The Ultimate Git Guide

”How to Git Gut”

**About this guide:**

This is a overview of the most important commands needed for simple version control. There are a lot of other important commands – but these are not strictly necessary in this kind of setup and will therefore not be discussed to avoid confusion. The guide is only complementary – users will need a better understanding of Git for things to go smoothly (here I recommend the first 3 chapters of “Pro Git” – a free book that can be found on Google).

**Git strategy:**

My limited knowledge on Git have led me to what I believe will be the best strategy for our version control. I believe the best way to work from day 1 will be to keep the “origin/master” (master branch on remote depository) sacred. What I mean by this is that we don’t use this branch for anything but our most stable/finished work. Let’s say collaborator A begins to code a Python script and collaborator B begins to code a R script. They therefore each begin by creating a local branch for their work. When they are finished for the day, they “upload” each of their local branches to the remote depository. When they are finished with their script they merge their remote branches to the remote master branch (origin/master). Repeat! Alternatively we can use rebasing (creates much cleaner commit history) – this is something we will discuss. Personally I’m against it as (1) the commit history then does not record things how it actually happened and (2) we easily break shit/create a chaotic mess (see page 98 of “Pro Git” book) if our communication is not on point (which is likely to happen at least 1 time through the project).

**First-time Git setup (Git responsible):**

To spare some extra work this project is based on a clone of an already existing remote repository created on one of the project members GitHub account. The collaborators need to be invited on GitHub (settings 🡪 manage access).

Steps:

1. Create local copy of repository:
   1. git clone <https://github>.com/LouisThygesen/02466\_fagprojekt.git

**First-time Git setup (Collaborators):**

The first-time setup is very simple. As described below I recommend creating a GitHub account as you will use it for (at least) other projects. It makes more sense to setup your Git identity to be the same as your GitHub account. You might possibly have to use a password supplied by the Git responsible every time you want to commit (we’ll see).

Steps:

1. Download Git: <https://git-scm.com/>
2. Create GitHub account: <https://github.com/>
3. Configure Git
   1. View settings:
      1. git config –list
   2. Set identity (GitHub username and email):
      1. git config –global user.name “<username>”
      2. git config –global user.email <email>
4. Create local copy of repository:
   1. git clone <https://github>.com/LouisThygesen/02466\_fagprojekt.git

**Forbidden commands:**

In general, please don’t do anything in your local repository manually unless you are 100% sure that it will not fuck up e.g. tracking in either your local and remote repository (in this or any parallel timeline). The exact same statement is valid for interaction with the remote depository through GitHub (web-browser). Some things can be faster/more intuitive to do through GitHub (web-browser), but please discuss it with your collaborators first. Also always remember to create meaningful comments/descriptions for e.g. commits. This will help the project later (easy to hunt down correct older versions of files). Ideally a plan should be made between collaborators about a common notation form (e.g. all commits should start with a commit number and what changes has been made).

Commands’/Actions’ whose names shall not be spoken:

1. Renaming, moving or deleting files manually (always use special git bash commands)
2. Updating local depository using “git pull”. This command is typically the same as using “git fetch” followed by “git merge”. It takes what is in the remote and tries to merge it into your local depository. Unpredictable for beginners – better to use “git fetch” and “git merge”.

**Commands: Navigating Local depository**

Many of these commands are general commands that are also relevant when dealing with the remote repository:

1. See status of your local depository:
   1. git status
2. Add file/files to staging area (tracked files/to be committed)
   1. git add <filename> (specific file)
   2. git add . (all)
3. See changes (compare working directory with what is staged/added):
   1. git diff
4. See changes (compare staging area to last commit):
   1. git diff –staged
5. Commit staged files
   1. git commit -m ”Commit 12: Fix benchmarks for speed”
6. Delete non-staged file from working directory and from being tracked in the future by Git (warning: manual deletion creates problems):
   1. rm <filename>
   2. Usual commit command
7. Delete staged file from working directory and from being tracked in the future by Git:
   1. rm <filename> -f
   2. Usual commit command
8. Remove file from staging area but not from working directory. This is smart – allows you e.g. to make changes to file and then stage it afterwards (if you staged the file to early).
   1. itr m –cached <filename>
9. Move files between folders:
   1. git mv file\_from file\_to
10. Rename file:
    1. git mv <old name> <new name>
11. View commit history:
    1. See logs over all commits:
       1. git log (branch)
       2. git log –all (all branches)
    2. See logs over changes in all commits (branch):
       1. git log –patch
       2. git log –stat
    3. See tree visualization over commits (all branches):
       1. git log –all –graph
       2. git log –all –oneline –decorate –graph
    4. See logs over all commits within timespan:
       1. Since some point in time:
          1. git log –since=2.weeks
          2. git log –since=”2 years 1 day 3 minutes ago”
          3. git log –since=”2008-01-15”
       2. Until some point in time:
          1. Same but with ‘–until’
    5. See only commits that changed the number of occurrence of string (e.g. added or removed reference to a specific function):
       1. git log -S function\_name
12. Undoing things
    1. Redo commit (deleting last commit) – takes your staging area and uses it for the commit. If you haven’t made any changes since your last commit (e.g. running this command immediately after your previous commit) the only change to the last commit will be the commit message. Can also be used to add extra file or fix e.g. a spelling error (if you forgot it in the last commit).
       1. git commit –amend
    2. Unstage a staged file (warning)
       1. git reset HEAD <filename>
    3. Unmodifying a unstaged modified file (warning – deletes local file and replaces it with the last committed version of the file)
       1. git checkout -- <filename>

**Commands: Branching, merging and remote depository**

Some of these commands are specifically for navigating the remote depository. However, most of the rest of the commands are also very relevant for navigating the remote depository:

1. Create a new branch
   1. git branch testing
2. Switch to another branch (local or remote)
   1. git checkout testing
3. Merge branches (e.g. to merge is553-branch into master-branch checkout master-branch and then use the merge command):
   1. git merge iss53
4. Delete branch after merging (there is typically no need for it more + this command does not work if on un-merged branch)
   1. git branch -d iss53
5. Merge conflicts
   1. Occur if you have changed the same file differently in two branches you are merging. No merge happens before the conflict has been solved. Gits add standard resolution markers. Fix the conflicts (see “Pro Git” book) and merge again.
6. See branches (with last commit)
   1. git branch -all -v
7. See branches merged into the branch you are currently on (e.g. to decide if they are safe to delete):
   1. git branch –merged
8. See branched not merged into the branch you are currently on
   1. git branch –no-merged
9. Change branch name (warning: not master)
   1. Locally:
      1. git branch –move bad-branch-name corrected-branch-name
   2. Remote:
      1. git push –set-upstream origin corrected-branch-name
      2. git push origin –delete bad-branch-name
10. Push local branch to remote (for sharing)
    1. git push origin <branch>
11. Download local copy of new remote branch
    1. Fetch data about remote: git fetch origin
    2. Create local copy: git checkout -b <branch (local)> <remote>/<branch (remote)>
12. Show tracking branches (local branches that track a remote branch)
    1. git fetch –all
    2. git branch -vv
13. Delete remote branch
    1. git push <remote> --delete <branch>
14. Showing your remotes
    1. git remote -v
15. Fetching and pushing (the name of the remote is origin)
    1. git fetch <remote>
    2. git push <remote>
16. Inspecting remote
    1. git remote show <remote>

**Commands: Tagging**

1. Tagging commits
   1. git tag -a v1.4 -m “my version 1.4”
2. Sharing tags (git push does not transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them. The process is just like sharing remote branches)
   1. git push origin –tags
3. Remove tags (from local repository)
   1. git tag -d v1.4-lw
4. Remove tags (from remote server)
   1. git push origin –delete <tagname>
5. Check out tags (warning – don’t make any commits – they disappear)
   1. git checkout v2.0.0

**Rebasing:**

* Described from page 93 in “Pro Git” book.