Dr. Joschua Thomas
Simon-Liedtke

Git Documentation

Oslo, den May 2, 2020

Yoshi's Documentations

Meta

Foreword

This document contains some helpful commands that can be used in git.

Contents

Chapter 1

Setup

1.1 General settings

1.1.1 Changing the user and their options

On your PC/Mac:

```
git config --global user.email "email@example.com"
got config --global user.name "John Doe"
```

You can check that everything has been set correctly by typing

```
got config --global user.name
```

On Overleaf: Go to Account>Account Settings and add the email address. Confirm the address by opening the email that you received in your mailbox.

On GitHub: Go to Settings>Emails and add the email address. Confirm the address by opening the email that you received in your mailbox. Choose a primary email address(, which is the one that is going to show up in your commits). OBS: Make sure that you did not mark the "Keep my email address private"-option. The name (that will actually show up in your commits) can be changed in your GitHub profile.

1.1.2 .gitignore

In many cases, you will have files in your local repository that you do not want to have committed because they are not useful, they are experimental or too big. In those cases you might want to create a .gitignore file in your folder and commit it to the repository. Git will know automatically that all the fields that are listed in the .gitignore file are not supposed to be tracked for changes.

1.2 Cross-platform text documents

In case you want to version control a text document with GitHub, follow this tutorial.

1.2.1 Starting in Overleaf

Start in Overleaf, then push it to GitHub, and download from GitHub to your computer ($Overleaf \rightarrow GitHub \rightarrow PC/Mac$).

1. Make a new project in Overleaf.
2. Choose the GitHub option in the menu and choose "Create a GitHub repository". Choose a name and add a description if wanted.
3. Choose the GitHub option in the menu and choose "Push Overleaf changes to GitHub".
4. Switch to your GitHub account and check if everything showed up correctly.
5. Open a terminal on your computer, create and navigate to the folder in which you want to clone the repository.
6. Obtain the url to the repository on the GitHub page under "Clone or download" and write the following lines in the terminal:

   ```
   git clone <path/to/github/repository> <folder>
   ```

   (If no folder is specified, it will take the repository name as default name.)

1.2.2 Starting on GitHub

Start on GitHub, then pull it from Overleaf and your PC/Mac ($Overleaf \leftarrow GitHub \rightarrow PC/Mac$).

1. Make a new repository on GitHub.
2. Copy the repository address on GitHub.
3. On your PC/Mac, navigate to the folder where you want to have the repository (or its parent folder).
4. Clone the GitHub repository into a new folder with the name of the git repository or into a specified folder:

   ```
   git clone <url-to-repository>
   git clone <url-to-repository> <path/name/of/folder>
   ```

5. Get all branches and tags:

   ```
   #git fetch origin
   ```

6. Check that all branches have been fetched:

   ```
   #git branch -a
   ```

7. Go to Overleaf and chose "Import from GitHub" under "New Project".
8. Choose the repository you want to import.

1.2.3 Starting on your PC/Mac

Start on your PC/Mac, then push to GitHub, and pull it from GitHub from Overleaf ($PC/Mac \rightarrow GitHub \rightarrow Overleaf$).

1. Create a new project on you PC/Mac:

   ```
   git init
   ```

2. Add files, make changes, commit to local repository.
3. Create a new and empty repository on GitHub. (No README file!)
4. Copy repository address.
5. Navigate to the git repository folder on your computer.
6. Remove existing origins from the repository:

   ```
   git remote rm origin
   ```

7. Add the GitHub repository as a new origin to your local repository on your PC/Mac:

   ```
   git remote add origin <url-to-repository>
   ```

8. Push all changes from your PC/Mac to GitHub including all branches and flags:

   ```
   git push --all origin
   git push --tags origin
   ```

9. Go to Overleaf and chose "Import from GitHub" under "New Project".
10. Choose the repository you want to import.

If you want to migrate a repository from an existing remote to a new remote, you can use the following commands (based on the following tutorial): Move repository from Cloudforge to Github.

1. Open a terminal and go to the folder the repository is located in.
2. Add a new remote origin:

   ```
   git remote add new-origin <path-to-new-repository>
   ```

3. Push all branches and tags to the new repository:

   ```
   git push --all new-origin
   git push --tags new-origin
   ```

4. Show existing remotes

   ```
   git remote -v
   ```

5. Remove old remote repository:

   ```
   git remote rm origin
   ```

6. Rename new remote repository to origin

   ```
   git remote rename new-origin origin
   ```

1.2.4 Pulling and pushing to two remotes for a project

From GitHub to GitLab

Unfortunately pull mirroring from GitHub to GitLab is not available in the free version of GitLab, so we have to do it manually. This means, we have to have implement a connection layer (the local repository) that is reponsible for updating between GitHub and GitLab. The sources can be found here and here. You can check the correct remotes with

```
git remote -v
```

1. On GitLab: Import existing GitHub project to GitLab by going to: New project > Import project > GitHub. Click on the Import button next to the desired project.
2. On your local computer: Add the GitLab repository as as push remote:

   ```
   git remote set-url origin --add https://gitlab.com/joschuaos/yoshis-hjem.git
   ```

3. On your local computer: Add the GitLab repository as a pull remote:

   ```
   git remote add origin-gitlab https://gitlab.com/joschuaos/yoshis-hjem.git
   ```

4. Update changes from you local computer:

   ```
   git push
   ```

5. Update changes fro your main remote, i.e. GitHub, through your local computer:

   ```
   git pull
   git push
   ```

6. Update changes from you secondary remote, i.e. GitLab, through your local computer:

   ```
   git fetch origin-gitlab
   git merge origin-gitlab/master
   git push
   ```

Chapter 2

Maintaining the code

Before committing new code, always make sure to update your local version from origin first!

2.1 Cross-platform text documents

2.1.1 From Overleaf

1. Make your changes and save with Ctrl + s / Cmd + s.
2. Update your Overleaf repository from origin by choosing the GitHub option in the menu and choosing "Pull GitHub changes into Overleaf".
3. Choose the GitHub option in the menu and choose "Push Overleaf changes to GitHub". Write a comment and commit.

2.1.2 From your PC/Mac

All commands can be done from the command line after navigating to the folder containing the git repository:

```
cd <path\to\github\repository>
```

1. Make your changes and save them.
2. Update your local repository from origin:

   ```
   git pull
   ```

3. Stage the file(s) that have changes.

   ```
   git add <name-of-file>
   ```

4. Commit the changes to the local repository with a change comment:

   ```
   git commit -m "Some useful and readable comment."
   ```

5. Push the changes to origin.

   ```
   git push
   ```

2.2 General commands

2.2.1 Renaming version-controlled files

If we want to change the name of a version-controlled document whil still preserving the history of its content, we can use the following command:

```
git mv old_filename new_filename
```

Then, commit and push the changes as always.

2.2.2 Moving version-controlled files

If we want to move a version-controlled document whil still preserving the history of its content, we can use the following command:

```
git mv filename dir
```

Then, commit and push the changes as always.

2.2.3 Removing/Deleting files

Remove a file from the repository by using:

```
git rm <file/folder>
```

If you want to keep it in the file system, while removing it from the repository, you can use:

```
git rm --cached <file/folder>
```

If you want to delete a file recursively, you can use:

```
git rm -r <folder>
```

If you want to see what would be deleted use:

```
-n
--dry-run
Don't actually remove any file(s).
Instead, just show if they exist in the index
and would otherwise be removed by the command.
```

2.2.4 Bash commands on Mac

Move multiple files as described here:

```
for FILE in folder/*.ext; do git mv $FILE new-folder/; done
```

Remove prefix from filename as described here:

```
for FILE in folder/*; do git mv "$FILE" "${FILE#prefix}"
```

2.2.5 Bash commands in Windows

You can use the following bash commands as described here:

```
for /r %i in (*) do echo %i
```

In bash scripts, we have to double the

```
for /r %%i in (*) do echo %%i
```

2.2.6 Removing files that are too big for the repository

When adding media files like mp3 the repository might increase in size drastically since on each new upload the current file is stored in the repository. At some point we MIGHT have to remove some of the files, and just calling $gitrm < fileurl >$ might not do the trick because the repository still contains the state of the previously committed and tracked media files. In that case we can follow the instructions described here:

```
git filter-branch --tree-filter 'rm path/to/your/bigfile' HEAD
git push origin master --force
```

If you encounter the following error

```
A previous backup already exists in refs/original/
```

You might have to call the command with the -f option as described here:

```
git filter-branch -f --tree-filter 'rm -rf path/to/your/bigfile' HEAD
```

2.2.7 Recover accidentally deleted files

We can restore accidentally deleted files as described here:

```
git checkout path/to/file-I-want-to-bring-back.sth
```

2.2.8 Show full history of moved files

You can see the complete log as described here:

```
git log --follow file
```

2.2.9 Creating new repository from folder of an existing repository

We start with this:

```
XYZ/
  .git/
  A/
  B/
  C/
```

but want to end up with this:

```
XYZ/
  .git/
  A/
  B/
C/
  .git/
  C/
```

Making new repository from a folder in an existing GIT repository following The Easy Way:

1. Prepare the old repository:

   ```
   pushd <old-repo>
   git subtree split -P <folder> -b <new-branch>
   popd
   ```

2. Create the new repository:

   ```
   mkdir <new-repo>
   pushd <new-repo>
   git init
   git pull <path/to/old-repo> <new-branch>
   ```

3. Link the new repository to Github:

   ```
   git remote add origin ...
   git push origin -u master
   ```

4. Cleanup if desired

   ```
   popd
   pushd <old-repo>
   git rm -rf <folder>
   git commit
   ```

2.3 Recurring Bugs

2.3.1 CRLF bug

In some cases, the following error might occur when trying to push/pull (a) commit(s):

```
LF will be replaced by CRLF.
```

A solution is described here:

```
git config core.autocrlf true
```

2.3.2 Ghost files in untracked files list

It could happen that the untracked files list shows files that do not really exist on the file system:

```
git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        file1.jpg
        file2.jpg
```

This can be solved by this solution:

```
git clean -f
```

Chapter 3

Diffs and conflicts

In some cases, it is very helpful to see differences between specific commits and/or the current version of the repository. Likewise, it is very normal that you will encounter some conflicts while working on a projects with multiple contributors. Don't worry and follow these steps.

3.1 Tools

When comparing two states in git, we can use the default commands to get a patch of the main changes:

```
git diff
git diff <sha-1> <sha-2>
```

If you have add the changes already you will have to call the following command:

```
git diff HEAD
```

This is very helpful, if we want to send a patch to a co-worker, for example.

However, this might be very clunky in some settings because we do not get a smoot visual representation of the changes. Thus, we might use common diff tools to visualize changes like TortoiseGit (Win), FileMerge (Mac) by using the following commands:

```
git difftool
git difftool <sha-1> <sha-2>
```

If you have add the changes already you will have to call the following command:

```
git difftool HEAD
git difftool --staged
```

3.1.1 FileMerge on Mac

If you are doing this for the first time, you might have to set up an appropritate diff tool. On Mac, you might want to use FileMerge that is already included on most MacOs's following these easy steps:

1. Define FileMerge for the mergetool:

   ```
   git config --global merge.tool opendiff
   ```

2. Define FileMerge for the difftool:

   ```
   git config --global diff.tool opendiff
   ```

3. Suppress prompt at every comparison:

   ```
   git config --global diff.tool opendiff
   ```

3.1.2 TortoiseGit on PC

On PC, you might want use Tortoise following these easy steps:

1. Download and install TortoiseGit.
2. Locate the global .gitconfig file in your home folder.
3. Add the following lines to the config file:

   ```
   [diff]
     tool = TortoiseGitDiff
   [difftool]
     prompt = false
   [difftool "TortoiseGitDiff"]
     cmd = \"C:/Program Files/Path/To/TortoiseGitMerge.exe\"
    -mine:"$REMOTE" -base:"$LOCAL"

   [merge]
     tool = TortoiseGitMerge
   [mergetool "TortoiseGitMerge"]
     cmd = \"C:/Program Files/Path/To/TortoiseGitMerge.exe\"
    -base:"$BASE" -mine:"$LOCAL" -theirs:"$REMOTE"
    -merged:"$MERGED"
   ```

   You might have to adjust the executable path to match the actual location of the program.

3.2 General approach

You will most likely encounter a merge conflict, after calling git pull from the command line. You will recognize a merge conflict by a message similar to this one:

```
error: Your local changes to the following files would be
       overwritten by merge:
       main.tex
Please commit your changes or stash them before you merge.
Aborting
```

We can solve it by the following simple steps:

1. Save the local changes:

   ```
   git stash
   ```

2. Download the latest changes from the remote repository:

   ```
   git pull
   ```

3. Reload changes on the local copy:

   ```
   git stash pop
   ```

4. Resolve conflict by calling the mergetool

   ```
   git mergetool
   ```

5. Commit and push your changes to the repository.
6. Call the following command to get rid of all the intermediary .orig-files. (If you want to delete folders recursively you can add the -d option):

   ```
   git clean -f
   git clean -fd
   ```

If you have some merge conflicts although you did not really have any local changes (e.g. merging wrong branches locally and remotely), you can abort merging with the following command:

```
git merge --abort
```

3.3 How to use a pull request on Github

1. Update from latest comit on master

   ```
   git pull origin master
   ```

2. Make a new branch and push it to Github

   ```
   git checkout -b pull-request-demo
   git push origin pull-request-demo
   ```

3. Do some changes and commit them.
4. Go to Github > Pull requests > New pull request. Choose the branch you want to merge into master under the compare drop-down menue. Add title and description, and press "Create pull request".
5. Merge pull request if you are content with the changes. If it cannot be done automatically, we have to do it locally which requires some more effort.

Chapter 4

Branches

Branches can be helpful when we have a main project, from which we derive minor side projects. A branch can stand by itself or merged again with the main branch (*master*).

4.1 General commands

Make new branch:

```
git branch <branch-name>
```

Delete an existing branch:

```
git branch -d <branch-name>
```

Rename an existing branch:

1. Rename the branch in your local repository:

   ```
   git branch -m <old-name> <new-name>
   ```

2. Push the new branch to remote:

   ```
   git push origin :<old-name> <new-name>
   ```

3. `git push origin -u <new-name>`

In some cases we want to merge to branches into each other, e.g. branch (b) into (a), as explained here:

1. Switch to branch (a):

   ```
   git checkout a
   ```

2. Merge branch (b) into (a):

   ```
   git merge b
   ```

3. Commit your changes:

   ```
   git commit -a
   ```

4.2 Recurring bugs

4.2.1 Not currently on a branch

"Git push master fatal: You are not currently on a branch" Explanation here or here. General idea:

```
git branch <tmp-branch>
git checkout master
git merge <tmp-branch>
git push origin master
```

4.2.2 Not tracking the remote branch

The local branch is not tracking the remote branch anymore. When you use gitx or gitk to browse the repository, you will notice that the different branches have a label for the local and the corresponding remote branch. If a branch is not tracking its remote branch anymore, you will only see the local label. In this case, you will have to set the upstream tracking again, using the following command:

```
git branch --set-upstream-to=origins/pupsi pupsi
```

4.2.3 Incompability of Overleaf with branches

In some projects, we might have various versions of the same document throughout different branches. Unfortunately, Overleaf is not able to track different branches in git.

Thus, I propose to create the constant document as a placeholder file which imports the various versions from separate files. In practice, we can create individual files for the version files on each branch and import them into the placeholder file. This will make it easier to merge each branch with the master. Once we merged all branches with the master, we can link it problem free with the Overleaf document and (un)comment the versions that we do (not) want.

For merging, we follow these steps:

1. Switch to master branch:

   ```
   git checkout master
   ```

2. Merge all changes from branch b into master:

```
git merge b
```

3. Commit your changes:

```
git commit -a
```

Chapter 5

Tags

Tags can be added to highlight specific releases. The can represent milestones, versions, etc.

5.1 General commands

Make a new tag:

1. `git tag <name>`
   `git tag <name> <short-sha>`

2. `git push origin <name>`
   `git push origin --tags`

Delete an existing tag:

1. `git tag -d <name>`

2. `git push origin :refs/tags/<name>`

Rename an existing tag:

1. `git tag <new-name> <old-name>`

2. `git tag -d <old-name>`

3. `git push origin :refs/tags/<old-name>`

4. `git push --tags`

Remind all co-workers to run the following command:

```
git pull --prune --tags
```

Chapter 6

LaTeX

This chapter contains some extra gimmicks for LaTeXdocuments. It is going to stay in this Git-documentation until we find a better location.

6.1 How to create hard links / symlinks for the build folder

The goal is to create a build folder that is added to .gitignore, in which we compile the PDF document of the LaTeXproject. The build folder is no ordinary folder but a hard link (win) or symlink (mac) that links to another folder in which we gather the PDFs of multiple projects. In that way we can publish PDFs of otherwise private LaTeXproject.

6.1.1 Hard links on Windows

Create a directory junction as described here:

```
MKLINK /J Link Original
```

6.1.2 Symlinks on Mac

Create a symmlink as described here:

```
ln -s original link
```

OBS: Use absolute url on Mac!

6.2 Conditionals

We can define variables in the preamble and then check if the variable has been set anywhere inside the document:

```
\def\var1{1} % You can set the variable to true (1) or false (0)

\begin{document}
```

```
\if\var1
... % Do something if var1 has been set to 1.
\else
... % Do something else if var has been set to 0.
\fi

\end{document}
```