**Setup and Basics**

* git config --local user.name “John Doe”
  + Sets “John Doe” as user name for current repo
* git config --local user.email “jdoe@gmail.com”
  + Sets “jdoe@gmail.com” as email for current repo
* git config --global user.name “John Doe”
  + Sets “John Doe” as user name globally.
* git config --global user.email “jdoe@gmail.com”
  + Sets “jdoe@gmail.com” as email globally
* git config --global core.editor “path/To/Editor”
  + Sets default text editor to specified editor
* git status
  + Checks if current directory is initialized as git directory
  + Reports changes on current branch
* git init
  + initialize current directory as a git repo
* git add <file1> <file2>
  + Adds file1, file2 from working directory to staging area
* git add <directory>
  + Adds directory from working directory to staging area
  + Use “.” in place of <directory> to add all (modified) files
* git commit
  + Adds all changed files from staging area to repo
  + Opens default text editor to record commit message
* git commit -m “message”
  + Adds all changed files from staging area to repo
  + Adds “message” as commit message
* git commit -a -m “message”
  + Shortcut for [git add + git commit]. Adds all changed files from working directory directly to repo
  + Adds “message” as commit message
* git log
  + Retrieves commit history
* git log --oneline
  + Retrieves commit history with commit messages showing only first line (separated by newline/return feed)

**Commits in details**

* git commit --amend
  + “Redo” the **previous commit**. Example including a file forgotten to add in previous commit or make changes to previous commit message
  + Example:
    - git commit -m “some commit”
    - git add forgotten\_file
    - git commit --amend
* [.gitignore]
  + Add files and directories in this file for git to ignore
  + This file should be placed in repos root folder
  + <https://gitignore.io/>

**Working with Branches**

* git branch
  + Retrieves list of all branches (local)
* git branch -r
  + Retrieves list of all branches from remote
* git branch <newbranchname>
  + Creates a new branch with name “newbranchname”
* git switch <branchname> / git checkout <branchname>
  + Switches to the branch with name “branchname”
* git switch -C <newbranchname>
  + **Creates AND Switches** to the branch with name “newbranchname”
* git switch --
  + Switch back to wherever HEAD previously was
* git branch -d <branchname>
  + Deletes branch with name branchname
  + Use -D instead of -d to force delete a branch
* git branch -m <newbranchname>
  + Renames **current branch** to newbranchname

**Merging Branches**

* git merge <branchname>
  + Merges branch with name branchname **into current branch**

**Comparing changes with Git Diff**

* git diff
  + Compares changes between Staging Area and Working Directory
* git diff HEAD
  + Compares changes between last commit and Working Directory
* git diff --staged / git diff --cached
  + Compares changes between Staging Area and last commit
* git diff <commit1> <commit2>
  + Compares changes between commit1 and commit2
* git diff HEAD [filename] / git diff --staged [filename]
  + Compares changes specific file with name filename
* git diff <branch1> <branch2>
  + Compares changes between branch1 and branch2

**Stashing**

* git stash
  + Stashes all uncommitted changes from current branch
* git stash pop
  + Pops previously stashed changes out of memory and applies to working directory
* git stash list
  + Lists all stashes in memory
* git stash apply
  + Pops previously stashed changes **without removing from memory** and applies to working directory
* git stash apply stash@{n}
  + Pops nth stash from memory and applies to working directory
* git stash drop stash@{n}
  + Removes nth stash from memory

**Undoing changes & travelling back in Time**

* git checkout <commit-hash>
  + Switches repo to the specified commit. Hash is first 7 digit of commit hash
  + Use git switch master/main to get out of detached head
* git checkout HEAD~n
  + Switches repo relative to nth commit before HEAD
* git checkout HEAD <filename> / git checkout -- <filename>
  + Discards changes from file filename and reverts it back to HEAD state
* git restore <filename>
  + Discards changes from file filename and reverts it back (to most recent commit)
* git restore --source HEAD~n <filename>
  + Discards changes from file filename and reverts it back nth commit before HEAD
* git restore --staged <filename>
  + Remove accidentally added file filename from staging area
* git reset <commit-hash>
  + Restores the entire repo back to specific commit
  + **The commits are deleted BUT the changes remain**
* git reset --hard <commit-hash>
  + Restores the entire repo back to specific commit
  + **The commits are deleted AND the changes as well**
* git revert <commit-hash>
  + Creates a new commit undoing the changes from given commit
  + The commits remain and the changes are undone on a new commit

**Github & Remote Basics**

* git clone <url>
  + Clones the repo from given url to current directory
* [Generating SSH Keys]
  + <https://docs.github.com/en/github/authenticating-to-github/connecting-to-github-with-ssh>
* git remote -v
  + List existing remotes for repo
  + Will be automatically setup if repo cloned from Github
* git remote add <name> <url>
  + Adds a remote to current repo from given url with alias as name
  + Name often is “origin”
* git remote rename <oldname> <newname>
  + Renames remote alias from oldname to newname
* git remote remove <name>
  + Removes remote named name
* git push <remote> <branchname>
  + Pushes branch branchname to remote of alias remote
  + Example: git push origin main

**Fetching & Pulling**

* git branch -r
  + Retrieves list of all branches from remote
* git switch <remotebranchname>
  + Creates local branch of same remotebranchname from remote and maps it to remote branch as well
* git fetch <remote>
  + Fetches branches and history from specified remote repo and **ONLY updates** remote tracking of branches
* git fetch <remote> <branch>
  + Fetches branches and history from specified remote branch and **ONLY updates** remote tracking of given branch
* git pull <remote> <branch>
  + Shortcut for [git fetch + git merge]. Pulls latest info from remote and merges to current branch
  + git pull will default remote to origin and branch to whatever connection is already configured on it

**Git Rebase**

* git rebase <branchname>
  + (When used instead of a merge commit) Rebasing rewrites history by creating new commits for each original <currentbranch> commits.

Takes commits from <currenbranch> and puts them at tip of <branchname>. Commits are moved around.

* git rebase -i HEAD~n
  + Enters interactive mode. Allows to edit commits, add files, drop commits, etc. The “n” specifies how many commits far to go back from HEAD.

**Git Tags**

* git tag
  + Print all tags in current repo.
* git tag -l “\*beta\*”
  + Print all tags that include “beta” in their name.
* git tag <tagname>
  + Create a **lightweight tag** with name of tagname referring to commit on which HEAD is pointing.
* git tag -a <tagname>
  + Create an **annotated tag** with name of tagname referring to commit on which HEAD is pointing.
* git tag <tagname> <commithash>
  + Tag a previous commit with hash commithash.
* git tag -a <tagname> <commithash>
  + (Annotated)Tag a previous commit with hash commithash.
* git show <tagname>
  + Shows data of an annotated tag.
* git checkout <tagname>
  + Switches repo to the given tag.
* git diff <tag1> <tag2>
  + View differences between tag1 and tag2.
* git tag -f <tagname>
  + Reuse a tag that is already referring to a commit.
* git tag -d <tagname>
  + Delete a tag.
* git push <remotename> --tags
  + Push all tags to remote branch remotename.
* git push <remotename> <tagname>
  + Push tag tagname to remote branch remotename.

**Git Reflog**

* git reflog <option> HEAD/<branchname>
  + Shows logs of reference (HEAD is default) or given branch.
  + Options include: show, expire, delete and exists.