DALES Git cheat sheet

Thijs Heus and Chiel van Heerwaarden

September 27, 2009

The DALES is managed with git as a content management system. The main distinction of Git over (e.g.) SVN, is that git is distributed. This means that everyone could carry all information at his local git repository. If you have no Internet, you can work on without a problem, if somebody screws up the main server, any copy from any of the users would suffice to get everything correct again.

1 Terminology

Branch Every different version of the code within the repository. The main version is called **master**

Repository The database with all the history of all the different branches available. When synchronized, all repositories contain all the data.

Local Repository The repository on your local computer. It is the one you work with and **commit** your modifications of the code to.

Remote Repository The repository on the central (gitorious) server. Communication between the local and remote repository goes with **pull**ing modified data from the remote repository into the local repository, and by **push**ing modifications from your local repository into the remote one.

working dir The directories that you do your editing in.

commit Putting the changes that you've made in your working dir to your local repository.

HEAD Is the version of the code as it was directly after your last commit. A 'clean HEAD' means that the version in your working dir is the same as the last committed version in your local repository. The most recent version before the current is called `HEAD~`, the second one is `HEAD~2`, the third one is called `HEAD~3` and so on.

Checkout To checkout a branch is to change from one branch to another, potentially with creating a new branch to work in.

Theoretically, we do not even need to set up a main git server, but it is nice to have one and refer to it as 'the main code'. Therefore, our code is located at www.gitorious.org/dales Everybody can download it, and remain up to date, but for developers it is necessary to set up a few more things.

1. Make sure you have git installed on your local system.
2. Set up an account at gitorious.org
3. Set up a local repository

2 How to set up an account?

First go to www.gitorious.org and set up an account over there. Then ask someone who is already part of dales-dev to add you to that group. Now you could edit all the repositories. There is one master repository that represents the main code, and one branch per developer where everyone is free to do in what he wants. Of course, other peoples branches should normally not be touched, and the master branch with much care, that is, for bugfixes and clear features that are well developed only. Pushing code to gitorious goes best over ssh, and to get that working smoothly you can add your ssh key (located in `~/.ssh/id_rsa.pub`; possibly you have to run `ssh-keygen` first) to your gitorious profile. Once you have completed your gitorious account, you can get the code by typing

```
git clone git@gitorious.org:dales/dales.git
```

at the command line. This gives you the master branch in subdirectory dales. If you prefer to clone the repository in a different dir, then append this dir name to the command. Now go to this subdirectory. Any git command you want to apply on your repository needs to be done from somewhere within this subdir.

Now, let's look at the different branches that are available. Just after the cloning, only the master branch is available in the local repository, but in the remote repository there are already a few branches available. So type:

```
git branch -a
```

to see all branches. The local branches are shown first, and the remote branches after that.

If you want to switch to a different branch that is already present in the remote repository, the first time you type:

```
git checkout -b <newbranch> origin/<newbranch>
```

This means that you make a local branch with the name `<newbranch>` which tracks the remote branch `origin/<newbranch>`, which whom you communicate through `git pull` and `git push`. Note that `origin` is a tag that points to the address that you specified in `git clone` when you created the repo. After that, changing branches can be done with:

```
git checkout <branchname>
```

If you want to know which branches are now available locally, type in: `git branch`. At this stage, you should see 2 branches: The master branch and the `<newbranch>` branch you just created. The one with a star in front of it is the active branch. Changes you make in the code will apply only to this active branch.

2.1 Inserting your current private code version into git

There are two possibilities to insert your non-gitted version into our repository. One is when the changes between your code and the current master are few, one is when there are many changes.

Similar to the master Assuming that the code looks similar to the master, you can create a new branch from the master with `git checkout -b <newbranch> master`. In the DALES repository this has already been done for the current main developers. Now copy the files you've altered from your old source dir into the git repository; include all the files you're not sure whether you've changed them. Git will find out about what's changed and what is not. Check whether the resulting code actually works. At the very least, ensure your code still compiles. If so, then you are ready to commit your initial version to the git repositories.

Quite different than the master If your code is quite different from the master, and there is also no other branch in the code that is similar, it is probably best to do a manual merge outside the git repository. So create the new branch like described above (if necessary), but do not copy any files directly into the git working dir. Instead, use a merge tool like kdiff3, meld or xdiff to compare file by file the differences and merge the two codes. This usually works with something like:
`<mergetool> <private-code-dir> <git-code-dir>`. This should give you an opportunity to review the merge on a case-by-case basis, and decide to take either your private version or the one that is already in git. After this manual merge, the code in the git working dir should be the code you want to work with. Try this out, at least by compiling the code. You are now ready to commit your initial version to the git repositories.

Performing the initial commit Now use
`git status`
and
`git diff`
to review what kind of differences there exist between your version and the one in the git repository. If you're happy with the changes, you can synchronize the local git repository with
`git commit -a`
. A vi screen will pop up; you should describe the changes here.

Now the local repository is up to date with your changes, you need to synchronize with the remote repository. This is done with
`git push`

3 Useful commands

The man pages of git are not very clear sometimes, but they are quite exhaustive. Be sure to give them a look. Updating your local repository with remote changes goes with git pull:
`git pull origin <branchname>`
to update just one branch, or just
`git pull`
to update everything (in the default settings). If you want to commit your changes to your (local) repository, type
`git commit -a`
and describe what you are changing in the commit file. It is advisable to only

commit if a feature/bug fix is in principle complete, and especially to keep the code in the repository compilable at all times. Putting stuff in the remote repository goes with
`git push`
If you want to check what has been changed since the last commit, use
`git status` and `git diff`
Adding files and directories to git goes with:
`git add`
Also, removing from and moving inside the repository needs to be done carefully: Use
`git rm` or `mv`
for that.

If you screwed up the code during coding, and did not commit any changes yet, you can reset your files to the latest state using:
`git checkout -f`
If you just want to restore the original state of file `uclalesrules.f90`, do:
`git checkout -f uclalesrules.f90`

If you want to merge the changes of branch1 into branch2, do:
`git checkout <branch2>`
`git merge <branch1>` If there are merge conflicts (could also happen with a `git pull`),
then you can use
`git mergetool`
to resolve the conflicts manually.

If there is a specific commit you would like to have merged into your branch, you can look up the commit number (e.g. at gitorious) and use cherry-pick:
`git cherry-pick 6159a97`
If you screwed up the code and want to get back to the way it was after your last commit, type:
`git reset --hard HEAD`
A faulty commit can be reset with
`git reset -soft HEAD^`
See the man page of `git-reset` for more information of the use of `-soft`In general, be careful with reset and especially reset –hard, since it can confuse the history of the commits. Never ever use reset –hard to get rid of commits that are already pushed to the main repository. To remove an earlier commit, or a commit that has already been pushed to the remote repository, you should use `git revert <commit-number>}`.
To make `revert` work, you need a clean HEAD, so you should either `git commit - a` all your current results, or `git reset --hard HEAD` your working dir to the latest committed version first.

To remove all files and directories that are not tracked by git from the current dir, use
`git clean -d -f`

Finally, a few graphical tools are available to play with if you like: `git gui`, `gitk` should at least be there.