

# GIT

(Global Information Tracker)

## Git and it's history:

Git came into existence in the year 2005 written in C programming language.

GIT is abbreviated as Global Information Tracker which is used to track the history of the files such as who has edited or made changes to the source code file, anyone who makes any changes the author name will be added so that it'd be easy to find out.

It is also called as a Source Code Management tool or Version Control System which is mainly used to move the source code to the repository to GitHub which can be later pulled or forked or cloned into any server to use the repo for CI/CD process. Git doesn't need GitHub but GitHub needs Git, without Git no source code file can be tracked, commit and pushed to the repo.

## STEP BY STEP PROCEDURE TO COMMIT AND PUSH A FILE TO A REPOSITORY:

- 1) Launch an ec2-instance with all free-tier eligible resources. You can either connect with console (with/without keypair), Putty (keypair) or MobaXterm (keypair).
- 2) Install git using '**git install -y**'.
- 3) Check if git is installed or using '**git --version**'.
- 4) Before git is initialized, create a folder (working directory) where you work on your project files to push the code to a particular repo to be available for further deployment.
- 5) Initialize the git using '**git init**'.
- 6) Now you can create files, track them and commit as well.

## Pictorial representation of the above steps:

- 1) Launched an ec2-instance and git is installed.

```

#
#####
#####\
\###|
\#/
V~' ->
    A newer version of Amazon Linux is available!

Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-22-236 ~]$ sudo -i
[root@ip-172-31-22-236 ~]# yum install git -y
Loaded plugins: extras_suggestions, langpacks, priorities, update-motd
amzn2-core
Resolving Dependencies
--> Running transaction check
--> Package git.x86_64 0:2.40.1-1.amzn2.0.3 will be installed
--> Processing Dependency: git-core = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: git-core-doc = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl-Git = 2.40.1-1.amzn2.0.3 for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Git) for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Processing Dependency: perl(Term::ReadKey) for package: git-2.40.1-1.amzn2.0.3.x86_64
--> Running transaction check
--> Package git-core.x86_64 0:2.40.1-1.amzn2.0.3 will be installed
--> Package git-core-doc.noarch 0:2.40.1-1.amzn2.0.3 will be installed
--> Package perl-Git.noarch 0:2.40.1-1.amzn2.0.3 will be installed
--> Processing Dependency: perl(Error) for package: perl-Git-2.40.1-1.amzn2.0.3.noarch
--> Package perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2 will be installed
--> Running transaction check
--> Package perl-Error.noarch 1:0.17020-2.amzn2 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

```

1.a) Installation of git is complete.

```

Installed:
  git.x86_64 0:2.40.1-1.amzn2.0.3

Dependency Installed:
  git-core.x86_64 0:2.40.1-1.amzn2.0.3
  perl-TermReadKey.x86_64 0:2.30-20.amzn2.0.2

Complete!

```

2) Ensuring that git is installed.

```

[ec2-user@ip-172-31-22-236 ~]$ sudo -i
[root@ip-172-31-22-236 ~]# git --version
git version 2.40.1

```

3) Creating a directory and initializing git.

```
[root@ip-172-31-22-236 ~]# mkdir mahi-repo
[root@ip-172-31-22-236 ~]# cd mahi-repo
[root@ip-172-31-22-236 mahi-repo]# ll
total 0
[root@ip-172-31-22-236 mahi-repo]# git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /root/mahi-repo/.git/
[root@ip-172-31-22-236 mahi-repo]# ll
total 0
```

4) A file is created for example and inserted some data into it. To save the entered data press ctrl d.

```
[root@ip-172-31-22-236 ~]# touch mahi.txt
[root@ip-172-31-22-236 ~]# ll
total 0
-rw-r--r-- 1 root root 0 Jun  8 14:16 mahi.txt
[root@ip-172-31-22-236 ~]# cat mahi.txt
[root@ip-172-31-22-236 ~]# cat > mahi.txt
This is the file to push into git repo
[root@ip-172-31-22-236 ~]# cat mahi.txt
This is the file to push into git repo
```

5) To track the file that we created, we need to add git. So that if any changes are made before it is committed we get to know about it.

```

[root@ip-172-31-22-236 ~]# cd mahi-repo/
[root@ip-172-31-22-236 mahi-repo]# ll
total 4
-rw-r--r-- 1 root root 39 Jun  8 14:16 mahi.txt
[root@ip-172-31-22-236 mahi-repo]# cat mahi.txt
This is the file to push into git repo
[root@ip-172-31-22-236 mahi-repo]# git add mahi.txt
[root@ip-172-31-22-236 mahi-repo]# git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   mahi.txt

```

6) After a file is tracked it will be in the staging area as it is rough draft space any changes can be again made before committing. Now at this stage we commit the file.

```

[root@ip-172-31-22-236 mahi-repo]# git commit -m 'first commit' mahi.txt
[master (root-commit) 59d880f] first commit
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 mahi.txt
[root@ip-172-31-22-236 mahi-repo]# git status
On branch master
nothing to commit, working tree clean

```

7) If you want to check if the file is committed or not you can check it by using git log it gives all the information about the commit id, branch it is committed to, author, date and time, with commit message as well.

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 59d880fb6891ab8caf30ec2e0c0fc17f1fc0eebc (HEAD -> master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 8 14:19:49 2024 +0000

    first commit
[root@ip-172-31-22-236 mahi-repo]#
```

8) A new branch (feature) is created, a new file is also created, tracked and committed. Therefore the branch for this file be feature instead of master as show below.

```
[root@ip-172-31-22-236 mahi-repo]# git branch feature
[root@ip-172-31-22-236 mahi-repo]# git branch
feature
* master
[root@ip-172-31-22-236 mahi-repo]# git checkout feature
Switched to branch 'feature'
[root@ip-172-31-22-236 mahi-repo]# touch ravi.pdf
[root@ip-172-31-22-236 mahi-repo]# cat> ravi.pdf
```

Here we're currently in the feature branch and the ravi.pdf file tracked but not yet committed. Also, we have the first file committed from the master and our 2nd file yet to be committed.

```
[root@ip-172-31-22-236 mahi-repo]# cat ravi.pdf
this is the 2nd commit in the feature branch.[root@ip-172-31-22-236 mahi-repo]# git add *
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   ravi.pdf

[root@ip-172-31-22-236 mahi-repo]# git log
commit 59d880fb6891ab8caf30ec2e0c0fc17f1fc0eebc (HEAD -> feature, master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 8 14:19:49 2024 +0000

    first commit
```

Now the 2nd file also been committed and now it shows changes made.

```
[root@ip-172-31-22-236 mahi-repo]# git commit -m "2nd commit" ravi.pdf
[feature 5426b12] 2nd commit
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 ravi.pdf
```

Here if you now check the git log, earlier we had only one commit and one branch, but now we've two files and two separate branches for them.

Master branch: git log (earlier)

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 59d880fb6891ab8caf30ec2e0c0fc17f1fc0eebc (HEAD -> master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 8 14:19:49 2024 +0000

    first commit
[root@ip-172-31-22-236 mahi-repo]#
```

Feature branch: git log (present after the creation of feature branch)

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 5426b12a466bb931a7eac106b992e0652b92b25d (HEAD -> feature)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Mon Jun 10 08:20:05 2024 +0000

    2nd commit

commit 59d880fb6891ab8caf30ec2e0c0fc17f1fc0eebc (master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 8 14:19:49 2024 +0000

    first commit
```

If you do not have files in your working directory to commit then if you give command `git status` it gives the below result.

```
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
nothing to commit, working tree clean
```

To check the number of branches in git, you can use the command `git branch`, and the result is as shown below.

```
[root@ip-172-31-22-236 mahi-repo]# git branch
* feature
master
```

9) To push a file to the git repository.

You can use the command '`git push`' to push the files to the repository.

- Create a new GitHub repo that contains a README file. ( create it on github )
- Use Git to clone the GitHub repo locally. ( **git clone "https url of the repo from github"** )
- Copy your project files into the folder created by the clone. ( **mv** to the created folder by clone)
- Perform a **git add .** and a **git commit -m 'msg name'** . ( track and commit )
- Push your changes up to GitHub. ( **git push origin -u branch name** if you've many or use main )

The below are the pictorial representation of the steps to push files to a remote repository on GitHub.

i) First create a file or ensure that there are files related to project are available in the local to track, commit and push.

If you want to create a new file just for a demo, you can create a file in the git initialized directory, then clone the repo into your local in the same git initialized directory.

Here: `cd mahi-repo` is the git initialized repository and `mahi-repo` in the directory is the remote repo of GitHub like `demo-repo`.

```

NN      \#/
NN      V~'  ->
NNN
NN.  .
  _/  _/
  /m/'

A newer version of Amazon Linux is available!

Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

No packages needed for security; 3 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-22-236 ~]$ sudo -i
[root@ip-172-31-22-236 ~]# cd mahi-repo
[root@ip-172-31-22-236 mahi-repo]# ll
total 0
drwxr-xr-x 3 root root 51 Jun 14 07:11 mahi-repo
[root@ip-172-31-22-236 mahi-repo]# touch test.pdf
[root@ip-172-31-22-236 mahi-repo]# vim test.pdf
[root@ip-172-31-22-236 mahi-repo]# git clone https://github.com/99mah43/demo-repo.git
Cloning into 'demo-repo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

ii) Go to your repositories > click on new > mention the name of the repo > add a description (optional) > select public or private > tick the box Add a README file > click on create repository.

The demo-repo has been created and doesn't have any files in it and cloned above to be available to push the files to GitHub repo.

a) The creation page of demo-repo.




# Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

Owner \*

 99mah43 ▾

Repository name \*

/ demo-repo

✔ demo-repo is available.

Great repository names are short and memorable. Need inspiration? How about **curly-octo-train** ?

Description (optional)

The repo for practice.



**Public**

Anyone on the internet can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

Initialize this repository with:



**Add a README file**

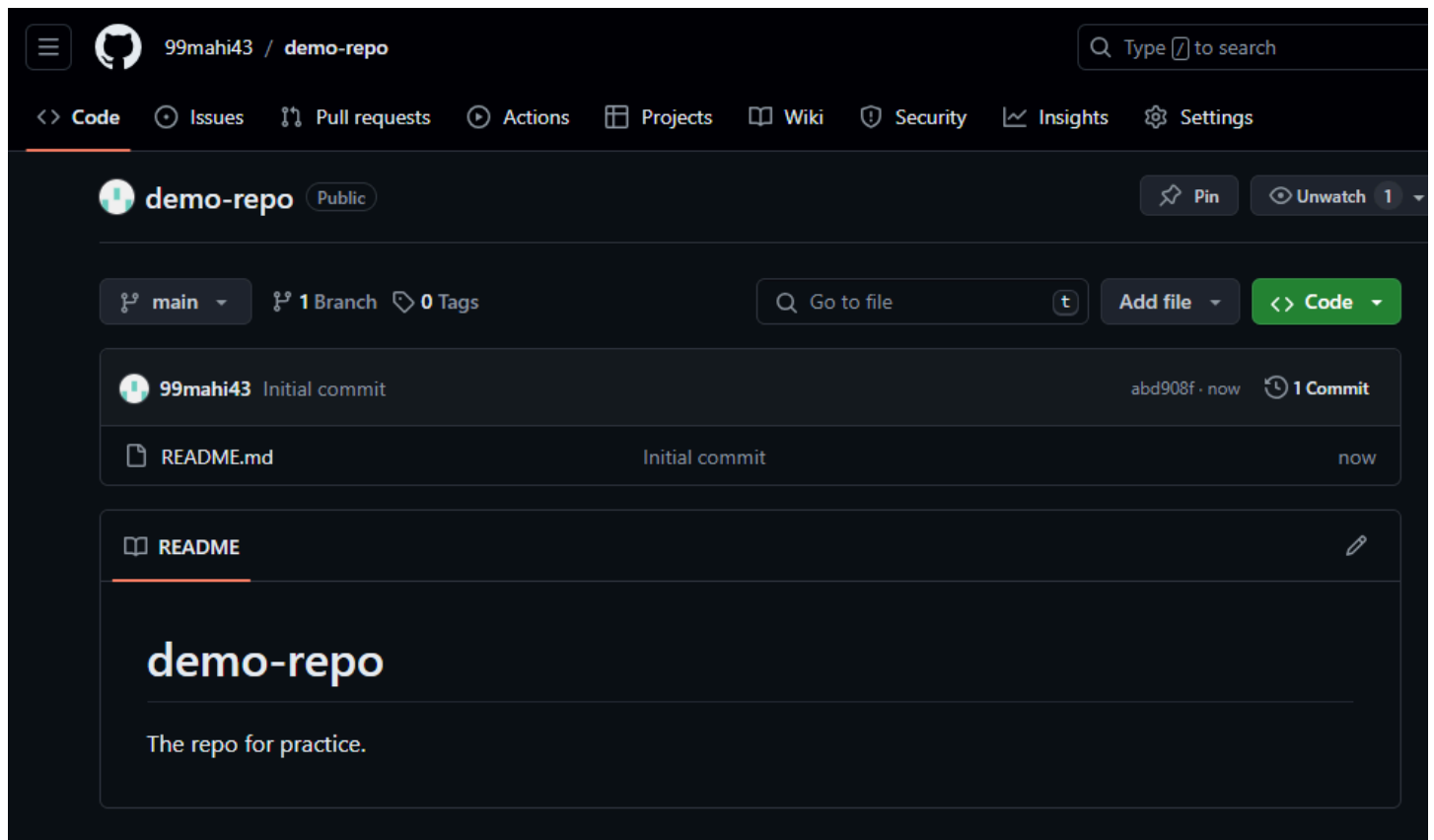
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

b) The demo-repo on GitHub platform.



iii) After cloning the demo-repo from GitHub we got the demo-repo available in our git initialized directory which is mahi-repo and not the blue one, it's also remote repo like demo-repo. We've copied/moved the project files (test.pdf) to README.md file in demo-repo so that all the files in the demo-repo can be pushed to remote repo of GitHub after tracked and committed.

```
[root@ip-172-31-22-236 mahi-repo]# ll
total 4
drwxr-xr-x 3 root root  35 Jun 14 07:42 demo-repo
drwxr-xr-x 3 root root  51 Jun 14 07:11 mahi-repo
-rw-r--r-- 1 root root 115 Jun 14 07:41 test.pdf
[root@ip-172-31-22-236 mahi-repo]# mv test.pdf demo-repo
[root@ip-172-31-22-236 mahi-repo]# cd demo-repo
[root@ip-172-31-22-236 demo-repo]# ll
total 8
-rw-r--r-- 1 root root  35 Jun 14 07:42 README.md
-rw-r--r-- 1 root root 115 Jun 14 07:41 test.pdf
[root@ip-172-31-22-236 demo-repo]# git add *
[root@ip-172-31-22-236 demo-repo]# git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   test.pdf

[root@ip-172-31-22-236 demo-repo]# git log
commit abd908f4afad68fae846c1c184818f5283355e7d (HEAD -> main, origin/main, origin/HEAD)
Author: MAHENDER Jangam <146831606+99mah43@users.noreply.github.com>
Date:   Fri Jun 14 13:07:04 2024 +0530

    Initial commit
```

iv) The file is committed and ready to be pushed to a remote repo on GitHub.

```
[root@ip-172-31-22-236 demo-repo]# git commit -m "demo commit" .
[main bb321ef] demo commit
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 2 insertions(+)
create mode 100644 test.pdf
[root@ip-172-31-22-236 demo-repo]# git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
[root@ip-172-31-22-236 demo-repo]# git log
commit bb321ef34feffab2df6e8f8c9e1a901d063b651f (HEAD -> main)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date:   Fri Jun 14 07:43:32 2024 +0000

    demo commit

commit abd908f4afad68fae846c1c184818f5283355e7d (origin/main, origin/HEAD)
Author: MAHENDER Jangam <146831606+99mahi43@users.noreply.github.com>
Date:   Fri Jun 14 13:07:04 2024 +0530

    Initial commit
```

v) Now the files in cloned demo-repo is pushed. In the above pic we've HEAD -> main as it is latest commit and after being pushed to a demo-repo then the main is changed to origin/main, origin/HEAD.

The username is your github username and the password is github token.

Go to Settings > Developer settings > Personal Access Tokens > Tokens (Classic) > Generate new token if the old one is expired.

```
[root@ip-172-31-22-236 demo-repo]# git push -u origin main
Username for 'https://github.com': 99mahi43
Password for 'https://99mahi43@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 369 bytes | 369.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/99mahi43/demo-repo.git
    abd908f..bb321ef  main -> main
branch 'main' set up to track 'origin/main'.
[root@ip-172-31-22-236 demo-repo]# git log
commit bb321ef34feffab2df6e8f8c9e1a901d063b651f (HEAD -> main, origin/main, origin/HEAD)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date:   Fri Jun 14 07:43:32 2024 +0000

    demo commit

commit abd908f4afad68fae846c1c184818f5283355e7d
Author: MAHENDER Jangam <146831606+99mahi43@users.noreply.github.com>
Date:   Fri Jun 14 13:07:04 2024 +0530

    Initial commit
[root@ip-172-31-22-236 demo-repo]#
```

vi) If you refresh the GitHub page, you'll get to see the test.pdf file is been pushed.

The screenshot shows the GitHub interface for a repository named 'demo-repo' by user '99mahi43'. The repository is public. The main branch is selected, showing 1 branch and 0 tags. The commit history table lists two commits: 'demo commit' by 'root' (bb321ef, 16 minutes ago) and 'Initial commit' (abd908f, 22 minutes ago). The file list shows 'README.md' (Initial commit, 22 minutes ago) and 'test.pdf' (demo commit, 16 minutes ago). The README content is visible, stating 'demo-repo' and 'The repo for practice.'

| Commit Hash | Author          | Message        | Time           |
|-------------|-----------------|----------------|----------------|
| bb321ef     | root            | demo commit    | 16 minutes ago |
| abd908f     | MAHENDER Jangam | Initial commit | 22 minutes ago |

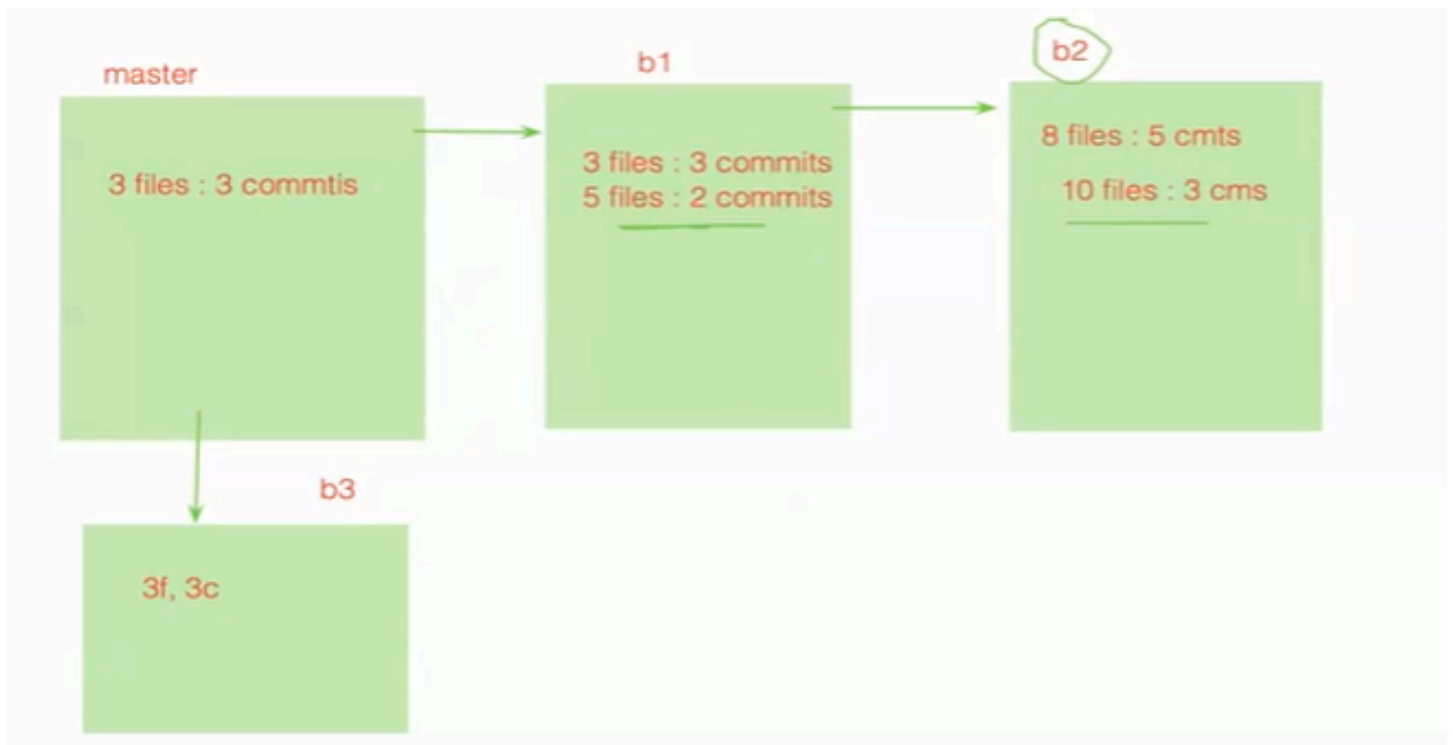
| File      | Commit         | Time           |
|-----------|----------------|----------------|
| README.md | Initial commit | 22 minutes ago |
| test.pdf  | demo commit    | 16 minutes ago |

**demo-repo**  
The repo for practice.

## GIT BRANCHES

A git branch is useful to isolate the environment to work on the source code based on the requirements. The crucial branch is master, and a few more branches such as feature, release. If you want to create a new branch like feature to work on any future requirements or enhancements. The command '**git branch branch-name**' i.e., **git branch feature** this will create a feature branch and if you use the command '**git checkout -b feature**' this will create a feature branch and you will be into the feature branch.

**NOTE:** Now matter which branch you're at, if you create a new branch while being in another branch you still get all the files from the previous branch to the newly created branch. Git has been built in that way where you can't stop including the files and history of commits from the previous branch to new branch.



→ First we've master branch with 3 files and 3 commits, if we create a branch 1 from master then we get 3 files and 3 commits from master branch to b1.

→ However, if we create 5 files and 2 commits in b1, we don't get to see them in master because the files and commits in one branch can't be seen in another branch, in order to see them we use merge.

→ Now, I have created b2 from b1 and I get total of 8 files and 5 commits, ( 3 x 3 from master and 5 x 2 from b1).

→ If I create b3 from master we don't get 8 files and 5 commits, instead we get 3 files and 3 commits from master. That's how the branching strategy works.

**GIT branching strategy:**

The main use of branching strategies is to avoid the disruptions in the production server, because whenever there's a new requirement typically a new feature or a service or an option that needs to be added to the existing application we can't make changes to the original source code that we've in our master branch which would be currently in the production server. Therefore, we use the branching strategy.

So we create a copy of master from the master branch called "Staging branch", from the staging branch we create a feature branch where we work on this branch regarding the new changes, it will be tested and once everything works fine, then the feature branch will be merged with the Master branch. And whenever it's ready to release into the market, then we create a Release branch and we deploy to make it available to the end user.

### Types of branches in git:

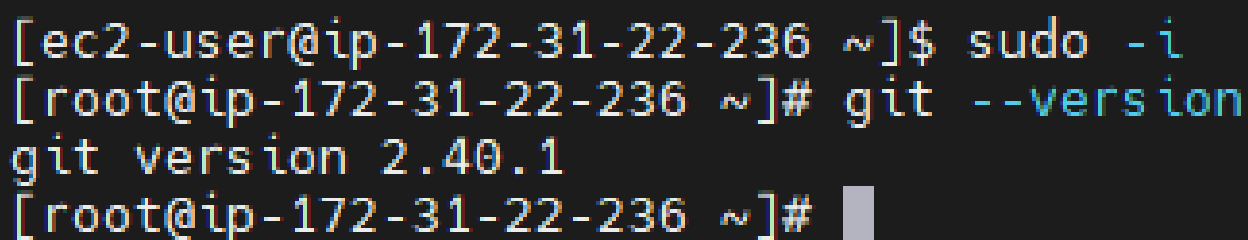
**1) MASTER BRANCH:** It's the active development branch that takes care of the original source code of the current running application in the production server.

**2) STAGING BRANCH:** It's the copy of the master branch, we take all the source code from the master branch.

**3) FEATURE BRANCH:** It's the branch where the future requirements or the new features that has to be added/introduced to the existing application will be coded, tested and will be merged to the master branch.

**4) RELEASE BRANCH:** After all the tests and all the release branch is created from the master during the release, then the code would be deployed to the production server. So since we're releasing the code to the market, it's called as the Release branch.

1) To implement the above concept, firstly check if the git is installed and initialized in the repository.

A terminal window screenshot with a black background and white text. The prompt is [ec2-user@ip-172-31-22-236 ~]\$ and the command is sudo -i. The prompt changes to [root@ip-172-31-22-236 ~]# and the command is git --version. The output is git version 2.40.1. The prompt returns to [root@ip-172-31-22-236 ~]#.

```
[ec2-user@ip-172-31-22-236 ~]$ sudo -i
[root@ip-172-31-22-236 ~]# git --version
git version 2.40.1
[root@ip-172-31-22-236 ~]#
```

2) This image shows a feature branch is created, shows the list of branches and we need to checkout to the created branch manually.

```
[root@ip-172-31-22-236 mahi-repo]# git branch feature
[root@ip-172-31-22-236 mahi-repo]# git branch
feature
* master
[root@ip-172-31-22-236 mahi-repo]# git checkout feature
Switched to branch 'feature'
```

3) This image represents the list of branches, if you use git checkout -b branch name it creates another new branch and we directly get into that branch instead of manually entering to go into the branch.

```
[root@ip-172-31-22-236 mahi-repo]# git branch
* feature
master
[root@ip-172-31-22-236 mahi-repo]# git checkout -b release
Switched to a new branch 'release'
[root@ip-172-31-22-236 mahi-repo]# git branch
feature
master
* release
```

**CONCLUSION:** The image 2 and 3 clearly represent the differences between manually getting inside a branch or checking out to the branch and in a automated way.

## GIT RESET

Git reset is useful to remove a specific count of commits, imagine if there are 10 commits and if you want to remove 9 counts you can do it by using git reset --hard HEAD~9. Therefore it will remove 9 commits and will leave the last/most recent commit in the commit history. By using HARD reset and SOFT reset you can't delete all the commits the most recent top commit will still be available there. In order to remove the last remaining commit or the most recent commit in the history we need to use the command

**git update-ref -d/D HEAD - - >**

**Pictorial representation of above concept:**

1) I've created a 3 files that has been added with some data and committed.



```
[root@ip-172-31-22-80 ~]# mkdir demo-repo
[root@ip-172-31-22-80 ~]# cd demo-repo/
[root@ip-172-31-22-80 demo-repo]# ll
total 0
[root@ip-172-31-22-80 demo-repo]# touch test exam sliptest
[root@ip-172-31-22-80 demo-repo]# cat>test
this si the test file
[root@ip-172-31-22-80 demo-repo]# cat>exam
this is the exam file
[root@ip-172-31-22-80 demo-repo]# cat>sliptest
this is the sliptest file
```

a) The first file 'test' has been committed.

```
[root@ip-172-31-22-80 demo-repo]# git add test
[root@ip-172-31-22-80 demo-repo]# git commit -m 'test commit' test
[feature (root-commit) c9b62a6] test commit
Committer: root <root@ip-172-31-22-80.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 demo-repo/test
```

b) All other remaining files have been committed.



```
[root@ip-172-31-22-80 demo-repo]# git add exam
[root@ip-172-31-22-80 demo-repo]# git commit -m 'exam commit' exam
[feature feab6fb] exam commit
Committer: root <root@ip-172-31-22-80.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 demo-repo/exam
[root@ip-172-31-22-80 demo-repo]# git add sliptest
[root@ip-172-31-22-80 demo-repo]# git commit -m 'sliptestcommit' sliptest
[feature 82b3383] sliptestcommit
Committer: root <root@ip-172-31-22-80.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 demo-repo/sliptest
```

## 2) Types of reset:

i) **SOFT RESET:** This reset is used to delete the commit history only, not the action i.e., like the file and the data within is not deleted.

**git reset --soft HEAD~1/2/3**

i (a) The commit history (git log).

```

[root@ip-172-31-22-80 demo-repo]# git log
commit 82b3383da8db97154c152c85c841a064e93fe912 (HEAD -> feature)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date:   Mon Jun 17 14:22:33 2024 +0000

    sliptestcommit

commit feab6fbab2925b16ad5c46737dfe7eb79a2eb2e6
Author: root <root@ip-172-31-22-80.ec2.internal>
Date:   Mon Jun 17 14:22:04 2024 +0000

    exam commit

commit c9b62a6cf12ceaab552c133a54d989aa35231ae3
Author: root <root@ip-172-31-22-80.ec2.internal>
Date:   Mon Jun 17 14:21:49 2024 +0000

    test commit
[root@ip-172-31-22-80 demo-repo]# git reset --soft HEAD~1

```

i (b) The picture represents that the soft reset (**HEAD~1**) has been applied and due to which the top commit (sliptest) has been deleted.

```

[root@ip-172-31-22-80 demo-repo]# git reset --soft HEAD~1
[root@ip-172-31-22-80 demo-repo]# git log
commit feab6fbab2925b16ad5c46737dfe7eb79a2eb2e6 (HEAD -> feature)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date:   Mon Jun 17 14:22:04 2024 +0000

    exam commit

commit c9b62a6cf12ceaab552c133a54d989aa35231ae3
Author: root <root@ip-172-31-22-80.ec2.internal>
Date:   Mon Jun 17 14:21:49 2024 +0000

    test commit
[root@ip-172-31-22-80 demo-repo]# cat sliptest
this is the sliptest file

```

ii (c) Even though the commit of sliptest has been deleted, but didn't the action.

```

[root@ip-172-31-22-80 demo-repo]# cat sliptest
this is the sliptest file

```

ii ) **HARD RESET**: This reset is used to delete the commit history along with the action (data).

**git reset --hard HEAD~1/2/3** respectively.

ii (a) After the hard reset, the commit along with the action (exam file) has been deleted and there's no data.

```
[root@ip-172-31-22-80 demo-repo]# git reset --hard HEAD~1
HEAD is now at c9b62a6 test commit
[root@ip-172-31-22-80 demo-repo]# git log
commit c9b62a6cf12ceaab552c133a54d989aa35231ae3 (HEAD -> feature)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date:   Mon Jun 17 14:21:49 2024 +0000

    test commit
[root@ip-172-31-22-80 demo-repo]# cat exam
cat: exam: No such file or directory
[root@ip-172-31-22-80 demo-repo]#
```

## GIT COMMANDS

1) To initialize git: **git init**

(either you can create a folder or you can choose a path or in root or based on requirement).

2) To install git: **yum install git -y**

3) To track a file: **git add filename**

4) To track multiple files: **git add file1 file2 file3**

5) To track all files: **git add \***

6) To untrack a file: **git rm --cached filename**

7) To commit a file: **git commit -m "commit message name"**

8) To commit all untracked files: **git commit -m "commit message name" .** (dot means all not pwd)

9) To check the status: **git status**

10) To check the history: **git log**

11) To get the history last 2/3 commits: **git log -2** or **git log -3**

12) To get the history of commit in stat format (++ , --) : **git show commit-id --stat**

13) To get commit-id's and commit messages: **git log --oneline** (gives partial commit ids and message of the commit)

```
[root@ip-172-31-22-236 mahi-repo]# git log --oneline
ffcb5d9 (HEAD -> release) exam file committed
200f666 (master) webinar file committed
75a9213 test file committed
```

14) To get full commit- id's and messages: **git log --pretty=oneline**

```
[root@ip-172-31-22-236 mahi-repo]# git log --pretty=oneline
ffcb5d976827e3b7e37a537676294913a68bd2eb (HEAD -> release) exam file committed
200f6665e13c7095f2e0dab8a043b535c08aadfa (master) webinar file committed
75a92136abedaa2a450da16988cb0e15d3f55a04 test file committed
```

15) To get full commits of a file: **git log commit --follow --all filename.**

16) To get the commit details along with the filename: **git show commit --name-only**

```
[root@ip-172-31-22-236 mahi-repo]# git show 29cedbdfdafd186e903a3e15c684f0a91bae316d --name-only
commit 29cedbdfdafd186e903a3e15c684f0a91bae316d (feature)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sun Jun 16 16:13:50 2024 +0000

    exam file committed

exam
```

17) To edit the latest commit message: **git commit --amend -m "msg-name"**

18) To commit the changes with the previous commit: **git commit --amend --no-edit**

(If there are 3 files but you committed 2 and want to commit the missing one to the same commit without a new commit, then you can use the above command).

19) To delete a commit and not data/action: **git reset --soft HEAD~1/2/3**

20) To delete a commit along with the data/action: **git reset --hard HEAD~1/2/3**

21) To delete the last remaining commit after soft or hard: **git update-ref -d HEAD**

22) To check all the deleted commits: **git reflog**

The history in reflog is by default set to expire in 90 days, however you can edit the time using the command **git reflog expire [--expire=<time>]**

23) To get the commit back after the deletion:

→ First check the commit id in the history using **git reflog**

→ Use the commit id and the command **git cherry-pick commit-id**

The configuration of username and email address can be done with the help of command and also by config file as well.

24) To create a branch : **git branch branch-name**

25) To edit branch name: **git branch -m branch-name new-branch name**

26) To delete a branch: **git branch -d branch name** and **git branch -D branch**

We always use -d to delete a branch, but if you create a branch from master or any other branch, you get all the commits from master to feature, if you have a few commits separately in feature and if you tend to delete the feature branch then it gives you an error as “not fully merged” and doesn’t get deleted. This is to avoid the data to loss from the feature, so before we delete, we merge the feature branch with master and then delete. If you do not need the commits in feature, then you can use the command with -D (forcefully).

27) To configure username: **git config user.name “our username”**

28) To configure e-mail address: **git config user.email “our e-mail address”**

### **TYPE-1: of configuring with the help of command.**

1) Username and email address is configured.

```
[root@ip-172-31-22-236 ~]# cd mahi-repo/
[root@ip-172-31-22-236 mahi-repo]# git config user.name 'kalki'
[root@ip-172-31-22-236 mahi-repo]# git config user-email 'kalki@2898ad.com'
error: key does not contain a section: user-email
[root@ip-172-31-22-236 mahi-repo]# git config user.email 'kalki@2898ad.com'
```

2) A file named kalki is created, tracked and committed. Now we do not have the author and email address as “salaar” instead we’ve “kalki”.

```

[root@ip-172-31-22-236 mahi-repo]# touch kalki.txt
[root@ip-172-31-22-236 mahi-repo]# vim kalki.txt
[root@ip-172-31-22-236 mahi-repo]# git add kalki.txt
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   kalki.txt
    deleted:    test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt

[root@ip-172-31-22-236 mahi-repo]# git commit -m 'kalki file committed' kalki.txt
[feature 0000a59] kalki file committed
 1 file changed, 1 insertion(+)
 create mode 100644 kalki.txt
[root@ip-172-31-22-236 mahi-repo]# git log
commit 0000a593e1d0c94bc4f1d02abc97d42a394f5f27 (HEAD -> feature)
Author: kalki <kalki@2898ad.com>
Date:   Mon Jun 10 14:50:03 2024 +0000

    kalki file committed

```

**TYPE-2: Another way is to input the details in the config file.**

1) Go to .git folder where the git is initialized and then go to config, you can add and change the username and email address as shown below.

```

[root@ip-172-31-22-236 mahi-repo]# ll -a
total 8
drwxr-xr-x 3 root root 84 Jun 10 13:24 .
dr-xr-x--- 4 root root 157 Jun 10 13:23 ..
drwxr-xr-x 8 root root 166 Jun 10 13:25 .git
-rw-r--r-- 1 root root 39 Jun 8 14:16 mahi.txt
-rw-r--r-- 1 root root 45 Jun 10 08:55 ravi.pdf
-rw-r--r-- 1 root root 0 Jun 10 13:24 salaar.pdf
-rw-r--r-- 1 root root 0 Jun 10 09:16 test.txt
[root@ip-172-31-22-236 mahi-repo]# cd .git/
[root@ip-172-31-22-236 .git]# ll
total 24
drwxr-xr-x 2 root root 6 Jun 8 14:17 branches
-rw-r--r-- 1 root root 22 Jun 10 13:25 COMMIT_EDITMSG
-rw-r--r-- 1 root root 153 Jun 10 13:23 config
-rw-r--r-- 1 root root 73 Jun 8 14:17 description
-rw-r--r-- 1 root root 24 Jun 10 09:28 HEAD
drwxr-xr-x 2 root root 4096 Jun 8 14:17 hooks
-rw-r--r-- 1 root root 289 Jun 10 13:25 index
drwxr-xr-x 2 root root 21 Jun 8 14:17 info
drwxr-xr-x 3 root root 30 Jun 8 14:19 logs
drwxr-xr-x 15 root root 140 Jun 10 13:25 objects
drwxr-xr-x 4 root root 31 Jun 8 14:17 refs
[root@ip-172-31-22-236 .git]# vim config

```



2) The config file where the credentials needs to be added.

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true

[user]
    name = salaar
    email = salaar@khansaar.com

~
~
~
```

3) Here I've created a salaar.pdf file and tracked.

```
[root@ip-172-31-22-236 mahi-repo]# touch salaar.pdf
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    deleted:    test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    salaar.pdf
    test.txt

[root@ip-172-31-22-236 mahi-repo]# git add salaar.pdf
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    test.txt -> salaar.pdf

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt
```

4) The file has been committed.

```
[root@ip-172-31-22-236 mahi-repo]# git commit -m 'salaar file committed' salaar.pdf
[feature ad268d7] salaar file committed
Committer: salaar <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 salaar.pdf
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt
```

5) Now if you check the author and e-mail address of the commit of the above it shows the details.

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 5c69daebe0d53885f6fb9304606c397e897e5a7c (HEAD -> feature)
Author: salaar <salaar@khansaar.com>
Date:   Mon Jun 10 13:31:29 2024 +0000

    salaar file committed
```

This is how the username and email address can be added without a command as well.

29) To fetch the changes from github: **git fetch**

30) To merge the changes from github: **git merge origin/branch-name**

31) To push the changes from local to github: **git push origin -u branch-name**

32) To pull the changes from github: **git pull**

**NOTE: PULL before you PUSH** ( If there are 3 commits in git and same 3 commits in github, but if there's a 4th commit that was made on github and if you try to do another 5th commit in git in local, then you get an error. In these situations, first you need to pull the change from github to local and then again you can push the 5th commit to github. The sequential order needs to be followed.)

33) To push or pull the changes to a remote repository from local git repository:

**git remote add origin https URL of the github repo**



34) To shows the URLs of the remote repositories associated with your local repository: **git remote -v**

## **GIT STASH**

Let's say there are two branches master and feature, in master we have 3 commits and 3 files and we created feature branch where we get all the data from master. In feature branch, if we write a few lines of code (LOC) again and until and unless we commit we can't go to the other branch (master branch). In order to go to the master without committing we can use stash this will help in temporarily delete the data and get back to feature to remove the stash to get the previously written LOC back again.

### **Stash commands:**

- 1) To store/delete the files temporarily - - > **git stash**
- 2) To get those files back to work upon/to commit - - > **git stash apply/ git stash pop**
- 3) To clear the stash/ to clear the files that were stashed - - > **git stash clear**
- 4) To check the list of stash - - > **git stash list**
- 5) To delete a specific stash - - > **git drop stash-name**
- 6) The files that are stashed/temporarily deleted are stored in the project's **.git directory; refs/stash**.

### **Practical representation of the usage of stash:**

- 1) First I have created 2 files and individually committed them.

```
drwxr-xr-x 2 root root 6 Jun 17 12:30 project-K
[root@ip-172-31-22-80 ~]# cd project-K/
[root@ip-172-31-22-80 project-K]# ll
total 0
[root@ip-172-31-22-80 project-K]# touch kalki
[root@ip-172-31-22-80 project-K]# cat>kalki
it's a myth sci-fi film
[root@ip-172-31-22-80 project-K]# cat kalki
it's a myth sci-fi film
[root@ip-172-31-22-80 project-K]# git add kalki
[root@ip-172-31-22-80 project-K]# git commit -m 'kalki commit' kalki
[master (root-commit) c8e3e17] kalki commit
Committer: root <root@ip-172-31-22-80.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:
```

```
git config --global --edit
```

After doing this, you may fix the identity used for this commit with:

```
git commit --amend --reset-author
```

```
1 file changed, 1 insertion(+)
create mode 100644 project-K/kalki
[root@ip-172-31-22-80 project-K]# git log
commit c8e3e17e0a66fe3b2c3c581f916d8b633ad3a2cd (HEAD -> master)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:32:48 2024 +0000
```

```
kalki commit
```

```
[root@ip-172-31-22-80 project-K]# git log
commit 5e69b09b5ec99ceec4d639d73d1820f229d64e4f (HEAD -> master)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:34:25 2024 +0000

    salaar commit

commit cb931693ae68c7ff89fd3616313a84f8231991f2
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:33:26 2024 +0000

    sahuo commit

commit c8e3e17e0a66fe3b2c3c581f916d8b633ad3a2cd
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:32:48 2024 +0000

    kalki commit
```

2) I've checked out to a new branch feature and got all the previous commits, files into this feature.

```
[root@ip-172-31-22-80 project-K]# git checkout -b feature
Switched to a new branch 'feature'
[root@ip-172-31-22-80 project-K]# git log
commit 5e69b09b5ec99ceec4d639d73d1820f229d64e4f (HEAD -> feature, master)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:34:25 2024 +0000

    salaar commit

commit cb931693ae68c7ff89fd3616313a84f8231991f2
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:33:26 2024 +0000

    sahuo commit

commit c8e3e17e0a66fe3b2c3c581f916d8b633ad3a2cd
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:32:48 2024 +0000

    kalki commit
```

3) At the feature branch, I've created a new file named spirit, tracked and committed for the first time. Later when I added another line of data and trying to switch to another branch (master) getting an error. So in these situations “**STASH**” is helpful.

With the help of stash we can checkout to master and come back to feature again without committing the 2nd line of code. Now after coming back to the feature branch we can use the command ‘**git stash pop**’ so

that we can see the 2nd line of code back again in the file and then later, if we wish, we can delete the code by git stash clear so the stash that was created will be removed.

3 (i) Creating a new file with name Spirit, added a single line of data, tracked and committed for the first time.

```
[root@ip-172-31-22-80 project-K]# touch spirit
[root@ip-172-31-22-80 project-K]# cat>spirit
it's a prabhas starrer film
[root@ip-172-31-22-80 project-K]# cat spirit
it's a prabhas starrer film
[root@ip-172-31-22-80 project-K]# git add spirit
[root@ip-172-31-22-80 project-K]# git commit -m 'spirit commit' spirit
[feature 46bad68] spirit commit
Committer: root <root@ip-172-31-22-80.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 project-K/spirit
```

3 (ii) Now we've the commit history of all the files.

```
[root@ip-172-31-22-80 project-K]# git log
commit 46bad686c76e74ce6bed5af12edfeeb85498daaa (HEAD -> feature)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:35:26 2024 +0000

    spirit commit

commit 5e69b09b5ec99ceec4d639d73d1820f229d64e4f (master)
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:34:25 2024 +0000

    salaar commit

commit cb931693ae68c7ff89fd3616313a84f8231991f2
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:33:26 2024 +0000

    sahuo commit

commit c8e3e17e0a66fe3b2c3c581f916d8b633ad3a2cd
Author: root <root@ip-172-31-22-80.ec2.internal>
Date: Mon Jun 17 12:32:48 2024 +0000

    kalki commit
[root@ip-172-31-22-80 project-K]#
```

3 (iii) The second line of the data has been added in the file again and not tracked, committed.

```
[root@ip-172-31-22-80 project-K]# cat>>spirit
it's a sandeep vanga film
```

3 (iv) Here's the whole data in the file.

```
[root@ip-172-31-22-80 project-K]# cat spirit
it's a prabhas starrer film
it's a sandeep vanga film
```

3 (v) When I am trying to checkout, it throws an error.

```
[root@ip-172-31-22-80 project-K]# git checkout master
error: Your local changes to the following files would be overwritten by checkout:
  project-K/spirit
Please commit your changes or stash them before you switch branches.
Aborting
```

3 (vi) Now in order to switch out, git stash is applied. After being switched out, if you check the list we got only two files. If you check out to feature branch, now if you check the spirit file you only have the first line of data, the second line of data is not visible.

```
[root@ip-172-31-22-80 project-K]# git stash
Saved working directory and index state WIP on feature: 46bad68 spirit commit
[root@ip-172-31-22-80 project-K]# git checkout master
Switched to branch 'master'
[root@ip-172-31-22-80 project-K]# ll
total 12
-rw-r--r-- 1 root root 24 Jun 17 12:32 kalki
-rw-r--r-- 1 root root 27 Jun 17 12:33 sahuo
-rw-r--r-- 1 root root 31 Jun 17 12:34 salaar
[root@ip-172-31-22-80 project-K]# git checkout feature
Switched to branch 'feature'
[root@ip-172-31-22-80 project-K]# lll
-bash: lll: command not found
[root@ip-172-31-22-80 project-K]# ls
kalki sahuo salaar spirit
[root@ip-172-31-22-80 project-K]# cat spirit
it's a prabhas starrer film
```

3 (vii) To get the 2nd line of data back, we need to use the command git stash pop. The spirit file got into modified state and now if you check the spirit file you've 2 lines of data back again.

## INTERVIEW QUESTIONS

**1) Is Git a distributed or centralized version control system? What is the difference between them?**

A: Git is a distributed version control system (DVCS) because each developer can have a whole copy of the project by cloning the repo and work on it,

whereas in centralized version control system (CVCS) it works on a client-server model, there's a server that acts as a main centralized repository which stores the code and due to which no two developers can work on the same code at the same time which would arise conflicts. In distributed many work on same code by cloning whereas in centralized it's not possible that's the difference.

However, centralized is useful with small teams where they can communicate but it is not ideal to use in larger number of developers work. In centralized if one developer is writing or making changes to the

piece of code the second can't work as it will be locked and once it is merged to the central repo then the other developer can work so it is not efficient way and affects the productivity as well.

Therefore, organizations prefer Distributed one over the centralized one model of version control system.

## 2) What is the workflow of Git?

A: So if we want to push a file to a repository, first we - - > create a file (touch demo) - - > add git to the file using '**git add**', once git is added that means that the file is in tracking state and will be moved to the staging area (here the changes can be made as it's a rough draft space), until and unless we commit, it can't be pushed to any repo. So we commit the file by using the command '**git commit -m "message name" filename**' and later it can be pushed into a repo.

## 3) What are the different stages of a Git file?

A: There are 3 different stages in GIT:

**i) Working directory:** This is the place where the git is initialized, the files are created which can be added with git and can be committed. This is the place where we've all the untracked files.

**ii) Staging area:** This is the place is where all the tracked files exist which are ready to commit.

**iii) Committed:** This is the place where we've all the committed files which can be moved to repo.

**NOTE:** If you make any changes to the committed file, then the file goes to the staging area as modified, the file needs to be committed again to go to any repo.

## 4) How do you create or initialize the git?

A: You can initialize the git in the working directory by using the 'git init' command. After initializing the git, you get .git folder in your local and this is responsible for tracking files, untracked files and everything that are related to git. In case if this folder is removed then all the work is gone.

## 5) How to track a file in git?

A: To track a file in git, we need to add the git to the file. The command would be "git add filename".

## 6) How to commit a file in git?

A: In order to commit a file, the file needs to be tracked first and then we need to commit, the command would be "git commit -m "message name" filename".

## 7) What are the Git commands that you use to commit the changes to your remote repository?

A: In order to commit the changes to the remote repository, we use three commands that are `git add`, `git commit` and `git push`.

## 8) What do you need to do if the developers mistakenly or unknowingly do not want to push your password to git?

A: All you need to do is to go to `.git -> hooks -> pre-commit hook` which will prevent in pushing the password to git.

## 9) How do you push the files in local to the repository?

A: You can use the command '`git push`' to push the files to the repository.

→ Create a new GitHub repo that contains a README file. ( create it on github )

→ Use Git to clone the GitHub repo locally. ( **`git clone "https url of the repo from github"`** )

→ Copy your project files into the folder created by the clone. ( **`mv`** to the created folder by clone)

→ Perform a **`git add .`** and a **`git commit -m 'msg name'`** . ( track and commit )

→ Push your changes up to GitHub. ( **`git push origin -u branch name`** if you've many or use main )

The below are the pictorial representation of the steps to push files to a remote repository on GitHub.

1) First create a file or ensure that there are files related to project are available in the local to track, commit and push.

If you want to create a new file just for a demo, you can create a file in the git initialized directory, then clone the repo into your local in the same git initialized directory.

Here: `cd mahi-repo` is the git initialized repository and `mahi-repo` in the directory is the remote repo of GitHub like `demo-repo`.



```

NN      \#/
NN      V~'  ->
NNN
NN  .
  _/
  /m/'

A newer version of Amazon Linux is available!

Amazon Linux 2023, GA and supported until 2028-03-15.
https://aws.amazon.com/linux/amazon-linux-2023/

No packages needed for security; 3 packages available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-22-236 ~]$ sudo -i
[root@ip-172-31-22-236 ~]# cd mahi-repo
[root@ip-172-31-22-236 mahi-repo]# ll
total 0
drwxr-xr-x 3 root root 51 Jun 14 07:11 mahi-repo
[root@ip-172-31-22-236 mahi-repo]# touch test.pdf
[root@ip-172-31-22-236 mahi-repo]# vim test.pdf
[root@ip-172-31-22-236 mahi-repo]# git clone https://github.com/99mah43/demo-repo.git
Cloning into 'demo-repo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

2) Go to your repositories > click on new > mention the name of the repo > add a description (optional) > select public or private > tick the box Add a README file > click on create repository.

The demo-repo has been created and doesn't have any files in it and cloned above to be available to push the files to GitHub repo.


a) The creation page of demo-repo.

# Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

*Required fields are marked with an asterisk (\*).*

Owner \*

 99mah43 ▾

Repository name \*

/ demo-repo

✔ demo-repo is available.

Great repository names are short and memorable. Need inspiration? How about **curly-octo-train** ?

Description (optional)

The repo for practice.



**Public**

Anyone on the internet can see this repository. You choose who can commit.



**Private**

You choose who can see and commit to this repository.

Initialize this repository with:



**Add a README file**

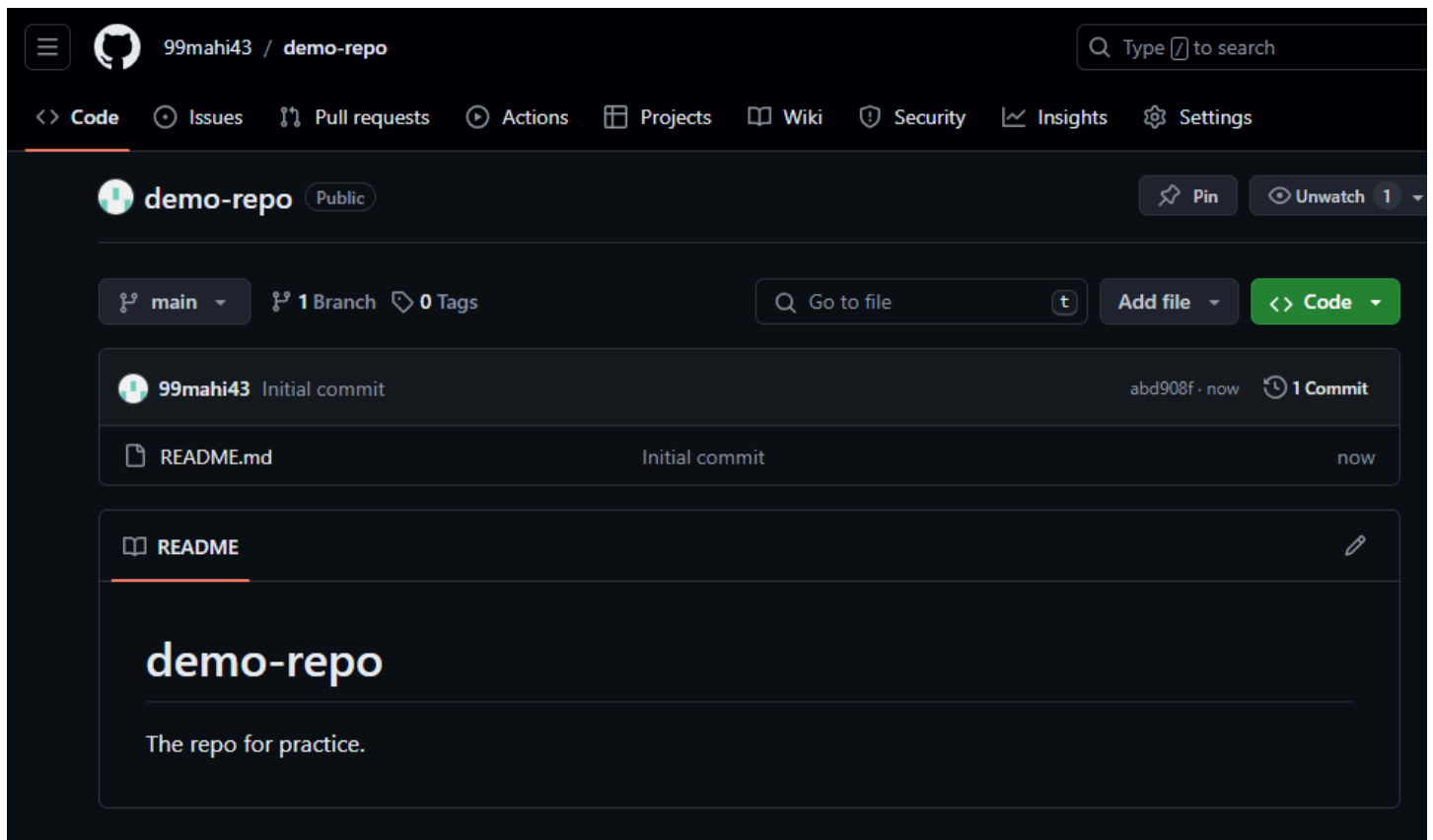
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

b) The demo-repo on GitHub platform.



3) After cloning the demo-repo from GitHub we got the demo-repo available in our git initialized directory which is mahi-repo and not the blue one, it's also remote repo like demo-repo. We've copied/moved the project files (test.pdf) to README.md file in demo-repo so that all the files in the demo-repo can be pushed to remote repo of GitHub after tracked and committed.

```
[root@ip-172-31-22-236 mahi-repo]# ll
total 4
drwxr-xr-x 3 root root  35 Jun 14 07:42 demo-repo
drwxr-xr-x 3 root root  51 Jun 14 07:11 mahi-repo
-rw-r--r-- 1 root root 115 Jun 14 07:41 test.pdf
[root@ip-172-31-22-236 mahi-repo]# mv test.pdf demo-repo
[root@ip-172-31-22-236 mahi-repo]# cd demo-repo
[root@ip-172-31-22-236 demo-repo]# ll
total 8
-rw-r--r-- 1 root root  35 Jun 14 07:42 README.md
-rw-r--r-- 1 root root 115 Jun 14 07:41 test.pdf
[root@ip-172-31-22-236 demo-repo]# git add *
[root@ip-172-31-22-236 demo-repo]# git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   test.pdf

[root@ip-172-31-22-236 demo-repo]# git log
commit abd908f4afad68fae846c1c184818f5283355e7d (HEAD -> main, origin/main, origin/HEAD)
Author: MAHENDER Jangam <146831606+99mah43@users.noreply.github.com>
Date:   Fri Jun 14 13:07:04 2024 +0530

    Initial commit
```

4) The file is committed and ready to be pushed to a remote repo on GitHub.

```
[root@ip-172-31-22-236 demo-repo]# git commit -m "demo commit" .
[main bb321ef] demo commit
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 2 insertions(+)
create mode 100644 test.pdf
[root@ip-172-31-22-236 demo-repo]# git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
[root@ip-172-31-22-236 demo-repo]# git log
commit bb321ef34feffab2df6e8f8c9e1a901d063b651f (HEAD -> main)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date:   Fri Jun 14 07:43:32 2024 +0000

    demo commit

commit abd908f4afad68fae846c1c184818f5283355e7d (origin/main, origin/HEAD)
Author: MAHENDER Jangam <146831606+99mahi43@users.noreply.github.com>
Date:   Fri Jun 14 13:07:04 2024 +0530

    Initial commit
```

5) Now the files in cloned demo-repo is pushed. In the above pic we've HEAD -> main as it is latest commit and after being pushed to a demo-repo then the main is changed to origin/main, origin/HEAD.

The username is your github username and the password is github token.

Go to Settings > Developer settings > Personal Access Tokens > Tokens (Classic) > Generate new token if the old one is expired.

```
[root@ip-172-31-22-236 demo-repo]# git push -u origin main
Username for 'https://github.com': 99mahi43
Password for 'https://99mahi43@github.com':
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 369 bytes | 369.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/99mahi43/demo-repo.git
    abd908f..bb321ef  main -> main
branch 'main' set up to track 'origin/main'.
[root@ip-172-31-22-236 demo-repo]# git log
commit bb321ef34feffab2df6e8f8c9e1a901d063b651f (HEAD -> main, origin/main, origin/HEAD)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date:   Fri Jun 14 07:43:32 2024 +0000

    demo commit

commit abd908f4afad68fae846c1c184818f5283355e7d
Author: MAHENDER Jangam <146831606+99mahi43@users.noreply.github.com>
Date:   Fri Jun 14 13:07:04 2024 +0530

    Initial commit
[root@ip-172-31-22-236 demo-repo]#
```

6) If you refresh the GitHub page, you'll get to see the test.pdf file is been pushed.

The screenshot shows the GitHub interface for a repository named 'demo-repo' by user '99mahi43'. The repository is public. The main branch is selected, showing 1 branch and 0 tags. The commit history table lists three commits:

| Commit Hash | Author          | Message        | Time           |
|-------------|-----------------|----------------|----------------|
| bb321ef     | root            | demo commit    | 16 minutes ago |
| abd908f     | MAHENDER Jangam | Initial commit | 22 minutes ago |
| abd908f     | MAHENDER Jangam | Initial commit | 22 minutes ago |

The file list shows two files: 'README.md' (Initial commit, 22 minutes ago) and 'test.pdf' (demo commit, 16 minutes ago). The README content is displayed below the file list, showing the title 'demo-repo' and the description 'The repo for practice.'

10) What is the difference between git clone and git fork?

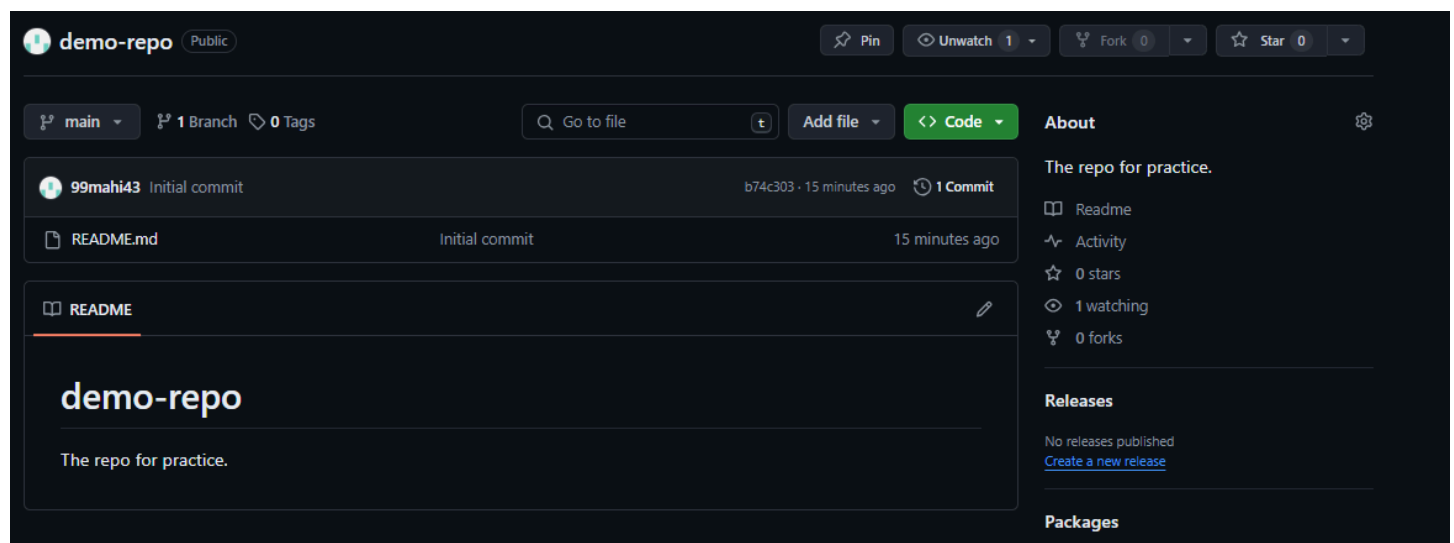
A: The difference here is, git clone is used to download the repo from the GitHub and git fork is used to get a copy of the repository from the GitHub. The fundamental advantage of Git is it's a Distributed Version Control System (DVCS) and it is justified by the git fork, with this, a repo can be distributed anywhere in the world.

## 11) What is the difference between git fetch and git pull?

A: git fetch checks the changes that are made in remote repo and retrieves the changes to the local repo like if there are any updates to the repo or if there are any significant changes but it doesn't merge the details to the working directory. Whereas git pull not only fetches the changes in the remote repo but also merges to the working directory, that's the whole difference.

Git pull is a combination of git fetch and git merge, first checks the changes - - > retrieves to the local repo - - > merges it to the working directory of the local branch - - > if there are any conflict changes we need to resolve it and then merge and if no conflicts, it can be merged directly.

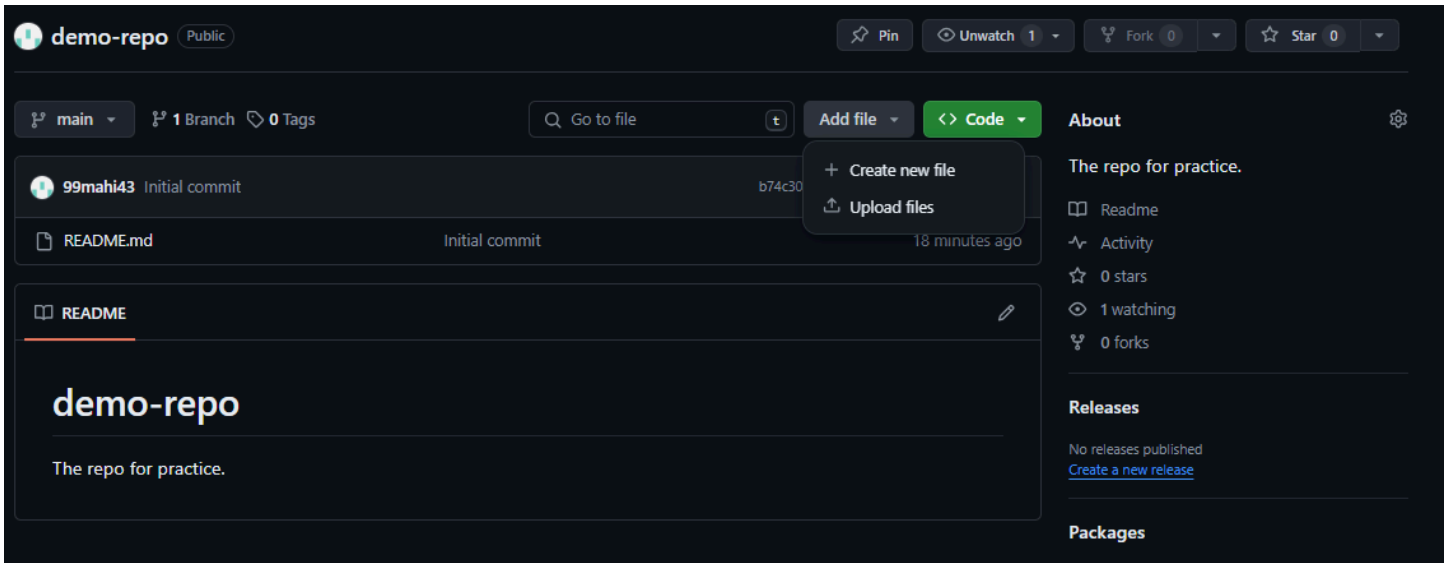
1) The remote repo on github.



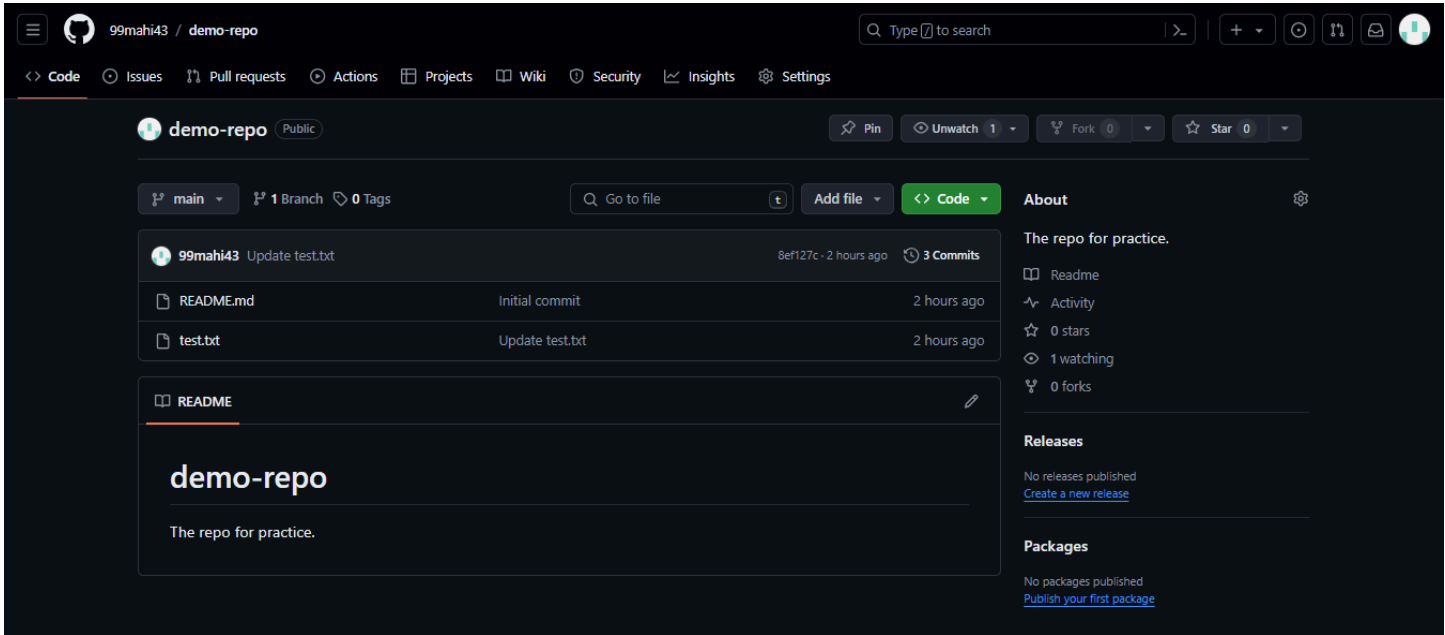
2) The repo has been cloned to local and the repo is created in the local machine.

```
[root@ip-172-31-22-236 mahi-repo]# git clone https://github.com/99mah43/demo-repo.git
Cloning into 'demo-repo'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
[root@ip-172-31-22-236 mahi-repo]# ll
total 0
drwxr-xr-x 3 root root 35 Jun 14 11:52 demo-repo
[root@ip-172-31-22-236 mahi-repo]# cd demo-repo
[root@ip-172-31-22-236 demo-repo]# ll
total 4
-rw-r--r-- 1 root root 35 Jun 14 11:52 README.md
[root@ip-172-31-22-236 demo-repo]#
```

3) Click on create a new file as shown in the below image.



4) A file named test.txt is created.



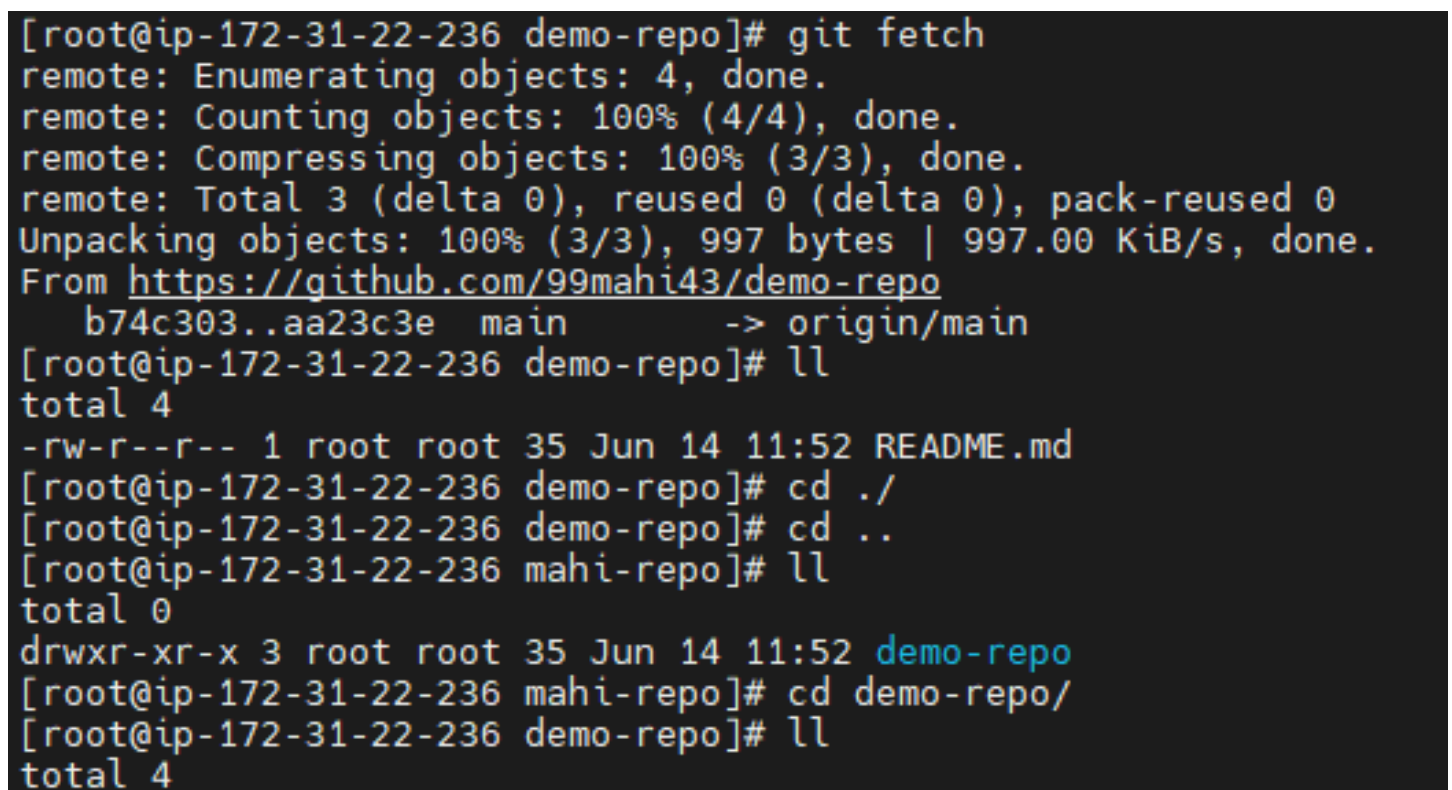
5) A few lines of data has been added.



The screenshot shows the GitHub interface for a file named `test.txt` in the repository `demo-repo`. The file was updated by user `99mah43` 2 hours ago. The file content is as follows:

```
1 This is the file used for demo practice on how git fetch and git pull works!  
2 Adding another line to check if these changes would be retrieved to the local repo.
```

6) Now if git fetch is used it retrieves the data and shows in the local with the commit id.



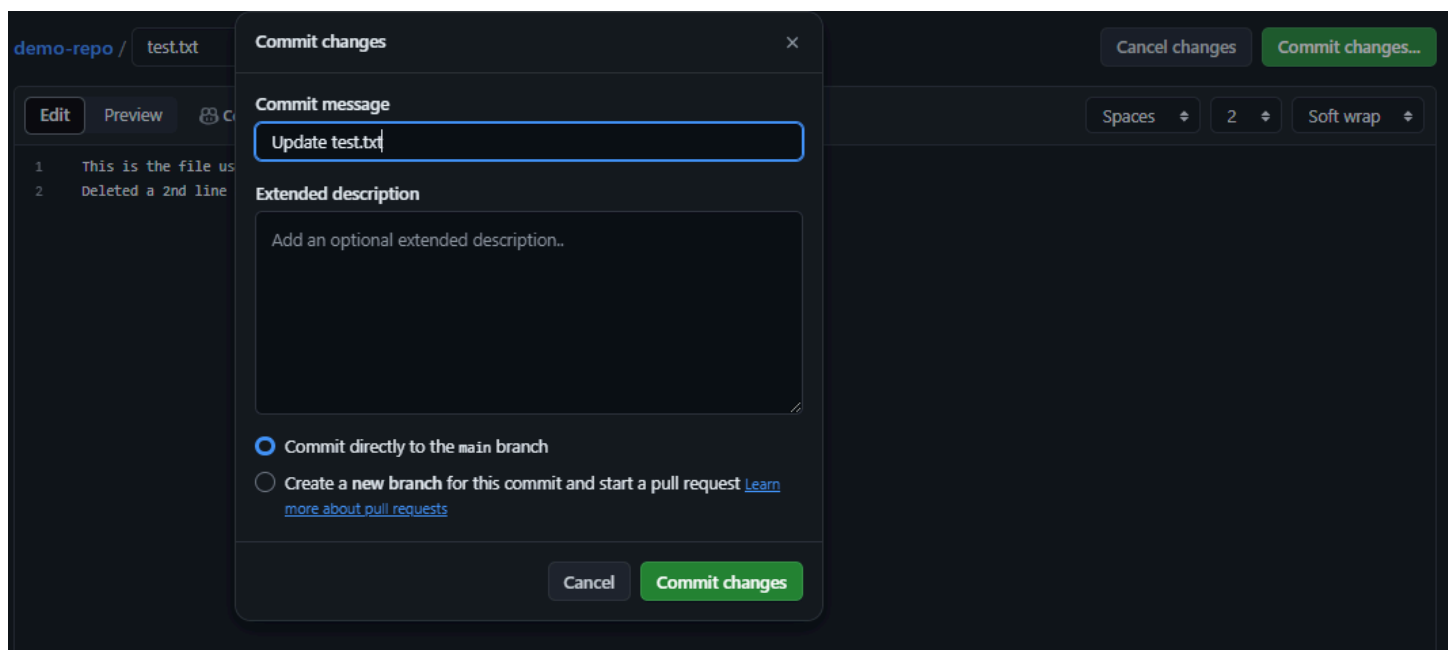
```
[root@ip-172-31-22-236 demo-repo]# git fetch  
remote: Enumerating objects: 4, done.  
remote: Counting objects: 100% (4/4), done.  
remote: Compressing objects: 100% (3/3), done.  
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), 997 bytes | 997.00 KiB/s, done.  
From https://github.com/99mah43/demo-repo  
   b74c303..aa23c3e  main      -> origin/main  
[root@ip-172-31-22-236 demo-repo]# ll  
total 4  
-rw-r--r-- 1 root root 35 Jun 14 11:52 README.md  
[root@ip-172-31-22-236 demo-repo]# cd ./  
[root@ip-172-31-22-236 demo-repo]# cd ..  
[root@ip-172-31-22-236 mahi-repo]# ll  
total 0  
drwxr-xr-x 3 root root 35 Jun 14 11:52 demo-repo  
[root@ip-172-31-22-236 mahi-repo]# cd demo-repo/  
[root@ip-172-31-22-236 demo-repo]# ll  
total 4
```

7) If git pull is used, it first fetches the data and also merges with the working directory. Hence, we also got the test file in the local machine.



```
[root@ip-172-31-22-236 demo-repo]# git pull
Updating b74c303..8ef127c
Fast-forward
 test.txt | 2 ++
 1 file changed, 2 insertions(+)
 create mode 100644 test.txt
[root@ip-172-31-22-236 demo-repo]# ll
total 8
-rw-r--r-- 1 root root 35 Jun 14 11:52 README.md
-rw-r--r-- 1 root root 161 Jun 14 12:17 test.txt
[root@ip-172-31-22-236 demo-repo]#
```

8) The previous line of data is deleted and new line is added.



9) The git fetch is used and it has retrieved the data with the commit id.

```
[root@ip-172-31-22-236 demo-repo]# git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 1.01 KiB | 1.01 MiB/s, done.
From https://github.com/99mah43/demo-repo
 8ef127c..4a974c0  main      -> origin/main
```

10) The data is merged to the working directory with the data that was changed.

```
[root@ip-172-31-22-236 demo-repo]# git pull
Updating 8ef127c..4a974c0
Fast-forward
 test.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
[root@ip-172-31-22-236 demo-repo]# cat test.txt
This is the file used for demo practice on how git fetch and git pull works!
Deleted a 2nd line and add again with different data.
[root@ip-172-31-22-236 demo-repo]#
```

11) If you use the command git show with the commit id it gives the whole information.

```
[root@ip-172-31-22-236 demo-repo]# cat test.txt
This is the file used for demo practice on how git fetch and git pull works!
Deleted a 2nd line and add again with different data.
[root@ip-172-31-22-236 demo-repo]# git show 8ef127c
commit 8ef127c27808a159425e23510cf41096f87affbd
Author: MAHENDER Jangam <146831606+99mah43@users.noreply.github.com>
Date:   Fri Jun 14 17:33:29 2024 +0530

    Update test.txt

diff --git a/test.txt b/test.txt
index ef4a270..2dffa07 100644
--- a/test.txt
+++ b/test.txt
@@ -1,2 @@
 This is the file used for demo practice on how git fetch and git pull works!
+Adding another line to check if these changes would be retrieved to the local repo.
```

12) What is the difference between git merge and git rebase?

A: git merge is something that does bring all the commits from one branch to another branch. Let's say if you're in a master branch and it has 10 commits and if you create a feature branch, checkout to it. Later if you run the command git merge branch name (master in this case) then **'the commits from master will be integrated to the feature branch and a new commit will also be created stating that this is the commit because of merge'**.

Where as git rebase just integrate the commits from one branch to another branch but **'it doesn't create the new commit after rebase'**. That's the key difference.

13) What is the difference between .git and .gitignore?

A: The .git folder is created when git is initialized in the working directory and it enables git to track changes, manage branches and it stores the history. It's like the heart of git repository that stores everything of git in it. .gitignore is very useful in situation like if there are many files like .pdf, .txt, and if

you want to track only .pdf files then you can add the remaining files in the .gitignore so that those can't be tracked that's how .gitignore file is useful.

**Ex:** Let's say there are a 10 files in pdf format and 10 files in text format. All files are already in tracked state, if you want to untrack them using .gitignore you can place either the \*.pdf or \*.txt in .gitignore in vim. However, they won't be tracked, first you need to untrack them from tracking state using `git rm --cached *.pdf` and then if you place them in .gitignore then they won't be in tracked and untracked.

Track all files except few :

`touch .gitignore` - - > or `vim .gitignore` - - > add those files which doesn't need to be committed and the remaining can be committed like `git add *.pdf`.

Except few files track from many:

go to vim editor - - > \*.pdf/txt likewise to add and track.

## 14) What is pre-commit hook and post-commit hook?

A: The pre-commit hook is something that runs before the commit is finalized like if you have your public-key and private-key or any credentials that are sensitive in your local to avoid committing them you can mention in the pre-commit hook script and in case if you still accidentally commit it the hook will avoid committing it that's how the pre-commit works and useful.

The post-commit hook is something that runs after a commit is completed. It is often used for tasks such as sending notifications, triggering builds, or updating issue trackers after a commit.

Both the pre-commit hook and the post-commit hook script resides in the **.git/hooks/** directory of your Git repository, typically named '**post-commit.sample**'. You can create or modify this script to customize the pre and post-commit behavior.

## 15) What is a webhook?

A: A webhook is like a function that is called to executed a task. Let me explain with an example instead of building a job manually all the time after the changes are committed, you can configure a webhook and whenever the changes are made and is committed it triggers the pipeline automatically to execute the job. This way a webhook is useful. You can also configure it for pull, push actions as well.

Payload URL: **URL of Jenkins till it's port number and followed by github-webhook/**  
Content type: **application/json** and then click on Add webhook.

**General**

Access

- Collaborators
- Moderation options

Code and automation

- Branches
- Tags
- Rules
- Actions
- Webhooks**
- Environments
- Codespaces
- Pages

Security

- Code security and analysis
- Deploy keys
- Secrets and variables

Integrations

- GitHub Apps
- Email notifications

### Webhooks / Add webhook

We'll send a post request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

**Payload URL \***

`http://3.89.209.132:8080/github-webhook/`

**Content type \***

`application/json`

**Secret**

**SSL verification**

By default, we verify SSL certificates when delivering payloads.

☒ Enable SSL verification ☐ Disable (not recommended)

**Which events would you like to trigger this webhook?**

☒ Just the push event.

☐ Send me everything.

☐ Let me select individual events.

☒ **Active**

We will deliver event details when this hook is triggered.

**Add webhook**

## 16) How do you pull and push changes to the git?

A: You can use git pull to pull any changes and git push for any changes to push to the git.

## 17) What is git status and git log?

A: The command git status provides the information about the tracked and untracked files in our server whereas git log is used to know the history of all the commits with author, date, commit id and time etc.

### git status:

```
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   kalki.txt
    deleted:    test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test.txt
```

### git log:

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 5c69daebe0d53885f6fb9304606c397e897e5a7c (HEAD -> feature)
Author: salaar <salaar@khansaar.com>
Date: Mon Jun 10 13:31:29 2024 +0000

    salaar file committed
```

### 18) What is the command that you used to switch to branch?

A: You can use git checkout branch name to switch to other branch and if you want to switch to a branch that's not available, you can still do it by git checkout -b branch name. This will create the new branch and will be switched to it.

```
[root@ip-172-31-22-236 mahi-repo]# git branch feature
[root@ip-172-31-22-236 mahi-repo]# git branch
feature
* master
[root@ip-172-31-22-236 mahi-repo]# git checkout feature
Switched to branch 'feature'
```

```
[root@ip-172-31-22-236 mahi-repo]# git branch
* feature
master
[root@ip-172-31-22-236 mahi-repo]# git checkout -b release
Switched to a new branch 'release'
[root@ip-172-31-22-236 mahi-repo]# git branch
feature
master
* release
```

### 19) What is cherry-pick in git?

A: If you want to copy or pick a specific commit from one branch to another branch you can use cherry-pick in git and with this you'll get the whole information of the commit with all the information without any alteration.

Also if a commit is deleted you can also get that back to the committed state. The process is first go to reflog, it has all the movements such as commits, branch references, but only just deletions. It helps to recover the lost commits and track changes.

i) Imagine there are 4 commits and a commit with webinar file name is deleted.

```

[root@ip-172-31-22-236 mahi-repo]# git log
commit 6425a3cb850d2cbda16d2e441b630e830334011e (HEAD -> master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:54 2024 +0000

    webinar file committed

commit bab9fd10b7a0b6462d633607920c0e096108ad64
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:24 2024 +0000

    assignment file committed

commit a9bf91ebfb66d494b946eacd0cceac3ff2b0e140
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:00 2024 +0000

    exam file committed

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed

```

ii) Now use the git reflog to find out the branch associated with, commit id and then we can get the commit back to the working directory.

```

[root@ip-172-31-22-236 mahi-repo]# git reflog
75a9213 (HEAD -> master) HEAD@{0}: reset: moving to HEAD~1
a9bf91e HEAD@{1}: reset: moving to HEAD~2
6425a3c HEAD@{2}: commit: webinar file committed
bab9fd1 HEAD@{3}: commit: assignment file committed
a9bf91e HEAD@{4}: commit: exam file committed
75a9213 (HEAD -> master) HEAD@{5}: commit (initial): test file committed

```

iii) Now you can use the cherry-pick command to get that commit to the committed state.

```
[root@ip-172-31-22-236 mahi-repo]# git cherry-pick 6425a3c
[master 200f666] webinar file committed
Date: Sat Jun 15 11:49:54 2024 +0000
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:
```

```
git config --global --edit
```

After doing this, you may fix the identity used for this commit with:

```
git commit --amend --reset-author
```

```
1 file changed, 1 insertion(+)
create mode 100644 webinar
```

iv) We've got the webinar file to the committed state. You can verify it with the commit-id that was earlier and now.

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 200f6665e13c7095f2e0dab8a043b535c08aadfa (HEAD -> master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:54 2024 +0000

    webinar file committed

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed
[root@ip-172-31-22-236 mahi-repo]#
```

**Cherry-pick is also used for picking up or copying a branch from one branch to another branch.**

i) Now I've created another branch with name feature and now after using the cherry-pick you can see that the exam file with the commit is at the master and feature. So here HEAD -> master, doesn't mean the direction it's specify that the same commit is in the master and release.

a) Created a new branch feature from master, a new file with name exam, tracked and committed.



```
[root@ip-172-31-22-236 mahi-repo]# git checkout -b feature
Switched to a new branch 'feature'
[root@ip-172-31-22-236 mahi-repo]# touch exam
[root@ip-172-31-22-236 mahi-repo]# cat>exam
exam file committed
[root@ip-172-31-22-236 mahi-repo]# ll
total 12
drwxr-xr-x 3 root root 35 Jun 15 11:43 demo-repo
-rw-r--r-- 1 root root 20 Jun 16 16:12 exam
-rw-r--r-- 1 root root 32 Jun 15 11:46 test
-rw-r--r-- 1 root root 29 Jun 15 12:51 webinar
[root@ip-172-31-22-236 mahi-repo]# git add exam
[root@ip-172-31-22-236 mahi-repo]# git commit -m 'exam file committed' exam
[feature 29cedbd] exam file committed
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 exam
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    demo-repo/

nothing added to commit but untracked files present (use "git add" to track)
```

b) This is a new file and committed under branch feature so this file is not under master and the same can be seen in the below image as well.

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 29cedbdfdafd186e903a3e15c684f0a91bae316d (HEAD -> feature)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sun Jun 16 16:13:50 2024 +0000

    exam file committed

commit 200f6665e13c7095f2e0dab8a043b535c08aadfa (master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:54 2024 +0000

    webinar file committed

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed
```

c) The release branch is created, checked out and if you check the git log, the file in feature is not there in release and master because the commit is made in feature only. if still there's any doubt about pic b and c, refer to branching strategy diagram.

So here, in the below images we've the branch at **HEAD - > release, master** which means that the **same commit is in master and feature**. It **doesn't specify the direction** of the cherry-pick; it just indicates where the commit currently resides.

```
[root@ip-172-31-22-236 mahi-repo]# git checkout master
Switched to branch 'master'
[root@ip-172-31-22-236 mahi-repo]# git branch release
[root@ip-172-31-22-236 mahi-repo]# git checkout release
Switched to branch 'release'
[root@ip-172-31-22-236 mahi-repo]# git log
commit 200f6665e13c7095f2e0dab8a043b535c08aadfa (HEAD -> release, master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:54 2024 +0000

    webinar file committed

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed
```

d) I am at the release branch and used cherry-pick for the commit-id that's in the feature branch (29c###16d). The cherry-pick is done.

```
[root@ip-172-31-22-236 mahi-repo]# git checkout release
Already on 'release'
[root@ip-172-31-22-236 mahi-repo]# git cherry-pick 29cedbdfdafd186e903a3e15c684f0a91bae316d
[release ffc5d9] exam file committed
Date: Sun Jun 16 16:13:50 2024 +0000
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 exam
```

e) Here the commit-id of the exam file is changed from 29c##16d to ffc###2eb after the cherry-pick.

The conclusion is you can cherry-pick a commit from one branch to another branch but also the commit gets changed after the commit is copied.

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit ffc5d976827e3b7e37a537676294913a68bd2eb (HEAD -> release)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sun Jun 16 16:13:50 2024 +0000

    exam file committed

commit 200f6665e13c7095f2e0dab8a043b535c08aadfa (master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:54 2024 +0000

    webinar file committed

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed
[root@ip-172-31-22-236 mahi-repo]#
```

## 20) How to amend a commit in git?

A: If you need to amend a commit in git, there must be some details like author name or e-mail, or the commit message has to be changed then if we change them automatically the commit id also gets changed.

**NOTE:** The amendment can only be done for the most recent commit and not for the old ones. If you look at the below images we have used the command for amendment but haven't mentioned the commit-id.

Hence the note is proved!

i) `git commit --amend -m "msg name"`

ii) `git commit --amend --author 'author-name <author-email@>'`

i (a) Before the amendment the commit message is "**webinar file committed**".

```
[root@ip-172-31-22-236 mahi-repo]# git log
commit 200f6665e13c7095f2e0dab8a043b535c08aadfa (HEAD -> feature, master)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:54 2024 +0000

    webinar file committed

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed
```

i (b) After the amendment, the commit message is "**web commit**" and the commit-id has also been changed.

```
[root@ip-172-31-22-236 mahi-repo]# git commit --amend -m 'web commit'
[feature c5b27b3] web commit
Date: Sat Jun 15 11:49:54 2024 +0000
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 webinar
[root@ip-172-31-22-236 mahi-repo]# git log
commit c5b27b348739d17f344613666297ef2006a0d216 (HEAD -> feature)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:49:54 2024 +0000

    web commit

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed
```

ii (a) The author for the first commit in the above picture is “**root**” but we can change using the `--amend` command.

**NOTE:** The amendment can only be done for the most recent commit and it won't get changed for the 2nd one. From here on, after the author is changed, whatever the commits would be done, they'd be done under the author name '**mahi**'.

After entering the command **git commit --amend --author “mahi <mahi@devops.com>”** and hit enter it gives you the below pop up in vim about the change. All you need to do is to just check, save and quit (**:wq**) and hit enter.

```
web commit

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Author:      mahi <mahi@devops.com>
# Date:        Sat Jun 15 11:49:54 2024 +0000
# Committer:   root <root@ip-172-31-22-236.ec2.internal>
#
# On branch feature
# Changes to be committed:
#       new file:   webinar
#
# Untracked files:
#       demo-repo/
#
~
```

ii (b) This is how the result looks like after saving and quitting from vim.

```
[root@ip-172-31-22-236 mahi-repo]# git commit --amend --author "mahि <mahi@devops.com>"
[feature a149ede] web commit
Author: mahi <mahi@devops.com>
Date: Sat Jun 15 11:49:54 2024 +0000
Committer: root <root@ip-172-31-22-236.ec2.internal>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 webinar
```

ii (c) The final result of the amendment of the author of the most recent commit can be seen below and the author is changed from root to mahi and the commit-id has also been changed.



```
[root@ip-172-31-22-236 mahi-repo]# git log
commit a149ede4ccb169f613d32142d3c3acb43d666528 (HEAD -> feature)
Author: mahi <mahi@devops.com>
Date: Sat Jun 15 11:49:54 2024 +0000

    web commit

commit 75a92136abedaa2a450da16988cb0e15d3f55a04
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Sat Jun 15 11:48:38 2024 +0000

    test file committed
```

## 21) What are merge conflicts and how do you resolve merge conflict in git?

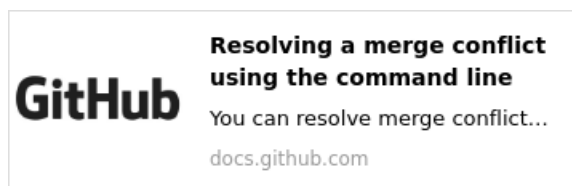
A: The merge conflict arise when competing changes are made to the same line of the code. In other words, if two persons have made changes to the same line of the code or if one person edits the file and other person deletes the same file.

To resolve the merge conflicts:

1. You can open the file in the vim editor or vscode and remove the conflict markers from the file and confirm which line of code needs to be intact and to be removed.
2. Then save the file.
3. Now add git (git add filename) and commit the changes in the file (git commit -m 'msg name' filename not required).

If you don't want the merge conflicts to happen, you can use the command **"git merge --abort"**.

Refer to below documentation for a better understanding with representation:





## 22) What is a branch in git?

A: A branch in git is used for isolating the source code by naming them as master, feature and release etc and we can work on the source code for future requirements or with the release code without disturbing or altering the main source code in the master branch.

## 23) What is git stash?

A: If there's a situation to just save the data in the current working copy without committing then we use git stash.

**\* If you want to temporarily delete the data or store the data (.git refs/stash) which is in the staging area, we use stash.**

**Ex:** Let's say there are two branches master and feature, in master we have 3 commits and 3 files and we created feature branch where we get all the data from master. In feature branch, if we write a few lines of code (LOC) again and until and unless we commit we can't go to the other branch (master branch). In order to go to the master without committing we can use stash this will help in temporarily delete the data and get back to feature to remove the stash to get the previously written LOC back again.

### **Stash commands:**

To store/delete the files temporarily - - > **git stash**

To get those files back to work upon/to commit - - > **git stash apply/ git stash pop**

To clear the stash/ to clear the files that were stashed - - > **git stash clear**

To check the list of stash - - > **git stash list**

To delete a specific stash - - > **git drop stash-name**

The files that are stashed/temporarily deleted are stored in the project's **.git directory; refs/stash**.

## 24) What is git diff?

A: git diff command is used to get the difference between two files or two commits within your git repository.

## 25) What is git blame?

A: git blame is used to know the changes that were made line-by-line and it also shows who the author is and what the modification is in github.

## 26) What is git tag?

A: In git, tag is useful to filter out a specific commit when needed, typically during the release if we want to check where the mistake is or might be to check the author or any information then git tag is very useful.

## 27) What is cloning in git?

A: Cloning in git means that you use the command called as 'git clone' to get the repository from the central to the local system (typically from GitHub to local machine). In other words it can also be defined as **"to pull the source code from the GitHub"**. Here through git clone you don't create your own copy of the repository as git fork does from the GitHub but, you just get/pull it to local machine.

## 28) How do you untrack a file or how do you move a file from staging area to working directory?

A: In working directory we've all the untracked files and in staging area we've the tracked files. So if we need to move a file from staging area to working directory or if we want to untrack a tracked file, then we need to use the command called **"git rm --cached filename"**.

There are no untracked files before using the command and the branch is empty.

```
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
nothing to commit, working tree clean
```

Please refer to the below images in order to move the test.txt file from staging area (tracked state file) to working directory (untracked state).

1) I've checked what is the commit-id associated to test.txt file with the help of the below command.

```
[root@ip-172-31-22-236 mahi-repo]# git show 786ea088cecfcc6ac2cfb30a411eca08684f780c --name-only
commit 786ea088cecfcc6ac2cfb30a411eca08684f780c (HEAD -> feature)
Author: root <root@ip-172-31-22-236.ec2.internal>
Date: Mon Jun 10 09:28:24 2024 +0000

    3rd commit

test.txt
```

2) Now use the command to untrack the file.

```
[root@ip-172-31-22-236 mahi-repo]# git rm --cached test.txt
rm 'test.txt'
[root@ip-172-31-22-236 mahi-repo]# git status
On branch feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        deleted:    test.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        test.txt
```

**NOTE:** If a file is moved from staging area to working directory or in other words if a file is untracked after being tracked and committed, **it will still be associated with the commit-id, will remain in the commit history and will not be tracked in future commits until and unless it is tracked again by git with the help of 'git add'.**

## 29) How do you delete the last remaining commit in the master branch?

A: The command "**git update-ref -d HEAD**" has its impact in two ways.

i) If there's only one commit this command will help to remove the last commit and will remain the branch empty.

ii) If there are multiple commits, this will only remove the most recent commit.

This process removes the most recent commit but keeps the rest of the branch history intact.

## 30) How do you completely reset a branch?

A: All you need to do is to create an orphan branch **git checkout --orphan temp-branch**, then remove all the files **git rm -rf .**, commit the empty state using **git commit --allow-empty -m "Initial empty state commit"**, force update the master branch using **git branch -M master temp-branch** and **git push --force origin master**.

**git branch -M master temp-branch** : This command renames the current branch (temp-branch) to master. The -M flag forces the rename, which will overwrite the existing master branch if it exists.

**git push --force origin master** : This command force-pushes the local master branch to the remote repository (origin), overwriting the remote master branch with the local master branch's state. The --force flag is necessary because this operation changes the history of the branch, which is normally not allowed without explicit confirmation.

