1 Using git source control

The following chapter will show you the basic usage of git via e.g. the terminal on Ubuntu. On Windows the console emulator *cmder*. The Git & GitHub Tutorial for Beginners by *The Net Ninja* on Youtube is recommended.

1.1 Git Version

Using `git --version` shows the installed git version.

1.2 Configuration

To show the user name and the email address use

```
git config user.name
git config user.email
```

You can edit these values via

```
git config --global user.name <a username>
git config --global user.email <an email>
```

1.3 Git Repositories

There are basically two ways to create a git repository (repo):

- initialize a new repo
- clone an existing repo

The command

```
git status
```

shows information of the active branch, e.g. untracked or edited files or files in staging area.

1.3.1 Initialize a repo locally

Open the desired folder, which should represent the repo, in the terminal.

```
git init
```

creates a repository in the current directory. The folder `.git`, containing all source control information, will be created[1].

1.3.2 Adding files to staging area

Before a file or multiple files can be commited, they have to be added to the staging area. Using `git add <a file>` adds a file to the staging area. Usually, you want to add all untracked files at once.

```
git add .
```

adds all untracked files to the staging area. Again, you can check the files inside the staging area using `git status`. In rare cases a file has to be removed from the staging area. Use `git rm <a file>` in this case.

1.3.3 Commiting files

```
git commit -m "<a comment>"
```

[1] Files and folders starting with a dot may be hidden in the file manager (or explorer on windows).

commits files in staging area to active branch. Pay attention that the comment is enquoted.

Using `git log` lists information about all commits and branches. `git log --oneline` is a condensd version of `git log`, e.g. only one line per commit. After you entering one of this command the *less* program may be executed, which shows the logs in a scrollable view. If you want to exit this program you have to hit `q`.

1.3.4 Branches

Usually, if you are working on a project containing version control, you create a new *branch* and edit, add or remove existing files inside this branch. After you a finished with the current task, the used branch gets merged into the main branch, mostly this branch is called the *master* branch.

```
git branch -a
```

shows all branches.

```
git branch <a branch>
```

creates a new branch from the active branch (generelly from the master branch).

```
git checkout <a branch>
```

switches to another branch.

```
git branch -d <a branch>
```

deletes a branch locally, must not be active branch.

```
git push origin --delete <a remote branch>
```

deletes a remote branch.

Using

```
git checkout -b <a branch>
```

creates a new branch and immediatly switches to this new branch.

1.3.5 Merging Branches

You have to switch to a branch in which another branch should be merged, mostly the master branch.

...

1.4 GitHub

As mentioned before there are two ways to create a git repo.

1.4.1 Push a local repo to GitHub

```
git push <an url> <a branch>
```

pushes a branch to a remote repo. The url is an HTTPS link from GitHub. Before pushing a branch check if everthing is commit using `git status`. Usually, we don't always want to type this address when we pull and push something from and to a remote repository. Therefore, an alias is set for this url using

```
git remote add <an alias> <a url>
```

An alias, often called *origin*, is added to the local repo. This will be automatically done if the repo is cloned from GitHub. For example the alias is used like `git push origin master`.

```
git remote -v
```

lists all aliases.

1.4.2 Clone a repo

```
git clone <an url>
```

clones the repository from e.g. GitHub into the current directory.

1.4.3 Pull a repo

If more then one person works on the same project containg a repo, you, always, want to be up to date of the remote repo before you make some changes to the project. For example, a coworker has pushed changes to the remote repo. You don't always have to clone the remote repo if someone has made a change. Using

```
git pull origin master
```

fetches the changes from the remote repo.

1.5 Working in a group

1. `git pull origin`, fetches changes from other users
2. create new branch from master, `git checkout -b <a new branch>`

3. edit, add or remove files
4. stage `git add .` and commit `git commit -m "<a message>"`
5. push branch to GitHub, `git push origin <the new branch>`
6. create a pull request for this new branch
7. users with permissions may merge the pull request
8. after the branch (pull request) is merged, the branch can be deleted

1.6 Others

1.6.1 Stashing

If you have not finished all work in a branch and you want to checkout and work in another branch, DO NOT COMMIT UNFINISHED WORK! You can stash the unfinished work and switch to another branch. The stashed work can be finished later.

```
git stash list
```

lists all stashes

```
git stash / git stash push
```

moves stash on stack

```
git stash apply
```

get last pushed stash

`git stash pop`

pop last pushed stash

`git stash drop`

remove last pushed stash

1.6.2 Forking

used when to collaborate in an open source project which you don't own.

...

1.6.3 Deleting branches

`git branch | grep -v "master" | xargs git branch -D`

delets all branches except for the master branch.

2 Useful terminal commands

`cd`	change directory
`cd /d Y:`	switch directory to other network drive (e.g. `Y:`)
`ls`	lists all files and folders of current directory
`mkdir`	make new directory
`touch <a file>`	create a new file
`atom <a file>`	opens a file via atom
`rm <a file`	removes a file
`rmdir <a dir>`	removes a directory
`net use Y:`	check if drive is mapped, if not the network path doesn't exist