Git Assignment

> `Link: https://youtu.be/tRZGeaHPoaw?si=33PGntB3gKMHvqdB`
>
> Tags : Git, GitHub, git config, git environments, git commit, Git Repository, Git Bash, git status, git log, git branch, git Rebase,

How to get help

For example:
Let's say you don't know what `Config` does
In order to get the manual you can type
`git help config`

- with this you get an offline manual for detailed help
  `git config -h`
- this gives you a quick help on the terminal

What are the Environments of Git?

In Git you have 3 environments for files.

1. Working Files (Editing Files stage)
2. Staging (Holding Stage)
3. Commit (An entry to the History books)

What is a Repository?

A centralized storage location for digital assets, such as software code, data, or files, that provides a structured way to store, manage, and share them

What to do

1. Open **Git Bash**(kind of like a terminal)
2. specify you name and email.

Specifying Name

`git config --global user.name "Yaseen Aliyev"`

Specifying Email

`git config --global user.email "yaseen.aliyev@gmai.com"`

3. Set a branch (the video doesn't give info about what branch is)

Setting branch

`git config --global init.default branch main`
main is the name of the branch, you can type anything else

Changing Directory to the Site Folder

4. set the directory to the **folder of your site**.
   `cd` command changes the directory.
   `cd C:/Users/Yasin/Desktop/gitsamplesite` (Make sure the folder name doesn't contain spaces)

Now we must turn this into a git **repository**.
We just type: `git init`

When you go to the folder, and choose to see hidden items you can see the .git folder

5. You can check the status by typing
   `git status`

In here you can see the **Untracked files**. This means that if we make any **changes** to these files git **would not care**. However, if we **track** these files the changes will be known by git.

In order to **track** a file we must type:
`git add (file name)`
After that we an check the status again using `git status`
We can also type `git add .` to track **all** of the files

If for whatever reason you want to **untrack** a file, you can type the command (written above tracked files) which is:
`git rm --cached (file name)`

Ignoring Files.

In order to make **git** ignore files when checking `git status`, we have to make a text file and call it **".gitignore"**. By doing so, the **.txt** file becomes a **Git Ignore Source File**.
After that we need to open the ".gitignore" file and edit it in notepad.
Type the following:
`*.txt`
And now when we check the status, we can see that the **.txt** files are ignored.

Committing

Committing is like creating a **snapshot** version of a file. If you ever want to go back to this point in the future, you can do that.

To commit we can type the following:
`git commit -m "(your message here)"`
This command will commit all of the files, and the message is like writing the current state of the files in the history book.

Now, when we check the status there will be nothing to commit, working tree clean.

Modifying a File after Committing

Let's say you go to one of files and change the content of the file. After you save what you changed, you can check the status by typing `git status`. A new message will appear saying, along with other lines.
modified: (file name)
In order to check what changes were made to the file we have to type the following command:
`git diff`
The file will be previewed in **Red**(old version) and **Green**(new version) to show what changed.

We can add this change to the history books by using `git add (file name)`
However, this doesn't update the file instantly. It puts it in a environment called **Staging**.
Staging is the holding pin that waits until you are ready to **Commit**.

We can take the file back to the "Working Files" state by typing:
`git restore --staged index.html`

You can also bypass the **Staging** part and **Commit** instantly.
For this we can type:
`git commit -a -m "updated text to free range"`
-m allows you to type a message

Deleting Files

You can either Delete it from the folder in file explorer
OR
you can type:

- `git rm ("file name")`
  If you check the status after deleting the deleted file will be shown in the deleted section.

To restore the file use the command `git restore ("file name")`

Renaming Files

To rename a file we have to type:

- `git mv "(File Name)" "(New Name)"`
  You can also add this to the history book by typing:
- `git commit -m "changed the file name of an image"`

Renaming Branches

We have to go to the branch that we want to change the name of.
To rename a branch we have to type:

- `git branch -m "New name"`

The Log

You can check the log with the `git log` command.

- The log goes from the most recent to the oldest **commit**
- After executing the command, we can see the author and the email of the author(this is known because we set it at the begging of the session)
- You can also an abbreviated version of the log by using the command:
  - `git log --oneline`

Let's say you made a mistake or typo in the commit. Luckily we can change the commit without re-doing the process again. We have to simply type the following:

- `git commit -m "(New commit message)" --amend`
  Afterwards, you can check the log again, and your commit message would be updated.

You can dig into the specifics of commits using the command:
`git log -p`

press Up/Down arrow key to navigate
You can press "q" to exit this view

- You can also jump back to different commits
- You can modify the history book so that commits appear in different order.
- You can set what can be viewed in the history book

This command lets you **edit** the history book and how commits appear.
`git rebase -i --root`
To exit this view press ": + x"

*Note: something weird happened when I exited Rebase. While the window showed the normal terminal it didn't go back to the main branch, it went to (main | rebase). In order to fix this you can type the following command

- `git rebase --abort`

New Branches

So far we have been working on a single branch (The main Branch), however you can have many branches, like the **copy** of the **main** branch which has the **same entries** in the history book. If there ever was a **bug or a problem** in the **main** branch you can cope the branch and fix the problem, once that is done and you are satisfied you can then merge the branch. This is not just for bugs, it can also be applied to making a new **feature** or any other type of change.

Creating the New Branch

To make a **New Branch** type:

- `git branch (Branch Name)`
  or
- `git switch -c (Branch Name)` (this command creates and switches to the newly created branch)

After making the new branch we can check which branches exist by typing:

- `git branch`
  The existing branches will be shown and to see which branch is the **active** one, you can check the branch which has a **"`*`" symbol** on the left of it.

To **switch** to a branch you can type:
`git switch (Branch Name)`

Merging Branches

After making changes in your new branch, you can now **merge** it with your **main** branch.

1. Go back to you main branch using `git switch (main branch)`
2. type: `git merge -m "Merge Fixtemp back to main" (New branch)`

Now your changes have been made!

Deleting Branches

You can now delete the branch by using the following command:
`git branch -d (new branch name)`
then type:
`git push origin --delete (old branch name)`
`git push --set-upstream origin (new branch name)`

What if main has changed since you created your branch?

In this case, you will have a **Merge Conflict**.
EXAMPLE

- We have a branch named **main** and we make and switch to a branch named **Update text**.
- There is a file called "ingredients" in both branches.
- We are in the **Update text** branch we can change the content inside without a worry.
- After making our changes, we can commit.
- Switch back to our **main** branch and change the content of the "ingredients" file to another thing then commit.
- Try to merge the two branches.
- The output will be **"Automatic merge Failed"**
- In order to fix it you have to go to **File explorer** (the folder).
- Open the file and both versions of the file including two texts will appear.
  1. <<<<<<<<<<< HEAD
  2. ==========
     - Delete the version which you don't want to keep and delete the "`<<<<<<HEAD`" and "`==========`" .
     - After this you can safely **commit** and it will be updated.

After uploading to GitHub, you may make changes over there. If you want to update your local files so it matches the ones in GitHub, type the following commands.
either:
`git fetch` then `git merge`
or
`git pull`