Purpose of document: guide on lectures with commands used and description

| Command | Explanation | Remarks / Common arguments | Example |
| --- | --- | --- | --- |
| [git clone](https://git-scm.com/docs/git-clone) | Clone a remote repository given the URL | 99% of the time you will pass the repository URL as the only argument to git clone. | git clone https://github.com/Pierian-Data/Git-and-GitHub-Zero-to-Hero.git |
|  |  |  |  |

| Command | Explanation | Remarks / Common arguments | Example |
| --- | --- | --- | --- |
| [git add](https://git-scm.com/docs/git-add) | Stage files / add files to the index for subsequent committing | *git add -A* : Stage all files  *git add -u* : Stage all tracked files (i.e files which have been added before after they were altered)  *git add /path/to/file* : Stage the file identified by its path | git add ReadMe.md |
| [git status](https://git-scm.com/docs/git-status) | Lists all added, changed and newly created files. | Typically no arguments necessary. You can use *git status -s* to get a shorter version | git status |
| [git reset](https://git-scm.com/docs/git-reset) | Undo changes / unstage files /  go back to commit | *git reset* without any arguments unstages all added files but preserves all changes.  *git reset –hard* unstages all added files and deletes all changes you made since the last commit. CAUTION!  *git reset /path/to/file* to unstage a single file  *git reset –hard commitId* to jump back to the commit with *commitId* | git reset  git reset test.txt  git reset –hard 5b331f3 |
| [git restore](https://git-scm.com/docs/git-restore) | Unstage specific files / undo specific changes | *git restore –staged* to unstage added files  *git restore /path/to/file* to undo changes since last commit. Only possible when unstaged.  [Difference between reset and restore](https://stackoverflow.com/questions/58003030/what-is-the-git-restore-command-and-what-is-the-difference-between-git-restor) | git restore test.txt  git restore –staged test.txt |
| [git log](https://git-scm.com/docs/git-log) | Show commit history | Pass no arguments to get the full log of the corresponding branch or use filtering arguments such as –after, –author or -n  More information on formatting and filtering can be found [here](https://www.atlassian.com/git/tutorials/git-log) | git log –after=“2022-1-1”  git log –after="yesterday"  git log -n 10  git log –author=“Jose” |
| [git diff](https://git-scm.com/docs/git-diff) | Visualize changes | *git diff* to list all changes since the last commit (unstaged files)  *git diff –cached* for staged files  *git diff /path/to/file* for a single file  *git diff commitID1 commitID2* to compare between commits  *git diff branch1 branch2* to compare between branches | git diff –cached test.txt  git diff 4598 3g62 test.txt  git diff main development |
| [git commit](https://git-scm.com/docs/git-commit) | Commit changes after staging them | Typically you only use:  *git commit -m “Message”*  An empty message aborts the commit command.  If *-m* is not passed, git opens a text editor to write the message  *git commit --amend -m “Changed message”* to change the commit message of the previous commit | git commit -m “Updated ReadMe.md”  git commit --amend -m "your new message" |
| [git push](https://git-scm.com/docs/git-push) | Push new commits to the remote repository | *git push* to push to the branch you are currently on and the remote repository defined in .git/config  *git push repository* to push to a different remote repository  *git push repository sourceBranch* to push the desired branch (i.e not one you are currently on)  *git push repository sourceBranch:targetBranch* to push to the targetBranch from the sourceBranch  *git push –force* to force push your current commit ignoring potential conflicts | git push  git push origin  git push origin main  git push origin main:test |

| Command | Explanation | Remarks / Common arguments | Example |
| --- | --- | --- | --- |
| [git branch](https://git-scm.com/docs/git-branch) | “List, create, or delete branches” | *git branch* to list all branches  *git branch name* to create a new branch called *name*  *git branch –delete name* to delete the branch called *name* | git branch  git branch development  git branch –delete development |
| [git switch](https://git-scm.com/docs/git-switch) | Switch to another branch | *git switch name* to switch to the branch name  *git switch -c na*me to create the branch name if it does not exist and switch to it  *git switch -d commitId* to switch to a previous commit  *git switch -m name* merges the changes of the current branch into name and switches to *name* | git switch development  git switch -c development2  git switch -d h98uab  git switch -m main |
| [git checkout](https://git-scm.com/docs/git-checkout) | “Switch branches or restore working tree files” | *git checkout name* to switch to the branch name  *git checkout -b name* to create the branch name if it does not exist and switch to it  *git checkout commitId* to switch to a previous commit  *git checkout -m name* merges the changes of the current branch into name and switches to *name*  *git checkout /path/to/file* to undo changes since the last commit (i.e git restore)  [Difference switch and checkout](https://stackoverflow.com/questions/57265785/whats-the-difference-between-git-switch-and-git-checkout-branch) | git checkout development  git checkout -b development2  git checkout h98uab  git checkout -m main  git checkout test.txt |
| [git merge](https://git-scm.com/docs/git-merge) | Merge / join two branches | *Typically done on github, but for the sake of completion:*  *git merge branch1 branch2* merges *branch1* and *branch2* on the current branch (i.e a new commit is created)  *git merge branch* to merge *branch* into the current branch  *git merge -s strategy branch* to define the merging strategy | git merge devel  git merge devel1 devel2  git merge -s ours devel |
| [git tag](https://git-scm.com/docs/git-tag) | “Create, list, delete or verify a tag” | Can also be done on github  *git tag* to list all tags  *git tag tagname* to create a tag called *tagname*  *git tag tagname -a* to add an annotated tag with the name *tagname* (will open editor)  *git tag –delete tagname* to delete the tag *tagname* | git tag v1.0  git tag v1.0 -a  git tag –delete v1.0 |
| [git fetch](https://git-scm.com/docs/git-fetch) | Fetch changes from the remote repository (does not update head) | *git fetch* to get the new commits from the branch you are currently on and the remote repository defined in .git/config  *git fetch repository* to update from a different remote repository  *git fetch repository sourceBranch* to get the desired branch (i.e not one you are currently on)  *git fetch repository sourceBranch:targetBranch* to get the sourceBranch into the targetBranch | git fetch  git fetch origin  git fetch origin main  git fetch origin main:test |
| [git pull](https://git-scm.com/docs/git-pull) | Update local version with remote version.  git fetch + git merge | *git pull* to pull from the branch you are currently on and the remote repository defined in .git/config  *git pull repository* to pull from a different remote repository  *git pull repository sourceBranch* to pull the desired branch (i.e not one you are currently on)  *git pull repository sourceBranch:targetBranch* to pull the sourceBranch into the targetBranch | git pull  git pull origin  git pull origin main  git pull origin main:test |

| Command | Explanation | Remarks / Common arguments | Example |
| --- | --- | --- | --- |
| [git rebase](https://git-scm.com/docs/git-rebase) | Rewrite commit history | *git rebase -i HEAD~n*  to rebase the last *n* commits in the interactive mode  *git rebase main* to rebase *main* on the current branch  *git rebase –onto newbase oldbase*  for more advanced rebasing with specific branches  More in depth guide [here](https://git-scm.com/book/en/v2/Git-Branching-Rebasing), [here](https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase)  Caution: Do not rebase after pushing! | git rebase -i HEAD~5 |
| [git revert](https://git-scm.com/docs/git-revert) | Revert existing commits and create new commit with these changes | *git revert commitId* creates a new commit containing the state of *commitId.*  The editor will be opened to enter the commit message  [Difference revert reset](https://stackoverflow.com/questions/8358035/whats-the-difference-between-git-revert-checkout-and-reset) | git revert 2fc0df |

| Command | Explanation | Remarks / Common arguments | Example |
| --- | --- | --- | --- |
| Day 4 |  |  |  |
| [git stash](https://git-scm.com/docs/git-stash) | Stash changes for later use | *git stash* to add a new stash entry with the current modifications and reset your state to the current HEAD  *git stash list* to get all stash entries  *git stash show* to visualize the changes (diff)  *git stash pop* to pop the first element of *git stash list*  *git stash pop stash@{i}* to get the *ith* element of the stack  *git stash apply* works similar to *git stash pop* but does not remove the stash from the list  [More examples](https://www.atlassian.com//git/tutorials/saving-changes/git-stash) | git stash  git stash list  git stash show  git stash pop  git stash pop stash@{2} |
| [git clean](https://git-scm.com/docs/git-clean) | Delete all files not tracked by git | *git clean* to recursively remove all files not tracked  *git clean -n* to list the files which would be deleted  *git clean -x to also delete files ignored via .gitignore*  *git clean -X to only delete ignored files* | git clean  git clean -n  git clean -x |