Git Commands Cheat Sheet

Status etc.

Description	Command	Comment

- git log
- Head

Staging Area

Description	Command	Comment
Show status of the staging area	`$ git status`	
List all files in the staging area	`$ git ls-files`	
Add all changed files to the staging area	`$ git add .`	
Add the files "f1.txt" and "f2.txt" to the staging area	`$ git add f1.txt f2.txt`	
Remove all changed files from the staging area	`$ git rm .`	
Remove the files "f1.txt" and "f2.txt" to the staging area	`$ git rm f1.txt f2.txt`	

Commits

Description	Command	Comment
List all commits	`$ git log`	exit log via `q`
Commit changes with a commit message	`$ git commit -m "commit message"`	
Remove a commit (soft)	`$ git reset --soft HEAD~1`	remove the last n=1 commits

Description	Command	Comment
Remove a commit	`$ git reset HEAD~1`	also remove changed files from the staging area
Remove a commit (hard)	`$ git reset --hard HEAD~1`	also remove changes to files

Stash

Description	Command	Comment
Move uncommitted and unstaged changes to the stash	`$ git stash`	the current branch has no changes anymore; `git stash` can be applied repeatedly, thus creating more stashes
Add a message to a new stash	`$ git stash push -m "my message"`	the pushed message is shown next to the stash when running `$ git stash list`
Inspect the stashes	`git stash list`	the latest stash is always at the top and stash indices are shown
Move changes from the latest stash to the current branch	`$ git stash apply`	apply the latest stash
Move changes from the third stash to the current branch	`$ git stash apply 3`	apply the third stash
Move the latest (=> index 0) stash to our project	`$ git stash pop 0`	the according stash is deleted this can be run repeatedly
Delete a specific stash via its index	`$ git stash drop 0`	

Description	Command	Comment
Delete the entire stash stack	`$ git stash clear`	

Reflog

The reflog (*referenc log*) stores all changes that we have made in our git project (but only for 30 days). For example, the reflog can restore changes on a feature branch that has been deleted.

Description	Command	Comment
List all changes (≤ 30 days) of the current branch	`$ git reflog`	IDs, head indices, etc. are listed
hard reset to a given ID (`1b3ec1a`)	`$ git reset --hard 1b3ec1a`	detached HEAD create a branch
hard reset to a given head index (`3`)	`$ git reset --hard HEAD~3`	detached HEAD create a branch

Else

Description	Command	Comment
Remove untracked changes from "f1.txt"	`$ git checkout f1.txt`	checkout the latest commit of "f1.txt"
Remove all untracked changes	`$ git checkout .`	checkout the latest commit of all files
Remove all untracked changes	`$ git restore .`	checkout the latest commit of all files
Reset (soft) the head by 1	`git reset --soft HEAD~1`	keep changes in the staging area

Description	Command	Comment
commit		
Reset the head by 1 commit	`git reset HEAD~1`	remove (i) the latest commit and (ii) clear the staging area
Reset (hard) the head by 1 commit	`git reset --hard HEAD~1`	remove (i) the latest commit, (ii) clear the staging area, and (iii) remove uncommitted file changes
List all untracked files	`$ git clean -dn`	listed files would be deleted upon `$ git clean -df`
Delete all untracked files	`$ git clean -df`	first, check for untracked files via `$ git clean -dn`
Go to a previous commit with ID `commitID`	`$ git checkout commitID`	
Create a branch `myBranch`	`$ git branch myBranch`	
Merge `myBranch` into `main`	`$ git merge myBranch`	while on `main`
Create and change to a new branch `myBranch2`	`$ git switch -b myBranch2`	`$ git checkout -b myBranch2` is equivalent
List all local branches	`$ git branch`	
Go to the `main` branch	`$ git switch main`	the latest commit of the current branch is the `HEAD`
Checkout the specific commit `56afce`	`$ git checkout 56afce`	commit `56afce` has to be on the current branch

Merging

Description	Command	Comment
fast-forward merge *feature* onto current branch	`$ git merge –ff feature`	condition: no new commits on current branch
recursively merge *feature* onto current branch	`$ git merge –no-ff feature`	
squash commits on *feature* and merge onto current branch	`$ git merge ––squash feature –ff feature`	combines several commits on *feature* into one on the current branch

- Apart from `fast-forward` and `recursive` / `ort` (**o**stensibly **r**ecursive's **t**win) there are also the `resolve`, `octopus`, `ours`, and `subtree` strategies.
- Merge conflicts can appear – and then need to be resolved – when two persons work on the same file.
- See the official documentation for further details.

Cherry-picking

Description	Command	Comment
bring one spefific commit to the current branch	`$ git cherry-pick 9c89d4502`	`9c89d4502` is the ID of a commit on another branch

Tags

Description	Command	Comment
list tags	`$ git tag`	
add tag `version-1.0` to commit `03d2cf9`	`$ git tag version-1.0 03d2cf9`	

Description	Command	Comment
checkout a commit via its tag	`$ git checkout version-1.0`	detached HEAD
show information about tag `version-1.0`	`$ git show version-1.0`	
delete the tag `version-1.0`	`$ git tag -d version-1.0`	

Branches

Description	Command	Comment
Delete the branch `xyz`	`$ git branch -d xyz`	
Force deletion of branch `xyz`	`$ git branch -D xyz`	also works if `xyz` has not been merged, yet

- create a local branch
- create a remote branch
- create a local tracking branch

Working with Remotes

Description	Command	Comment
add a remote repo via HTTPS	`$ git remote add origin https://rst.com/uvw/xyz.git`	`origin` is a common name
push the local `main` branch to the remote	`$ git push origin main`	
locally pull the remote repo	`$ git pull`	corresponds to `$ git fetch` followed by

Description	Command	Comment
		`$ git merge` or `$ git rebase`
log remote shortcuts (names) and urls	`$ git remote -v`	
set remote main as upstream of local main	`$ git push --set_upstream origin main`	this only sets the upstream branch
push changes upstream	`git push origin main`	push changes to the remote repo
list *all* branches, including remotes	`$ git branch -a`	
update the remote tracking branch `main`	`$ git pull origin main`	corresponds to `$ git fetch` followed by `$ git merge` or `$ git rebase`
fetch changes from the remote	`$ git fetch`	unlike `$ git pull`, this involves no commit

Tracking Branches

Description	Command	Comment
create a tracking	`$ git branch --track localBranch origin/remoteBranch`	`localBranch` is local and tracks

Description	Command	Comment
branch		the remote `remoteBranch`
list all tracking branches	`$ git branch -r`	
push changes (from a tracking branch)	`$ git push`	
fetch changes (from a tracking branch)	`$ git fetch`	
pull changes (from a tracking branch)	`$ git pull`	`$ git pull` corresponds to `$ git fetch`, followed by `$ git merge` or `$ git rebase`.

Pull requests

Description	Command	Comment

Detached `HEAD`

Description	Command	Comment
Directly checkout commit `f3c499`	`$ git checkout f3c499`	`$ git log` will show a detached `HEAD` the

Description	Command	Comment
		checked out commit is not part of any branch
Turn a detached `HEAD` into a branch	`$ git branch detached-head-branch`	

Ignoring files via `.gitignore`

- Use a `.gitignore` file for specifying which files git should ignore.
- Large files can trigger usage limits and thus trigger `$ git push` to fail.
- Automatically generated files (e.g. `.eslintcache`) do not need to be tracked.
- A single `.gitignore` file in the root folder of the repo – next to the `.git` folder – is usually sufficient.
- Ignore a specific file by adding it to `.gitignore` in a separate line, e.g., `file1.txt`.
- Ignore all files of a specific file type by adding and entry with a `*` wildcard, e.g., `*.bin`.
- Make exceptions and do track files that would be ignored via `!`, e.g., `!important.bin`.
- Ignore the files in a specific subfolder via, e.g., `app/js/*`, where `js` is the subfolder.
- When in doubt, refer to the official `.gitignore` documentation.

Extra 1: other useful shell commands

Description	Command
Space for file names etc.	`$ \_`
Present working directory	`$ pwd`
List files in current folder	`$ ls`
List files in folder `abc/xyz`	`$ ls abc/xyz`
Create empty / update files `f1.txt` and `f2.txt`	`$ touch f1.txt f2.txt`
Create new folders `folder1` and `folder2`	`$ mkdir folder1 folder2`
Navigate into `path/to/folder`	`$ cd path/to/folder`
Go to the home directory	`$ cd ~`

Description	Command
Go to the root directory	`$ cd \`
Move files from `origin` to `destination`	`$ mv origin destination`
Remove a folder `folder`	`$ rm -r folder`
Copy a file from `origin/f1.txt` to `destination/f1.txt`	`$ cp origin/f1.txt destination/f1.txt`
Copy a folder from `origin` to `destination`	`$ cp -r origin destination`

Extra 2: useful links

- https://git-scm.com/ (git)
- https://git-scm.com/doc (git documentation)
- https://github.com/ (Github)
- https://docs.github.com/en (GitHub documentation)
- https://about.gitlab.com/ (GitLab)
- https://docs.gitlab.com/ (GitLab documentation)