Git

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

```
git log / git log --oneline
```

Displays the last few commits

```
git --amend
```

If commit is already done, and there is another file to be added, git add <file-name>, then —amend (reverts the last commit), to commit all files together

```
git branch
```

Overview of all branches

```
git switch <branch-name> / git switch -c <branch-name>
```

First one switches between existing branches, second one creates new branch and moves HEAD to that branch

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

Deletes the branch (HEAD must be on another branch)

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

Merging two diverged branches

- Fast Forward merge: the parent branch does not change after the branching, while the child is ahead a few commits. With merging child to parent we sync all commits (HEAD must be at the parent/master, merge is done on the child branch)
- Merge with no conflicts: after branching changes are made both on child as well as the parent, but wile merging there are no conflicts, as changes do not overlap
- Merge with conflicts: changes, made on both branches, overlap and must be resolved manually

```
git diff / git diff HEAD / git diff —staged [file-name]
```

Lists all the changes in working directory that are not staged for commit

HEAD: lists all changes, staged or un-staged

—staged: lists the changes between staging area and last commit (shows what will be committed, if committed now

```
git diff branch1..branch2 / git diff commit1..commit2
```

Branch: comparison between two branches

Commit: comparison between two commits (commit hashes)

```
git diff HEAD HEAD~1
```

Comparing the current HEAD to the previous commit

Changes made on two different branches: if un-committed, the changes transfer to the other branch upon switching (if committed, they stay on their original branch)

```
git stash / git stash pop / git stash apply
```

Saves the uncommitted changes, without having to make a commit: undoes un-staged/staged changes in the working directory, so they do not transfer upon branch switching
pop: re-applies stashed changes to the working directory
apply: similar to pop, but enables to apply the stashed changes more times (pop can only re-apply it once)

```
git stash list / git stash apply stash@{1}
```

First one lists all the stashed changes, while the other one pops the second stashed change, if there are more

```
git stash drop stash@{2} / git stash clear
```

Deletes a particular or whole stash

```
git checkout <commit-hash> → git switch master HEAD
```

Travels back to the commit-hash (detached HEAD state): HEAD points to the referenced commit, enables to check the file few commit ago
Switch master HEAD then returns to the latest master branch reference

```
git checkout <commit-hash> → git switch -c <new-branch-name>
```

Branching off of a certain commit hash in the past (branch is based upon the HEAD, so all the changes after commit hash are not visible here)

```
git checkout HEAD~1
```

~1: refers to the commit before HEAD, ~2: would refer to 2 commits before HEAD
Referencing the commit without the hash id

```
git checkout HEAD <file> /git checkout — <file> / git restore <file-name>
```

Reverts the chosen file to the stage of the last commit, therefore it deletes all the changes made after the last commit

```
git restore —source HEAD~1 <file-name> → git restore <file-name>
```

Restoring the file back two commits prior to HEAD, changes, if uncommitted, are lost
Using ‘restore’ reverts the file back to the last commit, where HEAD is, while uncommitted changes are lost

```
git restore —staged <file-name>
```

If unwanted file was added with ‘git add’, using ‘restore’ removes it from staging, file is not tracked anymore

```
git reset <commit-hash> / git reset —hard <commit>
```

‘Reset’ undoes/deletes the commits after the commit hash, while not losing the changes in working directory
—hard: while reverting/deleting the commits, it also deletes the changes from working directory

```
git revert  <commit-hash>
```

Similar to reset, but instead of deleting the commits, it makes a new commit, reverting/undoing the unwanted commits, therefore does not delete the commits