian.org/how-can-i-help> (siehe S. 25).
- [18] *Teams*. 10. Jan. 2024 (siehe S. 25, 147).
- [19] Steve Langasek. „DEP-5: Machine-readable debian/copyright“. In: *Machine-readable debian/copyright* (24. Feb. 2012). URL: <https://dep-team.pages.debian.net/deps/dep5/> (siehe S. 27, 28, 31, 32, 220, 264).
- [20] *TeamsFTPMaster*. 13. März 2020. URL: <https://wiki.debian.org/TeamsFTPMaster> (siehe S. 27).
- [21] *CopyrightReviewTools*. 17. Dez. 2021. URL: <https://wiki.debian.org/CopyrightReviewTools> (siehe S. 27).
- [22] Mathew Palmer. „Debian-Mentors FAQ“. English. In: *Debian-Wiki* (2007). URL: <https://wiki.debian.org/DebianMentorsFaq>. GNU General Public License version 2 (siehe S. 29, 33).
- [23] Jilayne et al. Lovejoy. „Open Source License Compliance Handbook“. In: *Open Source License* (29. Apr. 2019). URL: <https://github.com/finos/OSLC-handbook/tree/master/output> (siehe S. 29).
- [24] *Using Quilt*. 22. Juli 2020. URL: <https://wiki.debian.org/UsingQuilt> (siehe S. 33).
- [25] Andreas Grünbacher. „How to Survive With Many Patches or Introduction to Quilt“. In: *Introduction to Quilt* (22. Feb. 2012). URL: <http://users.suse.com/~agruen/quilt.pdf> (siehe S. 33).
- [26] Raphael Hertzog. „DEP-3: Patch Tagging Guidelines“. In: *Patch Tagging Guidelines* (26. Nov. 2009). URL: <https://dep-team.pages.debian.net/deps/dep3/> (siehe S. 34, 297).

- [27] *Git-Mailinfo -Manpage*. 20. Apr. 2020. URL: <https://manpages.debian.org/unstable/git-man/git-mailinfo.1.en.html> (siehe S. 34).
- [28] Niels Thykier u. a. „Debian Policy for Java“. In: *Debian Java Policy* (27. Juli 2020). URL: <https://www.debian.org/doc/packaging-manuals/java-policy/> (siehe S. 41, 42, 103, 279).
- [29] Markus Koschany. „Packaging Java with Javatools“. In: <https://people.debian.org/~apo/java/tutorial.html> (2. Aug. 2018). URL: <https://people.debian.org/%5Ctextasciitilde%7B%7Dapo/java/tutorial.html> (siehe S. 41).
- [30] Torsten Werner twerner@debian.org Niels Thykier niels@thykier.net Javier Fernández-Sanguino Peña jfs@debian.org Sylvestre Ledru sylvestre@debian.org. „Debian Java FAQ.“ In: *Debian Java FAQ*. (22. Mai 2014). URL: <https://www.debian.org/doc/manuals/debian-java-faq/> (siehe S. 41).
- [31] *Help the Java Team distribute your project*. 31. Jan. 2019. URL: <https://java.debian.net/blog/2019/01/help-the-java-team-distribute-your-project.html> (siehe S. 41).
- [32] *Introduction to the Standard Directory Layout*. 29. Dez. 2020. URL: <https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html> (siehe S. 43).
- [33] *Devsripts*. 28. Nov. 2020. URL: <https://manpages.debian.org/unstable/devscripts/devscripts.1.en.html> (siehe S. 59).
- [34] *SBuild*. 15. Mai 2024. URL: <https://wiki.debian.org/sbuild> (siehe S. 71, 341).
- [35] *debian mit debootstrap in chroot-Umgebung installieren*. 16. Feb. 2014. URL: http://www.kai-hildebrandt.de/linux/debian%5C_chroot.html (siehe S. 72).
- [36] *Salsa*. 22. Feb. 2023. URL: <https://wiki.debian.org/Salsa> (siehe S. 79).
- [37] *Salsa-Doc*. 26. Aug. 2023. URL: wiki.debian.org/Salsa/Doc (siehe S. 79).
- [38] [wiki.debian.org](https://wiki.debian.org/DebianReleases). „Debian Releases“. In: <https://wiki.debian.org/DebianReleases>. 29. Nov. 2020. URL: <https://salsa.debian.org/debian/package-cycle/raw/master/package-cycle.svg> (siehe S. 84).
- [39] *ReleaseTeam*. 22. März 2021. URL: <https://wiki.debian.org/Teams/releaseTeams> (siehe S. 97).
- [40] Debian, Hrsg. *Lintian User's Manual*. 23. Jan. 2023. URL: <https://lintian.debian.org/manual/index.html> (siehe S. 103).
- [41] *GBP-Import-orig uscan*. 19. Juli 2022. URL: <https://manpages.debian.org/unstable/git-buildpackage/gbp-import-orig.1.en.html> (siehe S. 251).
- [42] wiki.debian.org. „Upstream METadata GAttered with YAml (UMEGAYA)“. In: <https://wiki.debian.org/UpstreamMetadata?action=recall&rev=138>. 31. Jan. 2020. URL: <https://wiki.debian.org/UpstreamMetadata?action=recall&rev=138> (siehe S. 262).
- [43] Charles Plessy und Andreas Tille. „DEP-12: Per-package machine-readable metadata about Upstream“. In: *Per-package machine-readable metadata about Upstream* (23. Feb. 2014). URL: <https://dep-team.pages.debian.net/deps/dep12/> (siehe S. 262).

- [44] Machine-Readable Copyright. „Machine-readable debian/copyright file“. In: *Debian Policy* (17. Nov. 2020). URL: <https://www.debian.org/doc/packaging-manuals/copyright-format/1.0/#stand-alone-license-paragraph> (siehe S. 265).
- [45] Debian Projekt, Hrsg. *Manpage uscan*. 4. Aug. 2019. URL: <https://manpages.debian.org/buster/devscripts/uscan.1.en.html> (siehe S. 269).
- [46] *GBP-PQ*. 1. Juli 2020. URL: <https://manpages.debian.org/unstable/git-buildpackage/gbp-pq.1.en.html> (siehe S. 293).

Stichwortverzeichnis

- LaTeX, 10
- TeX-Dateien, 450
- ~/ .bashrc, 62, 73

- Abhängigkeit, 30, 41
- Aktualisierung, 49
- ant, 47
- apt, 36

- Build-Umgebungen, 33

- Chroot, 63
- Chroot (maven), 44
- cme, 29
- compare-version, 35
- Copyright-Review, 27
- cowbuilder, 62
- cowdancer, 62
- Creative Commons, 3

- DEBEMAIL, 62, 145
- DEBFULLNAME, 62, 145
- debhelper, 22, 265
- Debian-Entwicklerreferenz, 21
- Debian-Keyring, 173
- Debian-Paket, 23
- Debian-Policy, 21, 27, 31
- Debian-Projekt, 23
- debian/watch, 248, 269
- debmake, 27
- devscripts, 36, 59
- DFSG, 27, 31
- dh_make, 39, 59
- Distribution, 9
- dpkg, 36
- dquilt, 72
- dversionmangle, 271

- E-Book, 10

- Entwicklerreferenz, 21
- EPUB-Dokument, 10
- experimental, 24

- Fehlerbehebung, 33
- FHS, 21
- filenamemangle, 272
- Fingerprint, 62, 173
- Freie Software, 27
- FTBFS, 63

- gbp buildpackage, 62
- gbp pq, 33
- gbp.conf, 75, 77, 248
- Geany, 11
- Gesellschaftsvertrag, 21
- git, 11
- git-buildpackage, 11, 22, 59
- Git-Repository, 144
- GNU General Public License, 3
- GPG-Schlüssel, 59, 246, 251

- Handbuch, 22
- Hauptprogramm, 112
- Hilfsprogramme, 39
- Hochladen, 373
- Homeverzeichnis, 117

- Java-Anwendung, 41
- Java-Bibliothek, 41, 42
- Java-Build-System, 43
- Java-FAQ, 41
- Java-Policy, 41
- Java-Programm, 42
- javahelper, 41

- Konfigurationsdatei, 61, 111, 116, 173, 177
- Kurzanleitung, 15

Leitfaden f. neue Betreuer, 22
Literate Programming, 10
Literatur, 22
Lizenz, 3, 27
Lizenzen, 27, 29
Lizenzprüfung, 27, 365
Log-Datei, 138

Mail-Extension, 50
main, 27
Maintainer, 147
Maintainer-Schlüssel, 62
Maven, 43, 44
maven, 47
mh_make, 44

noweave, 450
noweb, 10

oldoldstable, 24
oldstable, 24
Optionen (uscan), 270
Original-Autor, 29
Originalquelle, 36

Paketieren, 7
Paketmanagementsystem, 49
Paketverwaltungswerkzeug, 265
Patches, 33
pbuilder, 63
PDF-Dokument, 10
Perl-Format, 269
pom.xml, 43
pristine-tar, 168
Programmablauf, 111, 112
Programmskript, 111
Projekt, 113
Projektname, 114
projektspezifisch, 61
Proposed-Updates, 84, 86

Quelltext, 10
quilt, 33, 72

reguläre Ausdrücke, 269
Release-Team, 86
Reproduzierbarkeit, 23
Revisionsnummer, 35

Rückportierung, 84

Security-Patches, 33, 84
security-Team, 84
signieren, 62
stable, 24
Standard-Version, 266
Startdialog, 111
Systemnutzer, 49

testing, 24
Thunderbird, 50

unstable, 24
Uploader, 147
uscan, 36, 248, 269
uversionmangle, 271

Vergleichsregeln, 36
Versionierungsindex, 271
Versionierungsschema, 35, 214
Versionsbezeichnung, 35, 221
Verzeichnis, 138
Verzeichnisstruktur Maven, 43
Veröffentlichung, 10, 194

Watch (Datei), 36, 269
Werkzeuge, 10

Zugangstoken, 80, 443