Section 8: Comparing Changes With Git Diff

1.What Really Matters In This Section

2.Introducing The Git Diff Command

* We can use the "git diff" command to view changes between commits, branches, files, our

working directory, and more!

3.A Guide To Reading Diffs

* Chunk header starts with @@ and ends with @@
* '-'&'+' denotes which file it is from. (- mostly A and + mostly B)
* "- content" deleted content and "+ content" added content

4.Viewing Unstaged Changes

* To view changes made in files use the diff command even in unstaged files.

5.Viewing Working Directory Changes

* "git diff HEAD" lists all changes in the working tree since your last commit.

6.Viewing Staged Changes

* "git diff --staged" command will show changes only from staged items.
* "git diff --cached" command will show the hanges in your working tree relative to the name <commit>

7.Diffing Specific Files

* "git diff --staged <files> …" show the changes in a specific file.

8.Comparing Changes Across Branches

* "git diff branch1..branch2" command will show changes of branch1 to branch2

9.Comparing Changes Across Commits

* "git diff <commit1\_id>..<commit2\_id>" command will show the changes in 2 different commits.

10.Visualizing Diffs With GUIs

* Changes can be seen in gitkraken under diff view tab.

Section 9: The Ins and Outs of Stashing

1.What Really Matters In This Section

2.Why We Need Git Stash

* git stashing is used to stash uncommitted files before changing branch and return to it later.

3.Stashing Basics: Git Stash Save & Pop

* "git stash" or "git stash save" command is used to stash all the uncommitted changes(staged or unstaged).
* "git stash pop" command is used to remove most recently stashed changes in your stash and re-apply them to your working copy.

4.Practicing With Git Stash

5.Git Stash Apply

* "git stash apply" command will apply the stashed changes to current branch and is useful to apply it in different branch also.

6.Working With Multiple Stashes

* "git stash list" shows the stash list.
* We can stash multiple times.
* "git stash apply stash@{n}" command to apply that particular stashed item.

7.Dropping & Clearing The Stash

* "git stash drop stash@{n}" command to remove that particular stashed items.
* "git stash clear" command to clear the stash

Section 10: Undoing Changes & Time Traveling

1.What Really Matters In This Section

2.Checking Out Old Commits

* "git checkout <commit-hash>" command to revert back to a previous commit in a detached head state

3.Re-Attaching Our Detached HEAD!

* Detached HEAD to examine contents of the old commit.
* Switch the branch to re-attach HEAD.

4.Referencing Commits Relative to HEAD

* "git checkout HEAD~1" command is used to checkout parent commit.
* "git checkout HEAD~n" command is used to checkout n-parent commit.
* "git switch -" command will take you to where you left off.

5.Discarding Changes With Git Checkout

* "git checkout HEAD <filename>" or "git checkout -- <filename>" command will revert the file back to whatever it looked like when it was last commited.

6.Un-Modifying With Git Restore

* "git restore <filename>" command will revert the file back to whatever it looked like when it was last commited.
* "git restore --source HEAD~n <filename>" command will revert the file back to whatever it looked like when it was commited at HEAD~n position.

7.Un-Staging Changes With Git Restore

* "git restore --staged <file-name>" command will unstage a file which was accidently staged.

8.Undoing Commits With Git Reset

* "git reset <commit-hash>" command will delete a commit till the entered commit hash. Changes are kept.
* "git reset --hard <commit-hash>" command will delete a commit till the entered commit hash and will revert all the files back to that commit stage.

9.Reverting Commits With...Git Revert

* "git revert <commit-hash>" command will revert back to that commit stage.
* If you want to reverse some commits that other people already have on their machines, you should use revert.

Section 11:Github: The Basics

1.What Really Matters In This Section

2.What Does Github Do For Us?

* Github is a hosting platform for git repositories. We can put your own Git repos on Github and access them from anywhere and share them with people

around the world.

3.Why should use github

* Backing up the code
* For collaboration
* Open source Projects
* Exposure your work to others
* Stay Up to Date

4.Cloning Github repos with Git clone

* To get a local copy Of an existing repository .
* we can clone a remote repository hosted on Github or similar websites. All we need is a URL that we can tell Git to clone for use.
* Command: git clone <url>

5.Cloning Non-Github Repos

* Anyone can clone a repository from Github, provided the repo is public. So for that we need only URL of the Github.
* We need permission to do that pushing our own changes to the Git hub repository.

6.Github Setup: SSH config

* We generate and configure an SSH key! Once configured , we can connect to Github without having username and password.
* ssh-keygen -t ed25519 -C "email id"=substitute in our github email adresss.
* ssh-keygen -t rsa -b 4096 -C "email id" In legacy system.