**Viewing and Staging Changes**

List changed files in your working directory

$ git status

List changes to tracked files

$ git diff

List changes between staging and last version of tracked files (--staged is a synonym of --cached)

$ git diff --staged

View the changes in the last commit

$ git diff HEAD HEAD^

Add a file to the next commit

$ git add <file>

Add some of the changes (hunks) in <file> to the next commit

$ git add -p <file>

Add all current changes to the next commit

$ git add . / $ git add –all

Permanently mark a local file as unchanged

$ git update-index --assume-unchanged -- file

See all changes in a branch that came in with the last pull operation

$ git diff <branch>@{1} <branch>

**Create**

Clone an existing repository to your machine

$ git clone <url>

Create a new local repository

$ git init

**Commit**

Commit staged changes

$ git commit

Commit all local changes in tracked files

$ git commit -a

Change the last commit (Don‘t amend published commits!)

$ git commit –-amend

Commit with an inline message

$ git commit –m “<message>”

**Basic Commit History**

Show all commits, starting with newest

$ git log

Show changes over time for a specific file

$ git log -p <file>

Who changed what and when in a file

$ git blame <file>

**Branches & Tags**

List local branches

$ git branch

List all branches (including remote)

$ git branch -a

Switch to branch (automatically tracks remote)

$ git checkout <branch>

Create a new branch based on your

current HEAD

$ git branch <new-branch>

Create a new branch based on your current HEAD and switch to it

$ git checkout -b <new-branch>

Delete a local branch

$ git branch -d <branch>

Rename a local branch

$ git branch –m <old-name> <new-name>

Mark the current commit with a tag

$ git tag <tag-name>

**Remotes**

List all currently configured remotes

$ git remote -v

Show detailed information about a remote (local and remote branch listing, reference status)

$ git remote show <remote>

Add new remote repository

$ git remote add <shortname> <url>

Change a remote’s URL

$ git remote set-url <remote> <url>

**Rebase**

Rebase your current HEAD onto a branch

(Don’t rebase published commits!)

$ git rebase <branch>

Abort a rebase

$ git rebase --abort

Continue a rebase after resolving conflicts

$ git rebase –continue

Rebase by altering individual commits in the process / rewrite history

$ git rebase –i <base>

Apply an existing from to the HEAD

$ git cherry-pick <sha1>

Apply a range of commits to the HEAD

$ git cherry-pick <sha1>..<sha1>

**Network Operations**

Download all changes from <remote>,

but don’t integrate into HEAD

$ git fetch <remote>

Download changes and directly merge/integrate into HEAD

$ git pull <remote> <branch>

Publish local changes on a remote

$ git push <remote> <branch>

Push local changes to the tracked remote branch of the current branch

$ git push

Delete a branch on the remote

$ git branch -dr <remote/branch>

…or

$ git push <remote> :<branch>

…or

$ git push <remote> --delete <branch>

Publish your tags

$ git push –tags

Publish a newly created, local branch to a remote

$ git push –u <remote> <branch>

**Merge**

Merge a branch into your current HEAD

$ git merge <branch>

Merge a branch into your current HEAD, avoiding fast forward

$ git merge -–no-ff <branch>

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

$ git add <resolved-file>

…or if the conflicted file is no longer required

$ git rm <resolved-file>

**Patching**

Create a patch against a specified base

$ git format-patch <base> --stdout > <patch-name>.patch

Take a look at the change set in a patch

$ git apply --stat <patch-file>

Test if a patch is going to cause collisions

$ git apply --check <patch-file>

Apply a patch as the original sequence of commits that are packaged in it

$ git am <patch-file>

Apply a patch as the original sequence of commits that are packaged in it and keep the original timestamps

$ git am --committer-date-is-author-date <patch-file>

**Undo**

Discard all local changes in your working directory

$ git reset --hard HEAD

Discard local changes in a specific file

$ git checkout HEAD <file>

Revert a commit (by producing a new

commit with contrary changes)

$ git revert <commit>

Reset to a previous commit…

…and discard all changes since then

$ git reset --hard <commit>

…and preserve all changes as unstaged changes

$ git reset <commit>

…and preserve uncommitted local

changes

$ git reset --keep <commit>

Access the local action history (and potentially save lost work)

$ git reflog

Remove all untracked local files

$ git clean -f

Check which local files would be removed

$ git clean -n

**Logging**

Limit number of commits to be shown

$ git log -<limit>

Condense each commit to a single line

$ git log --oneline

Include which files were altered and the relative number of lines that were added or deleted from each of them

$ git log --stat

Display the full diff of each commit

$ git log -p

Search for commits by a particular author

$ git log --author="<pattern>"

Search for commits with a commit message that matches a pattern

$ git log --grep="<pattern>"

Show commits that occur between <since> and <until>. Arguments can be a commit ID, branch name, HEAD, or any other kind of revision reference

$ git log <since>..<until>

Only display commits that have the specified file

$ git log -- <file>

Draw a text-based graph of commits on left side of commit messages.

$ git log --graph

Add names of branches or tags of commits shown next to the graph

$ git log --graph --decorate

**Stashing**

Temporarily store all modified tracked files

$ git stash

Restore the most recently stashed files and throw away the stashed change set

$ git stash pop

Restore the most recently stashed files and keep the stashed change set

$ git stash apply

List all stashed change sets

$ git stash list

View contents of a stash change set

git stash show -p stash@{<stash id>}

Discard the most recently stashed change set

$ git stash drop

**Miscellaneous**

List all ignored files in this project

$ git ls-files --other --ignored --exclude-standard

Find the hash of the common ancestor of two commits

git merge-base --octopus <sha1> <sha1>

Show the contents of a commit or tag

$ git show <identifier>