Debian Packaging with Git

Workflows and Tools

David Bremner

26 July 2011

Talk pre-reqs and goals

Talk-Depends: basic-packaging (>=1.0), git-curious

- You know what a source package is
- You know what a git branch and tag is (or want to)

Talk pre-reqs and goals

Talk-Depends: basic-packaging (>=1.0), git-curious

- You know what a source package is
- You know what a git branch and tag is (or want to)

Goals for the talk

- take away some of the mystery
- know what is possible
- have some discussion about best practices

The Packaging Cycle

1. start with upstream source
2. possibly remove some non-free or just useless things
3. create or update ./debian
4. possibly patch upstream source
5. make a source package

Importing Upstream Source

- at least one of
  - get upstream $VCS history ▸ import git ▸ import svn
  - unpack tarball
- commit changes (including deletions) ▸ manual import
- maybe save a copy using pristine-tar
- maybe merge upstream in debian packaging branch
- maybe tag

pristine-tar

- Allows “checksum-identical” regeneration of tarballs, small overhead

```
pristine-tar commit foo.tar.gz [upstream-ref]
pristine-tar checkout where-ever/foo.tar.gz
```

- convenience feature, but for some rare tarballs, it doesn't work (well)

Tracking upstream $VCS and tarballs

- Get $VCS history on a branch
- Import tarball on top: diffs are contained in the import commit.
- Tag after import

Tools for importing tarballs

git-import-orig

- part of git-buildpackage; can be used “a la carte”
- Does the basic import
- optionally automatically use pristine-tar
- merges upstream into master
- Tags upstream, master

Tools for importing tarballs

git-import-orig

- part of git-buildpackage; can be used “a la carte”
- Does the basic import
- optionally automatically use pristine-tar
- merges upstream into master
- Tags upstream, master

git-dpm import-new-upstream

- works similarly to git-import-orig
- Does some git-dpm specific bookeeping
- optionally rebases the patch branch against the new upstream

Repacking

Delete-and-merge

- Delete $badstuff from upstream branch
- merge new upstream versions in. ▸ auto-resolve conflicts
- making your own upstream tags, post repack

Repacking

Delete-and-merge

- Delete $badstuff from upstream branch
- merge new upstream versions in. ▸ auto-resolve conflicts
- making your own upstream tags, post repack

Delete-before-import

- use your usual script/what have to to repack
- rely on import step to clean out any git copy of undesired stuff

./debian Best practices

Why keep upstream source in git at all?

- in general disk is cheaper than human time
- foreclose lots of interesting possibilities if you don't.

./debian Best practices

Why keep upstream source in git at all?

- in general disk is cheaper than human time
- foreclose lots of interesting possibilities if you don't.

Keep ./debian on its own branch?

- every change has to be made first on your debian-only branch, then merged
- git log makes it easy to classify commits according to what they touch

Ignore git, use quilt

- nothing new to learn.
- low-tech is good.
- nothing ventured, nothing gained.

Ignore quilt, use git

- convenient
- automagical handling of upstream merging of patches
- filtering commits to send upstream is easy
  - git classify script
- less obvious how to generate “nice” source packages.

Using git for patches

- Store patches as git branches (either one patch per branch, or one patch per commit)
- Export quilt patches
- Maybe commit patches
- Maybe merge patches into debian branch

Patch generation

- Need to sort out which commits should be (squashed into) quilt patches
- rebasing is one way
- keeping a “top-base” is another
- perhaps just scan history for “upstream” commits not on upstream branch.
  - in practice your history has to be **very** clean

topgit

- keeps history of patch as a branch
- handles dependencies between patches
- conceptually nice; can be frustrating when things go wrong
- might be overkill for debian packaging.
- still need to export patches, commit them to repo

Merging patches into debian branch

- Automatically done by git-dpm, but can be done for gitpkg, gbp-pq: just use git merge.
- relies on dpkg-source to detect patches are already applied
  - if first patch fails to apply, assume all are unapplied
- conflicts could cause things to go weird, but shouldn't happen if the patch branch is rebased.

git-buildpackage patch utility: gbp-pq

- a thin wrapper aroung git quilt-import and git format-patch
- most natural to patch against debian branch, have a single patch branch (import command)
- based on rebasing; sharing is via quilt-import

git-dpm patch handling

- one patch branch, one patch per commit.
- keeps patches merged into master branch
- keeps a copy of patches committed.
- based on rebasing “patched” upstream branch
- encodes private branch $SHA1 into ./debian/.git-dpm

gitpkg quilt export hook

- allows you to base patches on whatever
- debian/source/git-patches give "recipe"

```
upstream/${UPSTREAM_REF}..patches/${DEBIAN_REF}
upstream/${UPSTREAM_REF}..topic/${UPSTREAM_REF}
```

- Does not commit patches; there is a lintian check
- relies on tags for reproducability

making a source package

gitpkg

- makes a source package, taking upstream source from branch, tarball, pristine-tar

making a source package

gitpkg

- makes a source package, taking upstream source from branch, tarball, pristine-tar

git-dpm

- no built-in command
- can use dpkg-source -b etc..
- can use gitpkg/git-buildpackage to export from git, including pristine-tar support.

making a source package

gitpkg

- makes a source package, taking upstream source from branch, tarball, pristine-tar

git-dpm

- no built-in command
- can use dpkg-source -b etc..
- can use gitpkg/git-buildpackage to export from git, including pristine-tar support.

git-buildpackage

- by default, builds binary packages
- optionally makes tags

Summary: git-buildpackage

- many people and teams use it
- easy to get started, one command
- docs are worthwhile even if you don't use its
- convenience features
- patching workflow is less integrated compared to git-dpm

Summary: git-dpm

- sensible workflow
- a bit daunting to get started, many new commands
- depends whether you want to wrap git or not
- 30 minutes walking through the documentation pays off.
- a bit more “all or nothing” than the other solutions

Summary: gitpkg

- minimalist: least automated of the three
- flexible
- no assumptions about repo
- easy to extend and customize via hooks
- for people not avoiding git

Converting from SVN

- git-svn
- svn-all-fast-export

Managing large numbers of repos

- mr
- avoiding 2000 invocations of git pull: using commit hooks to track latest hashes (pkg-perl, experiment in progress)

Multi Tarballs

- what is needed?

Manually Importing Tarballs

back

```
# Wipe the working directory
git clean -fxd &&  git ls-files -z | xargs -0 rm -f

# Assuming the tarball is "nice"
tar --strip-components=1 -zxvf $1

# Commit changes
git add -A && git commit -m'Importing '`basename $1`
```

Automagically resolving conflicts in dfsg branches

◂ back

```
git checkout upstream
git merge -s recursive -X theirs $real_upstream_tag
```

- optionally auto resolve file deletion conflicts:

```
git status -s | egrep '^(DU|UA| U|U |UD)' | cut -c4- | \
        xargs git rm --ignore-unmatch DUMMY$$
git commit
```

Tracking upstream git

back

```
git remote add upstream-repo $url
git fetch upstream-repo
git checkout upstream && git merge $upstream-tag
```

Tracking upstream svn

back

- First time
  - git-svn init $url
  - git-svn fetch
  - git log refs/remotes/git-svn
  - git checkout -b upstream-svn refs/remotes/git-svn
  - git push origin upstream-svn:upstream-svn
- Next time
  - git config –remove-section svn-remote.svn 1>/dev/null 2>&1
  - git svn init $url
  - git show-ref origin/upstream-svn > ‘git rev-parse –git-dir‘/refs/remotes/git-svn