**Day 23: GIT**

**Introduction to the git.**

Git: This is a version control tool which is used to keep track the versions of files and directories.

Some of the version control tools are git, svn , bitbucket, gitlab, cvs

git init – This is used to initialize the git repo

**Day 24: Env Variables**

cat ~/.bashrc

cat ~/.profile

printenv – This will print environment profiles

if we want to run something when the system is up, then we need to add it to the profile.

If we want to use something through out, then we need to add it to the bashrc

<https://www.geeksforgeeks.org/environment-variables-in-linux-unix/>

**Day 25: git**

git clone url

cd Devops

vi file1

vi file2

git add . (or) git add file1 file2

git status

git commit -m “ commit message”

git commit -m “ commit message” filename

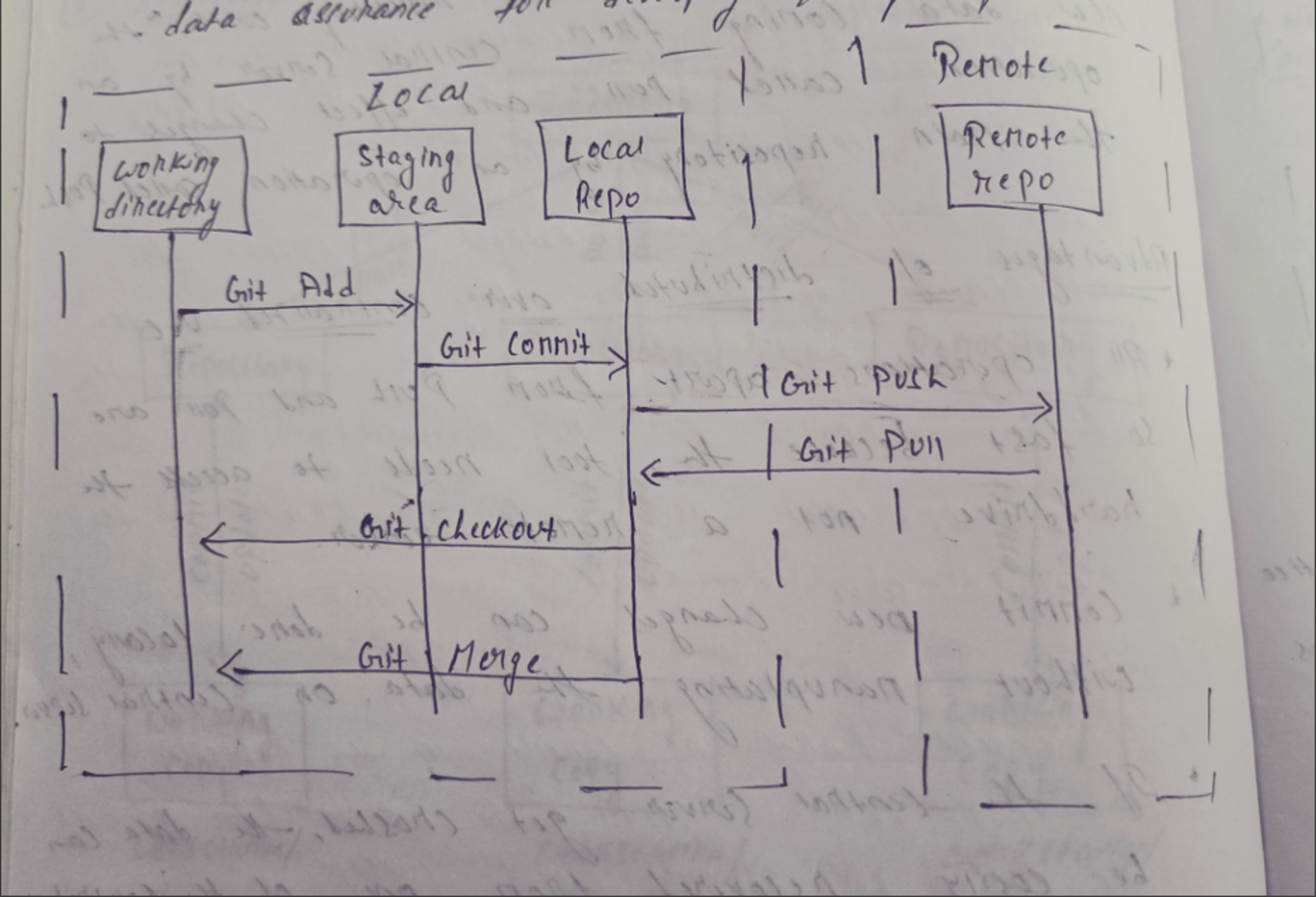
git push origin main

* username phaninandigam
* password: ghp\_lOGKKcO4wqoFWmJc29naRWrJtZC6mV14azsU

**Git architecture (git workflow)**

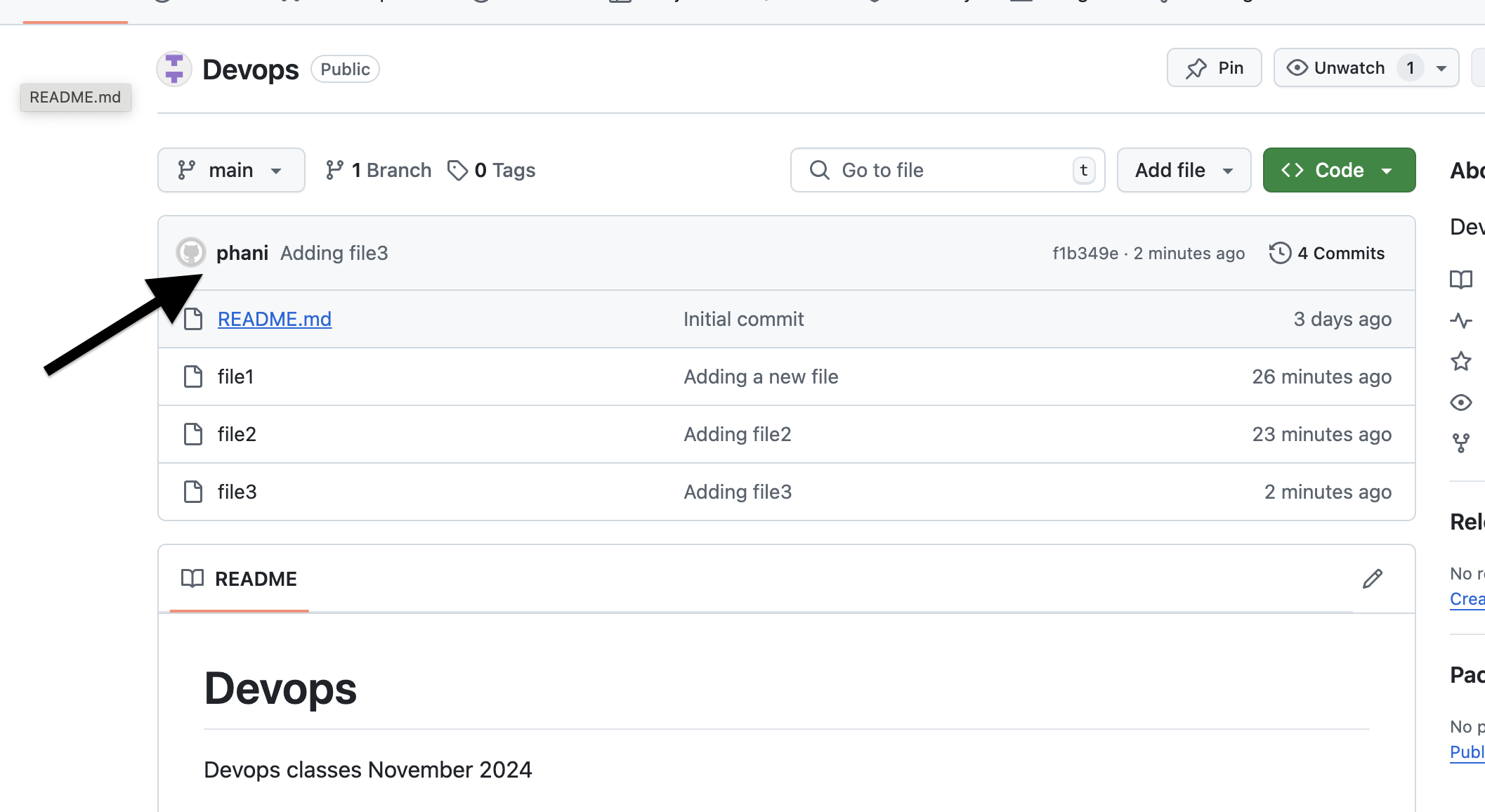
It consists of 3 steps

1. workspace: It’s a place where we edit project related files
2. staging area : It’s an intermediate area where we save the changes   
   git add filename
3. git repo/local repo : here versions of files will be tracked. After executing the git commit command, the files will be moved to the git repo.



**Git configuration:**

git config --global user.name "phani"

* This is used to configure the git user  
  

git config --global user.email [phani@gmail.com](mailto:phani@gmail.com)

* this is used to configure the email id of the user

## How to set a Git config username and password

To permanently set Git username and password fields, issue the following command in the terminal window for Git to use a credential store:

git config --global credential.helper store

If all things go right, the next time you perform a [fetch or pull](https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/Git-pull-vs-fetch-Whats-the-difference) from a remote repository, you are prompted for your Git username and password. Provide those, and they are stored permanently in a file named .gitcredentials in your account's home directory.

git status

* this is used to check whether the files are in workspace or in staging area or in git repo

git add filename / git add .

* this is used to move files from work space to staging area

git add –all

* this is used to move all changes to the stage area including deletion

git commit -m “message”

* this is used to move the files from staging area to the local repo

git clone url (HTTPs URL of the repo)

* this is used to bring the remote repo to the local repo for the first time

git clone -b dev <https://github.com/phaninandigam/Assignment.git>

* this is used to clone a specific branch

git push

* this is used to push the changes from the local repo to the remote repo

syntax:

git push origin branchname

Example:

git push origin main

git log

* this will display the history of the repo

git log -2

* this will display the recent 2 commits

git log filename

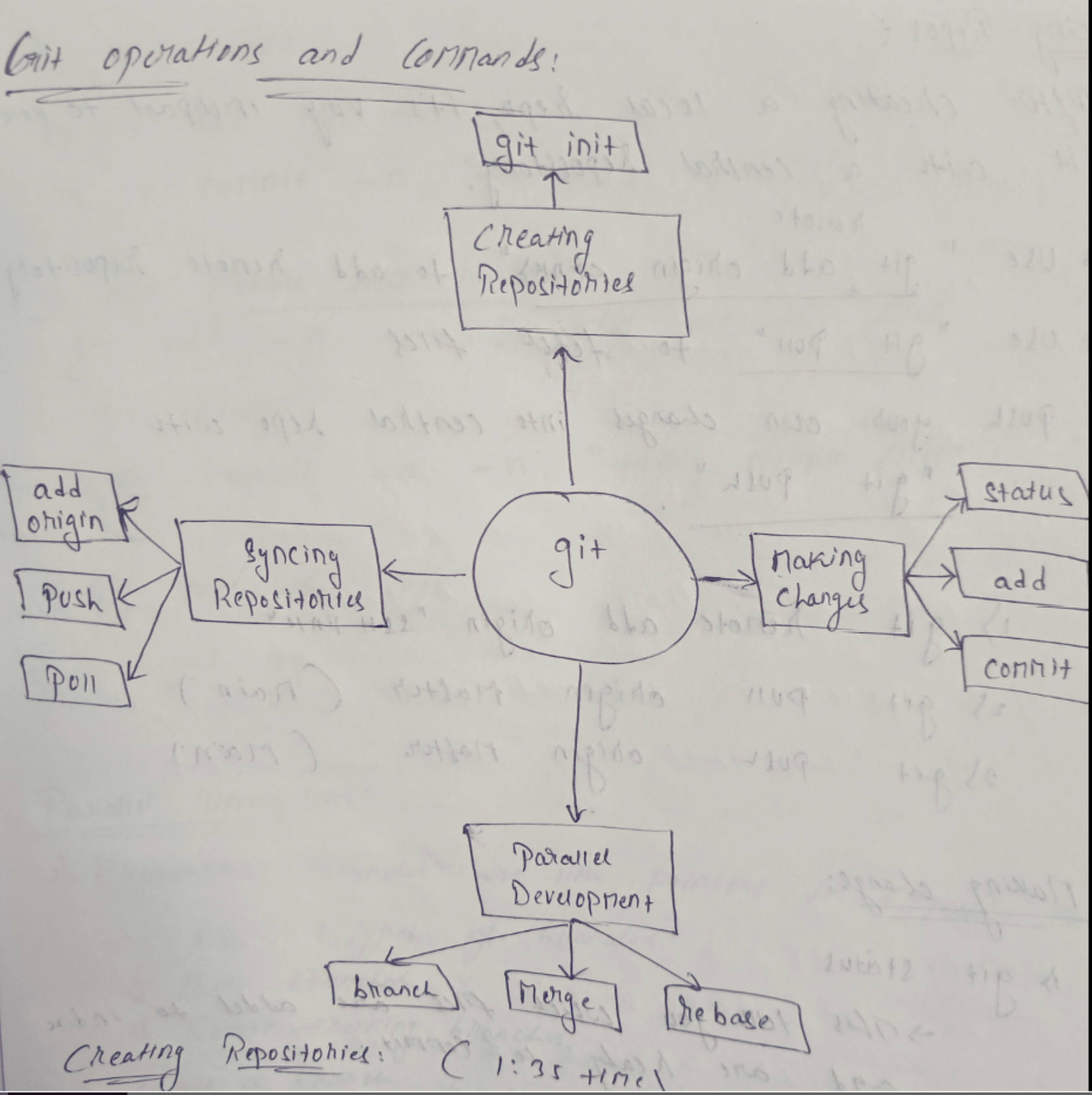
* this will display the commit history of the specified file
* using this command we can identify we has modified the file

git log commitid

* this will display the history from that commit
* let’s assume we might had done 6 commits and if we run the command with commit id as 4, then this will display the 4,3,2,1 commits

**how do I know how many files were modified under the specific commit?**

git show commitid



**Day 26: git**

**Branch**

* this is used for parallel development. If 2 people or 2 teams working on the same piece of code then we will create a new branch and we will integrate the changes by merging.

git branch

* this will list all the branches (Active)

git branch -a

* this will list all the branches (Active and inactive)

git branch branchname

* this will create new branch from checkout branch
* example:  
  git branch dev1

git checkout branchname

* this is used to switch the previous version of a code or also used to another branch
* this is used to checkout between branches
* example:  
  git checkout dev1

git checkout -b branchname

* this will create a new branch and switch to a new branch automatically
* example:  
  git checkout -b dev2

git branch -D branchname

* this is used to delete the branch locally
* example  
  got branch -D dev1 (Delete forcefully)

got branch -d dev1

git push origin –delete branchname

* this is used to delete the branch from the local and remote repo as well

git push origin –all

* this is used to push the changes from local repo to the remote repo from all the branches which contain changes

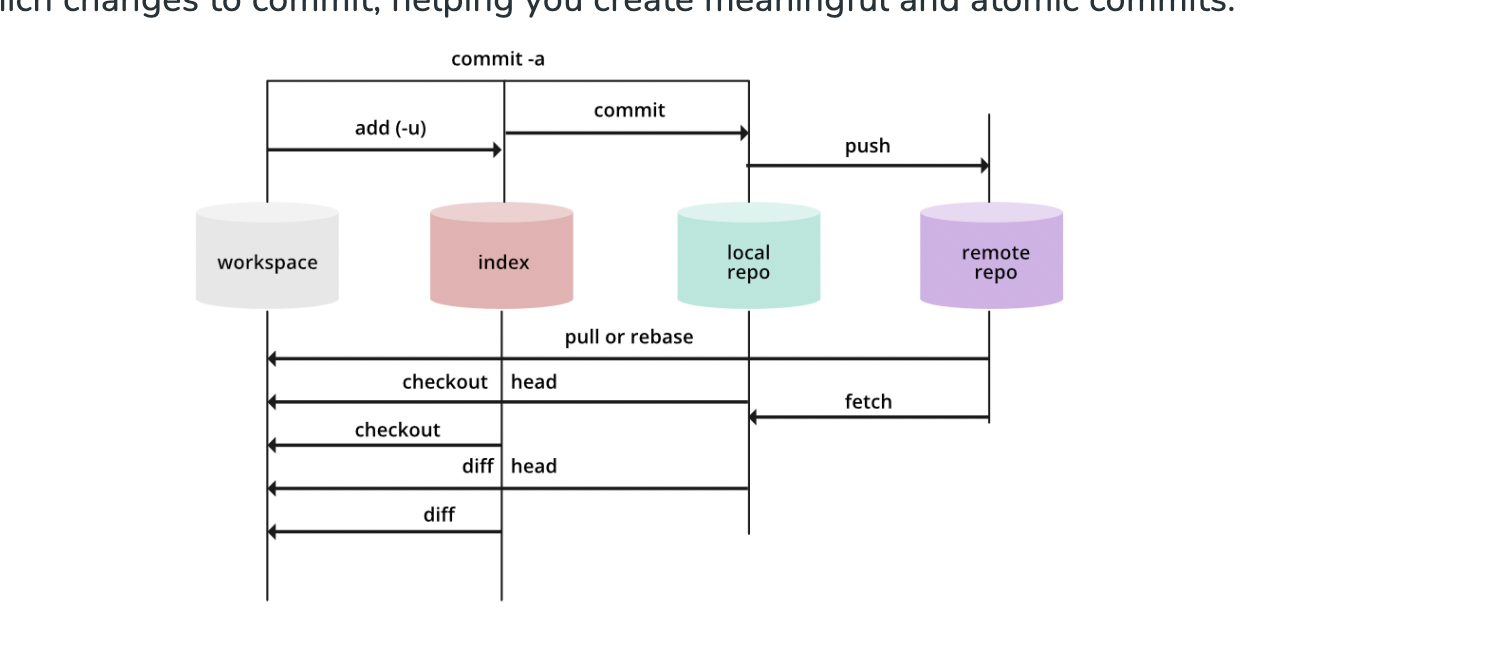
git merge

* this is used to merge the changes from one branch to another branch(Switched branch)
* example  
  git merge branchaname  
    
  If I need to merge the changes from test1 branch to the main branch  
  1. we need to switch to the main branch and integrate the changes  
  git checkout main  
  git merge test1

git pull

* this will bring the changes from the remote repo and merges to the local workspace automatically
* syntax  
  git pull origin branchname (This will pull the changes from the specific branch)  
  git pull (This will pull the changes from all the branches)

git index

* The Git index is a temporary staging area (also referred to as the "staging area" or "cache") used to prepare changes before committing them to the project history. When you perform actions such as git add , Git does not immediately commit the changes to your repository. Instead, it adds the changes to the index.
* <https://www.geeksforgeeks.org/git-index/>
* 

**Day 27: git**

**Tag:**

* it’s a name given to set of versions of files and directories.
* tag is easy to remember in future, it indicates the milestone of a project

git tag

* this will list the available tags

git tag tagname

* this is used to create a tag with updated version

git tag -d tagname

* this is used to delete the tag locally

git push origin –delete tagname

* this is used to delete the tag from the remote repo

git checkout tagname

* this is used to checkout the code from the specific tag.
* after checkout we need to create a new branch to save this code by using the below command
  + git switch -c branchname

git push origin tagname

* this will push the tag to the remote repo

git push –tags

* this will push all the tags to the remote repo

**Assignment:**

1. **create 2 files (file1 and file2) in main branch**
2. **create a release relae-1.0 from the main branch**
3. **create new branch dev from the main branch and add some content to the files**
4. **now create a release release-1.1 from dev branch**
5. **create one more file add some content to the dev branch**
6. **merge the changes from dev to main branch**
7. **create a release release-1.2 from the main branch**

git revert

* this is used to undo the commit changes
* history will be tracked
* syntax:  
  git revert commitid  
  it will revert the changes done in the mentioned commitID

git revert HEAD

* this will revert the last commit

git revert HEAD~1

* this will revert the last but 1 commit

**Difference between rebase and merge**

* merge will just integrate the changes from one branch to another.
* rebase is nothing but a merge, one branch will get added to the tip of another branch.
* we can squash multiple commits has a single commit and then we can merge
* while merging it doesn’t allow any git functions, until and unless git conflicts are resolved

git rebase branch

* this is used to merge the changes from one branch to another

git rebase -i HEAD~3

* this will squash, the last 3 commits has a single commit

**Day 28: git**

**Merge conflicts:**

* it will occur when same line of code is modified on two different branches or from 2 teams on same file & When we try to merge these changes of two branches, we get merge conflicts.

I don’t know whose changes should consider and which changes should consider to merge so I will contact developer who modified the codes on two different branches, and they will discuss and give the updated change.

I will put this new changes in the file and continue with the merge.

Using git log filename we can get developer who have modified the file

**How do you identify who has modified the file?**

git log filename

**Git Stash:** The git stash command takes your uncommitted changes saves them away for later use, and then reverts them from your working copy.

git stash : when am working on current branch if I get any critical bug which needs to be fixed in other branch, before switching to other branch I need to stash incomplete(half done) work on current branch.

Once I fixed the bug on another branch I need to switch back to the current branch and I will pop the stash entry and continue with the incomplete work.

This will avoid committing the changes on wrong branches.

**command:**

git stash

* this will add the code to the stash entry
* this will avoid committing the code on wrong other branches.

git stash: it will stash the incomplete work/ it will save the changes in stash entry. Before executing this need to execute git add filename or git add .

git stash pop : it will retrieve the last saved changes in the stash entries

git stash list : it will list the stash entries that we have saved

stash @{0} – last enrty

stash @{1} – last but one entry

git stash drop: it will remove the last stash entry from the list.(delete the recent stash entry and the files in the stash)

git stash clear: it will remove the all the stash entries

git stash pop stash @{1}

**Git reset:**

* this is used to undo the committed changes but history will not be tracked. The name itself indicates reset pointer HEAD to the previous commit.

Types of reset:

**Mixed**

* this is used to move the files from staging area back to the workspace

**Syntax:**

git reset --mixed commit id

**Soft**

* it will remove history and committed changes will be moved to the staging area from git repo

**Syntax:**

git reset --soft commit id

**Hard**

* there will be no clue that we have commit changes.
* it will remove the history and changes from everywhere (git repo, staging area and workspace)

**Syntax:**

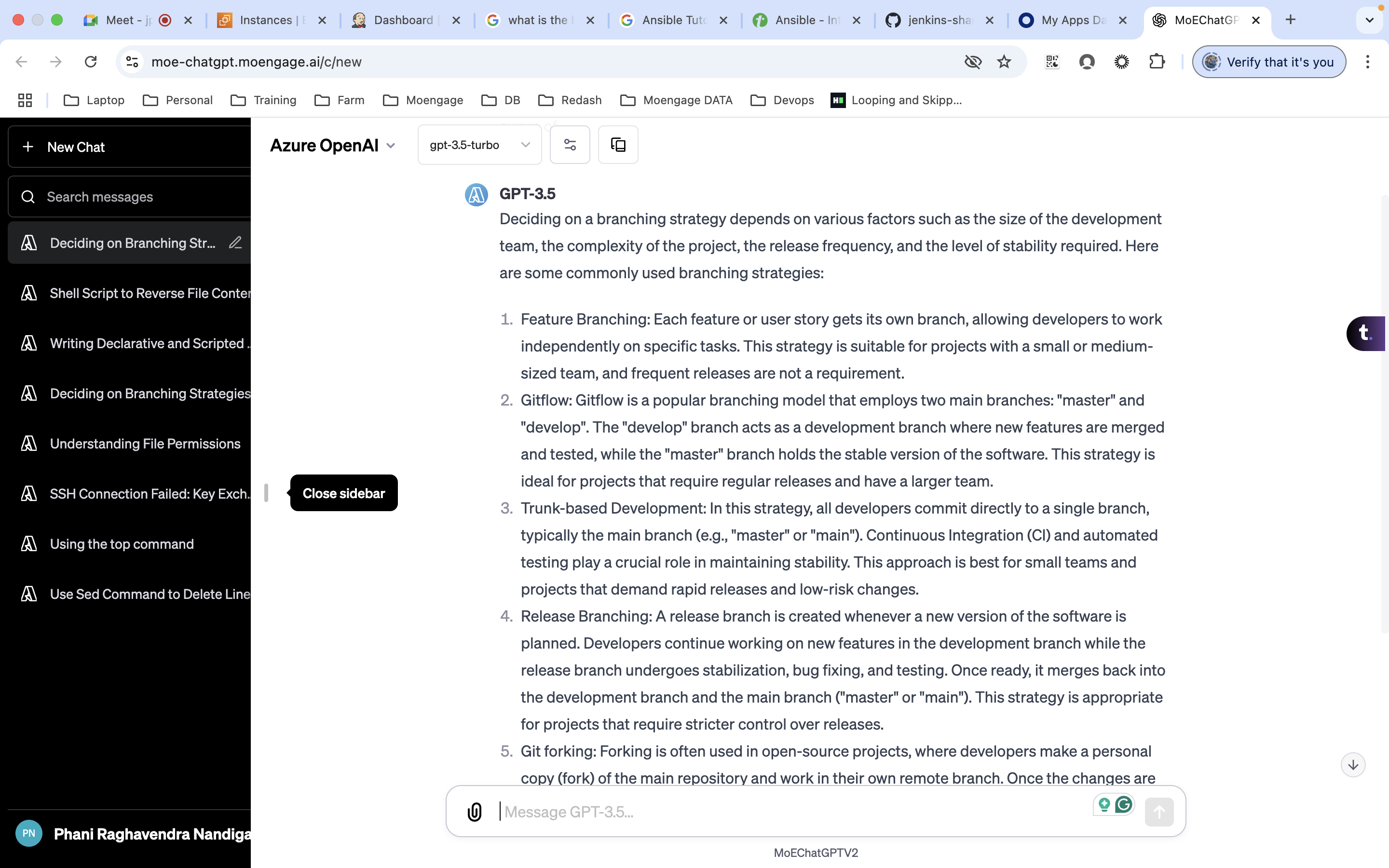
git reset --hard commit id

**Day 29: git**

Branching strategy:

* Branches can be created for multiple reason here we create branches for releases.
* We followed **GitFlow** as the branching strategy. Our Branching strategy has branches like , Feature branch , Dev branch , Master branch and Hotfix branches.
* **Feature branch:** developers will create their own feature branch and merge it to the dev branch.
* **Release branch**:used to maintain the releases.
* **Hot fix branch**:we use these branches to deliver critical fix to the production environment.
* **Master branch**:Would be the replica of our production environment , Its always a clean branch.
* Development will be going on dev branch once the code is ready for first release on dev branch, we create separate branch for first release, and we make release from release branch.
* Whatever the issues related to first release will be fixed on first release branch.
* Parallel development will be going on dev branch for second release.
* Once the code is ready for second release on dev branch, before we create release2 branch we merge the first release to the dev branch and then we create release2 branch for second release.
* Whatever the issues we have seen in first release will not be seen in second release.

How do you decide a branching strategy. What are the different branching strategies you are aware of?



**Difference between distributed and centralised version control tool(git vs svn(cvs))**

1. git is a distributed version control tool where the whole repo will be there in local repo.
2. if I want to go to the previous version of the code, I can check out in the local workspace itself
3. svn is centralized version control tool, only the latest version of code will be available in local workspace. Suppose if i want to checkout from previous version of the code, I need to checkout from remote repo
4. distributed version control tool has many advance features like reset, fetch, stash
5. in centralized version control tool, we don’t have direct commands for this.  
     
   git, gitlab, bitbucket – Distributes version control tool  
   svn, cvs - Centralized version control tools

**git fetch:**

* this will bring the changes from remote repo and store it in sperate branch in local repo. We can review the changes and merge if it’s required.
* syntax:  
  git fetch origin branchname

git switch -c branchname

Note: Git pull = git fetch + git merge

**How do you merge specific change or specific commit?**

git cherry-pick commitid

The cherry pick command can be helpful if you accidentally make a commit to the wrong branch. Cherry picking allows you to get those changes onto the correct branch without redoing any work.

detached HEAD mode

* when your git repo is not on any branch but rather on a specific commit. This happens when you directly checkout a commit or a tag. in this mode the user can still make changes and save the changes to a new branch using the command git switch -c branchname

**Types of repo:**

1. Bare repo  
   -- it acts as a remote repo where we can push and pull the code to this repo. we cannot run any git commands   
   syntax:  
   git init –bare
2. Non-bare repo  
   -- it’s a local repo where we edit project related files and we can run all git commands

.gitignore

* this is a file used in git to specify the files and directories to ignore by git. Files matching the patterns in .gitignore are excluded from being staged, committed or tracked to the repo

**Day 30: git**

git amend

* this is used to modify the recent commit by adding/modifying the files.
* we can also edit the commit message as well
* to add the file file1.txt to the previous commit, use the below commands  
  git add file1.txt  
  git commit –amend (It will allow the user to modify the commit message and save the changes to the previous commit)

git commit –ammend –no-edit (This will add the changes to the previous commit without editing the commit message)

git-blame: It is a Git command that annotates each line in a file with information about the last commit that modified the line. This information includes the commit hash, the author, the date of the change, and optionally, the line number. By using git-blame, you can track the origin of each line of code, making it easier to understand the history and context of changes.

git bisect  : It is a command in Git that helps find the commit that introduced a bug or caused a regression in your codebase. It uses a binary search algorithm to efficiently narrow down the range of commits to be examined.

git reflog : used to find the commit at the deleted branch's tip. git reflog shows a log of all changes to the repository’s branches