Git/GitHub/Bitbucket

1.VCS(Version Control System)

Software Tool Used By developers to manage changes to a source code & other files in a project

It allows multiple contributors to collaborate in a project

2.Check Version Using Terminal(GitBash/cmd/VS Code Terminal)

git --version

3.Setting global git configuration options

git config - - global user.name “Pushpan Bhaumik”

git config - - global user.email pushpanbhaumik200@gmail.com

i)When we make commits to a Git repository, Git records who made those changes

ii)Setting your name via git config - - global user.name **ensures that our commits are associated with our name**

iii)Some Git hosting platforms like GitHub, use our configured name to link commits to the user profile

4.Changing the name , email using git

git config --global –edit $\longrightarrow$ will open VM Editor $\longrightarrow$ Press I to insert some changes $\longrightarrow$ press Escape $\longrightarrow$:wq $\longrightarrow$ back to GitBash

5.Command to create a directory as git repo

i) Create a local Folder

ii)change directories using cd command to navigate to the local folder

iii)**git init :** command that initializes a new git repository in the current directory or in a specified directory

iv)When we run **git init** command, Git creates a new subdirectory named **.git** in the current directory

v) After **git init** command, it gives the below msg in the terminal:

Initialized empty Git repository in C:/Users/pbhaumik/Downloads/GitRepo2024-Feb/.git/

vi)ls -a $\longrightarrow$ . .. .git

6. Git Repositories

In Git, a repository, often abbreviated as "repo," refers to a central location where a collection of files and directories are stored, along with metadata about the project's history and changes. A Git repository contains everything needed to manage a project's version control

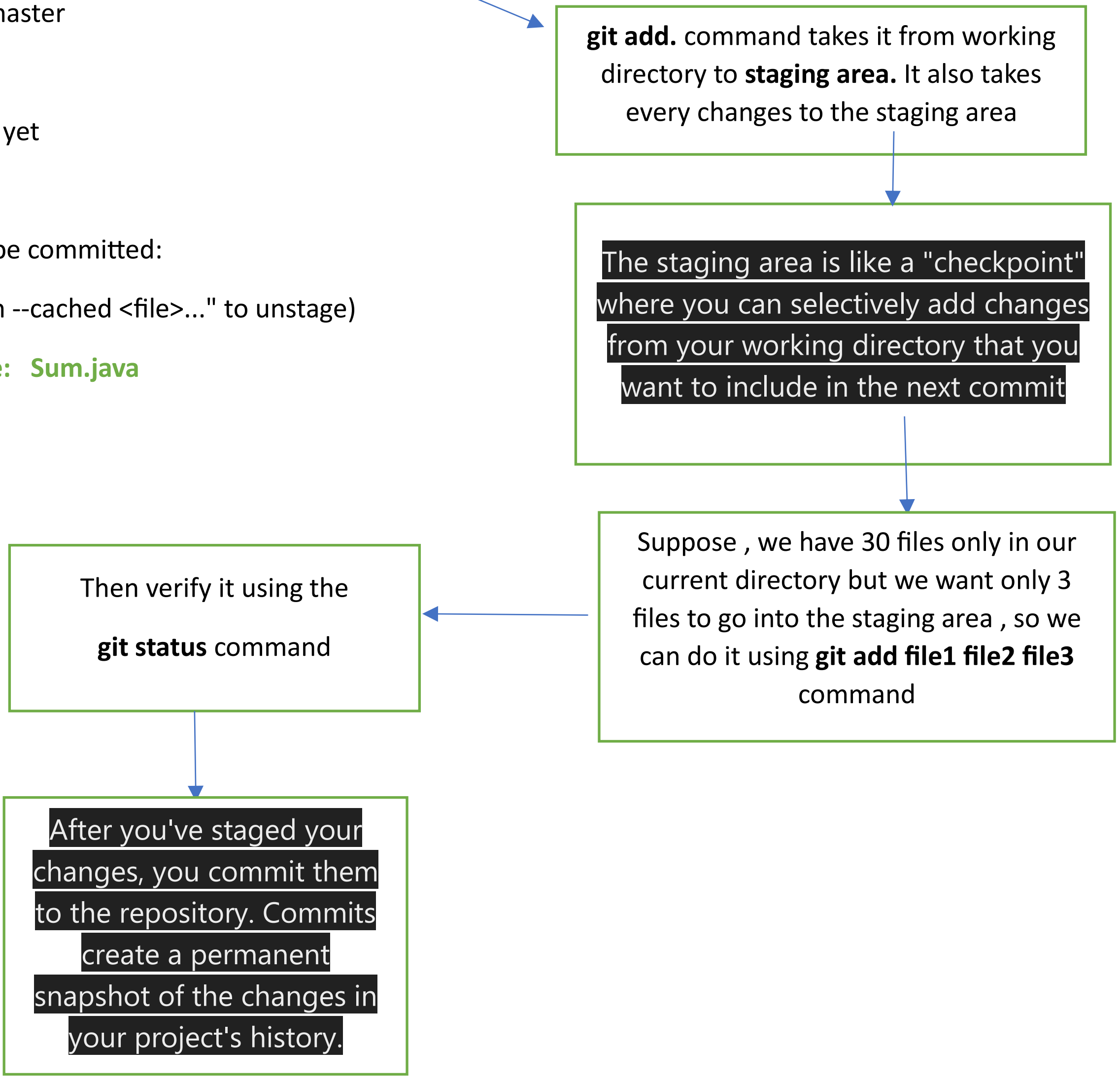
7. Git Working Directory to Git Staging Area

i)**git status** : This command gives us the current state of the repository

```
$ git status

On branch master

No commits yet


Untracked files:

  (use "git add <file>..." to include in what will be committed)

        Sum.java


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

After adding a file suppose Sum.java in the repository location(GitRepo2024-Feb) , if we use the git status **command after using the** git add . **command, it will give us the following info**.

```
$ git status

On branch master


No commits yet


Changes to be committed:

  (use "git rm --cached <file>..." to unstage)

        new file:   Sum.java
```



git add. command takes it from working directory to **staging area.** It also takes every changes to the staging area

The staging area is like a "checkpoint" where you can selectively add changes from your working directory that you want to include in the next commit

Suppose , we have 30 files only in our current directory but we want only 3 files to go into the staging area , so we can do it using **git add file1 file2 file3** command

Then verify it using the

git status command

After you've staged your changes, you commit them to the repository. Commits create a permanent snapshot of the changes in your project's history.

$$\begin{tikzcd}
\textbf{Working Directory} \arrow[r] & \textbf{git add.} \arrow[r] & \textbf{Staging Area} \arrow[r] & \textbf{git commit} \arrow[d] \\
 & & & \boxed{\textbf{Repository}}
\end{tikzcd}$$

8. git log command:

It is used to display the commit history of a repository.

9. git commit -m "initial commit message"

It is used to commit the changes from the staging area to the repository. Commit creates a permanent snapshots of the changes in the repository.

Suppose, I have added one new file as **Diff.java** and modified an existing **Sum.java** file. Now I need to commit it in the following way after adding them to the staging area.

Then we can use **git log** to check the commit history

We can use the **git status** command to check what are the changes made.

```
$ git status

On branch master

Untracked files:

  (use "git add <file>..." to include in what will be committed)

        Diff.java


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


pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (master)

$ git status

On branch master

Changes not staged for commit:

  (use "git add <file>..." to update what will be committed)

  (use "git restore <file>..." to discard changes in working directory)

        modified:   Sum.java


Untracked files:

  (use "git add <file>..." to include in what will be committed)

        Diff.java
```

```
$ git add Diff.java

pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   Diff.java

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   Sum.java
```

Now here, we have only added Diff.java to the staging area instead of the modifications in the Sum.java file. So, Changes to be committed is only showing **Diff.java**

Now, commit the changes from the staging area to the repository as follows:

```
$ git commit -m "Added Diff.java file"
[master 24d3aec] Added Diff.java file
 1 file changed, 7 insertions(+)
 create mode 100644 Diff.java
```

Now, we can use git log command to check the commit history:

```
$ git log
commit 24d3aece6612a4887764bb75fdd8712a07b24a14 (HEAD -> master)
Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>
Date:   Sat Feb 24 21:44:06 2024 +0530

    Added Diff.java file

commit 52f9669dfa20c3cf030a2a9ae951586044390141
Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>
Date:   Sat Feb 24 21:39:57 2024 +0530

    Added Sum.java file
```

Each commit basically has an unique hashcode

10.How to get back to previous commits:

git checkout <hash code> is used to move back to previous commits**.**

This command is used to **detach the HEAD and directly point HEAD** to a specific commit.

It is similar to point to a specific commit and all the commits after the commit in which we are checking out, will get discarded.

Suppose there are 3 commits:

```
$ git log

commit 0d2bd9c7741a2e48902cd7103cff9c71c91cd6c4 (HEAD -> master)

Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>

Date:   Sat Feb 24 21:44:49 2024 +0530


    Modified Sum.java


commit 24d3aece6612a4887764bb75fdd8712a07b24a14

Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>

Date:   Sat Feb 24 21:44:06 2024 +0530


    Added Diff.java file


commit 52f9669dfa20c3cf030a2a9ae951586044390141

Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>

Date:   Sat Feb 24 21:39:57 2024 +0530


    Added Sum.java file
```

Now using the git checkout command:

```
pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (master)
$ git checkout 52f9669dfa20c3cf030a2a9ae951586044390141
Note: switching to '52f9669dfa20c3cf030a2a9ae951586044390141'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 52f9669 Added Sum.java file
```

So, here we are checking out the commit with the hashcode

52f9669dfa20c3cf030a2a9ae951586044390141

Now, to get back to the master branch or to all the existing commits again

We use the following command:

git checkout master

```
$ git checkout master

Previous HEAD position was 52f9669 Added Sum.java file

Switched to branch 'master'


pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (master)

$ git log

commit 0d2bd9c7741a2e48902cd7103cff9c71c91cd6c4 (HEAD -> master)

Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>

Date:   Sat Feb 24 21:44:49 2024 +0530


    Modified Sum.java


commit 24d3aece6612a4887764bb75fdd8712a07b24a14

Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>

Date:   Sat Feb 24 21:44:06 2024 +0530


    Added Diff.java file


commit 52f9669dfa20c3cf030a2a9ae951586044390141

Author: Pushpan Bhaumik <pushpanbhaumik1997@gmail.com>

Date:   Sat Feb 24 21:39:57 2024 +0530


    Added Sum.java file
```

BRANCH:

In Git, branches are an essential part of the version control system that allow for parallel lines of development within a repository.

Branches enable developers to work on new features, bug fixes, or experiments without affecting the main codebase until the changes are ready to be integrated.

Suppose, we have a master branch , then we can create a different branch as dev branch , then inside the dev branch we can create another branch as pushpan/multiply

Useful commands:

We can give any name here

i)git branch dev : creates a dev branch from the master branch

ii)git checkout dev: creating a checkout of the branch dev, now if we make any commits here after adding the changes to the staging area, we can do it and our master branch won't even know it.

iii)git checkout -b <branch-name>: git checkout -b dev: This command creates a branch and checkout in this branch in one command instead of writing two different commands.

iv)git merge: Once, we are on the target branch, we use **git merge** followed by the branch name we want to merge in the branch where we have just taken a checkout. For example- to merge changes from a feature branch name as "pushpan/multiply" into the main branch:

For merging: first write, **git checkout dev**

Then, write **git merge pushpan/multiply ,** it will merge the changes of the **pushpan/multiply** branch

```
$ git checkout dev

Switched to branch 'dev'


pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (dev)

$ git merge pushpan/multiply

Updating 0d2bd9c..c738e33

Fast-forward

 Multiply.java | 7 +++++++

 1 file changed, 7 insertions(+)

 create mode 100644 Multiply.java
```

.gitignore: It is a configuration file used by Git to specify intentionally untracked files, the files that Git should ignore. These files are generated as part of the build process, temporary files, or editor-specific files that doesn't needs to be version-controlled

i)First of all, we need to create a .gitignore file in our branch

ii)Two ways to make it: use the command **touch .gitignore** and directly create a **.gitignore** file

iii) Suppose, we have a **secretKey.txt** file, which we don't want our git to track:

So, inside the **.gitignore** file , we can mention the **secretKey.txt file.**

Before mentioning it, if we use the **git status** command, we can see:

```
pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (master)

$ git status

On branch master

Untracked files:

  (use "git add <file>..." to include in what will be committed)

        .gitignore

        secretKey.txt
```

After mentioning it, if we use the **git status** command, we can see:

```
pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (master)

$ git status

On branch master

Untracked files:

  (use "git add <file>..." to include in what will be committed)

        .gitignore


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

Using GitHub:

Open GitHub and create a repository in GitHub.

Now as soon as we create a new repo, we can push a remote repository already we have created as follows:

```
git remote add origin https://github.com/GeekyPBhaumik/LearningGit.git
git branch -M main
git push -u origin main
```

Before the above first commands if we try **git remote -v** command, we won't get anything
But after the above first command, if we try **git remote -v** command, we will get

```
$ git remote -v
origin  https://github.com/GeekyPBhaumik/LearningGit.git (fetch)
origin  https://github.com/GeekyPBhaumik/LearningGit.git (push)

pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (master)
$ git branch -M main

pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (main)
$ git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 12, done.
Counting objects: 100% (12/12), done.
Delta compression using up to 8 threads
Compressing objects: 100% (11/11), done.
Writing objects: 100% (12/12), 1.36 KiB | 199.00 KiB/s, done.
Total 12 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/GeekyPBhaumik/LearningGit.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Similarly, we can push our dev branch too:

```
pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (main)
$ git checkout dev
Switched to branch 'dev'

pbhaumik@LIN64016186 MINGW64 ~/Downloads/GitRepo2024-Feb (dev)
$ git push -u origin dev
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
remote:
remote: Create a pull request for 'dev' on GitHub by visiting:
remote:      https://github.com/GeekyPBhaumik/LearningGit/pull/new/dev
remote:
To https://github.com/GeekyPBhaumik/LearningGit.git
 * [new branch]      dev -> dev
branch 'dev' set up to track 'origin/dev'.
```

Now, if we make some changes in our dev branch, then we can push it using the command:
git push

How Collaborators work in a project using GitHub:

i)Inviting a Collaborator:

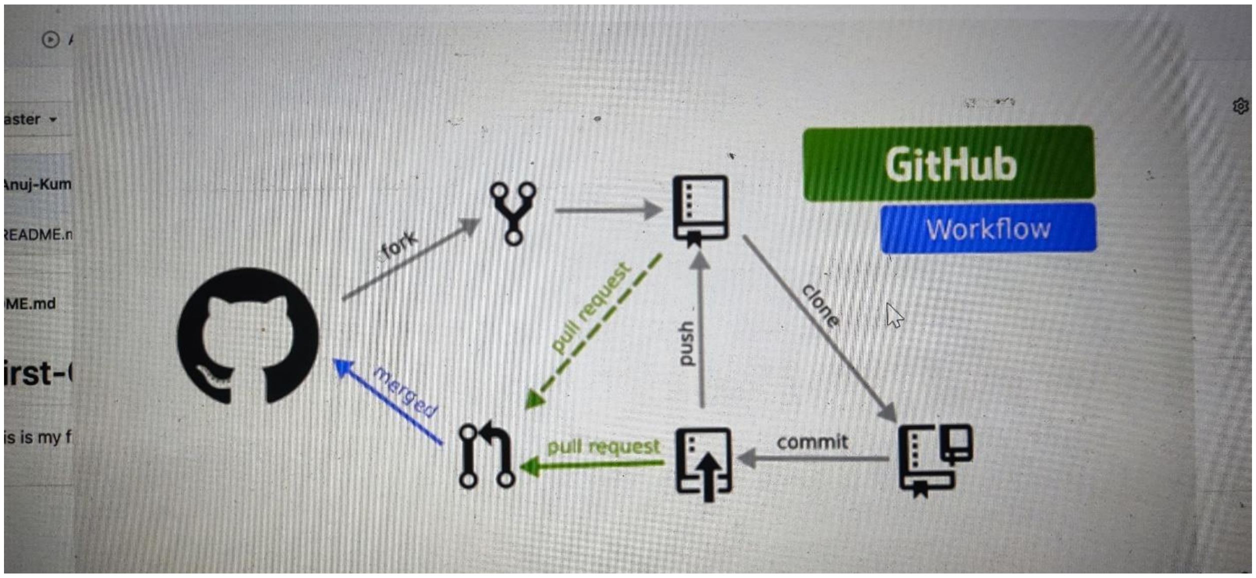
Settings -> **Collaborators** -> **Add People** -> **We can send invites to people for collaborating**

Now the person to whom the invite has been sent, will receive a mail regarding collaboration and can clone the repository in their system a, make changes and push the changes.

ii)Process of open source Contribution using GitHub:

- First, we need to find a repository to fork it
- Then fork it using the GitHub button, it will create a copy of that Repo in your GitHub profile
- We can create a clone of the forked repository in our system
- Then we can make changes in the cloned repository
- Then we can either make the changes in the master branch itself Or create a **branch** and make changes to it
- Then push the changes using **git push** or **git push -u origin <branch-name>**
- Then, using GitHub, we can create a pull request to merge the Changes in the repository from which we **forked**
- Then it depends on the repo owner to **review, add required comments, merge the pull request**

Forking a repository in the context of Git and platforms like GitHub, GitLab, and Bitbucket refers to creating a personal copy of someone else's repository into your own account



aster
Anuj-Kum
README.m
ME.md
irst-
is is my f
GitHub
Workflow
fork
pull request
push
clone
merged
pull request
commit