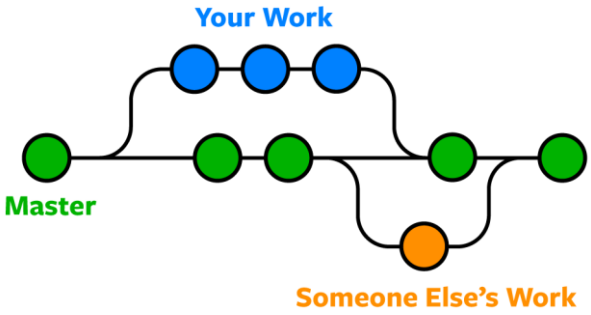
Day 10 Task: Advance Git & GitHub for DevOps Engineers.

Git Branching

A branch is a version of the repository that diverges from the main working project, In other words a branch is new/separate version of the main repository. It is a feature available in most modern version control systems. A Git project can have more than one branch.



These branches are a pointer to a snapshot of your changes. Branches allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository. So, it is complex to merge the unstable code with the main code base and also facilitates you to clean up your future history before merging with the main branch.

Git Revert and Reset

Two commonly used tools that git users will encounter are those of git reset and git revert. The benefit of both of these commands is that you can use them to remove or edit changes you've made in the code in previous commits.

Git Reset is used when we want to unstage a file and bring our changes back to the working directory. Git reset can also be used to remove commits from the local repository.

`git reset HEAD <filename>`

Git revert is used to remove the commits from the remote repository. Since now our changes are in the working directory, let's add those changes to the staging area and commit them.

git revert <commit id of the commit that needs to be removed>

Points to keep in mind –

- Using git revert we can undo any commit, not like git reset where we could just remove "n" recent commits.

Git Rebase and Merge

What Is Git Rebase?

Git rebase is a command that lets users integrate changes from one branch to another, and the logs are modified once the action is complete. Git rebase was developed to overcome merging's shortcomings, specifically regarding logs.

What Is Git Merge?

Git merge is a command that allows developers to merge Git branches while the logs of commits on branches remain intact.

The merge wording can be confusing because we have two methods of merging branches, and one of those ways is actually called "merge," even though both procedures do essentially the same thing.

Task 1:

Add a text file called version01.txt inside the Devops/Git/ with "This is first feature of our application" written inside. This should be in a branch coming from master, [hint try git checkout -b dev], swithch to dev branch (Make sure your commit message will reflect as "Added new feature"). [Hint use your knowledge of creating branches and Git commit command]

- version01.txt should reflect at local repo first followed by Remote repo for review. [Hint use your knowledge of Git push and git pull commands here]

```
ubuntu@ip-172-31-90-202:~/DevOps$ nano version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "Added feature in development br
anch"
[dev 7d1eace] Added feature in development branch
 1 file changed, 1 insertion(+), 1 deletion(-)
ubuntu@ip-172-31-90-202:~/DevOps$
```

Add new commit in dev branch after adding below mentioned content in Devops/Git/version01.txt: While writing the file make sure you write these lines

- 1st line>> This is the bug fix in development branch
- Commit this with message " Added feature2 in development branch"

```
ubuntu@ip-172-31-90-202:~/DevOps$ nano version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "Added feature2 in development brach"
[dev 5759e3b] Added feature2 in development brach
 1 file changed, 1 insertion(+)
ubuntu@ip-172-31-90-202:~/DevOps$
```

- 2nd line>> This is gadbad code
- Commit this with message " Added feature3 in development branch

```
ubuntu@ip-172-31-90-202:~/DevOps$ nano version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "Added feature3 in development branch"
[dev df1ee9b] Added feature3 in development branch
 1 file changed, 1 insertion(+)
ubuntu@ip-172-31-90-202:~/DevOps$
```

- 3rd line>> This feature will gadbad everything from now.

- Commit with message " Added feature4 in development branch

```
ubuntu@ip-172-31-90-202:~/DevOps$ nano version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git add version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "Added feature4 in development branch"
[dev ad4721d] Added feature4 in development branch
 1 file changed, 1 insertion(+)
```

Restore the file to a previous version where the content should be "This is the bug fix in development branch" [Hint use git revert or reset according to your knowledge]

```
ubuntu@ip-172-31-90-202:~/DevOps$ git log --oneline
ad4721d (HEAD -> dev) Added feature4 in development branch
df1ee9b Added feature3 in development branch
5759e3b Added feature2 in development brach
7d1eace Added feature in development branch
9604ed8 (master) Added new feature
ubuntu@ip-172-31-90-202:~/DevOps$
```

```
ubuntu@ip-172-31-90-202:~/DevOps$ git log
commit ad4721db5a0bf89089fa527fccd5c9ec11ba144a (HEAD -> dev)
Author: Your <Your@email.com>
Date:   Sat Jan 14 15:45:51 2023 +0000

    Added feature4 in development branch

commit df1ee9b72d4971e4b37d5f8208d8000f9fbce5e0
Author: Your <Your@email.com>
Date:   Sat Jan 14 15:43:34 2023 +0000

    Added feature3 in development branch

commit 5759e3b85ec55aff74358e8234050b16c19e76c5
Author: Your <Your@email.com>
Date:   Sat Jan 14 15:40:55 2023 +0000

    Added feature2 in development brach

commit 7d1eaceea6c4af7faa5913495b75576544333091
Author: Your <Your@email.com>
Date:   Sat Jan 14 15:36:48 2023 +0000

    Added feature in development branch

commit 9604ed8e672c4bcdcd1fc976634922ad785cf68d (master)
Author: Omkar <Omkar.email.com>
Date:   Sat Jan 14 14:58:27 2023 +0000
```

- Once commits are performed you can check log history with command
    - git log

After using git log you able to see the details : Unique commit id , Author details(Name & Email id) ,Date & Commit message

    - Later run the command : git reset <Commit id>

After using rest command if you check the log History again you will see only one commit as other commits deleted.

Task 2:

- Demonstrate the concept of branches with 2 or more branches with screenshot.

```
ubuntu@ip-172-31-90-202:~/DevOps$ git branch newdevops
ubuntu@ip-172-31-90-202:~/DevOps$ git branch
* dev
  master
  newdevops
```

- add some changes to dev branch and merge that branch in master

```
ubuntu@ip-172-31-90-202:~/DevOps$ git checkout master
Switched to branch 'master'
ubuntu@ip-172-31-90-202:~/DevOps$ ls
test.txt  version01.txt
ubuntu@ip-172-31-90-202:~/DevOps$ Devops git merge dev
Devops: command not found
ubuntu@ip-172-31-90-202:~/DevOps$ git merge dev
Updating 9604ed8..6c11526
Fast-forward
 version01.txt | 5 ++++-
 1 file changed, 4 insertions(+), 1 deletion(-)
ubuntu@ip-172-31-90-202:~/DevOps$ git branch
  dev
* master
  newdevops
ubuntu@ip-172-31-90-202:~/DevOps$ ls
test.txt  version01.txt
```

- as a practice try git rebase too, see what difference you get.

  git-rebase command is used to reapply commits on top of another base tip

```
ubuntu@ip-172-31-90-202:~/DevOps$ git commit -m "checking for rebase"
[master 09dcad0] checking for rebase
 2 files changed, 2 insertions(+)
 create mode 100644 rebase.txt
 create mode 100644 test.txt
ubuntu@ip-172-31-90-202:~/DevOps$ git rebase newdevops
Current branch master is up to date.
ubuntu@ip-172-31-90-202:~/DevOps$ git status
On branch master
nothing to commit, working tree clean
ubuntu@ip-172-31-90-202:~/DevOps$
```