Table of Contents

First steps with Git

1 Telling who you are

- Copy `.gitconfig` from *virtualboxconfiguration* into `~/.gitconfig`
- Edit the file and update your name and email

2 Every day work

2.1 Create an empty repository

```
$ mkdir quantum_time_machine
$ cd quantum_time_machine
$ git init
```

2.2 Commit a file

Two steps: add the file to the list of files to be committed, then commit.

```
$ git add file
$ git commit
```

Shortcut: `$ git commit -a`

2.3 Status

What files have I touched? What files will be committed in the next commit? What files are not being tracked by the repository?

```
$ git status
```

2.4 Differences with the repository

Difference between work tree (local) and index (local, files to be committed)

```
$ git diff
```

Difference between index (local, files to be committed) and HEAD (local, files committed)

```
$ git diff --cached
```

2.5 Remove untracked (and unwanted) files

Clean files not tracked by the repository (files generated by compilations, bak files, etc).

```
$ git clean -fd
```

Use with care! You may remove files you forgot to add to the repo.

3 Fixing mistakes

Reset your repo and forget all local changes

```
$ git reset --hard HEAD
```

Reset your repo. This changes the *history* of your local repo. All later commits will be forgotten (and lost)

```
$git reset --hard <commit>
```

Undo a commit

```
$ git revert <commit>
```

(this will generate a diff to undo the commit, you will have to make a commit to add the changes to the repo, you may need to fix conflicts in the revert)

4 Exploring history

Old school way

```
$ git log
```
`$ git logg` (non-standard git command, see Amadeus `.gitconfig` file)

Preferred way: install *QGit*, run in work tree

5 Branches

What you see in your work tree is the state of a branch of the repository. The repo contains all the history for all the branches. The default branch is called *master*.

- See repo branches

```
$ git branch
```

- Create a branch (no matter if you have local changes)

```
$ git branch flux_condenser
```

- Change to a branch

```
$ git checkout <branch>
```

5.1 Integrating branches

There are two ways to integrate changes from a branch into another: *merge* and *rebase*

Integrating branches is a lot of fun, in particular when the same files have been touched in the two branches. That's a *conflict* and must be solved by hand.

- Merge a branch

```
$ git merge <different_branch>
```

- Rebase: update a branch from another (takes all the patches from the changed branch and reapply them in the rebased branch)

```
$ git rebase <different_branch>
```

In summary:

- Merge: exactly same commits *merged* into the target branch (histories are *fusioned*)
- Rebase: new commits in the target branch, copies of the same changes in the origin branch (it is like you had been working in the target branch)

Merge or rebase? This is an important religious debate in the Git community. Nearly as important as the tabs vs. spaces debate.

6 Working with remote repositories

Two main operations: pull and push

- Pull: get remote changes, integrate them in your local repository

```
$ git pull
```

- Push: send your local commits to the remote repository (requires write permissions in the remote repo)

```
$ git push
```

But first get a local copy of the remote repository (run only once, the very first moment you start with the repo)

```
$ git clone <url>
```

When you clone, the user you have will remain *sticked* in your local repo. You can change by editing `.git/config` in your local repo.

7 Usual workflow with Git

7.1 Adding a new feature when you own the repo

We have a mature project with a master branch where more than one people are touching and changing stuff.

We want to add a new feature to the project:

`$ git pull`
`$ git branch my_feature`
`$ git checkout my_feature`
(lots of hacking...)
`$ git commit -a` (now everything is in the $my_{feature}$ branch)
(make sure everything works; ready to be merged in master)
`$ git checkout master`
`$ git merge my_feature` (or rebase)
`$ git push`

7.2 Adding/fixing something when someone else owns the repo

All steps similar, except that you don't merge into master and then push the changes.

Instead, you generate patches and send the patches to the owner of the repo (through email, personally, or to a mailing list).

`$ git pull`
`$ git branch my_feature`
`$ git checkout my_feature`
(lots of hacking...)
`$ git commit -a` (now everything is in the $my_{feature}$ branch)
(make sure everything works; ready to generate patches)
`$ git format-patch`

This creates a file for each commit you have made in the branch. You can send those files through email. Actually, they *are* emails (have a look at a patch).

If the patches are accepted, you will eventually see your commits in *master*. **That's why you should not merge your branch back to master** (if you do it, there will be a conflict). The *master* branch should always be immaculate, and synchronized with upstream.

7.3 Integrating patches from third parties

If you get a patch from someone, properly formatted using *format-patch*, you should apply them using `git am`.

This *merges* the patches giving the proper credit to the author of the patches. She will appear as *author* (different than *committer*) in the logs.

`$ cat *.patch > patches.mbox` `$ git am patches.mbox`
`$ git push` (commits are now published in the remote repo) `$ git clean -fd` (remove patches and untracked files)

Review all patches before committing!

7.4 What about Github?

Github is a hosting service for Git repositories, also offering projects management tools (wiki, issue tracking, downloads, etc.)

In Github, the usual workflow is:

- You fork the repository to your Github account
- You make the changes in your repo (in a branch)
- You create a pull request for the owner of the original repo
- The author integrates (or not) the changes by pulling from your repo
- You pull from the original repo to see your changes in your forked master branch

Date: 2014-03-25

Author: Israel Herraiz

Org version 7.9.3f with Emacs version 24

Validate XHTML 1.0