Notes about Git

Log of changes

History

```
git log
```

History specific file

```
git log path/file
```

Quick summary

```
git log --oneline
```

Detailed

```
git log --stat
```

Content changed

```
git log --patch
```

Graph plot of the commits

```
git log --graph --all --decorate --oneline
```

Observe changes

Show an specific commit

```
git show commit_hash
```

A way to refer to specific commit (Relative) HEAD is equivalent to the last commit HEAD~1 previous last etc.

```
git show HEAD~1
```

Between to commits (last and two steps before)

```
git diff HEAD..HEAD~2
```

See changes in a specific file

```
git annotate file1.txt
```

To color changes in a different ways

```
git diff --color-words
git diff --word-diff
```

You can also make differences between your last unstaged change and your last change as

```
git diff HEAD
```

Amount of lines changed

```
git diff --stat
```

Compare with respect to the actual head

```
git diff -r HEAD path/to/file
```

To compare a file's current state to the changes in the staging area, you can use git `diff -r HEAD path/to/file`. The `-r` flag means "compare to a particular revision", `HEAD` is a shortcut meaning "the most recent commit", and the path to the file is the relative to where you are (for example, the path from the root directory of the repository).

Ignoring files

Ignoring files

```
touch .gitignore
git add .gitignore
git commit -m " Preparing to ignore temporary files"
vim .gitignore
```

Then edit the file

```
#Ignore these
.m
#Keep these
!.r
wq
In this case we can ignore files  or file extensions
# Ignore these
# folders/filenames
pdf
data
# type
*.pyc
# Keep these
!.r
```

Configurations

Git options configuration

- System - settings for this machine
- Global - settings for every project
- Local - settings for one specific projects

```
git config --list
git config --list --system
git config --list --global
git config --list --local
```

Removing

Remove untracked files Files in the repository but untracked (Trace)

```
git clean -n
```

Files in the repository but untracked (Trace & Delete)

```
git clean -f
```

Remove a file (Erases the file from the system)

```
git rm file1.txt
```

Stage deleted files: First manually delete the files then add the delete commit! (The command . means current directory)

```
git add -u .
```

Remove a file without removing it from file system

```
git rm --cached file1.txt
```

Undoing changes

This left the file as in the RESET (unstage from an add)

```
git reset HEAD - filename
```

Discard things you changed in a file by rewriting what there is in the branch

```
git checkout -- file1.txt
```

Branches

Branches: Create a branch / delete a branch

```
git branch  add-startup-scripts
git branch -d add-startup-scripts
```

Verify actual branch

```
git branch
```

Changing branches towards `name-branch`

```
git checkout name-branch
```

Move files

This moves files

```
git mv title.txt header/title.txt
```

Commit files renamed. So you basically move your files as you want (manually) then you type the command and he will commit automatically the files that were moved (The . command tells stars at the current working directory, recursively). This works even if you modify the file

```
git add -A .
```

Check commits on a single file (Short version)

```
git log --stat -- file1.txt
```

Check commits on a single file (Full history)

```
git log --stat -M --follow -- file1.txt
```

Check ignored files not included in .gitignore

```
git ls-files --others --ignored --exclude-standard
```

You can put the pointer in a specific branch in a specific commit

```
git log
# Take the commit you need  commit-code
git checkout commit-code
# This is called 'detached head' because actually is like going backwards some steps in  your commits
git checkout branch-name
# Brings you back to your last commit last branch
```

Merging

Merge: Merges a specific branch into the new one.

```
git checkout master
git merge new-branch
```

When there are conficts not to be solved right now

```
git merge --abort
```

Bring all commits to a particular branch

```
git checkout master
git merge --squash new-branch
```

Delete all commits from the original merged branch

```
git branch -d  clean
```

Pull

Pulling changes is straightforward: the command `git pull remote-name branch-name` gets everything in `branch` in the remote repository identified by `remote` and merges it into the current branch of your local repository. For example, if you are in the `quarterly-report` branch of your local repository, the command:

```
git pull thunk lastest-analysis
```

would get changes from `latest-analysis branch` in the repository associated with the remote called `thunk` and merge them into your `quarterly-report branch`. A way to check remotes in a repository

```
git remote -v
```