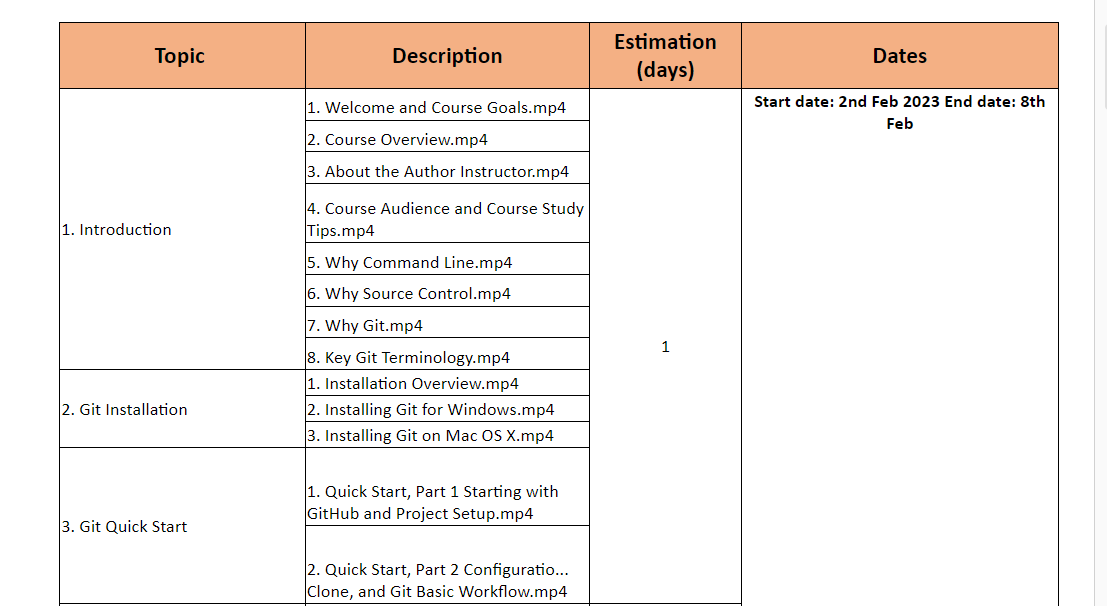
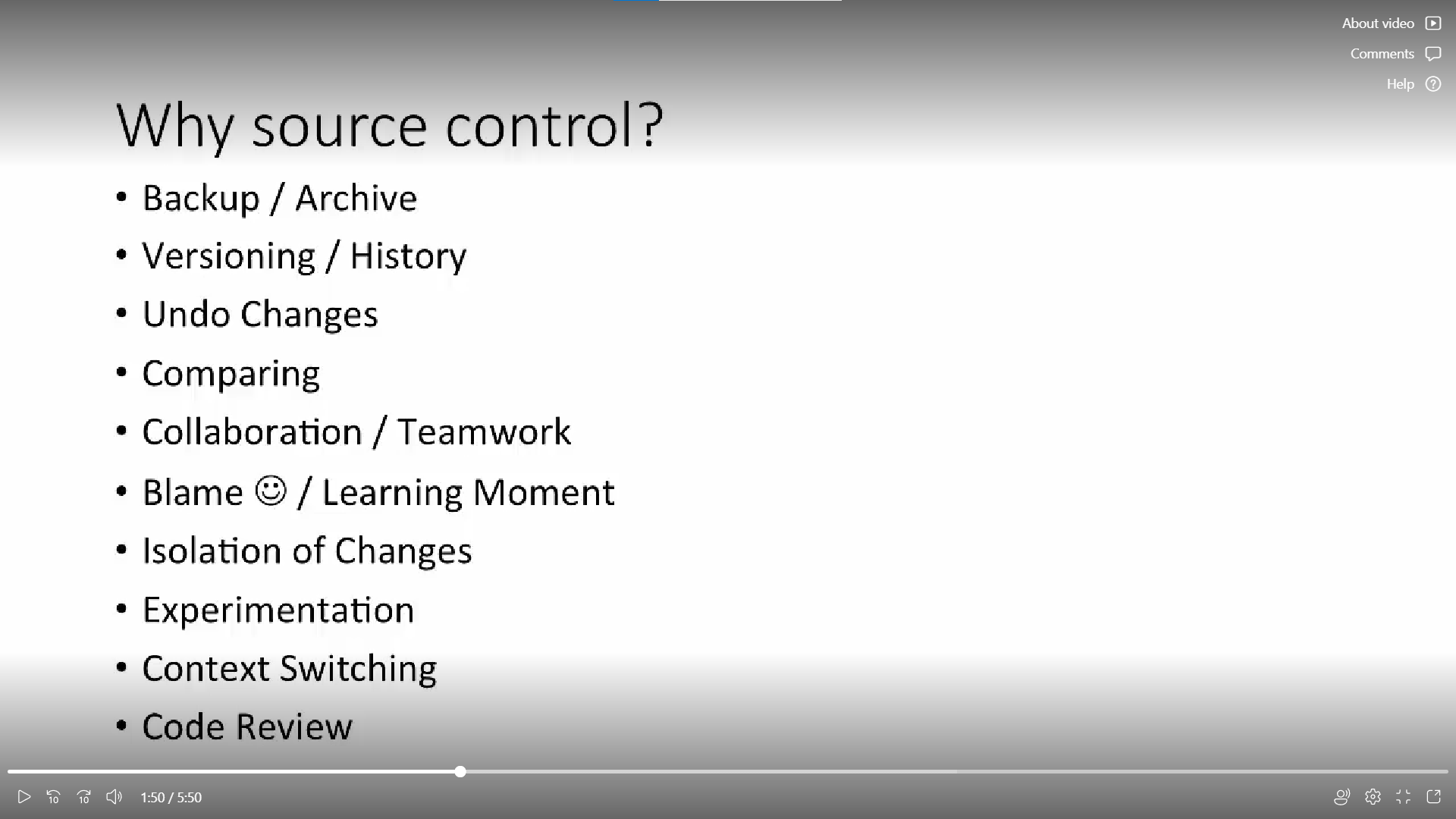
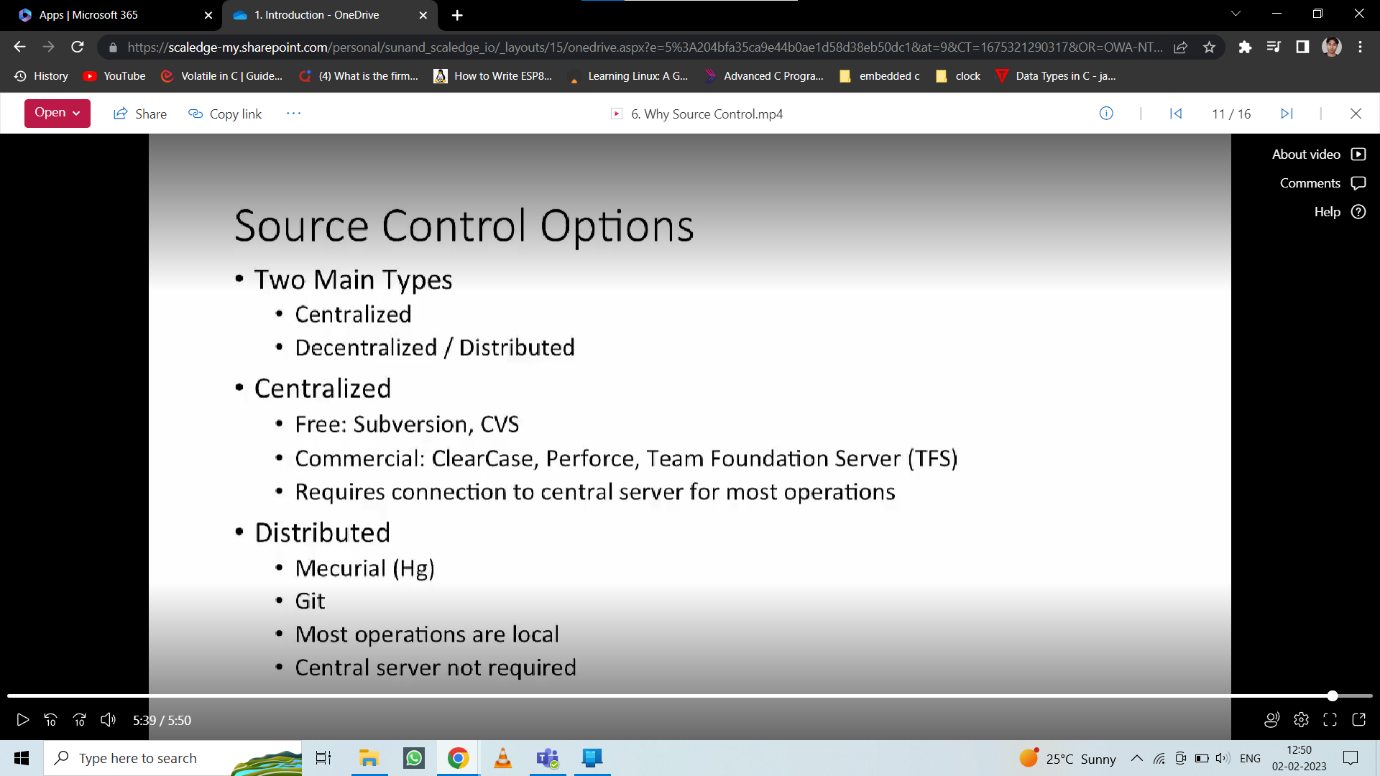
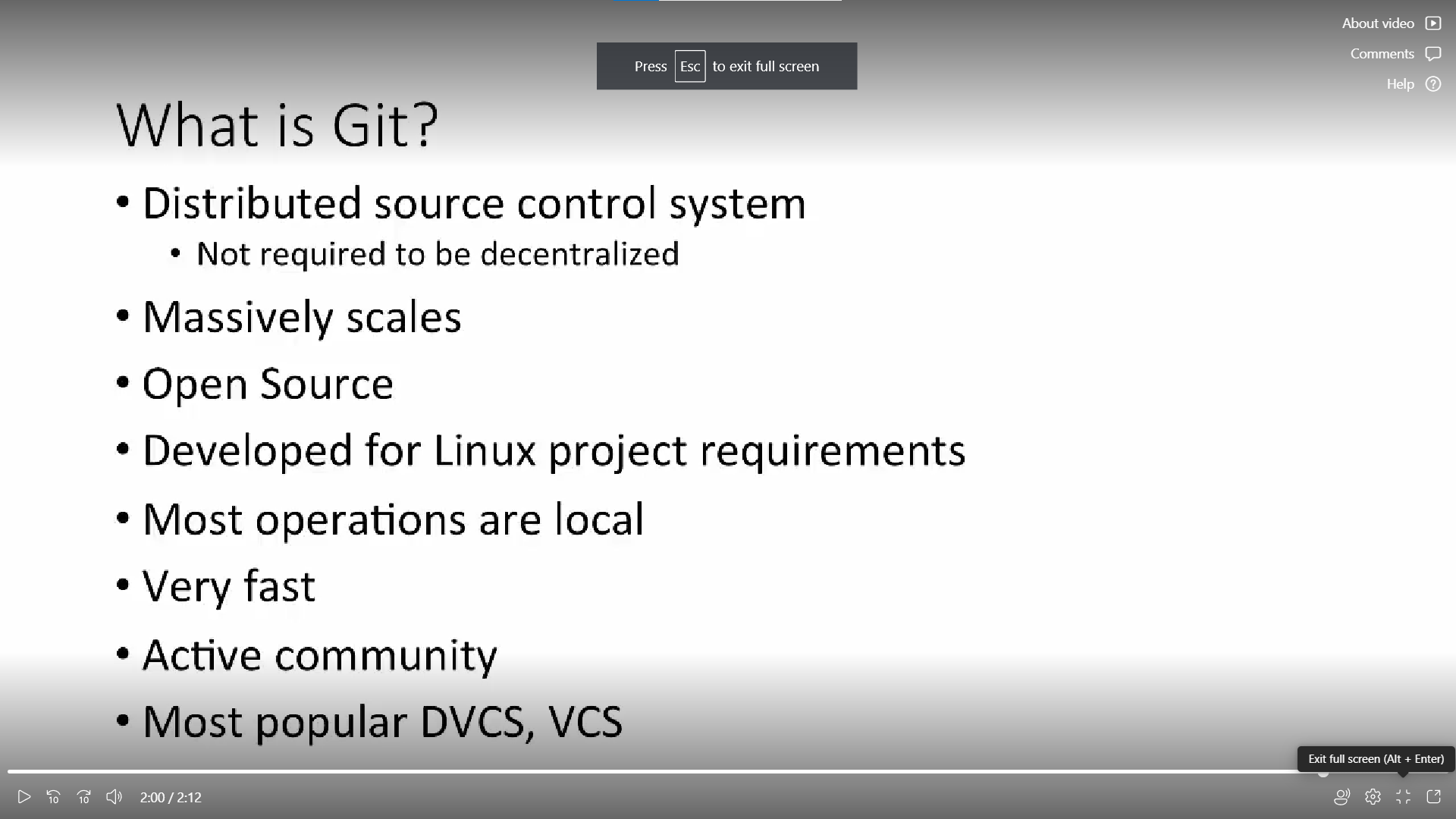
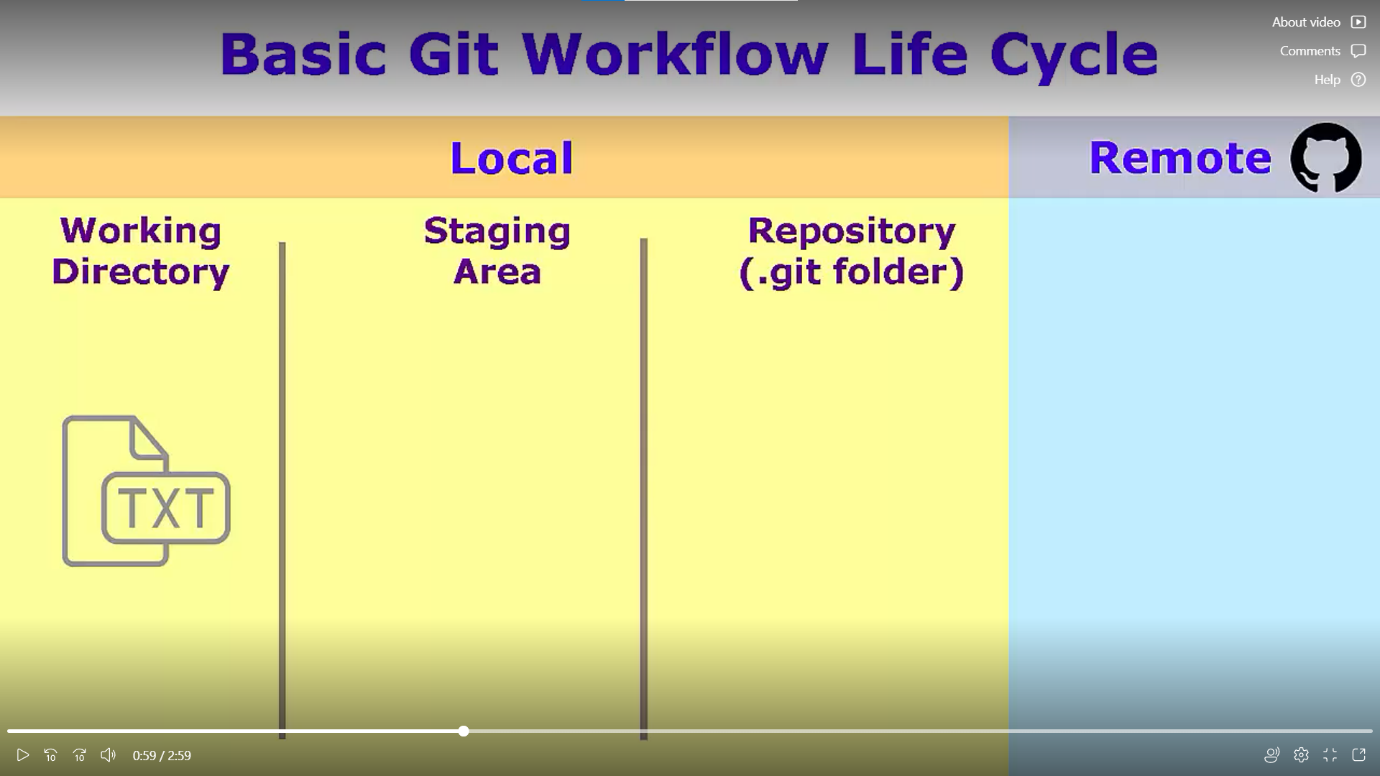
2 Feb 2023 Git











Some command:

**pwd** : print working directory

**mkdir** : make new directory

**cd**: change directory

Git commands:

**Git version**: to check version of git installed in computer

Clone URL from GitHub repository

**Git clone**: to get the copy of GitHub repository

**Echo**: to print string

**Git status**

**Git add**: to add file in staging area

**Git commit**: to move file from staging area to local repository

**Git push origin master**: to push file from local repository to remote name origin and branch master

**Exit**: to close program

**Some terminologies:**

**1. Git Repository**

* A Git repository is a directory that contains a Git working directory.
* The working directory contains all the files that are tracked by Git.
* The Git repository is the place where the files are stored.
* We can think of a Git repository as a file system. Where the files are stored.
* Git Repository has a directory structure.
* The directory structure is similar to the directory structure of a file system.

**2. Git Working Directory**

* A Git working directory is a directory that contains all the files that are tracked by Git.
* Tracked by Git means that Git tracks the files in the working directory that will be committed.

**3. Branch**

* As its name suggests, the branch can be thought of as an alternate repository that diverges from the Git Working Directory or main project.
* We can think of a Branch as an extra repository that is used to work on adding specific features to the project.
* A Branch can be a different version of the same repository.

**4. Push**

* Push is a command that is used to push the changes from the local repository to the remote repository.

**5. Status**

* Status is a command that is used to show the status of the local repository.
* A status can be staging area, unstaged changes, staged changes, untracked files.

**6. Master/Main**

* We usually call the main branch as master.
* It is the default branch. It is the main branch of the repository.
* It is the branch that is used to develop the project. It is the primary branch in the repository.

**7. Origin**

* Origin is basically an alternative local alias that is set in place of the remote repository URL.

It refers to the remote URL of the repository.

3Feb 2023

**Previously given assignment: .git**

The **.git** folder contains all information that is necessary for the project and all information relating commits, remote repository address, etc.

It also contains a log that stores the commit history. This log can help you to roll back to the desired version of the code.

**.git file:**

[core]

repositoryformatversion = 0

filemode = false

bare = false

logallrefupdates = true

symlinks = false

ignorecase = true

[remote "origin"]

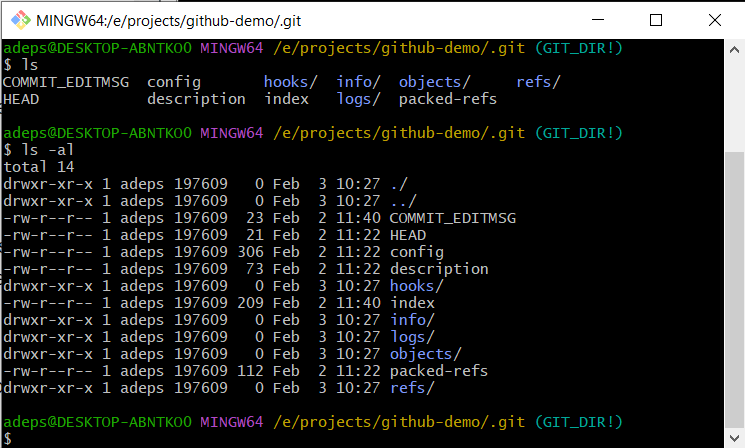
url = https://github.com/sumitadep002/github-demo.git

fetch = +refs/heads/\*:refs/remotes/origin/\*

[branch "main"]

remote = origin

merge = refs/heads/main



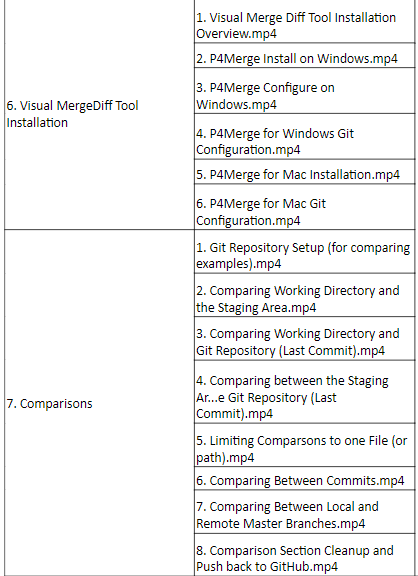
**WHY it is hidden?**

The **.git** folder will contain details of every single change made to the code base. All snapshots of the modifications will be recorded in this folder like a database, which makes it possible to undo the changes and rollback to the desired version of the code.

**The .git folder is hidden to prevent accidental deletion or modification of the folder. The version history of the code base will be lost if this folder is deleted. This means, we will not be able to rollback changes made to the code in future.**

* **hooks** − This folder contains script files. Git hooks are the scripts that are executed before or after events like commit, push etc.
* **objects** − This folder represents an object database of Git.
* **config** − This is the local configuration file.
* **refs** − This folder stores information about tags and branches.
* **HEAD** − This file stores reference to the current branch. It points to the master branch by default.
* **index** − This is a binary file and stores staging information

**Further topics of today:**

****

**git init <filename>**:will initialize new directory locally

**$ls –al:** show all files including hidden files

**git add <filename>**: used to add file into staging area but it is not committed yet

**git status** : will show newly created files, modifications or untracked files

**git commit :** will invoke default text editor which notepad++ in your case, which will allow multi line commit message

**rm –rf <directory or folder name> :**  to remove directory

**mv <file1> <file2> :** will rename the file1 as file2 or it will move file1’s content to file2 and delete file1

**git add .:** will add the files to staging area

**git commit –m “Custom message” :** will commit with custom message

**To bring up existing files from GitHub**

Visit the URL of that project and click on fork and after that visit your profile and copy URL from HTTP tab

After that run command: **git clone <URL>**

Above command will make a copy of that repository in your current working directory

**git pull origin master:** will update copy of local repository by remote repository

**git push origin master:** this command will make change in remote repository,

here origin is remote repository and master is default branch

**Recursive Add**

**Note:**  directories containing another directories into it than git status command will only show **parent directory only, Except you add all of them by** git add . Command

**Back out**

**git checkout -- filename:** will bring back the state of the file when it was unstagged

**Renaming**

Using git command like **git mv file1 file 2**

But if we are renaming by bash command like **mv file1 file2**

**Deletion of files**

If file is untracked then even after deleting file **won’t make any on status**

If file is tracked then we **have to commit** to make further changes

**Backout**

**git reset HEAD filename:** will unstage the file

**git checkout – filename:** will discard changes

06 Feb 2023

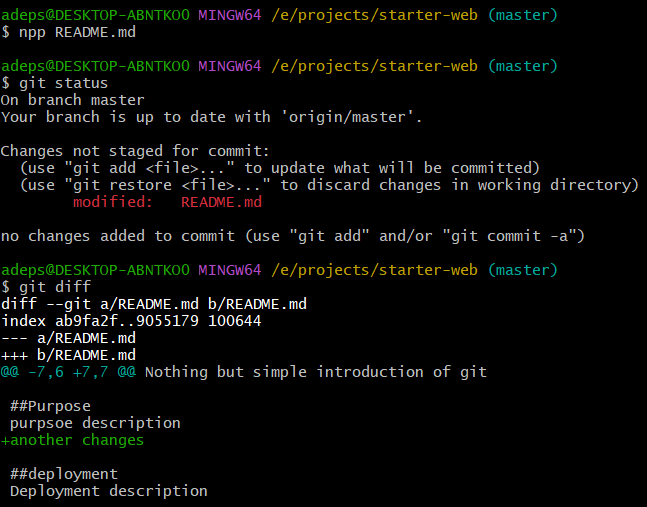
**Installation and configuration of P4MERGE:**

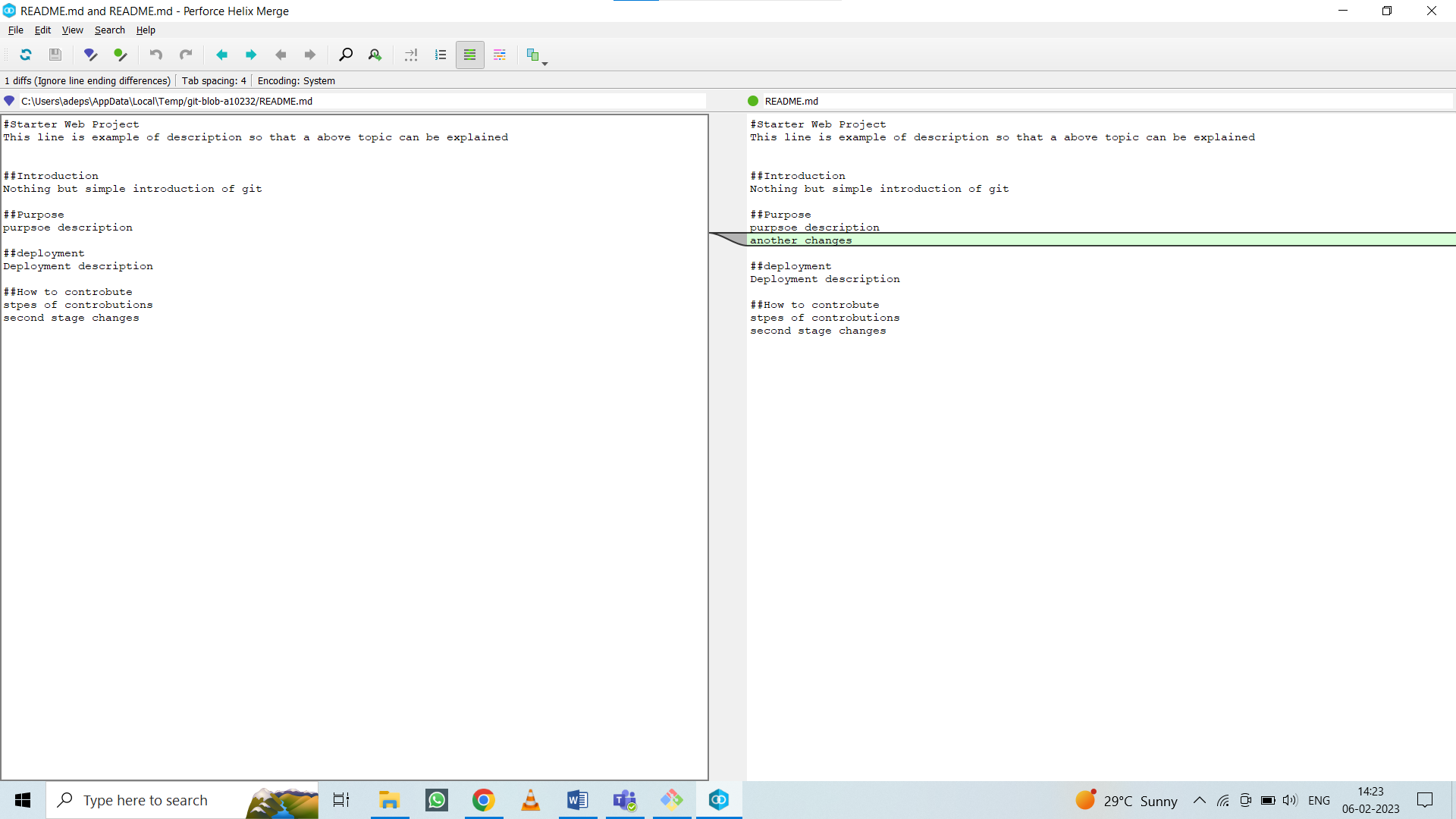
* Highlight and edit text file differences.
* Choose to include or ignore line endings/white spaces.
* Recognize line-ending conventions.
* Display line numbers when comparing and merging file

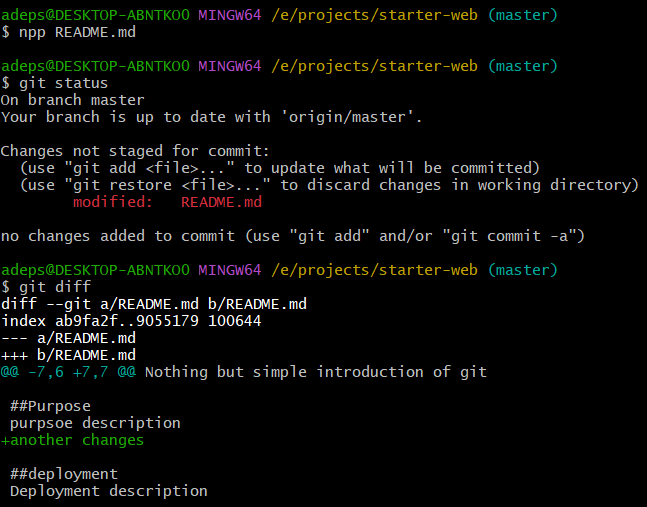
**README.md:**

* File that contains information for the user about the software, project, code, or game, or it might contain instructions, help, or details about the patches or updates.
* When sharing your code with the world, a problem that might occur is that they may not particularly understand how to use it or even understand it.
* So that is where the readme file helps. The readme file is used to explain what is uploaded and how we can install or use it. It even allows the uploader to add images

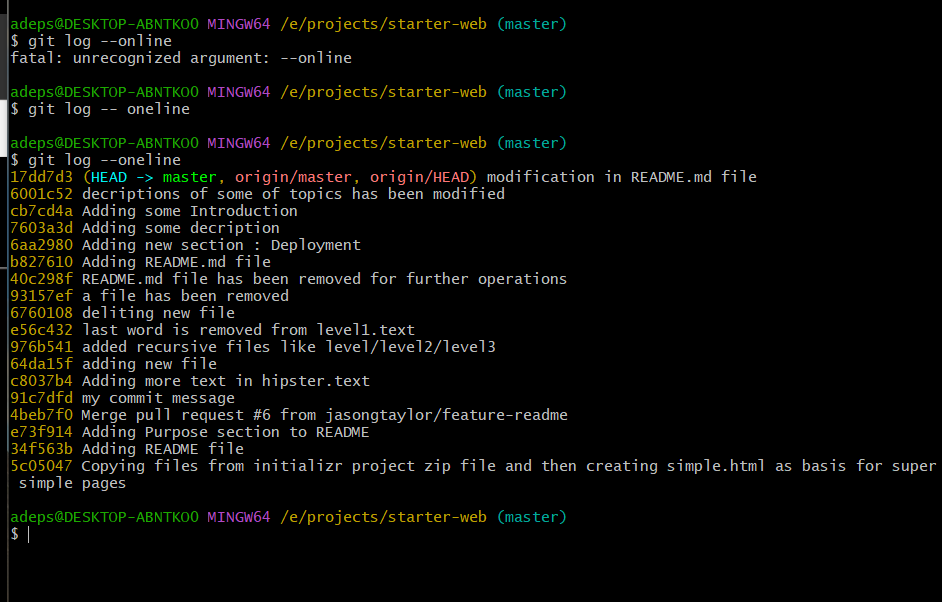
**git diff :** git status can show what **kind** **of changes** has done in file, but git diff shows you **exact changes** made into then file.



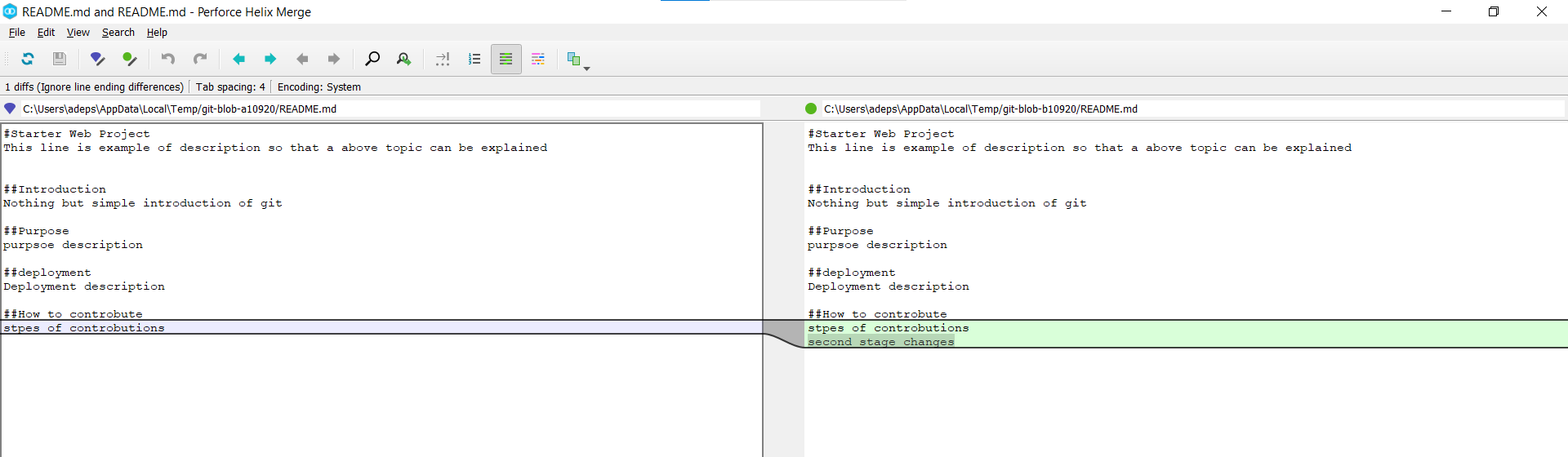
**git difftool :**  will visually show the difference between staging area and working directory side by side**.**[](06%20Feb%202023/CHANGE%20in%20readme%20file.png)

**git diff -- README.md:** will showcasechanges made in specified file

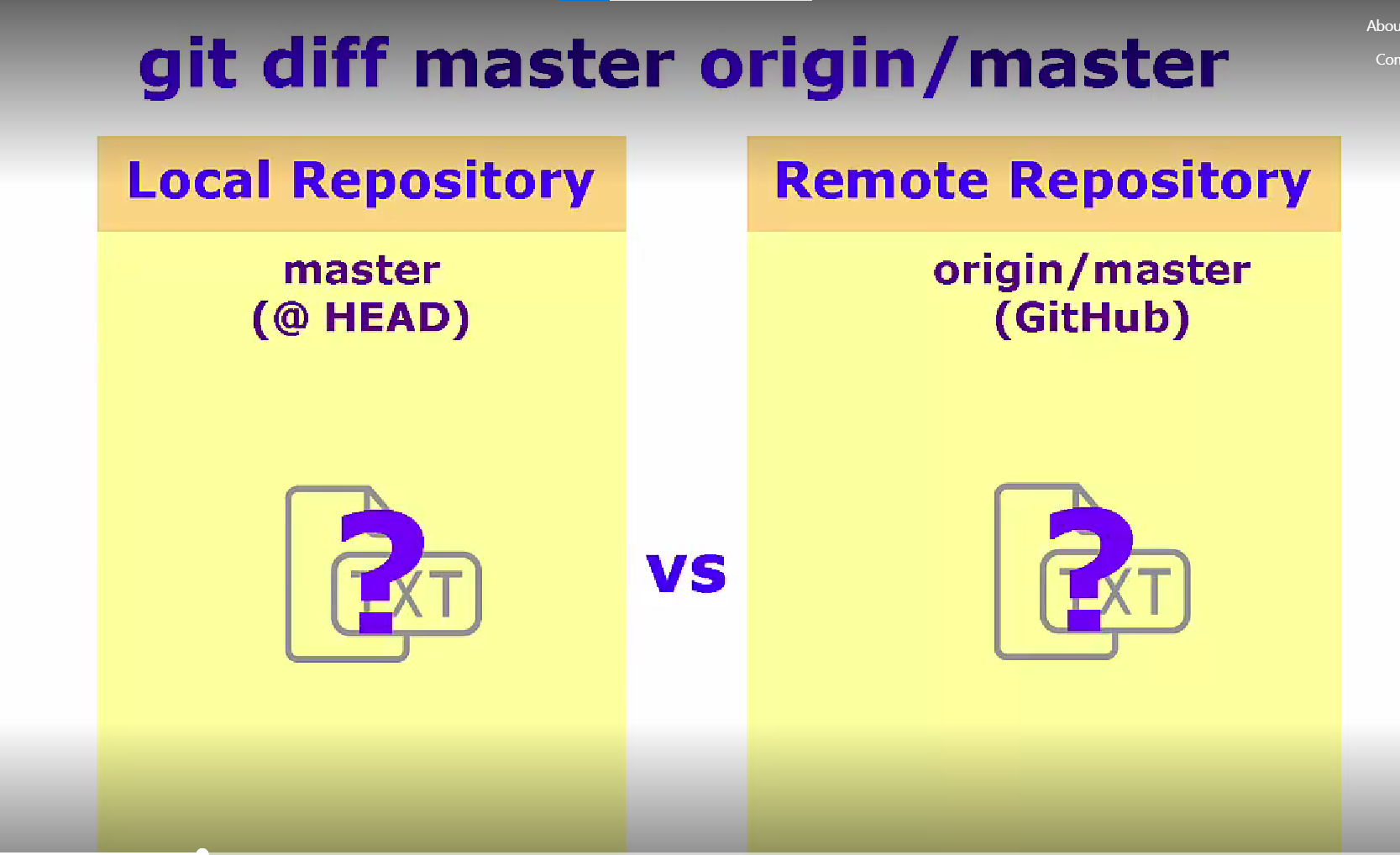
**Comparison between two commits:**

Every commit is associated with an ID

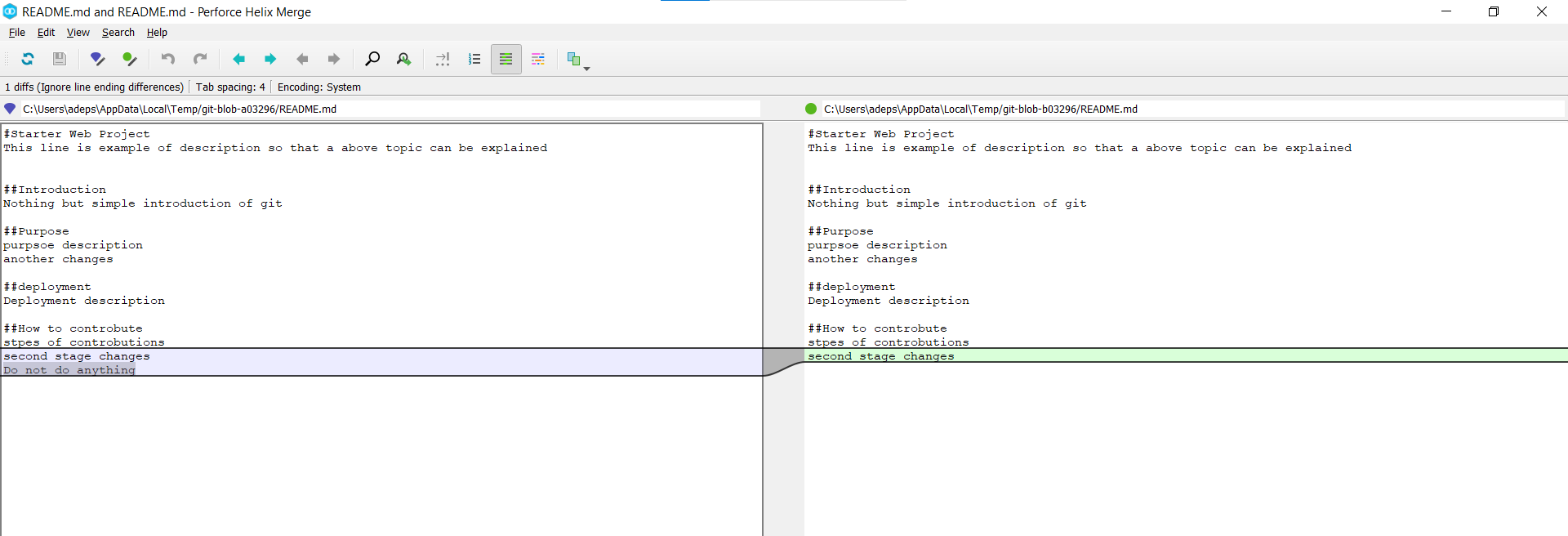
Let’s compare two commits:

**git difftool 6001c52 17dd7d3:** will give below shown response****

**Comparison between local and remote repository:**

****

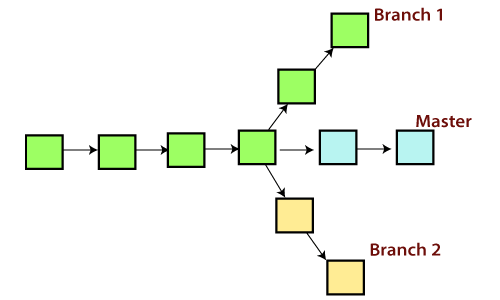
**Command:** git difftool master origin/master

[](06%20Feb%202023/local%20vs%20remote%20repository%20comparison.png)

07 Feb 2023

**Branch**

* A branch represents an independent line of development.
* A branch is a version of the repository that diverges from the main working project.
* In short it new branch isolates main or master branch.
* So if something goes wrong with new branch it won’t affect our main default branch.

****

**The Operations that can be performed on a branch**

**Create Branch**

You can create a new branch with the help of the **git branch** command. This command will be used as:

**Syntax:**$ git branch  **<branch** name**>**

### List Branch

You can List all of the available branches in your repository by using the following command.

Either we can use **git branch - list** or **git branch** command to list the available branches in the repository.

**Syntax:** $ git branch or $ git branch --list

### Delete Branch

You can delete the specified branch. It is a safe operation. In this command, Git prevents you from deleting the branch if it has unmerged changes. Below is the command to do this.

**Syntax:** $ git branch –d **<branch** name**>**

**Switch Branch**

Git allows you to switch between the branches without making a commit. You can switch between two branches with the **git checkout** command. To switch between the branches, below command is used:

$ git checkout **<branch** name**>**

**Rename Branch**

We can rename the branch with the help of the **git branch** command. To rename a branch, use the below command:

**Syntax:**

$ git branch -m **<old** branch name**> <new** branch name**>**

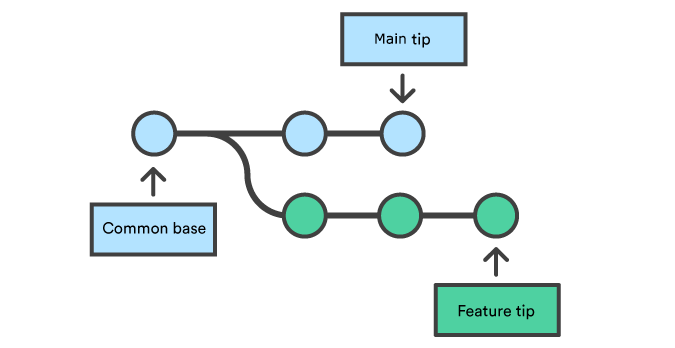
# **Git Merge**

Merging is Git's way of putting a **commits back together**.

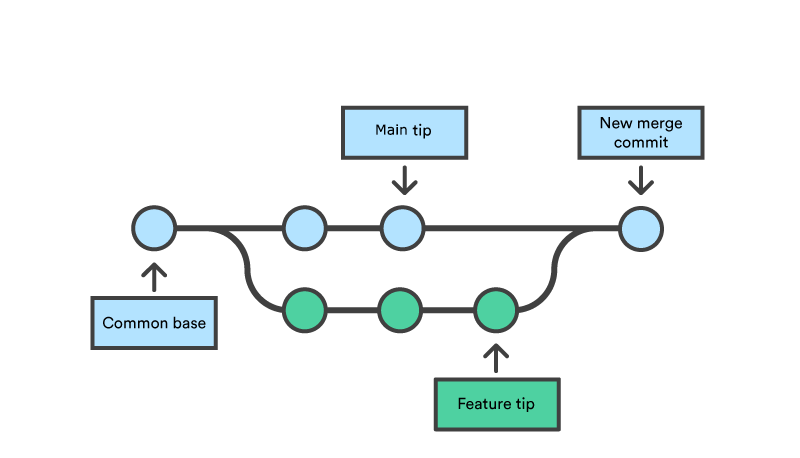
The git merge command lets you take the independent lines of development created by git branch and **integrate** them into a single branch.

## **How it works**

Git merge **will combine multiple sequences of commits into one unified history**. In the most frequent use cases, git merge is used to **combine two branches**.



This command will merge the specified branch feature into the current branch.



Merge commits are unique against other commits in the fact that they have two parent commits. When creating a merge commit Git will attempt to auto magically merge the separate histories for you.

If Git **encounters a piece of data that is changed** in both histories it will be unable to automatically combine them.

This scenario is a version control conflict and Git will need user intervention to continue.

## **Preparing to merge:**

Before performing a merge there are a couple of preparation steps to take to ensure the merge goes smoothly.

## **Confirm the receiving branch**

Execute git status to ensure that HEAD is pointing to the correct merge-receiving branch. If needed, execute git checkout to switch to the receiving branch. In our case we will execute git checkout main.

## **Fetch latest remote commits**

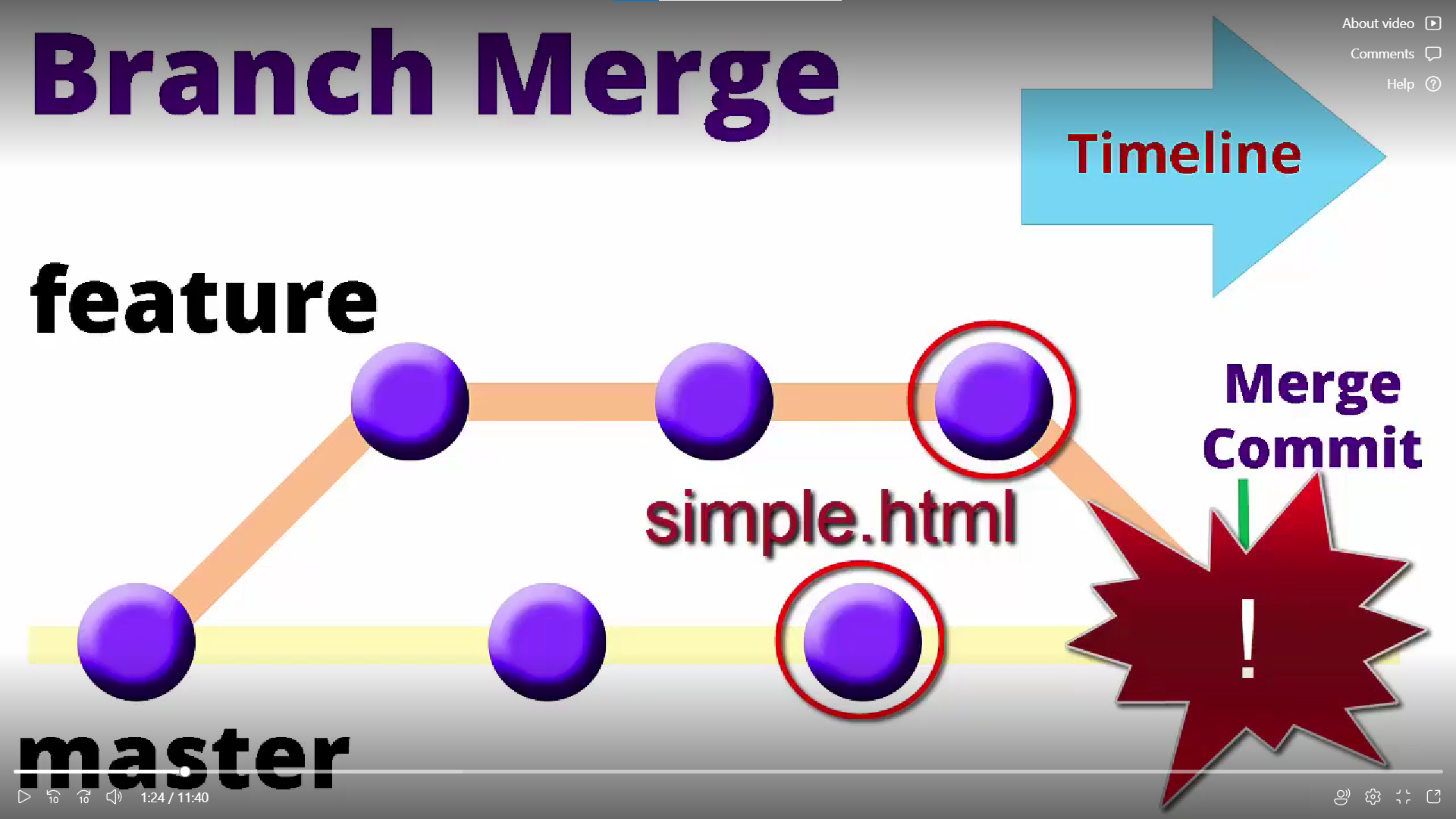
Make sure the receiving branch and the merging branch are up-to-date with the latest remote changes. Execute git fetch to pull the latest remote commits. Once the fetch is completed ensure the main branch has the latest updates by executing git pull.

## **Merging**

Once the previously discussed "preparing to merge" steps have been taken a merge can be initiated by executing git merge where is the name of the branch that will be merged into the receiving branch.

## **Git Merge Conflict**

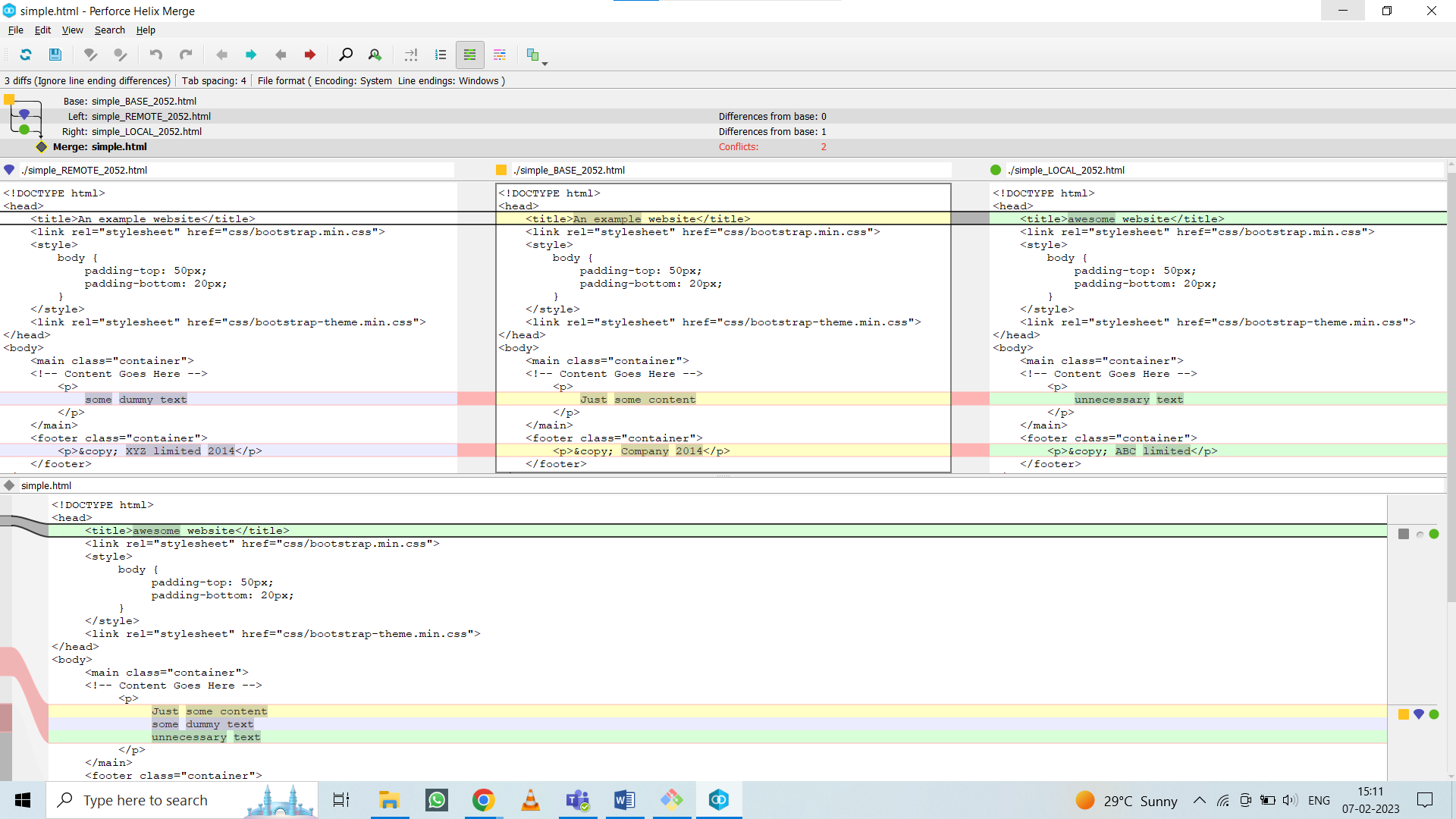
When two branches are trying to merge, and both are edited at the same time and in the same file, Git won't be able to identify which version is to take for changes. Such a situation is called merge conflict. If such a situation occurs, it **stops just before the merge commit so that you can resolve the conflicts manually**.



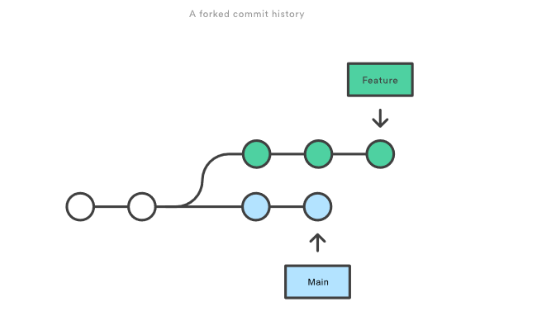
## **Resolve Conflict:**

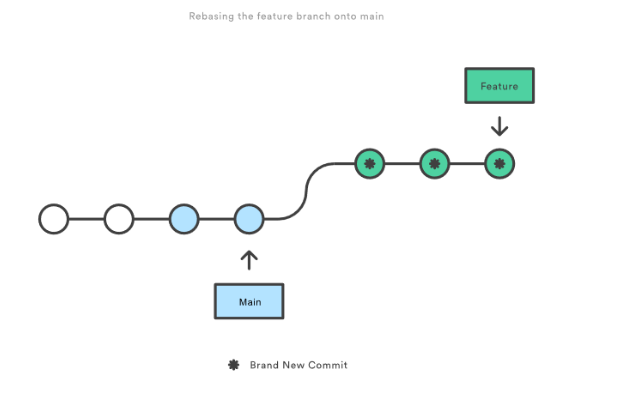
To resolve the conflict, it is necessary to know whether the conflict occurs and why it occurs. Git merge tool command is used to resolve the conflict. The merge command is used as follows:

$ git mergetool



## **Rebase:**

* Git utilities designed **to integrate changes** from one branch onto another.
* Git rebase is the **linear process of merging**.
* Here’s how Git rebasing compares to Git merging. Let's say you're a developer who is working on a new feature on a dedicated branch. Then, another development team member updates the main branch with some new commits. The situation looks like this:
* 
* If you use Git rebase, you move your whole feature branch, starting it on the tip of the main branch so that all the new commits are now part of the whole. This action **rewrites the project history by making new commits for each of the original branch's commits**. So, this is how the new branch looks:



**Git stash:**

**Command: git stash**

Git stash temporarily stored changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.

Stashing is handy if you need to quickly switch context and work on something else, but you're mid-way through a code change and aren't quite ready to commit.

**Command: git stash pop**

Will apply previous commits and delete them as well

**Command: git stash list**

Will show the list of stashes

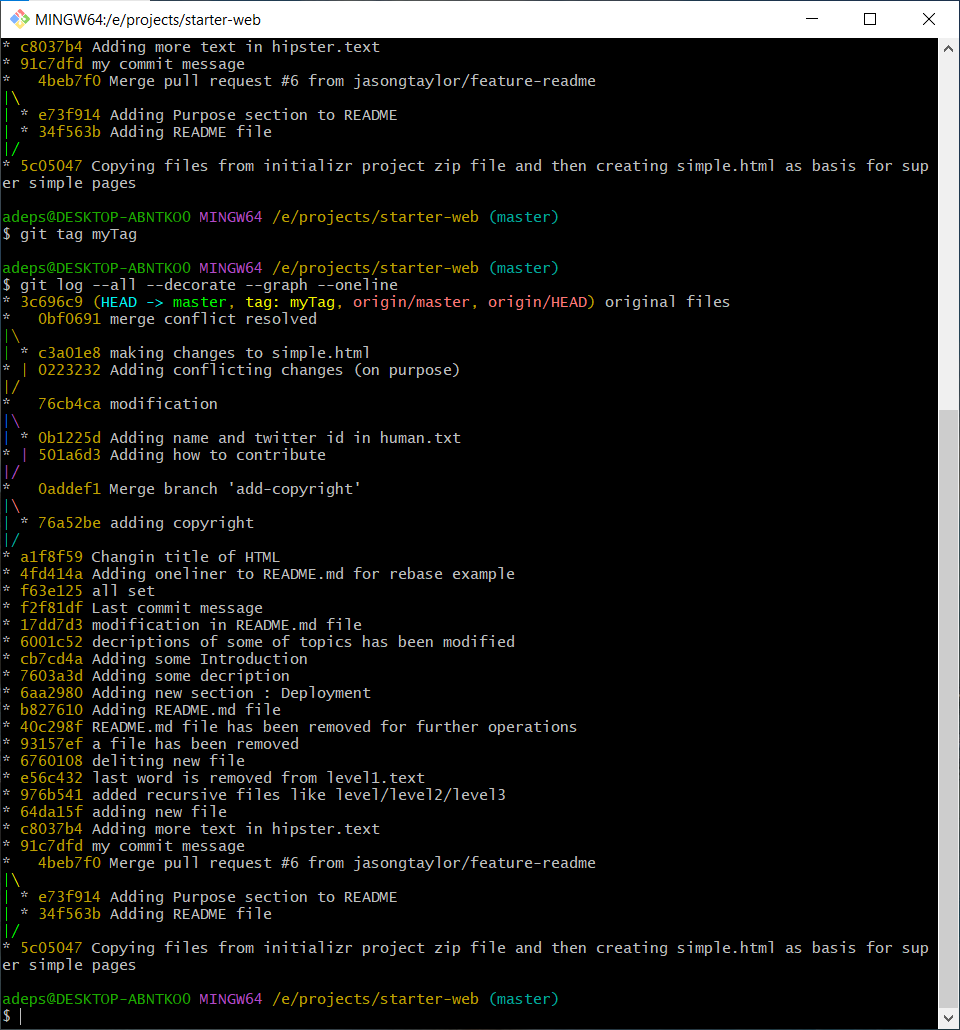
08 Feb 2023

**Tagging:**

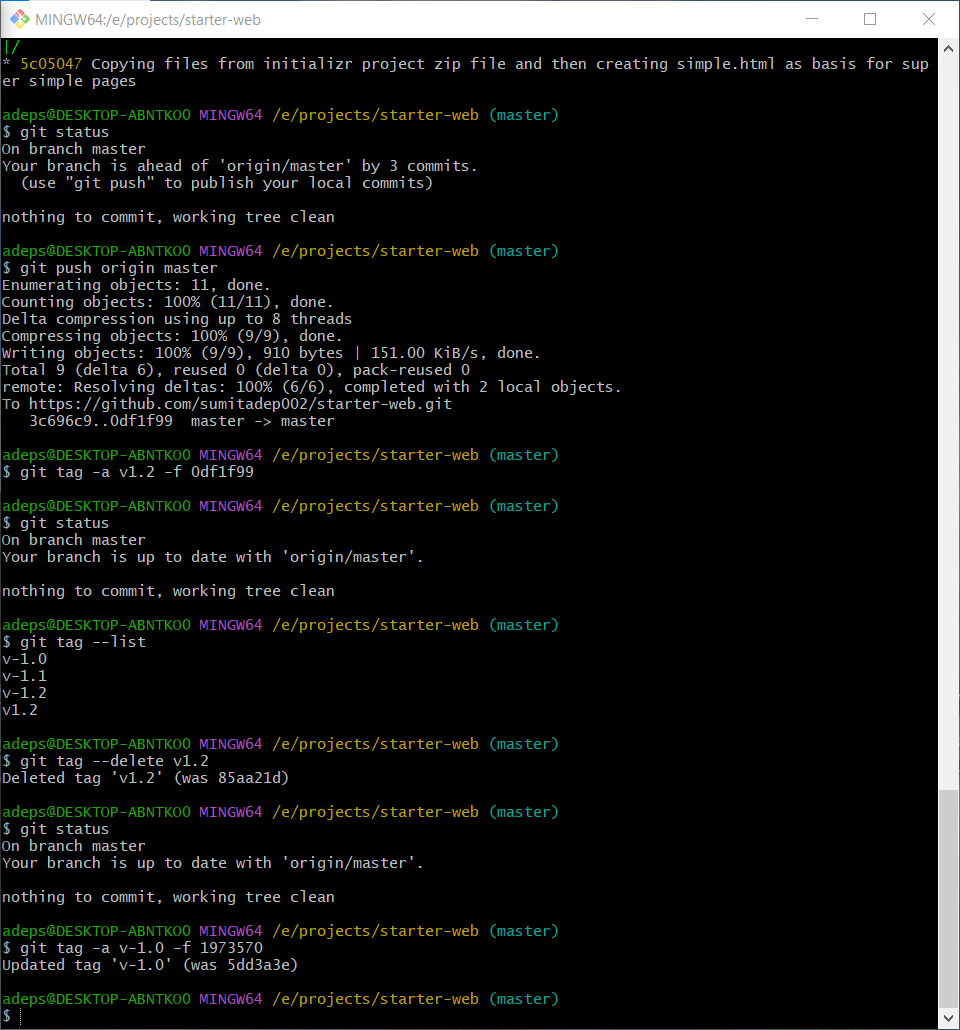
Process of applying label or tag to a commit.

**Commands:**

**To put a tag on latest commit:** git tag <name of tag>

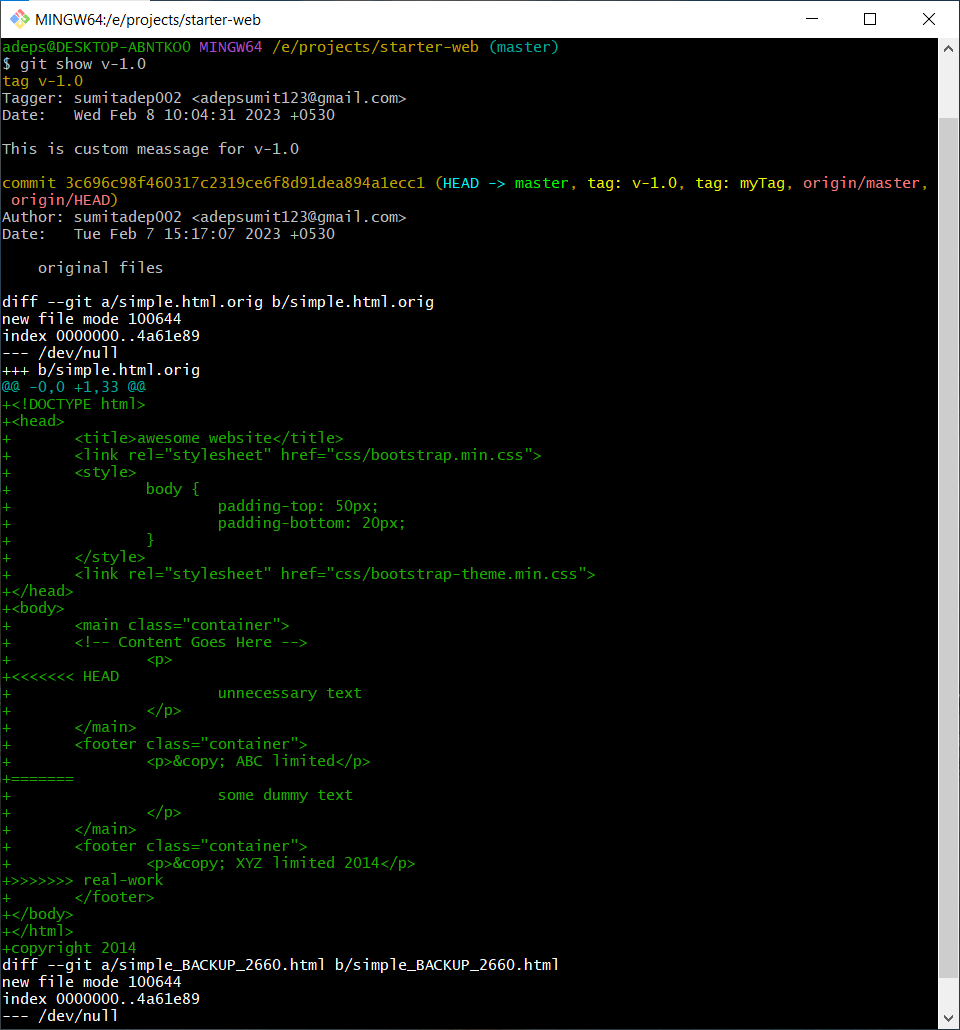


**To see list of tags:** git tag --list



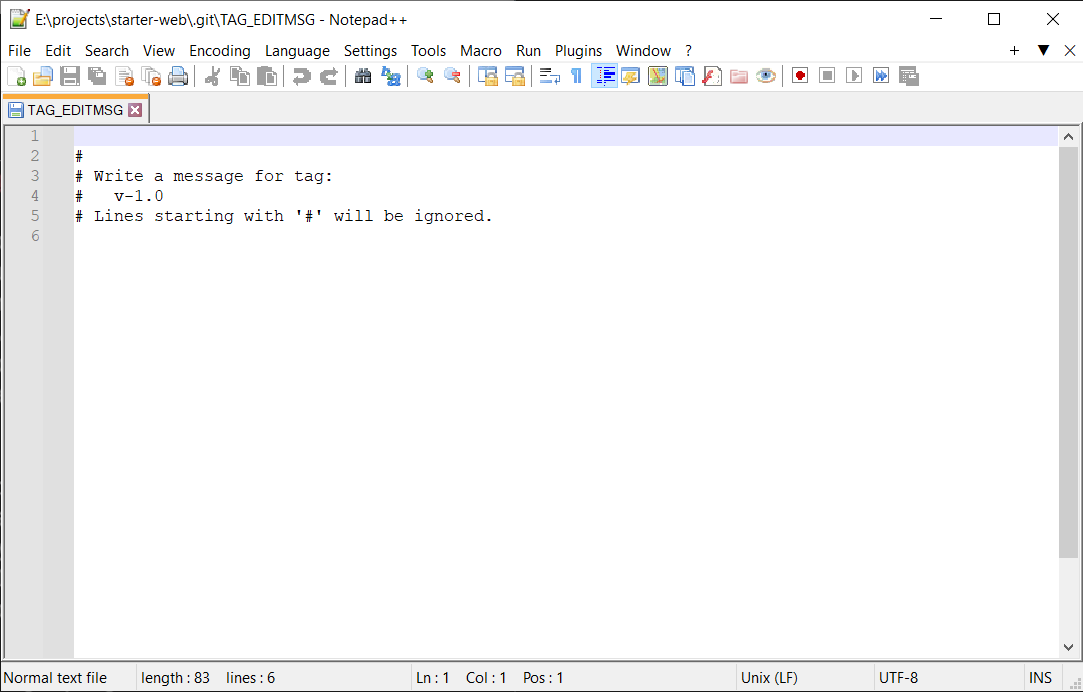
**To delete tag:** git tag –delete <name of tag>

**To see commit by its tag name:** git show <tag name>

****

**Annotated tag:** contains message or annotation with it

**Command:** git tag -a <tag name>

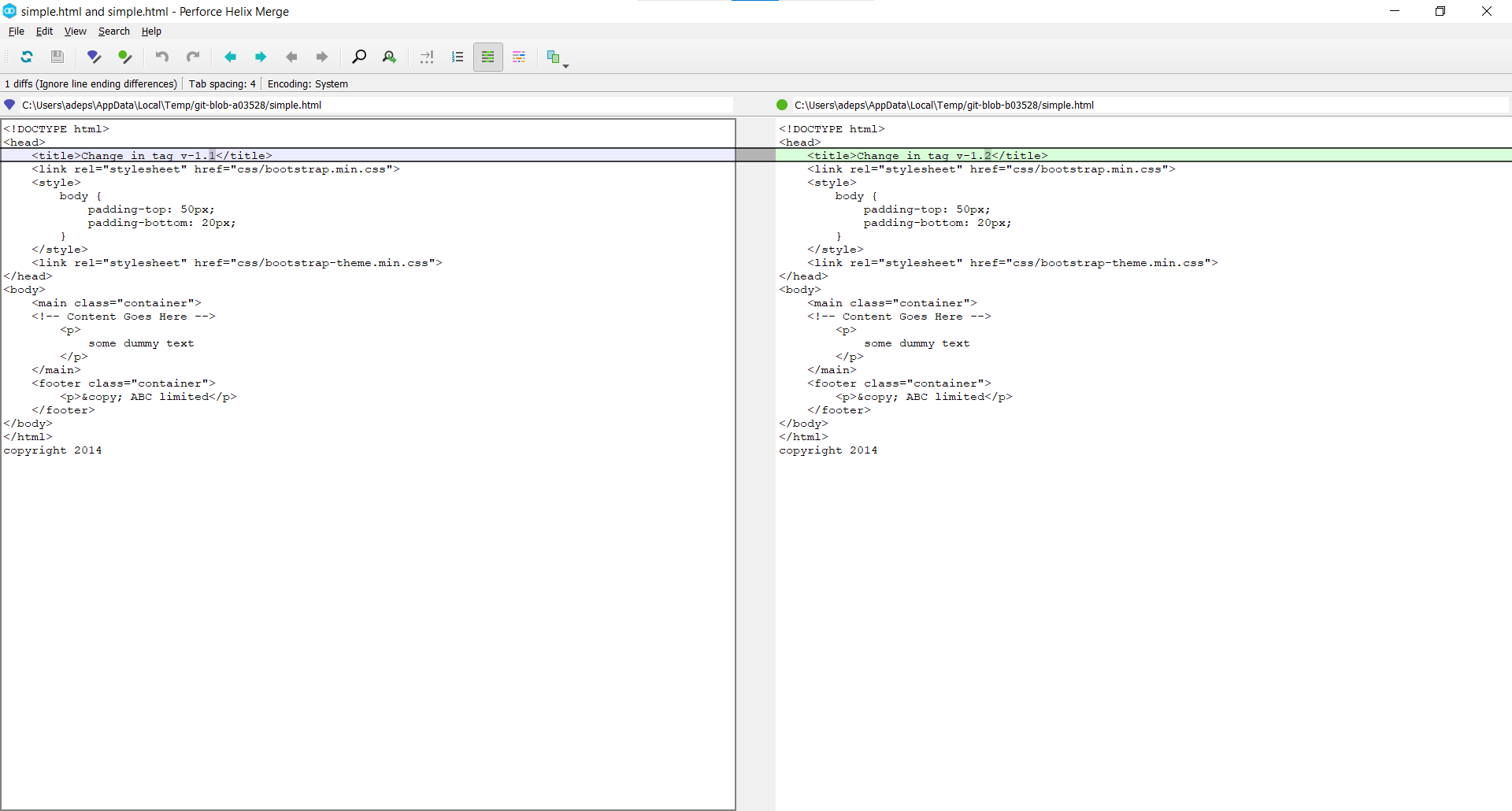
This command will invoke text editor and we can add our message text or any annotation. 

🡨You can add your text message here

**Comparison of tags:**

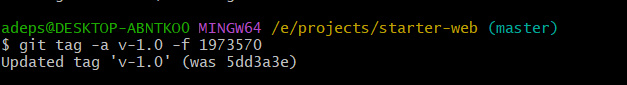
**Command:** git difftool <name of a tag> <name of another tag>

****

****

**Update Commit:**

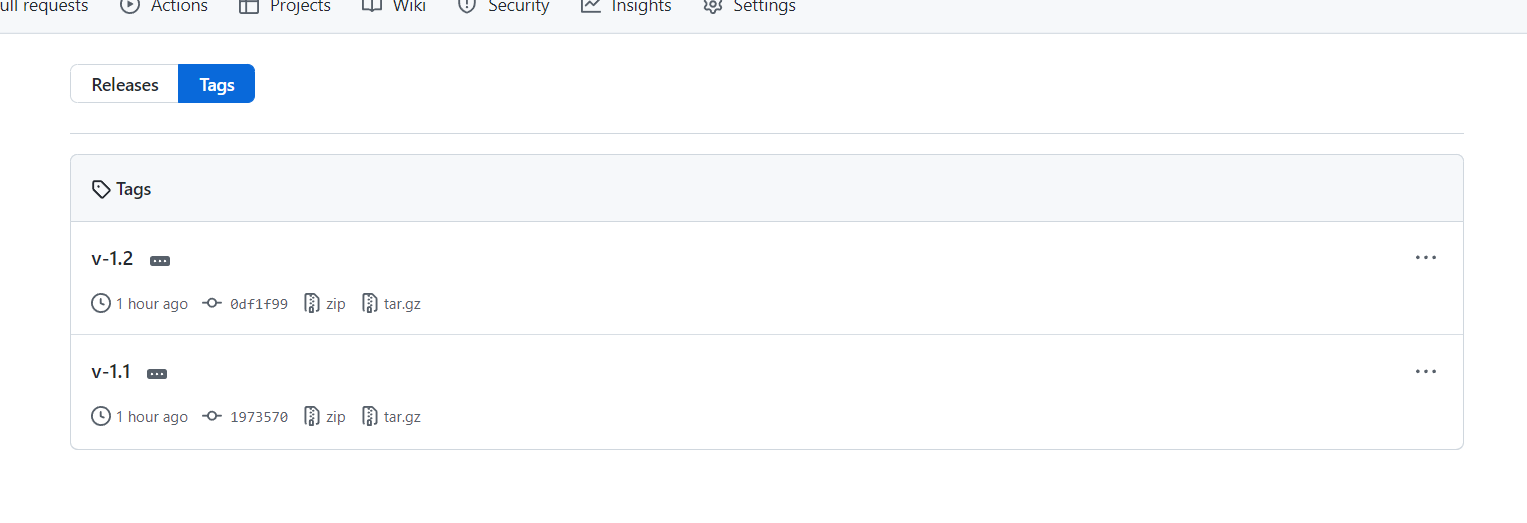
**Command:** git tag -a <name of tag> -f <commit id>



**Push tag to remote repo:**

**Command: git push origin <name of tag>**

You can check out your remote repository on your GitHub profile



You can toggle 3 dots to see commit message

