**GIT- submitted by: Kalaiselvan P**

1. What is GIT?
2. What is GITHUB?
3. What is diff btw GIT vs GITHUB?
4. What is diff btw GIT vs SVN?
5. What is GIT Merge?
6. What is GIT Rebase?
7. Diff btw GIT Merge & GIT Rebase?
8. How will you resolve Merge Conflicts?
9. Diff Btw Fork,Clone & Branch?
10. Diff btw Pull & Fetch?
11. What is GIT Push?
12. What is GIT Clone?
13. How many ways we clone our repo?
14. What is GIT Stash?
15. What is .gitignore?
16. What is the difference between git stash apply vs git stash pop command?
17. What is GIT Cherry-pick?
18. What is Pull Request in GIT?
19. What are the branching strategy or git workflow?
20. What is GIT Squash?
21. How to revert a bad commit which is already pushed?
22. How to revert previous commit in git?
23. Diff btw GIT soft reset & hard reset?
24. Diff btw GIT Local vs GIT Remote?
25. What is GIT Fork?
26. Diff btw Revert & reset?
27. How to disable forking in github?
28. How will you secure your github account?
29. Command to Create Branch, Delete branch, Rename a branch
30. How will you switch from one branch to another branch
31. Command to restore delete branch
32. Command to merge feature to master/main branch

33. Command to check merged branches in master

34. What is the command to modify or change your commit?

35. How to restrict commit on github?

36. How to remove a file from git without removing it from your file system?

37. When do you use git rebase instead of git merge?

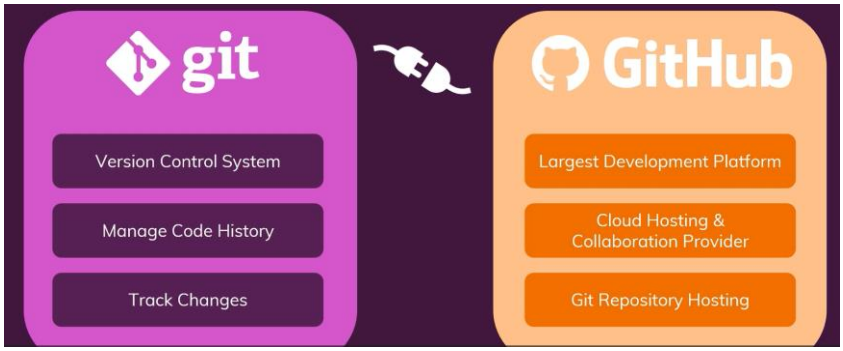
38. What is command to create/delete/rename a user in github?

39. What is GITHUB Actions and how it works?

40. Diff Btw GITHUB & Github Actions? Diff btw Github actions & Jenkins?

1. **What is GIT?**
   Git is a distributed version control system. It allows multiple developers to collaborate and work on the same code base. We can save multiple versions. We can switch between them easily to any point in time. Changes made by each developer can be merged. Git ensures code safety, team work, and simplifies the software development process.

2. **What is GITHUB?**
   - The git hub is largest development platform
   - It is a cloud hosting and collaboration provider
   - It is made for git repository hosting




**Code hosting platform**: we can store our code by creating repositories.
**Store, share, collaborate on projects:** developers can share their code or fork others code for collaboration
**Version control system:** with the help of the git, versions of the codes can be managed.
**Revert to previous versions:** git helps to move to previous versions of our code and to compare with any versions to track the change in the code over time
**Popular tool used by developers:** developers can store, share and collaborate

**Github additional functions:**
**Documentation:** GitHub provides a built-in documentation system.
**Issue tracking:** GitHub makes it easy to track bugs and feature requests**.**
**Continuous integration and continuous delivery (CI/CD):** GitHub can be integrated with CI/CD tools to automate the build, test, and deployment of your code.
**Security:** GitHub offers a variety of security features to help protect your code.

3. **What is diff btw GIT vs GITHUB?**

| Feature | Git | GitHub |
|---|---|---|
| **Type** | Software | Service |
| **Functionality** | Tracks changes in code | For Storing and hosting Git repositories |
| **Capabilities** | Version control, branching, merging, conflict resolution | Code hosting, issue tracking, documentation, CI/CD, security |
| **Installation** | Locally on your computer or cloud | Online |
| **Cost** | Free and open source | Free for public repositories, paid plans for private repositories |

4. **What is diff btw GIT vs SVN?**

| Feature | Git | SVN |
|---|---|---|
| **Model** | Distributed | Centralized |
| **Branches** | Unlimited | 1000 |
| **Merging** | Fast and efficient (sophisticated algorithm) | Can be slow and complex |
| **Conflict resolution** | Easy (conflict resolution tool) | Can be difficult |
| **Working copy** | Local | Remote |
| **Network required** | No | Yes |
| **Learning curve** | Steeper | Gentler |
| **Popularity** | More popular | Less popular |

5. **What is GIT Merge?**
   - Git merge is used to combine changes from one branch (source branch) to other (target branch) in its new commit.
   - Git merge is commonly used to integrate new features or bug fixes developed in feature branches back into the main branch (usually called the "master" or "main" branch).
   - The process resolves any conflicts that arise if the same lines of code were modified in both branches, ensuring a smooth integration of changes.

6. **What is GIT Rebase?**
   - Git Rebase takes the changes from the source branch and integrates them into the target branch, overwriting the history of the target branch.
   - This means that the history of the project is not preserved, but it can make the history of the target branch more linear and easier to follow.

7. **Diff btw GIT Merge & GIT Rebase?**

| Feature | Git Merge | Git Rebase |
|---|---|---|
| **How it works** | Creates a new commit that combines the changes from both branches | Takes the changes from the source branch and integrates them into the target branch, overwriting the history of the target branch |
| **History preservation** | Preserves the history of the project | Does not preserve the history of the project |
| **Linearity of history** | Can be less linear, due to merge commits | Can be more linear, as the history of the target branch is overwritten |

| | | |
|---|---|---|
| **Usefulness** | Good for merging changes from two branches that have not diverged too much | Good for merging changes from two branches that have diverged a lot |

8. **How will you resolve Merge Conflicts?**

We need three command lines to resolve merge conflicts

| Command line | purpose |
|---|---|
| git config --global merge.tool**vimdiff** | This command is to set global merge tool to vimdiff.<br>During merge conflict, vimdiff will allow us to **open the conflicting files to resolve** the conflict manually. |
| git config --global merge.conflictstyle diff3 | This is to set the conflict style to diff3.<br>when Git displays the conflicting files in vimdiff, it will show the **common ancestor of the two versions** of the file. This can be helpful for resolving conflicts. |
| git config --global mergetool.prompt false | This disables the prompt to open the merge tool. |

After opening the vimdiff vindows, “ctrl+ww” will allow us to shift between windows, selecting any required window and type “:wqa” for resolving it. In case of new modifications, go to the last “edit” window to modify and type “:wqa!”. Once resolving the conflicts, a new commit has to be done to incorporating the changes.

Then check the logs with “git log –oneline”.

9. **Diff Btw Fork,Clone & Branch?**

| **Feature** | **Fork** | **Clone** | **Branch** |
|---|---|---|---|
| Purpose | To create an independent copy of a repository under your GitHub account for independent work. | To create a local copy of a repository, usually from a remote repository, on your local machine. | To work on a separate line of development within the same repository without affecting the main development (e.g., master) |
| Location | GitHub servers. | Your local machine. | Your local machine. |
| Relation | Derived from the original repository but remains separate. | Direct copy of the repository, including all branches, and can be a fork or someone else's repository. | Derived from an existing branch (often the master branch) within the same repository. |
| Ownership | You own the forked repository. | You own the cloned repository on your local machine, which is not affected by the original repository. | Part of the same repository, and ownership is shared with other collaborators of the repository. |

| | | | |
|---|---|---|---|
| Syncing | Can be synced with the original repository through pull requests. | Can be synced with the original repository through fetch and pull operations. | Can be merged back into the main development branch after development is complete. |
| Collaboration | Can be used for contributing changes to the original repository. | Used for individual work or collaborative development with a team. | Allows multiple developers to work on different features or bug fixes simultaneously, maintaining isolation. |
| Branching | Creates a separate repository with its branches. | Creates a copy of the entire repository and its branches on your local machine. | Creates a new line of development within the repository, sharing the commit history with the master branch. |

10. Diff btw Pull & Fetch?

| Feature | Pull | Fetch |
|---|---|---|
| Operation | Combines fetching and merging into one command. | Fetches changes from a remote repository but doesn't automatically merge them into the local branch. |
| Command | git pull [remote] [branch] | git fetch [remote] |
| Usage | Used to update the local branch with the latest changes from the remote repository. | Used to bring the latest changes from the remote repository to the local repository without modifying the working copy. |
| Updates | Automatically updates the local branch by merging the remote changes. | Updates the remote-tracking branches (e.g., origin/master) to reflect the latest changes on the remote repository. |
| Conflicts | May result in merge conflicts if the local and remote branches have conflicting | Fetch operation itself doesn't cause merge conflicts since it doesn't modify the working directory or the local branch. |
| changes in the same lines of code. | However, if you perform subsequent merge or rebase operations after fetching, conflicts may arise. | |
| Safety | Since it automatically merges, it can potentially cause unintended changes. | Safer option since it only updates the remote-tracking branches, allowing you to review changes before merging. |
| Usage | Suitable when you want to quickly update your local branch with remote changes. | Useful when you want to see the changes made in the remote repository but decide later when to integrate them. |

**Pull**: Combines fetching and merging into one command. It automatically updates the local branch by merging the remote changes, potentially causing unintended changes and conflicts if the local and remote branches have conflicting changes.

**Fetch**: Fetches changes from a remote repository but doesn't automatically merge them into the local branch. It updates the remote-tracking branches to reflect the latest changes on the remote repository, allowing you to review changes before merging. Fetching is a safer option when you want to see the changes made in the remote repository but decide later when to integrate them.

**11. What is GIT Push?**

In Git, git push is used to send your local code changes to a remote repository, typically hosted on platforms like GitHub. It allows you to share your work with others and keep the remote repository up-to-date with your latest changes.

Syntax: git push <remote-name> <branch-name>

For example, git push origin main would push your local changes from the "main" branch to the remote repository named "origin."

**12. What is GIT Clone?**

Clone creates a local copy of a repository on your computer. This means that the clone is synchronized with the original repository, so any changes that are made to the original repository will be reflected in your clone. Clones are typically used to work on a project locally, or to create a backup of a repository.

Fork creates a remote copy of a repository on a hosting service like GitHub or Bitbucket. This means that the fork is independent of the original repository, so you can make changes to your fork without affecting the original repository. Forks are typically used to start working on a project that you don't have permission to contribute to, or to create a copy of a repository for testing purposes.

**13. How many ways we clone our repo?**

| Method | Description |
|---|---|
| Command line | The git clone command is the most common way to clone a Git repository. |
| GUI | There are many different GUIs that you can use to clone a Git repository. Some popular GUIs include GitHub Desktop, Sourcetree, and GitKraken. |
| Web browser | You can also clone a Git repository using a web browser. To do this, you will need to go to the repository's website and click on the "Clone" button. |
| USB drive | You can clone a repository from a USB drive by copying the repository's files to the USB drive. |
| Network share | You can clone a repository from a network share by copying the repository's files to the network share. |

**14. What is GIT Stash?**

Git stash is a command that temporarily saves your changes (both staged and unstaged) so you can work on something else, and then come back and re-apply them later on. This is a useful tool if you need to switch branches, but you're mid-way through a code change and aren't quite ready to commit.

```
Start by making some changes to your working directory.
echo "This is a new line." >> file.txt

# Save your changes to a stash.
git stash

# Switch to a different branch.
git checkout other-branch

# Make some changes to the other branch.
echo "This is a new line in the other branch." >> file.txt

# Switch back to the original branch.
git checkout master

# Apply the stashed changes.
git stash apply

# Check the file to see that the changes have been applied.
cat file.txt
```

**15. What is .gitignore?**

- A .gitignore file is a text file that tells Git which files and directories to ignore when you make a commit. This can be helpful if you have files in your project that you don't want to track with Git, such as temporary files, build artifacts, or configuration files.
- The .gitignore file is a powerful tool that can help you to keep your Git repository clean and organized. It can also help to improve the performance of your Git repository by reducing the number of files that Git has to track.

Here are some of the benefits of using .gitignore:

- It can help you to keep your Git repository clean and organized. If you have a lot of files in your project that you don't want to track with Git, you can use .gitignore to ignore them. This can help to keep your Git repository clean and organized.
- It can improve the performance of your Git repository. By reducing the number of files that Git has to track, you can improve the performance of your Git repository.
- It can help you to avoid merge conflicts. If you have files in your project that are ignored by .gitignore, they will not be included in your commits. This can help to avoid merge conflicts when you are working with other people on the same project.

**16. What is the difference between git stash apply vs git stash pop command?**

The git stash apply and git stash pop commands are both used to reapply stashed changes, but they have some key differences.

git stash apply leaves the stash entry in the stash list. This means that you can reapply the changes again later if you need to.
git stash pop removes the stash entry from the stash list. This means that you can't reapply the changes again unless you save them to a new stash entry.

In other words, git stash apply is a good option if you want to keep the stashed changes available for later use. git stash pop is a good option if you want to remove the stashed changes from your repository and start fresh.

| Command | Description |
|---|---|
| git stash apply | Reapplies the changes from the most recent stash entry. The stash entry is left in the stash list. |
| git stash pop | Reapplies the changes from the most recent stash entry and then removes the stash entry from the stash list. |

**17. What is GIT Cherry-pick?**

Git cherry-pick is a command that allows you to apply a specific commit from one branch to another. This is useful if you want to incorporate a particular change from a different branch into your current branch without bringing in all of the other changes from that branch.
Here are some of the benefits of using git cherry-pick:

It allows you to incorporate specific changes from other branches without having to merge the entire branch. This can be useful if you only want to incorporate a few specific changes from another branch.
It can be used to fix merge conflicts. If you have a merge conflict, you can use git cherry-pick to apply the changes from one branch to the other without having to resolve the merge conflict.
It can be used to backport changes. If you make a change to one branch and you want to apply that change to another branch, you can use git cherry-pick to backport the change.

**18. What is Pull Request in GIT?**

A pull request (PR) is a way to propose changes to a codebase that are reviewed by other developers before being merged into the main branch. Pull requests are a common way to collaborate on software projects, and they can help to ensure that changes are made in a consistent and well-tested way.

To create a pull request, you first need to create a branch to contain your changes. Once you have made your changes, you can then create a pull request by opening a new issue in the repository and selecting the "Pull Request" option. The pull request will then be reviewed by other developers, and if they approve the changes, the changes will be merged into the main branch.

**19. What are the branching strategy or git workflow?**

A branching strategy is a set of rules for how to create and manage branches in a Git repository. There are many different branching strategies, but some of the most common include:

**Gitflow**: This is a popular branching strategy that is used by many open source projects. It defines two main branches: the master branch and the develop branch. The master branch is the stable branch that contains the latest released code. The develop branch is the development branch that contains the latest changes that are not yet ready to be released.

**Forking Workflow**: This is a branching strategy that is often used for open source projects. It involves creating a fork of the main repository and then working on the fork independently. Once the changes are ready, they can be submitted as a pull request to the main repository.

**Trunk-Based Development**: This is a branching strategy that is becoming increasingly popular. It involves working on the master branch directly. Any changes that are not yet ready to be released are tagged.
Here are some of the benefits of using a branching strategy:

It can help to keep your codebase organized. By creating different branches for different purposes, you can keep your codebase organized and easy to manage.
It can help to prevent merge conflicts. By merging changes from different branches in a controlled way, you can help to prevent merge conflicts.
It can help to improve collaboration. By using a branching strategy, you can make it easier for other developers to collaborate on your project.

**20. What is GIT Squash?**

Git squash is a command that allows you to combine multiple commits into a single commit. This can be useful if you want to clean up your commit history or if you want to make a single commit that represents a larger change.

To use git squash, you need to specify the commits that you want to squash. You can do this by using the git log command to list the commits that you want to squash. Once you have the commit hashes, you can use the following command to squash the commits:

```
git squash <commit-hash1> <commit-hash2> ...
```

This will create a new commit that combines the changes from the three original commits.

Here are some of the benefits of using git squash:

It can help to clean up your commit history. If you have a lot of small commits, you can use git squash to combine them into a few larger commits. This can make your commit history easier to read and understand.

It can make it easier to collaborate with others. If you are working on a project with other developers, you can use git squash to combine your changes into a single commit. This can make it easier for other developers to understand your changes and to merge your changes into the main branch.

**21. How to revert a bad commit which is already pushed?**

There are a few ways to revert a bad Git commit that has already been pushed. Here are two of the most common methods:

**Method 1: Using the git revert command**

The git revert command allows you to create a new commit that reverses the changes made by an existing commit. To use this method, you will need to know the commit hash of the commit you want to revert. You can find the commit hash by looking at the commit history in your Git repository.

Once you have the commit hash, you can use the following command to revert the commit:

git revert <commit-hash>
For example, if the commit hash is 12345678, you would use the following command:

git revert 12345678
This will create a new commit that reverses the changes made by the commit with the hash 12345678.

**Method 2: Force-pushing a new commit**

Another way to revert a bad Git commit is to force-push a new commit that reverses the changes made by the bad commit. To do this, you will need to create a new commit that reverses the changes made by the bad commit. Once you have created the new commit, you can use the following command to force-push it to your remote repository:

git push --force
This will overwrite the existing commit with the new commit, which will effectively revert the bad commit.

Which method should I use?

The best method to use will depend on your specific situation. If you are only reverting a single commit, then the git revert command is the simplest way to do it. However, if you are reverting multiple commits, or if you need to revert a commit that has already been pushed to a remote repository, then you may need to use the force-push method.

It is important to note that reverting a commit can have unintended consequences. For example, if you revert a commit that has already been pushed to a remote repository, you may need to force-push the new commit to the remote repository. This can cause problems for other users who have already pulled the bad commit.

**22. How to revert previous commit in git?**

To revert a previous commit in Git:

i. Find the commit hash using git log.
ii. Use git revert <commit-hash> to create a new commit that undoes the changes.
iii. Save the revert commit message.
iv. Optionally, push the changes to the remote repository with git push.

**23. Difference between GIT soft reset & hard reset?**

| | **Soft Reset** | **Hard Reset** |
|---|---|---|
| **Purpose** | Move branch pointer to a previous commit and keep changes for recommitting or modification | Move branch pointer to a previous commit and discard changes after that point |
| **Effect on Working Directory** | Changes are preserved in the working directory | Discards changes in the working directory, resetting it to the commit state |
| **Effect on Staging Area (Index)** | Changes are preserved in the staging area (index) | Discards changes in the staging area (index) |
| **Commit History** | Creates a new commit to "undo" the most recent commit | Rewrites commit history by discarding commits after the reset point |
| **Caution Needed** | Less risky, generally safe to use | More aggressive, potentially data-destructive, use with caution |
| **Use Cases** | - To recommit changes after making modifications. <br> - To split a large commit into smaller ones. | - To completely undo changes made after a certain commit. <br> - To start fresh with a previous code state. |

**24. Differentiate between the GIT Local vs GIT Remote?**

| | Git Local | Git Remote |
|---|---|---|
| Definition | Refers to the repository on your computer where you work and make changes locally. | Refers to the repository hosted on a remote server, often used for collaboration and sharing code. |
| Location | Exists on your local computer or workstation. | Exists on a remote server, which could be a platform like GitHub, GitLab, Bitbucket, or a company server. |
| Access | Accessible without an internet connection. | Requires an internet connection to interact with the remote repository. |
| Operations | You can perform various Git operations like commit, branch, merge, etc., within the local repository. | You can push, fetch, pull, and clone repositories, collaborate with others, and share code. |
| Collaboration | Suitable for individual work and experimentation. Changes are not visible to others until pushed to the remote. | Facilitates collaborative development. Multiple developers can work on the same project, see each other's changes, and resolve conflicts. |
| Backup | Acts as a backup of your work until you push it to the remote repository. | Acts as a centralized location for code storage, reducing the risk of data loss. |
| Branches | You can create and work with branches locally. | Remote branches allow you to see and work with |

| | | |
|---|---|---|
| | | branches created by other developers. |
| Integration | Merges and other operations are done locally before pushing to the remote. | Developers integrate their changes through pull requests or direct pushes to the remote. |
| Pull | You can't pull changes from your own local repository. | You pull changes from the remote repository to update your local codebase. |
| Push | You push changes from your local repository to the remote to share them with others. | You can't push directly to someone else's repository; you need proper access rights. |
| Security | Generally considered safer as it is under your control. | Requires proper access controls and authentication to ensure security on the remote server. |

**25. What is GIT Fork?**

Git fork is a copy of a Git repository that is stored on a remote server. Forking allows you to make changes to the code without affecting the original repository. This is useful for collaboration, experimentation, and contribution.

Commands:

git fork: Creates a fork of a remote repository.
git clone: Clones a repository, either a fork or the original repository.
git pull: Pulls changes from a remote repository.
git push: Pushes changes to a remote repository.
git merge: Merges changes from one branch into another branch.

**26. Diff btw Revert & reset?**

The git revert and git reset commands are both used to undo changes in Git, but they do so in different ways.

**git revert** creates a new commit that reverses the changes made in the specified commit. This means that the new commit will undo the changes, but it will also create a new entry in the commit history.

**git reset** moves the HEAD pointer to the specified commit. This means that the changes made after the specified commit will be discarded, and the working directory will be reset to the state of the specified commit.

The main difference between git revert and git reset is that git revert does not change the commit history, while git reset does. This means that git revert is a safer option if you want to undo changes that have already been pushed to a remote repository.

| Feature | git revert | git reset |
|---|---|---|
| Changes the commit history | No | Yes |

| | | |
|---|---|---|
| Creates a new commit | Yes | No |
| Discards changes after the specified commit | No | Yes |
| Safe to use on public branches | Yes | No |
| Feature | git revert | git reset |

27. How to disable forking in github?

**Bing:**

To disable forking on a private repository in Github, you can follow these steps1:

I. Go to your Github repository and click on “Settings” in the top right corner.
II. Scroll down to the “Danger Zone” section and click on “Repository Settings”.
III. Under “Repository Settings”, find the “Forking” section and uncheck the “Allow forking” checkbox.
IV. Click on “Save” to apply the changes.

(or)

**Bard**:

**There are two ways to disable forking in GitHub:**

**For a specific repository:**

Go to the repository's settings page.

Under "Features," uncheck the "Allow forking" checkbox.

Click "Save."

**For an organization:**

Go to the organization's settings page.

In the "Access" section, click "Member privileges."

Under "Repository forking," select "Do not allow forking."

Click "Save."

**28. How will you secure your github account?**

Ways to secure your GitHub account:

**Use a strong password.** Your password should be at least 12 characters long and include a mix of uppercase and lowercase letters, numbers, and symbols.
**Enable two-factor authentication (2FA).** 2FA adds an extra layer of security to your account by requiring you to enter a code from your phone in addition to your password when you sign in.

**Be careful about what you share.** Don't share your GitHub username or password with anyone you don't trust.
**Keep your software up to date.** GitHub regularly releases security updates for its software. Make sure you install these updates as soon as they are available.
**Be careful about what you click on.** If you receive an email or message that asks you to click on a link, make sure you trust the sender before you click.
**Monitor your account activity.** You can use GitHub's security settings to monitor your account activity. This will help you to identify any unauthorized login attempts.

**Here are some additional tips:**
Use a password manager. A password manager can help you to create and store strong passwords for all of your online accounts.
Be aware of phishing attacks. Phishing attacks are emails or messages that are designed to trick you into revealing your personal information. Be careful about what information you share online, and never click on links in emails or messages from people you don't know.
Use a VPN. A VPN can help to protect your privacy and security when you are using public Wi-Fi.

29. **Command to Create Branch, Delete branch, Rename a branch**
The commands to create, delete, and rename a branch in Git:

**To create a branch:**

git branch <branch-name>
For example, to create a branch called new-feature, you would run the following command:
git branch new-feature

**To delete a branch:**

git branch -d <branch-name>
For example, to delete the branch called new-feature, you would run the following command:

git branch -d new-feature

**To rename a branch:**

git branch -m <old-branch-name> <new-branch-name>
For example, to rename the branch called new-feature to feature-1, you would run the following

command:

git branch -m new-feature feature-1

**Here** are some additional things to keep in mind:

- When you create a branch, it is not automatically checked out. You will need to use the git checkout command to switch to the new branch.
- When you delete a branch, it is not immediately removed from the repository. It is first marked as "deleted". You can still recover a deleted branch by using the git reflog command.

- When you rename a branch, the old branch is not immediately deleted. It is first renamed to a temporary name. The old branch is then deleted after a period of time.

**30. How will you switch from one branch to another branch**

To switch from one branch to another branch in Git, you can use the git checkout command. The syntax for the git checkout command is:

**git checkout <branch-name>**

For example, to switch to the branch called new-feature, you would run the following command:

**git checkout new-feature**

If the branch you want to switch to does not exist, you can use the -b option to create the branch and switch to it at the same time. The syntax for the git checkout command with the -b option is:

**git checkout -b <branch-name>**

For example, to create a new branch called feature-1 and switch to it, you would run the following command:

**git checkout -b feature-1**

Here are some additional things to keep in mind:

When you switch branches, your working directory will be updated to reflect the state of the new branch.
If you have uncommitted changes in your working directory, you will be prompted to save them before switching branches.
If you are currently on a branch that has not been pushed to a remote repository, you will be prompted to push the branch before switching.

**31. Command to restore delete branch**

There are two methods to restore the deleted branch
1.reflog method
2.stash method

1. reflog method:
   I. Make sure you are in the main branch of your repository.
   II. Run the git reflog command to list all the commits that have been made to your repository, including the deletion of branches.
   III. Find the SHA-1 hash of the commit that you want to checkout.
   IV. Run the git checkout -b <branch_name> <commit_hash> command to restore the branch.

**git reflog**

git reflog

This will list all the commits that have been made to your repository, including the deletion of branches. Find the SHA-1 hash of the commit that you want to checkout. In this example, the SHA-1 hash is 97f8a60.

git checkout -b feature/bugfix 97f8a60

This will restore the branch feature/bugfix to the state it was in at the commit with the SHA-1 hash of 97f8a60.

2. Using the stash:
   I. Make sure you are in the main branch of your repository.
   II. Run the git stash command to stash your uncommitted changes.
   III. Run the git checkout -b <branch_name> command to checkout the deleted branch.
   IV. Run the git stash pop command to restore your uncommitted changes.

   To restore the deleted branch feature/bugfix using the stash:

   ```
   git stash
   git checkout -b feature/bugfix
   git stash pop
   ```

   This will first stash your uncommitted changes, then checkout the deleted branch feature/bugfix. Finally, it will restore your uncommitted changes.

**32. Command to merge feature to master/main branch**

Goto the master and run:

```
git merge branch name
```

**33. Command to check merged branches in master**

Git branch --merged master

**34. What is the command to modify or change your commit?**

There are two main commands that you can use to modify or change your commit:

I. git commit --amend: This command allows you to modify the most recent commit. You can add new changes to the commit, change the commit message, or both.

II. git rebase -i <commit-hash>: This command allows you to interactively edit your commit history. You can change the commit message, squash commits together, or even remove commits altogether.

**35. How to restrict commit on github?**

1. Branch protection rules,
2. Pull Requests,
3. Code Owners to restrict commit on GitHub

**Branch protection rules:**

The steps on how to restrict commit on GitHub using branch protection rules:

I. Go to the repository that you want to restrict commit on.
II. Click on the Settings tab.
III. Click on the Branches tab.
IV. Click on the Protect this branch checkbox.
V. Select the users or teams that you want to allow to push commits to the branch.
VI. Specify any other requirements that you want commits to meet, such as requiring all commits to be reviewed by another developer before they can be merged.
VII. Click on the Save button.

Once you have saved the branch protection rules, only the users or teams that you have specified will be able to push commits to the branch.

**Pull Requests:**

The steps on how to restrict commit on GitHub using Pull Requests:

I. Create a new Pull Request.
II. In the Reviewers section, add the users or teams that you want to review the Pull Request.
III. In the Discussion section, add any comments that you want to give the reviewers.
IV. Click on the Send button.

Once you have sent the Pull Request, the reviewers will be able to review the changes and approve or reject the Pull Request. If the Pull Request is approved, the changes will be merged into the main branch.

**Using Code Owners:**

The steps on how to restrict commit on GitHub using Code Owners:

I. Go to the repository that you want to restrict commit on.
II. Click on the Settings tab.
III. Click on the Code Owners tab.
IV. Click on the Add a code owner button.
V. Enter the username or email address of the user or team that you want to add as a code owner.
VI. Click on the Add button.

Once you have added a code owner, that user or team will be able to review and approve changes to any files or directories that they are assigned to.

**36. How to remove a file from git without removing it from your file system?**

There are two ways to remove a file from Git without removing it from your file system:

I. Using the git rm --cached command: This command will remove the file from the Git index, but it will not remove the file from your file system.
II. Adding the file to the .gitignore file: This will tell Git to ignore the file, so it will not be tracked by Git

**i) Using the git rm --cached command:**

steps on how to remove a file from Git using the git rm --cached command:

I. Open a terminal window and navigate to the directory where your Git repository is located.
II. Run the following command:
III. git rm --cached <file_name>

(Replace <file_name> with the name of the file that you want to remove from Git.)

IV. Run the following command to verify that the file has been removed from Git:

git status.

V. The file should be listed as "Untracked".

**ii) Adding the file to the .gitignore file:**

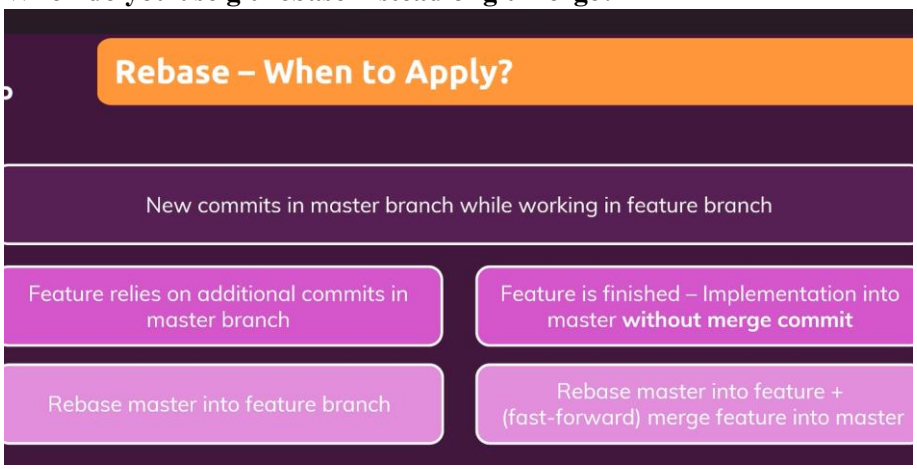
The steps on how to remove a file from Git by adding it to the .gitignore file:

I. Open the .gitignore file in a text editor.
II. Add the name of the file that you want to ignore to the file.
III. Save the file.
IV. Run the following command to update the Git index:
V. git add .
VI. Run the following command to commit the changes:

git commit -m "Removed <file_name> from Git"

The file will be ignored by Git and will not be tracked by Git.

**37. When do you use git rebase instead of git merge?**




I. Some advantages of git rebase are that it creates a linear and clean history, avoids merge conflicts and makes it easier to review changes.
II. Use git rebase if you want to clean up the history of your branches.
III. Use git rebase if you are working on a private branch and you do not need to preserve the original history.

**38. <u>What is command to create/delete/rename a user in github?</u>**

However, you can rename a user in Git by changing the author name and email address in the Git config file. This will only affect future commits, not past ones.

To rename a user in Git, follow these steps:

Open the Git config file. On Windows, it is typically located at C:\Users\<username>\.gitconfig. On macOS and Linux, it is typically located at ~/.gitconfig.
Find the [user] section in the file.
Change the name and email fields to the new user name and email address.
Save the file.
Once you have saved the file, any new commits you make will be attributed to the new user name and email address.

**39. What is GITHUB Actions and how it works?**

GitHub Actions is a **CI/CD (Continuous Integration/ Continuous Deployment) platform** that allows you to automate your build, test, and deployment process12. You can create **workflows** that run on GitHub servers when an event occurs, such as a pull request or a

push13. Workflows are composed of **steps**, which are code packages in Docker containers that perform specific tasks3.

GitHub Actions is designed to help simplify workflows with flexible automation and offer easy-to-use CI/CD capabilities built by developers for developers.

The benefits of GitHub Actions:

**Automation in GitHub Flow:**
Automate tasks and workflows directly within your repositories.
Create custom actions or use existing ones from the GitHub Marketplace.
Integrate your preferred tools seamlessly into your workflow.

**Hosted Virtual Machines with OS Options:**
Access hosted virtual machines for Ubuntu Linux, Windows, and macOS.
Build, test, and deploy code on your preferred operating system.
Run workflows simultaneously on multiple OS environments.

**Pre-written CI Templates:**
Integrate continuous integration (CI) seamlessly into your GitHub workflow.
Utilize templates created by developers for various programming languages and frameworks.
Customize and create your own CI workflows and continuous deployment (CD) workflows.

**Container and OS Testing:**
Support for Docker containers to manage and deploy applications.
Access to hosted instances of Ubuntu Linux, Windows, and macOS for testing.
Simplified automation of build and test workflows across different systems.

**Free for Public Repositories:**
GitHub Actions is available for free on all public repositories.
For private repositories, enjoy up to 2,000 minutes per month of hosted workflows for free.
Unlimited minutes for private repositories if you host your own GitHub Action server.
GitHub Actions streamlines your development process by offering automated workflows, versatile OS options, ready-to-use CI templates, container and OS testing capabilities, and cost-effective options for public and private repositories.

**40. Diff Btw GITHUB & Github Actions? Diff btw Github actions & Jenkins?**

| Feature | GitHub | GitHub Actions |
|---|---|---|

| | | |
|---|---|---|
| What is it? | A hosted version control system | A feature of GitHub that allows you to create custom workflows |
| Where can it be used? | GitHub | GitHub only |
| What can it do? | Track changes in source code, collaborate on projects, and host code repositories | Automate build, test, and deployment pipelines |
| Who uses it? | Developers, teams, and organizations | Developers, teams, and organizations |
| Pros | Free for public repositories, easy to use, well-integrated with GitHub | Powerful, flexible, customizable |
| Cons | Can be complex to set up, not as widely supported as other CI/CD tools | Not as widely used as other CI/CD tools |

| Feature | GitHub Actions | Jenkins |
|---|---|---|
| What is it? | A feature of GitHub that allows you to create custom workflows | An open-source automation server that can be used to build, test, and deploy software |
| Where can it be used? | GitHub | Anywhere |
| What can it do? | Automate build, test, and deployment pipelines | Automate build, test, and deployment pipelines |
| Who uses it? | Developers, teams, and organizations | Developers, teams, and organizations |
| Pros | Easy to use, well-integrated with GitHub, free for public repositories | Powerful, flexible, customizable, widely supported |
| Cons | Not as widely used as Jenkins, can be complex to set up | Can be difficult to manage, requires more infrastructure |

Here is a more detailed comparison of the two tools:

**Ease of use:** GitHub Actions is generally easier to use than Jenkins. GitHub Actions uses YAML files to define workflows, which are easier to read and write than Jenkins' Groovy syntax. GitHub Actions also has a built-in UI that makes it easy to visualize and manage your workflows.

**Flexibility:** Jenkins is more flexible than GitHub Actions. Jenkins can be used to automate a wider range of tasks, and it offers more customization options. However, this flexibility can also make Jenkins more complex to set up and manage.

**Cost:** GitHub Actions is free for public repositories, but there is a charge for private repositories. Jenkins is open-source, so it is free to use. However, you may need to pay for additional infrastructure, such as a dedicated server.

**Community:** GitHub Actions has a smaller community than Jenkins. However, the GitHub Actions community is growing rapidly, and there are a number of resources available to help you get started.