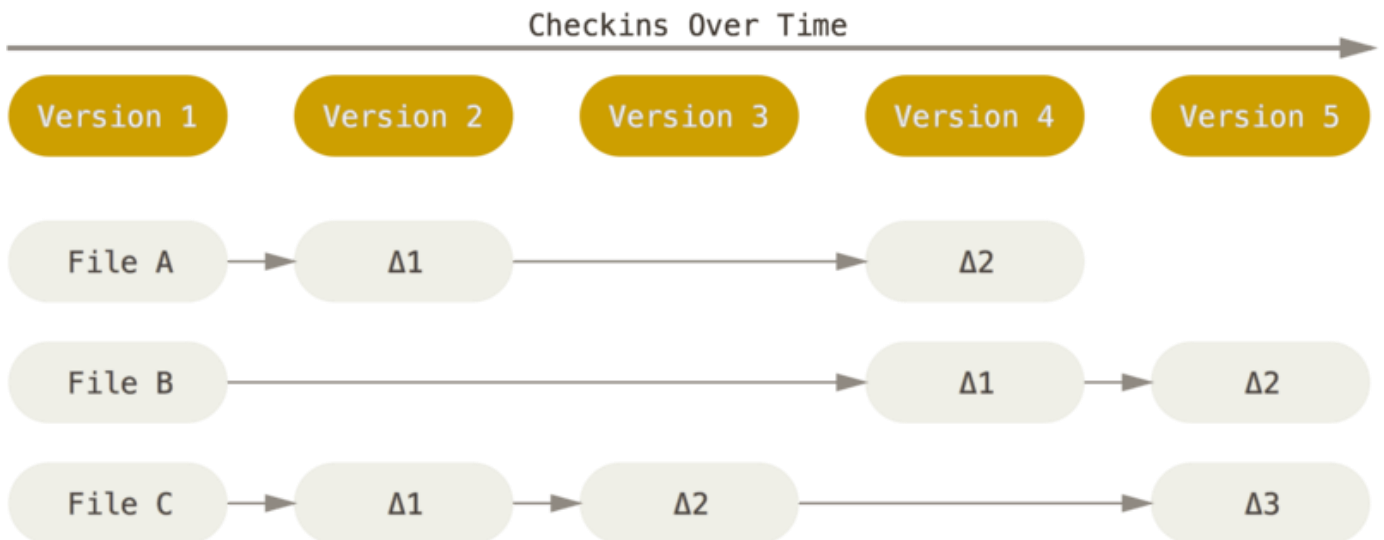


GIT BASICS NOTES

WHAT IS GIT?

Git is a version control system used for tracking changes in computer files.



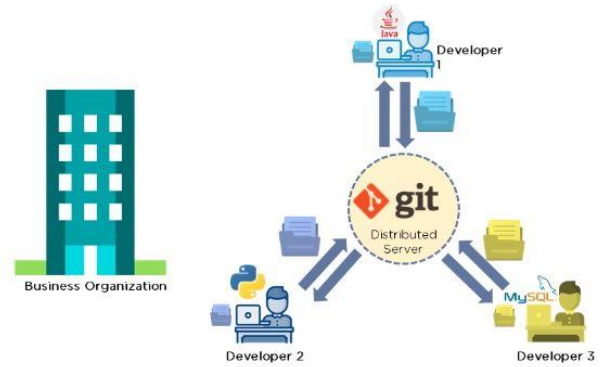
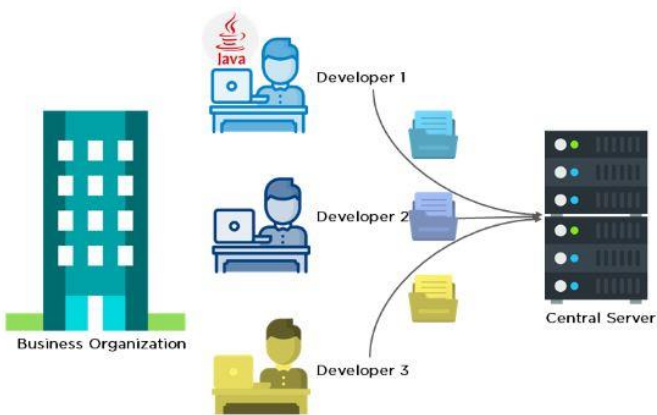
WHY GIT?

- **Streamlined Development:** Git enables synchronized, efficient collaboration. Each team member has a local code instance, facilitating concurrent work on branches. Offline capabilities ensure uninterrupted productivity.
- **Enhanced Team Efficiency:** Git simplifies change tracking, freeing up your team to focus on coding. Its local repository speeds up operations, boosting productivity.
- **Global Collaboration:** Git's open-source nature welcomes global contributions, enhancing software with features and plugins. Linux, boasting 15M lines of code, demonstrates its power.
- **Robust Security:** Utilizing SHA-1 cryptography, Git safeguards your work, ensuring version, file, and directory integrity.
- **Industry Standard:** Git enjoys widespread popularity and comprehensive IDE support, solidifying its status as an industry-standard tool.

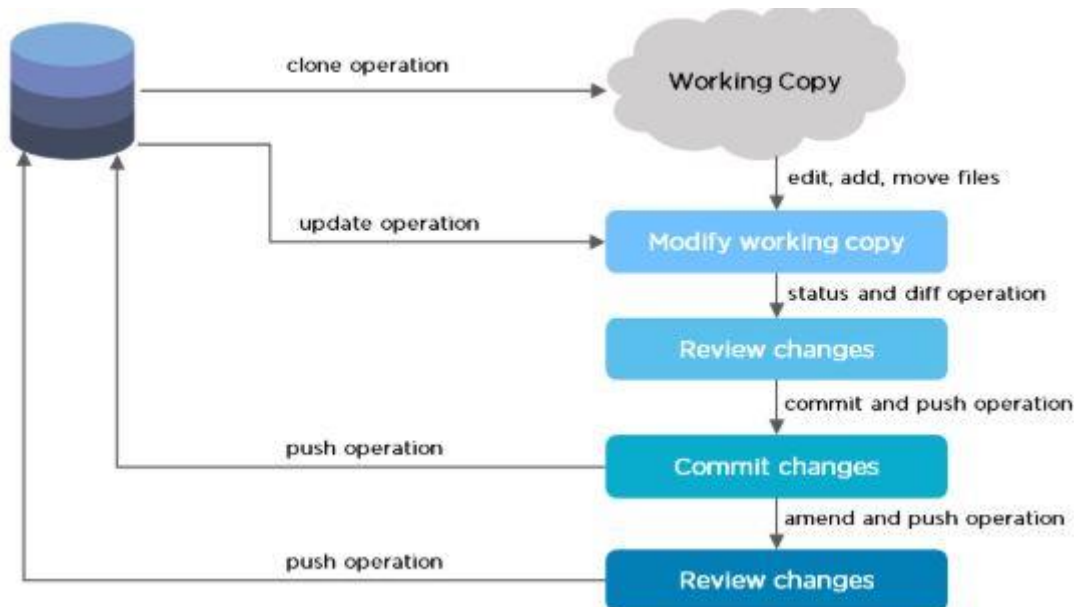
Features of Git

- **Tracks history** → Git meticulously records changes to your code over time, providing a historical timeline of edits. For instance, you can trace the evolution of a software project by examining its commit history on GitHub.
- **Free and open source** → Git is both cost-free and open to the community for improvement. It powers many open-source projects, such as the Linux kernel, which benefits from global contributions.
- **Supports non-linear development** → Git allows developers to work on multiple features simultaneously, like creating separate branches for new features or bug fixes within a project. This enhances code organization and efficiency.
- **Creates backups** → Git serves as a robust backup system. Every committed change acts as a backup, preserving your work's state. In the event of data loss, Git offers a reliable source for recovery.
- **Scalable** → Git effortlessly handles projects of varying sizes. Whether you're working on a small personal repository or a massive enterprise-level codebase, Git scales to meet your needs.
- **Supports collaboration** → Git fosters seamless teamwork, allowing multiple developers to collaborate on a single codebase. Tools like GitLab and Bitbucket facilitate this by offering collaborative features and code review capabilities.
- **Branching is easier** → Creating and managing branches in Git is straightforward. For instance, you can quickly create a new branch to work on a feature, isolating your changes until they're ready for integration into the main codebase.
- **Distributed development** → Git's distributed nature enables developers to work independently and collaboratively, even when they're geographically dispersed. Each contributor has their own local copy of the repository, promoting parallel development and merging of changes.

BEFORE GIT VS AFTER GIT



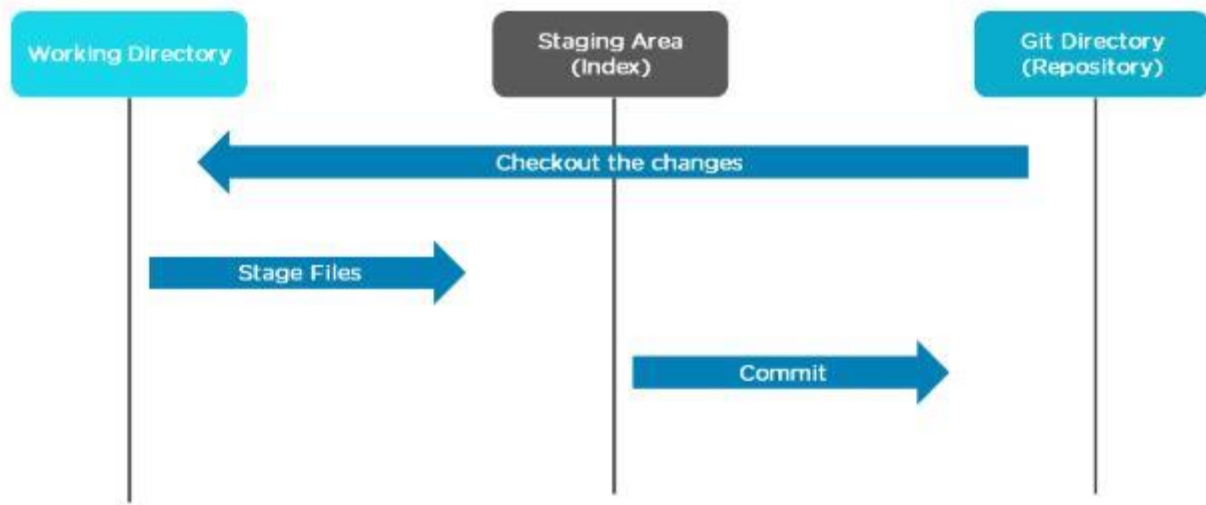
GIT WORKFLOW



The Git workflow is divided into three states:

- Working directory - Modify files in your working directory
- Staging area (Index) - Stage the files and add snapshots of them to your staging area
- Git directory (Repository) - Perform a commit that stores the snapshots permanently to your Git directory. Checkout any existing version, make changes, stage them and commit.

Below is the pictorial representation of git workflow



Terms we use

- **Repository (Repo):** A Git repository is a collection of files and version history for a project. It can be local (on your computer) or hosted on a remote server (like GitHub).
- **Branch:** A branch in Git is a separate line of development. It allows you to work on features, bug fixes, or experiments without affecting the main codebase.
- **Commit:** A commit is a snapshot of changes made to the code at a specific point in time. It includes a unique identifier, a message describing the changes, and references to the previous commit(s).
- **Clone:** Cloning a repository means creating a local copy of a remote repository. It allows you to work on the code locally and interact with the remote repository.
- **Pull:** Pulling involves fetching changes from a remote repository and merging them into the current branch. It updates your local copy with the latest changes.

- **Push:** Pushing refers to sending your local commits to a remote repository. It makes your changes available to others working on the same project.
- **Merge:** Merging combines changes from one branch into another. It's often used to integrate features or bug fixes into the main codebase.
- **Pull Request (PR):** A pull request is a request to merge changes from one branch (usually created by a contributor) into another branch (usually the main branch). It's common in open-source and collaborative development.
- **Fork:** Forking a repository means creating a copy of it in your GitHub account. It allows you to make changes independently and, if desired, create pull requests to contribute changes back to the original repository.
- **Conflict:** A conflict occurs when Git is unable to automatically merge changes from different branches or commits. It requires manual resolution.
- **Stash:** Stashing allows you to temporarily save changes that you're not ready to commit. It's useful when you need to switch branches or address an urgent issue.
- **Remote:** A remote is a version of a repository hosted on a server, typically on platforms like GitHub or GitLab. It's a reference to a remote repository.
- **Origin:** "Origin" is a common default name for the remote repository you cloned from. It's often used as the default remote reference in Git.
- **Checkout:** Checking out a branch or commit means switching your working directory and HEAD to that specific branch or commit.

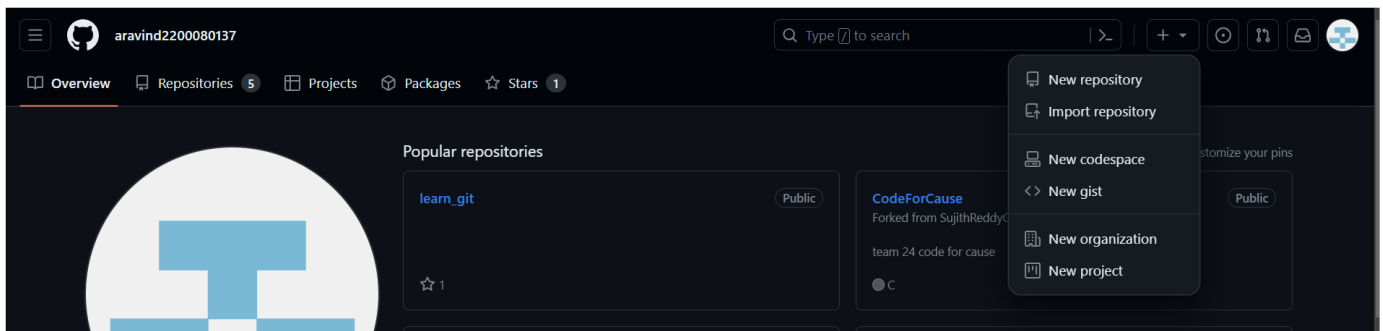
HOW TO CREATE A REPOSITORY

Prerequisite : Account in gitlab/github/ Bitbucket

Note : I'm pasting the images/command's according to github documentation based on your intermediate medium it may vary for more information visit your intermediate application documentation

Steps to create your own repository(repo) :

- From github(my intermediate application), click in the global nav bar click on '+' → choose "new repository"




- Choose repo name

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 * Repository name *

 aravind2200080137 / learngit

✔ learngit is available.

- Choose type of repo

☒  **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.

- And then click on "create repository" your repo is ready to use

HOW TO CLONE THE CREATED REPOSITORY INTO LOCAL SYSTEM

NOTE : → prefer “git bash” for following commands

→ For installation visit “ <https://git-scm.com/> ” > Downloads

- Navigate to some location on your local system, in my case I am choosing “Documents”
- If you are confused with your current location you can check it by using ‘pwd’ command

```
MINGW64:/c/Users/91837
91837@dishuuuu MINGW64 ~
$ pwd
/c/Users/91837
```

- Now to navigate to your desired location we use ‘cd’ command, in my case it’s downloads

```
91837@dishuuuu MINGW64 ~
$ cd documents
91837@dishuuuu MINGW64 ~/documents
$ |
```

NOTE : If you want to check the available folders or files on your current working location, we can do it by using ‘ls’ command

```
91837@dishuuuu MINGW64 ~/documents
$ ls
2200080137/  Aravind/  Django-Manage-Pet-adoptions/  'My Music'@  'My Pictures'@  'My Videos'@  main.c  my_pfsd_project/
```

- Now we need a local directory to contain the repositories so we make a new directory/folder we can do it with GUI and also with the help of ‘mkdir’ command

```
91837@dishuuuu MINGW64 ~/documents
$ mkdir repos
91837@dishuuuu MINGW64 ~/documents
$ |
```

- Now navigate to created directory/folder with ‘cd’ command

- Now to clone the repository we created in github we need the repository link for that
- open github
 - choose the repository that you need to clone click on it
 - search for '<> code' (or) 'clone' (or) 'clone this repository' click on it
 - you can see this kind of pop which show options like “**Clone with SSH**”, “**Clone with HTTPS**”, depending on your settings the protocol will automatically set in github to 'HTTPS', 'SSH'. In my case the protocol is set to 'HTTPS'
 - copy the link and use 'git clone <your_repo_link>' command to clone the repository to your local system

this is how it's done

```
MINGW64/c/Users/91837/documents/repos/learn_git
91837@dishuuuu MINGW64 ~/documents
$ ls
2200080137/  Aravind/  Django-Manage-Pet-adoptions/  'My Music'@  'My Pictures'@  'My Videos'@  main.c  my_pfsd_project/  nanda  repos/
91837@dishuuuu MINGW64 ~/documents
$ cd repos/
91837@dishuuuu MINGW64 ~/documents/repos
$ git clone https://github.com/aravind2200080137/learn_git.git
Cloning into 'learn_git'...
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.
91837@dishuuuu MINGW64 ~/documents/repos
$ ls
learn_git/
91837@dishuuuu MINGW64 ~/documents/repos
$ cd learn_git/
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (main)
$ |
```

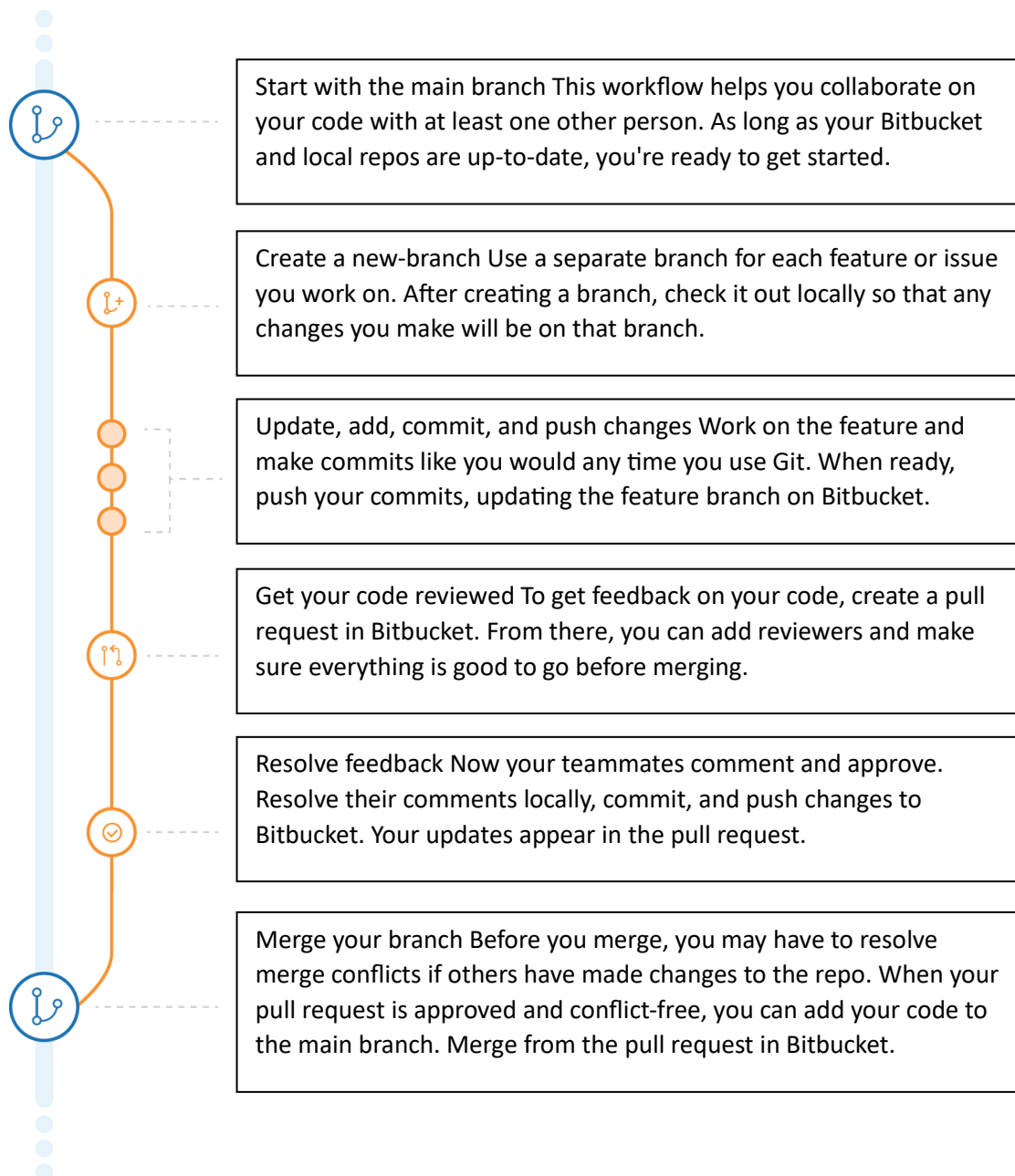

BRANCH IN GIT

WHAT IS BRANCH ?

A branch in Git is like a separate line of development that diverges from the main codebase (usually the master or main branch). It allows developers to work on new features, bug fixes, or experiments in isolation without affecting the main codebase.

(OR)

A branch represents an independent line of development. Branches serve as an abstraction for the edit/stage/commit process. You can think of them as a way to request a brand new working directory, staging area, and project history.



“Git branch – git branch ” allows us to done operations like

- ✓ create
- ✓ list
- ✓ rename
- ✓ delete

Lets create a branch in our repository

- First lets us check what branch are you working on currently to do that we use

```
MINGW64:/c/Users/91837/documents/repos/learn_git
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (main)
$ git branch
* main
```

- By default main branch is created
- To create a new branch we use command “git branch <branch_name>” (or) “git branch feature/my-feature” this command will create a new branch but it won’t shift to it

```
MINGW64:/c/Users/91837/documents/repos/learn_git
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (main)
$ git branch
* main

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (main)
$ git branch new_branch

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (main)
$ git branch
* main
  new_branch
```

NOTE : ‘Asterisk(*)’ symbol represents the current branch you are in

- To change the current branch to some other branch we use “git checkout <branch_name>”

```

MINGW64:/c/Users/91837/documents/repos/learn_git

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (main)
$ git checkout new_branch
Switched to branch 'new_branch'

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (new_branch)
$ git branch
main
* new_branch

```

- We do both Creation and shifting to other branch with the help of “git checkout -b <branch_name>”

```

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (new_branch)
$ git checkout -b new_branch1
Switched to a new branch 'new_branch1'

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (new_branch1)
$ git branch
main
new_branch
* new_branch1

```

- Now to rename a specific branch name from the available branches we use “git branch -m <old_branch_name> <new_branch_name>”

```

MINGW64:/c/Users/91837/documents/repos/learn_git

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (new_branch1)
$ git branch -m new_branch test

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (new_branch1)
$ git branch
main
* new_branch1
test

```

- To rename current working branch i.e ‘*new_branch1’ we use similar command “git branch -m <new_branch_name>”

```

MINGW64:/c/Users/91837/documents/repos/learn_git

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (new_branch1)
$ git branch
main
* new_branch1
test

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (new_branch1)
$ git branch -m Feature

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git branch
* Feature
main
test

```

- To delete a branch in Git, you can use the git branch command with the -d or -D option followed by the branch name. i.e “git branch -d <branch_name>”

```

MINGW64:/c/Users/91837/documents/repos/learn_git

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git branch -d test
Deleted branch test (was 63d782d).

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git branch
* Feature
main

```

NOTE

- Safe Delete (with -d): used to delete a branch only if it has been fully merged into the branch you are currently on. If the branch contains changes that haven't been merged yet, Git will prevent you from deleting it with this option.
- Force Delete (with -D): used to forcefully delete a branch regardless of whether it has been fully merged or not. Be cautious when using this option, as it can lead to data loss if you delete a branch with unmerged changes.

IMPORTANT OPTIONS

→ -f/--force

Reset <branchname> to <start-point>, even if <branchname> exists already. Without -f, git branch refuses to change an existing branch.

Note : git branch -f <branchname> [<start-point>], even with -f, refuses to change an existing branch <branchname> that is checked out in another work tree linked to the same repository.

→ -m/--move

Move/rename a branch, together with its config and reflog.

→ -M

Shortcut for --move --force.

HOW TO PUSH THE LOCALLY PRESENT FILE TO SERVER

- First we need a file that needed to be pushed so to create new file we use 'touch' command (or) we can go to that location and file/files graphically

```
MINGW64:/c/Users/91837/documents/repos/learn_git
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ ls
README.md
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ touch example.txt
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ ls
README.md  example.txt
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$
```

- We added 'example.txt' to our repository to push it we need to initialize to do that we use "git init"
- Next we need to add and commit the file to track, if you are confused then use "git status" to get the list of untracked and tracked list of files
- To add file we use "git add <file_name>"
- If there are more files we can use "git add <file_name1> <file_name2> ..." (or) "git add ." (or) "git add *" this commands will add multiple files
- To commit the changes we use 'git commit -m "<message/ Description of changes>" '

```
MINGW64:/c/Users/91837/documents/repos/learn_git
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git init
Reinitialized existing Git repository in C:/Users/91837/Documents/repos/learn_git/.git/
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git status
On branch Feature
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    example.txt

nothing added to commit but untracked files present (use "git add" to track)
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git add example.txt
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git status
On branch Feature
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   example.txt
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git commit -m "first commit"
[Feature 56e342e] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 example.txt
```

- ➔ Set up a remote : You need to specify the remote repository where you want to push your changes. This is typically done using the “git remote add” command example : `git remote add <name_for_remote_repository> <remote_repository_url>`

```
MINGW64:/c/Users/91837/documents/repos/learn_git
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git remote add test https://github.com/aravind2200080137/learn_git.git
```

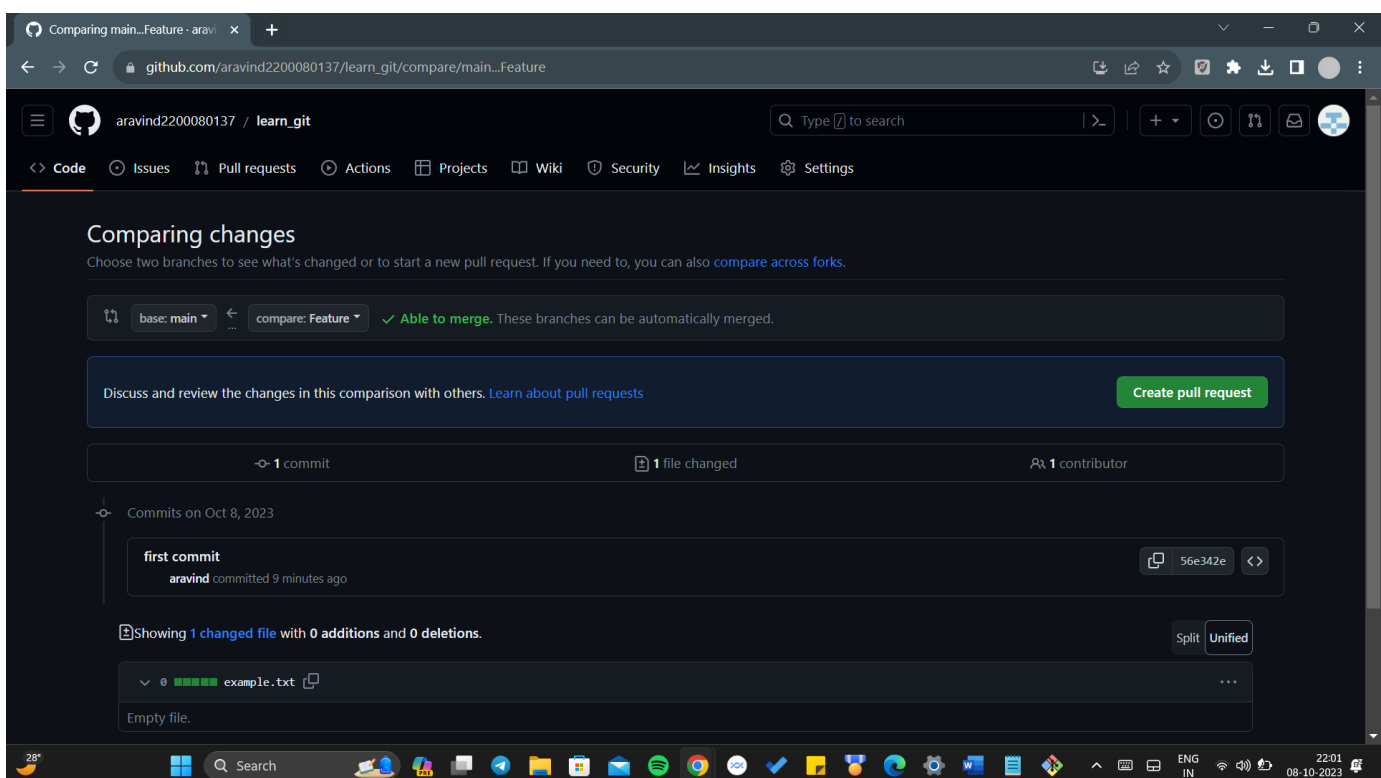
- ➔ Push to the Remote Repository

Finally, you can push your changes to the remote repository using the git push command “`git push -u < name_for_remote_repository > <branch_name>`”

```
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git push -u test Feature
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 278 bytes | 278.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'Feature' on GitHub by visiting:
remote:   https://github.com/aravind2200080137/learn_git/pull/new/Feature
remote:
To https://github.com/aravind2200080137/learn_git.git
 * [new branch]      Feature -> Feature
branch 'Feature' set up to track 'test/Feature'.

91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git status
On branch Feature
Your branch is up to date with 'test/Feature'.

nothing to commit, working tree clean
```



HOW TO PULL FROM SERVER TO LOCAL SYSTEM

➔ Navigate to Your Local Repository:

Open a terminal or command prompt and navigate to the local directory of your Git repository using the `cd` command.

➔ Pull Changes from the Remote Server:

Use the `git pull` command followed by the name of the remote repository (usually "origin" by convention) and the branch you want to pull changes from. By default, this will pull changes into the currently checked-out branch using “`git pull <remote_repository> branch_name`”

```
MINGW64:/c/Users/91837/documents/repos/learn_git
91837@dishuuuu MINGW64 ~/documents/repos/learn_git (Feature)
$ git pull test Feature
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 681 bytes | 85.00 KiB/s, done.
From https://github.com/aravind2200080137/learn_git
* branch          Feature      -> FETCH_HEAD
   56e342e..dc24744 Feature      -> test/Feature
Updating 56e342e..dc24744
Fast-forward
 example.txt | 1 +
 1 file changed, 1 insertion(+)
```


SOME MORE COMMANDS

GIT BASICS

<code>git init <directory></code>	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
<code>git clone <repo></code>	Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
<code>git config user.name <name></code>	Define author name to be used for all commits in current repo. Devs commonly use <code>--global</code> flag to set config options for current user.
<code>git add <directory></code>	Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file.
<code>git commit -m "<message>"</code>	Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.
<code>git status</code>	List which files are staged, unstaged, and untracked.
<code>git log</code>	Display the entire commit history using the default format. For customization see additional options.
<code>git diff</code>	Show unstaged changes between your index and working directory.

UNDOING CHANGES

<code>git revert <commit></code>	Create new commit that undoes all of the changes made in <commit>, then apply it to the current branch.
<code>git reset <file></code>	Remove <file> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes.
<code>git clean -n</code>	Shows which files would be removed from working directory. Use the <code>-f</code> flag in place of the <code>-n</code> flag to execute the clean.

REWRITING GIT HISTORY

<code>git commit --amend</code>	Replace the last commit with the staged changes and last commit combined. Use with nothing staged to edit the last commit's message.
<code>git rebase <base></code>	Rebase the current branch onto <base>. <base> can be a commit ID, branch name, a tag, or a relative reference to HEAD.
<code>git reflog</code>	Show a log of changes to the local repository's HEAD. Add <code>--relative-date</code> flag to show date info or <code>--all</code> to show all refs.

GIT BRANCHES

<code>git branch</code>	List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>.
<code>git checkout -b <branch></code>	Create and check out a new branch named <branch>. Drop the <code>-b</code> flag to checkout an existing branch.
<code>git merge <branch></code>	Merge <branch> into the current branch.

REMOTE REPOSITORIES

<code>git remote add <name> <url></code>	Create a new connection to a remote repo. After adding a remote, you can use <name> as a shortcut for <url> in other commands.
<code>git fetch <remote> <branch></code>	Fetches a specific <branch>, from the repo. Leave off <branch> to fetch all remote refs.
<code>git pull <remote></code>	Fetch the specified remote's copy of current branch and immediately merge it into the local copy.
<code>git push <remote> <branch></code>	Push the branch to <remote>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn't exist.

GIT CONFIG

<code>git config --global user.name <name></code>	Define the author name to be used for all commits by the current user.
<code>git config --global user.email <email></code>	Define the author email to be used for all commits by the current user.
<code>git config --global alias. <alias-name> <git-command></code>	Create shortcut for a Git command. E.g. <code>alias.glog "log --graph --oneline"</code> will set "git glog" equivalent to "git log --graph --oneline".
<code>git config --system core.editor <editor></code>	Set text editor used by commands for all users on the machine. <editor> arg should be the command that launches the desired editor (e.g., vi).
<code>git config --global --edit</code>	Open the global configuration file in a text editor for manual editing.

GIT DIFF

<code>git diff HEAD</code>	Show difference between working directory and last commit.
<code>git diff --cached</code>	Show difference between staged changes and last commit

GIT RESET

<code>git reset</code>	Reset staging area to match most recent commit, but leave the working directory unchanged.
<code>git reset --hard</code>	Reset staging area and working directory to match most recent commit and overwrites all changes in the working directory.
<code>git reset <commit></code>	Move the current branch tip backward to <commit>, reset the staging area to match, but leave the working directory alone.
<code>git reset --hard <commit></code>	Same as previous, but resets both the staging area & working directory to match. Deletes uncommitted changes, and all commits after <commit> .

GIT REBASE

<code>git rebase -i <base></code>	Interactively rebase current branch onto <base>. Launches editor to enter commands for how each commit will be transferred to the new base.
---	---

GIT PULL

<code>git pull --rebase <remote></code>	Fetch the remote's copy of current branch and rebases it into the local copy. Uses git rebase instead of merge to integrate the branches.
---	---

GIT PUSH

<code>git push <remote> --force</code>	Forces the git push even if it results in a non-fast-forward merge. Do not use the --force flag unless you're absolutely sure you know what you're doing.
<code>git push <remote> --all</code>	Push all of your local branches to the specified remote.
<code>git push <remote> --tags</code>	Tags aren't automatically pushed when you push a branch or use the --all flag. The --tags flag sends all of your local tags to the remote repo.

FOR MORE RELATED INFORMATION VIST

- <https://git-scm.com/doc>
- <https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>
- <https://www.w3schools.com/git/>