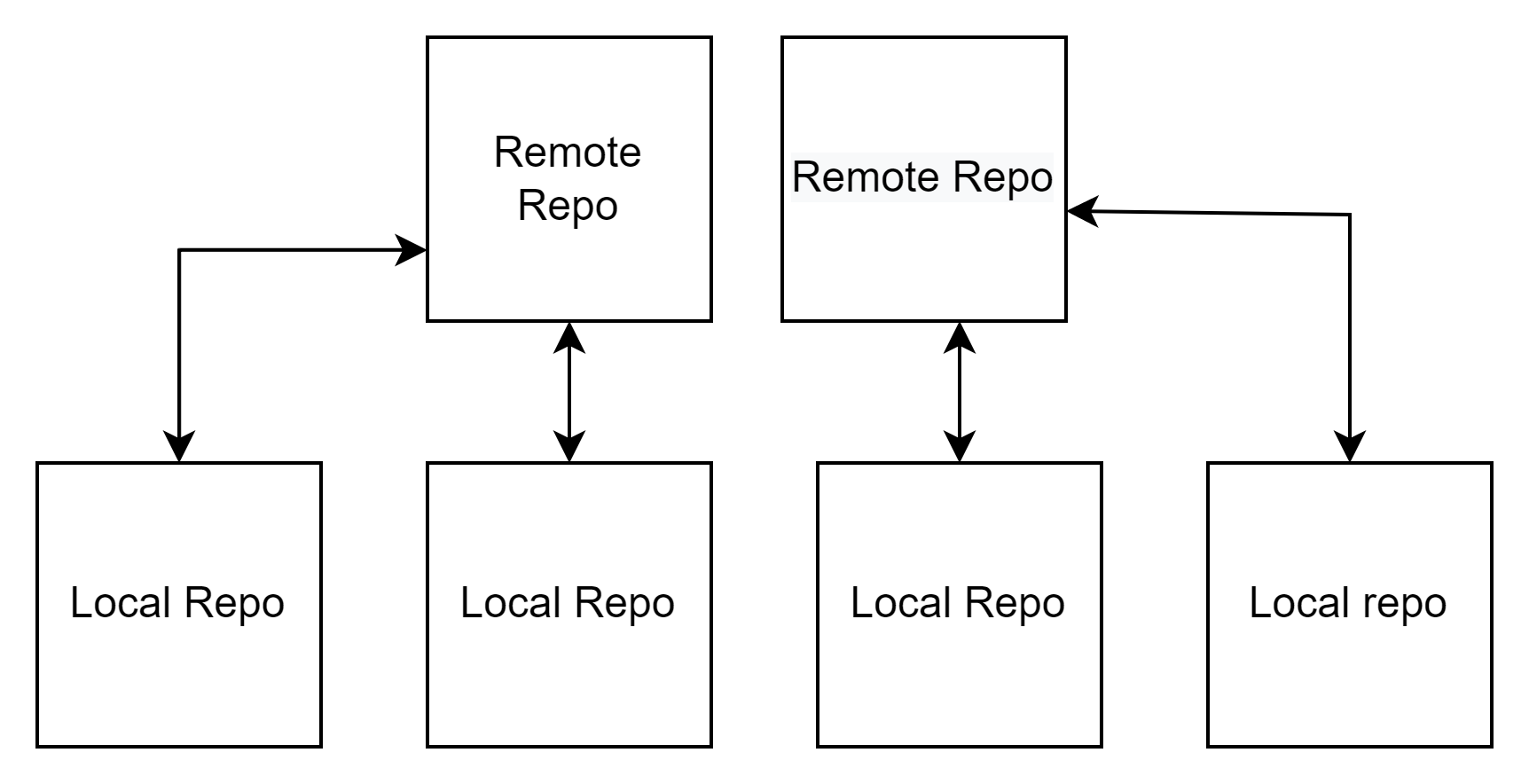
**Git**

Git is a Distributed Version Control system and we can called it as a Blind file tracker.



**Github:**

Git hub is manage your Git repositories, review code like a pro, track bugs and features, power your CI/CD and DevOps workflows, and secure code before you commit it.

Git is having two types of repositories

1. Local Repo
2. Remote Repository

**Local Repo:**

This is the individual repo which placed on your workstation (either laptop or might be jump box)

Local repos are having three areas

1. Working area
2. Staging area
3. Commit

**Working Area:** The place where new changes will be done

**Staging Area:** New changes moved from working area to staging to make commits. So this is the place where the commit will happen soon.

**Commit:**  Committed files with version (commit IDs)

**Remote Repo:**

Centralised location where multiple local repos will be merged and created as a central copy to give access for different operations like CI/CD etc.

To install git

LINUX:

Yum install git -y

Ubuntu:

Apt install git -y

In Windows you can download gitbash and install it as a shell prompt in linux.

To check git version:

#git —version

To initialise local repo :

1. Choose a directory to create your local repo, if its not present create it with complete path

#mkdir mygitrepo

#cd mygitrepo

1. Initialise the local repo using belo command

#git init

1. Check mygitrepo directory and you should found a new hidden directory .git should add inside.

$ mkdir mygitrepo

$ cd mygitrepo/

$ ls -la

total 12

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 ./

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 ../

$ git init

Initialized empty Git repository in C:/Users/Lakshmi/mygitrepo/.git/

$ ls -la

total 16

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 ./

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 ../

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 .git/

1. Observe the directory hierarchy of .git folder

$ cd .git

$ ls -la

total 11

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 ./

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 ../

-rw-r--r-- 1 Lakshmi 197121 23 Feb 14 21:46 HEAD

-rw-r--r-- 1 Lakshmi 197121 130 Feb 14 21:46 config

-rw-r--r-- 1 Lakshmi 197121 73 Feb 14 21:46 description

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 hooks/

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 info/

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 objects/

drwxr-xr-x 1 Lakshmi 197121 0 Feb 14 21:46 refs/

Observe all sub commands which we can do using git.

$ git --help

usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | -P | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

[--super-prefix=<path>] [--config-env=<name>=<envvar>]

<command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

restore Restore working tree files

rm Remove files from the working tree and from the index

sparse-checkout Initialize and modify the sparse-checkout

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

diff Show changes between commits, commit and working tree, etc

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

commit Record changes to the repository

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

reset Reset current HEAD to the specified state

switch Switch branches

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

See 'git help git' for an overview of the system.

Move files from working area to Staging which are untracked/modifed:

#git add <file name>

To move all files which are untracked/modified from working area to staging

#git add .

Configure your git with user name and mail ID to track who did the commits.

#git config user.name "name"

#git config user.email "email"

Commit the files which are present in the staging area

#git commit -c “comment”

**GIT log:**

Git log is the command which will show you the commit information.

Some example git log commands

git log

git log --name-only -> which shows you the file which has modified with respect to commit

git log -n 1 -> which shows you the last 1 commit in place of 1 u can give number to see those many recent commits

git log -n 2

git log -n 3

git log -n 3 --name-only

git log --graph --decorate —> will display the graph format

prasad (master)$ git log --graph --decorate feature/signout

\* commit 9fc23a136910ab70d50b8b4b1627e4fe31de6529 (myfetu/signout)

| Author: prasad <prasad@example.com>

| Date: Tue Feb 15 01:35:47 2022 +0000

|

| Add signout page

|

\* commit a3f7e9a9a345c8d5c5b822b437fe6adaeeaed752 (myfetu/signup)

| Author: prasad <prasad@example.com>

| Date: Tue Feb 15 01:35:47 2022 +0000

|

| Add signup page

|

\* commit 6522c1cba1df5d82b77385153401101fda7ea3da (HEAD -> master)

Author: prasad <prasad@example.com>

Date: Tue Feb 15 01:35:47 2022 +0000

Added main page

git log -n 1 --oneline → will display the one liner information

git log -n 1 --oneline --name-only

git log -n 2 --oneline --name-only

git log -help

git log test2 → display the commits on file

**Branch:**

* Branch is a virtual folder which isolates our tasks.
* It is very hard to maintain all tasks directly in single branch
* Typically every feature/enhancement/bug should be implemented in a separate branch.
* In git “branch” is lightweight, it is just a pointer.

Git Branch related commands

1. git branch (display local branches)

2. git branch -a (display local and remote branches)

3. git branch <branch-name> (creates new branch)

4. git checkout <branch-name> (Switch branch)

5. git branch -d <branch name>

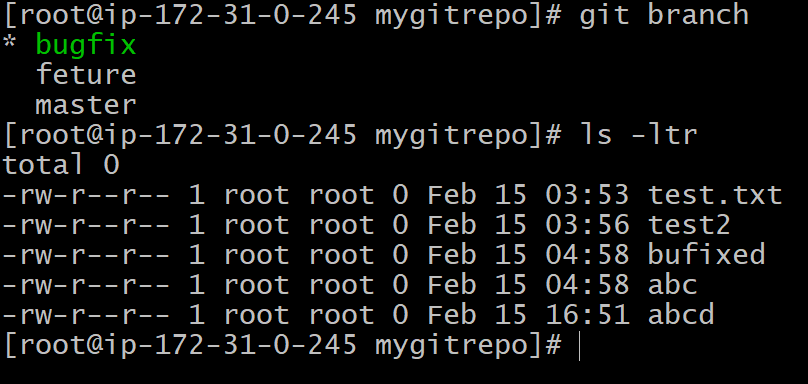
**Merge the branches:**

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

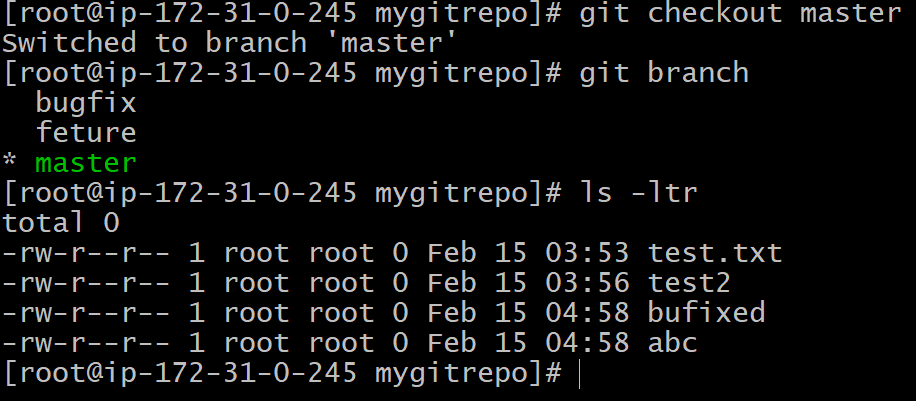
The current branch will be updated to reflect the merge, but the target branch will be completely unaffected

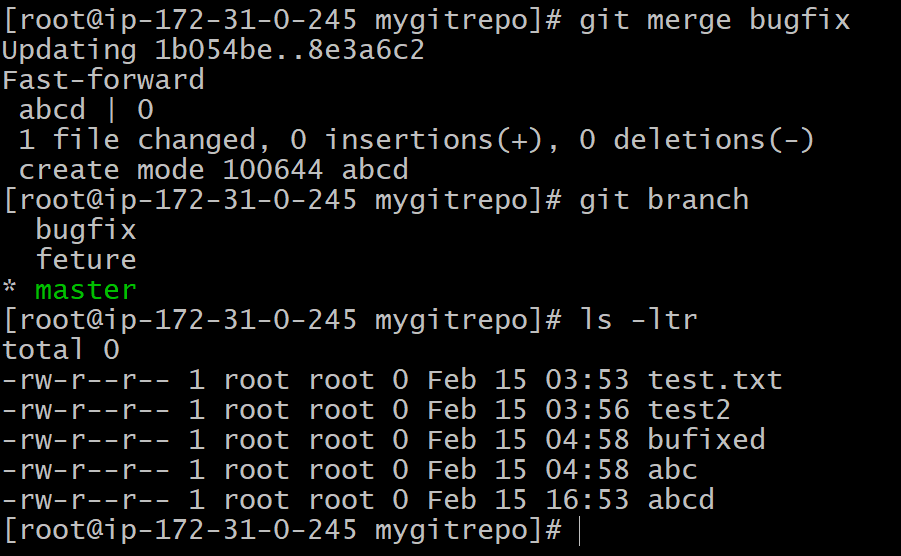
1. Check out the target branch in which you want to merge with current branch example target is master and source is bugfix

#git branch #to see the current branch



#git checkout master



1. Merge the bugfix with master
2. #git merge bugfix

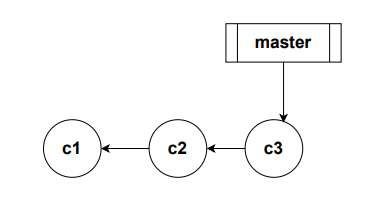
Once your merged bugfix with master, the master got updated with all changes which are done on bugfix (ex: abcd file has listed in master) but bugfix branch will not having any effect after merge (same has mentioned in merge definition as well)

# **fast-forward merge in Git**

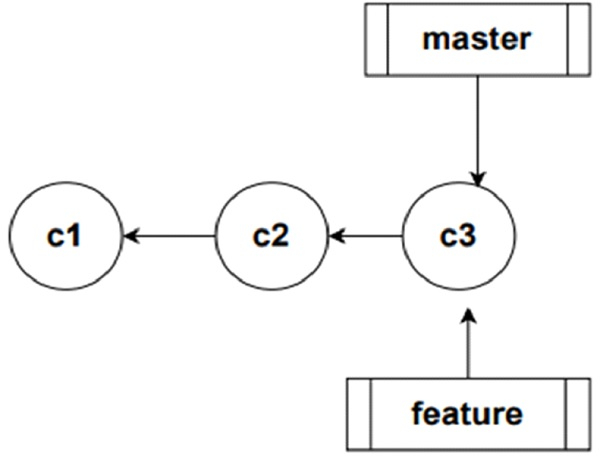
Fast forward merge can be performed when there is a direct linear path from the source branch to the target branch. In fast-forward merge, git simply moves the source branch pointer to the target branch pointer without creating an extra merge commit.

Example:-

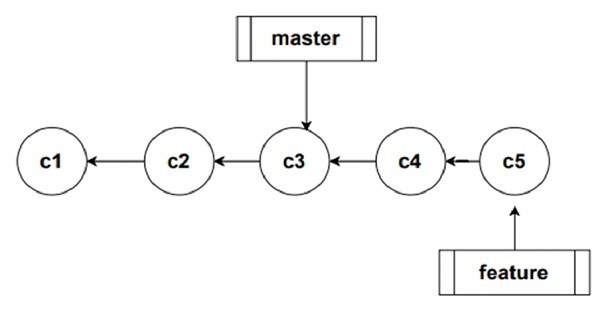
We have a master branch with 3 commits.



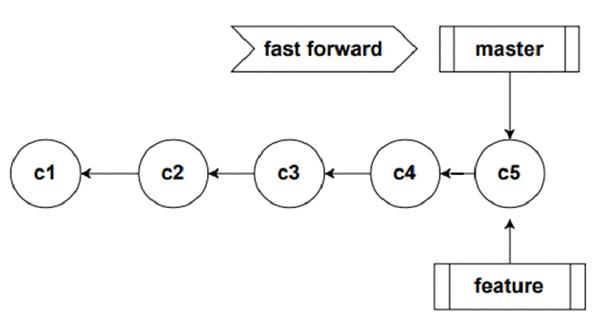
Next, we create a branch called feature branch. In git a branch is nothing but a pointer to a commit. At this point both feature and master are pointing to the same commit.



Now let us switch to the feature branch and do a couple of commits. Now we need to bring the changes to the master branch. There is a linear path from feature to master.



In order to merge the changes to the master branch, all git has to do is to change the pointer of master forward. This is what we call fast-forward merge.



Git Branch policies

1. No one should directly push changes to main
2. We want to allow pull request to merge only if it is approved

Git branching strategies

1. Branching strategy is very important to keep our process seamless
2. Choosing a best strategy is important.

1. Feature branch
2. It is short lived branch that contains specific feature
3. Feature branch naming conventions
   * 1. feature/hari/task-1
     2. feature/john/task-2
4. develop

. Servers as integration point for development team

a. This branch is long lived

b. If your team have multiple development team, it's good to have multiple develop branches

c. Development branch naming conventions

* + 1. develop/bird
    2. develop/firmware

1. release

. Release branches are long lived branches, used for planning our application releases.

a. Naming conventions could be like

* + 1. release/1.0
    2. release/2.0
    3. release/2.1

1. hotfix

. hotfix branch is used for fixing production defects.

a. Naming convention

* + 1. hotfix/bug-123
    2. hotfix/bug-456

**Configure git remote repository:**

1. List your remote repository using below command:

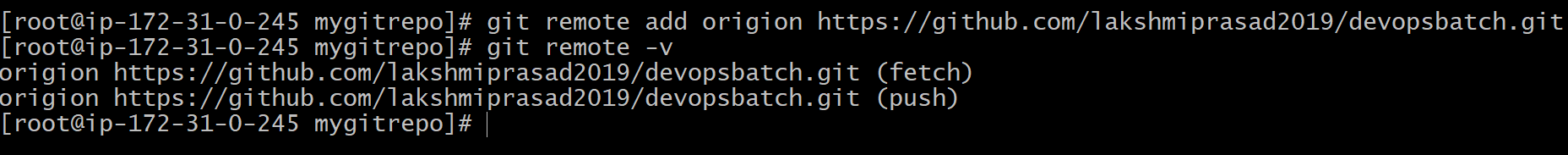
#git remote -v

It should not list any

1. Create your github account in github <https://github.com/>
2. Create your personal token as per the Demo
3. Then configure your remote repo to push the code

#git remote add <name> <url>

#git remote -v



Push the local changes to remote:

#git push <name of remote> <Branch>

Ex: #git push origin master

origin → refers to remote repository url

master → is a branch

Most commonly used remote repositories are

1. Gighub
2. Bitbucket
3. Gitlab
4. SVN

We can have multiple remotes can be configured

NAME

git-remote - Manage set of tracked repositories

SYNOPSIS

git remote [-v | --verbose]

git remote add [-t <branch>] [-m <master>] [-f] [--[no-]tags] [--mirror=(fetch|push)] <name> <url>

git remote rename <old> <new>

git remote remove <name>

git remote set-head <name> (-a | --auto | -d | --delete | <branch>)

git remote set-branches [--add] <name> <branch>...

git remote get-url [--push] [--all] <name>

git remote set-url [--push] <name> <newurl> [<oldurl>]

git remote set-url --add [--push] <name> <newurl>

git remote set-url --delete [--push] <name> <url>

git remote [-v | --verbose] show [-n] <name>...

git remote prune [-n | --dry-run] <name>...

git remote [-v | --verbose] update [-p | --prune] [(<group> | <remote>)...]

Git Clone:

Normal usecase: To get the code from the remote repository 1st time we can use clone

git clone is a Git command line utility which is used to target an existing repository and create a clone, or copy of the target repository.

#git clone <ssh link of remote repo>

Git Pull

* Pulls remote commits to local and merges with local repository

Git Fetch

Get remote commits to local and don't merge with local repository.

What is the difference between git pull and fetch? (imp)

Ans) Pull will fetch and merge & Fetch will fetch and will not merge.

**What is HEAD in git?**

HEAD is a lightweight pointer, it tells currently where you're pointing to. For example if you are currently in feature-1 branch then your HEAD is point to feature-1.

**What is detached HEAD in git?**

If HEAD is not pointing to the latest commit in the branch, then we could say it is in detached head state.

**Bare Repository:**

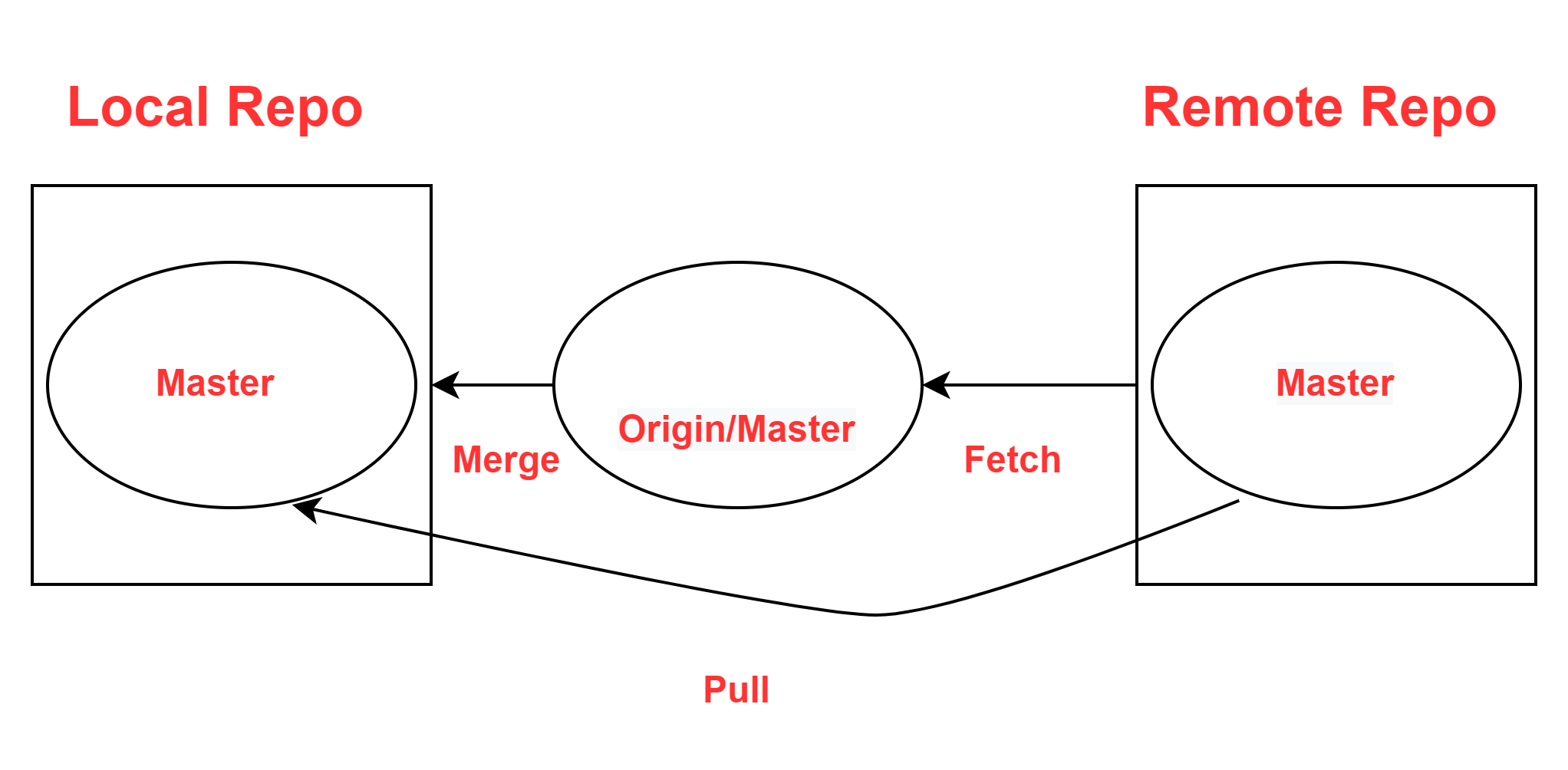
Bare repository is an empty repo which we can create but if you create a repo as a bare no working/staging area present for that and no commit will be allowed on bare repo.

#git inti --bare

This command creates a bare repo, which apparently is a repo without a working directory, meaning you can’t do something like `ls` to see the contents of the git repo. But clearly it has all the underlying plumbing when you do a `ls –la` on the folder’s contents.

**Git Pull/Pull request:**

The term pull is used to receive data from GitHub. It fetches and merges changes from the remote server to your working directory. The git pull command is used to pull a repository.



Pull request is a process for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their remote server account. Pull request announces all the team members that they need to review the code and merge it into the master branch.

**Merge conflicts:**

A conflict arises when two separate branches have made edits to the same line in a file, or when a file has been deleted in one branch but edited in the other. Conflicts will most likely happen when working in a team environment.

To resolve it we have to manually modify the files and merge again.

**Fork:**

A fork is a rough copy of a repository. Forking a repository allows you to freely test and debug with changes without affecting the original project. One of the excessive use of forking is to propose changes for bug fixing. To resolve an issue for a bug that you found, you can:

1. Fork the repository.
2. Make the fix.
3. Forward a pull request to the project owner.

Forking is not a Git function; it is a feature of Git services like GitHub.

**Git Rebase:**

Rebasing is a process to reapply commits on top of another base trip. It is used to apply a sequence of commits from distinct branches into a final commit. It is an alternative of git merge command. It is a linear process of merging.

Rebase will not merge and create a new commit, it will copy the difference and just change your base master branch with an updated one.

prasad (story/hare-and-tortoise)$ git log --oneline

58d0e78 (HEAD -> story/hare-and-tortoise) Add first draft of hare-and-tortoise story

749433b Added the story index file

ddc7eeb (story/frogs-and-ox) Completed frogs-and-ox story

bf1ee3e Add incomplete frogs-and-ox story

9063134 Added the lion and mouse story

prasad (story/hare-and-tortoise)$ git rebase master

Successfully rebased and updated refs/heads/story/hare-and-tortoise.

prasad (story/hare-and-tortoise)$ git log --oneline

553f021 (HEAD -> story/hare-and-tortoise) Add first draft of hare-and-tortoise story

4ebd115 (master) Merge branch 'master' of http://git.example.com/prasad/story-blog

e62f265 (origin/master) Added the wolf and goat story

749433b Added the story index file

8814169 Updated the story index file

92bc757 Added the fox and grapes story

ddc7eeb (story/frogs-and-ox) Completed frogs-and-ox story

bf1ee3e Add incomplete frogs-and-ox story

9063134 Added the lion and mouse story

#git rebase master (from your local branch)

Interactive rebase will give you the option you to pick which commit you want to rebase.

Using pick and squash you can pick what ever commits you want you can rebase it.

#git rebase -i HEAD~4

**Git cherry-pick:**

It would happen if one branch is having certain changes and you want to apply those in some other branch but not all. That time you can pick some commit up to where you want changes so you can cherry-pick that commit.

#git cherry-pick <required commit>

$ git status

On branch master

Your branch is up to date with 'origin/master'.

You are currently cherry-picking commit c95e0dd.

(fix conflicts and run "git cherry-pick --continue")

(use "git cherry-pick --skip" to skip this patch)

(use "git cherry-pick --abort" to cancel the cherry-pick operation)

# **Resetting and Reverting**

To make undo your commits

#git revert

#git reset

#git reset –soft <it will save the latest commit data>

#git reset –heard <it will remove entire history of last commits or above commits>

# **Stashing:**

If you want to move one branch to other branch by saving all your current working files, just stach it.

#git stash

#git stash pop

#git stash show

**.gitignore:**

It is the file which used to ignore the pushing some sensitive files to remote repository or public repositories. Create file as .gitignore and put what are the files you want to avoid to push the remote/public repos.