Installing git

Install git on windows

- Download git (https://git-scm.com/downloads)
- Run the setup and install git
- Open git bash and run command
  - Git –version

Install git on RHEL8

- To check if git is available or not
  - Rpm -qa | grep git
  - Which git
  - Whereis git
  - Sudo yum install git
  - Git –version

Install git on Ubuntu

- Apt-get update
- Apt-get install -y git-core
- Git –version

1. Git configuration

- Git config : Get and set configuration variables that control all facets of how Git looks and operates.
- Set the name:
  - $ git config --global user.name "User name"
- Set the email:
  - $ git config --global user.email "prashantshastri@moweb.com"
- Set the default editor:
  - $ git config --global core.editor “Vim”
- Check the setting:
  - $ git config -list
- allows Git to remember how you resolved conflicts in the past, and it can automatically apply the same resolution to future conflicts in the same file.(not used in shared repo).
  - Git config –global rerere.enabled true

Git alias

- Set up an alias for each command:
  - $ git config --global alias.co checkout
  - $ git config --global alias.br branch
  - $ git config --global alias.ci commit
  - $ git config --global alias.st status

2. Starting a project

- Git init : Create a local repository:

- $ git init

❖ Git clone : Make a local copy of the server repository.
- $ git clone <url>

3. Local changes

❖ Git add : Add a file to staging (Index) area:
- $ git add Filename

❖ Add all files of a repo to staging (Index) area:
- $ git add *
- $ git add -A

❖ Git commit : Record or snapshots the file permanently in the version history with a message.
- $ git commit -m " Commit Message"

4. Track changes

❖ Git diff : Track the changes that have not been staged:
- $ git diff

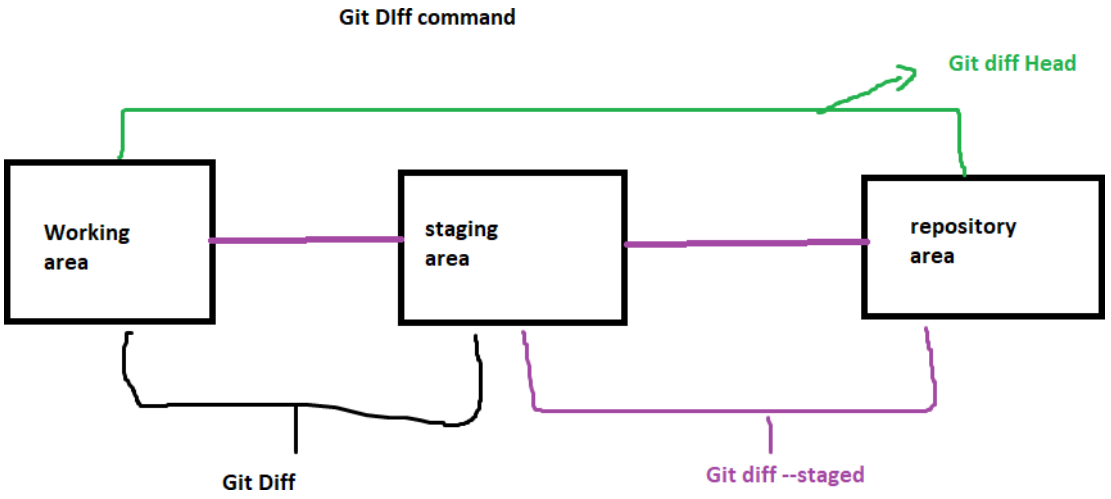
❖ Track the changes that have staged but not committed:
- $ git diff --staged

❖ Track the changes after committing a file:
- $ git diff HEAD

❖ Track the changes between two commits:
- $ git diff < branch 2>

❖ Git status : Display the state of the working directory and the staging area.
- $ git status

❖ Git show Shows objects:
- $ git show
- Git diff-tree -r commitid



5. Commit History

- Git log : Display the most recent commits and the status of the head:
  - $ git log
- Display the output as one commit per line:
  - $ git log -oneline
- Displays the files that have been modified:
  - $ git log -stat
  - Git log –raw
- Display the modified files with location:
  - $ git log -p
- Git blame : Display the modification on each line of a file:
  - $ git blame <file name>
- Git shortlog
  - Git shortlog -n
  - Git shortlog -s
  - Git shortlog -e
- Git log –author=”Prashant Shastri”
- Git log –grep”bug”
- Git log –pretty=format:”%cn%h%cd”
- Git log –merges –oneline
- Git log –no-merges
- list commits that are present in the feature branch but not in the master branch
  - $ git log feature..master

Git Checkout

- Moves from one branch to another
- Create a new branch if not existed and moves head to that branch
- Also not only branch shifts to particular commits hash = detached head
  - Git checkout branch name
  - Git checkout -b branchname
  - Git checkout -
    - Previous state
  - Git checkout Head~2
    - Behind 2 commit
  - Git checkout also used to discard changes in the file (revert)
    - **Git checkout head filename**
    - Git checkout – filename

Git switch

- Moves head from one branch to another
  - Git switch branch name
  - Git switch -c newbracnh name

-using switch command we r not able to move to commit while checkout does

- using checkout we are also able to revert the changes but using switch we are unable to do this type of advanced things.

6. Ignoring files

- .gitignore
- Specify intentionally untracked files that Git should ignore. Create .gitignore:
  - $ vim .gitignore -> wright filename
  - $ git ls-files -i --exclude-standard

7. Branching

- List Branch:
  - $ git branch
  - $ git branch --list
- Delete a Branch:
  - $ git branch -d <branch name>
- Delete a remote branch
  - Git push origin -delete <branch namye>
- Rename branch
  - Git branch -m <old name> <new name>
- create a new branch based on an existing branch, and allow you to change the name of the new branch in the same command.
  - Git branch -c oldname newname

Creating multiple branching

- For Variable in nameofbranch{1..3}; do git branch $variable; done;

Change the default branch

- Git config –list init.defaultbranch Prashant

GIt Head (detached head)

- When we are working we only checkout one branch at a time. Called as a head branch.
- Git makes note of this branch and stores it in .git/head as the reference for the path of the branch.
- Reference to the branch & the commit sha1.
- If the head points to a specific commit then it is called a detached head.
- 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.
-

- Git checkout commitid
    - Create further commit

Git stash

- Switch branches without committing the current branch.
- git stash temporary shelves (or stashes) changes you've made to your working copy so you can work on something else, and then come back and re-apply them later on.
- Sometimes you want to switch branches, but you are working on an incomplete part of your current project. You don't want to make a commit of half-done work.
- Store something safely in a hidden place. Git temporarily saves your data safely without commiting.
- Stash current work:
    - $ git stash
- Saving stashes with a message:
    - $ git stash save ""
        - Before stash save you have to add changes to git working tree "staging area"
- Check the stored stashes:
    - $ git stash list
- Re-apply the changes that you just stashed:
    - $ git stash apply <id>
- Track the stashes and their changes:
    - $ git stash show
- Re-apply the previous commits:(take recent stash)n
    - $ git stash pop
- Delete a most recent stash from the queue:
    - $ git stash drop <id>
- Delete all the available stashes at once:
    - $ git stash clear
- Stash work on a separate branch:
    - $ git stash branch <branch name> stashid

Git cherry pick

- If you want to apply a particular commit from one branch into another branch.
- Is mainly used if you don't want to merge the whole branch and you want some of the commits.
- It can cause duplicate commits.
- Is used for the bug fixes where you want to place that bugfix commit in all the versions branches.
- Avoid useless conflict
- It is also used when we accidentally make a commit in the wrong branch.
    - $ git cherry-pick <commit-id>

8. Merging

Git merge

- Merge the branches:
- This command will merge the changes from the source-branch into the current branch. The current branch is the branch you are currently on when you run the command.
    - $ git merge <source branch>
    - Git merge –squash sourcebranch

Git rebase

- Often used as an alternative of merging
- Rebasing a branch updates one branch with another by applying the commit of one branch on top of the commit of another branch.
- Is used to clean up our local commit history.
- Diff is merge preserve history while rebase doesn't preserve history
- Do not use rebase when the branch is public when it is shared to all the developers.
- Use : cleaning up your commits before sharing your branch.
    - Pulling changes from another branch without merge.
- Apply a sequence of commits from distinct branches into a final commit.
    - $ git rebase branch1
    - Git log –oneline –graph
- Interactive rebase
    - To combine multiple commit into 1 single commit
    - Git rebase -i branchname
        - See commits - decide which commit u want to take.
        - S = squash: use commit, but meld into previous commit

Modify or change last commit using amend command

- Changes is appended into the previous commit
- Git commit –amend

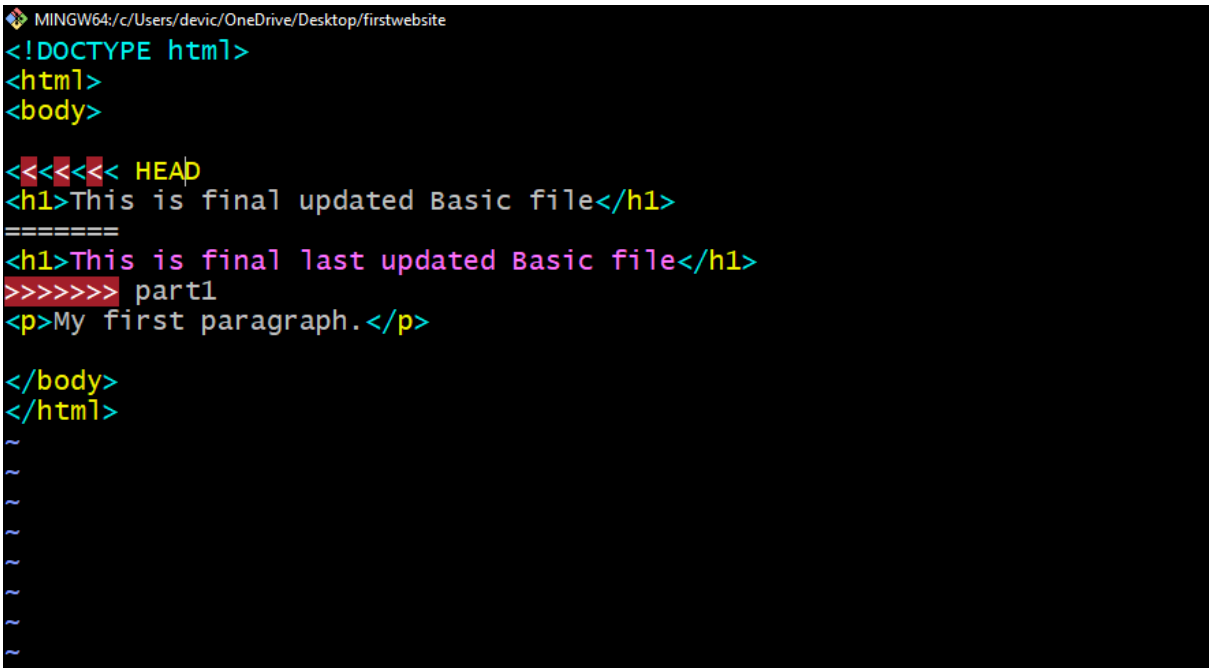
Git merge vs git rebase

Git merge safeguards history	Git rebase doesn't safeguard the history.
Git command that allows to take the independent lines of development created by git branch and integrate them into a single branch	Moves or combines a sequence of commits to a new base commit that provides an easy visualization of features branching workflow.
Git merge is comparatively easy	Git rebase is comparatively harder
Git merge forms a chain-like structure.	Git rebase forms a linear structure.
Git merge is preferable for large no. of	Git rebase is preferable for a small group of

people working on a project	people.
Command: git merge target source	Command : git rebase main

resolve conflict while merge

- If you have a same name file in 2 branches and you merge branches then conflict occurs. It is called a **merge conflict** .
- To resolve conflicts we open files manually on any editor, in my case i am using vim, and there we have to remove unnecessary content manually, once its done after then we can add to the staging area and then commit it.



- Remove unnecessary things accordingly and save it.
  - Git merge –abort
  - terminates the merger that you have just carried out and separated the two versions of your file
- How to resolve conflict steps
  - Identify the files responsible for the conflicts
  - Implemented desired changes to the files
  - Add the file using the git add command.
  - Commit the changes in the files using git commit command.

Rename

- Git mv oldname newname

9. Remote

- Git remote : Check the configuration of the remote server:
  - $ git remote -v

- Add a remote for the repository:
  - $ git remote add <name> <url>
- Fetch the data from the remote server:
  - $ git fetch
- Remove a remote connection from the repository:
  - $ git remote rm <name>
- Rename remote server:
  - $ git remote rename <old name> <new name>
- Show additional information about a particular remote:
  - $ git remote show
- Change remote:
  - $ git remote set-url <url>

Git origin master

- Push data to the remote server:
  - $ git push origin master
- Pull data from remote server:
  - $ git pull origin master

10. Pushing Updates

- Git push : Transfer the commits from your local repository to a remote server.
- Push data to the remote server:
  - $ git push origin master
- Force push data:
  - $ git push -f
- Delete a remote branch by push command:
  - $ git push origin -delete edited

11. Pulling updates

- Git pull : Pull the data from the server:
  - $ git pull origin master
- Pull a remote branch:
  - $ git pull
- Git fetch : Download branches and tags from one or more repositories.
- Fetch the remote repository:
  - $ git fetch< repository Url>
- Fetch a specific branch:
  - $ git fetch
- Fetch all the branches simultaneously:
  - $ git fetch -all
- Synchronize the local repository:
  - $ git fetch origin

12. Undo changes

- Discard changes in working area
  - Git restore filename
- Unstage file
  - Git restore –staged filename
- Move backward using git restore
  - Git restore –source head~2 index.html
- Git revert : Undo the changes
  - creates a new commit that undoes the changes made by a previous commit. It is a safe way to undo changes because it does not remove any commits from the history of your repository.
  - It preserve the history.
  - is a safer way to undo changes, especially for changes that have already been pushed to a shared repository.
  - $ git revert commithash
- Revert a particular commit:
  - $ git revert
- Git reset : Reset the changes:
  - It is used to move the branch
  - Reset moves the current branch and optionally copies the data from the repositories to the working or staging area.
  - used to undo changes by removing commits from the repository history.
  - it can permanently remove commits from the repository history.
  - If u already shared ur code to remote repo, try to avoid this.
  - $ git reset -hard
  - $ git reset –hard head~1
    - Moves the files both to working area and staging area
  - $ git reset -soft:
    - Does not move the file
  - $ git reset --mixed
    - Moves the files only to the stage area.(default option)

13. Removing files

- Git rm : Remove the files from the working tree and from the index:
  - $ git rm <file Name>
- Remove files from the Git But keep the files in your local repository:
  - $ git rm --cached

14. GitHub

- Github is the hosting platform for the git repository.
- Github allows us to host our git repository in the cloud.
- This provides that we can access the code from anywhere and also share the code with people around the world.

Git vs github

Git is a version control system that runs locally on your machine.	github is a service that host repositories in the cloud and makes it easier to collaborate with other people
You don’t need to register an account	You do need to sign up for an account to use github
You don’t need the internet to use it.	It's an online place to share work that is done using git.
Git is command line tool	Github is a GUI.
Git is a software	Github is service
Git is maintained by linux	Github is maintained by microsoft.

- There are so many tools that provide similar hosting and collaboration features.
    - Gitlab
    - Bitbucket
    - Gerrit
- Github was founded in 2008. It is the world's largest host of source code.
- Github offers its basic services for free.
- While it does offer paid team and enterprise tiers
- The basic free tier allows for unlimited public and private repos and unlimited collaborates and more.

Advantage

- Collaboration
- Contribute to an Open source project
- Exposure : sort of resume
- Stay up to date
- Track changes
- Integration option
- Bug tracking
- Code hosting
- Project & team management

Git Cloning

- To work with the remote repositories hosted in the github we need to get the local copy into the computer.
- For that we need a URL that we can tell git to clone the repository.

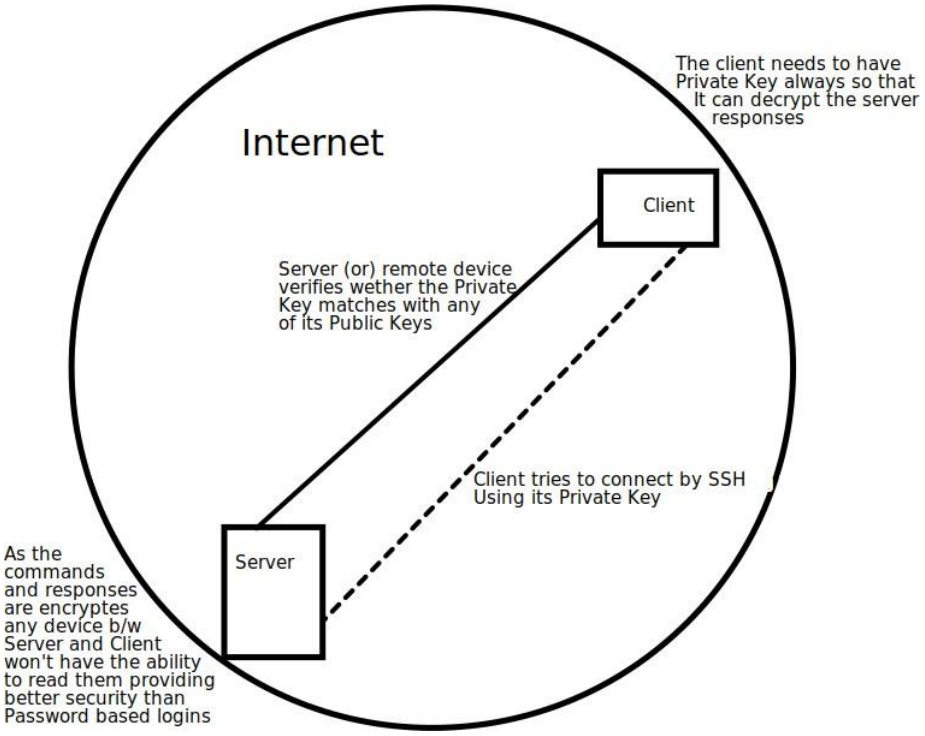
- Git clone gets the repository that is not present in your machine based on the url we provide.
- Git will retrieve all the files associated with the repository and will copy them to your local machine.
- In addition to that, git initialises a new repository on your machine giving you access to the fill history of the cloned project.
- To clone a repo
    - Git clone <url>
    - git clone --depth 1 <repository_url>

Github with SSH

- We can connect github using 2 methods
    - Using https : asking email and password every time
    - Using ssh : key based authentication
- When working with a github repository, you often need to identify yourself to github using your username and password.
- An SSH key is an alternative way to identify yourself that doesn't require you to enter your username and password every time.
- Using the SSH protocol, you can connect and authenticate to remote servers and services.

Practical

- To implement SSH keys in the system
- We need to create the SSH key pair using
    - Ssh-keygen -t ed25519 -C "email"
        - It is a digital signature
        - elliptic curve based public-key system commonly used for SSH authentication
- Next add your SSH key to ssh agent
- Ensure the ssh-agent is running
    - Eval `ssh-agent -s`
- Next add the ssh key to the ssh agent
    - Ssh-add ~/.ssh/id_ed25519
- We need to go to the github profile and add the ssh key pair in the setting:
- We can check that we are authenticate the github or not using
    - Ssh -t git@github.com



- Rsa : RSA works best in the cases of encryption and verification.
- Dsa : DSA works best in the cases of signing (digital) and decryption.

Create Repo in Github and connect to local machine

- How do I get my code in github?
  - Existing repo
    - Create a new repo on github
    - Connect your local repo (add a remote)
    - Push up your changes to github
  - Start from scratch
    - Create a new repo on github
    - Clone it down to your machine
    - Do some work locally
    - Push your changes to github
- We need to tell you about our remote repository present on github.

- Viewing remote
  - To view existing remotes for your repository, we can run
    - Git remote
    - Git remote -v
  - Display list of remotes.

- Adding a remote
  - Git remote add origin <url>

- Origin is the short name for the url
  - That means when ever i use the name origin, i am referring to the particular github url like an alias name
  - When we clone a github repo, the default remote name setup for us is called origin.

❖ Renaming the remote
- Git remote rename oldname newname

❖ Remove the remote
- Git remote remove name

❖ Push the change :Push data to the remote server:

- Git push origin branchname

❖ When we try to push the branch to github with command git push origin master
- You are actually creating the branch in the github and pushing the changes to that branch
- Can also push the changes in the master or any branch to the different branch in github
  - Git push remote localbranch:remotebranch

❖ -u option
- Allows us to set the upstream of the branch we are pushing.
- A link between the local branch to a branch in the github.
  - Git push -u origin master
- Sets the upstream of the local master branch so that it tracks the master branch on the origin repo.
- Now this set up to track the remote branch
  - So we have to just run command
  - Git push
- To set upstream
  - Git push –set-upstream origin new_branch

❖ Viewing remote branch
- Git branch -a
  - Main
  - origin/main

Checkout the remote tracking branches in the local git repo.

❖ By default when were we try to clone github repo, it clone only default branch
- If u want then, below command show remote branches
- Git branch -r
- Git checkout origin/branch1
- Git switch branch1
  - It check whether any local branch is preset if not, then whether there is remote branch name branch1 if yes then it try to create new branch1 and connection between branch to origin/branch

GIt fetch

- When you are working with other collaborators on a github repo.
- One of your teammates has pushed up the changes to the master branch, but my local repo doesn't know!
- Then how do I get those changes?
    - Git fetch and pull get those changes from the github repo to your local repo.
- Git fetching allows us to download changes from remote repo.
- But those changes will not be automatically integrated to our working dir.
- It just lets you see what others have been working on, without merging those changes into your local rep.
- Fetch is like a link - go and get the latest information from github, but don;t add it into my working dir.
    - Git fetch remote
- This command fetches branches and history from a specific remote repo. It only updates the remote tracking branches.
    - Git fetch remote branch
- After fetching the changes, I will have those changes on my machine.
- But if i want to see it then i have to do the checkout to origin/master.
    - Your local master branch will be untouched.
    - Git fetch
    - Git pull
    - Git switch branchname

Git pull

- used to retrieve changes from the remote repository.
- Unlike fetch, pull actually updates our HEAD branch with whatever changes are retrieved from the remote.
- go and download data from github and immediately update my local repo with those changes.
- Git pull = git fetch + git merge
    - Git fetch = update the remote tracking branch with the latest changes from the remote repository.
    - Git merge = update my current branch with whatever changes are on the remote tracking branch.
    - Git pull remote branch
- Git pull origin master would fetch the latest information from the origins master branch and merge those changes into our current branch.
- Pull can result in the merge conflicts
    - Sometimes you might have some work locally that is not on github and github has some commits.
    - When you pull down, there may be conflicts.

Diff between git fetch and git pull

Git fetch	Git pull
Gets changes from remote branches	Gets changes from remote branches
Updates the remote-tracking branches with new changes	Updates the current branch with new changes by merging them in
Does not merge changes onto your current head branch	Can result merge conflict
Safe to do at any time	Not recommended if you have uncommitted changes

Git Readme

- A readme file is used to communicate important information about a repo including
    - What the project does
    - How to run the project
    - Why it's noteworthy
    - Who maintain the project
- Md : mark down
- If you put a README in the root of your project
    - Github will recognize it and automatically display it in the repo's home page.
    - Readme is like an entry point to learn more about the project or application.
- Readme are markdown files, ending with the .md extension, markdown is convenient syntax to generate formatted text. It's easy to pick up!
- https://markdown-it.github.io/

Github Gists

- Github gists are the simple way to share the code snippets and useful fragments to others.
- Gists are much easier to create, but offer few features compared to normal git repo.
- Every gists is a git repo, which means it can be cloned.
- Gists will be associated with your account and see it in your list of gists when you navigate to your gist home page.
- Gists can be public or secret.
    - Public gists show up in discover, where people can browse new gists as they're created. Also searchable
    - Secret gists don't show up in discover and are not searchable. Secret gists aren't private.
        - If u send the URL of a secret gists to a friend, they'll be able to see it.
    - https://gist.github.com/starred

Github pages

- Are the public web pages that are hosted and published by github
- They allow you to create a website by simply pushing your code to github,
- Is a hosting service for serving static web pages.
- It does not support service side code like php, python, ruby or node.
- Only supports html, css and js code only.
- You get unlimited project sites in the github
- Each github repo can have a corresponding hosted website.
- Default url
    - http://username.github.com/repo-name
- One website for account or org.
    - But unlimited project repo.

pull request

- Pull requests are a feature built into products like github & bitbucket.
- They allow developers to alert team members to work that needs to be reviewed.
- They provide a mechanism to approve or reject the work on a given branch.
- They also help facilitate discussion and feedback on the specified commits
- Merging in feature branches
- At some point the work we did on the feature branch needed to be merged into the master branch.
- There are couple of option for how to do this
    1. Meger at will, without any sort of discussion with teammates. Just do whatever you want.
    2. Send an email or chat message or something to your team to discuss if the change should be merged in.
    3. Pull request.
- Pull request workflow
    - Do some work locally on a feature branch.
    - Push up the feature branch to github.
    - Open a pull request using the feature branch just pushed up to github.
    - Wait for the PR to be approved and merged. Start a discussion on the PR.

Branch protection rule

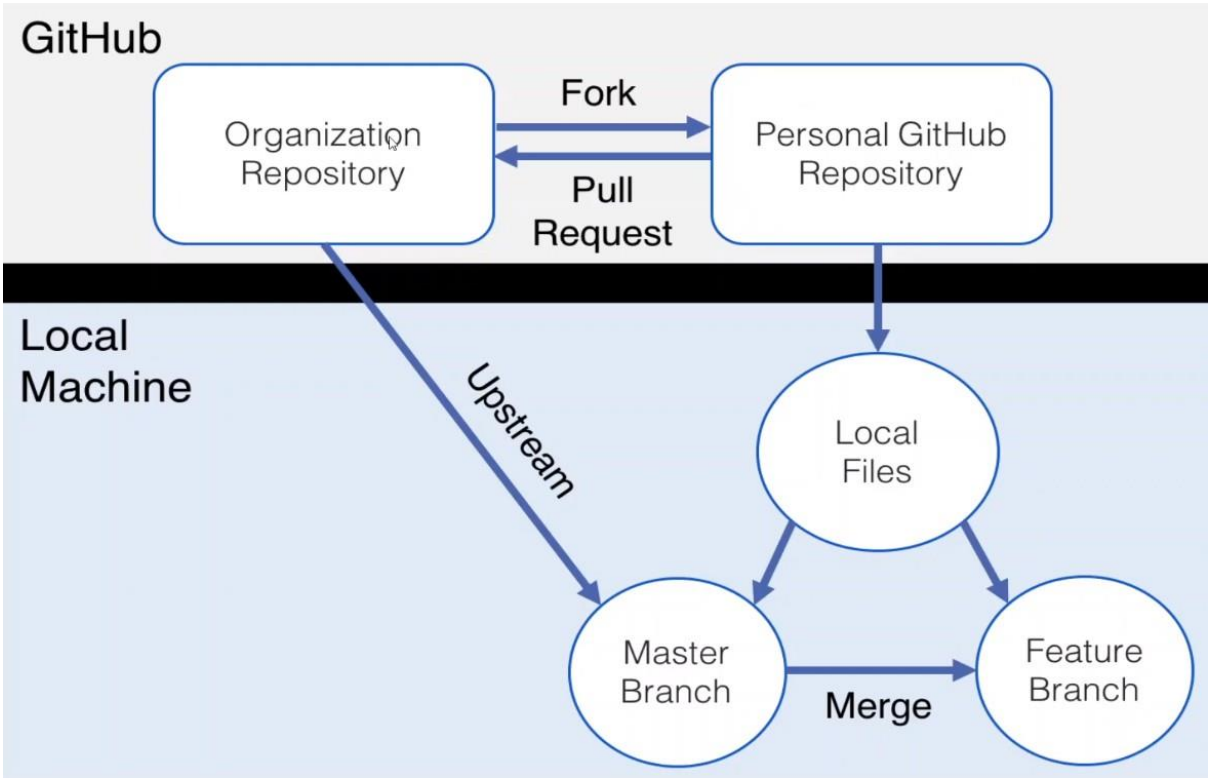
- Branch protection rules define whether collaborators can delete or force push to the branch and set requirements for any pushes to the branch, such as passing status checks or a linear commit history.
- Protect matching branches
    - Require a pull request before merging
    - When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
    - Require status check to pass before merging
    - Require conversion resolution before merging

- Required signed commits
- Require linear history
- Require deployment to success before merging
- Lock branch
- Do not allow bypassing above setting
- Allow force pushes
- Allow deletions

- Limit how many branches and tags can be updated in a single push
- rename repository name in github
- pull request : allow merge commit, allow squash merging, allow rebase merging.
- - automatically delete head branches when pull request are merged.
- - change visibility
- - transfer ownership
- - archive repository
- - delete repository.
- - collaborators. - add people.
- - change default branch (base branch)
- git remote set-head origin -a
- - by default restrictions of a branch protection rule don't apply to people with admin permission or custom rule with the "bypass branch protection" permission.

GIt fork

- When you are having a large open source project with lots of contributors.
- They employ this forking strategy or workflow where there might be a handful of actual maintainers.
- They cannot add thousands of people as direct contributors of collaborate
- This forking workflow enables anybody to try and make a contribution for the repo.
- There is no permissions needed. You can make your own copy. You try making changes and then you make a PR.
- Anybody can make a pull request
- Github and other similar tools allow us to create personal copies of other people's repo.
- We call those copies as a fork of the original
- When we for a repo, we basically asking github “make me my own copy of this repo please”
- It's not a git feature, the ability to fork is implemented by github.
- I can clone my fork and make changes, add features and break things without fear of disturbing the original repo.
- If I want to share my work, I can make a pull request from my fork to the original repo.
- Means that whole bunch of people can fork and can work on the project without actually having permissions to them.



- It allows a project manager to accept contributions from developers all around the world without having them as actual owner of the main project repo or worry about giving them all permission to push to repo.

Understanding semantic versioning for software.

- version consists of 3 numbers separated by periods. (4.2.1) - 4 = major, 2 = minor, 1 = patch
- typically the initial release (first release) for any project will be 1.0.0

Patch release(1.0.1)

- Patch releases normally do not contain new features or significant changes. They typically signify bug fixes and other changes that do not impact how the code is used.

Minor release(1.1.0)

- Signify that new features or functionality have been added, but the project is still backward compatible. No breaking changes. The new functionality is optional and should not force users to rewrite their own code.

Major release(2.0.0)

- Major releases signify significant changes that are no longer backwards compatible. Features may be removed or changed substantially.

Git tags

- We can tag particular commits so we can label commits by creating a tag, a reference to a moment in time.
- Are pointers that refer to particular points in git history. We can mark a particular moment in time with a tag.
- Tags are most often used to mark version releases in projects.
- Label for commit
- There are 2 type of tags
    - Lightweight tag
        - They are just name/label that point to a particular commit
        - Does not change - pointer to a commit
    - Annotated tags
        - Store extra meta data including the author name and email, the date and a tagging message (like a commit message)
        - Stored as full objects in the git database
        - It's generally recommended that you create annotate tags you have a full information

Viewing tags

- Git tag
- Git tag -l “*beta*”
- Git checkout tag
    - Go to commit, this puts us in a detached head.
- Git diff tag1 tag2

Creating tag

- Git tag tagname
- Git tag - a versionname
    - Enter description
    - Git show tagname
        - To view annotated tag

Pushing tag

- When we push the code tags are not automatically push to remote location to do we have to run below command
    - Git push origin tagname
    - Git push origin –tags

Git reflogs

- Short form for reference logs.
- These are just logs that git keeps us for as a record
- Git keeps a record of when the tips of branches and other references were updated in the repo.
- We can view and update these reflog using the git reflog command

    - Git only keeps reflog on your local activity. They are not shared with collaborators
    - Reflogs also expire. Git cleans out old entries after around 90 days. Though this can be configured.
    - Git reflog show
        - Show the log of a specific reference
        - Git reflog show main
        - Git reflog show head
        - Git reflog show head@{3}
            - Show 3 to end.
    - .git/logs/head
- Git checkout head~2
- Git reflog show master@{1.day.ago}
    - Git reflog show master@{1.week.ago}
    - Git reflog show master@{yesterday}
    - Git reflog diff master master@{1.week.ago}
        - Git reset master@{4} –hard

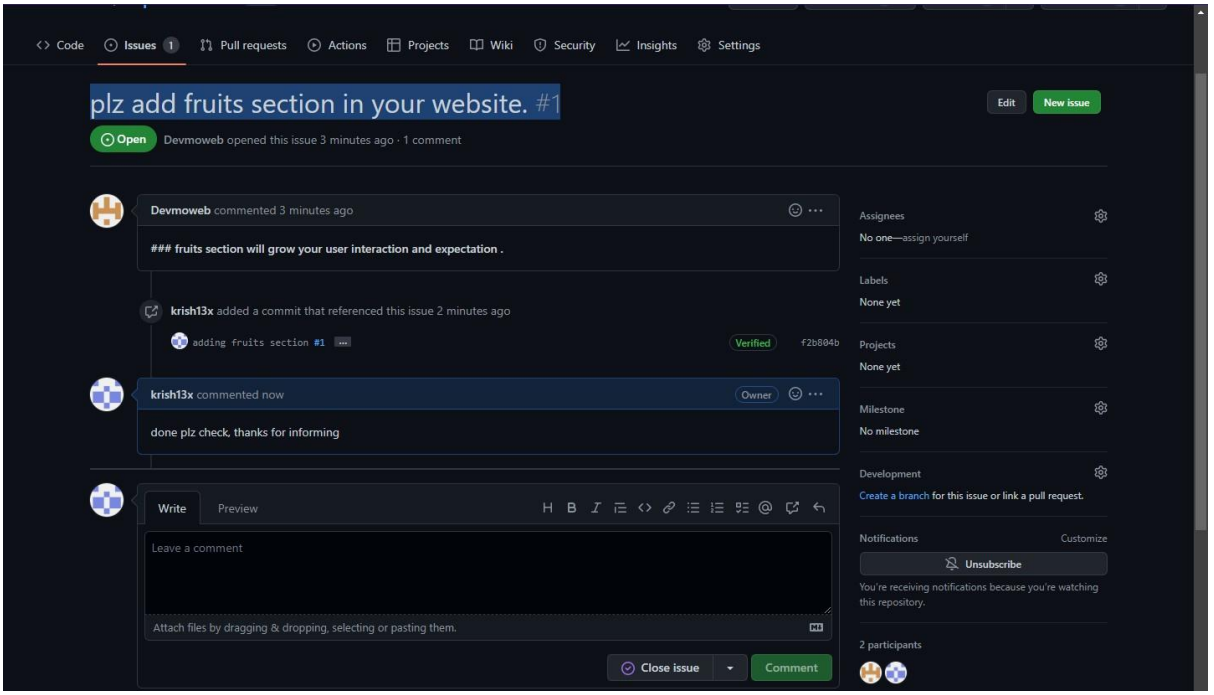
Git alias

- Git config –list –show-origin –global
- Modified giconfig
    - [alias]
        - S = status
- Git config –global alias.br “branch”

Github issues

- allows developers to track and manage bugs, feature requests, and other tasks related to their projects.
- Once an issue is created, it can be tracked, updated, and discussed by the project team.
- GitHub Issues is a valuable tool for project management and collaboration, helping developers and teams stay organized, efficient, and responsive to user feedback.
- Practical

-



Github import branch

- allows users to import a Git repository from a remote location into a new or existing repository on GitHub.
- Once the import process is complete, the repository and its entire commit history, branches, and tags will be available on GitHub.
- During the import process, you'll need to provide the URL for your existing Git repository, as well as any necessary authentication credentials.
- Once the import process is complete, your entire Git repository, including all of its branches, tags, and commit history, will be available on GitHub.
- Importing a repository can be useful in a variety of scenarios, such as when a user wants to move their code from one source control provider to another, or when a user wants to start using GitHub to manage their codebase after working with Git locally.

Git LFS

- Git LFS is designed to handle large files, which are files that are too big to be efficiently stored and managed using the default Git workflow. Examples of large files include media files (such as images and videos), database files, and binary files.
- Git LFS works by storing large files outside of the Git repository, and replacing them with "pointers" (small text files) that refer to the actual file in the external storage.
    - This means that Git can still track changes to the large files, but the actual files are not stored in the Git repository.
- Git LFS supports a range of external storage providers, including GitHub, Bitbucket, and Amazon S3.
    - It also includes a command-line interface and a set of Git hooks that help automate the process of managing large files.

- Overall, Git LFS is a useful tool for managing large files in Git repositories, and can help developers to more efficiently manage their code and other project files.

Git GC

- short for "Git Garbage Collector". helps to clean up unnecessary files and optimize the storage of a Git repository.
- Git creates new objects such as commits, branches, tags, and blobs every time you make changes to the repository.
    - Over time, these objects accumulate and can take up a significant amount of disk space.
    - helps to clean up these objects that are no longer being used or are no longer part of the repository's history.
- When you run "git gc", Git will analyze the repository's object database and remove any objects that are no longer being used.
- It will also pack the remaining objects into a more efficient format, which helps to reduce the size of the repository and improve Git's performance.
- By default, Git automatically runs "git gc" periodically in the background, so you don't need to run it manually.
- ./git/objet/
- ./git/object/pack/

Clone a specific branch

- Git clone -b branchname URL
- Git clone –single-branch -b branchname URL
- Git clone –branch=new1 URL

Bare repo

- Only contains the git metadata and history
- Identical to main repo
- Isolated from main repo
- It is a bare repo
- No working tree
- Changes can be seen in config file and refs folder
- Not possible to use push
- Not possible to use pull, status, rebase etc.
- Purpose is to relocate the main repo to another location.
- Git clone –bare url

Mirror repo

- A complete exact copy of the remote copy.
- Identical to main repo
- Non isolated from main repo
- It is bare repo itself
- No working tree

- Changes can be seen in config file and refs folder
- Possible to use push
- Not possible to use pull, status, rebase etc.
- Purpose is to create a secondary copy of the main repo.
- Git clone –mirror url

Git flow

- Is a branching model and workflow for git.
- Provides a set of guidelines and best practices for managing the branching and merging of a git repo.
- Easier to manage large projects with multiple contributors, and to maintain consistent and organized codebases.
- Define 2 primary long-running branches.
    - Master and develop
    - Master
        - Contains the stable and production-ready code
    - Develop
        - Contains the latest development code.
- Defines several types of short-lived branches that are used for feature development, bug fixing and release management.
- Advantage
    - Clear and well-defined branching and merging practices that make it easier to manage the codebase and keep it organized.
    - Better collaboration and coordination among team members
    - Improved release management.

Git hotfix

- To run hotfix we must installed git flow
    - sudo apt-get install git-flow
- Is a specific type of branch- used to quickly address critical issues or bugs in a release or main branch of a project.
- Designs to be applied and merged into the main branch as quickly as possible, without disrupting the regular development workflow or introducing new features.
- Practical - create a new branch - hotfix/login-issue - make necessary changes and fix the issues. - once complete - merge back into the main branch.
- Allow developers to quickly address critical issues and bugs without disrupting thre regular development cycle.
- Once finished, it will create a new tag and automatically delta locally hotfix branch and merge them to any other branches.
    - Git flow init
    - Git flow hotfix start ‘0.1.1”
    - Git branch -a
        - Fix the issue
    - Git add .
    - Git commit -m “fixed issue”

- Git flow hotfix finish “0.1.1”
- Git tag -l

Credential cache

- git config --global credential.helper cache
- git config --global credential.helper 'cache --timeout=3600'
- git clone -b branch_name --single-branch https://usr@github.com/usr/repo.git
- git branch --edit-description
- git config branch.feature1.pushRemote no_push
- you can set a remote repository to be read-only
  [branch "BRANCH_NAME"]
  [access]
  deny = USER_NAME

Giving access to collaborator

- No permission
  - Members will only be able to clone and pull public repositories. To give a member additional access, you’ll need to add them to teams or make them collaborators on individual repositories.
- Read
  - Members will be able to clone and pull all repositories.
- Write
  - Members will be able to clone, pull, and push all repositories.
- Admin
  - Members will be able to clone, pull, push, and add new collaborators to all repositories.

Repository role

- Roles are used to grant access and permissions for teams and members.

- Read
  - Read and clone repositories. Open and comment on issues and pull requests.
- Triage
  - Read permissions plus manage issues and pull requests.
- Write
  - Triage permissions plus read, clone and push to repositories.
- Maintain
  - Write permissions plus manage issues, pull requests and some repository settings.
- Admin
  - Full access to repositories including sensitive and destructive actions

How can I know if a branch has been already merged into master?

- git branch --merged master
    - lists branches merged into master
- git branch --merged
    - lists branches merged into HEAD (i.e. tip of current branch)
-
- git branch --no-merged
    - lists branches that have not been merged
-