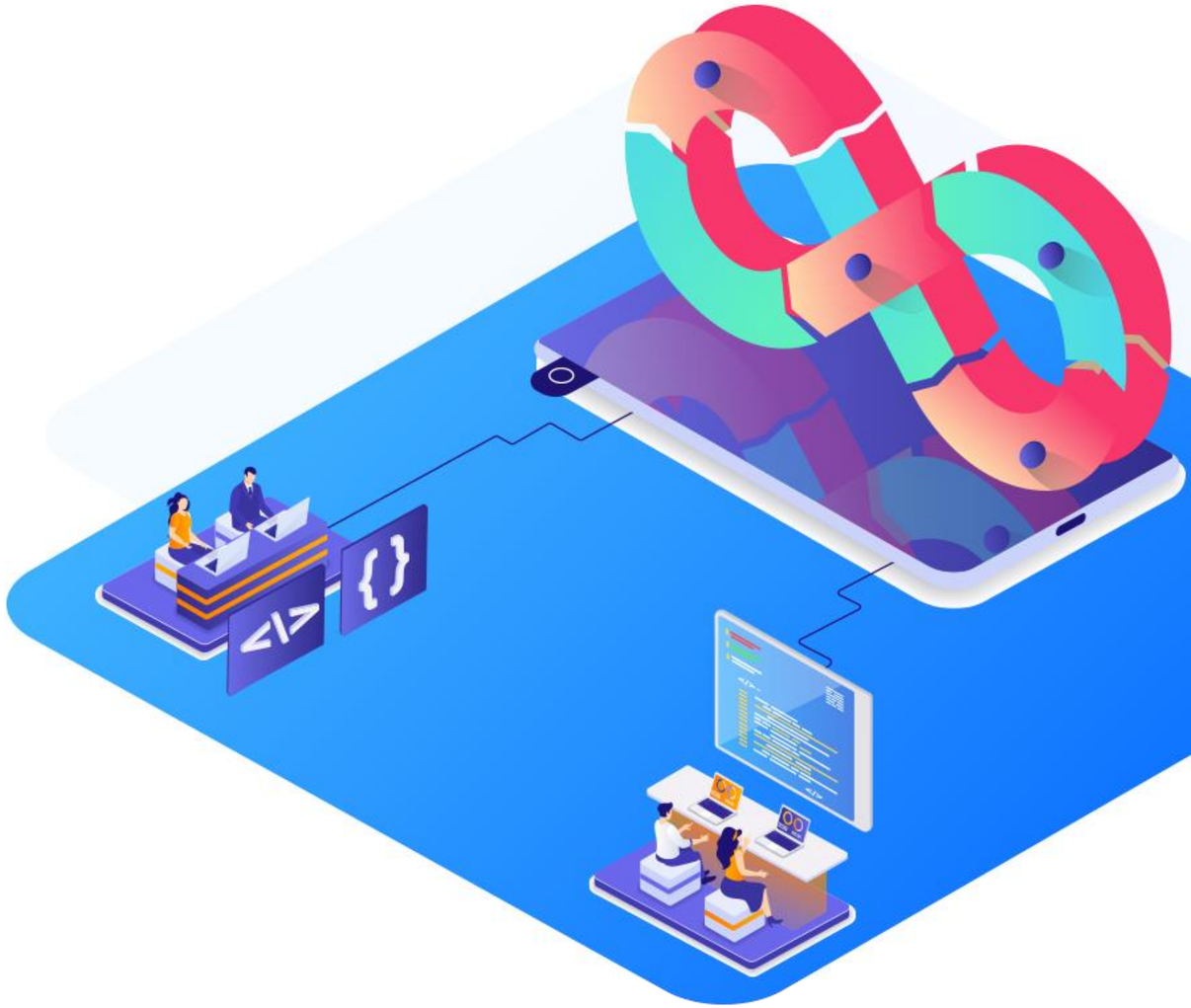
DevOps Foundations: Version Control and CI/CD with Jenkins



Basics of Version Control System with Git

Learning Objectives

By the end of this lesson, you will be able to:

- Utilize a version control system to identify and understand its key components for tracking changes in software projects
- Create and clone GitHub repositories to understand the fundamentals of repository management and version control
- Use the *git push* command to upload changes from a local repository to a remote Git repository
- Create a pull request to propose merging the changes into a remote GitHub codebase

Learning Objectives

By the end of this lesson, you will be able to:

- Apply branching and merging techniques to facilitate simultaneous collaboration in project development
- Utilize the Git commands to create and switch to a new branch simultaneously to streamline a Git workflow
- Assess and resolve basic merge conflicts in a version control system by addressing the differences between conflicting code blocks

Overview of Version Control System

Source Code Management (SCM)

It is a system used to manage changes to source code over time.

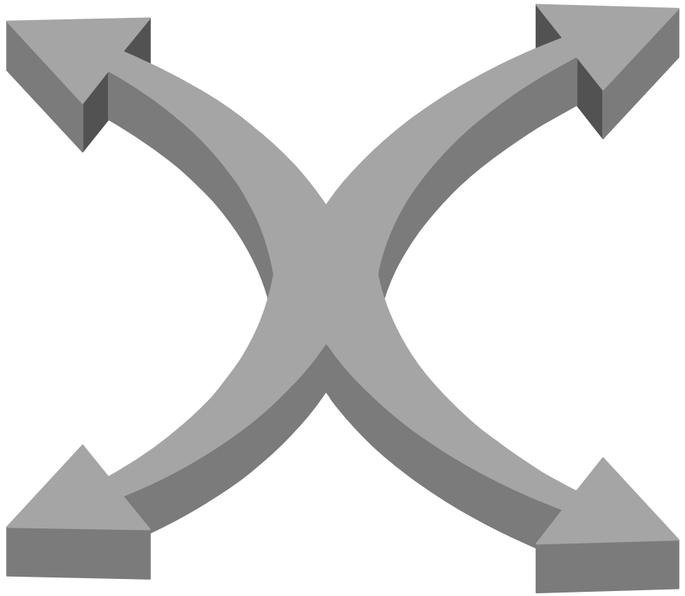
Purpose

- Track modifications to a source code repository and monitor changes made to the code
- Maintain a running history of changes to a codebase
- Manage code revisions and updates

Source Code Management: Benefits

These are a few major benefits of a source code management system:

Work together in teams:
Enables collaborative coding without conflicts



Version history:
Tracks changes in version history for easy retrieval

Generate release notes:
Automates the generation of release notes, saving time and effort

Backup of code:
Ensures centralized storage in SCM for code backup and easy access

What Is a Version Control System (VCS)?

It is a system that records changes to a set of files over a period to recall specific versions. It is used to store every version of an image or layout.

Example:

Advantages of VCS

Some benefit of version control system are:

Restoring previous versions
Revert to earlier versions of files or code

Storing versions
Track the changes and maintain a history of revisions

Collaborating
Facilitate teamwork by allowing multiple users to work on the same files

Code evolution
Gain insights into the evolution of code or files through version history

Backing up
Ensure data safety by having backups of all versions stored in the system

VCS vs. SCM

Aspect	VCS	SCM
Focus	Manage file versions	Manage file versions along with collaboration
Purpose	Tracks changes in files	Tracks changes and facilitates collaboration
Usage	For code versioning and collaboration	For managing source code across development phases
Examples	Git, SVN, Mercurial	GitLab, GitHub, Bitbucket

Key Concepts of a VCS

Repository: Maintain a central location that stores and displays all project versions and history

Branching: Enable parallel development for multiple versions of a project to coexist

Tagging: Create named markers for specific points in the project history

Merging: Combine changes from one branch into another

Commit: Save changes into the version history with a description

Cloning: Create a local copy of a remote repository

Key Concepts of a VCS

Forking: Create an independent copy of a repository

Pull request: Propose changes to merge from one branch or fork into another

Annotate: Identify who made specific changes in the code

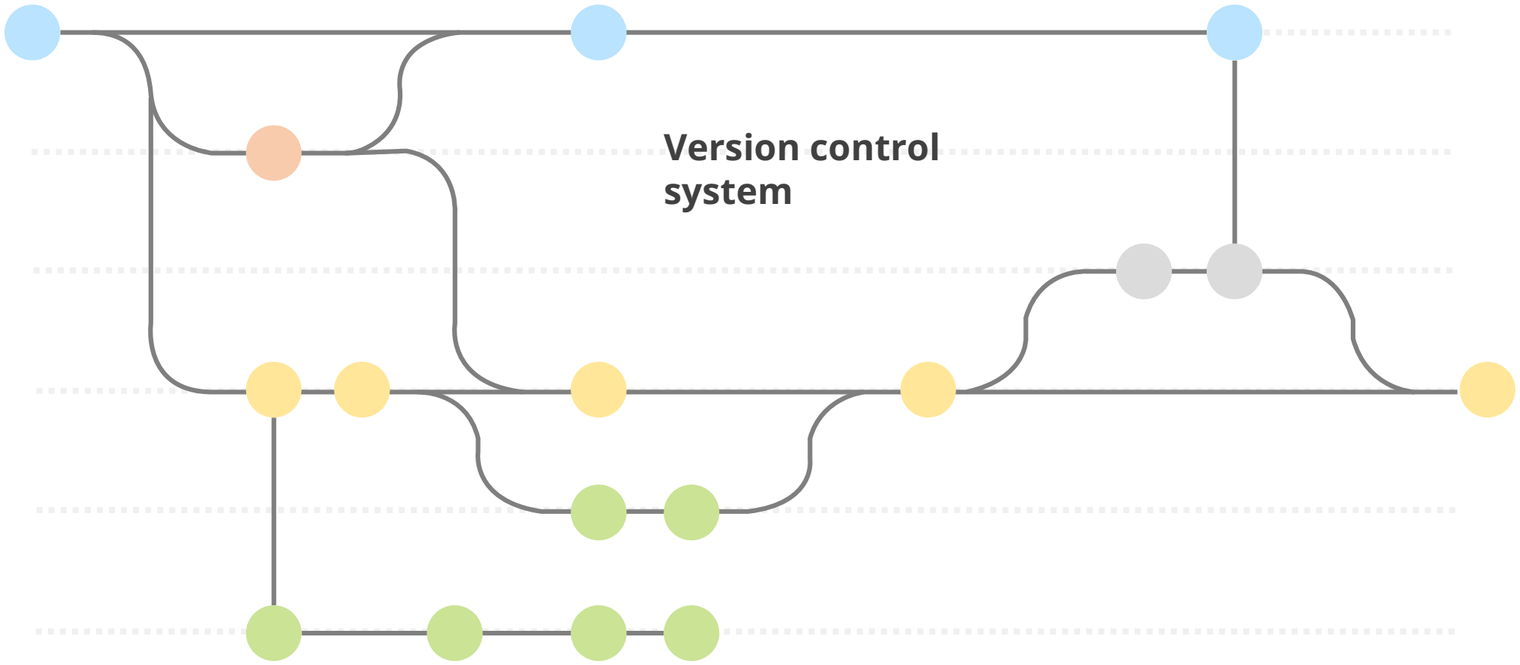
Rollback or Revert: Return to a previous state of the codebase

Stash: Save changes temporarily that are not ready for committing

Changelog: Document all the changes made to a project

Role of VCS in DevOps Lifecycle

Version Control Systems play a pivotal role in the DevOps lifecycle by enabling efficient collaboration and streamlined code management.



Role of VCS in DevOps Lifecycle

VCS enables seamless integration, faster recovery, and collaboration in the DevOps lifecycle. Below are the phases of the DevOps lifecycle where VCS plays a crucial role:

01 **Planning phase**: Organize tasks and document progress to track changes and maintain version history

02 **Design phase**: Facilitate collaboration on software blueprints, design, and iterate system architecture

03 **Development phase**: Manage code, version control, and collaboration among developers during coding and testing

Role of VCS in DevOps Lifecycle

Below are the phases of the DevOps lifecycle where VCS plays a crucial role:

04 **Testing phase**: Track code changes, manage test scripts, and ensure test coverage for software testing

05 **Release phase**: Prepare for deployment, tag versions, and manage release branches for deployment activities

06 **Support phase**: Manage bug fixes, track issues, and maintain code stability post-deployment for ongoing support and maintenance

Types of VCS

There are three main types of version control systems:

Local version control systems

For storing changes locally before pushing them to a single code version

Central version control systems

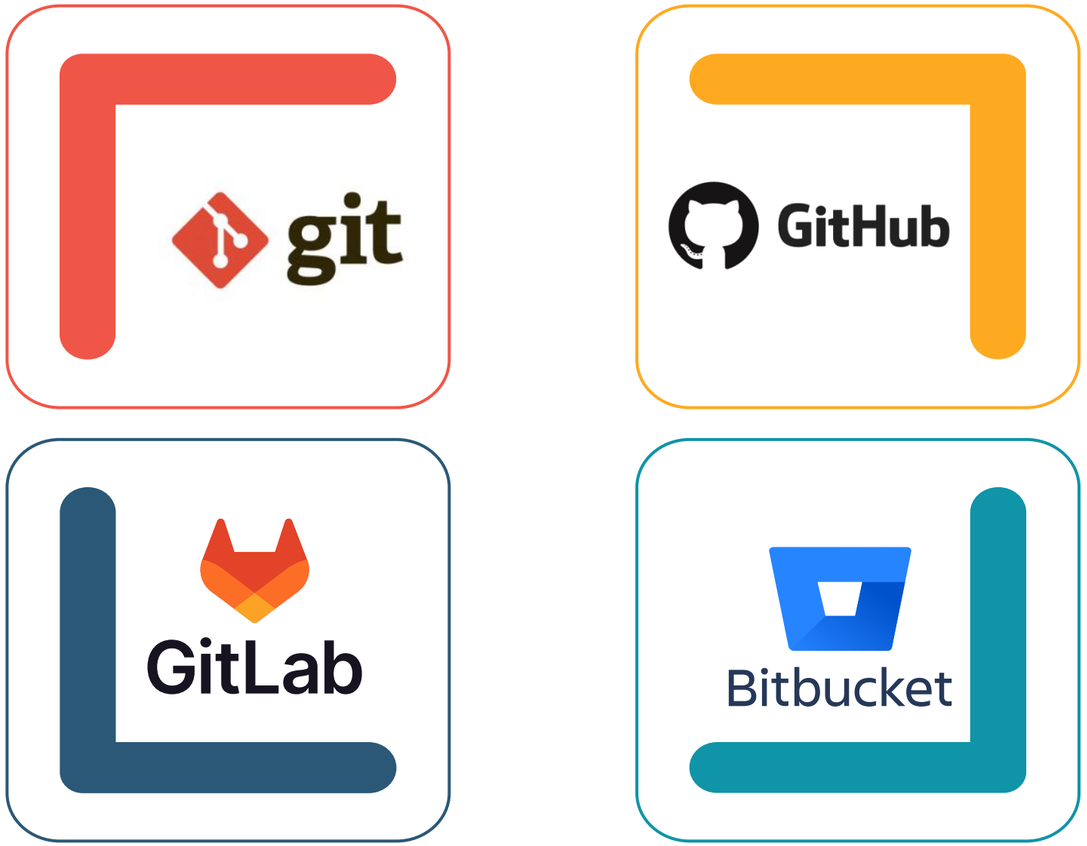
For hosting various code versions in a central repository to access and manage changes

Distributed version control systems

For mirroring the central repository history in each local repository to ensure robustness for easy retrieval

Tools for VCS

Here are some of the most widely used VCS tools:



Overview of VCS Tools

Git is a version control system for tracking code file changes and is generally used for source code management in software development.

- Employs distributed version control system
- Minimizes memory footprint
- Manages changes in any file
- Is utilized by major companies like Google, Facebook, and Microsoft

Overview of VCS Tools

GitHub provides a web-based Git repository hosting service that provides a web interface to upload files.

- Facilitates code collaboration and version control
- Tracks code modifications and history
- Allows correction of mistakes by reverting changes made to the codebase
- Supports collaboration among team members

Overview of VCS Tools

Bitbucket is a professional Git repository management solution for teams, centralizing repository management, source code collaboration, and development guidance.

- Offers code collaboration platform by Atlassian
- Supports code branches for parallel development
- Allows in-line comments and debate for code review
- Enables pull requests for code contribution and review

Overview of VCS Tools

GitLab is a service that provides remote access to Git repositories.

- Provides internal management of git repositories
- Keeps the user code private
- Deploys the change on the user code
- Provides user-friendly web interface layer

Version Control: Best Practices

Best practices for implementing version control systems:

- Commit code changes granularly with clear messages
- Utilize branches for managing software versions and patches
- Ensure compliance with disaster recovery and access control measures
- Backup the version control repository to prevent data loss in case of emergencies
- Monitor and audit version control activities to ensure security and integrity of codebase

Quick Check

In development environment, which tool is chosen for streamlined code management and collaboration, serving as a version control system (VCS) across projects?

A. Java

B. Git

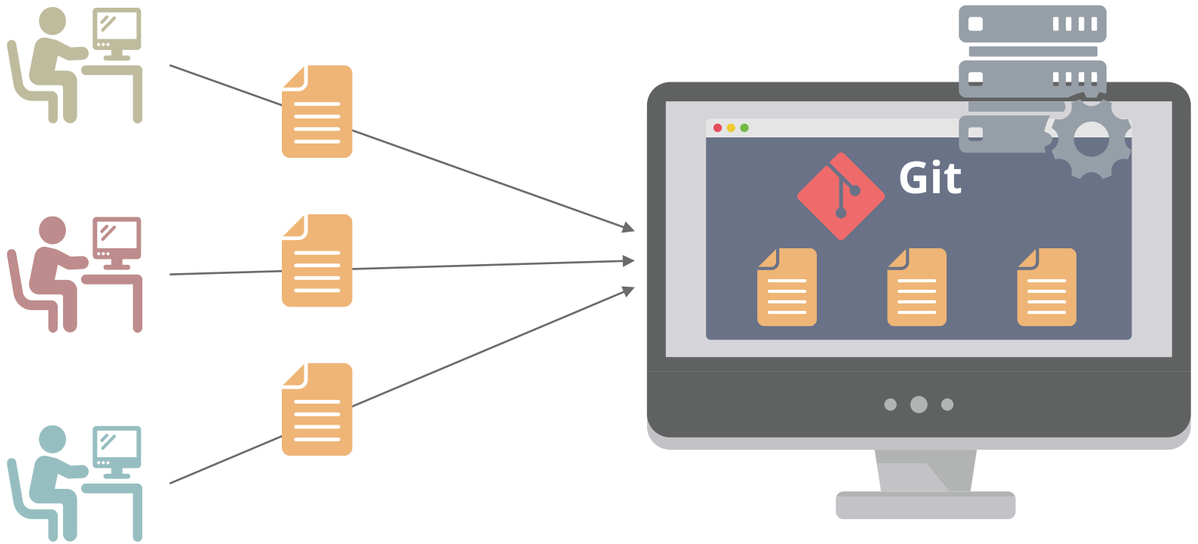
C. Visual Studio

D. Docker

Introduction to Git as a VCS

What Is Git?

Git is an open-source distributed version control system used for tracking changes in source code during software development.



It allows multiple developers to work on the same project simultaneously, facilitating collaboration and enabling efficient management of changes to the codebase.

Purpose of Using Git

Track changes and revert to previous versions

Allow collaboration without conflicts

Provide branching and merging feature

Manage the development of large projects efficiently

Features of Git

Features of Git

Keeps track of every change

Is open source

Is secure

Is a distributed version control system

Provides a staging area

Supports non-linear development

Is scalable

Enables branching and merging

Git vs. GitHub

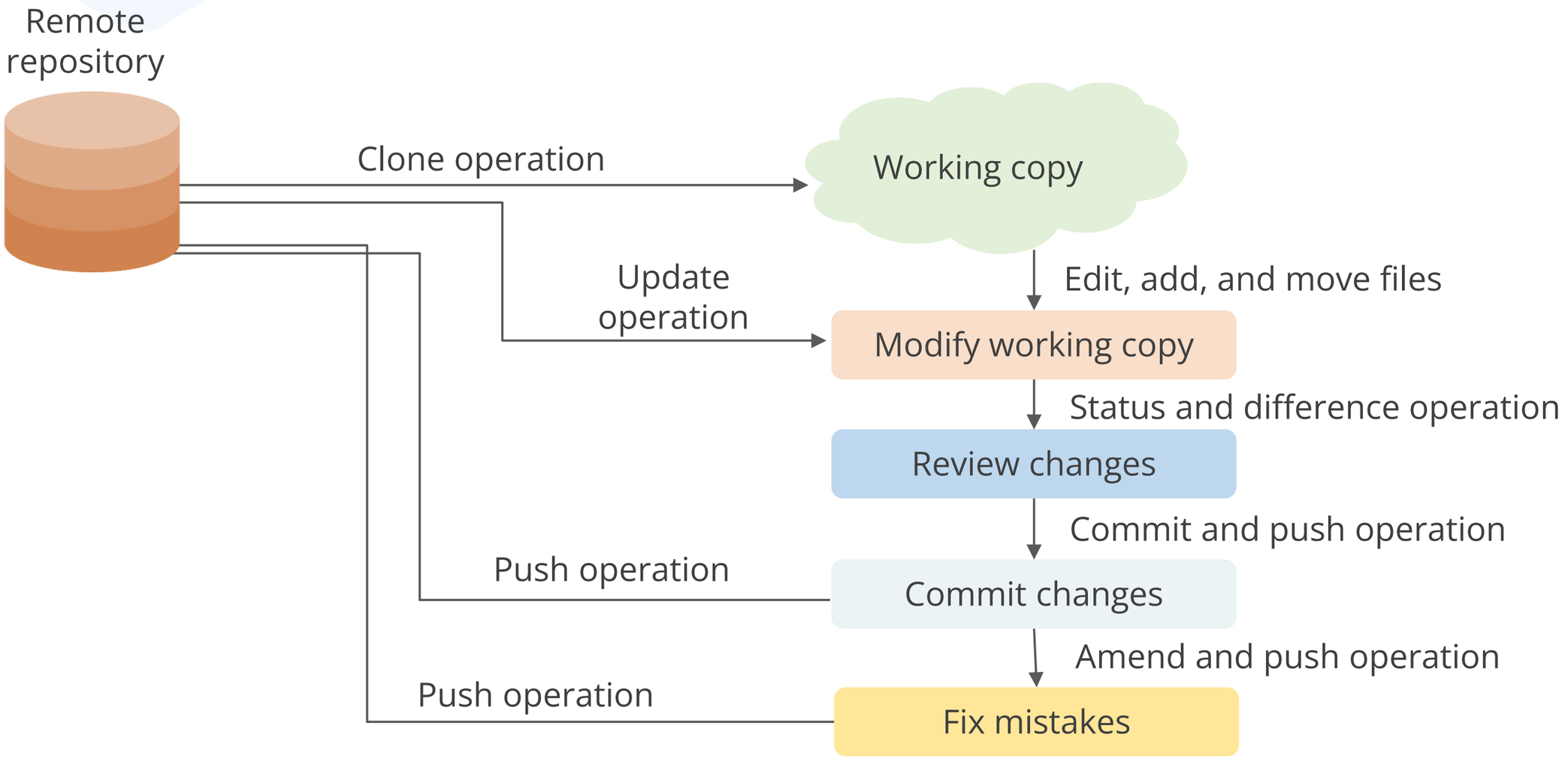
Git

- It is a version control system installed on the local system.
- It has a command line interface (CLI).
- It tracks changes in files locally.
- It offers no built-in user management.
- It is free and open-source.

GitHub

- It is a web-based online Git hosting service.
- It has both GUI and CLI.
- It tracks changes by hosting Git repositories.
- It offers built-in user management and access control.
- It has a free tier with paid plans for additional features.

Git Lifecycle



Companies Using Git

Basic Git Commands

Below are some commonly used Git commands that can perform various version control tasks:

Operation	Command
Configures the username and email address	git config --global user.name “testusername" git config --global user.emailtestname@example.com
Initializes a new Git repository in the current directory	git init
Creates a copy of a remote Git repository on a local machine	git clone [repository URL]
Commits stage changes	git add [filename]

Basic Git Commands

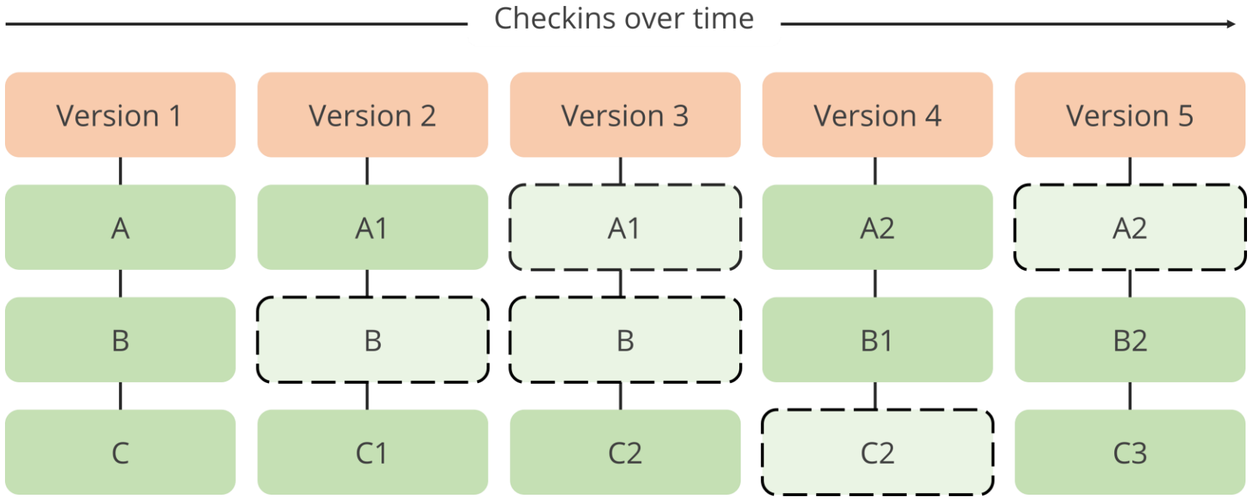
Operation	Command
Commits all stage changes at once	git add .
Records staged changes with a descriptive message	git commit -m "[commit message]"
Shows the status of your working directory	git status
Fetches changes from a remote repository and integrates them into the current branch	git pull
Uploads local branch changes to a remote repository	git push
Links local repository to a remote one	git remote add [remote name] [remote URL]

Basic Git Commands

Operation	Command
Retrieves changes from a remote repository but doesn't merge them	git fetch
Shows a list of remote repositories associated with the current project	git remote -v
Combines changes from the specified branch into the current branch	git merge [branch name]
Lists all branches in the repository, highlighting the current branch	git branch
Displays a chronological list of commits with their details	git log
Creates a named reference to a specific commit, often used for versioning	git tag [tag name]

Git Snapshots

A Git snapshot refers to a **commit**. Every time a commit occurs, it captures a complete picture of the entire project's state, including all files and staging status.

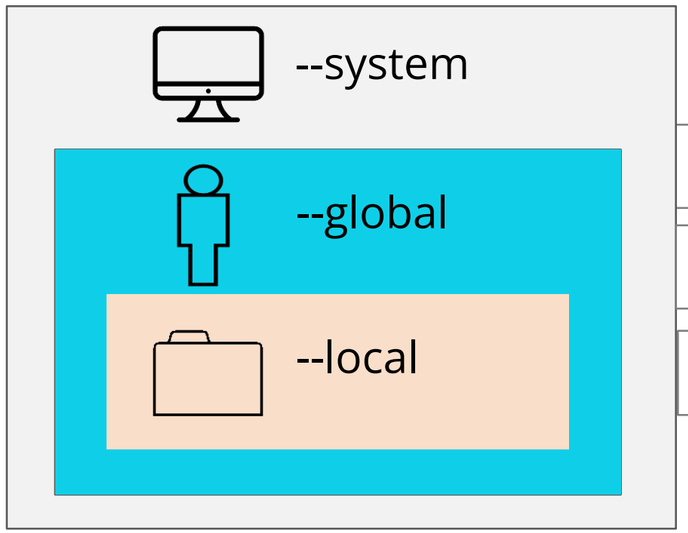


Storing data as snapshots of the project over time

It allows Git to function like a miniature file system, with a set of snapshots, distinguishing it from other VCS. These snapshots are kept in a sub-folder named **.git**.

Git Configuration Level

Git configuration levels define the scope for which specific settings apply. Following are the three main levels:



Command: *git config --system*

Command: *git config --global*

Command: *git config --local*

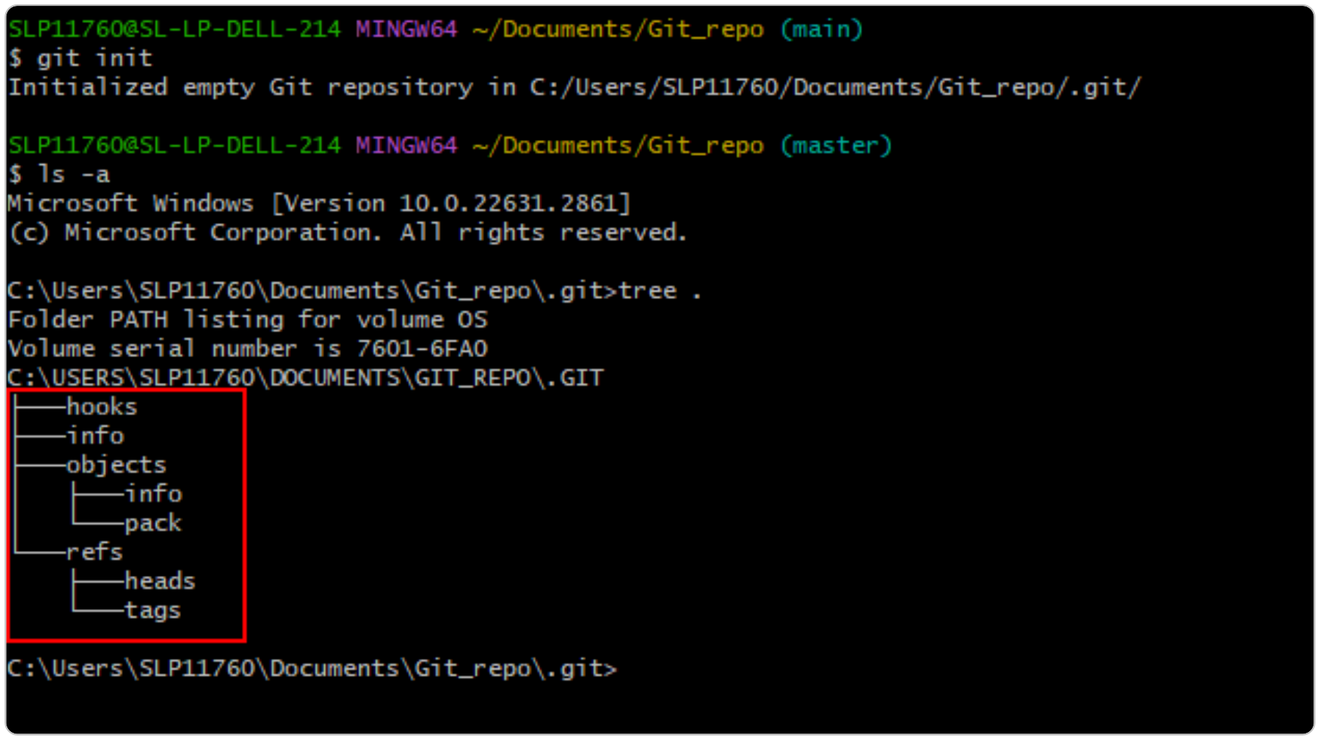
NOTE

Local overrides global and global overrides system level.

Git Repositories

A Git repository is a central storage location that keeps track of changes made to a set of files. It is initialized using the command **git init** inside the project directory.

Whenever a Git repository is initialized, a hidden **.git** folder is created inside the project directory that contains several sub-directories for version controlling:



Git Repositories

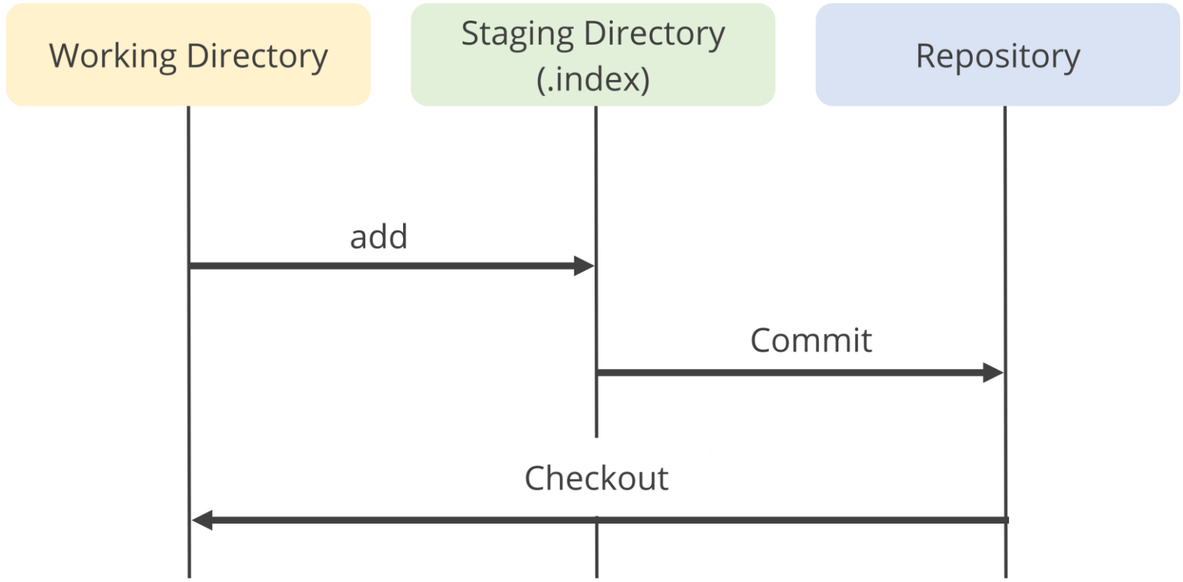
The **.git** folder contains all necessary data regarding commits, remote repository addresses, and so on.

Following are some important sub-directories:

hooks	This folder contains scripts that run at specific points of a Git workflow.
objects	This folder acts as an object database of Git that stores all the different versions of files.
refs	This folder stores references to specific commits and branches.
HEAD	The files contain references to the current branch of a Git repository.
index	This file stores staging information in a binary form.

Git Index

The Git index is the staging area or cache that acts as an intermediate zone between the working directory and the Git repository. It is a stage to select and review the changes before commit.

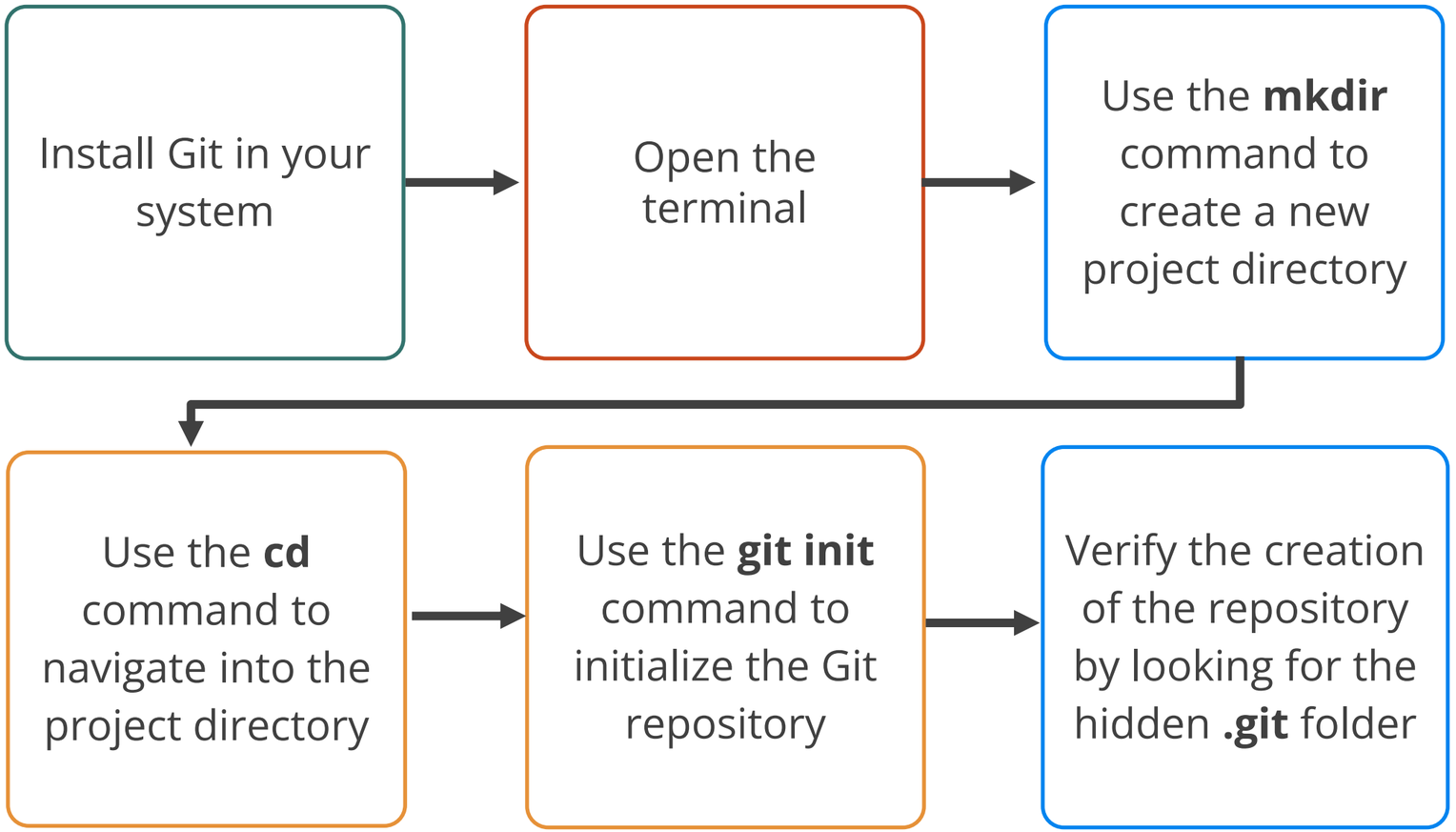


Basic workflow of file handling using Git index:

- The files are modified in the working directory.
- The *git add* command is used to stage those modified files.
- The *git commit* command is used to capture the staged changes permanently in the repository.

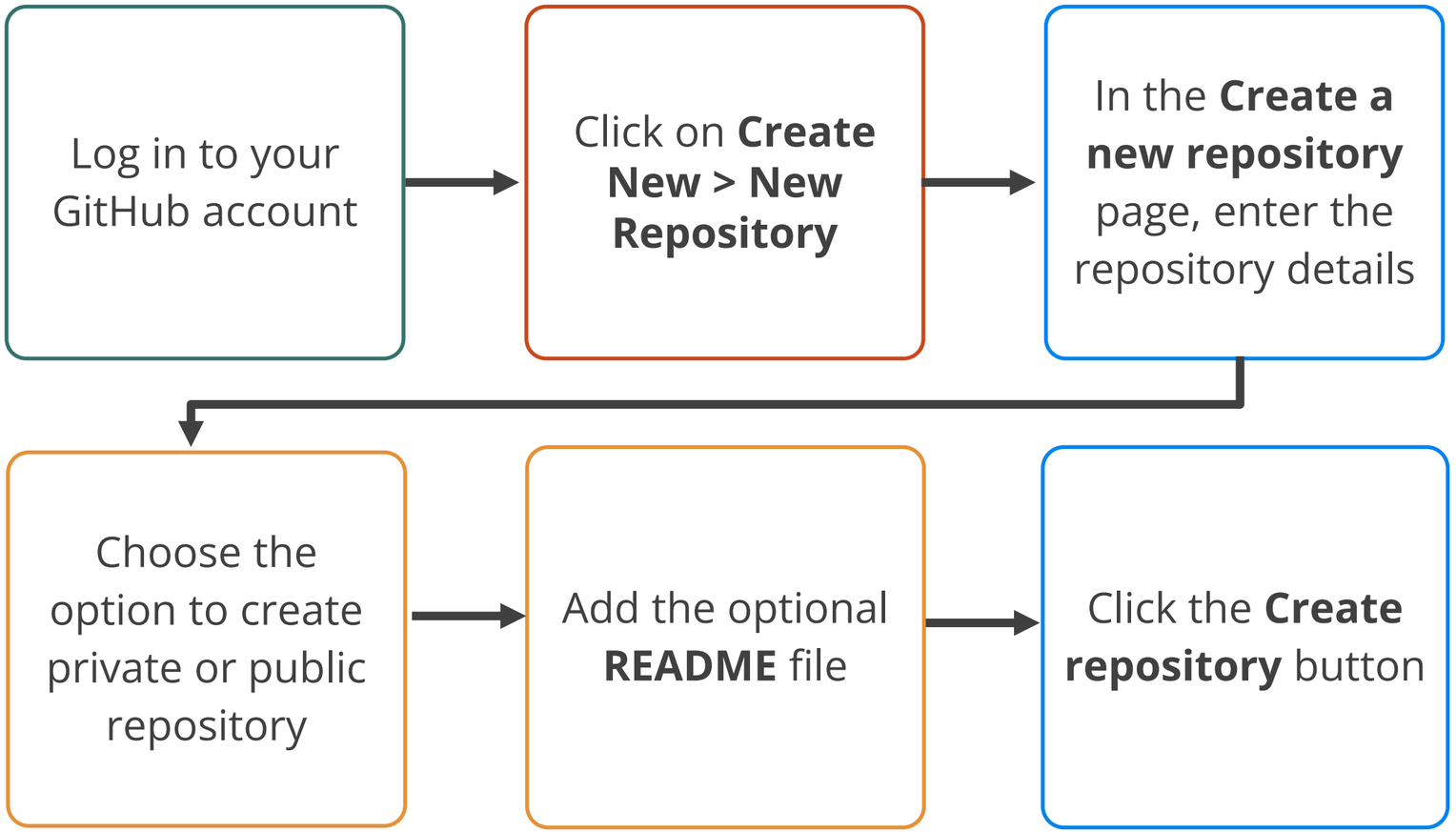
Creating a Local Git Repository

The following are the steps to create local Git repository:



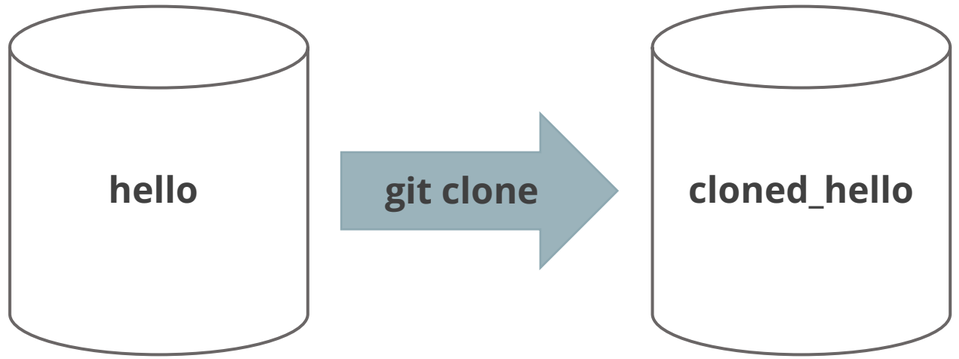
Creating a Remote GitHub Repository

Following are the steps to create a remote GitHub repository:



Cloning a Remote Git Repository

It is the process of creating a copy of an existing repository from a remote server onto the local machine to get a local repository.



Original repository

Copy of original repository

This local repository is not linked with the main remote repository; hence, it is not synchronized with the main repository until done manually.

Assisted Practice

Creating and cloning a GitHub repository **Duration: 10 Min.**

Problem statement:

You have been assigned a task to create and clone a GitHub repository for understanding the fundamentals of repository management and version control using Git and GitHub.

Outcome:

By completing this demo, you will gain comprehensive knowledge of the fundamentals of repository management and version control using Git and GitHub.

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new GitHub repository
2. Clone the GitHub repository

ASSISTED PRACTICE

Quick Check

You are collaborating on a project stored in a remote GitHub repository. Although you haven't contributed yet but want to start working on the code locally. What is the correct process to achieve this using Git?

A. Create a local Git repository and push an empty repository to GitHub

B. Clone the remote repository from GitHub and add all files and commit your initial setup

C. Fork the remote repository on GitHub and clone your forked repository to your local machine

D. Grant the write access to the remote repository for your account and download the project files directly from GitHub

Working with Remote Git Repositories

Pushing Changes into a Git Repository

Pushing changes into a Git repository refers to uploading the local code changes to a remote repository, hosted on a platform like GitHub or GitLab.

Following is the process of pushing changes into the remote Git repository from the local repository:

Make local changes

- Edit code files in the local working directory

Stage the changes

- Use *git add* command to stage the changes for commit

Commit the changes

- Run the *git commit* command to commit changes

Push Changes

- Use the *git push* command to update the changes

Git Push

Following is the basic syntax of the Git push command:

```
git push remote <branch_name>
```

Some commonly used Git push options:

- **<repository>:** Specify URL or name of destination remote repository
- **<refspec>:** Specify local branch mapping to remote repository
- **-a or --all:** Push all local branches to respective remote tracking branch
- **-u:** Set the upstream branch for the local branch
- **--tags:** Push all local tags along with their branches
- **--force:** Force the push of all the local commits
- **-n or --dry-run:** Simulate the push command without performing
- **-v or --verbose:** Provide more detailed output during the push process

Assisted Practice

Pushing a file to the GitHub repository **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate the process of pushing a file to a GitHub repository using Git commands for version control and collaboration.

Outcome:

By completing this demo, you will gain proficiency in using the Git push command for version control and collaboration. You will create a GitHub repository, set up a local repository, push changes to GitHub, and check the status of both local and remote repositories.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines

Steps to be followed:

1. Create a GitHub repository
2. Create a repository on the local machine
3. Push the changes in the local repository to GitHub
4. Check the status of the local and remote repository

Pulling from a Remote Git Repository

Pulling in Git refers to the process of downloading the latest changes from a remote repository and integrating them into the local repository.

There are two main ways to retrieve changes from a remote Git repository:

Git fetch

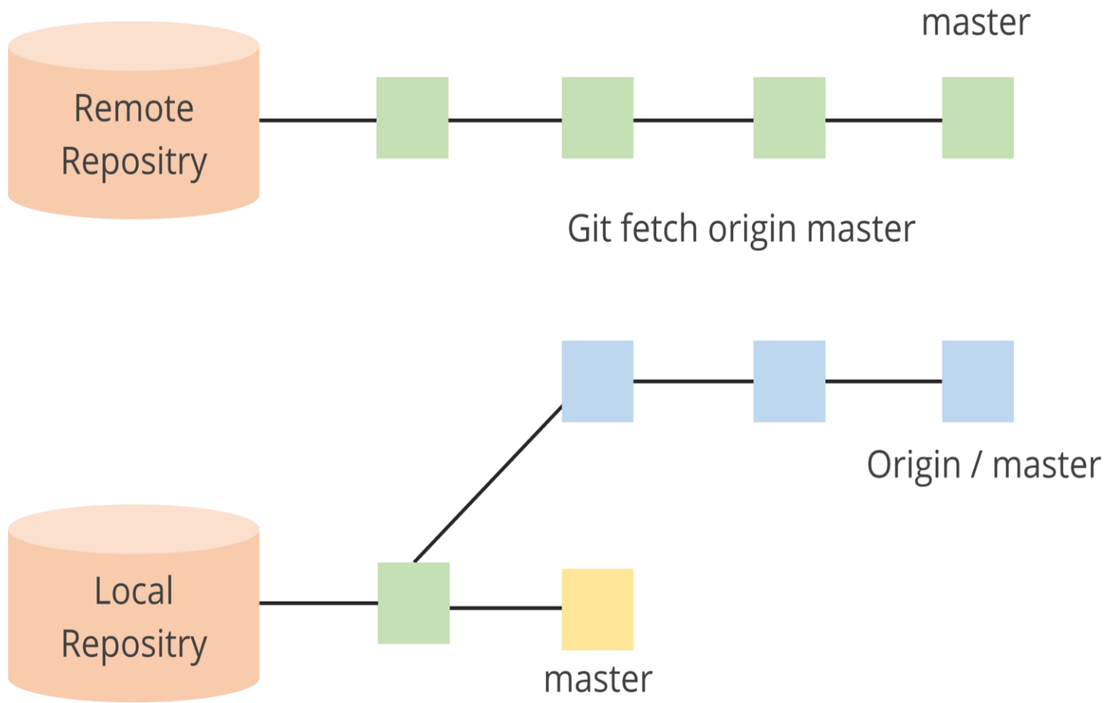
This command downloads the latest changes from the remote repository but does not automatically merge them with the local branch.

Git pull

This command downloads the changes from the remote repository and automatically merges them with the working directory.

Git Fetch

The **git fetch** command downloads the latest commits, objects, and refs from the main branch of the remote Git repository in the local **.git** directory. This does not modify the working directory.



After reviewing the remote changes, the **git merge origin/master** command can be used to merge changes in the local repository.

Git Fetch

The following are the scenarios for using the **git fetch** command:

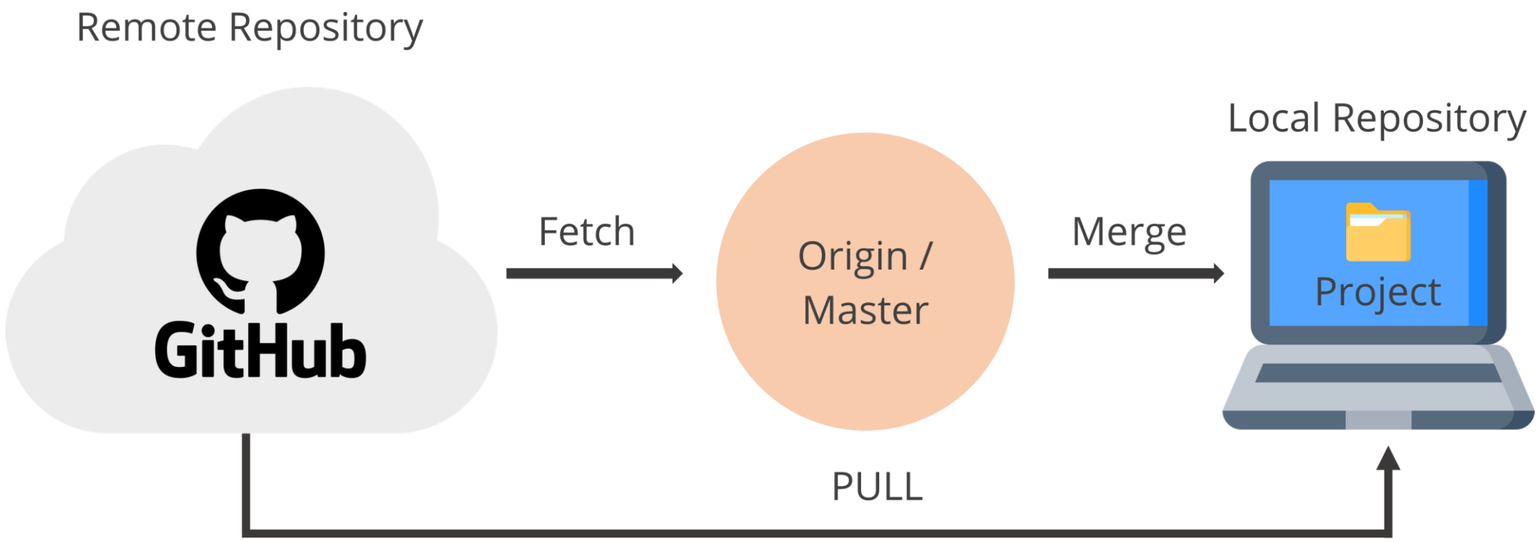
Fetch the complete remote repository	To fetch the complete remote repository of the codebase, the **git fetch <remote_repository_URL>** command is used
Fetch a specific branch	To fetch a specific branch from a repository, the **git fetch <branch_URL> <branch_name>** command is used
Fetch all branches	To fetch all the branches of a repository simultaneously, the **git fetch -all** command is used
Fetch new updates	To add new updates from the codebase to the local repository, the **git fetch origin** command is used

Git Pull

The **git pull** command integrates changes from a remote repository into the local working directory by performing fetching and merging.

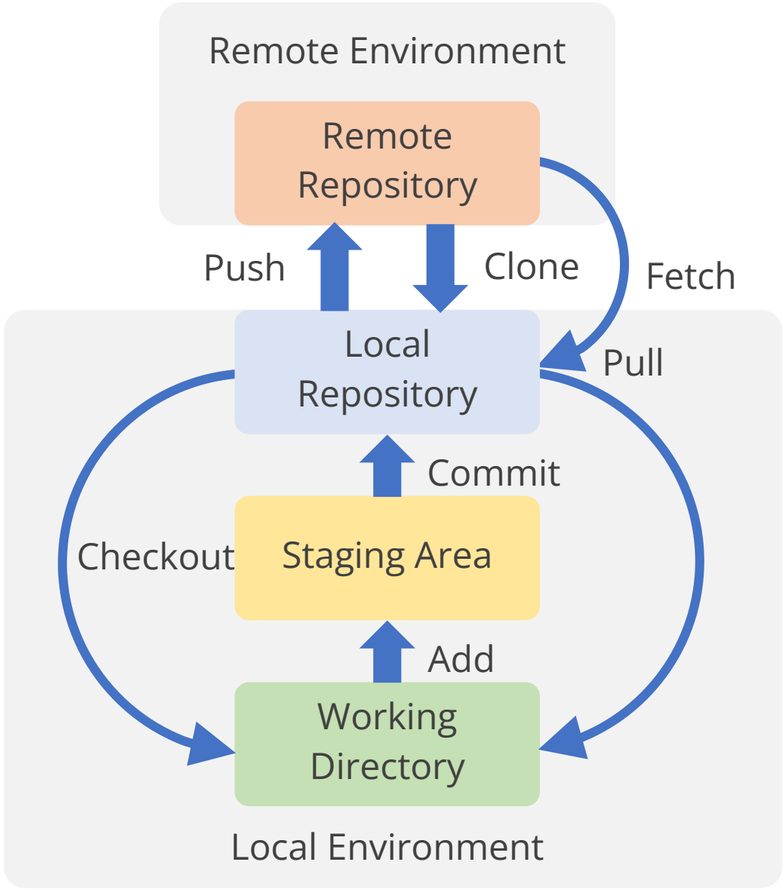
Syntax

```
git pull <option> [<repository URL><refspec>...]
```



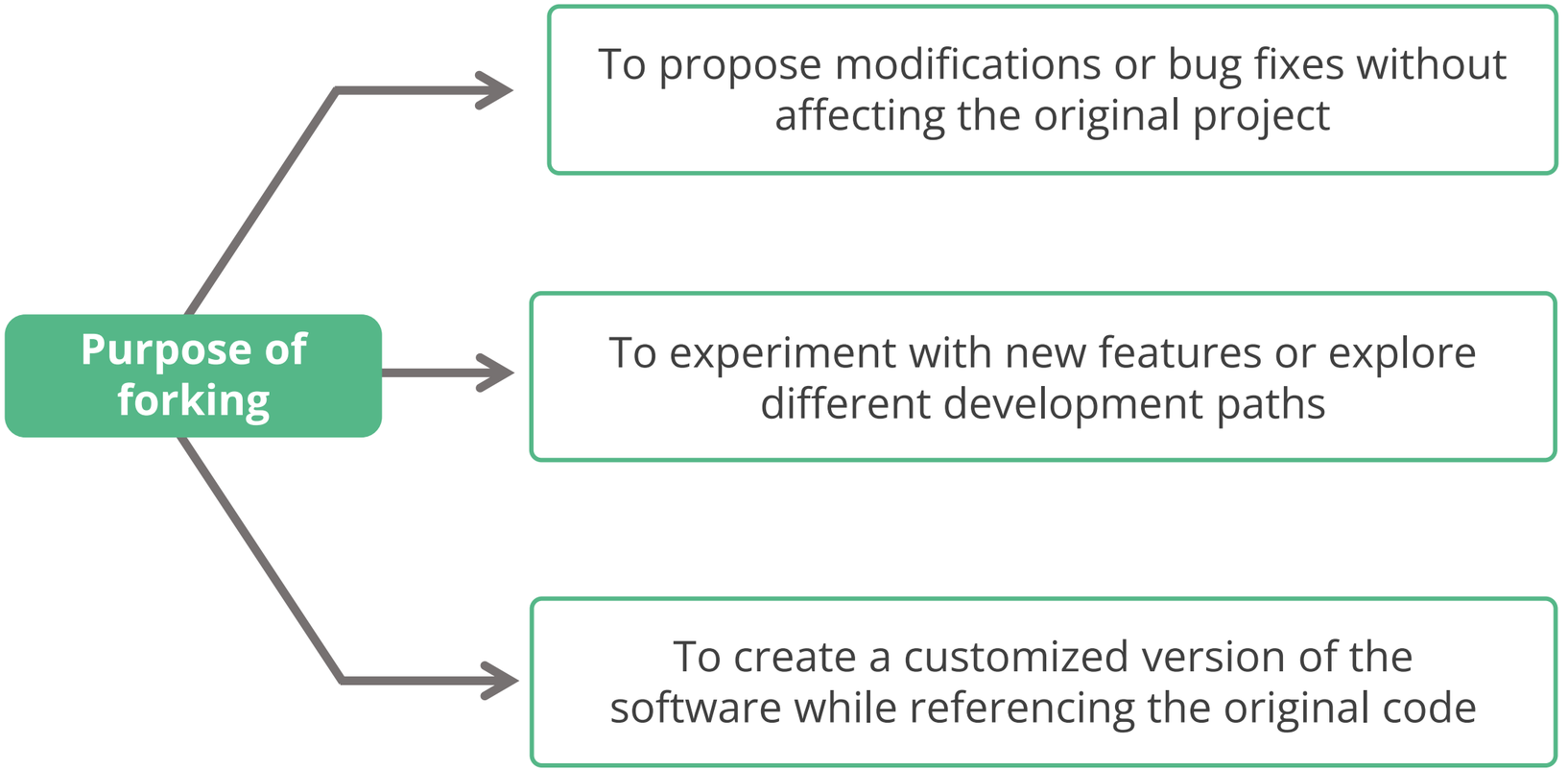
Push and Pull Workflow in Git

The following shows a typical workflow that occurs in Git for tracking and managing files using add, commit, push, and pull:



Forking in Git

It refers to creating a distinct and independent copy of a remote codebase. It allows developers to freely modify the code without affecting the original project.



Forking vs. Cloning

The following are the fundamental differences between forking and cloning a remote repository:

Forking

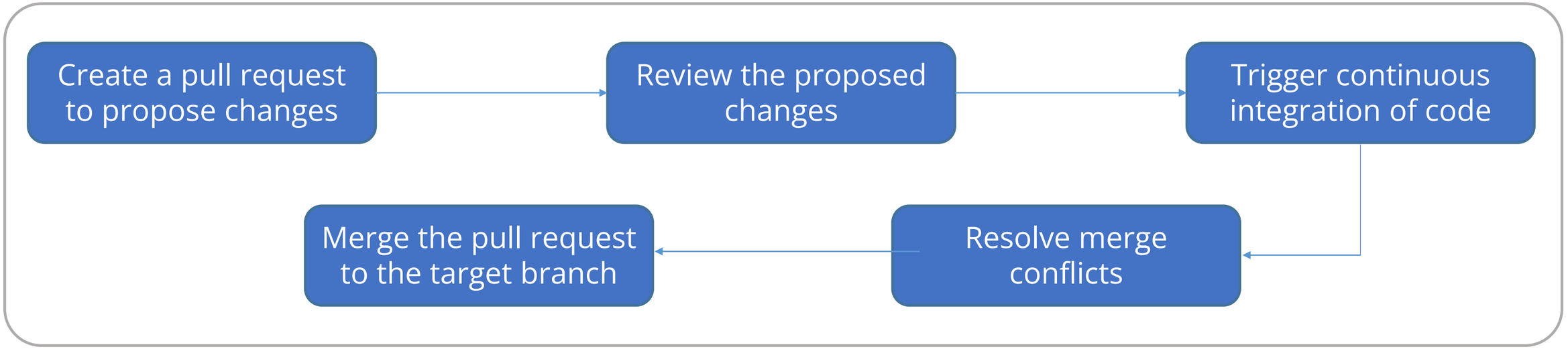
- Forking allows individuals to create a copy of the original project.
- Forking is often used in open-source development.
- Forking does not affect the original repository.

Cloning

- Cloning makes a local copy of the original project which developers do not own.
- Cloning is often used for development within an organization.
- Cloning maintains a direct connection to the original repository.

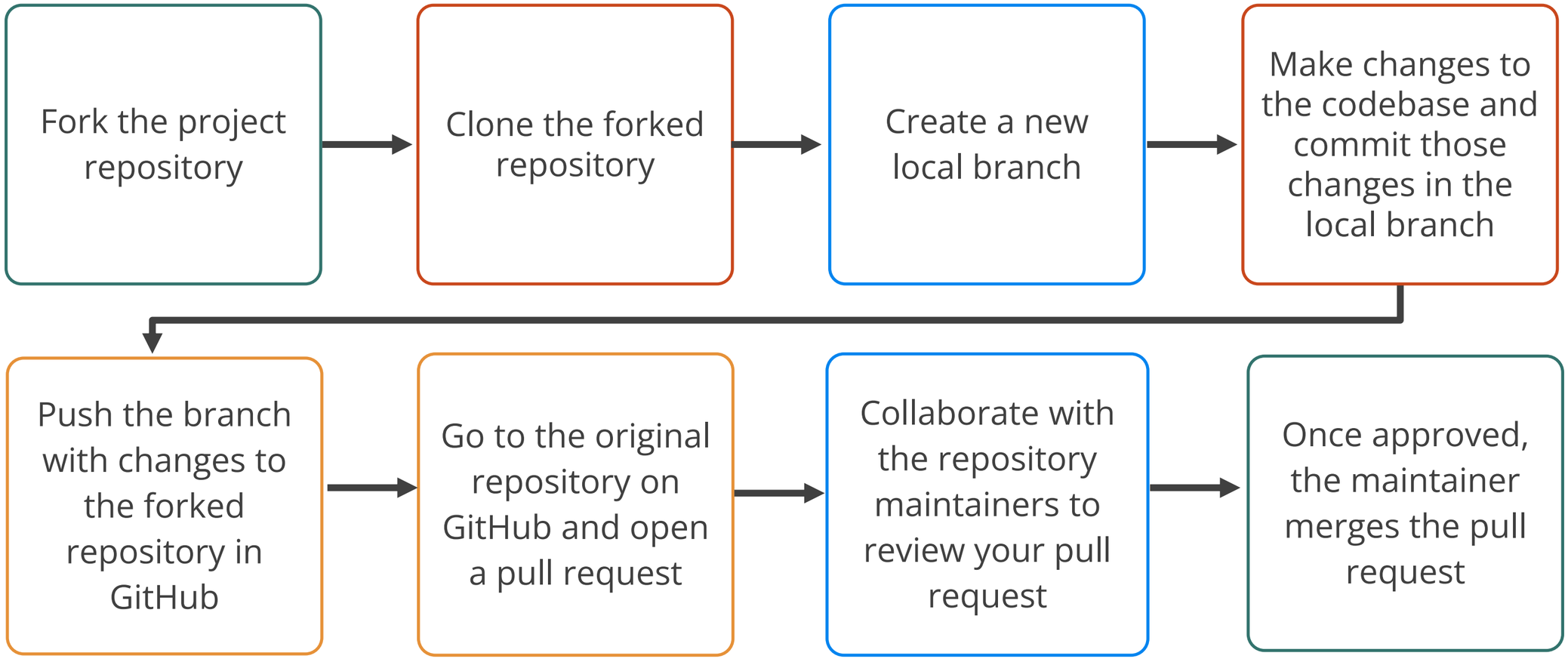
What Is a Pull Request?

A pull request in Git is a method used to propose changes made to a codebase and request that they be merged into another branch, typically the main or master branch.



Creating a Pull Request

The following are the steps to create a pull request:



Assisted Practice

Creating a pull request in Git **Duration: 10 Min.**

Problem statement:

You have been assigned a task to create a pull request for merging the local changes into a remote Git repository.

Outcome:

By completing this demo, you will gain proficiency in creating a pull request to merge local changes into a remote Git repository. You will learn how to fork a repository, clone the fork, sync it with the original repository, push changes to the fork, and create a pull request.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines

Steps to be followed:

1. Create a fork
2. Clone the fork
3. Sync the fork with the original repository
4. Push changes to the GitHub fork
5. Create a pull request

Quick Check

As a DevOps engineer, you've forked a repository to implement a new feature. After local changes, ensure your fork is up-to-date with the main repository before pushing and creating a pull request. What sequence of Git commands should you use to update your fork, push your changes, and create a pull request?

A. git pull, git push origin master, create a pull request

B. git fetch upstream, git merge upstream/master, git push origin master, create a pull request

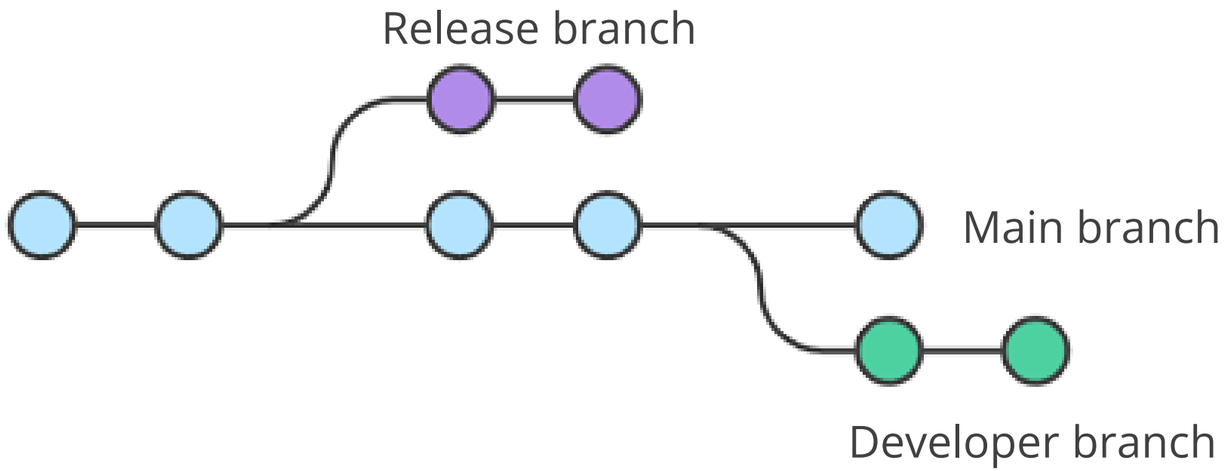
C. git clone [repository URL], git pull, git push, create a pull request

D. git fetch origin, git rebase origin/master, git push origin, create a pull request

Branching and Merging in Git

Branching in Git

A Git branch is a lightweight movable pointer to the commits. It allows developers to create isolated copies of the codebase at specific points of the development line within the project.



It enables the developers to experiment with new features, fix bugs, or collaborate on different parts of the project without affecting the main codebase.

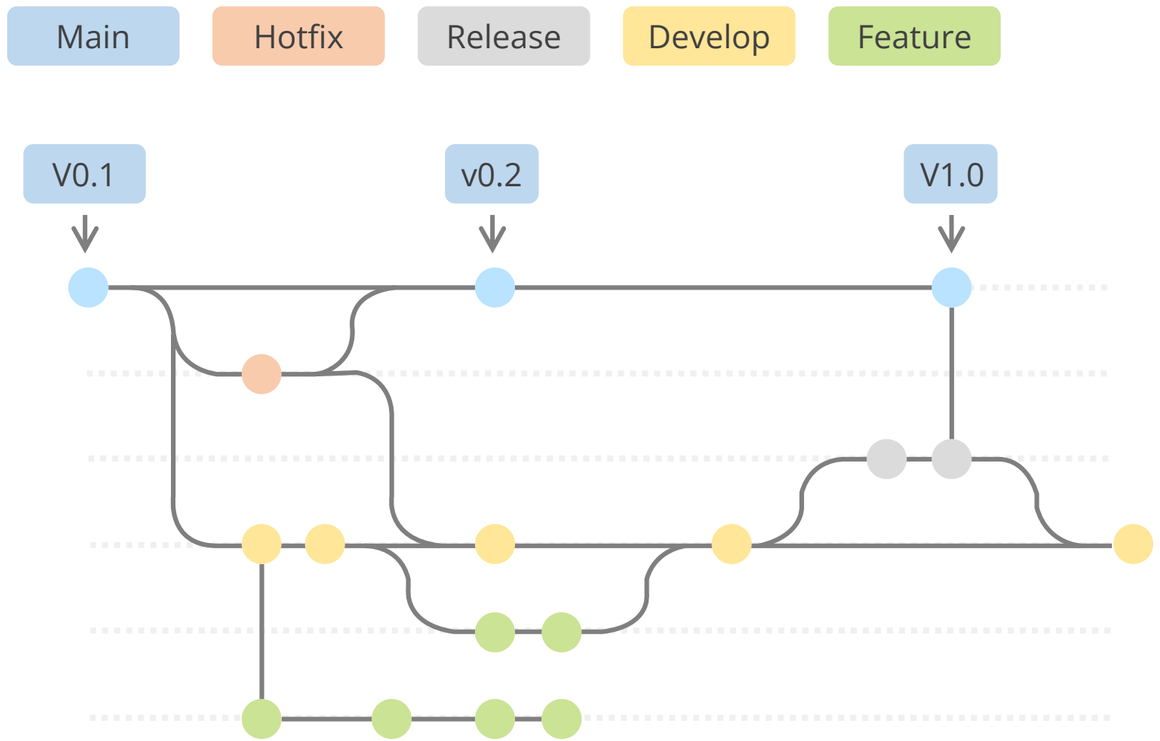
Purpose of Branching

The primary purpose of branching in Git is to create isolated environments for development within the project. Additionally, here are some key reasons to use branches:

- **Experimentation and feature development:** It provides developers with separate space to test and experiment with new software features.
- **Collaboration and parallel development:** It allows developers to work simultaneously without conflicts or overwriting.
- **Version control and code history:** It allows tracking of changes, reverting to previous versions, and understanding the evolution of the codebase.
- **Deployment strategies:** It can be used to create separate environments for deployment purposes.

Git Branching Model

A Git branching model or Git flow is a set of guidelines or conventions that define how to create, manage, and use branches in a Git workflow.



It primarily has **main, develop, feature, hotfix, and release** branches.

Git Branching Model

Following are the main branches used in a Git branching model:

Branch	Action	Purpose
Main	Mainline for stable releases	The deployment of the final code to the pre-production and production environment
Develop branch	Feature development	Developers create feature branches and submit pull requests to merge the features
Feature branch	Feature development	Developers work on features in these branches

Git Branching Model

Following are the main branches used in a Git branching model:

Branch	Action	Purpose
Release branch	Release preparation	The release branch from the develop branch is used for staging deployment and release preparation
Bugfix branch (if needed)	Bug fixing	If bugs are found in staging, a bugfix branch is created from the release branch to fix the bugs.
Hotfix branch (if needed)	Hotfix bugs	The issues found in pre-production or production are fixed in the hotfix branch.

Operations in Branching

Operations	Description
Create a branch	The first step is to create a branch from the main branch of the codebase.
Merge a branch	The merging of branches is done after working on a specific feature.
Delete a branch	An existing branch can be deleted when the branch has done its job.
Checkout a branch	It refers to switching from one branch to another in a remote repository.

Branch Commands

$ git branch <branch name>	Creates a new branch
$ git branch	Lists all branches in the current repository
$ git checkout <branch name>	Switches to different branches
$ git merge <branch name>	Merges two branches

NOTE

To create a new branch and switch to it, execute: *$ git checkout -b<branch-name>*

Assisted Practice

Creating a branch in Git **Duration: 10 Min.**

Problem statement:

You have been assigned a task to create a branch in a Git repository to facilitate a collaborative and organized approach to software development

Outcome:

By completing this demo, you will gain proficiency in facilitating a collaborative and organized approach to software development. You will create and manage branches in a Git repository, including creating a new GitHub repository, cloning it, listing all branches, creating a new branch, and deleting a branch.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new GitHub repository
2. Clone the GitHub repository
3. List all the branches in your repository
4. Create a new branch
5. Delete the branch

ASSISTED PRACTICE

Switching Branches in Git

Switching branches in Git is a fundamental concept for managing the development workflow. It isolates the changes that the developer is working on and seamlessly switches between different functionalities.

There are two Git commands that are used to switch between branches:

git checkout

This command enables the switching of branches by updating the files in the working tree to match the version stored in the desired branch.

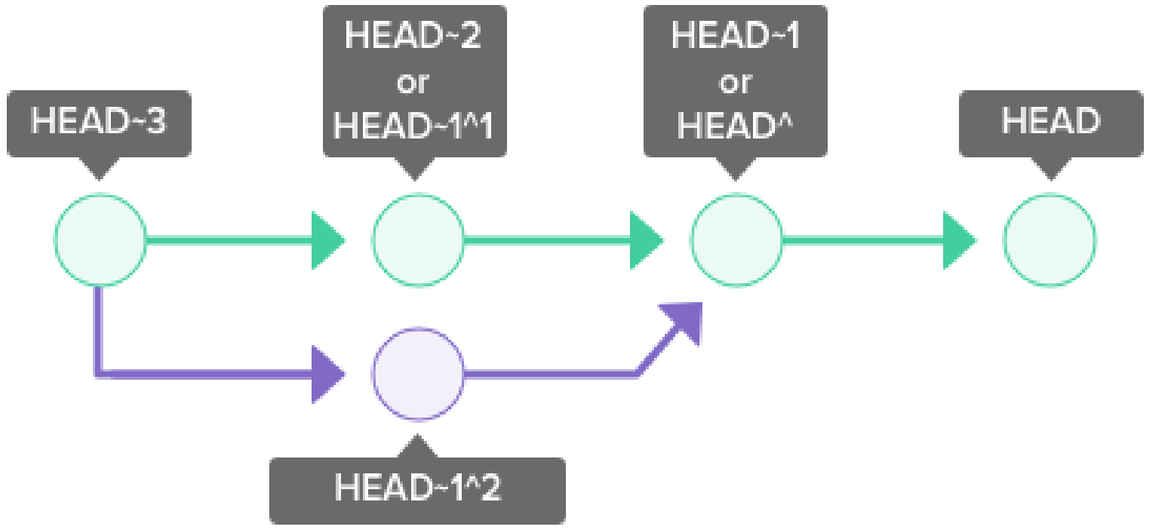
git switch

This command enables switching the current HEAD branch. It's a relatively new feature, offering a simpler alternative to the traditional checkout command.

Switching Branches in Git

Git HEAD

HEAD is used to represent the current snapshot of a branch. For a new repository, Git will, by default, point HEAD to the master branch. Changing where HEAD is pointing will update your current active branch.



Git HEAD in branching

Switching Branches in Git

The following are the Git commands to switch between branches within a Git repository:

Git operations	git checkout	git switch
To switch from one branch to another	**git checkout <branchname>**	**git switch <branchname>**
To create a new branch and switch to it	**git checkout -b <branchname>**	**git switch -c <branchname>**

Assisted Practice

Switching branches in Git **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate how to switch branches in Git to understand the branch management and version control workflows in GitHub.

Outcome:

By completing this demo, you will gain proficiency in understanding branch management and version control workflows in GitHub. You will create a new repository, clone it, list all branches, create and switch to a new branch, commit changes, and switch back to the main branch.

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new GitHub repository
2. Clone the GitHub repository
3. List all the branches in your repository
4. Create and switch to the new branch
5. Create a file and commit the changes
6. Switch back to the main branch

Quick Check

Imagine you are working on a software development project using Git for version control. You've just been tasked to add a new feature to the application, and you want to create a separate branch to work on this feature without affecting the main codebase.
Which Git command would you use to perform the creation of the new branch and switch to it in a single step?

A. git clone

B. git branch

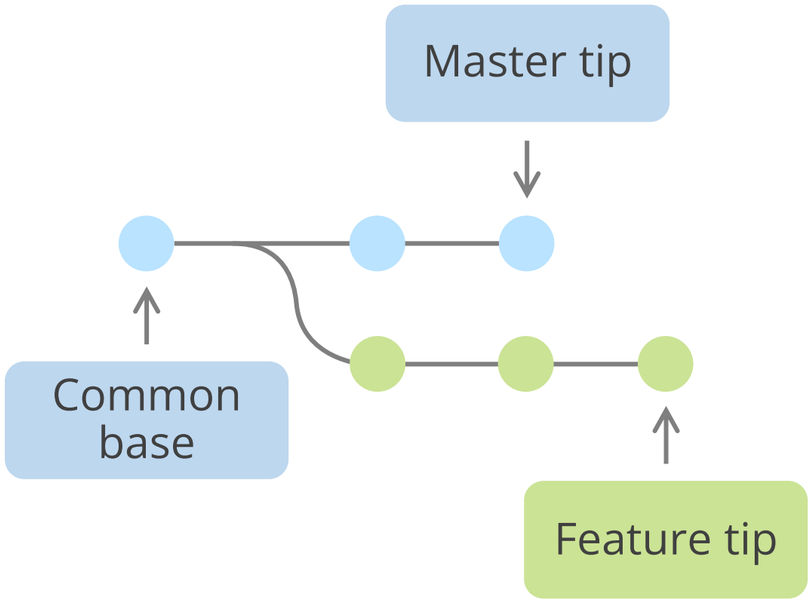
C. git checkout -b

D. git merge

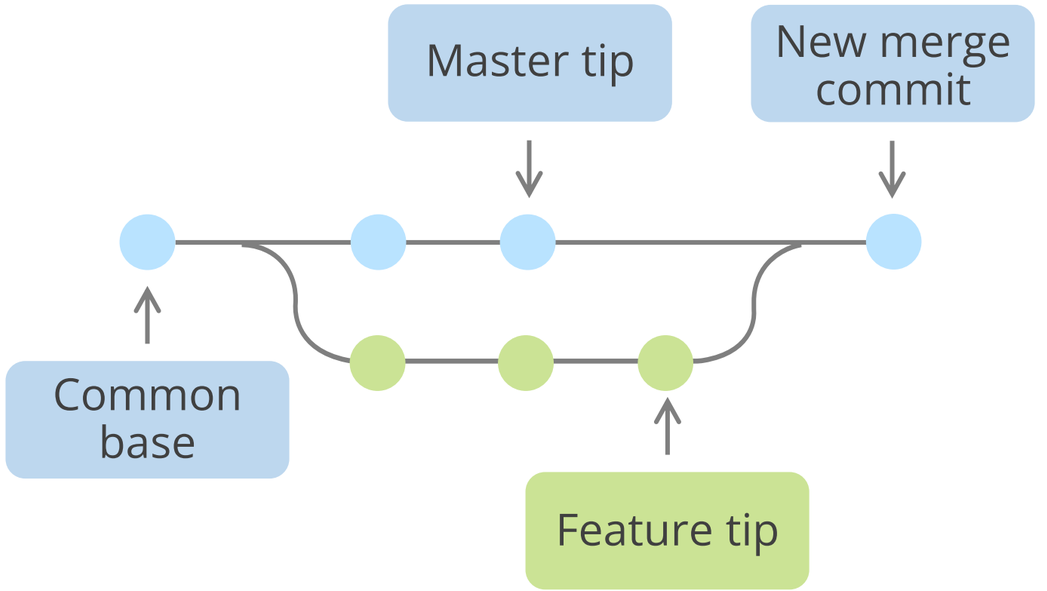
Merging Branches in Git

Merging branches in Git is a technique for integrating changes from different development lines into a single branch.

The Git merge combines multiple sequences of commits into one unified history. In most of the cases, git merge is used to combine two branches.



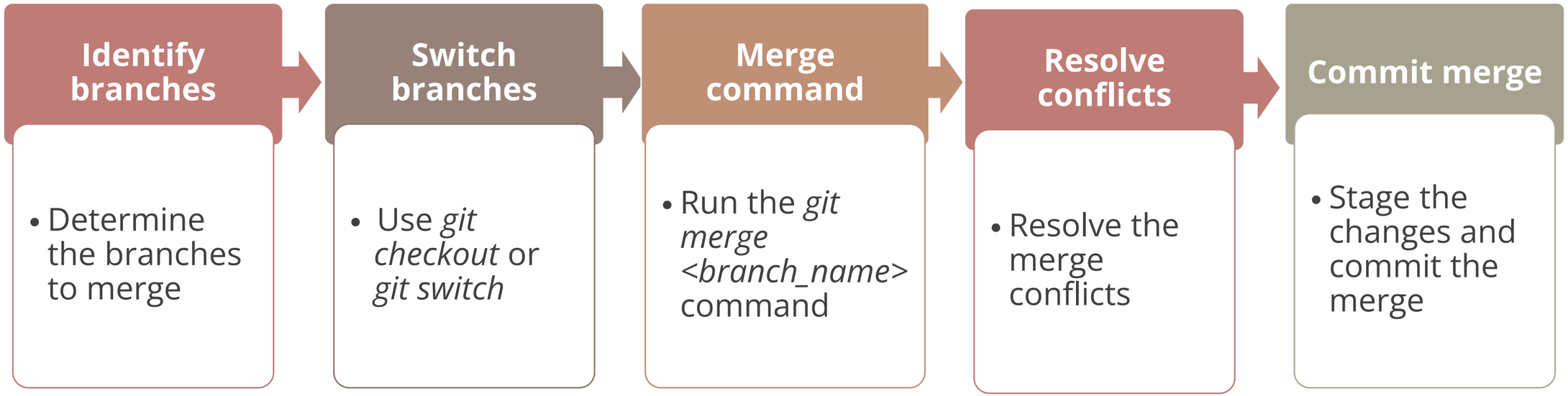
Before merging



After merging

Merging Branches in Git

The following shows the process of merging branches in a Git repository:



Assisted Practice

Merging branches in Git **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate how to merge branches in Git to integrate changes from one branch into another while maintaining a cohesive codebase and version history

Outcome:

By completing this demo, you will gain proficiency in merging branches within a Git repository. You will create a new GitHub repository, clone it, list branches, create and switch to a new branch, make and commit changes, check branch status, switch back to the main branch, and merge branches.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new GitHub repository
2. Clone the GitHub repository
3. List all the branches in your repository
4. Create and switch to the new branch
5. Create a file and commit the changes
6. Check the status of the new branch
7. Switch back to the main branch
8. Merge the branches

Merge Conflicts in Git

Merging branches in Git integrates changes into the codebase, but sometimes conflicts arise when different branches modify the same lines of code.

Conflicts occur when Git encounters changes made to the same lines of code in separate branches while merging.

Reasons for merge conflicts

Git cannot automatically decide which version to keep, so it flags the conflict and requires manual intervention.

Assisted Practice

Resolving merge conflicts in Git **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate how to merge branches in Git to integrate changes from one branch into another while maintaining a cohesive codebase and version history.

Outcome:

By completing this demo, you will gain proficiency in seamlessly integrating branch changes in Git, ensuring a cohesive codebase and version history.

Note: Refer to the demo document for detailed steps

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new GitHub repository
2. Clone the GitHub repository
3. List all the branches in your repository
4. Create and switch to the new branch
5. Create a file and commit the changes
6. Check the status of the new branch
7. Switch back to the main branch
8. Merge the branches

Assisted Practice

Resolving merge conflicts on file modifications **Duration: 10 Min.**

Problem statement:

You have been assigned a task to demonstrate the resolution of merge conflicts when resolving conflicts in Git when the same file is modified on different branches.

Outcome:

By completing this demo, you will be able to resolve merge conflicts in Git when modifications to the same file occur on different branches, ensuring seamless integration of changes

Note: Refer to the demo document for detailed steps

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create a new branch
2. Create a new file and commit changes
3. Switch back to the main branch
4. Make different changes to the same file
5. Merge the branches and resolve the conflict

Quick Check

You've been working on a bug fix in a separate branch that is ready to merge into the **develop** branch. After running **git merge develop**, the terminal displays a bunch of code with markers like **<<<<<<<** and **>>>>>>>**. What should be done next?

A. These markers indicate a successful merge. Commit the changes with a descriptive message.

B. Stash the uncommitted changes from the bug fix branch and switch back to the develop branch.

C. Manually edit the files to resolve the conflicts, remove the markers, and then commit the merge.

D. Ignore the markers and commit the merge anyway. Most likely they won't cause any issues.

Key Takeaways

- A source code management system is used to track modifications to a source code repository and facilitate collaboration among development teams.
- Version control systems track changes to files over time, enabling collaboration and rollback.
- Git is an open-source distributed version control system used for tracking changes in source code during software development.
- Git snapshots are like a miniature file system, with a set of snapshots of every commit.
- A Git repository serves as a central storage location that records and tracks changes made to a collection of files.
- The primary purpose of branching in Git is to create isolated environments for development within the project.

Implementing Basics of Version Control System with Git

Duration: 25 Min.

Project agenda: To create a remote GitHub repository and use Git commands to perform various version control operations for managing the remote repository

Description: You work as a developer in an IT firm. Your company is undertaking a project that consists of three modules, and you have been asked to work on one of these modules. You have also been instructed to upload all the project files to the GitHub repository. To avoid impacting the main codebase, create a new branch and conduct your work there. Once your module is complete, merge it into the main branch.

LESSON-END PROJECT

Implementing Basics of Version Control System with Git

Duration: 25 Min.

Perform the following:

1. Create a new repository
2. Clone the GitHub repository
3. Create a new branch and verify it
4. Rename the existing branch and list all the branches
5. Create a new branch and switch to the new branch
6. Create a file, commit the changes, and check the status of the new branch
7. Delete the branch and verify it
8. Switch back to the main branch and merge

Expected deliverables: A step-by-step guide to creating, managing, and deleting branches and files in a Git repository, including scripts and commands to streamline version control for collaborative development

LESSON-END PROJECT

Thank You