Git workflow proposal

Rok Strniša and Matthias Görgens

October 5, 2010

1 Abstract

In this document, we describe a Git workflow that replicates our current Mercurial approach.

2 Why git?

Using Mercurial and many extensions, one can accomplish the same tasks as in Git.[1] However, we know from experimentation that Git is friendlier: (a) probable subsequent commands are suggested, and (b) Git does not execute commands that would lead to a corrupted state (*e.g.* compared to Mercurial pull with patches applied).

In our current Mercurial workflow, we are employing commits, patch queues and, to a limited extent, branches. In the Git workflow described in this document, we replace Mercurial patch queues with Git's lightweight branches and rebasing.

Lastly, Git is becoming more and more popular. The online hub for Git repositories, GitHub, already has the largest number of repositories of any hub, and the second largest number of users (after SourceForge).[2]

Advantages of using GitHub:

- visibility;
- source availability;
- pull requests;
- comments on commits, source lines, etc; and,
- web-based editing.

The full costs and benefits analysis is given in section 7.

[1] `http://mercurial.selenic.com/wiki/GitConcepts`

[2] `http://en.wikipedia.org/wiki/Comparison_of_open_source_software_hosting_facilities`

3 Static overview

The primary copy of each open source repository would be on GitHub. However, each such repository would also have a Citrix-local cache (on an internal server). We would set up a post-change hook at GitHub, which causes the local cache to pull changes from GitHub as soon as they are made..

Each repository has a main branch (*e.g. trunk* or *master*).

4 Developer workflow

The build system clones the locally cached repository. The developers (as well as the gatekeeper), however, create a GitHub fork of the main GitHub repository (owned by a group user *xen.org*), then clone their own copy (which is used by the build system as a *myrepos* repository).

The user, therefore, performs the following:

```
# creates the GitHub fork (only needs to be done once), then
$ cd ⟨myrepos⟩
$ git clone git@github.com:⟨user-name⟩/⟨rep-name⟩.git
$ cd ⟨rep-name⟩
```

Inside the developer's clone, the developer's GitHub fork is referred to as *origin*.

To work on some feature A, the developer creates a branch, *e.g.* named *A*, off the main branch. Then, the developer makes regular commits to *A*, and also pushes the whole branch to his GitHub fork.

The developer starts with the following to make and checkout a new feature branch A:

```
$ git branch A
$ git checkout A
```

Then, the following steps are repeated as necessary:

```
$ ed ⟨files⟩          # Make changes here.
$ git status          # What changes have been made?
$ git add ⟨files⟩     # Select changes to commit.
$ git commit -m "My amazing changes, part x."
$ git push origin A   # Push changes to my GitHub fork.
```

Once the feature is stable, the developer *makes a pull request*[3] to the main repository on GitHub. Every user within the user group *xen.org* can then see that pull request, can review it, can comment on it (discussions for each commit/pull request/etc are supported), and can accept or reject it.

[3] `http://help.github.com/pull-requests/`

5 Gatekeeper workflow

The role of a repository gatekeeper is to deal with the general administration of the main GitHub repository. This will likely mostly include dealing with the pull requests. After a pull request has been reviewed, it can be merged into the main repository with the following commands (assuming that the gatekeeper has a clone of the main repository):

```
$ git checkout master
$ git remote add ⟨user-name⟩ git://github.com/⟨user-name⟩/⟨rep-name⟩.git
$ git fetch ⟨user-name⟩
$ git merge ⟨user-name⟩/⟨branch-name⟩
$ git push origin master
```

Or just with:

```
$ git checkout master
$ curl http://github.com/⟨user-name⟩/⟨rep-name⟩/pull/⟨req-no⟩ | git am
$ git push origin master
```

If the pull request results in a conflict, the gatekeeper puts this into a comment of the pull request. Then, the author can re-base their branch to the current trunk (and so remove all conflicts).

The gatekeeper must also ensure to never rebase versions tagged with *stable*. This is useful for both internal and external developers.

Note that *gatekeeper*, *developer* and *reviewer* are roles and not persons. A specific branch's developer and reviewer should not be the same person, but the gatekeeper can also develop or review.

6 Use case: dependency on another branch

After having made some commits for A, the developer realises the need for a yet-unmerged feature B from another user. This can be done with merging:

```
$ git checkout A
$ git remote add ⟨user-name⟩ git://github.com/⟨user-name⟩/⟨rep-name⟩.git
$ git fetch ⟨user-name⟩
$ git merge ⟨user-name⟩/B
```

Or with rebasing:

```
$ git checkout A
$ git remote add ⟨user-name⟩ git://github.com/⟨user-name⟩/⟨rep-name⟩.git
$ git fetch ⟨user-name⟩
$ git rebase ⟨user-name⟩/B
```

If others are already relying on feature A, merge should be used; otherwise, the user should use rebasing, since it produces simpler commit graphs (see `git help rebase`).

7 Costs and benefits analysis

This section tries to best estimate the costs and benefits of switching to the described proposal.

7.1 Costs/cons

user adjustment While Mercurial and Git are similar, there would be some adjustment cost. Here, sites like `http://mercurial.selenic.com/wiki/GitConcepts#Command_equivalence_table` can help a lot.

caching required We would need to cache GitHub repositories locally mainly for purposes of speed (less so for reliability). This can be done quite easily, and GitHub supports hooks that would cause the cache to pull changes as soon as they are made.

build system adjustment The build system needs to be able to cope with Git repositories. According to the build system engineers, this should not be too hard to implement.

no JIRA support Although GitHub does have its own issue tracking, it currently cannot be integrated with JIRA.

7.2 Benefits/pros

great visibility As discussed, GitHub is becoming highly popular, which would give us great exposure.

pull requests GitHub supports users to make explicit (and recorded) requests to the gatekeeper. This alleviates much administration off the gatekeeper.

better overview GitHub can also track issues (but it's not clear whether its tracker is good enough).

code reviewing support All pull requests can be easily reviewed, commented on, rejected with reason, etc.

no patch queues We eliminate patch queues with light branches. This makes the commit structure more flexible, and gives a better overview.

distribution branching We can use the same branching system for different distributions as well, e.g. have a separate "Cowley beta" branch.

other GitHub features GitHub also gives us a versioned Wiki page, a downloads page (a place where we can place source/binary distributables), and a versioned home (HTML) page.