

GIT-LAB-

Git-version: -

Is a tool that tracks changes to a file over time so that you can recall any specific version (you can point to any specific version)

Commit: reference point to a snapshot

For Window

You can install git on your local laptop: <https://git-scm.com/download/win>
First install git. 2.29.2

Then create one folder and open it

And then inside just right click and open git bash
And then use below commands'

```
=====
=====
and EC2 instance.
```

To install on EC2, Follow below steps.

Create EC2 instance

Login ec2 instance

```
sudo su -
sudo yum install git -y          # install git sudo package
git version # to check the version
mkdir git-demo1                  # creating new directory
cd git-demo1                     #change directory
```

git init

(This command turns a directory into an empty Git repository. This is the first step in creating a repository. After running git init , adding and committing files/directories is possible).

```
ls -a                            # list all the file
vi index1.html # new file creation, and save and exit.
then get into insert mode press "i" then type
esc then shift: wq
```

```
git status #to check the current status, file will be in working area.
# we will get red color which indicates that files are in
# working area
```

```
git add index1.html              # moving files from working to staging area
```

(Adds files into the staging area for Git. Before a file is available to commit to a repository, the file needs to be added to the Git index (staging area). There are a few different ways to use git add, by adding entire directories, specific files, or all unstaged files.)

git status # tracked but uncommitted.

(This command returns the current state of the repository. Git status will return the current working branch. If a file is in the staging area, but not committed, it shows with git status. Or, if there are no changes it'll return nothing to commit, working directory clean.)

{we can add multiple file:-

```
git add index1.html index2.html index3.html
```

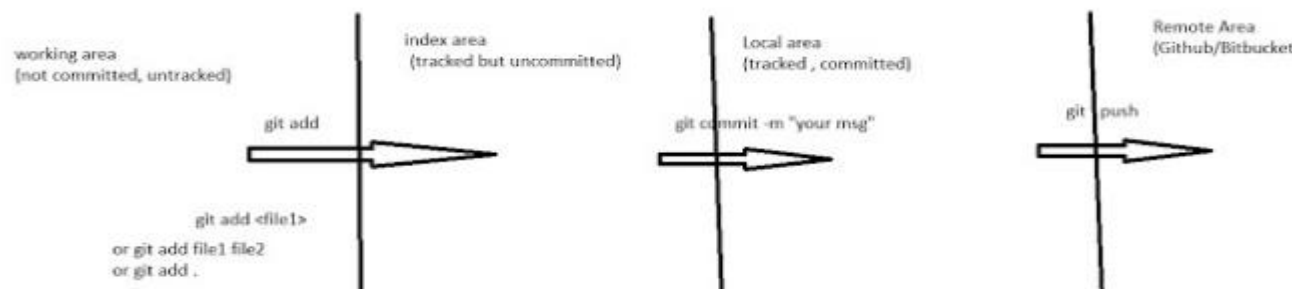
we can also add all the file:-

```
git add .  
}
```

git commit -m "index1.html file created" # committing files.

(Git Commit -Record the changes made to the files to a local repository. For easy reference, each commit has a unique ID. It's best practice to include a message with each commit explaining the changes made in a commit. Adding a commit message helps to find a particular change or understanding the changes.)

git status # you will nothing. because all the files are in committed mode.



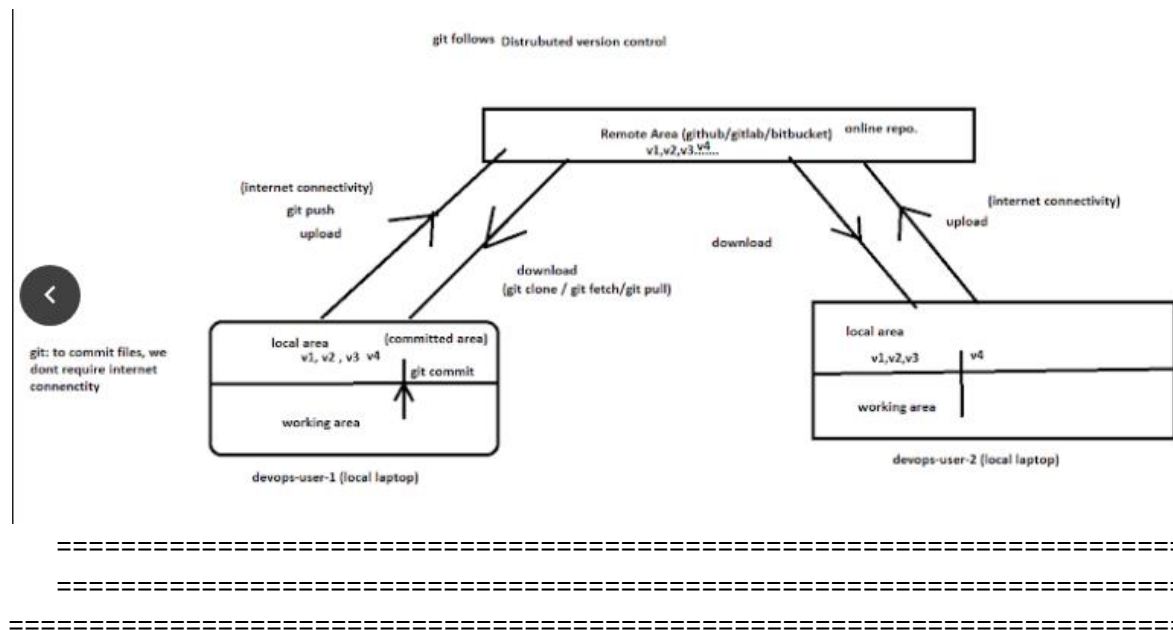
o git log # to check all the git logs

- To show the chronological commit history for a repository. This helps give context and history for a repository. gitlog is available immediately on a recently cloned repository to see history.

- o git config --global user.email "prakashk0301@gmail.com" # configured user email
- o git config --global user.name "prakash" # configure user name
 - vi index1.html >>>. this is prakash, update1
- o git add index1.html
- o git commit -m "udpated file index1.html" # generate new commit ID
- o git log # this time it displays your name n email.
- o git show <commit Id> # display the chnages onthat commit id
- o git log --online # display all the commit id in shorter format

```
git add  
git commit  
git status  
git config  
git log --online
```

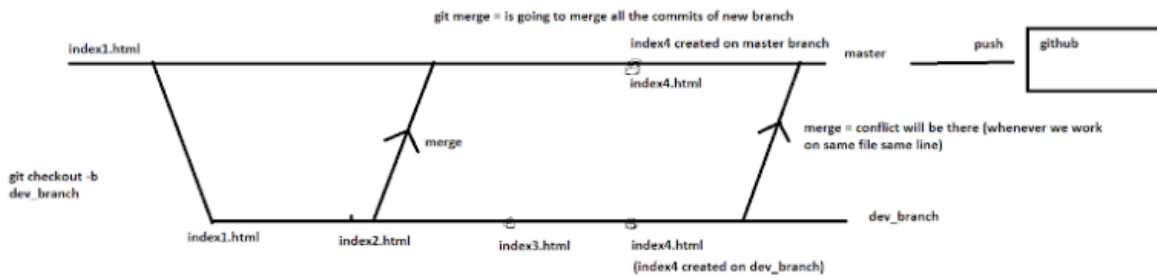
git show



So when you have to merge the feature branch ,first u need to switch on to dev branch and then execute the above command ,then whatever content in feature branch will reflect in dev branch.

when we are working same file & same line, will get conflict

- Now create one file in master branch index 4.
- and two files in sub branch with the index3 and index4
- now git merge
- now two files with the same name
- so will get the conflicts.
- If contents are different you will get conflict
- Will get conflict in index4.
- Now open index 4
- Cat index 4
- Vi index 4
- Un-necessary lines are there which is not part of code. Just remove it
- And save
- Conflict is due to mismatch of data.
- Ideally we should not with the same file name,
- Now git add
- And git commit
- Cat index 4
- Now you can see the updated one. now u can have updated one
- Now checkout to the dev
- Now edit the file
- And same as you did in master
- In master we have 1line
- Line 1 same but second line in 2
- Now merger



Git merge

Now again getting conflict

Now vi open the file and make changes and save

Means we have manually merge it.

We can use **diff checker** to check the difference.

We have git diff command

Then use git add and commit as it is required whenever you edit the file.

git branch -a

----> display all the branches (including hidden branches)

Git Pull (downloads only changes not the entire code)

pull only changes from remote to local (file level + content level).

Lab

Create new file in repository.

And then commit the file.

If I check my master branch locally I wont find that file

Then execute pull commands.

Then u can see that file here

File level and content level changes you can see.

git pull = git fetch + git merge

git fetch: git fetch is similar to git pull, it pulls all changes on a single branch -: remotes/origin/master (not on master)

git branch -a

git checkout remotes/origin/master (validate all code, if changes are correct you can consider merging changes with master branch manually.

git fetch is safer than git pull)

git checkout master

git merge remotes/origin/master

gitrm

Remove files or directories from the working index (staging area). With gitrm , there are two options to keep in mind: force and cached. Running the command with force deletes the file. The cached command removes the file from the working index. When removing an entire directory, a recursive command is necessary.

Usage:

To remove a file from the working index (cached):

\$ gitrm--cached <file name>

```
# To delete a file (force):
$ gitrm-f<file name>
# To remove an entire directory from the working index (cached):
$ gitrm-r --cached <directory name>
# To delete an entire directory (force):
$ gitrm-r -f <file name>.
```

=====

Q. How to delete git branch

git branch -d <branch name>

=====

```
vi index1.html
git add .
git commit -m "file created"
git log --online
```

git branch: to check the current branch, * "star" points to the current branch
(* dev_branch it means your current branch is dev_branch)

Q. How to ceate new branch ?

syntax: **git checkout -b <branch_name>**

ex: git checkout -b dev_branch

git status --> files should be in committed status
git branch --> files should be in committed status

```
vi index2.html
```

Q. How to switch between branch

git checkout <branch name>

Q. How to take changes of another branch?

```
git merge <branch name>
git merge dev_branch
ls
git checkout dev_branch
git diff
https://git-scm.com/docs/git-mergetool --> tool to fix conflict
Once you are okay with changes you can push code to the github/gitlab/bitbucket
```

=====

Assignments:

1. push code to gitlab : `git push origin1 <gitlab url>`
2. push code to bitbucket: `git push origin2 <bitbucket url>`
3. how to push code from all the local branches? `git push --all`
4. Gitlab role: <https://docs.gitlab.com/ee/user/permissions.html>

Once your account is ready, create a public repo.

===== **Git Remote:-**

To connect a local repository with a remote repository. A remote repository can have a name set to avoid having to remember the URL of the repository

```
git remote add origin https://github.com/<repo ID>/may-devops-batch.git
```

```
git push origin master
```

we are going to add remote repo as alias "origin"

```
(git remote add prakash https://github.com/<repo Id>/may-devops-batch.git)
```

```
git remote add origin git@gitlab.com:<your repo id>/maypdevops.git  
git push -u origin master
```

Bitbucket: - <https://bitbucket.org/>

===== **Git stash:**

- When you want to record the current state (not ready feature) of your working/index directory but want to work on some other issues (hide files from working & index area for some time, so that you can work on on-going issue). This command moves all file (working/index) to stash area (hidden) for some time. This command is for uncommitted files.

This command moves your current working area content (files) to stash area.

It allows us to work on any other file. Once you done with changes you can move stash data to the current working area.

```
vi abc.txt          --new feature  
git add .  
git status          ----> there is a issue in prod, ur manager ask u to work on prodiction_fix.txt  
git stash
```

```
ls  
git status
```

```
vi prodiction_fix.txt      ----> make sure we have clean working area, only then  
we can work on prod issue  
git add .  
git commit -m "prodiction_fix.txt"  
git push origin master
```

now you can move stash data to working area.

```
git stash list          ----> display reference list
```

- git stash show ----> display stash files
- git stash pop ----> move files from stash to working area

Git push -u origin master --force

----> push master branch changes forcefully

Keep same file content in local and make changes in the same file in remote repository.
Then try to push
Then you will get as it is
Whatever content you have locally we can push.

```
git push -u origin <any branch name>      ----> push changes of <any branch>
git push -u origin <any branch name> --force  ----> push changes forcefully of <any branch>,
very dang
```

we are going to rewrite history.

git remote -v ----> to check remote alias

```
git remote rm <alias name>          ----> remove alias
```

```
git remove rm origin
```

git remote add <alias><repo url> ----> you can add any alias

git pull: it downloads the changes from remote repo to local. (it downloads the changes of entire branch, it works at branch level not the file level)

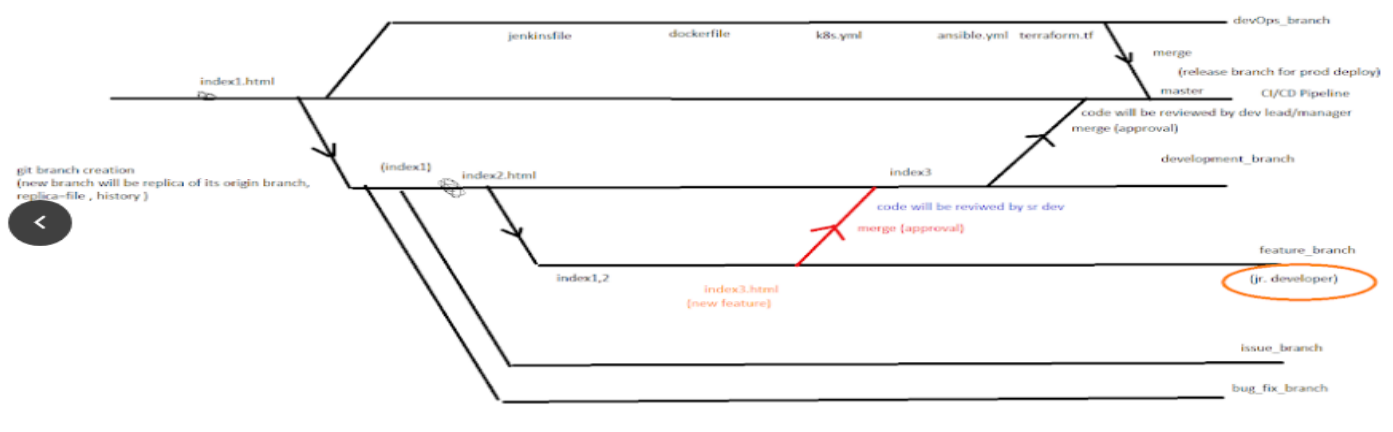
git pull is combination of git fetch and git merge

git pull = git fetch + git merge

```
git remote -v
```

```
git pull origin master.
```

10 kb



Git Clone:

git clone<URL> ----> download code from remote to local repo (clone master branch by default)
For that in repository go to code section and copy URL and execute the command.

How to clone any specific branch.

I don't want to clone master branch then we have one feature below.

But url you copy from repository will remain same.

`git clone --branch <branch name><remote-repo-url>`

(git clone --branch devbranch <url from repository>

Now we can see that particular branch there.

Clone command create new folder.

Ideally we should download the master branch.

To create a local working copy of an existing remote repository, use git clone to copy and download the repository to a computer. Cloning is the equivalent of gitinit when working with a remote repository. Git will create a directory locally with all files and repository history.

The git clone command downloads (it downloads the entire code) an existing git repo to your local laptop.

From ec2 also possible to download to local laptop.

`git clone <github URL>`

ex: `git clone https://github.com/prakashk0301/may-devops-batch.git`

In clone-code comes in local repository.

In pull-Only updates will come in working directory.

To get the latest version of a repository run gitpull. This pulls the changes from the remote repository to the local computer.

Once you have code update it and push to the github..

`git add .`

`git commit -m "chanegs"`

`git push origin <branch name>`

=====

Q. diff b/w git clone and git pull?

Git Pull: it downloads only changes from remote to local repo (condition: you should have that code, if there is any changes in remote repo then using git clone u can download changes.)

Git Clone: it downloads entire code from remote to local laptop (if you dont have code then execute git clone command)

First thing in the morning just execute the git pull commands to that u will get the latest changes.

Git push -all

Gitpush

Sends local commits to the remote repository. gitpush requires two parameters: the remote repository and the branch that the push is for.

Usage:

```
$ gitpush<remote_URL/remote_name><branch>
```

```
# Push all local branches to remote repository
```

```
$ gitpush --all
```

```
=====
```

Git fetch command: Git fetch command is used to pull the updates from remote-tracking branches to local fetched branches.

We can verify the remote changes, once we are okay with changes we can merge with the current branch.

(update remote repo by adding new line at github side)

```
git fetch                      ----> download changes
git branch -a
```

----> list all the hidden branches

```
git checkout remotes/origin/HEAD      ----> switching branch
verify the changes, if u are okay then you can merge with the current branch (master)
```

```
git status
```

```
git checkout master
git merge remotes/origin/HEAD
```

```
-----
```

Q. How to merge any specific commits? How to pick changes from any specific commit IDs?

Ans: **git cherry-pick**<commit ID of ready code>

git cherry-pick: it allows us to take changes of any specific files. (in other words it allows us to merge only specific files)

```
mkdir demo1234
cd demo1234
git init
vi index1.html
git add .
git commit -m "index1"
git checkout -b dev
vi index2.html
git add .
git commit -m "index2"
vi index3.html
git add .
git commit -m "index3"
git log --online
git checkout master
git cherry-pick <commit id of index2>
```

ls

Rebase-No commit id
Merge-Commit id
To push entire directory
Git add project/*

git clone vs pull:

Git pull means you are fetching the last modified repository. **Git push** means you are returning the repository after modifying it. In Layman's term **Git clone** is downloading and **Git pull** is refreshing. **clone**: copying the remote server repository to your local machine.

Git Revert

is used to **undo a previous commit**. In git, you can't alter or erase an earlier commit. (Actually, you can, but it can cause problems.) So instead of editing the earlier commit, revert introduces a new commit that reverses an earlier one.
Revert changes.

Create new file at git bash and commit.
And push these code to remote.
But won't be pushed if our local is not updated then will have to update it first by git pull.
So execute git pull commands... download all changes.
Now use git push command
Now remote repository is up to date.
Once u upload the file on git hub... CI/CD will be triggered.
But prod is not stable then will have to revert the changes.
So that file will be in previous step.
Use git log
Now execute
Git revert and commit id
And save
Git ls
you won't find that file
Create a revert-delete.
Again push git origin master
So that recent changes will be updated.
Again revert to revert
Then will get deleted file.
Then u will get one more revert id.

Once u revert it will create once more revert id as it is version control.

EX: create a new directory, create a new file and **commit** it.

Now if you don't want those changes then use revert command

git revert <commit ID>

Mkdir
Git init
Vi 1
Git add
Git commit
Vi 1
Add some changes
Git add
Git commit
Git log
Take first **five digit** of commit id
Git revert commit id

Git Reset

- We have so many changes from last one month and my code not working.
- So how we can go back to previous state
- And if we have multiple commit ids. Here we have git reset.
- We have 3 modes you may lose whole data and you can go back to a specific commit id. you can point head to any specific commit id. you may lose whole data.
- In git log-head is pointing to master and corresponding commit id..git will delete
- If I ask head to point to other commit id then it will take that changes **.No commit id will be created.**
- Hard mode-
- Git reset --hard HEAD~2
- Now in git log
- We have two commit ids
- Other commit id gone
- We can't execute git revert as we have two commit id.

git reset : is very dangerous, because it overrides the git commits. you may lose whole data (if you ask your HEAD to point to any specific, lets one month old commit ID, git will delete other files). You can go back to a specific commit ID (You can point your HEAD to any specific commit ID),

git reset HEAD~<Position of commits>

git reset HEAD~0 (latest commit)
HEAD~1 (2nd latest commit)
HEAD~2 (3rd latest commit)
HEAD~3 (4th latest commit)

or

git reset --hard <commit Id> ---> you can point your HEAD to any specific commit ID

hard mode: - to clean working and staging area (it will delete all the committed file from staging/index area)
dangerous.

```
touch abcde.txt
git add .
git status
git reset --hard
ls
git status
```

Mixed mode (mixed is a default mode of git reset): - it also cleans staging, but do not delete any file ,
file will be moved from staging area to working area

```
touch abc.txt
git add .
git status
git reset --mixed or git reset
ls
git status
```

Green to red.

Soft mode: - it doesnt clean directory (useless)

- We don't use hard one. If not sure do not use git hard command on master branch.
 - We require git to clone code from github to jenkins.
 - We require git tool on jenkins server so that we can clone the code.
 - We require git on dev, developer and jenkins server.
-
-

When we join in any team/ organization .

Before you use git push(as it always ask for the password hence to overcome this
We need to generate token)

```
open github
git bash
ssh-keygen ----> to geneate ssh key for perticular system or laptop
```

copy id_rsa.pub key

github/gitlab/bitbucket account-> setting-> ssh and GPG key -> add your ssh public Id

git push origin master
now u can use ssh url now.
HTTP-for user and password

Q. Diff b/w git reset and revert?

git revert: revert the changes of any specific commit id. it maintains git history.

git reset: we can point out HEAD to any specific commit ID. its dang bcas it rewrites the git history.

Git Rebase

- With the `rebase` command, you can take all the changes that were committed on one branch and replay them on another one.
- Ex: you can take the patch of the change that was introduced in `b1` and reapply it on top of `b2`. In Git, this is called *rebasing*
- is similar to git merge, (when your project is linear, or you are working alone)
- Do not use git rebase whenever you are working in a team, because it may rewrites the git history.
- Directly I created feature branch here.
- Will take the code from directly feature branch
- Git rebase-can also be used to take code from one branch to another
- Here I have not created any development branch... we are working alone or very limited people. when we work in big team then will require approval

Demo:-

Create file

Touch 1.html

Add and commit

Git log --oneline

Ls

Git checkout -b feature branch

Now create one more file on feature branch

Git add and commit

Git checkout master

Git rebase and branch name

Straight forward update

Rebase is for linear, stream line, faster. If we have code in gb... time consuming will be less compare to merge.

Merge for non linear and complex... here we have approval and all those things.

We need to maintain history but no approval is required.

if project is non linear (ex: teams)

we always prefer merge over rebase: because git history can managed very easily, also we can assign merge request to the reviewer.

```
mkdir efg
cd efg
vi index1.html
git add.
git commit -m "index1 on master"

git checkout dev_branch
vi index2.html
git add .
git commit -m "index2 created on dev_branch"
```

.....you can take the changes from dev_branch to the master branch

```
git chekcout master
git rebase dev_branch
```

Git Merge vs Rebase

The major benefit of rebasing is that **you get a much cleaner project history**.

First, it eliminates the unnecessary merge commits required by git merge.

Second, as you can see, rebasing also results in a perfectly linear project history

Git Fetch vs Pull:

The git fetch command downloads commits, files, and refs from a remote repository into your local repo. Fetching is what you do when you want to see what everybody else has been working on. It lets you see how the central history has progressed, but it doesn't force you to actually merge the changes into your repository. Git isolates fetched content as a from existing local content, it has absolutely no effect on your local development work. Fetched content has to be explicitly checked out using the git checkout command. This makes fetching a safe way to review commits before integrating them with your local repository.

Step 1. Create a folder-create a file then commit it.

Step 2. Create a remote repo

Step 3. At local

git remote add origin <repo url>

git branch -a

Step 4. Then push to remote repo

git push origin master

Step 5. Create a file at remote repo.

Step 6. Use git fetch to fetch updates

git fetch origin master ->(Still you can't see updates, file which we created at remote repo)

git branch -a -->to list all branches

(You can see updates are not merged with our master branch, updates are available only at remote branch)

git log origin/master ---->(You can see commits of remote branch)

git merge origin/master

git clone <github URL>

ex:<https://github.com/prakashk0301/may-devops-batch.git>