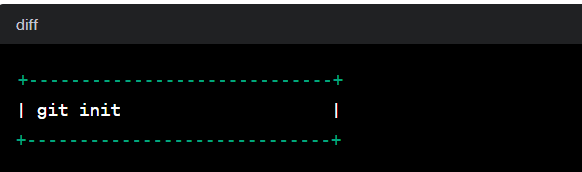
**GIT ALL COMMANDS**

**1. Git init:**

* Purpose: Initializes a new Git repository.
* Example: **git init**

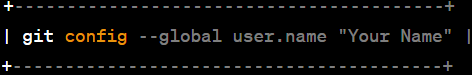


* Usage: **git init [repository name]**
* We have to navigate to our project directory and type the command **git init** to initialize a Git repository for our local project folder. Git will create a hidden. git directory and use it for keeping its files organized in other subdirectories.

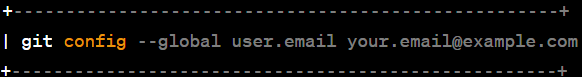


**2. Git config:**

* **Purpose: Gets and sets repository or global options.**
* **Example: git config --global user.name "Your Name"**

****

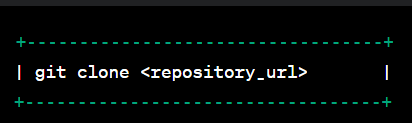
* **Explanation:**
* **git config --global user.name "Your Name"**
* **Purpose: Sets the user name for your Git commits.**
* **Example: git config --global user.name "John Doe"**
* **Explanation: This command is used to configure your global Git user name. The name you provide here will be associated with your commits across all Git repositories on your machine.**
* **git config --global user.email** [your.email@example.com](mailto:your.email@example.com)

****

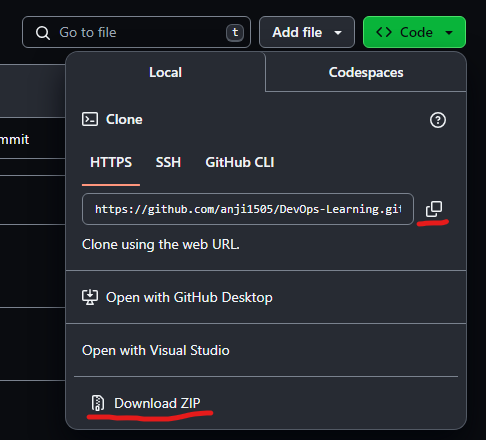
* **Purpose: Sets the email address for your Git commits.**
* **Example: git config --global user.email "john.doe@example.com"**
* **Explanation: This command configures your global Git email address. The email you provide here is associated with your commits and is often used for communication and identification.**

**3. Git clone:**

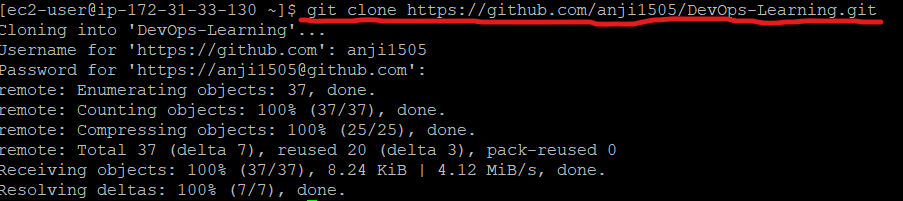
* Purpose: Clones a repository into a new directory.
* Example: **git clone <repository\_url>**



* Suppose, we want to work on a file that is on a remote GitHub repository as another developer. How can we do that? We can work on this file by clicking on **Clone or** **Download**and copying the link and pasting it on the terminal with the git clone command. This will import the files of a project from the remote repository to our local system.
* The below screenshot shows from where to copy the link:

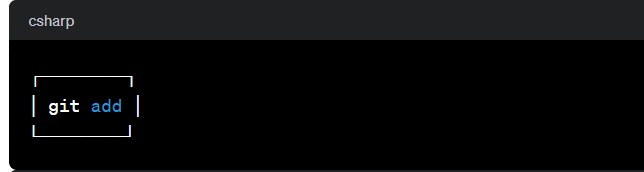
****

* To create a local folder, we have to use the following command:
* **mkdir [directory-name]**
* **cd [directory-name]**
* **git clone [url]**
* Now, paste the copied link along with the git clone command as shown below:

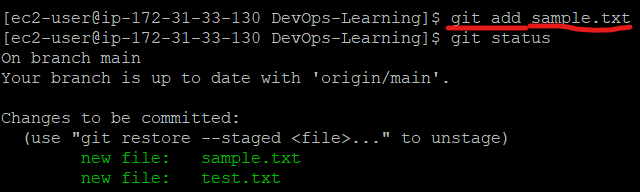


**4. Git add**:

* Purpose: Adds changes to the staging area before committing.
* Example: git add file.txt



* **Explanation**: This will add the specified file(s) into the Git repository, the staging area, where they are already being tracked by Git and now ready to be committed.

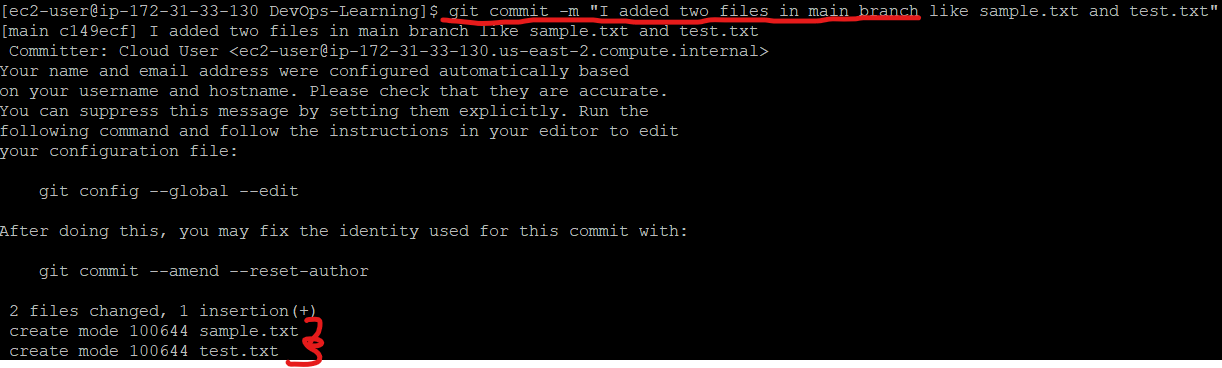


**5. Git commit:**

* Purpose: Records changes to the repository with a descriptive message.
* Example: git commit.

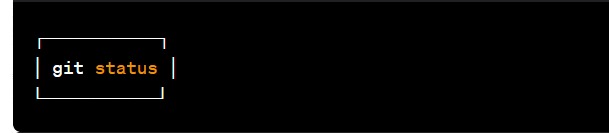


* **Usage: git commit -m “message”**
* Explanation:
* This command records or snapshots files permanently in the version history. All the files, which are there in the directory right now, are being saved in the Git file system.

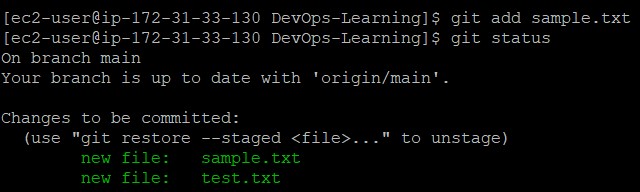


**6. Git Status:**

* Purpose: Shows the status of changes as untracked, modified, or staged.
* Example: git status

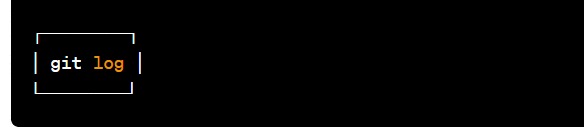


* Explanation: This command will show the modified status of an existing file and the file addition status of a new file, if any, that has to be committed.

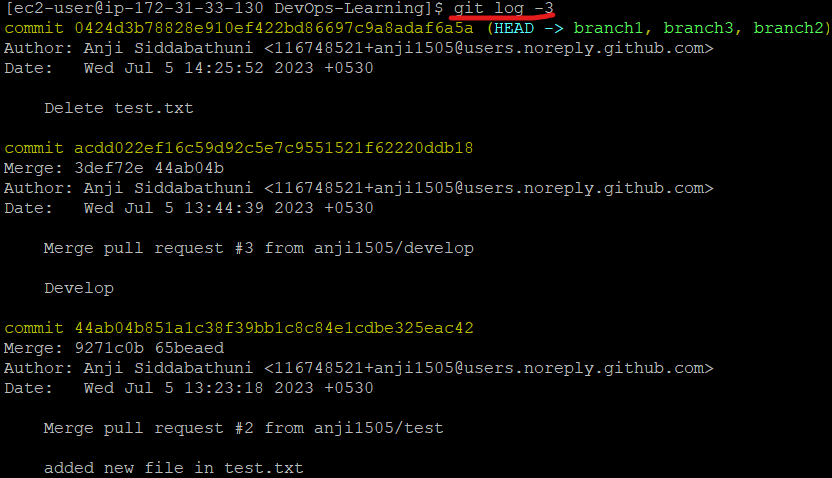


**7. Git log:**

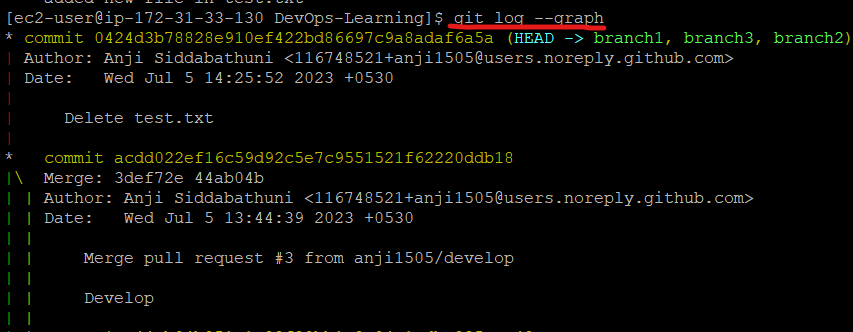
* Purpose: Displays a log of all previous commits.
* Example: git log



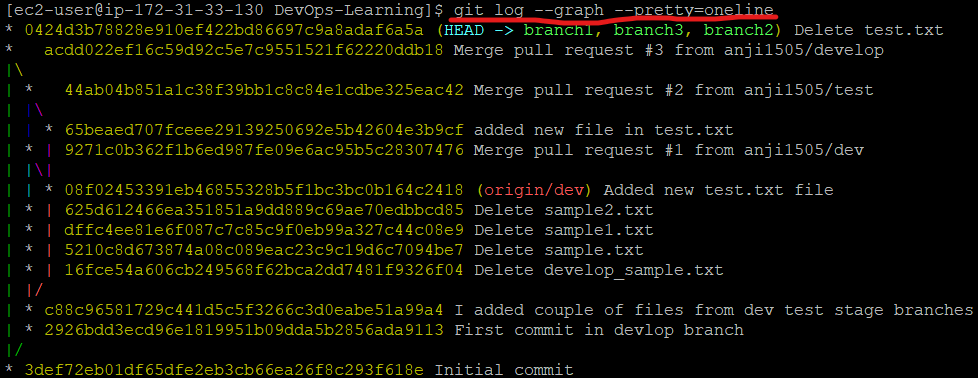
* Explanation:
* Usage-1: The “git log” command is handy when we want to examine the detailed log of every commit in our repository. By executing this command, we can view the log specific to the branch we are currently in. Additionally, we can use “git log -3” to display the last three logs.



* Usage-2: **git log –graph**
* For a visual representation of the commit history, we can utilize “git log –graph”. This option presents the commit-graph, showcasing the branching and merging of commits

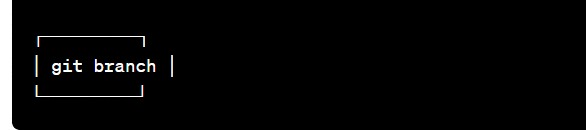


* Usage-3: **git log –graph –pretty=oneline**
* To further customize the output, we can use “git log –graph –pretty=oneline”. This format displays the commit graph along with a concise one-line description for each commit.

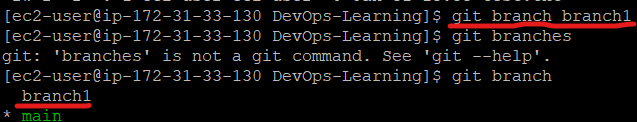


**8. Git branch:**

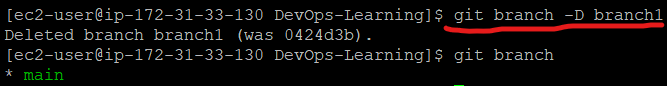
* Purpose: Lists, creates, or deletes branches.
* Example: git branch



* Explanation:
* **Usage-1: git branch [name-of-the-branch]**
* When multiple developers are collaborating on a project or repository, branches become essential for managing different workspaces. Using this command, we can create a new branch (for example, ‘branch1’). This allows developers to work independently on their respective branches, making changes and commits without affecting the main branch or other branches.

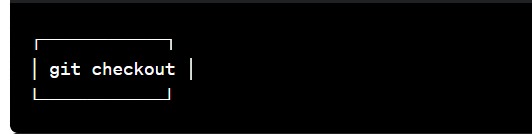


* **Usage-2: git branch -D [name-of-the-branch]**
* Likewise, to delete a branch, we utilize the “git branch -D” command. This enables us to remove a specific branch (e.g., ‘name-of-the-branch’) that is no longer needed, cleaning up the repository and reducing clutter.

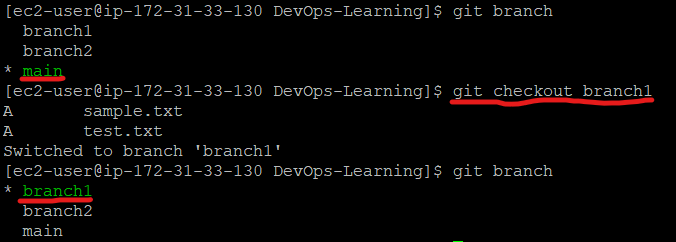


**9. Git checkout:**

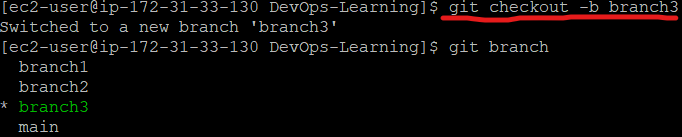
* Purpose: Switches between branches or restores working files.
* Example: git checkout



* Explanation:
* Usage-1**: git checkout [name-of-the-new-branch]: -**
* This command allows us to switch to an existing branch within our repository. It facilitates navigating to the desired branch, enabling us to add new files, make changes, and commit those files within that specific branch.



* **Usage-2: git checkout -b [name-of-the-new-branch]**
* This command serves a dual function. Firstly, it creates a new branch with the given name (for example, ‘branch3’). Secondly, it immediately switches our working environment to that newly created branch. This allows us to seamlessly begin working within the newly created branch, making it convenient to add files, make modifications, and commit changes exclusively within that branch.

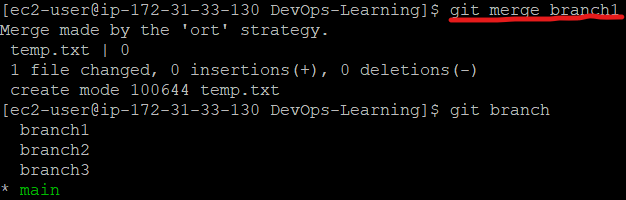


**10. Git merge:**

* Purpose: Combines changes from different branches.
* Example: git merge

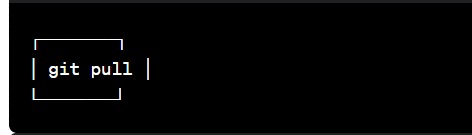


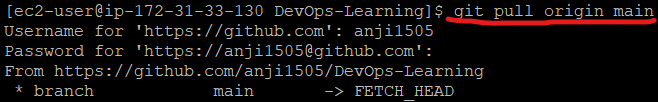
* Explanation:
* **git merge [another-file-name]**
* This command will combine multiple sequences of commits into one unified history. In the most frequent use cases, git merge is used to combine two branches. The git merge command takes two commit pointers, usually the branch tips, and finds a common base commit between them. Once it finds a common base commit, it will create a commit sequence.



**11. Git pull:**

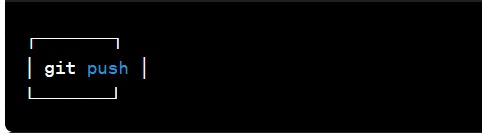
* Purpose: Fetches changes from a remote repository and merges them into the current branch.
* Example: git pull origin master



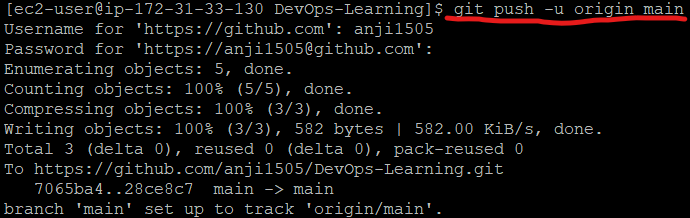
* Explanation:
* Usage: **git pull origin master**
* The git pull command first runs ‘git fetch’ which downloads the content from the specified remote repository and then immediately updates the local repo to match the content.

**12. Git push:**

* Purpose: Pushes local changes to a remote repository.
* Example: git push



* Explanation:
* Usage: **git push origin [branch name]**
* Suppose, we have made some changes in the file and want to push the changes to our remote repository on a particular branch. By using the command ‘git push,’ the local repository’s files can be synced with the remote repository on GitHub.



**13. Git fetch:**

* Purpose: Downloads objects and refs from another repository.
* Example: git fetch origin

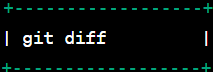


* Explanation:
* Usage**: Git fetch**
* When we use the command git fetch, Git gathers any commit from the target branch that does not exist in our current branch and stores it in our local repository. However, it does not merge it with our current branch.

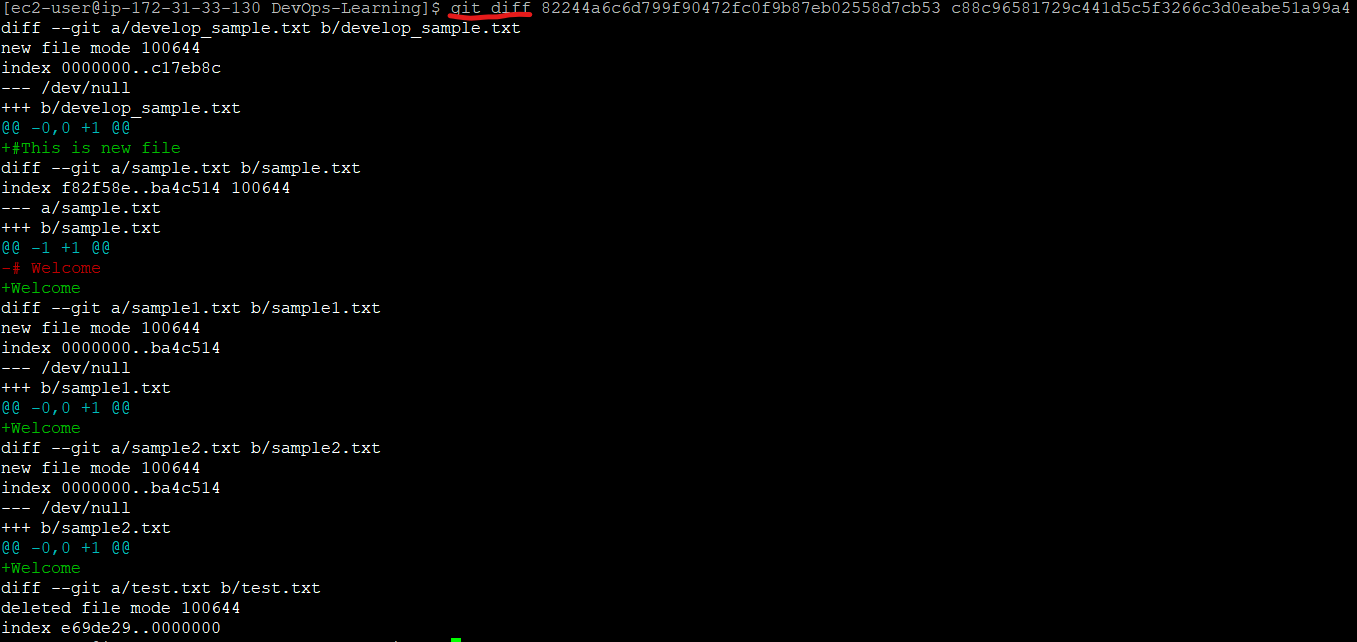


**14. Git diff:**

* Purpose: Shows changes between commits.
* Example: git diff

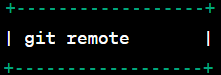


* Explanation:
* **Usage: git diff [commit-id-of-version-x] [commit-id-of-version-y]**
* Diffing is a function that takes two input datasets and outputs the changes between them. The git diff command is a multi-use Git command which, when executed, runs a diff function on Git data sources. These data sources can be commits, branches, files, and more. The git diff command is often used along with the git status and git log commands to analyze the current state of our Git repository. We use **git log** to get the details of commit IDs.
* Let’s compare the working directory with the index as shown below:

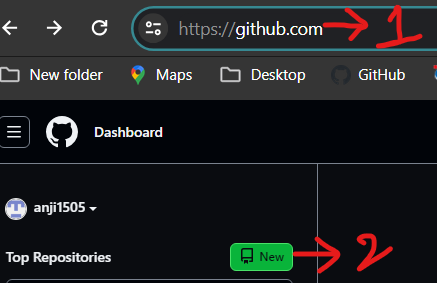


**15. Git remote:**

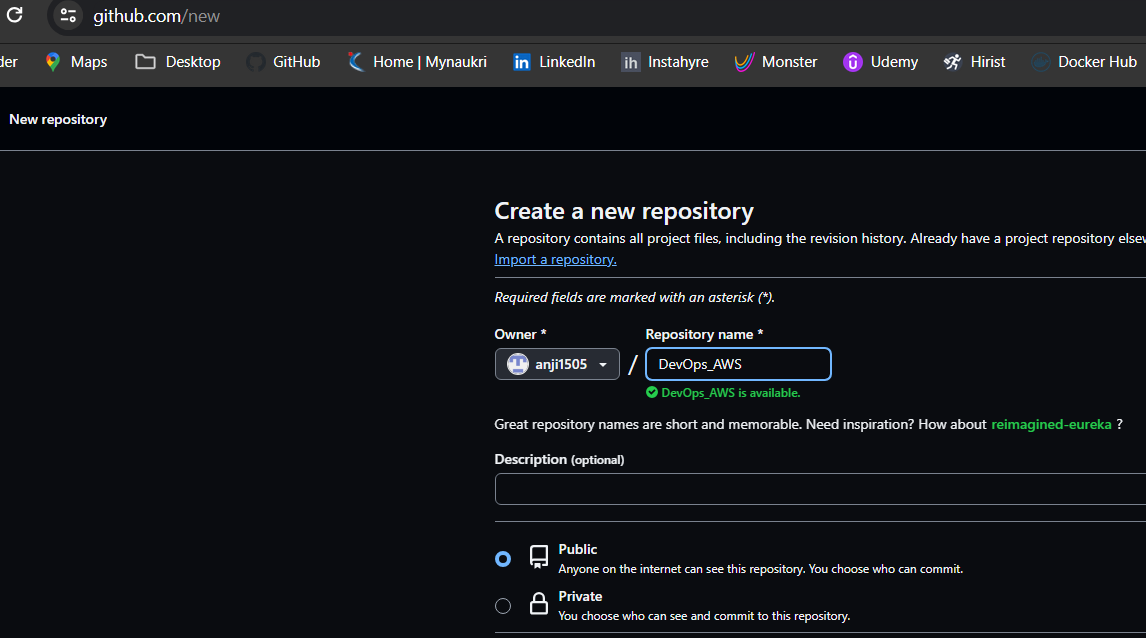
* Purpose: Manages a set of tracked repositories.
* Example: git remote



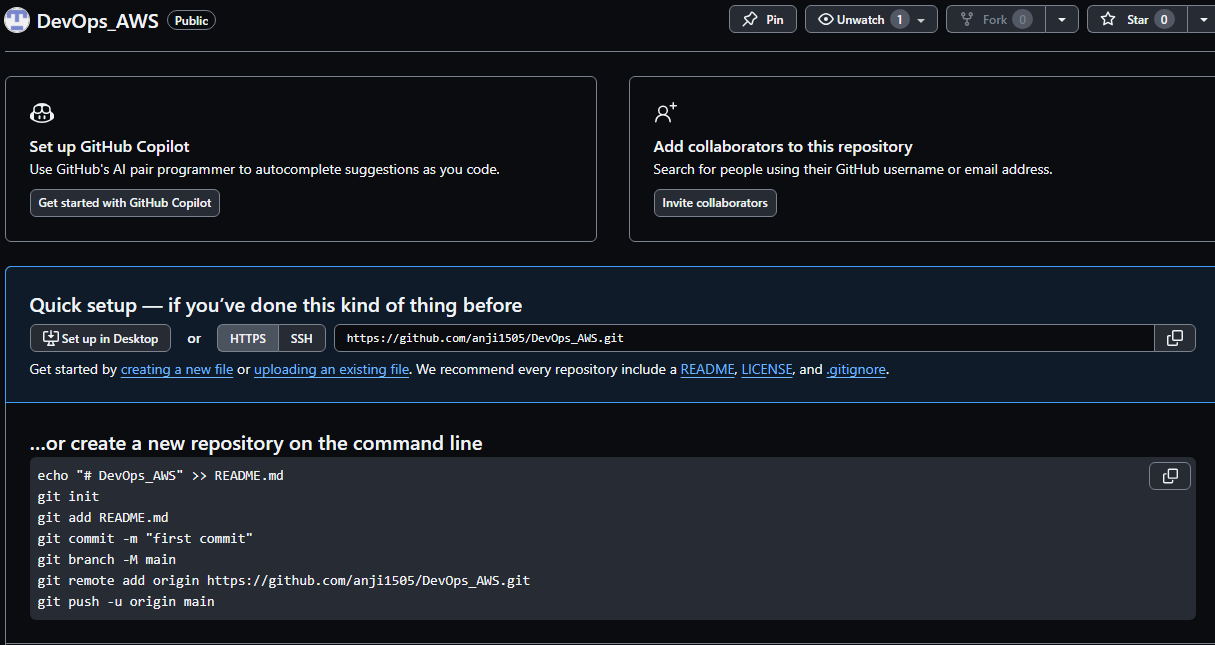
* Explanation:
* **Usage: git remote add origin “[URL]”**
* Once everything is ready on our local system, we can start pushing our code to the remote (central) repository of the project. For that, follow the below steps:
* **Step 1:**  
  **(1)**Login to the **GitHub account** if the account already exists (If not, sign up on github.com)  
  **(2)** Click on**New**



* **Step 2:**Now, we have to create a new repository. Provide a **name**to our**repository**, select the **privacy**of the repository as**Public**, and then click on **Create repository**



* Once we are done with filling up the new repository form, we would land on a page as follows:

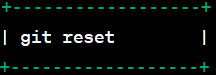


* **Step 3:** Click on the Copy icon on the right side of the URL box of the Github repository to copy the link and paste it as shown below:
* **git remote add origin “url”**
* Now, we are ready to operate the remote commands in our repository that we have just created.

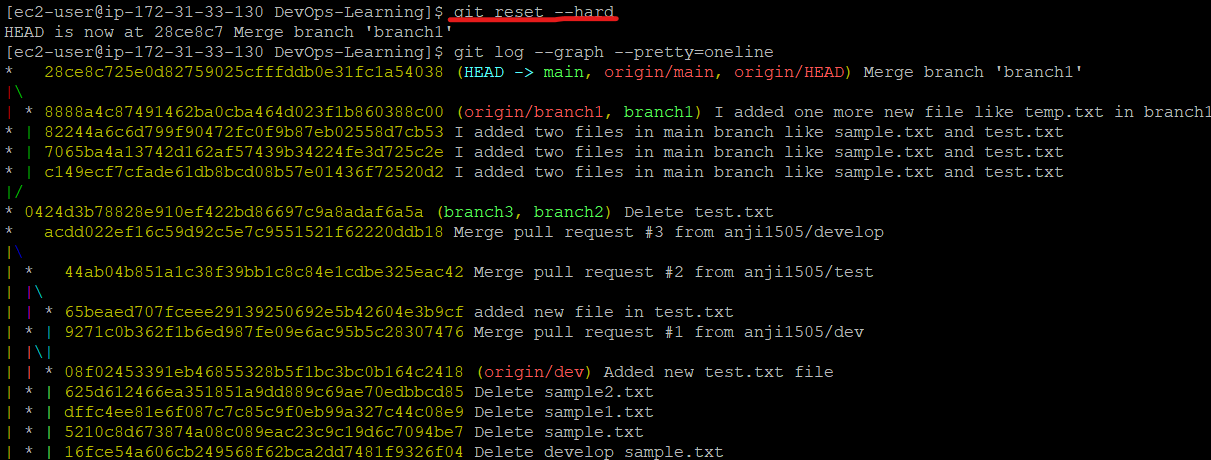


**16. Git reset:**

* Purpose: Resets the current HEAD (pointer) to a specified state.
* Example: git reset HEAD~1



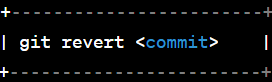
* Explanation:
* **git reset –hard [SOME-COMMIT]**
* We use this command to **return** the entire working tree to the last committed state.



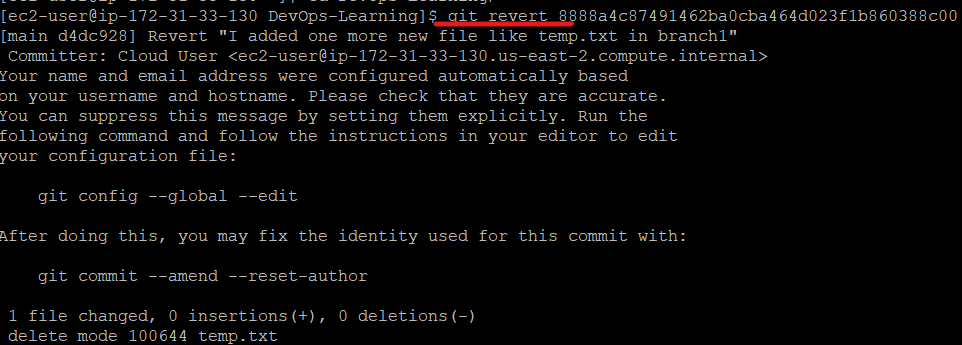
* This will discard commits in a private branch or throw away the uncommitted changes!
* Here, we have executed a ‘hard reset’ using the **–hard** option. Git displays the output indicating that the HEAD is pointing to the latest commit. Now, when we check the state of the repo with git status, Git will indicate that there are no pending changes (if any prior addition of a new file or modification of an existing file is done before using the ‘git reset –hard’ command). Our modifications to an existing file, if not committed, and the addition of a new file, if not staged, will be destroyed. It is critical to take note that this data loss cannot be undone.
* If we do**git reset –hard [SOME-COMMIT]**, then Git will**:**
* Make our current branch (typically master) back to point <SOME-COMMIT>
* Make the files in our working tree and the index (“staging area”) the same as the versions committed at <SOME-COMMIT>

**17. Git revert:**

* Purpose: Creates new commits that undo changes.
* Example: git revert <commit\_hash>



* Explanation:
* Usage: **git revert [commit id]**
* The git revert command can be considered as an ‘undo’ command. However, it does not work as the traditional ‘undo’ operation. It figures out how to invert the changes introduced by the commit and appends a new commit with the resulting inverse content.



**18. Git rebase:**

* Purpose: Reapplies commits on top of another base tip.
* Example: git rebase master

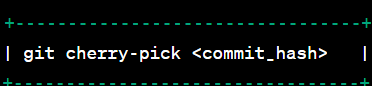


* Explanation:
* **Usage: git rebase [base]**
* **Rebase** is the process of moving and combining a sequence of commits to a new base commit. Rebasing is changing the base of our branch from one commit to another, making it appear as if we’ve created our branch from a different commit. Internally, Git accomplishes this by creating new commits and applying them to the specified base. It’s very important to understand that even though the branch looks the same, it is composed of entirely new commits.
* The git rebase command performs an automatic **git checkout <branch>** before doing anything else. Otherwise, it remains on the current branch.

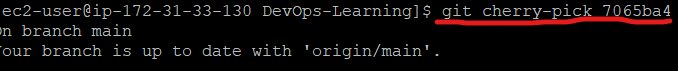
****

**19. Git cherry-pick:**

* Purpose: Applies changes introduced by some existing commits.
* Example: git cherry-pick <commit\_hash>

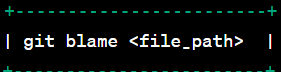


* Explanation:
* **Git cherry-pick** is a Git command used to apply changes from a specific commit onto the current branch. This allows you to selectively incorporate changes without merging entire branches. The command takes the commit's SHA, branch name, or a commit range as an argument, applying those changes to the current branch. If conflicts arise, manual resolution is required before completing the cherry-pick with **git cherry-pick --continue**.



**20.** **Git blame:**

* Purpose: Shows changes introduced by each commit for a specific file.
* Example: git blame file.txt



* Explanation:
* Shows who last modified each line in a file.



**21. Git stash:**

* Purpose: Temporarily saves changes for later use.
* Example: git stash



* Explanation:
* Usage: **git stash**
* This command can be used when we want to save our work without staging or committing the code to our Git repository and want to switch between branches.

****

* Usage-2**: git stash -u**
* This command is used when we want to stash the untracked files.

****

* Usage-3**: git stash pop**
* This command is used when we are back on our branch and want to retrieve the code.

****