Using Git with PostgreSQL

Andrew Dunstan

andrew@dunslane.net
andrew.dunstan@pgexperts.com

Dunslane Consulting, LLC

PGX POSTGRESQL EXPERTS, INC.

Playing along

- Community repository
  - git clone git://git.postgresql.org/git/postgresql.git pgsql
- My repository:
  - git clone git://github.com/oicu/pg-cvs-mirror.git pgsql
  - Has clean (so far) versions of all live back branches

Useful references

- http://book.git-scm.com (Git Community Book)
- http://progit.org (Pro Git)
- http://oreilly.com/catalog/9780596520137/ (Version Control with Git/ Loeliger)
- http://wiki.postgresql.org/wiki/Working_with_Git
- http://wiki.postgresql.org/wiki/Switching_PostgreSQL_from_CVS_to_Git

CVS work pattern – developer

- `cvs checkout pgsql`
- Repeat till done:
  - Work
  - Test
  - Occasionally `cvs update`
    - Hope it doesn't blow up your work
    - If it does, fix it by hand

CVS developer problems

- Can't branch
- Can't easily checkpoint code or roll it back
- No merge support to speak of
- Have to fool cvs about added files for patch inclusion/deletion

Git work pattern – developer

- `git clone repo-url`
- `git checkout -b mydev`
- Repeat till done:
  - Work
  - Test
  - Every so often, `git commit -a`
  - Occasionally, `git pull origin`
    - Fix conflicts with git mergetool
- Can add branches for parallel development

Advantages

- Checkpointing code
  - Commit early and often
- Can branch multiple times for parallel lines of development
- Easy to abandon unpromising lines, or roll back to a previous commit

Developing on multiple platforms

- Make one local repo the master, set up a server on it, push to / pull from it
  - e.g. parallel pg_restore
    - Developed on Linux and Windows
    - Syncing by hand was a pain
    - Git would have made it much easier

Extracting diffs

- CVS:
  - `cvs diff -c > patchfile`
- Git:
  - `git diff master devbr > patchfile`
  - No context diffs natively ☹

Setting up for context diffs

- Copy <http://anarazel.de/pg/git-external-diff> to your libexec/git-core directory
- `git config diff.external git-external-diff`
  - Add `–global` if you want to use it everywhere.
  - To ignore white space, use

```
DIFF_OPTS=-pcdw git diff ...
```

Adding commands

- `git config –global alias.co checkout`
  - Now can do:
    - `git co devbranch`

Publishing work

- CVS: email patch place on web
- Git: can also push to a public repository

Buildfarm client changes

- Step 1: abstract out CVS specific code into an SCM object
- Step 2: create a git flavor of the SCM object

Buildfarm SCM object creation and access

- `new()` - class level factory method. Returns a member of appropriate subclass (PGBuild::SCM::CVS or PGBuild::SCM::Git)
- `check_access()` - sanity check for CVS pserver logins. Noop for git.

Buildfarm Ignored files

- `find_ignore()` - get the contents of .cvsignore files
  - CVS: prune `CVS` directories from search
  - Git: prune `.git` directory from search
    - Open item: will we just convert .cvsignore to .gitignore when we move to git?

SCM Object Utilities

- `get_build_path()` returns a path where the build will occur
  - SCM dependant because it is different for CVS export method
- `copy_source_required()` - false if using CVS export method, otherwise true
- `copy_source()` - copies the source to the build path
  - Git: avoids copying `.git` directory

SCM object API – CVS checkout

- If using export method, call `cvs export`
- Otherwise
  - If source directory exists, call `cvs update`
  - Else call `cvs checkout $branch`
  - Parse output and possibly call `cvs status` to make sure directory is clean

SCM object API – Git checkout

- If source directory exists, call `git pull`
- Else:
  - Call `git clone`
    - Use `-reference` parameter if configured
  - Call `git checkout -b bf_$branch -t origin/$branch`
- Call `git status` and parse output to make sure directory is clean.
  - Unnecessary if we just cloned, but very cheap

SCM object API – file info

- `find_changed()` - get lists of what has changed since the last time we ran, and since the last time we ran successfully
  - CVS: uses file modification time
  - Git: uses

    ```
    git log --since $ts [ --until $ts2 ]
    ```
    - <u>Much</u> more robust

SCM object API file info 2

- `get_versions()` - turns a list of files into a list of {file version} pairs
  - CVS: uses `cvs status`
  - Git: uses info from git log already stashed away in `find_changed()`
    - “version” ID is commit hash
    - ~~Assumes that repo is cloned directly or indirectly from `git://git.postgresql.org/git/postgresql.git`~~

That's it!

- Should be easy to add another SCM if anyone ever wanted to
  - Mercurial anyone? Monotone?

Buildfarm server changes

- Very minor
  - `alter table build_status add scm text, add scmurl text;`
  - Change `pgstatus.pl` script to populate fields from config setting
  - Change `show_log.pl` script to point to git repo change set in changed files links if the scm is git.

Buildfarm config file changes

- New param `scm`
  - defaults to `cvs`
- New params `scm_repo and scm_url`
  - default to community repo according to value of `scm`
- New param git_reference
  - Used in `git clone` operation if set
- Legacy param `cvsrepo` still supported

Setting up a local repo clone

- Very desirable if you are running multiple buildfarm members / branches
- Also desirable to reduce proliferation of .git directories

Local repo in CVS:

- `rsync anoncvs.postgresql.org::pgsql-cvs /home/cvsmirror/pg`
  - Called from `cron`
- If buildfarm members run on multiple machines:
  - Set up a local pserver
  - Point buildfarm members at local pserver
- Else
  - Point buildfarm members at repo directory

Local repo in git

- Simple setup: make one clone on each buildfarm machine:
  - git clone --mirror git://git.postgresql.org/git/postgresql.git /home/gitmirror/postgresql.git
  - cd /home/gitmirror/postgresql.git && git fetch
    - Called from cron or scheduler

Using simple git setup for buildfarm members

- Point each buildfarm member at local mirror:
  - scm_repo => '/home/gitmirror/postgresql.git'
    - Cloning local mirror uses hard links to .git dir pack objects

Making a tree of clones

- Clone community repo to one local machine
- Set up local git server
  - Use `git daemon` or `gitosis`
- Clone local server bare to each buildfarm machine as in simple setup.
- Reduces external network traffic

So why isn't the buildfarm client code in git?

- Sanity check on server:
  - Reject status from clients that are too old. Done by checking CVS version number.
- Git doesn't have version numbers
  - “Duh! It's distributed!” ☺
- It does have commit Ids, but they are not ordered.
- Could use a commit date, but git won't fill in a date keyword!

Getting someone else's work

- Create a branch:
  - `git checkout -b devbranch master`
    - Or a release branch
- Apply patch from contributor
  - `patch -p 1 < patchfile`
    - `git apply` doesn't work with context diffs ☹
  - `git add {list of new files}`
- Or, pull from a public repo:
  - `git pull remote-repo remote-branch`

Intriguing possibilities

- Build unofficial branches
  - Put your code on a repo (github?)
  - Point your buildfarm member there
  - Server requirements:
    - Don't notify mailing lists
    - Separate dashboard for unofficial brtanches

Committer / Tester / Developer workflow

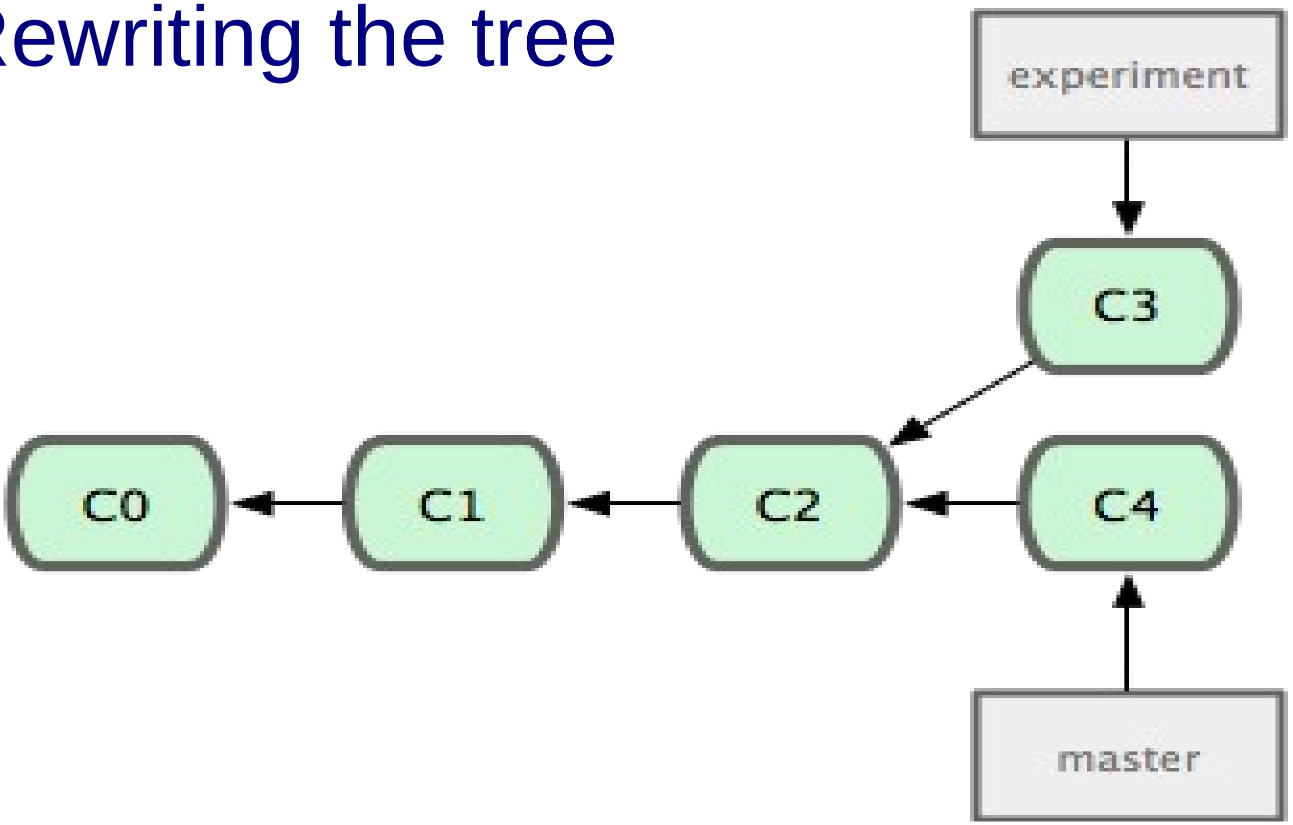
- Repeat till done:
  - Work, commit, test
  - Commit a lot, it won't affect anyone else
- Publish patches made with `git diff` and optionally push to a public, non-authoritative repo

Committer workflow

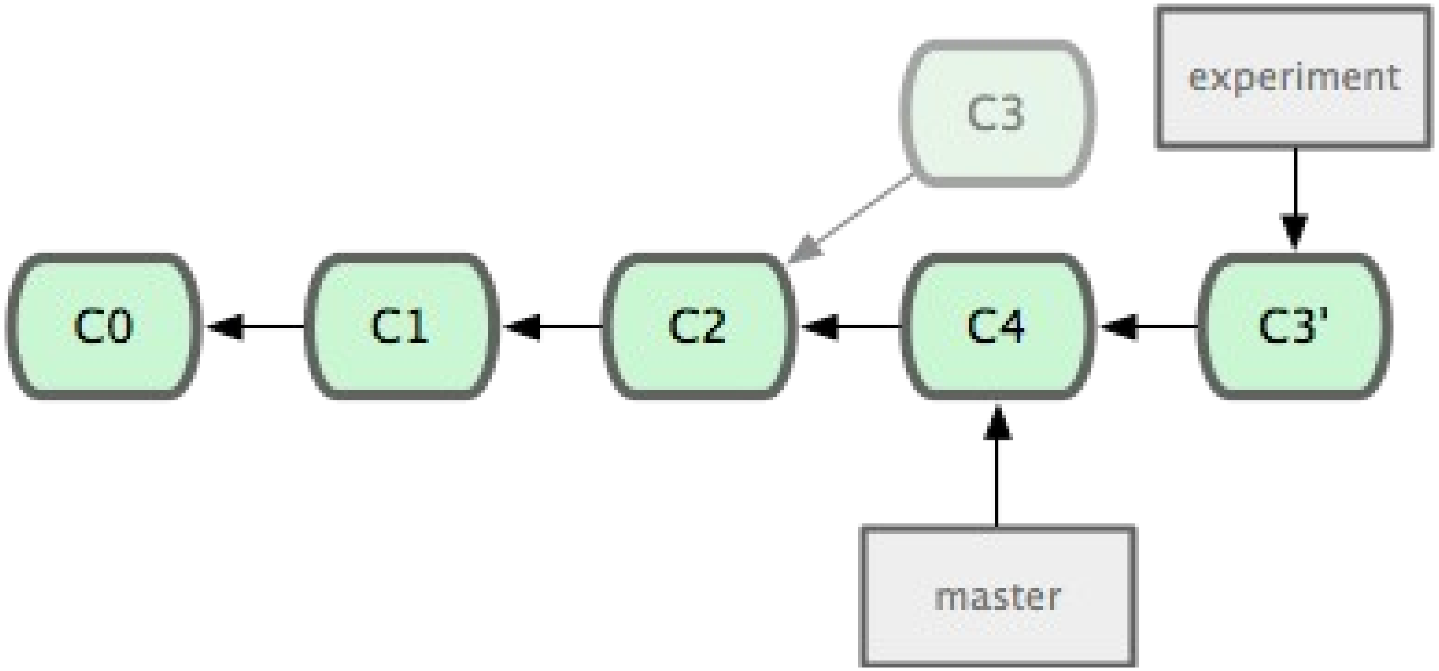
- Switch back to main branch
  - `git checkout master`
    - Or release branch if working on a stable branch
- Make sure it's up to date
  - `git pull origin HEAD`
- Merge in changes
  - `git merge –squash devbranch`
- Send changes to server
  - `git push origin HEAD`

Rebasing

- Rewriting the tree



Rebase result



And the moral of that is ...

- Do not ever rebase a commit that you have pushed elsewhere.
- For beginners with PostgreSQL workflow, rebasing is possibly not necessary at all.

How many trees?

- Strategy one:
  - Keep one clone, switch between branches using checkout
- Strategy two:
  - Keep one clone per live branch
  - Keep a bare clone you fetch to, clone from there

Multi-tree recipe (h.t. Aidan van Dyk)

- git clone --bare --mirror git://committer.postgresql.org/PostgreSQL.git PostgreSQL.git
- git clone --reference PostgreSQL.git git://committer.postgresql.org/PostgreSQL.git master
- git clone --reference PostgreSQL.git git://committer.postgresql.org/PostgreSQL.git REL8_4_STABLE
- cd REL8_4_STABLE/ && git checkout --track -b REL8_4_STABLE origin/REL8_4_STABLE
- cd /path/to/base/PostgreSQL.git && git fetch
    - Called from cron

Backporting

- Try git cherry-pick
  - Only from the same tree
  - If using many trees, pull in branch from other tree:
    - git pull ../other_branch_dir branchname
  - Other suggestions have been made, nobody seems terribly sure (see wiki discussion)
  - Do we need to write some tools?