Basic usecases for DVCS: Workflow Failures

Arne Babenhauserheide

April 17, 2013

Contents

1 Intro

I recently tried contributing to a new project again, and I was quite surprised which hurdles can be in your way, when you did not setup your environment, yet.

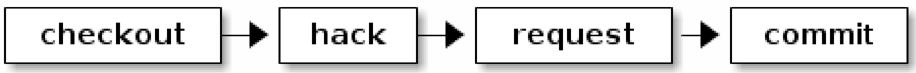
So I decided to put together a small test for the basic workflow: Cloning a project, doing and testing a change and pushing it back.

I did that for Git and Mercurial, because both break at different points.

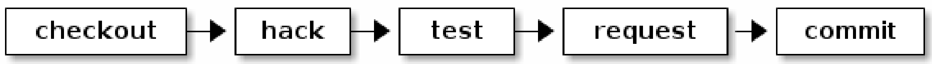
I'll express the basic usecase in Subversion:

- svn checkout [project]
- (hack, test, repeat)
- (request commit rights)
- svn commit -m "added X"

You can also replace the request for commit rights with creating a patch and sending it to a mailing list. But let's take the easiest case of a new contributor who is directly welcomed into the project as trusted committer.



A slightly more advanced workflow adds testing in a clean tree. In Subversion it looks almost like the simple commit:



2 Git

Let's start with Linus' DVCS. And since we're using a DVCS, let's also try it out in real life

2.1 Setup the test

```
LC_ALL=C
LANG=C
PS1="$"
rm -rf /tmp/gitflow > /dev/null
mkdir -p /tmp/gitflow > /dev/null
cd /tmp/gitflow > /dev/null
# init the repo
git init orig  > /dev/null
cd orig > /dev/null
echo 1 > 1
# add a commit
git add 1 > /dev/null
git config user.name upstream > /dev/null
git config user.email up@stream > /dev/null
git commit -m 1 > /dev/null
# checkout another branch but master. YES, YOU SHOULD DO THAT on the shared repo. We\T1\textquoteright ll
git checkout -b never-pull-this-temporary-useless-branch master 2> /dev/null
cd .. > /dev/null
echo # purely cosmetic and implementation detail: this adds a new line to the output
ls
```

```
$$$$$$$$$$$$$$$$$
orig
```

```
git --version
```

```
git version 1.8.1.5
```

2.2 Simplest case

2.2.1 Get the repo

First I get the repo

```
git clone orig mine
echo $ ls
ls
```

```
Cloning into 'mine'...
done.
$ ls
mine  orig
```

2.2.2 Hack a bit

```
cd mine
echo 2 > 1
git commit -m "hack"
```

```
$# On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

modified:   1
no changes added to commit (use "git add" and/or "git commit -a")
```

ARGL... but let's paste the commands into the shell. I do not use –global, since I do not want to shoot my test environment here.

```
git config user.email "contributor"
git config user.name "con@tribut.or"
```

and try again

```
git commit -m "hack"
```

```
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)

modified:   1
no changes added to commit (use "git add" and/or "git commit -a")
```

ARGL... well, paste it in again...

```
git add 1
git commit -m "hack"
```

```
[master 28b541b] hack
 1 file changed, 1 insertion(+), 1 deletion(-)
```

Finally I managed to commit my file. Now, let's push it back.

2.2.3 Push it back

```
git push
```

```
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 5, done.
(1/3)
Writing objects:  66% (2/3)
Writing objects: 100% (3/3)
Writing objects: 100% (3/3), 222 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To /tmp/gitflow/orig
master
```

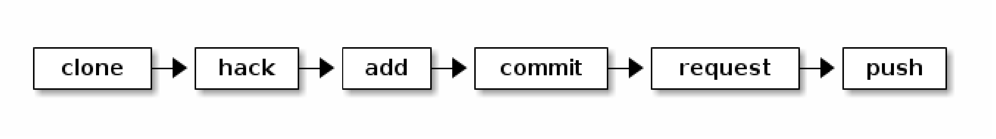
HA! It's in.

2.2.4 Overview

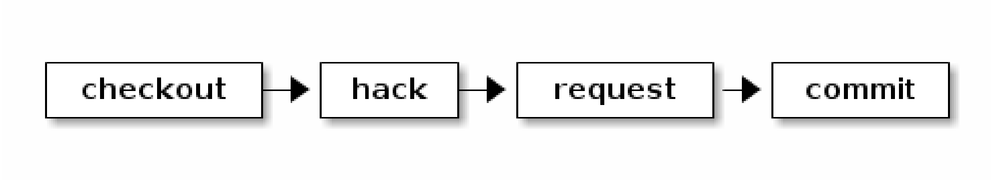
In short the required commands look like this:

- git clone orig mine
- cd mine; (hack)
- git config user.email "contributor"
- git config user.name "con@tribut.or"
- git add 1
- git commit -m "hack"
- (request permission to push)

- git push



compare Subversion:



Now let's see what that initial setup with setting a non-master branch was about…

2.3 With testing

2.3.1 Test something

I want to test a change and ensure, that it works with a fresh clone. So I just clone my local repo and commit there.

```
cd ..
git clone mine test
cd test
# setup the user locally again. Normally you do not need that again, since you\T1\textquoteright d use --g
git config user.email "contributor"
git config user.name "con@tribut.or"
# hack and commit
echo test > 1
git add 1
echo # cosmetic
git commit -m "change to test" >/dev/null
# (run the tests)
```

2.3.2 Push it back

```
git push
```

```
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:
  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 5, done.
(1/3)
Writing objects:  66% (2/3)
Writing objects: 100% (3/3)
Writing objects: 100% (3/3), 234 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: error: refusing to update checked out branch: refs/heads/master
remote: error: By default, updating the current branch in a non-bare repository
remote: error: is denied, because it will make the index and work tree inconsistent
remote: error: with what you pushed, and will require 'git reset --hard' to match
remote: error: the work tree to HEAD.
remote: error:
remote: error: You can set 'receive.denyCurrentBranch' configuration variable to
remote: error: 'ignore' or 'warn' in the remote repository to allow pushing into
remote: error: its current branch; however, this is not recommended unless you
remote: error: arranged to update its work tree to match what you pushed in some
remote: error: other way.
remote: error:
remote: error: To squelch this message and still keep the default behaviour, set
remote: error: 'receive.denyCurrentBranch' configuration variable to 'refuse'.
To /tmp/gitflow/mine
master (branch is currently checked out)
error: failed to push some refs to '/tmp/gitflow/mine'
```

Uh... what? If I were a real first time user, at this point I would just send a patch...

The simple local test clone does not work: You actually have to also checkout a different branch if you want to be able to push back (needless duplication of information - and effort). And it actually breaks this simple workflow.

(experienced git users will now tell me that you should always checkout a work branch. But that would mean that I would have to add the additional branching step to the simplest case without testing repo, too, raising the bar for contribution even higher)

```
git checkout -b testing master
git push ../mine testing
```

```
Switched to a new branch 'testing'
Counting objects: 5, done.
```

```
 (1/3)
Writing objects:  66% (2/3)
Writing objects: 100% (3/3)
Writing objects: 100% (3/3), 234 bytes, done.
 Total 3 (delta 0), reused 0 (delta 0)
 To ../mine
 testing
```

Since I only pushed to mine, I now have to go there, merge and push.

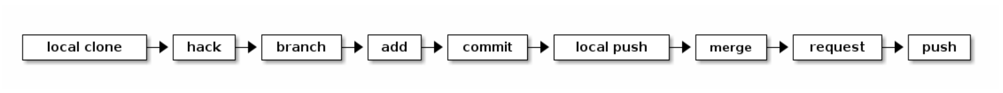
```
cd ../mine
git merge testing
git push
```

```
Updating 28b541b..6994390
Fast-forward
 1 | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 5, done.
(1/3)
Writing objects:  66% (2/3)
Writing objects: 100% (3/3)
Writing objects: 100% (3/3), 234 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To /tmp/gitflow/orig
master
```

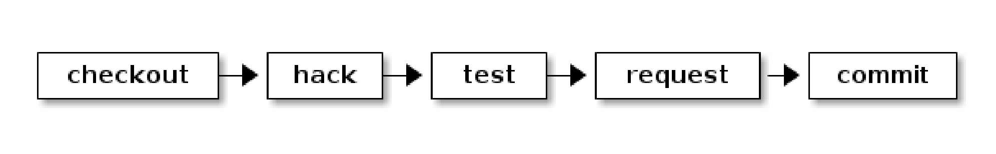
2.3.3 Overview

In short the required commands for testing look like this:

- git clone mine test
- cd test; (hack)
- git add 1
- git checkout -b testing master
- git commit -m “hack”
- git push ../mine testing
- cd ../mine
- git merge testing
- git push



Compare to Subversion



2.4 Wrapup

The git workflows broke at several places:
Simplest:

- Set the username (minor: it’s just pasting shell commands)
- Add every change (==staging. Minor: paste shell commands again - or use ‘commit -a‘)

Testing clone (only additional breakages):

- Cannot push to the local clone (major: it spews about 20 lines of error messages which do not tell me how to actually get my changes into the local clone)
- Have to use a temporary branch in a local clone to be able to push back (annoyance: makes using clean local clones really annoying).

3 Mercurial

Now let’s try the same

3.1 Setup the test

```
LC_ALL=C
LANG=C
PS1="$"
rm -rf /tmp/hgflow > /dev/null
mkdir -p /tmp/hgflow > /dev/null
cd /tmp/hgflow > /dev/null
# init the repo
hg init orig  > /dev/null
cd orig > /dev/null
echo 1 > 1 > /dev/null
# add a commit
hg add 1 > /dev/null
hg commit -u upstream -m 1 > /dev/null
cd .. >/dev/null
echo # purely cosmetic and implementation detail: this adds a new line to the output
ls
```

```
$$$$$$$$$$$$$
orig
```

```
hg --version
```

```
Mercurial Distributed SCM (version 2.5.2)
(see http://mercurial.selenic.com for more information)

Copyright (C) 2005-2012 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

3.2 Simplest case

3.2.1 Get the repo

```
hg clone orig mine
echo $ ls
ls
```

```
updating to branch default
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
$ ls
mine  orig
```

3.2.2 Hack a bit

```
cd mine
echo 2 > 1
echo
# I disable the username to show the problem
hg --config ui.username= commit -m "hack"
```

```
$
$abort: no username supplied (see "hg help config")
```

ARGL, what???

Well, let's do what it says (but only see the first 30 lines to avoid blowing up this example):

```
hg help config | head -n 30 | grep -B 3 -A 1 per-repository
```

```
These files do not exist by default and you will have to create the
    appropriate configuration files yourself: global configuration like the
USERPROFILE%\mercurial.ini" or
HOME/.hgrc" and local configuration is put into the per-repository
/.hg/hgrc" file.
```

Are you serious??? I have to actually read a guide just to commit my change??? As normal user this would tip my frustration with the tool over the edge and likely get me to just send a patch…

But I am no normal user, since I want to write this guide. So I assume a really patient user, who does the following (after reading for 3 minutes):

```
echo '[ui]
username = "contributor"' >> .hg/hgrc
```

and tries again:

```
hg commit -m "hack"
```

Now it worked. But this is MAJOR BREAKAGE.

3.2.3 Push it back

```
hg push
```

```
pushing to /tmp/hgflow/orig
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

Done. This was easy, and I did not get yelled at (different from the experience with git :)).

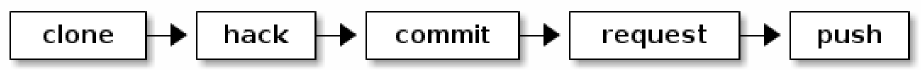
3.2.4 Overview

In short the required commands look like this:

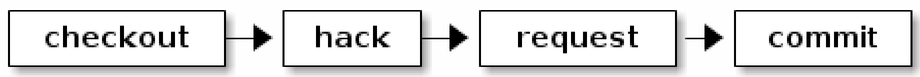
- hg clone orig mine
- cd mine; (hack)
- hg help config ; (read) ; echo '[ui]

username = "contributor"' » .hg/hgrc (are you *serious*?)

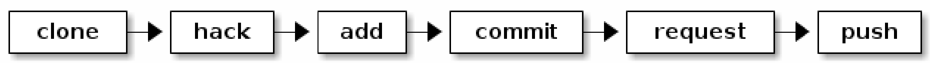
- hg commit -m "hack"
- (request permission to push)
- hg push



Compare to Subversion



and to git



3.3 With testing

3.3.1 Test something

```
cd ..
hg clone mine test
cd test
# setup the user locally again. Normally you do not need that again, since you\T1\textquoteright d use --g
echo '[ui]
username = "contributor"' >> .hg/hgrc
# hack and commit
echo test > 1
echo # cosmetic
hg commit -m "change to test"
# (run the tests)
```

```
updating to branch default
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
$$> $$$
```

3.3.2 Push it back

```
hg push
```

```
pushing to /tmp/hgflow/mine
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

It's in mine now, but I still need to push it from there.

```
cd ../mine
hg push
```

```
pushing to /tmp/hgflow/orig
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

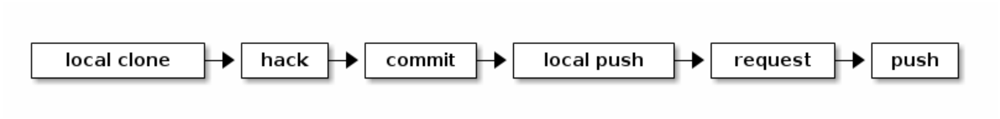
Done.

If I had worked on mine in the meantime, I would have to merge there, too - just as with git with the exception that I would not have to give a branch name. But since we're in the simplest case, we don't need to do that.

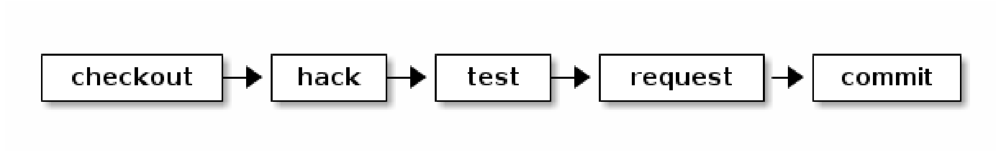
3.3.3 Overview

In short the required commands for testing look like this:

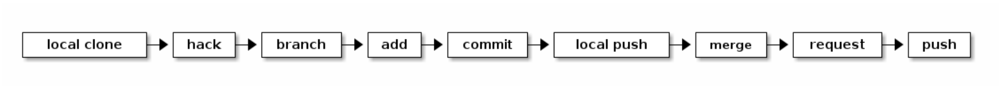
- hg clone mine test
- cd test; (hack)
- hg commit -m "hack"
- hg push ../mine
- cd ../mine
- hg push



Compare to Subversion



and to git



3.4 Wrapup

The Mercurial workflow broke only ONCE, but there it broke HARD: To commit you actually have to READ THE HELP PAGE on config to find out how to set your username.

So, to wrap it up: ARE YOU SERIOUS?

That's a really nice workflow, disturbed by a devastating user experience for just one of the commands.

This is a place where hg should learn from git: The initial setup must be possible from the commandline, without reading a help page and without changing to an editor and then back into the commandline.

4 Summary

- Git broke at several places, and in one place it broke hard: Pushing between local clones is a huge hassle, even though that should be a strong point of DVCSs.
- Mercurial broke only once, but there it broke hard: Setting the username actually requires reading help output and hand-editing a text file.

Also the workflows for a user who gets permission to push always required some additional steps compared to Subversion.

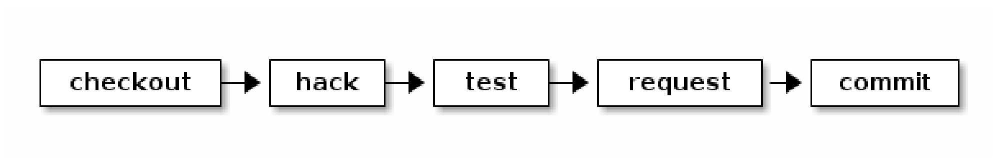
One of the additional steps cannot be avoided without losing offline-commits (which are a major strength of DVCS), because those make it necessary to split svn commit into commit and push: That separates storing changes from sharing them.

But git actually requires additional steps which are only necessary due to implementation details of its storage layer: Pushing to a repo with the same branch checked out is not allowed, so you have to create an additional branch in your local clone and merge it in the other repo, even if all your changes are siblings of the changes in the other repository, and it requires either a flag to every commit command or explicit adding of changes. That does not amount to the one unavoidable additional command, but actually further three commands, so the number of commands to get code, hack on it and share it increases from 5 to 9. And if you work in a team where people trust you to write good code, that does not actually reduce the required effort to share your changes.

On the other hand, both Mercurial and Git allow you to work offline, and you can do as many testing steps in between as you like, without needing to get the changes from the server every time (because you can simply clone a local repo for that).

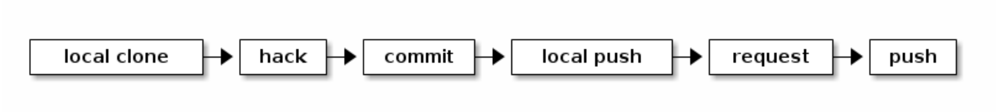
4.1 Visually

4.1.1 Subversion



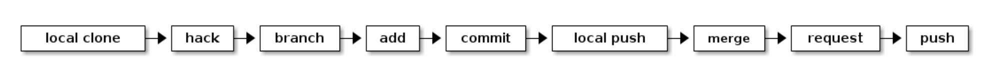
$$\begin{tikzcd}\boxed{\text{checkout}} \arrow[r] & \boxed{\text{hack}} \arrow[r] & \boxed{\text{test}} \arrow[r] & \boxed{\text{request}} \arrow[r] & \boxed{\text{commit}}\end{tikzcd}$$

4.1.2 Mercurial



$$\begin{tikzcd}\boxed{\text{local clone}} \arrow[r] & \boxed{\text{hack}} \arrow[r] & \boxed{\text{commit}} \arrow[r] & \boxed{\text{local push}} \arrow[r] & \boxed{\text{request}} \arrow[r] & \boxed{\text{push}}\end{tikzcd}$$

4.1.3 Git



$$\begin{tikzcd}\boxed{\text{local clone}} \arrow[r] & \boxed{\text{hack}} \arrow[r] & \boxed{\text{branch}} \arrow[r] & \boxed{\text{add}} \arrow[r] & \boxed{\text{commit}} \arrow[r] & \boxed{\text{local push}} \arrow[r] & \boxed{\text{merge}} \arrow[r] & \boxed{\text{request}} \arrow[r] & \boxed{\text{push}}\end{tikzcd}$$