Debian Pakete bauen mit Git

Eine kurze Einführung

Guido Günther <*agx@sigxcpu.org*>

Hackin' Erpel 0, Erpel

”Routineaufgaben” beim Debian Paketbau

1. Neues Debian Paket erstellen
2. Neue Upstream Version importieren
3. Ein bestehendes Debian Paket importieren
4. Ändern, bauen, testen, ändern, bauen, testen
5. Der spannende Teil. . .

Die Akteure

- *dh* - erledigt Standardtasks automatisch, flexibel bei Besonderheiten
- *git* - kann Upstream speichern, Merges und Cherry-Picks
- *gbp* - Schnittstelle zwischen Debian Paketierung und Git
- *pbuilder,cowbuilder* - Bauen in einer sauberen Umgebung mit minimalen Abhängigkeiten
- (*pristine-tar* - Reproduzieren des exakten Upstream Tarballs)

Upstream Tarball importieren

- Leeres Git Repository erstellen

```
git init python-dateutil
cd $_
```

- Tarball importieren

```
gbp import-orig \
    ../python-dateutil_1.4.1.orig.tar.gz
```

debian/ erstellen

Das gehört zur Paketierung:

- *rules*: Bauanleitung um Binär-Paket zu bauen
- *control*: Meta-Informationen
- *changelog*: Historie
- *copyright*: Lizenz-Information
- *{pre,post}{inst,rm}*: Maintainer-Skripte
- Unterstützer: watch, gbp.conf, Dateien für dh_*-Helfer

Debhelper

Nimmt einem die ganze Arbeit in *debian/rules* ab:

- Bauen (*dh_auto_**)
- Integration in das System
  - Dateien installieren, Verzeichnisse anlegen
  - Init-Skripte
  - Manpages
  - Shared Libraries
  - Menu Dateien
  - Icons
  - Logrotate
  - MD5 Prüfsummen
- dh_*: $\rightarrow$ debian/{install,dirs,docs,...}

Debhelper

debian/rules:

```
#!/usr/bin/make -f
%:
	dh $@
```

Debhelper

debian/rules:

```
#!/usr/bin/make -f
%:
	dh $@

override_dh_auto_make:
	./autogen.sh
	dh_auto_make

override_dh_install:
	dh_install --sourcedir=weirdsourcedir/
```

dh-make

Erstellt ein Gerüst für *debian/*:

- dh_make

  ```
  dh_make -p python-dateutil_1.4.1
  ```

- Aber: *.ex *.EX

Das erste Mal bauen

```
git add debian
git commit -m"Add Debian packaging"
gbp buildpackage
```

Tarball importieren

- Entweder wie oben
- Oder Import from upstream URL:

```
gbp import-orig --uscan
```

Tarball importieren

- Entweder wie oben
- Oder Import from upstream URL:

```
gbp import-orig --uscan
```

Importiert Upstream Sourcen auf *upstream* Branch, Debian Paketierung erfolgt auf *master*.

Wie sieht das Repository aus?

Commits, Tags, Branches

```
* 23667ff (HEAD, master) Bump standards version
*   eef58a9 Merge commit 'upstream/1.5'
|\
| * 46ca33a (tag: upstream/1.5, upstream) Imported Upstream version 1.5
* | 23396fc Document changes and release 1.4.1-2
* | e181ff2 Remove comment
* | 539f202 (tag: debian/1.4.1-1) Imported Debian patch 1.4.1-1
|/
* 4e5dda0 (tag: upstream/1.4.1) Imported Upstream version 1.4.1
```

- Upstream branch: upstream

Wie sieht das Repository aus?

Commits, Tags, Branches

```
* 23667ff (HEAD, master) Bump standards version
*   eef58a9 Merge commit 'upstream/1.5'
|\
| * 46ca33a (tag: upstream/1.5, upstream) Imported Upstream version 1.5
* | 23396fc Document changes and release 1.4.1-2
* | e181ff2 Remove comment
* | 539f202 (tag: debian/1.4.1-1) Imported Debian patch 1.4.1-1
|/
* 4e5dda0 (tag: upstream/1.4.1) Imported Upstream version 1.4.1
```

- Upstream branch: upstream
- Debian branch: master

Wie sieht das Repository aus?

Commits, Tags, Branches

```
* 23667ff (HEAD, master) Bump standards version
*   eef58a9 Merge commit 'upstream/1.5'
|\
| * 46ca33a (tag: upstream/1.5, upstream) Imported Upstream version 1.5
* | 23396fc Document changes and release 1.4.1-2
* | e181ff2 Remove comment
* | 539f202 (tag: debian/1.4.1-1) Imported Debian patch 1.4.1-1
|/
* 4e5dda0 (tag: upstream/1.4.1) Imported Upstream version 1.4.1
```

- Upstream branch: upstream
- Debian branch: master
- Upstream tag: upstream/<version>

Wie sieht das Repository aus?

Commits, Tags, Branches

```
* 23667ff (HEAD, master) Bump standards version
*   eef58a9 Merge commit 'upstream/1.5'
|\
| * 46ca33a (tag: upstream/1.5, upstream) Imported Upstream version 1.5
* | 23396fc Document changes and release 1.4.1-2
* | e181ff2 Remove comment
* | 539f202 (tag: debian/1.4.1-1) Imported Debian patch 1.4.1-1
|/
* 4e5dda0 (tag: upstream/1.4.1) Imported Upstream version 1.4.1
```

- Upstream branch: upstream
- Debian branch: master
- Upstream tag: upstream/<version>
- Debian tag: debian/<debian-version>

Wie sieht das Repository aus?

Commits, Tags, Branches

```
* 23667ff (HEAD, master) Bump standards version
*   eef58a9 Merge commit 'upstream/1.5'
|\
| * 46ca33a (tag: upstream/1.5, upstream) Imported Upstream version 1.5
* | 23396fc Document changes and release 1.4.1-2
* | e181ff2 Remove comment
* | 539f202 (tag: debian/1.4.1-1) Imported Debian patch 1.4.1-1
|/
* 4e5dda0 (tag: upstream/1.4.1) Imported Upstream version 1.4.1
```

- Upstream branch: upstream
- Debian branch: master
- Upstream tag: upstream/<version>
- Debian tag: debian/<debian-version>
- Das sind die Defaults.

Wie sieht das Repository aus?

Commits, Tags, Branches

```
* 23667ff (HEAD, master) Bump standards version
*   eef58a9 Merge commit 'upstream/1.5'
|\
| * 46ca33a (tag: upstream/1.5, upstream) Imported Upstream version 1.5
* | 23396fc Document changes and release 1.4.1-2
* | e181ff2 Remove comment
* | 539f202 (tag: debian/1.4.1-1) Imported Debian patch 1.4.1-1
|/
* 4e5dda0 (tag: upstream/1.4.1) Imported Upstream version 1.4.1
```

- Upstream branch: upstream
- Debian branch: master
- Upstream tag: upstream/<version>
- Debian tag: debian/<debian-version>
- Das sind die Defaults.
- Patches: patch-queue/master

Wie sieht das Repository aus?

Commits, Tags, Branches

```
* 23667ff (HEAD, master) Bump standards version
*   eef58a9 Merge commit 'upstream/1.5'
|\
| * 46ca33a (tag: upstream/1.5, upstream) Imported Upstream version 1.5
* | 23396fc Document changes and release 1.4.1-2
* | e181ff2 Remove comment
* | 539f202 (tag: debian/1.4.1-1) Imported Debian patch 1.4.1-1
|/
* 4e5dda0 (tag: upstream/1.4.1) Imported Upstream version 1.4.1
```

- Upstream branch: upstream
- Debian branch: master
- Upstream tag: upstream/<version>
- Debian tag: debian/<debian-version>
- Das sind die Defaults.
- Patches: patch-queue/master
- Binary delta: pristine-tar

Changelog aktualisieren

Changelog Schreiben ist mühsam, wir haben ja schon eine Git Commit-Historie:

```
gbp dch --snapshot --auto
gbp dch --release --auto
```

Importieren I

- ...aus Dateisystem

  ```
  gbp import-dsc python-dateutil_1.4.1-1.dsc
  ```

- ...von URL

  ```
  gbp import-dsc --download http://....dsc
  ```

- ...via apt

  ```
  gbp import-dsc --download python-dateutil
  ```

Importieren I

- ...aus Dateisystem

  ```
  gbp import-dsc python-dateutil_1.4.1-1.dsc
  ```

- ...von URL

  ```
  gbp import-dsc --download http://....dsc
  ```

- ...via apt

  ```
  gbp import-dsc --download python-dateutil
  ```

- geht auch inkrementell

Importing II

- . . . alle alten Versionen auf einen Rutsch:

```
gbp import-dscs *.dsc
```

- . . . oder von snapshotes.debian.org:

```
gbp import-dscs --debsnap \
    python-dateutil
```

Typischer Ablauf

```
gbp dch --snapshot --auto
gbp buildpackage --git-ignore-new
```

Typischer Ablauf

```
gbp dch --snapshot --auto
gbp buildpackage --git-ignore-new

# ...  rewind and repeat, oder
# git commit, reset, amend
```

Typischer Ablauf

```
gbp dch --snapshot --auto
gbp buildpackage --git-ignore-new

# ...  rewind and repeat, oder
# git commit, reset, amend

# Dann
gbp dch --release --auto
gbp buildpackage --git-tag
```

Reproduzierbar bauen

- pbuilder/cowbuilder erzeugen ein Chroot in dem Pakete gebaut werden können

```
git-pbuilder create # einmalig
```

Reproduzierbar bauen

- pbuilder/cowbuilder erzeugen ein Chroot in dem Pakete gebaut werden können

```
git-pbuilder create # einmalig
```

- ...das kann *gbp buildpackage* benutzen:

```
gbp buildpackage --git-pbuilder
```

Reproduzierbar bauen

- pbuilder/cowbuilder erzeugen ein Chroot in dem Pakete gebaut werden können

```
git-pbuilder create # einmalig
```

- ...das kann *gbp buildpackage* benutzen:

```
gbp buildpackage --git-pbuilder
```

- ...oder für eine andere Release

```
DIST=wheezy git-pbuilder create # einmalig
gbp buildpackage --git-pbuilder \
    --git-dist=wheezy
```

Reproduzierbar bauen

- pbuilder/cowbuilder erzeugen ein Chroot in dem Pakete gebaut werden können

```
git-pbuilder create # einmalig
```

- ...das kann *gbp buildpackage* benutzen:

```
gbp buildpackage --git-pbuilder
```

- ...oder für eine andere Release

```
DIST=wheezy git-pbuilder create # einmalig
gbp buildpackage --git-pbuilder \
    --git-dist=wheezy
```

- Aktualisieren

```
git-pbuilder update
```

CI eines Paketes mit Jenkins

Warum?

CI eines Paketes mit Jenkins

Warum?

- Unit-Tests regelmässig ausführen
- Testbarer Snapshot für User immer verfügbar
- CPU-Leistung/Plattenplatz von Build-Slaves Nutzen
- Builds und Tests für verschiedene Architekturen

CI eines Paketes mit Jenkins

Warum?

- Unit-Tests regelmässig ausführen
- Testbarer Snapshot für User immer verfügbar
- CPU-Leistung/Plattenplatz von Build-Slaves Nutzen
- Builds und Tests für verschiedene Architekturen
  - Jenkins Git Plugin
  - /u/s/d/git-buildpackage/example/jenkins-scratchbuilder

CI eines Paketes mit Jenkins

Warum?

- Unit-Tests regelmässig ausführen
- Testbarer Snapshot für User immer verfügbar
- CPU-Leistung/Plattenplatz von Build-Slaves Nutzen
- Builds und Tests für verschiedene Architekturen
  - Jenkins Git Plugin
  - /u/s/d/git-buildpackage/example/jenkins-scratchbuilder
- TODO: Jenkins Job Template

Mehrere Branches für unterschiedliche Releases

Sid,

Mehrere Branches für unterschiedliche Releases

Sid, Experimental,

Mehrere Branches für unterschiedliche Releases

Sid, Experimental, Stable Point Releases,

Mehrere Branches für unterschiedliche Releases

Sid, Experimental, Stable Point Releases, Backports,

Mehrere Branches für unterschiedliche Releases

Sid, Experimental, Stable Point Releases, Backports, Security Upadates, ... - Wie sieht das Layout in Git aus?

Mehrere Branches für unterschiedliche Releases

Sid, Experimental, Stable Point Releases, Backports, Security Upadates, ... - Wie sieht das Layout in Git aus?
Empfehlung:

- debian/<release> $\rightarrow$ upstream/<release>
- Backports: backports/<release>
- Security Updates: security/<release>

Quilt Patches

- Patches: *debian/patches/*.{patch,diff}*
- Apply: debian/patches/series
- Source format 3.0 (quilt) ist sehr verbreitet

Quilt Patches

- Patches: *debian/patches/*.{patch,diff}*
- Apply: debian/patches/series
- Source format 3.0 (quilt) ist sehr verbreitet

- Patches importieren:

  `gbp pq import`
- Patches exportieren:

  `gbp pq export`
- Patches aktualisieren:

  `gbp pq rebase`
- Patch-queue Branch loswerden:

  `gbp pq drop`

Quilt Patches

- Patches: *debian/patches/*.{patch,diff}*
- Apply: debian/patches/series
- Source format 3.0 (quilt) ist sehr verbreitet

- Patches importieren:

  `gbp pq import`
- Patches exportieren:

  `gbp pq export`
- Patches aktualisieren:

  `gbp pq rebase`
- Patch-queue Branch loswerden:

  `gbp pq drop`

Einfaches Patch System - Alternativen: TopGit, TNT

Remote Repositories

Remote repositories erzeugen und Upstream und Debian Branch dort hin pushen:

```
gbp create-remote-repo \
    --remote-url-pattern=<url-pattern>
```

Git-buildpackage Hooks

git-buildpackage kann durch Hooks erweitert werden:

- postbuild: –git-postbuild z.B. um Lintian aufzurufen:
  `--git-postbuild='lintian $GBP_CHANGES_FILE'`
- posttag: –git-posttag z.B. */u/s/d/git-buildpackage/examples/gbp-posttag-push*

Changelogs im Detail

- *gbp dch* findet neue Upstream Versionen automatisch
- Snapshots verwenden bis zur Release
- Tags in Commit Messages vermeiden doppelte Buchführung
  - Git-Dch: {Ignore, Full, Topic}
  - Closes:
  - Thanks:

More on gbp dch

Commit-Message:

```
commit a7fe7c4678a48072c11bb57fd1f99ca7b8118158
Author: Guido Günther <agx@sigxcpu.org>
Date:   Thu Jan 28 19:03:33 2010 +0100

    Add basic bash completion for git-buildpackage

    Thanks: Siegfried-Angel Gevatter
    Closes: #567313
```

debian/changelog:

```
  * [a7fe7c4] Add basic bash completion for git-buildpackage (Closes: #567313)
    - thanks to Siegfried-Angel Gevatter
```

cl2vcs

Changelogs mit Historie verknüpfen:

- –id-length: Changelog entry $\leftrightarrow$ Commit
- Über den Vcs-Git: link in *debian/control* finden wir das Debian Git Repo

cl2vcs

Changelogs mit Historie verknüpfen:

- –id-length: Changelog entry $\leftrightarrow$ Commit
- Über den Vcs-Git: link in *debian/control* finden wir das Debian Git Repo
- $\rightarrow$ Also kann man die Histories browsen ohne zu clonen:

`https://honk.sigxcpu.org/cl2vcs/?pkg=libvirt`

Upstream benutzt git

- Upstream clonen und Debian Paketierbranch einführen:
  - –upstream-branch=master
  - –debian-branch=debian/<release>
  - –upstream-tag='v%(version)s'
- Wenn man den Upstream Tarball 1:1 verwenden will: *–upstream-vcs-tag*

Konfiguration

- */etc/git-buildpackage/gbp.conf*
- ~/.gbp.conf
- (<repo>/.git/gbp.conf)
- <repo>/debian/gbp.conf

Konfiguration

- */etc/git-buildpackage/gbp.conf*
- ~/.gbp.conf
- (<repo>/.git/gbp.conf)
- <repo>/debian/gbp.conf

- Tag- and branch Namensmuster festlegen
- Tags signieren
- pristine-tar benutzen
- Build Kommands (z.B. git-pbuilder)
- Source Tree exportieren (ähnlich svn-buildpackage)
- Tarball Kompression
- Import Filter
- Hooks

Konfiguration

- */etc/git-buildpackage/gbp.conf*
- ~/.gbp.conf
- (<repo>/.git/gbp.conf)
- <repo>/debian/gbp.conf

- Tag- and branch Namensmuster festlegen
- Tags signieren
- pristine-tar benutzen
- Build Kommands (z.B. git-pbuilder)
- Source Tree exportieren (ähnlich svn-buildpackage)
- Tarball Kompression
- Import Filter
- Hooks

```
gbp <command> --help
```

shows the current defaults.

git-buildpackage

- Wiki:
  `https://honk.sigxcpu.org/piki/projects/git-buildpackage/`
- Manual:
  `http://honk.sigxcpu.org/projects/git-buildpackage/manual-html/gbp.html`

git-buildpackage

- Wiki:
  `https://honk.sigxcpu.org/piki/projects/git-buildpackage/`
- Manual:
  `http://honk.sigxcpu.org/projects/git-buildpackage/manual-html/gbp.html`

- Thanks: git, devscripts, pristine-tar, contributors and bug reporters
- Fragen? Kommentare?