Extra topics

Summary:

1) Aliases
2) Stash
3) Restore and clean
4) Ignoring files and directories
5) Tag and release

1) Aliases

Definition

- It’s basically giving a certain command another name (alias)
- Example: **git config --global alias.st status**

Now: **git st** is equivalent to **git status**

Example 2: **git config --global alias.cm “commit –m”**

Now: **git cm “message”** is equivalent to **git commit –m “message”**

Note: it’s better when you make an alias to use two characters minimum and don’t make an alias of one character.

Alias list

- Commnad: git config --global --edit
- this command will show you the config file, if you made any aliases the result will be:

Alias list

- You can modify the config file using vim and add your aliases directly in the file.
- There are some alias lists made by programmers with experience so it’s better to search and paste them directly in the config file and use them.

2)Stash

What is stash?

In Git, the **git stash** command is used to save changes that are not ready to be committed, so they will be like “hidden in a stash or a box, when you type git status you won’t find them ”, allowing you to switch branches or perform other tasks without committing your work. Here's a brief explanation along with examples for various **git stash** commands:

1. Stash changes in the working directory: **git stash**
2. Stash changes with a descriptive message: **git stash save “message”**
3. Lists all stashes with their reference IDs: **git stash list**

Note: The list is like a stack (LIFO) so every command by default works on the last element in the stash list

More commands

4.Apply the latest stash and remove it from the stash list: **git stash pop**

5. Apply the latest stash without removing it from the stash list:

git stash apply / git stash apply stash@{id}

6. Apply a specific stash by its ID and remove it:

git stash pop stash@{id}

7.Drop a stash: **git stash drop / git stash drop stash@{id}**

8.Show changes in a stash: **git stash show / git stash show stash@{id}**

9.Erase all stashes in the stash list: **git stash clear**

3)Restore and clean

Removing files from staging area

Command: **git restore --staged filename**

Remove all files from staging area: **git restore –staged**

Note: you can remove files from staging area directly with vscode’s GUI

Clean:

The git clean command is used to remove untracked files from your working directory. Untracked files are files that are not part of the Git repository and are not being ignored by a .gitignore file. This command can be useful when you want to remove files that are not needed for your project.

Commands:

Show a list of untracked files: **git clean –n**

Delete untracked files: **git clean -f**

4)Ignoring files and directories

Some files like node modules , the .log files , ide config… shouldn’t be committed or pushed to your GitHub repo.

They need to be in the gitignore file

Commands:

touch .gitignore

And fill your file

***.log**

folderName/

…

Note: you can fill it manually but it’s better to search on git ignore patterns and paste them right away (or use ai)

5) Tag and release

Tag

Tag and release are used in versions, every version has its tag

So you can download each version separately

Commands:

git tag v1.0

git push origin v1.0

Unannotated tag:

git tag –a v2.0 –m “message”

git push origin v2.0

On GitHub you’ll see the message next to the tag

Tag

To see your tags: **git tag**

If you work and commit , the commit will be shown on all tags because we're working in one branch.

To list some tags: **git tag –l "v1.*"**

The result will be: v1.0, v1.1, v1,2 ...

Delete a tag:

From git (local): **git tag –d tagName**

From GitHub(remote): **git push origin --delete tagName**

Release

On github, you enter a certain tag

Create release, you give it a name a description …

Then publish your release

You can make a draft then publish it

You can compare also between releases on github