Project Management with Git		Semester	3
Course Code	BCS358C	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0: 0 : 2: 0	SEE Marks	50
Credits	01	Exam Marks	100
Examination type (SEE)	Practical		

Course objectives:

- .To familiar with basic command of Git
- To create and manage branches
- To understand how to collaborate and work with Remote Repositories
- To familiar with virion controlling commands

Sl.NO	Experiments
1	**Setting Up and Basic Commands** Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.
2	**Creating and Managing Branches** Create a new branch named "feature-branch." Switch to the "master" branch. Merge the "feature-branch" into "master."
3	**Creating and Managing Branches** Write the commands to stash your changes, switch branches, and then apply the stashed changes.
4	**Collaboration and Remote Repositories** Clone a remote Git repository to your local machine.
5	**Collaboration and Remote Repositories** Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.
6	**Collaboration and Remote Repositories** Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.
7	**Git Tags and Releases** Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.
8	**Advanced Git Operations**

	Write the command to cherry-pick a range of commits from "source-branch" to the current branch.
9	**Analysing and Changing Git History** Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?
10	**Analysing and Changing Git History** Write the command to list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31."
11	**Analysing and Changing Git History** Write the command to display the last five commits in the repository's history.
12	**Analysing and Changing Git History** Write the command to undo the changes introduced by the commit with the ID "abc123".

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

- Use the basics commands related to git repository
- Create and manage the branches
- Apply commands related to Collaboration and Remote Repositories
- Use the commands related to Git Tags, Releases and advanced git operations
- Analyse and change the git history

Program Outcomes

1. **Engineering Knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem Analysis**: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct Investigations of Complex Problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern Tool Usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The Engineer and Society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and Sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and Team Work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project Management and Finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-Long Learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes

PSO 1: Apply the skills of core computer science engineering, artificial intelligence, machine learning, deep learning to solve futuristic problems.

PSO 2: Demonstrate computer knowledge, practical competency and innovative ideas in computer science engineering, artificial intelligence and machine learning using machine tools and techniques.

Course Contents :

1. **Git Basics.**
2. **Git Installation**
3. **Git Basic Commands**
4. **Experiments**

1. **Setting Up and Basic Commands**
   Initialize a new Git repository in a directory. Create a new file and add it to the staging area and commit the changes with an appropriate commit message.

2. **Creating and Managing Branches**
   Create a new branch named "feature-branch." Switch to the "master" branch. Merge the"feature-branch" into "master."

3. **Creating and Managing Branches**
   Write the commands to stash your changes, switch branches, and then apply the stashedchanges.

4. **Collaboration and Remote Repositories**
   Clone a remote Git repository to your local machine.

5. **Collaboration and Remote Repositories**
   Fetch the latest changes from a remote repository and rebase your local branch onto theupdated remote branch.

6. **Collaboration and Remote Repositories**
   Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

7. **Git Tags and Releases**
   Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.
   /
8. **Advanced Git Operations**
   Write the command to cherry-pick a range of commits from "source-branch" to the current.

9. **Analysing and Changing Git History**
   Given a commit ID, how would you use Git to view the details of that specific commit,including the author, date, and commit message?

10. Analysing and Changing Git History

Write the command to list all commits made by the author "JohnDoe" between "2023-01-01"and "2023-12-31."

11. Analysing and Changing Git History

Write the command to display the last five commits in the repository's history.

12. Analysing and Changing Git History

Write the command to undo the changes introduced by the commit with the ID "abc123".

2. Git Installation

To install Git on your computer, you can follow the steps for your specific operating system:

1. Installing Git on Windows:

a. Using Git for Windows (Git Bash):

- Go to the official Git for Windows website: https://gitforwindows.org/
- Download the latest version of Git for Windows.
- Run the installer and follow the installation steps. You can choose the default settings for most options.

b. Using GitHub Desktop (Optional):

- If you prefer a graphical user interface (GUI) for Git, you can also install GitHub Desktop, which includes Git. Download it from https://desktop.github.com/ and follow the installation instructions.

2. Installing Git from Source (Advanced):

- If you prefer to compile Git from source, you can download the source code from the official Git website (https://git-scm.com/downloads) and follow the compilation instructions provided there. This is usually only necessary for advanced users.

After installation, you can open a terminal or command prompt and verify that Git is correctly installed by running the following command:

```
$ git --version
```

If Git is installed successfully, you will see the Git version displayed in the terminal. You can now start using Git for version control and collaborate on software development projects.

How to Configure the Git?

Configuring Git involves setting up your identity (your name and email), customizing Git options, and configuring your remote repositories. Git has three levels of configuration: system, global, and repository-specific. Here's how you can configure Git at each level:

1. System Configuration:

- System-level configuration affects all users on your computer. It is typically used for site-specific configurations and is stored in the /etc/gitconfig file.

To set system-level configuration, you can use the git config command with the --system flag (usually requires administrator privileges). For example:

```
$ git config --system user.name "Your Name"
$ git config --system user.email "your.email@example.com"
```

1. Global Configuration:

- Global configuration is specific to your user account and applies to all Git repositories on your computer. This is where you usually set your name and email.

To set global configuration, you can use the git config command with the --global flag. For example:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "your.email@example.com"
```

You can also view your global Git configuration by using:

```
$ git config --global --list
```

3. Git Commands List

Git is a popular version control system used for tracking changes in software development projects. Here's a list of common Git commands along with brief explanations:

1. **git init**: Initializes a new Git repository in the current directory.
2. **git clone <repository URL>**: Creates a copy of a remote repository on your local machine.
3. **git add <file>**: Stages a file to be committed, marking it for tracking in the next commit.
4. **git commit -m "message"**: Records the changes you've staged with a descriptive commit message.
5. **git status**: Shows the status of your working directory and the files that have been modified or staged.
6. **git log**: Displays a log of all previous commits, including commit hashes, authors, dates, and commit messages.
7. **git diff**: Shows the differences between the working directory and the last committed version.
8. **git branch**: Lists all branches in the repository and highlights the currently checked-out branch.
9. **git branch <branchname>**: Creates a new branch with the specified name.
10. **git checkout <branchname>**: Switches to a different branch.
11. **git merge <branchname>**: Merges changes from the specified branch into the currently checked-out branch.
12. **git pull**: Fetches changes from a remote repository and merges them into the current branch.
13. **git push**: Pushes your local commits to a remote repository.
14. **git remote**: Lists the remote repositories that your local repository is connected to.
15. **git fetch**: Retrieves changes from a remote repository without merging them.
16. **git reset <file>**: Unstages a file that was previously staged for commit.
17. **git reset --hard <commit>**: Resets the branch to a specific commit, discarding all changes after that commit.
18. **git stash**: Temporarily saves your changes to a "stash" so you can switch branches without committing or losing your work.
19. **git tag**: Lists and manages tags (usually used for marking specific points in history, like releases).
20. **git blame <file>**: Shows who made each change to a file and when.
21. **git rm <file>**: Removes a file from both your working directory and the Git repository.
22. **git mv <oldfile> <newfile>**: Renames a file and stages the change.

These are some of the most com

mon Git commands, but Git offers a wide range of features and options for more advanced

usage. You can use git --help followed by the command nameto get more information about any specific command, e.g., git help commit.

Experiments On

Project Management with Git

(As Per VTU Syllabus)

Experiment 1.

Setting Up and Basic Commands:

Initialize a new Git repository in a directory. Create a new file and add it to the staging areaand commit the changes with an appropriate commit message.

Solution:

To initialize a new Git repository in a directory, create a new file, add it to the staging area, and commit the changes with an appropriate commit message, follow these steps:

1. Open your terminal and navigate to the directory where you want to create the Git repository.
2. Initialize a new Git repository in that directory:

```
$ git init
```

3. Create a new file in the directory. For example, let's create a file named "my_file.txt." You can use any text editor or command-line tools to create the file.
4. Add the newly created file to the staging area. Replace "my_file.txt" with the actual name of your file:

```
$ git add my_file.txt
```

This command stages the file for the upcoming commit.

5. Commit the changes with an appropriate commit message. Replace "Your commit message here" with a meaningful description of your changes:

```
$ git commit -m "Your commit message here"
```

Your commit message should briefly describe the purpose or nature of the changes you made. For example:

```
$ git commit -m "Add a new file called my_file.txt"
```

After these steps, your changes will be committed to the Git repository with the provided commit message. You now have a version of the repository with the new file and its history stored in Git.

Experiment 2.

Creating and Managing Branches:

Create a new branch named "feature-branch." Switch to the "master" branch. Merge the"feature-branch" into "master."

Solution:

To create a new branch named "feature-branch," switch to the "master" branch, and merge the "feature-branch" into "master" in Git, follow these steps:

1. Make sure you are in the "master" branch by switching to it:

$ git checkout master

2. Create a new branch named "feature-branch" and switch to it:

$ git checkout -b feature-branch

This command will create a new branch called "feature-branch" and switch to it.

3. Make your changes in the "feature-branch" by adding, modifying, or deleting files as needed.
4. Stage and commit your changes in the "feature-branch":

$ git add .
$ git commit -m "Your commit message for feature-branch"

Replace "Your commit message for feature-branch" with a descriptive commit message for the changes you made in the "feature-branch."

5. Switch back to the "master" branch:

$ git checkout master

6. Merge the "feature-branch" into the "master" branch:

$ git merge feature-branch

This command will incorporate the changes from the "feature-branch" into the "master" branch.

Now, your changes from the "feature-branch" have been merged into the "master" branch. Your project's history will reflect the changes made in both branches

Experiment 3.

Creating and Managing Branches:

Write the commands to stash your changes, switch branches, and then apply the stashedchanges.

Solution:

To stash your changes, switch branches, and then apply the stashed changes in Git, you can use the following commands:

1. Stash your changes:

$ git stash save "Your stash message"

This command will save your changes in a stash, which acts like a temporary storage for changes that are not ready to be committed.

2. Switch to the desired branch:

$ git checkout target-branch

Replace "target-branch" with the name of the branch you want to switch to.

3. Apply the stashed changes:

$ git stash apply

This command will apply the most recent stash to your current working branch. If you have multiple stashes, you can specify a stash by name or reference (e.g., git stash apply stash@{2}) if needed.

If you want to remove the stash after applying it, you can use git stash pop instead of git stash apply.

Remember to replace "Your stash message" and "target-branch" with the actual message you want for your stash and the name of the branch you want to switch to.

Experiment 4.

Collaboration and Remote Repositories:

Clone a remote Git repository to your local machine.

Solution:

To clone a remote Git repository to your local machine, follow these steps:

1. Open your terminal or command prompt.
2. Navigate to the directory where you want to clone the remote Git repository. You can use the cd command to change your working directory.
3. Use the git clone command to clone the remote repository. Replace <repository_url> with the URL of the remote Git repository you want to clone. For example, if you were cloning a repository from GitHub, the URL might look like this:

$ git clone <repository_url>

Here's a full example:

$ git clone https://github.com/username/repo-name.git

Replace https://github.com/username/repo-name.git with the actual URL of the repository you want to clone.

4. Git will clone the repository to your local machine. Once the process is complete, you will have a local copy of the remote repository in your chosen directory.

You can now work with the cloned repository on your local machine, make changes, and push those changes back to the remote repository as needed.

Experiment 5.

Collaboration and Remote Repositories:

Fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch.

Solution:

To fetch the latest changes from a remote repository and rebase your local branch onto the updated remote branch in Git, follow these steps:

1. Open your terminal or command prompt.
2. Make sure you are in the local branch that you want to rebase. You can switch to the branch using the following command, replacing <branch-name> with your actual branch name:

$ git checkout <branch-name>

3. Fetch the latest changes from the remote repository. This will update your local repository with the changes from the remote without merging them into your local branch:

$ git fetch origin

Here, origin is the default name for the remote repository. If you have multiple remotes, replace origin with the name of the specific remote you want to fetch from.

4. Once you have fetched the latest changes, rebase your local branch onto the updated remote branch:

$ git rebase origin/<branch-name>

Replace <branch-name> with the name of the remote branch you want to rebase onto. This command will reapply your local commits on top of the latest changes from the remote branch, effectively incorporating the remote changes into your branch history.

5. Resolve any conflicts that may arise during the rebase process. Git will stop and notify you if there are conflicts that need to be resolved. Use a text editor to edit the conflicting files, save the changes, and then continue the rebase with:

$ git rebase --continue

6. After resolving any conflicts and completing the rebase, you have successfully updated your local branch with the latest changes from the remote branch.
7. If you want to push your rebased changes to the remote repository, use the git push command. However, be cautious when pushing to a shared remote branch, as it can potentially overwrite other developers' changes:

$ git push origin <branch-name>

Replace <branch-name> with the name of your local branch. By following these steps, you can keep your local branch up to date with the latest changes from the remote repository and maintain a clean and linear history through rebasing.

Experiment 6.

Collaboration and Remote Repositories:

Write the command to merge "feature-branch" into "master" while providing a custom commit message for the merge.

Solution:

To merge the "feature-branch" into "master" in Git while providing a custom commit message for the merge, you can use the following command:

```
$ git checkout master
$ git merge feature-branch -m "Your custom commit message here"
```

Replace "Your custom commit message here" with a meaningful and descriptive commit message for the merge. This message will be associated with the merge commit that is created when you merge "feature-branch" into "master."

Experiment 7**.**

Git Tags and Releases:

Write the command to create a lightweight Git tag named "v1.0" for a commit in your local repository.

Solution:

To create a lightweight Git tag named "v1.0" for a commit in your local repository, you can use the following command:

$ git tag v1.0

This command will create a lightweight tag called "v1.0" for the most recent commit in your current branch. If you want to tag a specific commit other than the most recent one, you can specify the commit's SHA-1 hash after the tag name. For example:

$ git tag v1.0 <commit-SHA>

Replace <commit-SHA> with the actual SHA-1 hash of the commit you want to tag.

Experiment 8.

Advanced Git Operations:

Write the command to cherry-pick a range of commits from "source-branch" to the current branch.

Solution:

To cherry-pick a range of commits from "source-branch" to the current branch, you can use the following command:

$ git cherry-pick <start-commit>^..<end-commit>

Replace <start-commit> with the commit at the beginning of the range, and <end-commit> with the commit at the end of the range. The ^ symbol is used to exclude the <start-commit> itself and include all commits after it up to and including <end-commit>. This will apply the changes from the specified range of commits to your current branch.

For example, if you want to cherry-pick a range of commits from "source-branch" starting from commit ABC123 and ending at commit DEF456, you would use:

$ git cherry-pick ABC123^..DEF456

Make sure you are on the branch where you want to apply these changes before running the cherry-pick command.

Experiment 9.

Analysing and Changing Git History:

Given a commit ID, how would you use Git to view the details of that specific commit, including the author, date, and commit message?

Solution:

To view the details of a specific commit, including the author, date, and commit message, you can use the git show or git log command with the commit ID. Here are both options:

1. Using git show:

bash
git show <commit-ID>

Replace <commit-ID> with the actual commit ID you want to view. This command will display detailed information about the specified commit, including the commit message, author, date, and the changes introduced by that commit.

For example:

$ git show abc123

2. Using git log:

$ git log -n 1 <commit-ID>

The -n 1 option tells Git to show only one commit. Replace <commit-ID> with the actual commit ID. This command will display a condensed view of the specified commit, including its commit message, author, date, and commit ID.

For example:

$ git log -n 1 abc123

Both of these commands will provide you with the necessary information about the specific commit you're interested in.

Experiment 10.

Analysing and Changing Git History

Write the command to list all commits made by the author "JohnDoe" between "2023-01-01"and "2023-12-31."

Solution:

To list all commits made by the author "JohnDoe" between "2023-01-01" and "2023-12-31" in Git, you can use the git log command with the --author and --since and --until options. Here's the command:

```
$ git log --author="JohnDoe" --since="2023-01-01" --until="2023-12-31"
```

This command will display a list of commits made by the author "JohnDoe" that fall within the specified date range, from January 1, 2023, to December 31, 2023. Make sure to adjust the author name and date range as needed for your specific use case.

Experiment 11**.**

Analysing and Changing Git History

Write the command to display the last five commits in the repository's history.

Solution:

To display the last five commits in a Git repository's history, you can use the git log command with the -n option, which limits the number of displayed commits. Here's the command:

```
$ git log -n 5
```

This command will show the last five commits in the repository's history. You can adjust the number after -n to display a different number of commits if needed.

Experiment 12.

Analysing and Changing Git History

Write the command to undo the changes introduced by the commit with the ID "abc123".

Solution:

To undo the changes introduced by a specific commit with the ID "abc123" in Git, you can use the git revert command. The git revert command creates a new commit that undoes the changes made by the specified commit, effectively "reverting" the commit. Here's the command:

```
$ git revert abc123
```

Replace "abc123" with the actual commit ID that you want to revert. After running this command, Git will create a new commit that negates the changes introduced by the specified commit. This is a safe way to undo changes in Git because it preserves the commit history andcreates a new commit to record the reversal of the changes.

Viva questions.

1. What is Git?

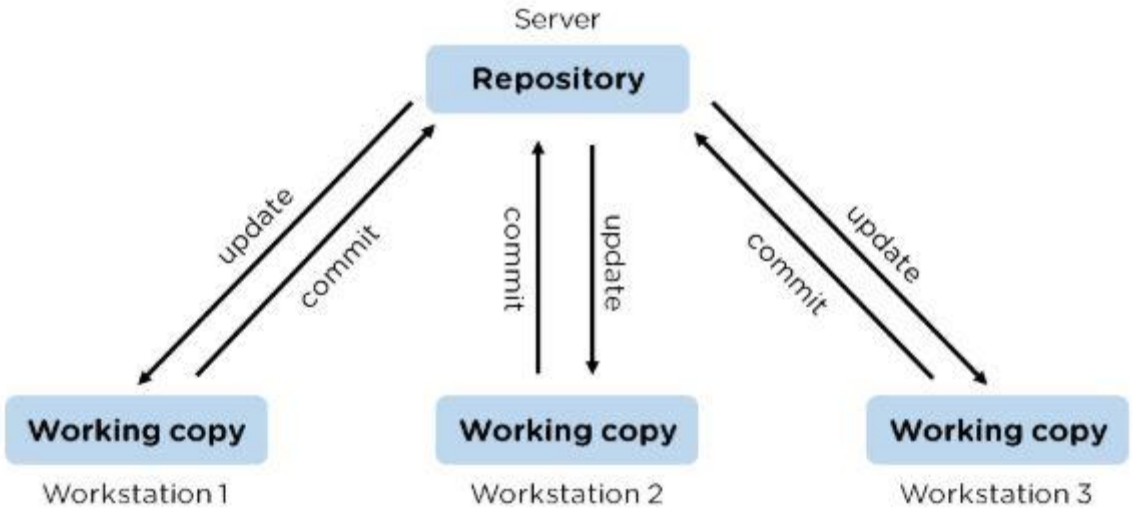
Git is a version control system for tracking changes in computer files and is used to help coordinate work among several people on a project while tracking progress over time. In other words, it's a tool that facilitates source code management in software development.

Git favors both programmers and non-technical users by keeping track of their project files. It enables multiple users to work together and handles large projects efficiently.

2. What do you understand by the term 'Version Control System'?

A version control system (VCS) records all the changes made to a file or set of data, so a specific version may be called later if needed.

This helps ensure that all team members are working on the latest version of the file



3. What is GitHub?

To provide Internet hosting for version control and software development, GitHub makes use of Git.

4. Mention some popular Git hosting services.

- GitHub
- SourceForge
- GitLab
- Bitbucket

5. Different types of version control systems

- Local version control systems have a database that stores all file changes under revision control on disc in a special format.
- Centralized version control systems have a single repository, from which each user receives their working copy.
- Distributed version control systems contain multiple repositories, and different users can access each one with their working copy.

6. What benefits come with using GIT?

- Data replication and redundancy are both possible.
- It is a service with high availability.
- There can only be one Git directory per repository.
- Excellent network and disc performance are achieved.
- On any project, collaboration is very simple.

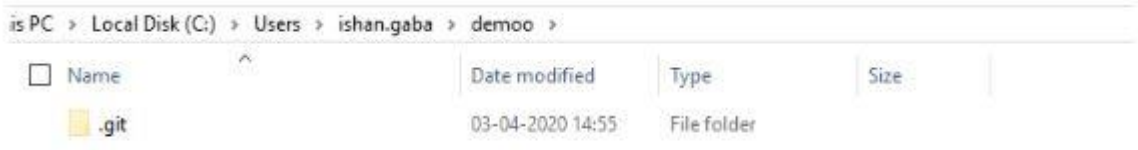
7. What's the difference between Git and GitHub?

Git	GitHub
Git is a software	GitHub is a service
Git can be installed locally on the system	GitHub is hosted on the web
Provides a desktop interface called git GUI	Provides a desktop interface called GitHub Desktop.
It does not support user management features	Provides built-in user

	management

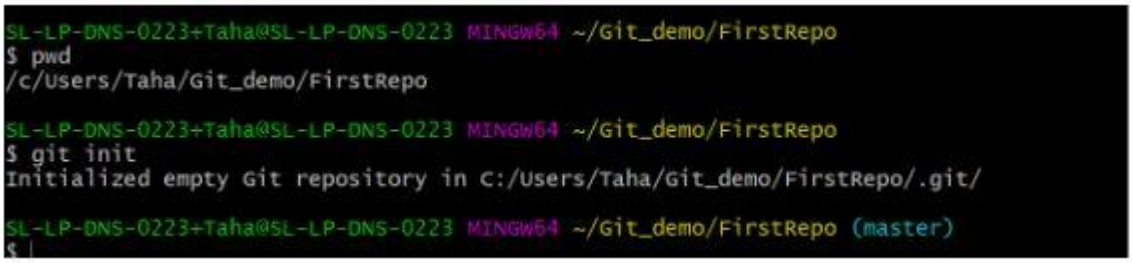
8. What is a Git repository?

Git repository refers to a place where all the Git files are stored. These files can either be stored on the local repository or on the remote repository.



9. How can you initialize a repository in Git?

If you want to initialize an empty repository to a directory in Git, you need to enter the git init command. After this command, a hidden .git folder will appear.



10. How is Git different from Subversion (SVN)?

GIT	SVN

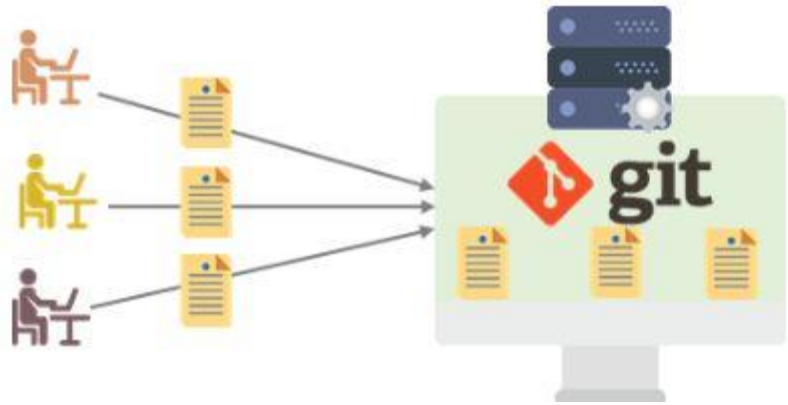
Git is a distributed decentralized version control system	SVN is a centralized version control system.
Git stores content in the form of metadata.	SVN stored data in the form of files.
The master contains the latest stable release.	In SVN, the trunk directory has the latest stable release
The contents of Git are hashed using the SHA-1 hash algorithm.	SVN doesn't support hashed contents.

11. Name a few Git commands with their function.

- Git config - Configure the username and email address
- Git add - Add one or more files to the staging area
- Git diff - View the changes made to the file
- Git init - Initialize an empty Git repository
- Git commit - Commit changes to head but not to the remote repository

12. What are the advantages of using Git?

- Faster release cycles
- Easy team collaboration
- Widespread acceptance
- Maintains the integrity of source code
- Pull requests



13. What language is used in Git?

Git is a fast and reliable version control system, and the language that makes this possible is ‘C.’

Using C language reduces the overhead of run times, which are common in high-level languages.

14. What is the correct syntax to add a message to a commit?

git commit -m "x files created"

15. Which command is used to create an empty Git repository?

git init - This command helps to create an empty repository while working on a project.

16. What does git pull origin master do?

The git pull origin master fetches all the changes from the master branch onto the origin and integrates them into the local branch.

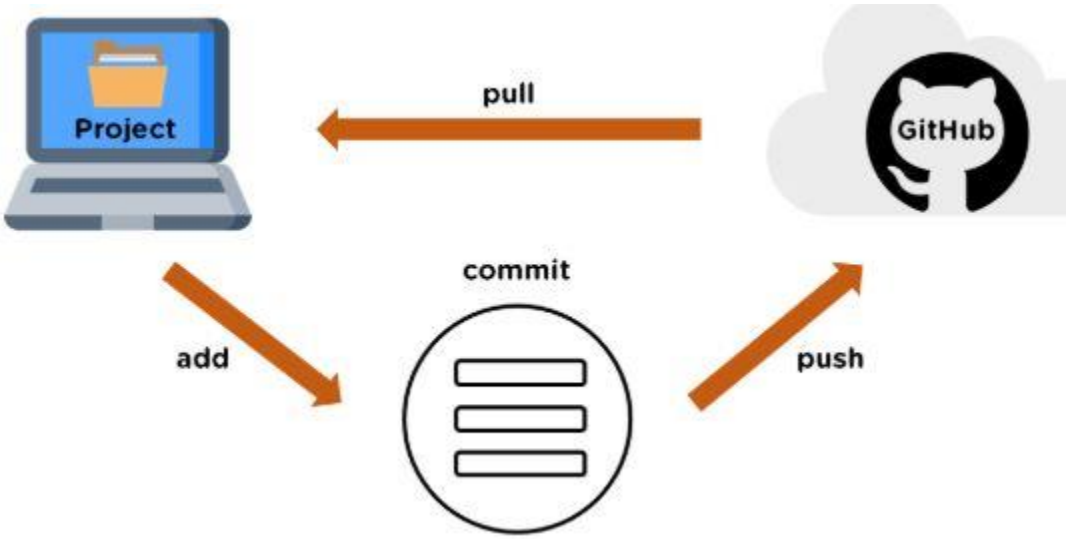
git pull = git fetch + git merge origin/ master

After having gone through the beginner level Git interview questions, let us now look at intermediate GIT interview questions and answers.

Intermediate Git Interview Questions

17. What does the git push command do?

The Git push command is used to push the content in a local repository to a remote repository. After a local repository has been modified, a push is executed to share the modifications with remote team members.



18. Difference between git fetch and git pull.

Git Fetch	Git Pull
The Git fetch command only downloads new data from a remote repository.	Git pull updates the current HEAD branch with the latest changes from the remote server.
It does not integrate any of these new data into your working files.	Downloads new data and integrate it with the current working files.
Command - git fetch origin git fetch –all	Tries to merge remote changes with your local ones. Command - git pull origin master

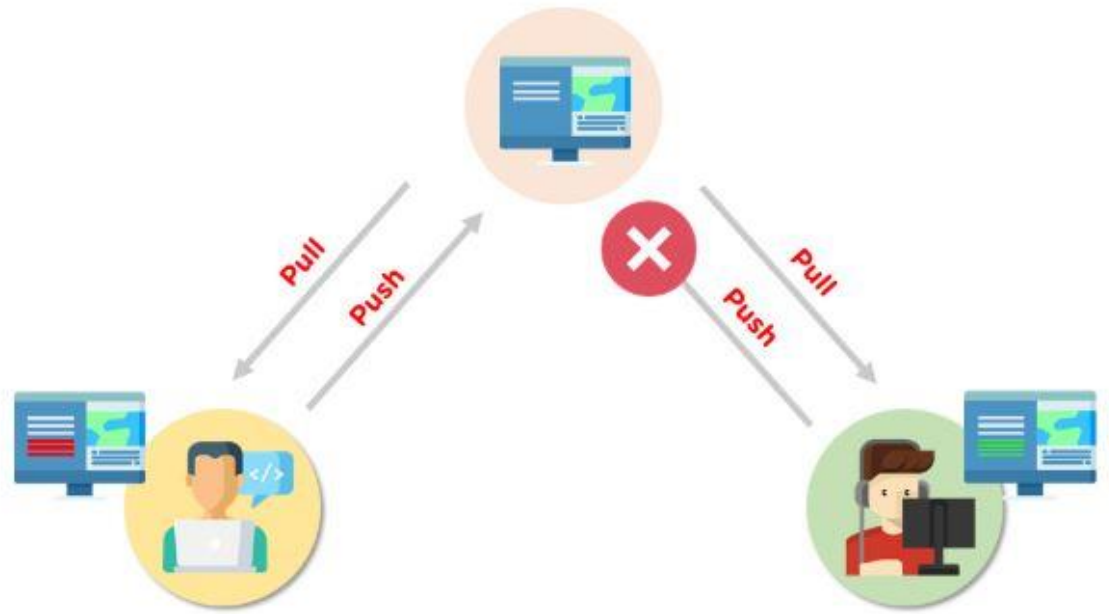
19. GitHub, GitLab and Bitbucket are examples of git repository _____ function?

hosting. All the three are services for hosting Git repositories

20. What do you understand about the Git merge conflict?

A [Git merge conflict] is an event that occurs when Git is unable to resolve the differences in code between the two commits automatically.

Git is capable of automatically merging the changes only if the commits are on different lines or branches.



21. How do you resolve conflicts in Git?

Here are the steps that will help you resolve conflicts in Git:

- Identify the files responsible for the conflicts.
- Implement the desired changes to the files
- Add the files using the git add command.
- The last step is to commit the changes in the file with the help of the git commit command.

22. What is the functionality of git ls-tree?

The git ls-tree command is used to list the contents of a tree object.

23. What is the process to revert a commit that has already been pushed and made public?

There are two processes through which you can revert a commit:

1. Remove or fix the bad file in a new commit and push it to the remote repository. Then commit it to the remote repository using:

git commit –m “commit message”

2. Create a new commit to undo all the changes that were made in the bad commit. Use the following command:

git revert <commit id>

24. How is a bare repository different from the standard way of initializing a Git repository?

Standard way	Bare way
You create a working directory with the git init command.	Does not contain any working or checked out copy of source files.

A .git subfolder is created with all the git-related change history.	Bare repositories store git revision history in the root folder of your repository instead of the .git subfolder.

25. What does git clone do?

Git clone allows you to create a local copy of the remote GitHub repository. Once you clone a repo, you can make edits locally in your system rather than directly in the source files of the remote repo

26. What is Git stash?

Let’s say you're a developer and you want to switch branches to work on something else. The issue is you don’t want to make commits in uncompleted work, so you just want to get back to this point later. The solution here is the Git stash.

Git stash takes your modified tracked files and saves it on a stack of unfinished changes that you can reapply at any time. To go back to the work you can use the stash pop.

27. What does the git reset --mixed and git merge --abort commands do?

git reset --mixed is used to undo changes made in the working directory and staging area.

git merge --abort helps stop the merge process and return back to the state before the merging began.

28. What do you understand about the

Staging area in Git?

The Staging Area in Git is when it starts to track and save the changes that occur in files. These saved changes reflect in the .git directory. Staging is an intermediate area that helps to format and review commits before their completion.

29. What is Git Bisect and how do you use it?

The Git Bisect command performs a binary search to detect the commit which introduced a bug or regression in the project's history.

Syntax: git bisect <subcommand> <options>

30. How do you find a list of files that has been changed in a particular commit?

The command to get a list of files that has been changed in a particular commit is:

git diff-tree –r {commit hash}

- -r flag allows the command to list individual files
- commit hash lists all the files that were changed or added in the commit.

31. What is the use of the git config command?

The git config command is used to set git configuration values on a global or local level. It alters the configuration options in your git installation. It is generally used to set your Git email, editor, and any aliases you want to use with the git command.

32. What is the functionality of git clean command?

The git clean command removes the untracked files from the working directory.

33. What is SubGit and why is it used?

SubGit is a tool that is used to migrate SVN to Git. It transforms the SVN repositories to Git and allows you to work on both systems concurrently. It auto-syncs the SVN with Git.

34. If you recover a deleted branch, what work is restored?

The files that were stashed and saved in the stashed index can be recovered. The files that were untracked will be lost. Hence, it's always a good idea to stage and commit your work or stash them.

35. Explain these commands one by one– git status, git log, git diff, git revert <commit>, git reset <file>.

- Git status - It shows the current status of the working directory and the staging area.
- Git revert<commit> - It is used for undoing changes to a repository's commit history.
- Git log- It is a key tool for reviewing and reading the history of everything that happens to a repository.
- Git diff- It is a multi-purpose Git command that performs a diff function on Git data sources when executed.
- Git reset<file>- it is used to unstage a file.

36. What exactly is tagging in Git?

Tagging enables developers to mark all significant checkpoints as their projects progress.

37. What exactly is forking in Git?

It is a repository duplicate and forking allows one to experiment with changes without being concerned about the original project.

38. How to change any older commit messages?

You can change the most recent commit message with the git commit —amend command.

39. How to handle huge binary files in Git?

Git LFS is a Git extension for dealing with large and binary files in a separate Git repository.

40. Name a few GIT tools.

Git comes with a few built-in tools like Git Bash and Git GUI.

41. Will you make a new commit or amend an existing one?

The git commit —amend command allows you to easily modify the most recent commit.

42. What do you mean by branching strategy?

It is employed by a software development team while writing and managing code with a version control system.

43. Difference between head, working tree,

and index.

They are all names for various branches. Even Though a single git repository can track an arbitrary number of branches, the working tree is only associated with one of them, and HEAD points to that branch.

44. Is there a git GUI client available for Linux?

Git includes built-in GUI tools for committing (git-gui) and browsing (gitk), but there are a number of third-party tools available for users seeking platform-specific experience.

45. What is the benefit of a version control system?

Version control enables software teams to maintain efficiency and agility while the team grows by adding more developers

46. What do you mean by git instaweb?

It is a script used to set up a temporary instance of Gitweb.

47. What exactly is the forking workflow?

Forking is a git clone operation that is performed on a server copy of a project's repository.

48. Mention benefits of forking workflow.

Contributions can be integrated without everyone trying to push to a single central repository.

49. What is the Gitflow workflow?

The Gitflow Workflow specifies a strict branching model centered on the project release.

50. What does the commit object contain?

The commit object contains a tree of blob objects and other tree objects that represent the project revision's directory structure.

51. Write the syntax of rebasing in git.

Syntax is as follows: $git rebase <branch name>

52. What are Git Hooks?

They are scripts that are executed automatically whenever a specific event occurs in a Git repository.

53. What is Git stash vs Git stash pop?

Git stash pop removes the (topmost, by default) stash when applied, whereas git stash apply keeps it in the stash list for future use.

54. Explain git reflog This command is used by Git to record changes made to the branches' tips.

55. Role of the git annotate command.

In git, it is used to track each line of the file based on the commit information.

56. What is a git Directory?

It is the storage place of the metadata and object database of the project.

57. How can a conflict be settled in Git?

Edit the files to resolve any incompatible changes first, then use "git add" to add the corrected files and "git commit" to save the repaired merge.

58. What is the standard method for branching in GIT?

In GIT, the best way to create a branch is to have one'main' branch and then another branch for implementing the changes that we want to make.

59. How do you set up a Git repository?

If you want to add an empty repository to a directory in Git, use the git init command.

60. What is the proper syntax for appending a message to a commit?

Git commit -m "x files created" is the syntax.

61. Use of git instaweb.

It is used to launch a web browser and a webserver with an interface into a local repository automatically.

62. Describe git is-tree.

It represents a tree object with each item's mode and name included.

63. What exactly is git cherry-pick?

A command typically used to move specific commits from one branch of a repository to another.

64. State the difference between “git remote” and “got clone”?

“Git remote” allows you to create an entry in the git configuration which specify a URL.

“Git clone” lets you create a new git repository by letting you copy it from the current URL.

65. Difference between “pull request” and “branch”?

“Pull request” is done when you feel like changing the developer’s change to another person's code branch. And “Branch” is just a separate version of code.

66. How might you recover a branch that has previously pushed changes in the main repository yet has been coincidentally erased from each team member's local machines?

We can easily recover this by seeing the latest commit of the branch in the reflog and then going through the new branch.

67. What is a detached head?

Detach head refers to that the currently checked repository is not in the local branch.

68. What command helps us to know the

branches merged into master and which are not?

git branch - -merged lets us get the lost of the branches which are currently merged into the current branch

git branch - - no- merged shows the branches which are not merged

69. Is LDAP Authentication Supported?

GitLab API only supports LDAP authentication since version 6.0 and higher.

Now let’s raise the level of difficulty with advanced Git interview questions and answers.

Interested to learn more about Git? Check out the DevOps Engineer Master's Program and get certified today.

Popular Git Interview Questions

70. A simple definition of Git

Git is a free and open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

71. Is the C++ language used in Git?

Yes, but C programming language is a widely used language in Git.

72. How does Git work?

Git works by tracking changes to files in a project and allowing developers to easily revert to previous versions if necessary. Git also makes it easy to collaborate on

projects, as it allows multiple developers to work on the same codebase simultaneously.

73. What are some of the most popular Git commands?

Some of the most popular Git commands are,

- "git init" (which initializes a Git repository)
- "git add" (which adds files to a Git repository)
- And "git commit" (which saves changes to a Git repository).

74. List out the functions provided by Git repository hosting service

There are many functions that a Git repository hosting service can provide. Some of the most common and useful functions include:

- Providing a web interface for users to interact with the repositories
- Allowing users to clone repositories
- Allowing users to view and download statistics about the repositories
- Providing a way for users to push changes to the repositories
- Keeping track of changes made to the repositories

75. What is the difference between Git and other revision control systems?

- Git is a distributed revision control system, which means that it can be used without a central server. This allows for a great deal of flexibility in how projects are managed.

- On the other side, revision control systems are often centralized, which can limit the flexibility of how projects are managed.

76. How does Git work?

Git works by tracking changes to files in a repository. When a file is changed, Git calculates a unique identifier for the change, called a "commit hash". The commit hash allows Git to identify the change and track it over time.

77. How do I install Git?

Installing Git is simple. Just download the latest version from the Git website (https://git-scm.com/).

78. How do I use Git?

To use Git, a developer first creates a local repository on their computer. This repository contains all the files for a project and the history of all the changes made to those files.

Or just follow the instructions in the Git documentation (https://git-scm.com/doc).

79. What are some of the drawbacks of Git?

- One of the main drawbacks is that it can be difficult to learn and use, especially for those who are not familiar with version control systems.
- And Git is not always reliable and can sometimes be slow.

80. What are some of the most important commands in Git?

Some of the most important Git commands are "commit", "push", and "pull".

- The "commit" command is used to save changes to the local repository.

- And the "push" command is used to send changes to the remote repository.
- Then the "pull" command is used to retrieve changes from the remote repository.

81. What are some of the most important features of Git?

Some of the most important features of Git are its distributed nature, its ability to track changes, and its support for branches.

- The distributed nature of Git allows developers to work independently and offline.
- The ability to track changes helps developers to keep track of their work and revert to previous versions if necessary.
- The support for branches allows developers to experiment with new features without affecting the main codebase.

82. What is a branch in Git?

A branch is a way to isolate development work on a particular aspect of a project. When a branch is created, it diverges from the primary branch. It allows developers to work on a new feature or bug fix without affecting the main codebase.

83. What is a commit in Git?

A commit is a way to save changes to a branch. When a commit is made, a snapshot of the current state of the branch is created. This snapshot can be used to revert the branch to that state if necessary.

84. What is conflict in Git?

Conflict in Git occurs when two or more developers have made changes to the same part of a file, and those changes can't be automatically merged. When this happens, Git will mark the file as conflicted and leave it up to the developers to resolve the conflict.

Resolving a conflict can be done by manually editing the file to choose which changes should be kept, or by using a tool like Git's merge command to automatically merge the changes.

85. What does the git status command do?

The git status command is used to obtain the current state of a Git repository. This command can be used to determine whether the repository is clean or dirty, and to see which files have been modified. The git status command will also show which branch is currently checked out and whether there are any uncommitted changes.

86. Why is it considered to be easy to work on Git?

There are many reasons that Git is considered an easy tool to work with.

- It has a straightforward learning curve. Even those new to programming can easily learn how to use Git with just a few hours of practice.
- Git is highly flexible and can be easily customized to fit the needs of any project.
- And, Git is very stable and reliable, so users can trust that their work will be safe and sound.

87. What do you know about Git Stash?

Git stash is a powerful tool that allows you to save your changes and revert your working directory to a previous state. This is especially useful when switching branches or reverting to a previous commit.

And Git Stash takes a snapshot of your changes and stores them away for later use.

88. What differentiates between the commands git remote and git clone?

The main difference between the git remote and git clone commands is that the git remote adds a remote repository as a shortcut to your current repository, while the git clone creates an entirely new copy of a remote repository.

89. Tell me the difference between git pull and git fetch?

Both of these commands will fetch any new commits from the remote repository, but they differ in how they handle these commits.

Git pull will merge the remote commits into the current branch, while git fetch will simply retrieve the commits and store them in the local repository. This means that if you have any uncommitted changes, git pull may result in merge conflicts, while git fetch will not.

90. Is Git and GitHub the same thing?

No, Git and GitHub are two different things.

- Git is a version control system that lets you track changes to your code.
- GitHub is a hosting service for Git repositories. You can use GitHub to store your code remotely, or you can use it to collaborate with other developers on a project.

91. What about Git reflog?

Git reflog is a history of all the changes made to a git repository. It is a valuable tool for debugging and troubleshooting purposes.

And Git reflog can be used to view the history of a repository, see who made what changes, and when those changes were made.

92. What is a detached head?

A detached HEAD is a state where the HEAD pointer is not pointing to the current commit. This can happen if you check out a commit that is not the most recent, or if you reset your head to a previous commit.

93. How to avoid a detached head?

There are a few different ways to avoid a detached HEAD.

- The first is to simply commit your changes before switching branches. This will ensure that your changes are saved to a specific branch, and you won't have to worry about them being lost when you switch branches.
- Another way to avoid detached HEAD is to use the "git checkout" command with the "-b" option.

94. How will you resolve conflict in Git?

To resolve a conflict in Git, you will need to first identify the source of the conflict. Once you have identified the source of the conflict, you can use the "git pull" command. This will pull the latest changes from the remote repository and merge them with your local copy.

If the "git pull" command doesn't resolve the conflict, you can try the "git merge" command. This will merge the two versions of the code manually. You will need to resolve the conflicts manually and then commit the merged code.

95. What is Subgit and where do you use Subgit?

Subgit is a tool for managing Git repositories with Subversion history. It allows you to keep your existing subversion history while moving to Git, and it also provides a way to keep your Git history synchronized with subversion.

There are many reasons that you might want to use Subgit,

- For example, if you have a large subversion repository with a lot of history, moving to Git can be a huge undertaking; that's where Subgit can help transition smoother.
- And if you work in an environment where subversion is the primary version control system, using Subgit can help you keep your Git history in sync with the rest of the team.

Advanced Git Interview Questions

96. Explain the different points when a merge can enter a conflicted stage.

There are two stages when a merge can enter a conflicted stage.

1. Starting the merge process

If there are changes in the working directory of the stage area in the current project, the merge will fail to start. In this case, conflicts happen due to pending changes that need to be stabilized using different Git commands.

2. During the merge process

The failure during the merge process indicates that there’s a conflict between the local branch and the branch being merged. In this case, Git resolves as much as possible, but some things have to be fixed manually in the conflicted files.

97. What has to be run to squash the last N commits into a single commit?

In Git, squashing commits means combining two or more commits into one.

Use the below command to write a new commit message from the beginning.

git reset -soft HEAD~N &&git commit

But, if you want to edit a new commit message and add the existing commit messages, then you must extract the messages and pass them to Git commit.

The below command will help you achieve this:

git reset -soft HEAD~N &&git commit -edit -m“$(git log -format=%B -reverse .HEAD@{N})”

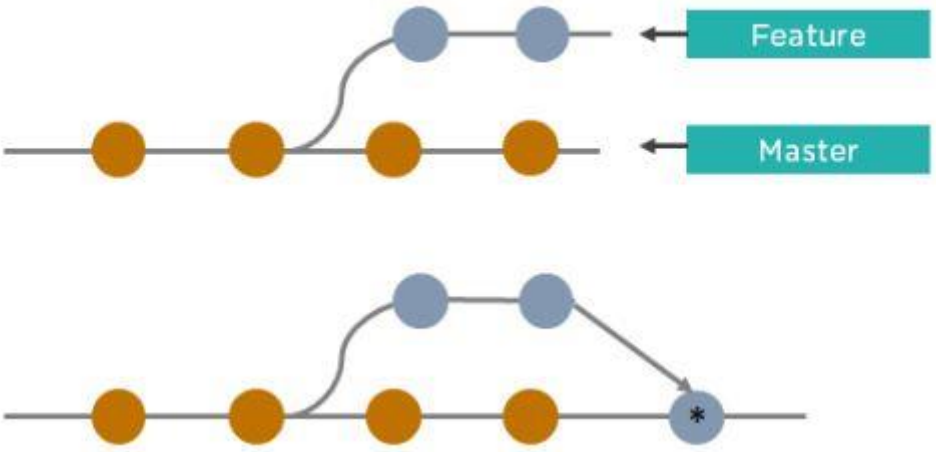
98. What is the difference between fork, branch, and clone?

Fork	Branch	Clone
The fork is the process when a copy of the repository is made. It's usually experimentation in the project without affecting the original project. They’re used to advise changes or take inspiration from someone else’s project.	Git branches refer to individual projects within a git repository. If there are several branches in a repository, then each branch can have entirely different files and folders.	Git clone refers to creating a clone or a copy of an existing git repository in a new directory. Cloning automatically creates a connection that points back to the original repository, which makes it very easy to interact with the central repository.

99. How is Git merge different from Git

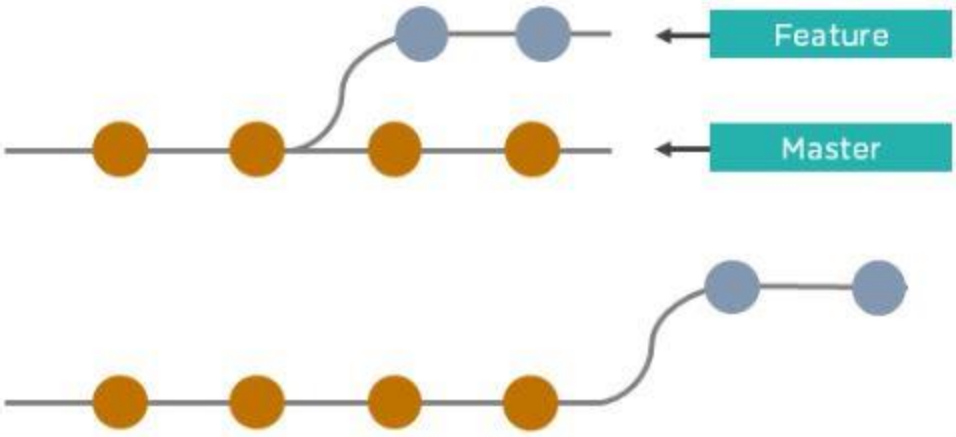
rebase?

Git merge is used to incorporate new commits into your feature branch.



- Git merge creates an extra merge commit every time you need to incorporate changes.
- It pollutes your feature branch history.

As an alternative to merging, you can rebase the feature branch into master.



- Git rebase Incorporates all the new commits in the master branch.
- It rewrites the project history by creating brand new commits for each commit in the original branch

100. What is the command used to fix a broken commit?

To fix a broken commit in Git, you may use the “git commit --amend” command, which helps you combine the staged changes with the previous commits instead of creating an entirely new commit.

101. How do you recover a deleted branch that was not merged?

To recover a deleted branch, first, you can use the git reflog command. It will list the local recorded logs for all the references. Then, you can identify the history stamp and recover it using the git checkout command.

102. What is git stash drop?

The Git stash drop command is used to remove a particular stash. If there’s a stash you're no longer using or you want to remove a specific item of stash from the list, you can use the stash commands.

Let’s say you want to delete an item named stash@{abc}; you can use the command:

git stash drop stash@{abc}.

103. What’s the difference between reverting and resetting?

Reverting	Resetting
The revert command in Git is used to create a new	Git reset is a command that is

commit that undoes the changes made in the previous commit. When you use this command, a new history is added to the project; the existing history is not modified.	used to undo the local changes that have been made to a Git repository. Git reset operates on the following: commit history, the staging index, and the working directory.

104. How can you discover if a branch has already been merged or not?

There are two commands to determine these two different things.

git branch --merged - Returns the list of branches that have been merged into the current branch.

git branch --no-merged - Returns the list of branches that have not been merged.

105. What is “git cherry-pick”?

The command git cherry-pick enables you to pick up commits from a branch within a repository and apply it to another branch. This command is useful to undo changes when any commit is accidentally made to the wrong branch. Then, you can switch to the correct branch and use this command to cherry-pick the commit.

1. **What Is Gitlab?**

Answer: GitLab is a web tool to assist with visualizing and managing your git projects. GitLab includes Git repository management, code reviews, issue tracking, wikis, and more. Collaborate with your team using issues, milestones, and line-by-line code review or view activity streams of projects or the people you work with.

Git and GitLab are here to help with managing projects, merging development between different people, with different time zones, and giving Sealed Air the ability to manage all of its source code in 1 location.

2. **When Should I Consider Gitlab?**

Answer: Version control software, like Git, allows you to have “versions” of a project, which show the changes that were made to code, or text files, over time, and allows you to backtrack if necessary and undo those changes or merge your code with others, line by line.

3. **What are the minimum systems Requirements?**

Answer:

- GitLab Control requires at least iOS 8.1 on the iPhone, iPod Touch, iPad and iPad Mini.
- It currently supports GitLab server version 9.x and higher, but it is highly optimized for GitLab 10.x.
- Please note that some of the features may not be available or not work at all on versions prior to 9.0.

4. **What Is The Cost Per User Or Project?**

Answer: GitLab is an Open Source software application under MIT license running on our private and secure instance. There is no cost to add users to GitLab.

5. **What Happens When Active Gitlab Users Are Terminated?**

Answer: We have implemented Single Sign-On (SSO) for GitLab. This means that terminated users will not be able to access GitLab.

6. **Disadvantages of Gitlab**

Answer: Even if GitLab is simple to use, it’s a big piece of software that can sometimes become slow on the web user interface. Moreover, the review system is sometimes not so easy to use compared to other competitors.

7. **What is GitLab used for?**

Answer: GitLab is a management platform for Git repositories that provides integrated features like continuous integration, issue tracking, team support, and wiki documentation.

8. **Why GitLab is better than Jenkins?**

Answer: GitLab provides more than what Jenkins is hoping to evolve to, by providing a fully integrated single application for the entire DevOps lifecycle. More than Jenkins' goals, GitLab also provides planning, SCM, packaging, release, configuration, and monitoring (in addition to the CI/CD that Jenkins is focused on).

9. **Should I use GitLab or GitHub?**

Answer: it is safe to say that most code in the world resides on either GitLab or GitHub. … If you were to choose purely based on general popularity, GitHub would be the clear winner, with over 56 million users and more than 190 million repositories (including at least 28 million public repositories). But GitLab has its niche.

10. **Where are my Git Data Stored? How secure is it?**

Answer: GitLab Control stores your data both online (on your host) and locally. We use the device keychain to store your credentials and an encrypted database to store all the others data. Every message transferred is encrypted with TLS/SSL only when the appropriate option is enabled both on the app and on your GitLab instance.

11. **How do you revert a commit that had already been pushed and made public?**

Answer: One or more commits can be reverted with the use of git revert. This command, in essence, creates a new commit with patches that cancel out the changes introduced in specific commits. In case the commit that needs to be reverted has already been published or changing the repository history is not an option, git revert can be used to revert commits. Running the following command will revert the last two commits:

```
git revert HEAD~2..HEAD
```

12. **Advantages of Gitlab**

Answer: The significant advantage of GitLab is it is available for free and easy to manage and configure it. it enables only a limited number of private repository, integrate several API and third-party server and consume only a reliable uptime. The code reviews and pulls up requests made it more compact and user-friendly.

13. **Is GitLab better than bitbucket?**

Answer: Atlassian Bitbucket gives teams Git code management, but also one place to plan projects, collaborate on code, test, and deploy. … GitLab is a complete DevOps platform, delivered as a single application, with built-in project management, source code management, CI/CD, monitoring and more.

14. **Is Gitlab The Only Git Repository That I Can Use At Sealed Air?**

Answer: Whilst there is nothing stopping you from running Git only on your local machine to track changes, you will really start to see benefits by using GitLab and getting others involved in your projects. It is built for sharing and working together on code and text.

15. **When Should I Consider Gitlab?**

Answer: Version control software, like Git, allows you to have “versions” of a project, which show the changes that were made to code, or text files, over time, and allows you to backtrack if necessary and undo those changes or merge your code with others, line by line.

16. **Should I Be Concerned With Confidential Or Proprietary Solutions In Gitlab?**

Answer: GitLab is ideal for storing source code for projects; however, confidential information such as passwords and credentials should not be stored in git. With regards to hosting our code, there should be no issue with the hosting of code that is either open license (MIT, GPL, and BSD), we have created or licensed for us to use, however, code that has a license for limited use should be handled carefully. If you are not sure, please contact Risk Management or Legal teams.

17. **What Is The Cost Per User Or Project?**

Answer: GitLab is an Open Source software application under MIT license running on our private and secure instance. There is no cost to add users to GitLab.

18. **Is Ldap Authentication Supported?**

Answer: GitLab API supports LDAP authentication only since version 6.0 and higher.

19. **What Happens When Active Gitlab Users Are Terminated?**

Answer: We have implemented Single Sign-On (SSO) for GitLab. This means that terminated users will not be able to access GitLab.

20. **How do you set up a script to run every time a repository receives new commits through push?**

Answer: To configure a script to run every time a repository receives new commits through push, one needs to define either a pre-receive, update, or a post-receive hook depending on when exactly the script needs to be triggered.

The pre-receive hook in the destination repository is invoked when commits are pushed to it. Any script bound to this hook will be executed before any references are updated. This is a useful hook to run scripts that help enforce development policies.

Update hook works in a similar manner to pre-receive hook and is also triggered before any updates are actually made. However, the update hook is called once for every commit that has been pushed to the destination repository.

Finally, the post-receive hook in the repository is invoked after the updates have been accepted into the destination repository. This is an ideal place to configure simple deployment scripts, invoke some continuous integration systems, dispatch notification emails to repository maintainers, etc.

Hooks are local to every Git repository and are not versioned. Scripts can either be created within the hooks directory inside the ".git" directory, or they can be created elsewhere and links to those scripts can be placed within the directory.

21. **How do you squash the last N commits into a single commit?**

Answer: Squashing multiple commits into a single commit will overwrite history, and should be done with caution. However, this is useful when working in feature branches. To squash, the last N commits of the current branch, run the following command (with {N} replaced with the number of commits that you want to squash):

```
git rebase -i HEAD~{N}
```

Upon running this command, an editor will open with a list of these N commit messages, one per line. Each of these lines will begin with the word "pick". Replacing

“pick” with “squash” or “s” will tell Git to combine the commit with the commit before it. To combine all N commits into one, set every commit in the list to be squash except the first one. Upon exiting the editor, and if no conflict arises, git rebase will allow you to create a new commit message for the new combined commit.

22. **Is GitLab an alternative to Jenkins?**
Answer: Both are open-source tools used in the software development and deployment process. Gitlab is for Version Control and Code Collaboration, whereas Jenkins is for Continuous Integration.
23. **What is Git Bisect? How can you use it to determine the source of a (regression) bug?**
Answer: Git provides a rather efficient mechanism to find bad commits. Instead of making the user try out every single commit to find out the first one that introduced some particular issue into the code, git bisect allows the user to perform a sort of binary search on the entire history of a repository.
By issuing the command git bisect start, the repository enters bisect mode. After this, all you have to do is identify a bad and a good commit:

git bisect bad # marks the current version as bad

git bisect good {hash or tag} # marks the given hash or tag as good, ideally of some earlier commit

Once this is done, Git will then have a range of commits that it needs to explore. At every step, it will check out a certain commit from this range, and require you to identify it as good or bad. After which the range will be effectively halved, and the whole search will require a lot fewer steps than the actual number of commits involved in the range. Once the first bad commit has been found, or the bisect mode needs to be ended, the following command can be used to exit the mode and reset the bisection state:

24. **Why GitLab is better than Jenkins?**

Answer: GitLab provides more than what Jenkins is hoping to evolve to, by providing a fully integrated single application for the entire DevOps lifecycle. More than Jenkins' goals, GitLab also provides planning, SCM, packaging, release, configuration, and monitoring (in addition to the CI/CD that Jenkins is focused on).

25. **What is GitLab DevOps?**

Answer: GitLab is The **DevOps** platform that empowers organizations to maximize the overall return on software development by delivering software faster, more efficiently, while strengthening security and compliance.