Sessions 16-20

#16 - Resetting The Head

- Git reset is a powerful command that allow us to undo local changes to the state of a Git repo.
- Git reset operates on "The Three Trees of Git". These trees are the Commit History (HEAD), the Staging Index, and the Working Directory.
- Suppose that we created two files good.txt and bad.txt, then we added those file to our staging area and we commited them in the local repo.
- To check these commits we use

```
git log
```

- commits work similar to the linked list, the head points at the last commit we did and the last commit points at its precedent
- To delete ‘The bad text’ commit all we have to is to use the command ‘reset’ on the commit’s hash which we want the head to point at, in our case it’s ‘The Good Text’

```
#'97731228c165fa7e55556de9043c5087543fbdec' is the hash of 'The Good Text'
git reset --hard 97731228c165fa7e55556de9043c5087543fbdec
```

- Now to update our remote repo

```
#--force to force git to make an update without making the remote repo telling us that
we are behind by one commit or more
git push origin main --force
```

- Suppose I have 4 commits, 1, 2, 3, and 4. If I want to delete the first 3 commits all I have to is to point the head at the hash of the 4th commit

```
git reset --hard HASH_OF_NUMBER_4
```

#17 - Ignoring Files And Directories

- Sometimes I may need to ignore files which are not necessary to be push to remote repo.
- For example if I am working on a node.js project I may not need to push the node_modules files which contain all dependencies of the project, since the folder's size is large.
- First create a file called '.gitignore' which will contain the files and the file extension that git need to ignore

```
touch .gitignore
```

- Then let's say we want to ignore all files that end with '.log' extension
- To do that you can open the '.gitignore' file using notepad or any code editor and inside write

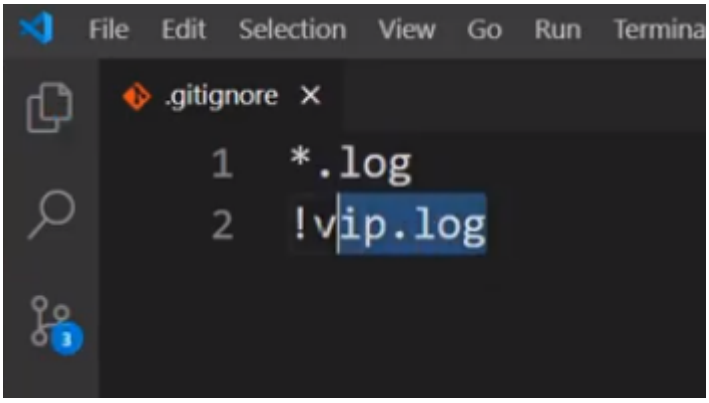
  >*.log

- Now if we create log files they will be ignored by git

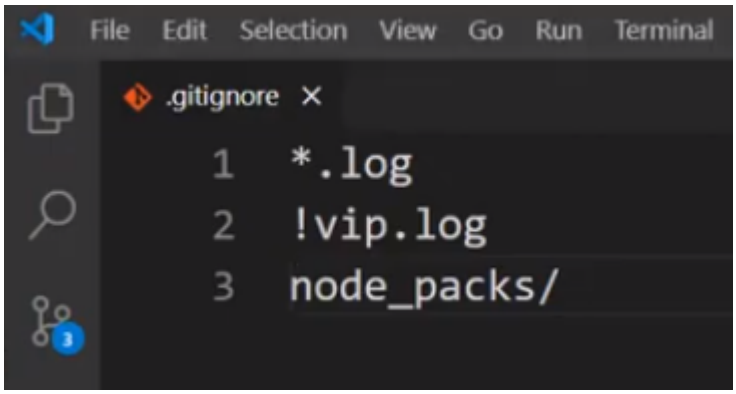
```
$ touch ahmed.log

Ahmad@DESKTOP-JNHOR7N MINGW64 ~/Desktop/Web Dev/Git and github/Elzero (main)
$ git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
                    notice how git ignored 'ahmed.log' file
nothing added to commit but untracked files present (use "git add" to track)

Ahmad@DESKTOP-JNHOR7N MINGW64 ~/Desktop/Web Dev/Git and github/Elzero (main)
```

- What if I want to ignore all '.log' files except one '.log' file called 'vip'
- In the log file add

  >!vip.log



- To ignore a folder or a directory, just mention that folder inside the .gitignore file. For example, our folder is called node_packs/

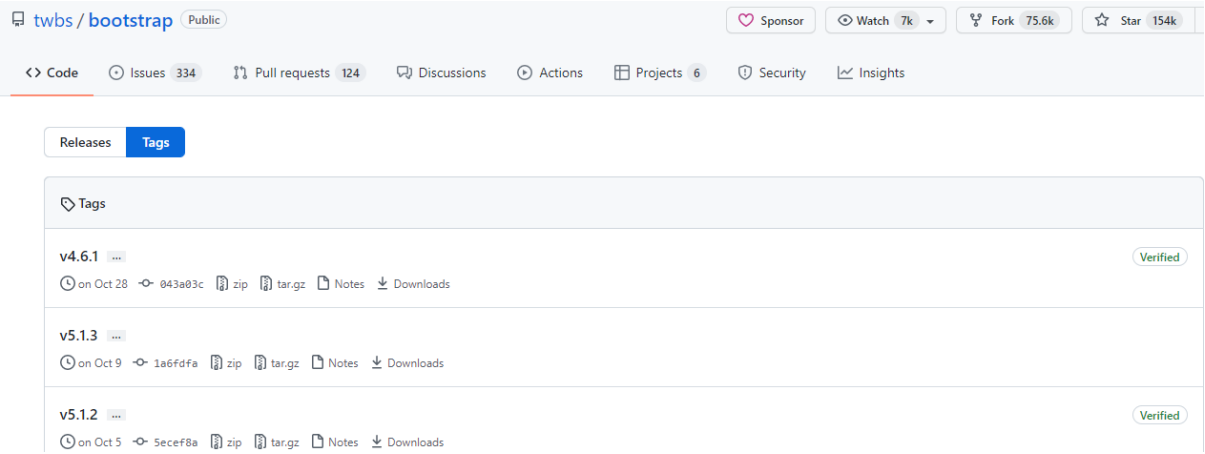


- To have more info about '.gitignore' write in google this keyword 'git ignore patterns'

- Lastly don't forget to add the '.gitignore' file in the staging area

#18 - Tagging And Releasing Part 1

- Tagging is generally used to capture a point in history that is used for a marked version release (i.e. v1.0.1).
- For example if you go to the Bootstrap repo, you will how they have multiple tags since they have multiple releases and versions



- let's say we create a file, committed, and push it to our remote main branch
- We can create two type of tags, lightweight tags and annotated tag
- Lightweight tags are tags that take their info like their descript and meta data from the last commit we make. However when define annotated tags, we provide them with the needed info like description and so on.
- To show list of tags

```
git tag
```

- Git suggest to name our tags in this form 'vN.0'. In this example we will create a lightweight tag

```
#create our first tag or version of our project
git tag v1.0
```

- Push this tag to our remote repo

```
git push origin v1.0
```

- Let's do an example, where I want to create a new file, push this file to my remote repo, but this time I'm create a new annotated *tag* or version.

```
touch new_feature.txt
git add new_feature.txt
git commit -m "Added new feature"
git push origin main
git tag -a v2.0 -m "Our Project => Second Version"
git push origin v2.0
```

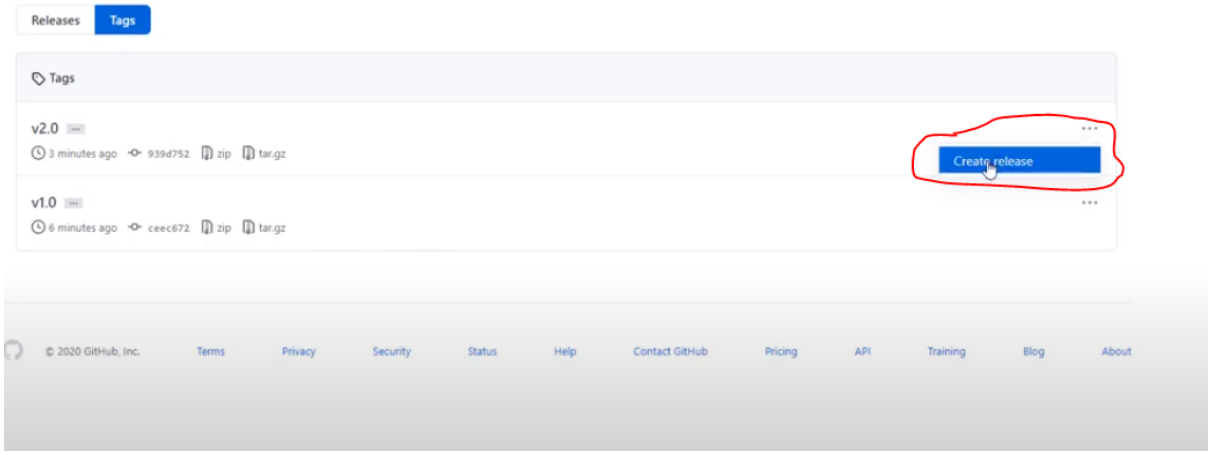
- Note that since we are working in the same branch, any work or commits that will be push to the remote will be added to the 'v1.0' and 'v2.0' tags.
- If you want to separate the work from each tag, those tags should be in different branches.

#19 - Tagging And Releasing Part 2

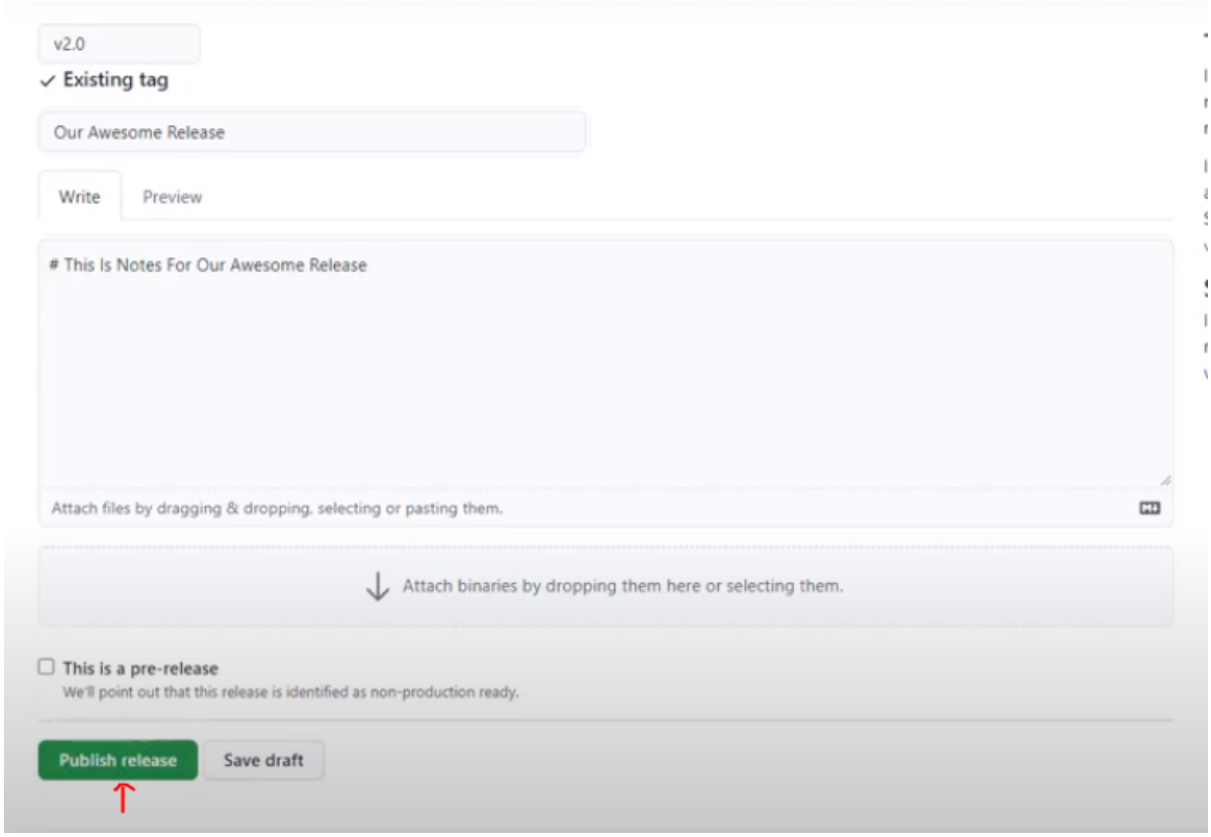
- If I want to view a list for my all version 1 releases

```
git tag -l "v1.*"
```

- **Releases** are deployable software that can be package and available for other users to use. They are based on **Git tags** and they may contain release notes about commits and links to binary files, for other people to use.
- To create a release



- In the image below in the description we write markdowns to explain what does this release contain



- If we delete tags in Github we have to delete them in our command line.

```
git tag -d v1.0
```

- If I want to delete the tag in the remote repo using the command line

```
git push origin --delete v1.0
```

#20 - The End And Advices

- Try always to use git and Github in any project you create.
- Learning Git and Github is a very essential skill you need as a developer, because big companies use these technologies in managing their projects