**Git Reference**

By: supersimple.dev

Tutorial link: https://www.youtube.com/watch?v=hrTQipWp6co

**Command Line (Terminal / PowerShell)**

ls

cd ~/Desktop/folder

List the files and folders in the current folder

Change the folder that the command line is

running in

Note: git commands must be run inside the folder that contains all the code.

**Creating Commits**

In Git, version = commit

Version history = commit history

git init Git will start tracking all changes in the current folder

git status Show all changes since the previous commit

git add <file|folder> Pick changes to go into next commit

git add file

git add folder/

Pick individual file

Pick all files inside a folder (and subfolders)

git add . Pick all files (in folder command line is running in)

git commit -m "message" Creates a commit with a message attached

git commit -m "message" --amend Update previous commit instead of creating new one

git log View the commit history

git log --all Show all commits (not just current branch)

git log --all --graph Show branching visually in the command line

**Configure Name & Email for Commits**

git config --global user.name "Your Name"

git config --global user.email [email@example.com](mailto:email@example.com)

**Working Area = contains changes start in the working area**

**Staging Area = contains changes that will go into the next commit**

git add . working => staging

git commit -m "message" staging => commit history

git reset <file|folder>

git reset file

git reset folder/

git reset .

staging => working

git checkout -- <file|folder>

git checkout -- file

git checkout -- folder/

git checkout -- .

working => remove the changes

**Viewing Previous Commits**

git checkout <commit\_hash|branch\_name> View a previous commit

master = branch name

1. You can git checkout branch

2. Always points to latest commit

on the branch.

HEAD = indicates which commit

you are currently viewing

**Restoring to a Previous Commit**

git checkout <hash|branch> <file|folder> Restore the contents of files back to a

previous commit

git checkout <hash|branch> file Restore a file

git checkout <hash|branch> folder/ Restore all files in folder (& subfolders)

git checkout <hash|branch> . Restore all files in project

**Other Features of Git**

git config --global alias.shortcut <command> Creates an alias (a shortcut)

git config --global alias.s "status" git s = git status

.gitignore Tell git which files/folders it SHOULD NOT track

rm -rf .git Remove git from project

**GitHub**

Repository = a folder containing code where any changes to the code are tracked by git.

(To create a repository, we create a new folder on our computer, and then run git init)

GitHub = a service that lets us save our git repositories online. It also helps us:

- backup our code in case we delete it on our computer

- see the history of our code changes more easily

- alternatives include Bitbucket and GitLab

Local repository = a git repository saved on our computer

Remote repository = a git repository saved online (for example on GitHub)

**Uploading Code to GitHub**

git remote add <remote\_name> <url> Link a local repository to a remote repository and

give a name for this link

git remote add origin https://github.com/SuperSimpleDev/repository1

^

The above command links a local repository to a GitHub repository (located at the url

https://github.com/SuperSimpleDev/repository1) and gives it a name "origin"

git remote List all remote repositories that are linked

git remote -v List all remote repositories (but with more detail)

git remote remove <remote\_name> Removes a link to a remote repository

git remote remove origin Removes the link to the remote repository named

"origin"

git config --global credential.username <username>

Configure your GitHub username so you can get

access to your Github repository

git push <remote\_name> <branch> Upload a branch of your git version history to your

remote repository

git branch Shows a list of available branches

git log --all --graph Shows the branches visually in the history

git push origin main Upload the branch "main" to the remote repository

named "origin"

git push <remote\_name> <branch> --set-upstream Sets up a shortcut for this

branch and remote repository

git push origin main --set-upstream Next time you are on the main

branch and you run git push, it

will automatically push the

main branch to origin

git push <remote\_name> <branch> -f Force-push the branch to the remote repository (it

will overwrite what's on the remote repository)

**Downloading Code from GitHub**

git clone <url> Download a remote repository from a url

git clone https://github.com/SuperSimpleDev/repository1

git clone <url> <folder\_name> Download the repository and give it a different

folder name

git fetch Updates all remote tracking branches. Remote

tracking branches (like origin/main) show what

the branch looks like in the remote repository

git pull <remote\_name> <branch> Update the local branch with any updates from

the remote repository (on GitHub)

git pull origin main Downloads any new commits from the main

branch on origin, and updates the local main

branch with those new commits

git pull origin main --set-upstream

^

Sets up a shortcut so that the next time you are on the main branch and run git pull, it will

automatically git pull origin main

**Branching**

Branching = create a copy of the version history that we can work on without affecting the

original version history. This lets us work on multiple things (features + fixes) at the same time.

git branch <branch\_name> Creates a new branch

git branch feature1 Create a new branch named feature1

git checkout <branch\_name> Switch to a different branch and start working on

that branch

git checkout feature1 Switch to the feature1 branch. New commits will

now be added to the feature1 branch

HEAD = points to which branch we are currently working on

HEAD -> feature1 = we are currently working on the feature1 branch. Any new commits will

be added to the feature1 branch

git branch -D <branch\_name>

git branch -D feature1

Deletes a branch

Deletes the feature1 branch

**Merging**

git merge <branch\_name> -m "message" Merge the current branch (indicated by HEAD ->)

with another branch (<branch\_name>). Saves

the result of the merge as a commit on the

current branch

git checkout main

git merge feature1 -m "message"

1. First switch to the main branch

2. Then merge the main branch with the

feature1 branch. The result of the merge will be

added to main as a commit (a "merge commit")

**Merge Conflicts**

<<<<<<< HEAD

code1

=======

code2

>>>>>>> branch

If there is a merge conflict (git doesn't know

what the final code should be), it will add this in

your code.

(This is just for your convenience, the <<<<<<<

and >>>>>>> don't have special meaning)

<<<<<<< HEAD

... <-- Code in the current branch (indicated by HEAD ->)

=======

... <-- Code in the branch that is being merged into HEAD

>>>>>>> branch

**To resolve a merge conflict:**

1. Delete all the extra code and just leave the final code that you want.

~~<<<<<<< HEAD~~

~~code1~~

~~=======~~ => code2

code2

~~>>>>>>> branch~~

2. If there are conflicts in multiple places in your code, repeat step 1 for all those places.

3. Create a commit.

git add .

git commit -m "message"

**Feature Branch Workflow**

A popular process that companies use when adding new features to their software.

1. Create a branch for the new feature (called a "feature branch").

git branch new-feature

git checkout new-feature

Make some changes to the code...

git add .

git commit -m "new feature message"

2. Upload the feature branch to GitHub.

git push origin new-feature

3. Create a pull request on GitHub (a pull request lets teammates do code reviews and add

comments).

4. Merge the feature branch into the main branch (by opening the pull request in the browser

and clicking "Merge pull request")

5. After merging, update the local repository (so that it stays in sync with the remote repository

on GitHub).

git checkout main

git pull origin main

**Merge Conflicts in the Feature Branch Workflow**

A merge conflict can happen if 2 or more pull requests change the same file and the same line.

We can either:

1. Resolve the merge conflict on GitHub.

2. Resolve the merge conflict on our computer.

1) Get the latest updates from main

git checkout main

git pull origin main

2) Get the latest updates from the feature branch.

git checkout feature4

git pull origin feature4

3) Merge main into the feature branch (feature4). Notice the direction of the merge: we want

the merge commit to stay on the feature branch so our teammates can review it.

git checkout feature4

git merge master

4) Push the resolved feature branch to GitHub.

git push origin feature4

Now the pull request should be ready to merge again.