Section 5: Commits in Details (And Related Topics)

1.What really Matters in this section

2.Navigating The git Documentation

* Documentation is available in <https://www.git-scm.com/doc>

3.Keeping Your Commits Atomic

* Atomic Commits: When possible, a commit should encompass a

single feature, change, or fix. In Other words, try to keep each commit

focused on a single thing.

* Commit each and everything seperately to track the status.

4.Commit Messages:Present Or Past Tense?

* Many developers prefer past-tense in the commit message.

5.Escaping VIM & Configuring Git's Default

* Go to git's website and search for editor configuration and select a editor which is suitable for you.
* If commit command is used without -m option then instead of vim editor, your prefered editor will start

6.A closer Look at the git Log command

* Git log command has lots of option to print the log

|  |  |
| --- | --- |
| options | explaination |
| --all | You can force the log tool display all commits |
| -n | View n Most Recent Commits |
| --author <name>  --committer <name> | Filter Commits By Author or Committer |
| --before <date>  --after <date> | Filter Commits by X Days Ago |

7.Committing with a GUI

* Commit using gitkraken application.

8.Fixing Mistakes with Amend

* If we made a mistake in previous commit we can rectify it using --amend option.
* Add the file which was left out in staging area and commit using amend option.

9.Ignoring Files w/ .gitignore

* Ignoring Files :

We can tell Git which files and directories to ignore in a given repository, using a .gitignore file. This is useful for files you know you NEVER want to commit, including:

* Secrets, API keys, credentials, etc.
* Operating System files (.DS\_Store on Mac)
* Log files
* Dependencies 8 packages
* Create a file called .gitignore in the root of a repository. Inside the file, we can write patterns to tell Git which files & folders to ignore:

• .DS\_Store will ignore files named .DS\_Store

• folderName/ will ignore an entire directory

• \*.log will ignore any files with the .log extension.

Section 6: Working with Branches

1.What really Matters in this section

2.Introducing Branches

* In Git, branches are a part of your everyday development process. Git branches are effectively a pointer to a snapshot of your changes.

3.The Master Branch (Or Is It Main?)

* The master branch is a default branch in Git. It is instantiated when first commit made on the project. When you make the first commit, you're given a master branch to the starting commit point.

4.What On Earth Is HEAD?

* Head is simply a pointer that refers to the current "location" in your repository. It points to a particular branch reference.

5.Viewing All Branches With Git Branch

* Use git branch to view your existing branches. The default branch in every git repo is master,

though you can configure this.

* Active branch has a "\*" at the beginning.

6.Creating & Switching Branches

* Use "git branch <branch-name>" to make a new branch based upon the current HEAD
* This just creates the branch. It does not switch you to that branch (the HEAD stays the same)
* Once you have created a new branch, use "git switch <branch-name>" to switch to it.
* "git switch -c <branch-name>" command is used to create a branch and make HEAD point to it.

7.More Practice With Branching

* "git commit -a -m "<message>"" command is used to add all files to stage area and commit it.

8.Another Option: Git Checkout Vs. Git Switch

* "git checkout <branch-name>" command was previously used in git and is still usable.

9.Switching Branches With Unstaged Changes?

* If we switch between branches and if there is any change in the files, it will pop an error.

10.Deleting & Renaming Branches

* To delete a branch, you must not be on that branch.
* "git branch -d <branch-name>" command will remove that branch with prompt.
* "git branch -D <branch-name>" command will forcefully remove that branch.
* To rename a branch, move to that branch and use "git branch -m <new-name>" command.

11.How Git Stores HEAD & Branches

* Head is stored Inside .git folder -> Head file.

12.Branching Exercise



Section 7: Merging Branches, Oh Boy!

1. What Really Matters In This Section

2.An Introduction To Merging

* Merging: Branching makes it super easy to work within self-contained contexts, but often we want to incorporate changes from one branch into another!
* To merge branches, switch to receiving branch and use command "git merge <name>"

3.Performing A Fast Forward Merge

* Merging a branch to head after a lot of commit is known as fast forward merge.

4.Visualizing Merges

* Using gitkraken to see GUI of merging branches

5.Generating Merge Commits

* merge commit is one that has multiple parents and is displayed in GitX by the convergence of two or more branch tracks.

6.Oh No! Merge Conflicts!

* Depending on the specific changes you are trying to merge, Git may not be able to automatically merge. This results in merge conflicts, which you need to manually resolve.

7.Resolving Merge Conflicts

* Open up the file(s) with merge conflicts
* Edit the file(s) to remove the conflicts. Decide which

branch's content you want to keep in each conflict. Or

keep the content from both.

* Remove the conflict "markers" in the document
* Add your changes and then make a commit!

8.Using VSCode To Resolve Conflicts

* Vscode has feature to resolve conflicts with prompt buttons.