GIT Cheatsheet

Stephan Gabler

December 5, 2010

Abstract

Short reminder about **git** and explanation of the most basic commands

1 Overview - The Concept

The most common use-case of git is to have a central repository that every user connects to and downloads a personal working copy from. When changes are made to the working copy, they can be uploaded to the repository. If these changes conflict with changes other people may have uploaded since the last time you updated your working copy, git tries to merge these files and solve the conflicts. You can also go back in the history of you files and track who made which changes. This is the most common usage, but it is useful whenever you have text-files (code) which change regularly. Then you can have a local repository to track these changes and go back to older versions. A really nice introduction can be found on: `http://hoth.entp.com/output/git_for_designers.html`

2 Guidelines

only running code Never push code where you are not really really sure that it does what it is supposed to do to the central repository. Make as many commits as you want to your local repo, but only push it when you are sure it works.

code vs. results try to separate your code from the results. When someone wants to check out you code, he does not need megabytes of MATLAB figures.

comments comments are one of the most important part of the code. Not only for other people working with your code, also for you if you look at it after some time. So: One short description (maybe only one sentence) at the beginning of each file, explaining shortly what it is good for. Also short comment for every *block* (if, loop, try catch, etc..) in a file.

commit messages always write meaningful commit messages. One sentence summary, one empty line and then a longer explanation of you commit.

gitignore Use the file *.gitignore* so exclude auto-generated or useless files from you commit (for example the MATLAB .asv files).

3 Commands

If you want to use GIT on the terminal, these are the basic commands. I mention and explain them, because you'll find equivalents of this commands in every GIT GUI.

clone get a working copy of a existing project

pull get changes from the central repository

push send changes to central repository

add add your changes to the next commit

commit commit changes to your working repository

status list of changes on current repository

diff see changes between different versions

blame find out who made which changes

log history of changes

4 GUIs

Much more convenient is using GIT with a GUI. I had a short look over some programs and recommend the following.

TortoiseGIT (Windows) It is a very intuitive and easy to use freeware program that integrates into the Windows Explorer.

Website `http://code.google.com/p/tortoisegit/`

video I found a link to video tutorial which seams to be quite good although I think the guy is really annoying: `http://www.youtube.com/watch?v=Mdfi5FSUyXQ`

GitX (Mac) Nice and simple layout for doing commits.

Website `http://gitx.frim.nl/`

5 Help

Have fun with it, I hope it helps you a lot.

In case you have some problems and the weblinks and videos do not help you solving it, write me a mail: stephan.gabler@gmail.com