Version Control System

Engage and Think

Imagine you and your friends are working on a group project, like writing a document for a competition. Each person is responsible for different sections, but when you combine everything, you realize some changes are missing and others have been overwritten. You are frustrated because now you must manually fix everything and figure out who changed what.

How can multiple people work on the same project without losing changes or overwriting each other's work?

Learning Objectives

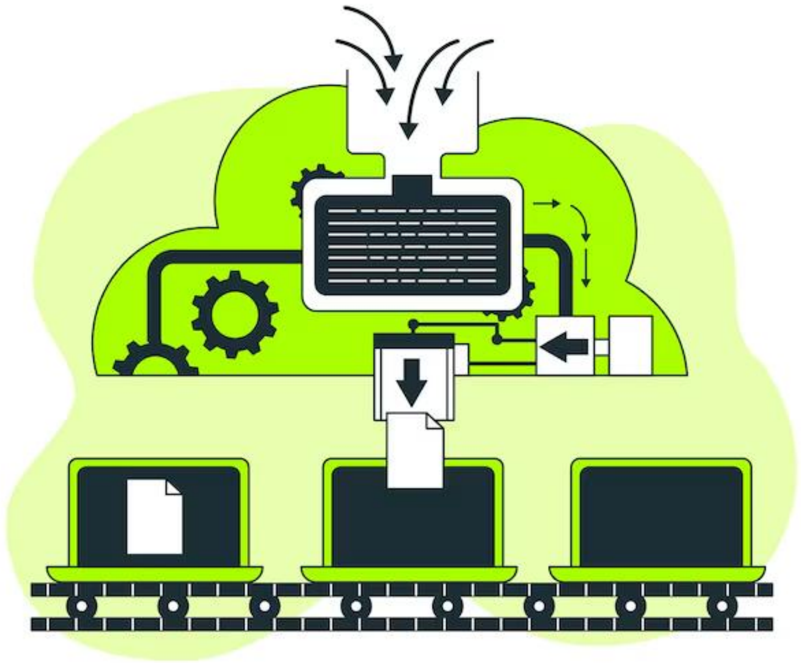
By the end of this lesson, you will be able to:

- Differentiate between Git, GitHub, GitLab, Bitbucket, and Subversion based on their functionality and use cases
- Apply fundamental Git operations in a repository, such as initializing repositories, staging files, committing changes, and checking status
- Implement branching strategies, switch between branches, and merge changes within a Git workflow to streamline development
- Utilize Git and GitHub features such as forking, pull requests, and issue tracking to manage team projects efficiently
- Configure Git settings and manage user credentials based on different configuration levels
- Implement best practices such as frequent commits, writing clear commit messages, and reviewing changes before pushing updates

Introduction to Version Control System

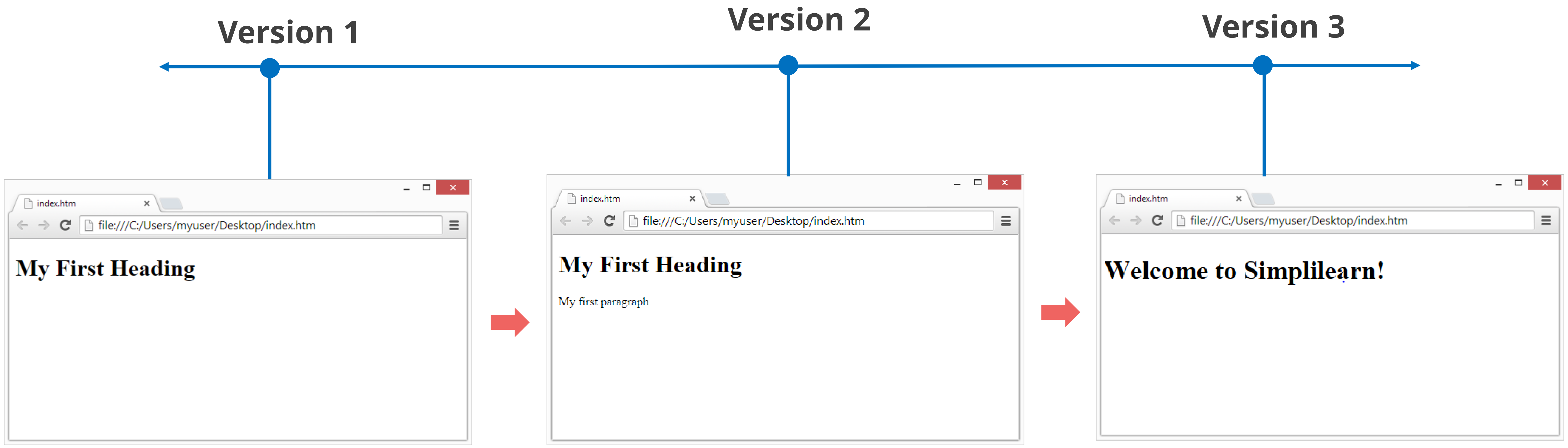
What Is a Version Control System?

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



It enables multiple users to collaborate, maintain a history of modifications, and revert to previous versions if needed.

Version Control Systems: Example



Version 1

index.htm

file:///C:/Users/myuser/Desktop/index.htm

My First Heading

Created

Version 2

index.htm

file:///C:/Users/myuser/Desktop/index.htm

My First Heading

My first paragraph.

Modified

Version 3

index.htm

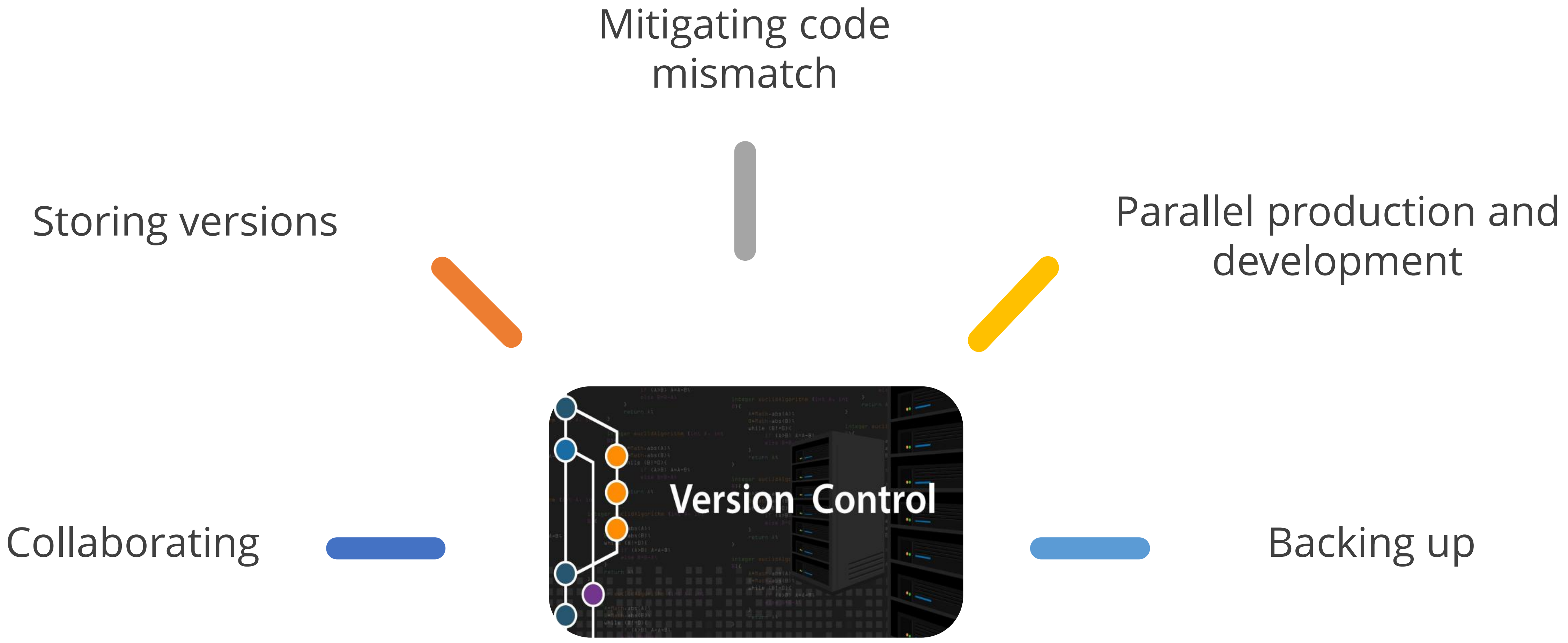
file:///C:/Users/myuser/Desktop/index.htm

Welcome to Simplilearn!

Modified

Version Control System: Benefits

Some benefits of the version control system are:



Mitigating code mismatch

Storing versions

Parallel production and development

Collaborating

Backing up

A version control system plays a crucial role in Source Code Management (SCM), ensuring efficient tracking, collaboration, and maintenance of code changes across software projects.

Source Code Management

It is a process used in software development to track, manage, and control changes to source code. It helps developers collaborate efficiently, maintain a history of changes, and ensure code integrity.

It helps teams maintain a structured development process, improve code quality, and accelerate project timelines.

Source Code Management: Purpose

The process of monitoring and managing modifications to a software application's source code is known as SCM (Source Code Management).

Below are the main purposes of SCM:

It keeps track of all code changes, allowing developers to revert to previous versions when needed.

It tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors.

It supports the creation of separate branches for different features, bug fixes, or experiments.

Source Code Management: Best Practices

- **Commit often**: Save changes frequently to track progress and prevent data loss

- **Ensure that the user is working on the latest version**: Pull the latest updates before making changes

- **Make detailed notes**: Write clear commit messages to document changes effectively

- **Review changes before committing**: Verify modifications to maintain code quality and avoid errors

Source Code Management: Tools

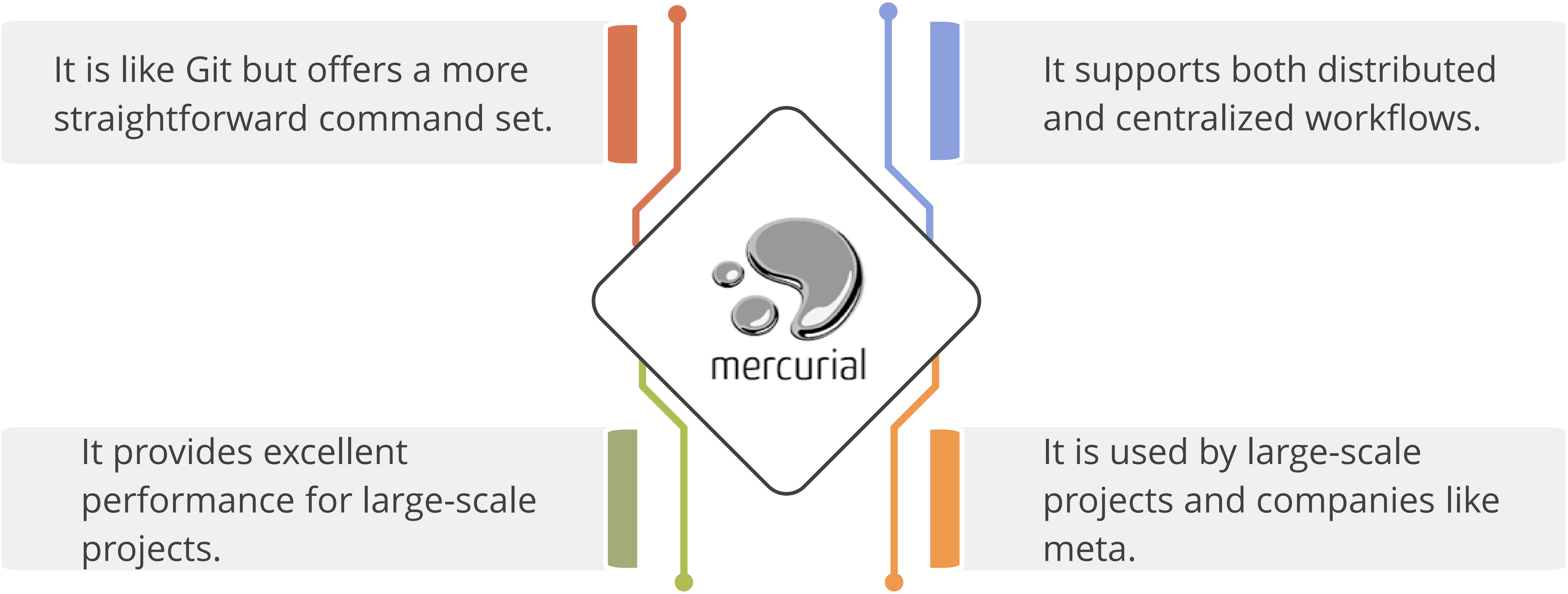
SCM tools enable teams to work simultaneously on a project without overwriting each other's code. Some of the preferred tools for source code management are:

Note: The next section introduces various Source Code Management (SCM) tools, including Mercurial, Bitbucket, and Subversion. However, Git is not covered here, as it will be explained in detail in the upcoming topics.

Introduction to Mercurial

Mercurial is a distributed version control system (DVCS) tool designed for high-performance and simplicity.

Here are the key features of Mercurial:



It is like Git but offers a more straightforward command set.

It supports both distributed and centralized workflows.

It provides excellent performance for large-scale projects.

It is used by large-scale projects and companies like meta.

Introduction to Bitbucket

It is a Git-based source code management (SCM) tool that enables teams to collaborate, manage repositories, and integrate with CI/CD pipelines.

Here are the key features of Bitbucket:

Access control to restrict access to the source code

Pull requests with in-line commenting for collaboration on code review

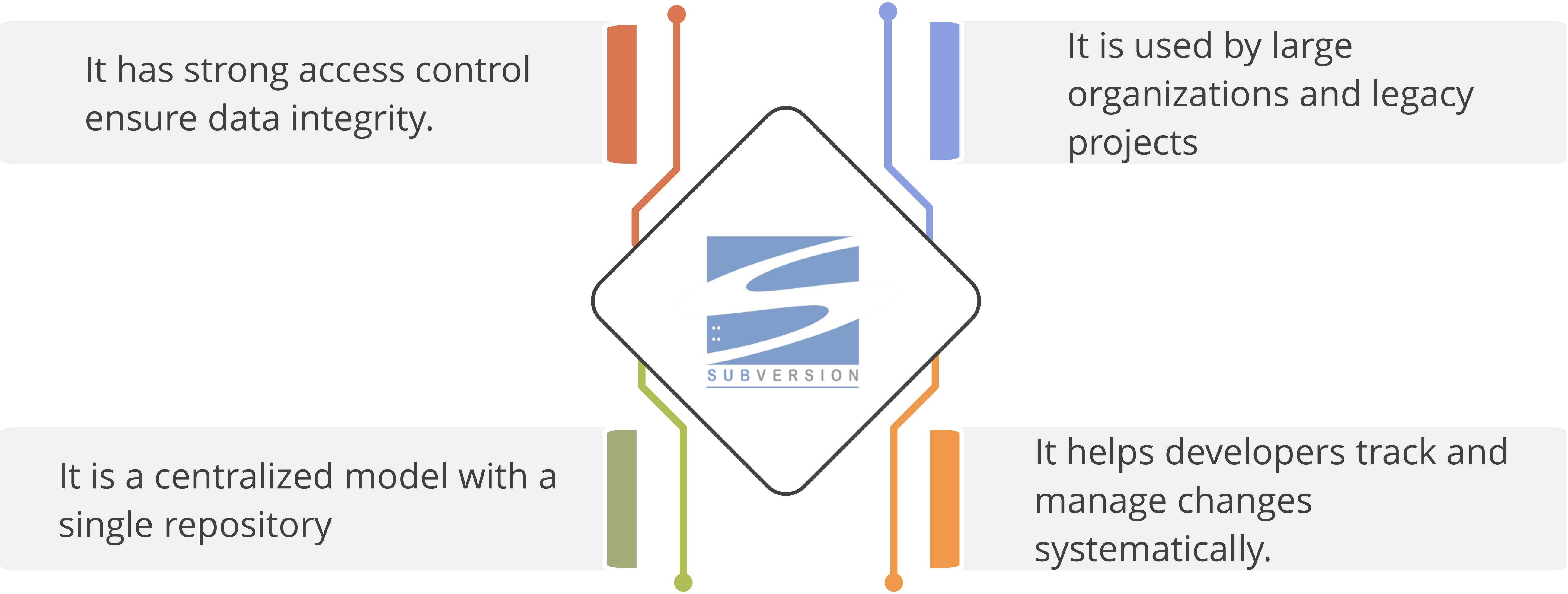
Workflow control to enforce a project or team workflow

Jira integration for full development traceability

Introduction to Subversion (SVN)

Subversion (SVN) is a centralized version control system (CVCS) tool that helps developers track file changes in a structured manner.

Here are the key features of Subversion:



It has strong access control ensure data integrity.

It is used by large organizations and legacy projects

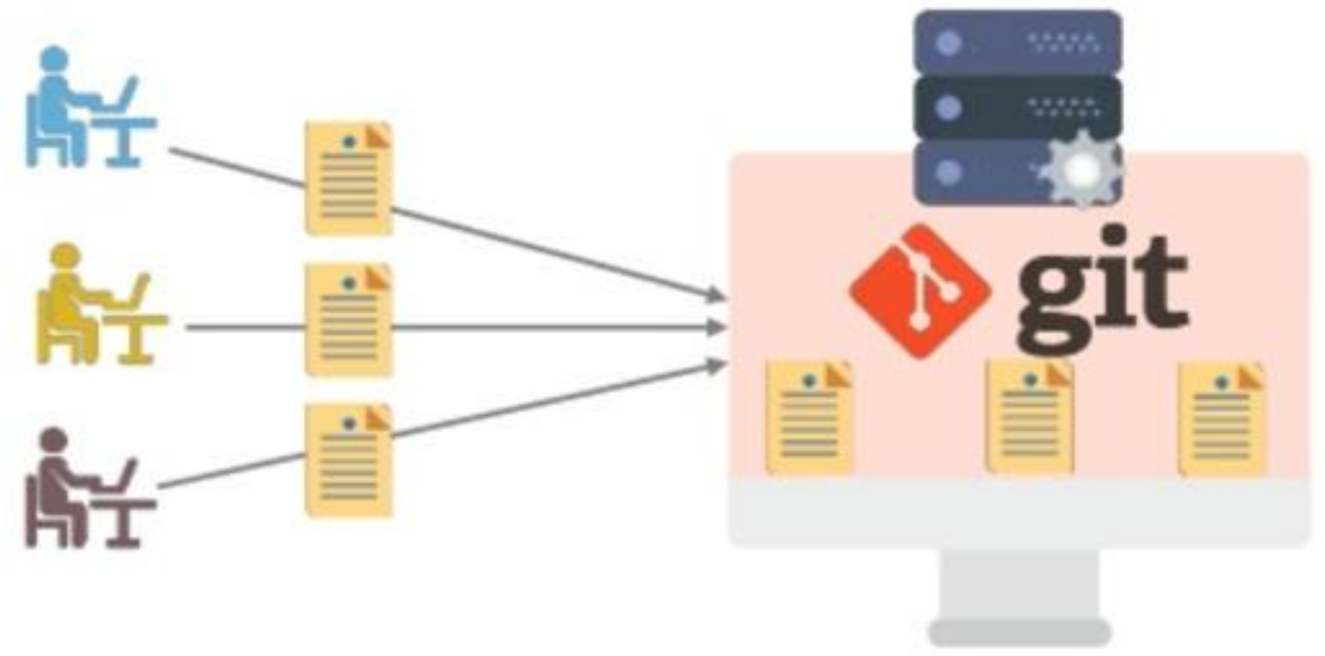
It is a centralized model with a single repository

It helps developers track and manage changes systematically.

Understanding the Working of Git

What Is Git?

Git is a version control system for tracking changes in computer files. It is generally used for source code management in software development.



Developers

Git server

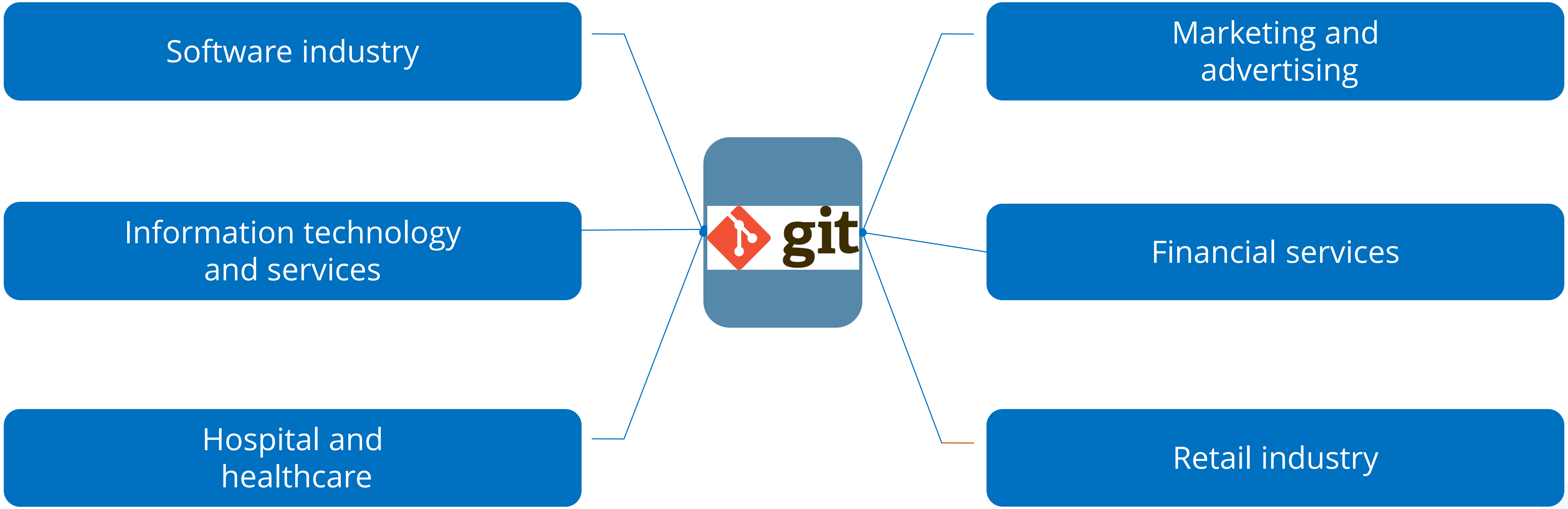
Git: Purpose

Below are the reasons to use Git:

- Tracks changes in the source code
- Uses distributed version control tool for source code management
- Allows multiple developers to work together
- Supports non-linear development because of its several parallel branches

Who Uses Git?

Git is widely used across industries and by various professionals involved in:



Software industry

Information technology and services

Hospital and healthcare

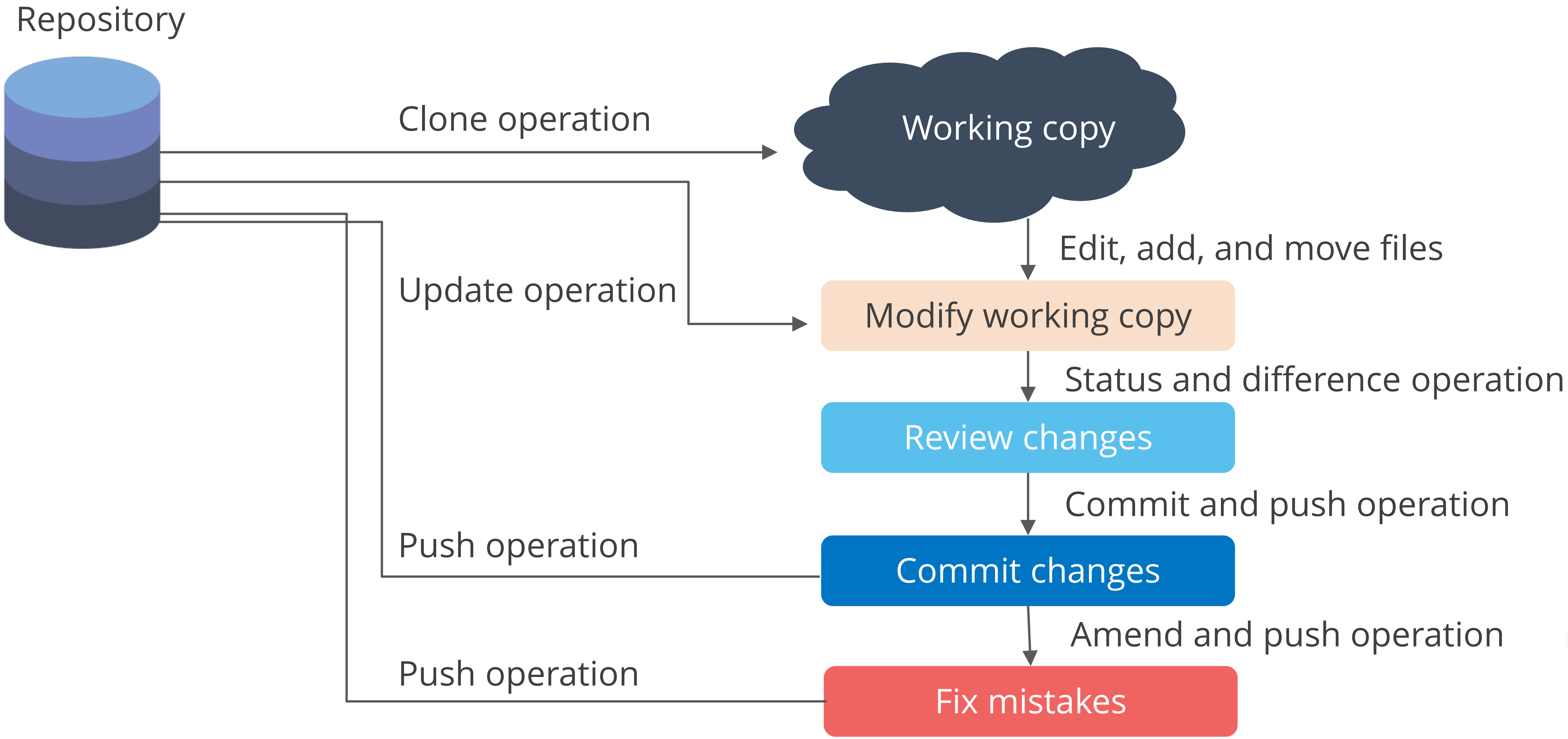
Marketing and advertising

Financial services

Retail industry

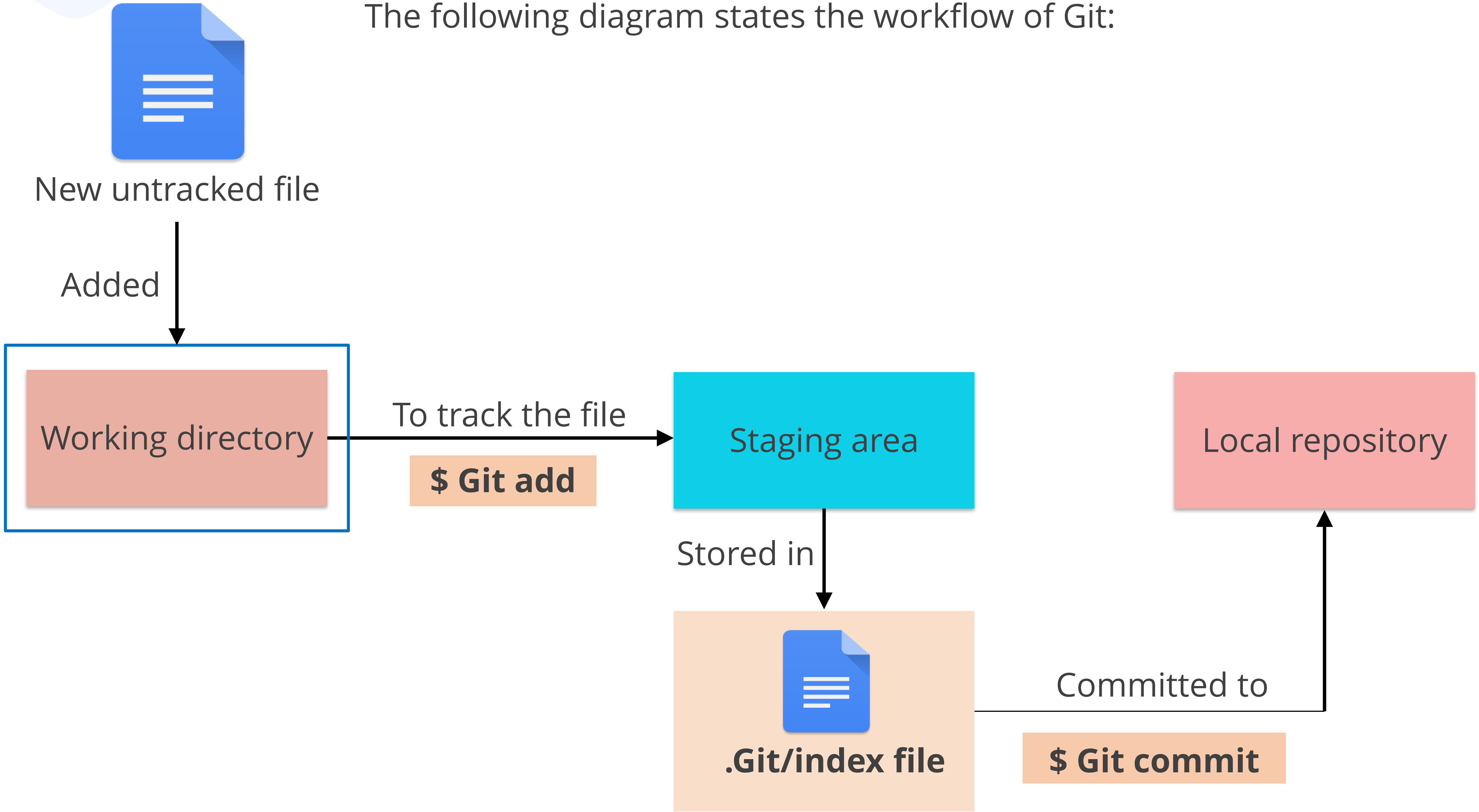
Lifecycle of Git

The following diagram shows the lifecycle of Git:



Git Workflow

The following diagram states the workflow of Git:



New untracked file

Added

Working directory

To track the file

$ Git add

Staging area

Stored in

.Git/index file

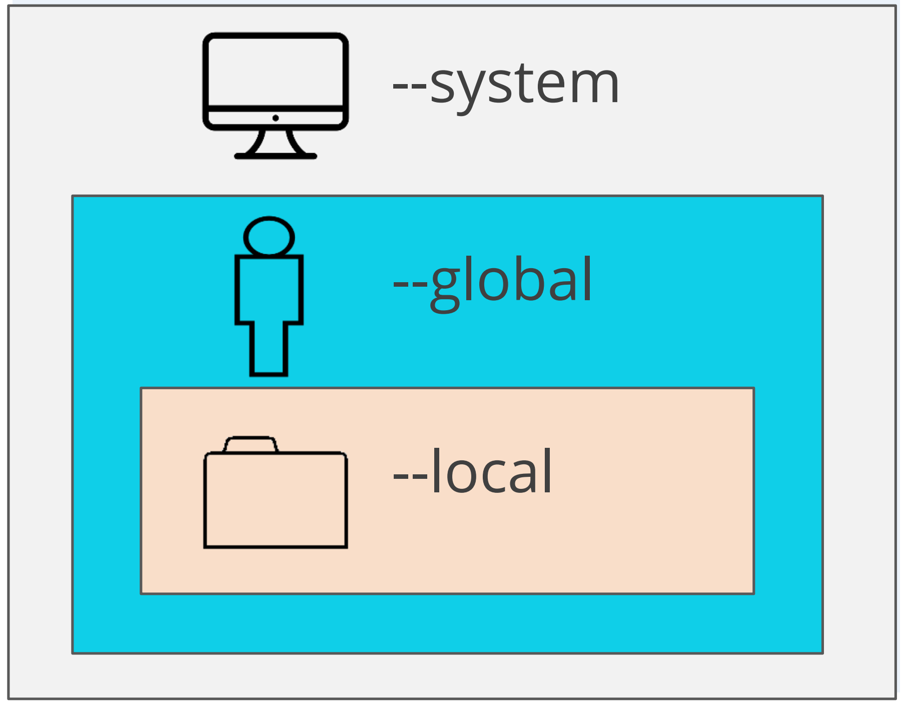
Committed to

$ Git commit

Local repository

Git Configuration Level

The Git config command allows the configuration of the Git settings.
Below is the hierarchy of Git configuration levels:



Command: ‘Git config --system’

Command: ‘Git config --global’

Command: ‘Git config --local’

Basic Git Commands

Some of the basic Git commands are listed below:

Task	Explanation	Commands
Tell Git who the user is	Configure the author’s name and email address	Git config --global user.name “Simplilearn" Git config --global user.emailsimplilearn@example.com
Create a new local repository	Create a repository	Git init
Check the repository	Create a working copy of a local repository	Git clone /path/to/repository
Check the repository	Use a remote server	Git clone username@host:/path/to/repository

Basic Git Commands

Some of the basic Git commands are listed below:

Task	Explanation	Commands
Add files	Add one or more files to staging	Git add <filename> Git add *
Push	Send changes to the master branch	Git push origin master
Commit	Commit changes to the head	Git commit -m "Commit message"
Commit	Commit files added with **Git add** and the files changed	Git commit -a

Basic Git Commands

Some of the basic Git commands are listed below:

Task	Explanation	Commands
Status	List the files that need to be changed, added, or committed	Git status
Connect to a remote repository	Add the server to push for the connection	Git remote add origin <server>
Connect to a remote repository	List all currently configured remote repositories	Git remote -v
Search	Search the working directory for foo()	Git grep "foo()"

Basic Git Commands

Some of the basic Git commands are listed below:

Task	Explanation	Commands
Branches	Create a new branch and switch	Git checkout -b <branchname>
Branches	Switch from one branch to another	Git checkout <branchname>
Branches	List all the branches that tell you what branch you're currently in	Git branch
Branches	Delete the feature branch	Git branch -d <branchname>

Basic Git Commands

Some of the basic Git commands are listed below:

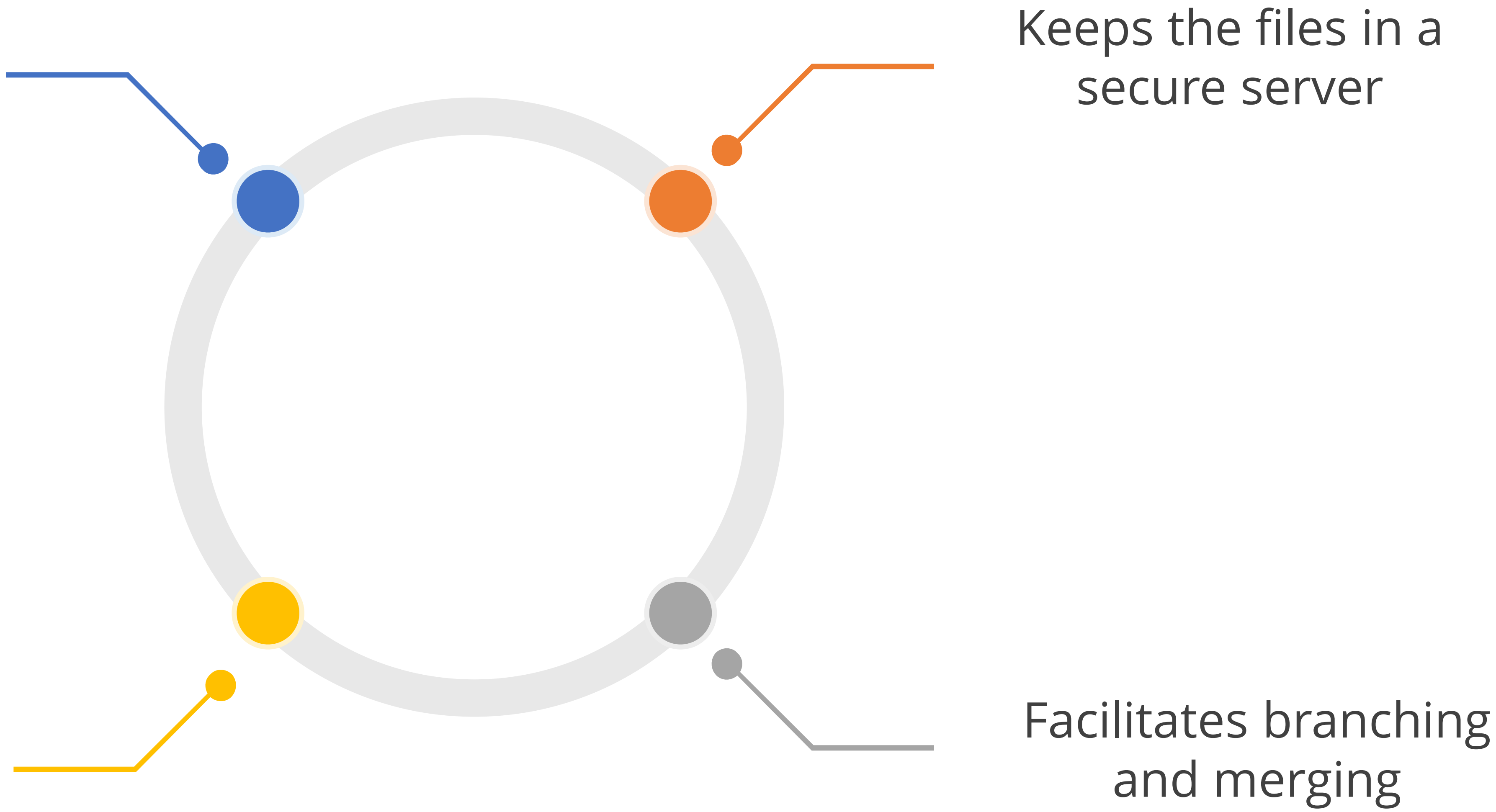
Task	Explanation	Commands
Branches	Push the branch to your remote repository	Git push origin <branchname>
Branches	Push all the branches to your remote repository	Git push --all
Branches	Delete a branch from your remote repository	Git push origin :<branchname>

Basic Git Commands

Some of the basic Git commands are listed below:

Task	Explanation	Commands
Update from the remote repository	Fetch and merge changes on the remote server	Git pull
Update from the remote repository	Merge a different branch in an active branch	Git merge <branchname>

Advantages of Git



Has flexible environment

Keeps the files in a secure server

Creates and manages remote repositories

Facilitates branching and merging

Platforms like GitLab extend Git’s functionality by offering integrated CI/CD pipelines, issue tracking, and collaboration tools for streamlined DevOps workflows.

What Is GitLab?

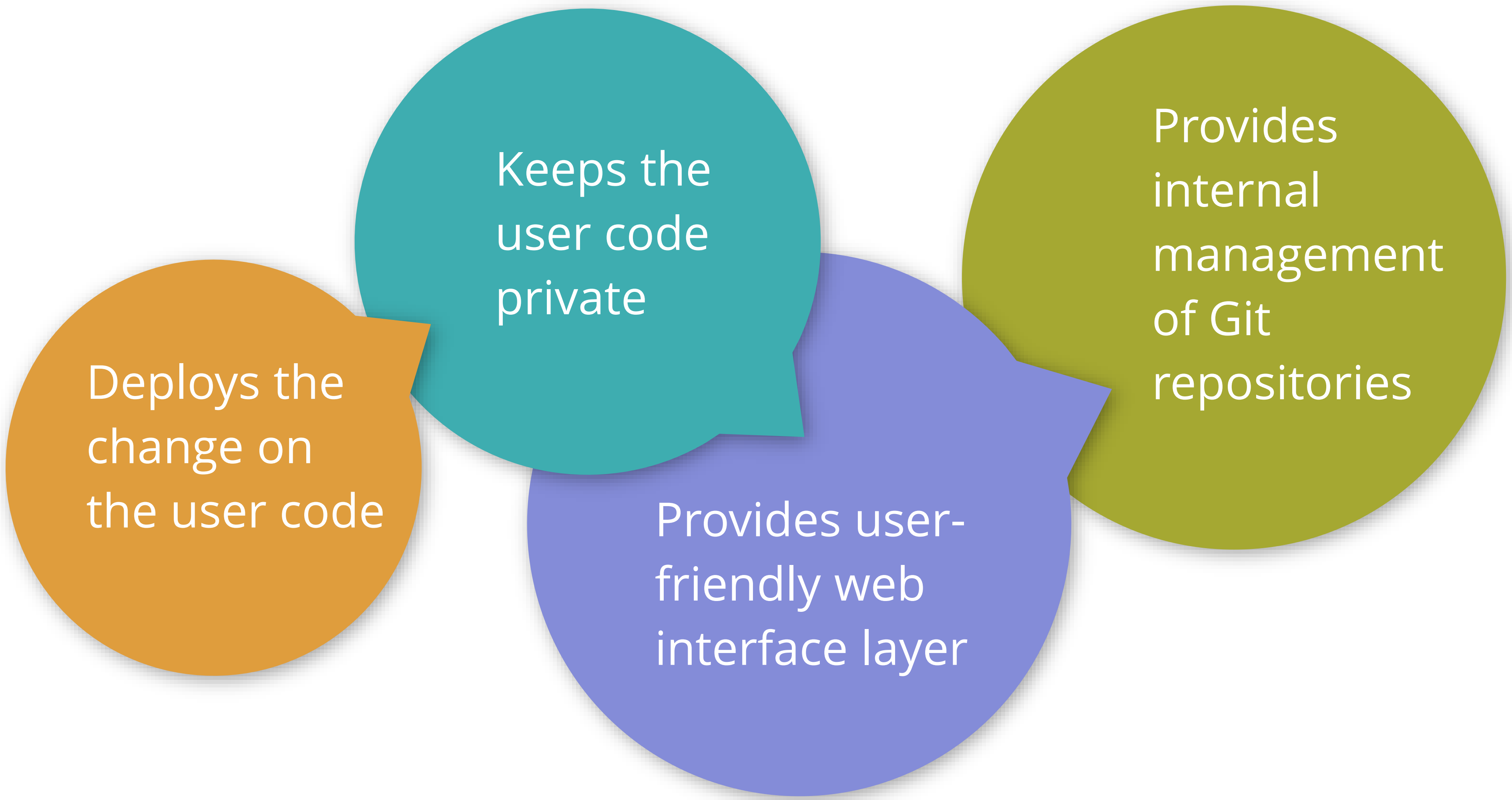
It is an open-source code repository and collaborative software development platform for large DevOps and DevSecOps projects. GitLab provides remote access to Git repositories.

It is built on Git and offers a complete DevOps lifecycle solution in a single application.

GitLab: Features

GitLab is widely used by development teams and enterprises to streamline their DevOps workflows and enhance collaboration, security, and automation in software projects.

The features of GitLab include:



Deploys the change on the user code

Keeps the user code private

Provides user-friendly web interface layer

Provides internal management of Git repositories

Quick Check

A development team uses Git for version control but faces challenges with tracking changes, merging conflicts, and collaboration. To streamline their workflow, they need a platform with built-in CI/CD, security, and issue tracking. How does Git improve version control, and how can GitLab enhance collaboration and automation?

A. Git enables version control, while GitLab offers CI/CD and collaboration tools.

B. Git executes code, and GitLab acts as a virtual machine.

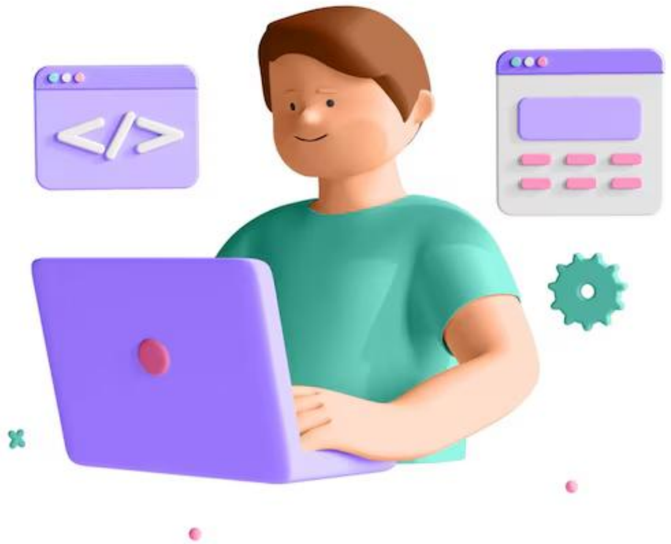
C. Git manages databases, and GitLab provides cloud storage.

D. Git replaces documentation, and GitLab writes code automatically.

GitHub as an SCM Tool

What Is GitHub?

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



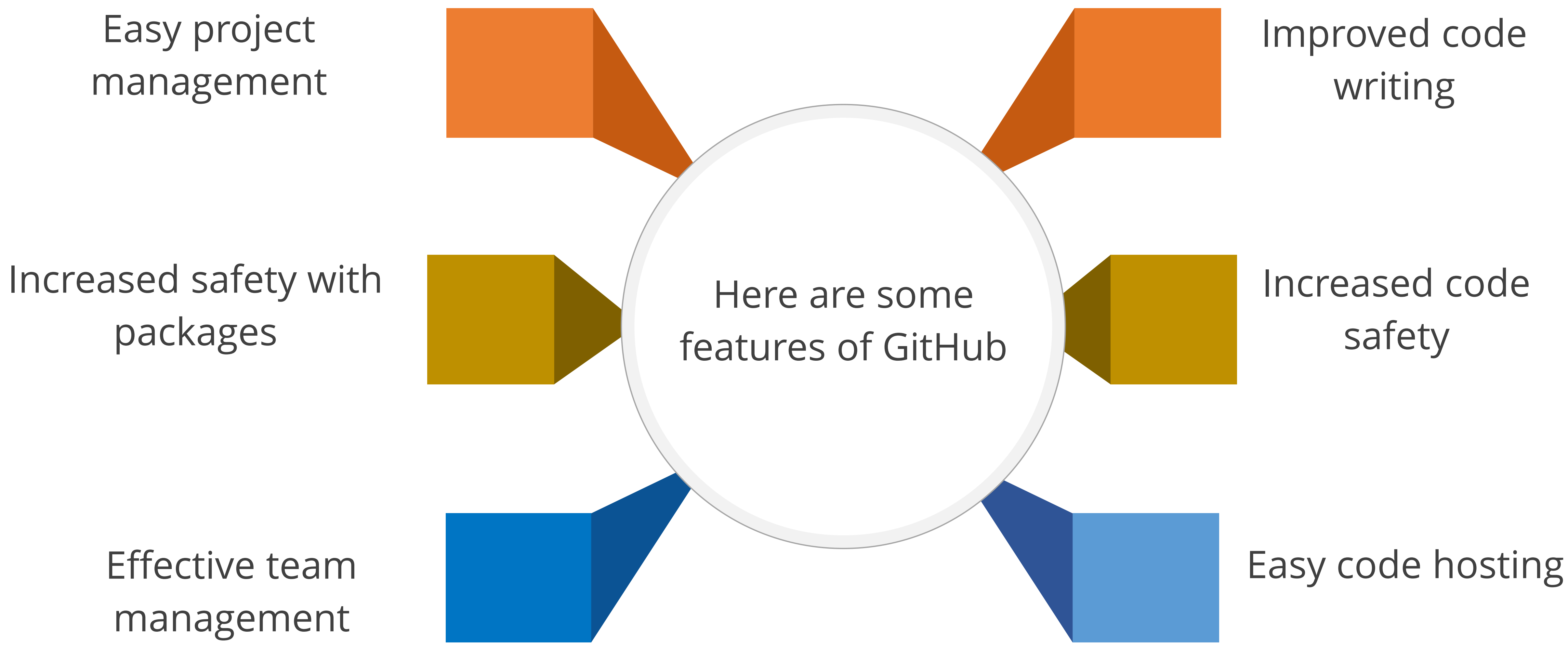
It involves collecting data from public sources, using tools, and analyzing the information to plan further actions during a cyberattack.

GitHub: Purpose

Below are the reasons to use GitHub:

- Enables teams to work on projects simultaneously without overwriting each other’s code
- Stores code in a centralized repository accessible from anywhere
- Allows developers to create branches for features, fix bugs, and merge updates seamlessly

Features of GitHub



Here are some features of GitHub

Easy project management

Increased safety with packages

Effective team management

Improved code writing

Increased code safety

Easy code hosting

Who Uses GitHub?

GitHub can be used by:

Individual
Developers use GitHub to manage personal projects and showcase their work.

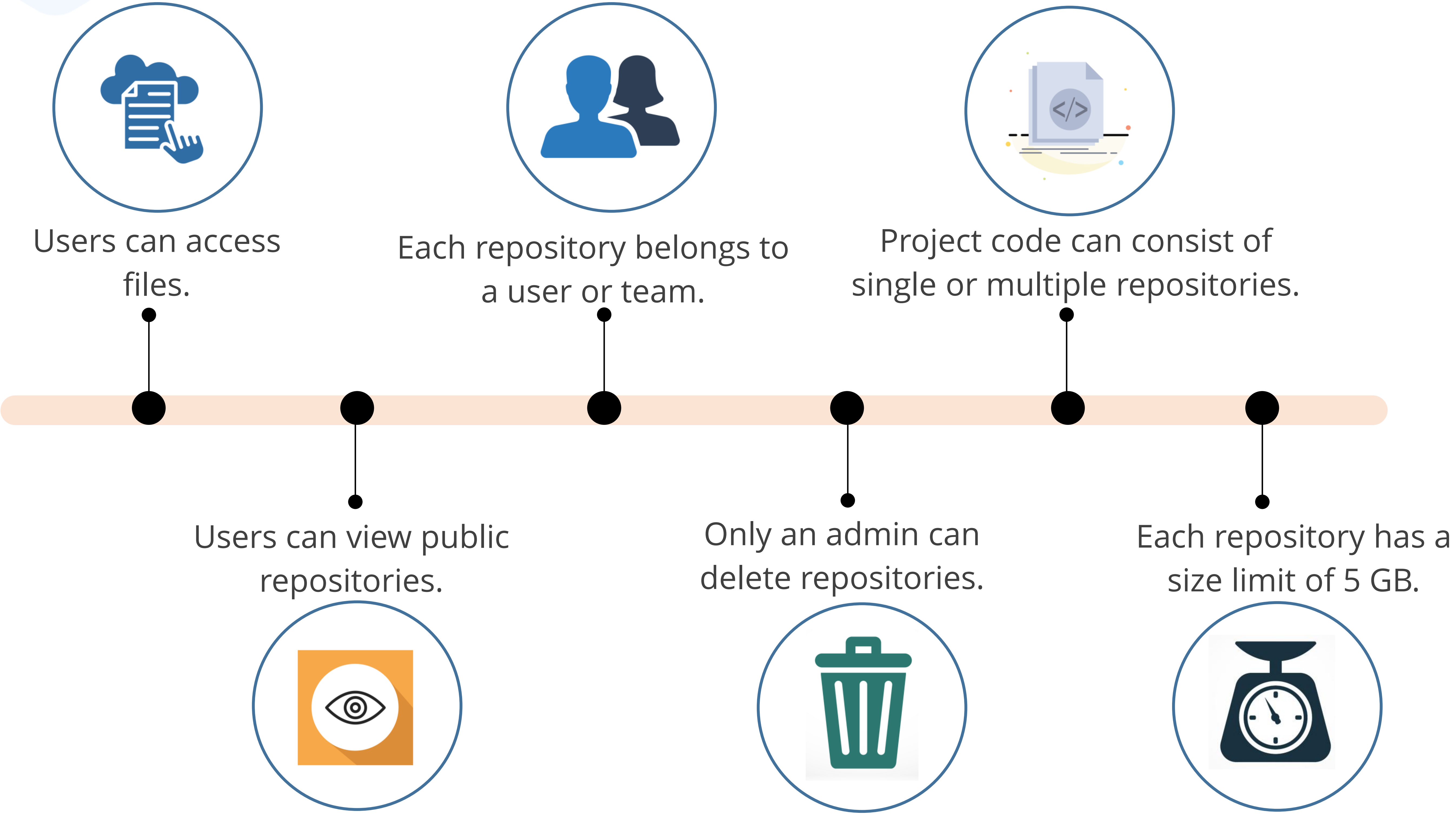
Community
Open-source contributors collaborate on projects and improve shared codebases.

Business
Companies use GitHub for team collaboration, version control, and software development.

Git vs. GitHub

Git	GitHub
Git is a Version Control System (VCS).	GitHub is a web-based version control and collaboration platform built on top of Git.
It is installed and maintained on the local system.	It is hosted on the web.
It is a command line tool.	It is a graphical interface.
It is a tool to manage different versions of the file in a Git repository.	It provides a web-based Git repository hosting service that provides a web interface to upload files.

Characteristics of GitHub Repository



Users can access files.

Users can view public repositories.

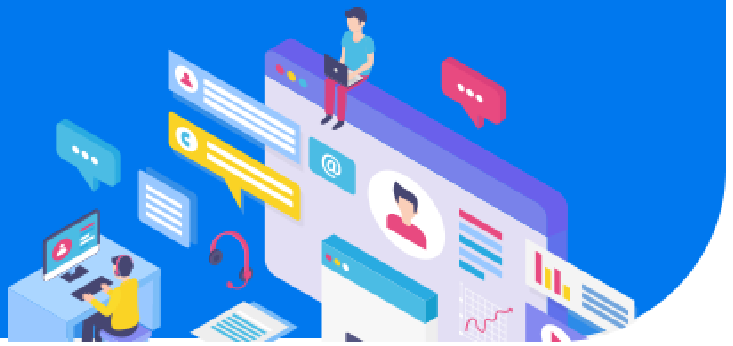
Each repository belongs to a user or team.

Only an admin can delete repositories.

Project code can consist of single or multiple repositories.

Each repository has a size limit of 5 GB.

Creating and Cloning a GitHub Repository



Duration: 20 Min.

Problem statement:

You have been assigned a task to create a centralized GitHub repository for version control and team collaboration.

Outcome:

By the end of this demo, you will be able to create a GitHub repository, initialize it with essential files, and configure repository settings.

Note: Refer to the demo document for detailed steps:
01_Creating_and_Cloning_a_GitHub_Repository

Creating and Cloning a GitHub Repository

Duration: 20 Min.

Steps to be followed:

1. Create a new GitHub repository
2. Edit the README file
3. Create a file in the repository
4. Clone the GitHub repository

Note: Refer to the demo document for detailed steps:
01_Creating_and_Cloning_a_GitHub_Repository

Quick Check

As a DevOps engineer, you need to configure multiple developers' Git environments for consistency and set up a Git workflow with GitHub for collaboration. Which approach best ensures proper Git configuration and seamless teamwork?

A. Configure Git at local, global, and system levels, and use GitHub for collaboration

B. Use only local Git configuration and manually sync changes

C. Rely on GitHub’s settings without configuring local Git

D. Use Git for version control and a separate tool for collaboration

Introduction to Fork, Push, and Pull

What Is Fork, Push, and Pull?

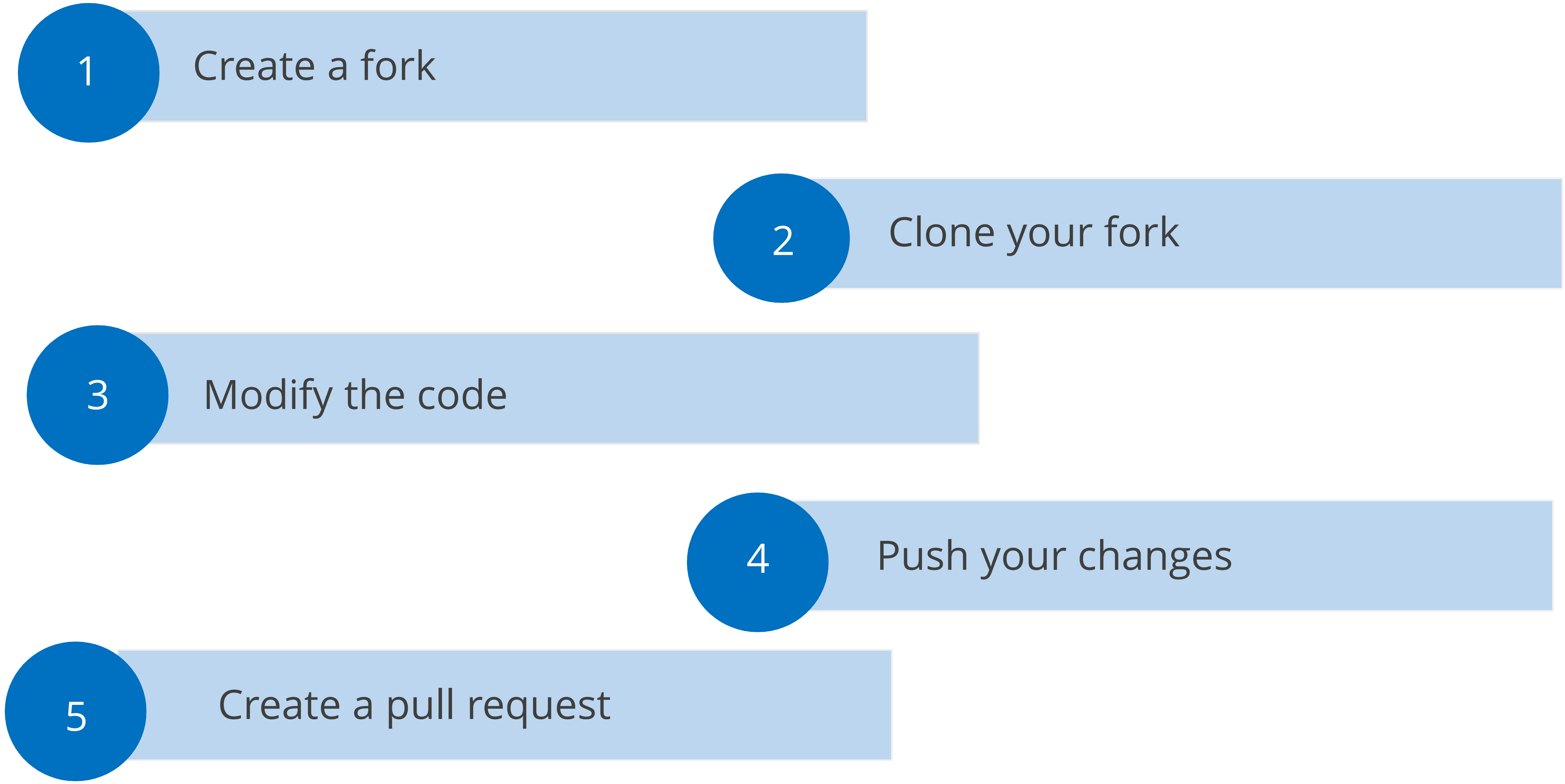
A fork is a new repository that shares code and visibility settings with the original upstream repository.

Push is the process of sending committed changes from a local repository to a remote repository, such as GitHub.

Pull involves fetching and merging changes from a remote repository into a local branch.

Creating a Repository: Steps

The steps to create a repository in GitHub using fork and pull requests are:



1 Create a fork

2 Clone your fork

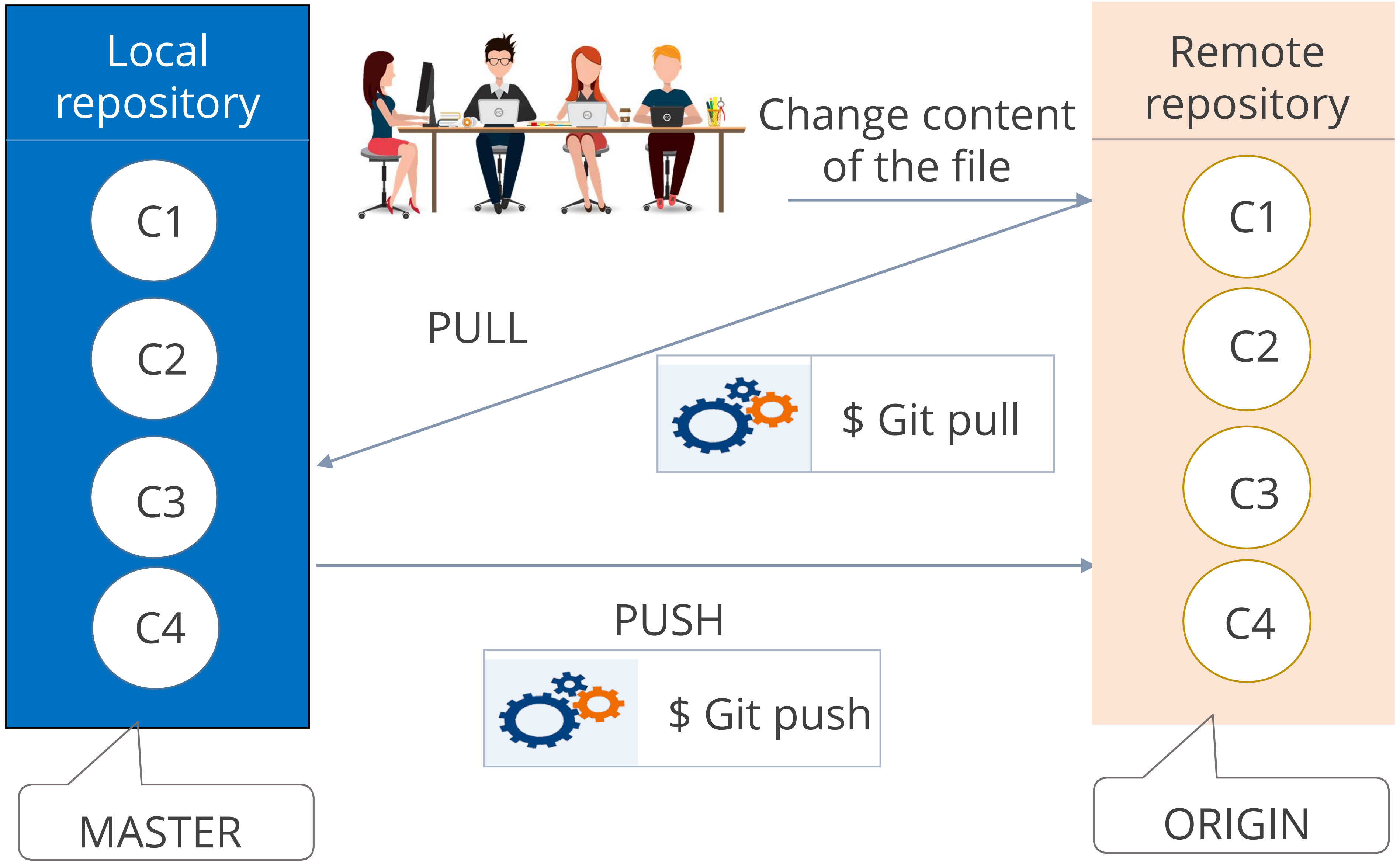
3 Modify the code

4 Push your changes

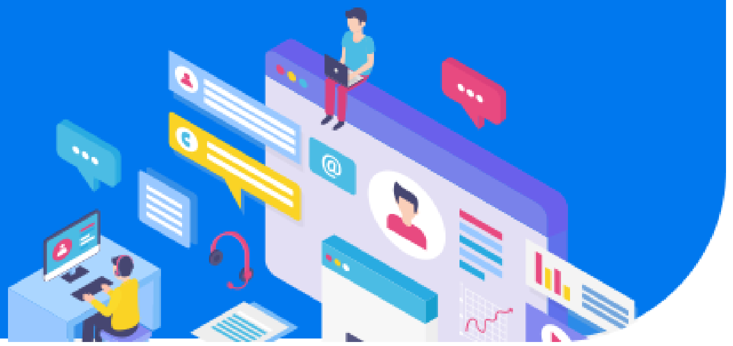
5 Create a pull request

Pushing and Pulling

The diagram shows the workflow of push and pull in Git:



Creating a Pull Request in Git



ASSISTED PRACTICE

Duration: 20 Min.

Problem statement:

You have been assigned a task to create a pull request in Git to propose changes, collaborate with reviewers, and merge updates into the main branch.

Outcome:

By the end of this demo, you will be able to create a pull request in Git, compare changes between branches, and submit it for review and approval.

Note: Refer to the demo document for detailed steps:
02_Creating_a_Pull_Request_in_Git

Creating a Pull Request in Git

Duration: 20 Min.

Steps to be followed:

1. Create a fork
2. Clone your fork
3. Sync fork with the original repository
4. Make changes in the file and create a commit message
5. Push your changes
6. Create a pull request

Note: Refer to the demo document for detailed steps:
02_Creating_a_Pull_Request_in_Git

Pushing Files to GitHub Repository

ASSISTED PRACTICE

Duration: 20 Min.

Problem statement:

You have been assigned a task to update a GitHub repository by adding new files or modifying existing ones. You need to ensure that your changes are properly tracked, committed, and pushed to the remote repository while maintaining version control and collaboration.

Outcome:

By the end of this demo, you will be able to add files to a GitHub repository, stage and commit changes using Git commands, and push the updates to the remote repository. You will also learn how to verify the changes on GitHub and manage version history for effective collaboration.

Note: Refer to the demo document for detailed steps:
03_Pushing_Files_to_GitHub_Repository

Pushing Files to GitHub Repository

ASSISTED PRACTICE

Duration: 10 Min.

Steps to be followed:

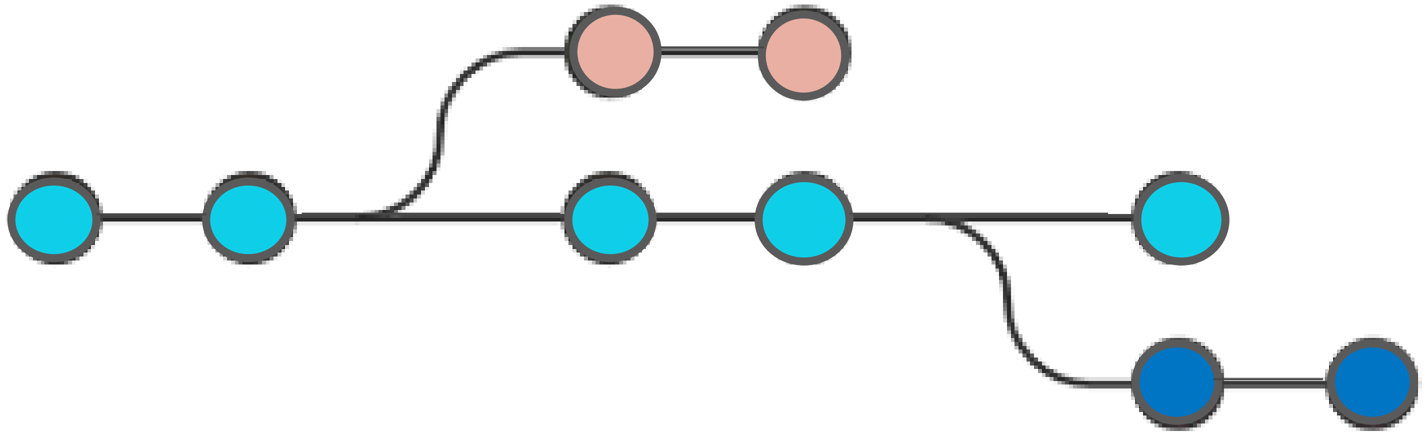
1. Create a GitHub repository
2. Create a repository on the local machine
3. Push the changes in the local repository to GitHub
4. Check the status of the local and remote repository

Note: Refer to the demo document for detailed steps:
03_Pushing_Files_to_GitHub_Repository

Introduction to Branching in Git

What Are Branches?

It is a separate workspace, usually created when you need to experiment or test something without impacting the main code base. A Git branch is a lightweight movable pointer to the commits.



Branches help in managing changes efficiently by enabling isolated development, testing, and merging of updates when they are ready.

Purpose of Branches

Purpose

- They allow users to work independently on different modules and merge them after development is complete.
- They can be created and deleted quickly.
- They require minimal storage as only the differences between branches are saved.
- They enable teams across different locations to work on independent features that combine into a cohesive project.
- They provide flexibility in development and collaboration.

Branch Operations

Operations	Description
Create a branch	The first step in the process is to create a branch. The user can start with the default branch or create a new branch.
Merge a branch	Every branch in the Git repository may be merged with an already running branch. Merging a branch can help when the user is finished with the branch and wants the code to integrate into another branch code.
Delete a branch	The user can remove an existing branch from the Git repository. When the branch has completed its task, that is, it has already been merged, or the user no longer needs it in the repository for some reason, you may delete it.
Checkout a branch	A user can pull or checkout a branch that is already running to create a clone of the branch so that they can work on it.

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 branches
$ Git merge <branch name>	Merges branches

NOTE

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

Creating a Branch in Git

ASSISTED PRACTICE

Duration: 20 Min.

Problem statement:

You have been assigned a task to create a branch in Git to work on new features or bug fixes without affecting the main codebase.

Outcome:

By the end of this demo, you will be able to create a new branch in Git, switch between branches, and work on isolated changes without impacting the main branch. You will also understand how branching improves collaboration and version control in a team environment.

Note: Refer to the demo document for detailed steps:
04_Creating_a_Branch_in_Git

Creating a Branch in Git

Duration: 20 Min.

Steps to be followed:

1. Create a new repository
2. Clone the GitHub repository
3. List all the branches in your repository
4. Create a new branch and verify it
5. Rename an existing branch
6. Delete the branch and verify it

Note: Refer to the demo document for detailed steps:
04_Creating_a_Branch_in_Git

Quick Check

A development team is working on a new feature while other teammates continue making updates to the main branch. They want to avoid conflicts, test the feature separately, and integrate it later.

What is the best approach to manage this scenario using Git branching?

A. Create a feature branch, make changes, and merge it

B. Modify code directly in the main branch

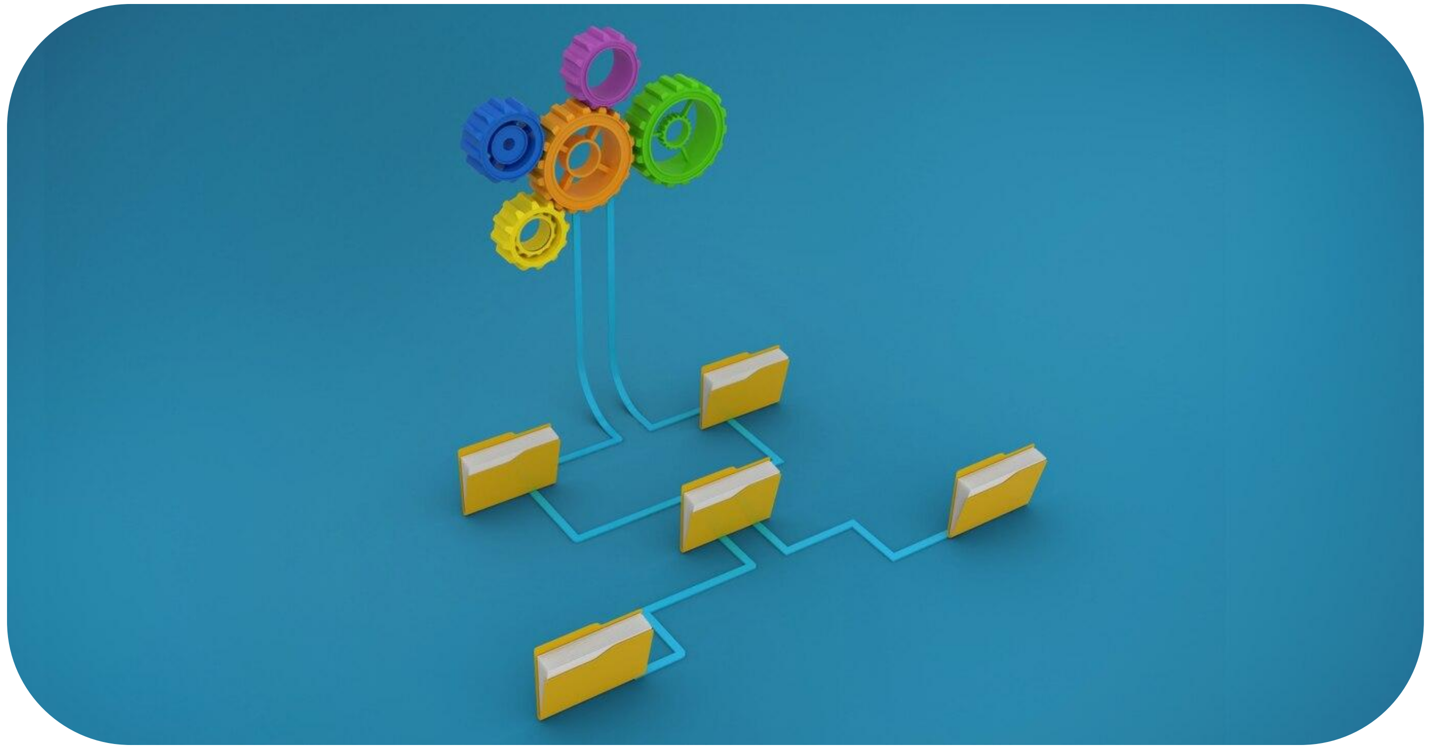
C. Clone the repository and merge changes manually

D. Use Git tags instead of branches

Switching Branches in Git

Switching Branches in Git Overview

It refers to moving from one branch to another within a repository. This allows developers to work on different features, bug fixes, or versions of a project without affecting the main codebase.



The switch command allows you to switch your current HEAD branch. It's relatively new and provides a simpler alternative to the classic checkout command.

Checkout in Git

The Git checkout command allows you to switch branches by updating the files in the working tree to match the version stored in the branch that the user wishes to switch to.

Checking out a branch updates the files in the working directory to match the version stored in that branch, and it tells Git to record all new commits on that branch.

Difference Between Git checkout and Git switch

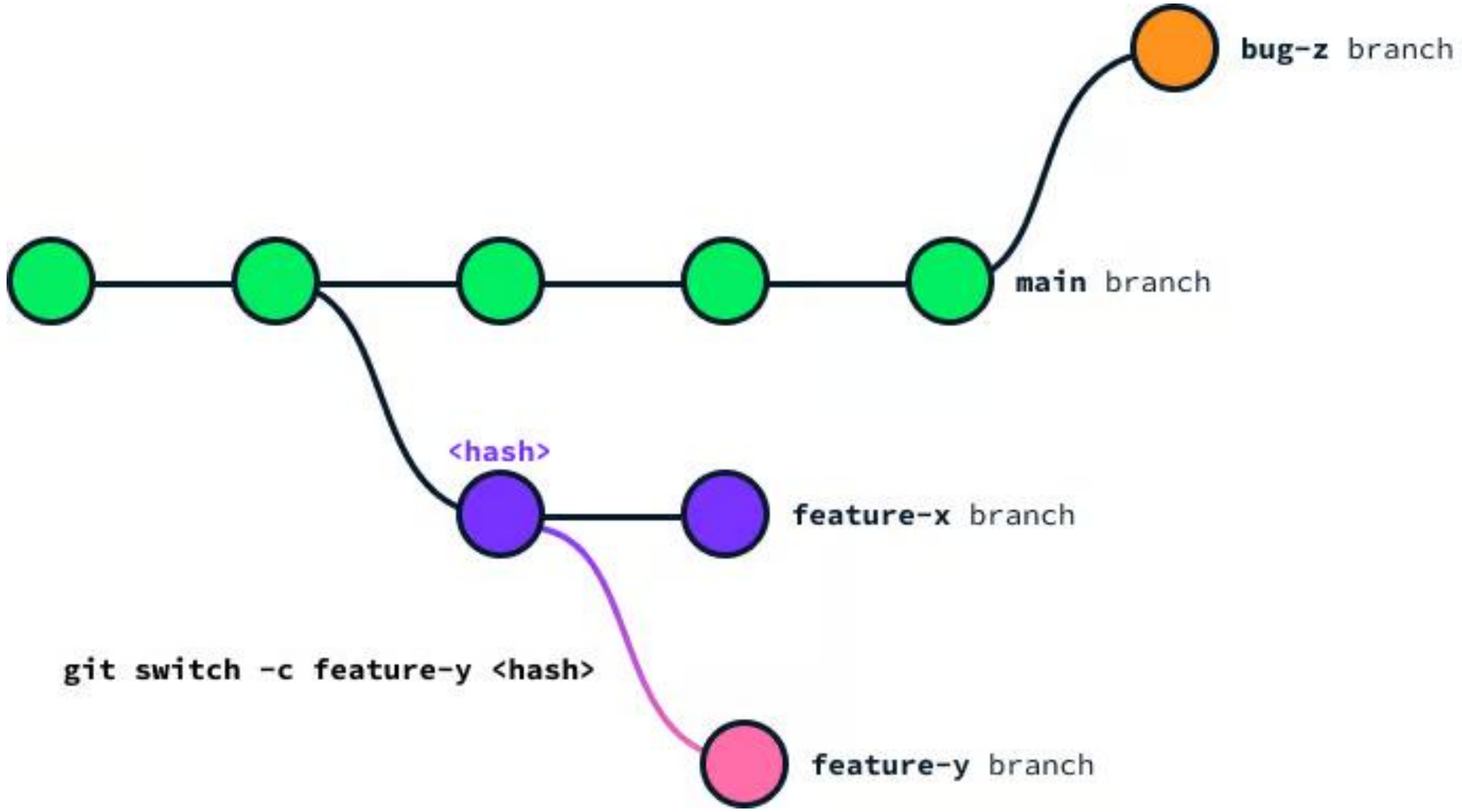
The table below compares the traditional Git checkout command with the newer Git switch command for branch switching and creation in Git:

Git operations	Git checkout	Git switch
To switch from one branch to another	Git checkout <branchname>	Git switch <branchname>
Creates a new branch and also switches to it	Git checkout -b <branchname>	Git switch -c <branchname>

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.



Switching Branches in Git

Duration: 20 Min.

Problem statement:

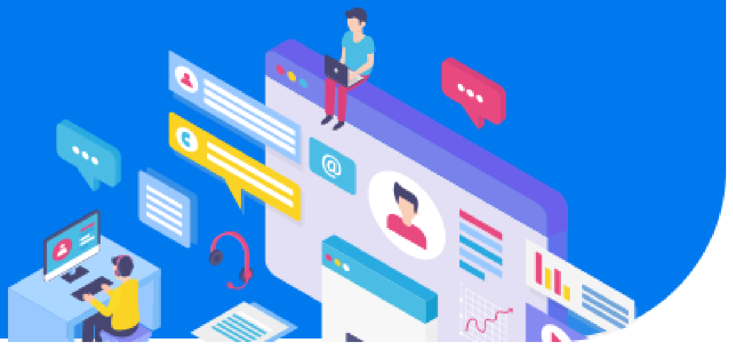
You have been assigned a task to switch between different branches in a Git repository to work on various features, bug fixes, or releases. Efficient branch switching is crucial for seamless collaboration and maintaining code stability.

Outcome:

By the end of this demo, you will be able to switch between branches using Git commands such as Git checkout or Git switch. You will also understand how to manage uncommitted changes while transitioning between branches, ensuring a smooth development workflow.

Note: Refer to the demo document for detailed steps:
05_Switching_Branches_in_Git

Switching Branches in Git



ASSISTED PRACTICE

Duration: 20 Min.

Steps to be followed:

1. Create a new branch
2. Switch to the new branch
3. Create a file and commit the changes
4. Check the status of the new branch
5. Switch back to the main branch

Note: Refer to the demo document for detailed steps:
05_Switching_Branches_in_Git

Merging Branches in Git

Introduction to Merging Branches in Git

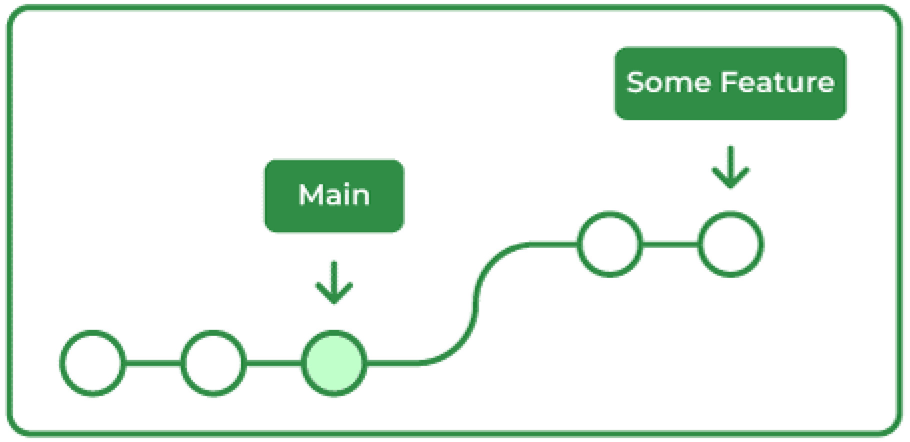
Merging is Git's way of putting a forked history back together again.
The **Git merge** command integrates the independent development lines created by the Git branch into a single branch.

Steps to merge branches:

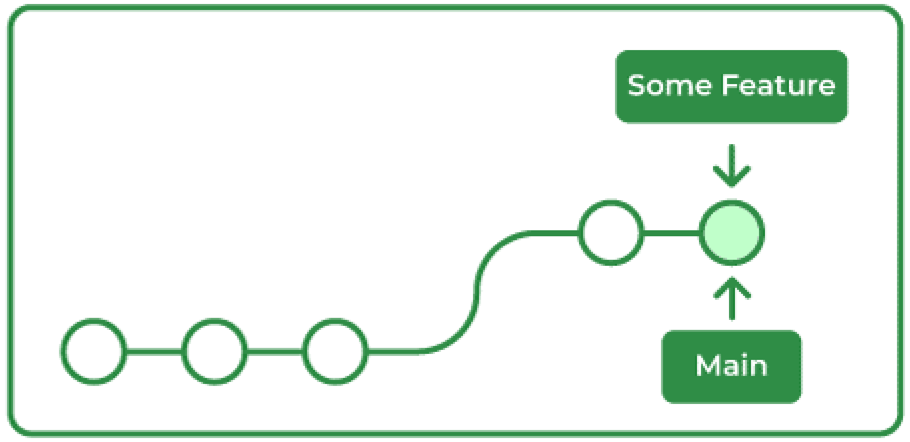
- Switch to the branch you want to merge
- Execute: **$ Git merge <branch name>**

Introduction to Merging Branches in Git

Git merge will combine multiple sequences of commits into one unified history. In most cases, Git merge is used to combine two branches. Below is the representation of before merging and after merging:



Before merging



After merging

After merging, it is essential to check the status of files to ensure all changes have been successfully integrated and no conflicts remain.

Status of a file in Git

The Git status command displays the current state of the repository. It enables the user to see the tracked, untracked, and changed files and changes. This command returns no commit records or information.

Syntax:

```
$ Git status
```

Merging Branches in Git

Duration: 20 Min.

Problem statement:

You have been assigned a task to merge branches in Git to integrate changes from a feature branch into the main branch while preserving version history.

Outcome:

By the end of this demo, you will be able to merge branches in Git using commands like Git merge, resolve merge conflicts if they arise, and verify that changes are successfully incorporated.

Note: Refer to the demo document for detailed steps:
06_Merging_Branches_in_Git

Merging Branches in Git

Duration: 20 Min.

Steps to be followed:

1. Create a new branch
2. Create a new file in the new branch
3. Switch to the main branch
4. Merge the branches
5. Push the changes to the remote repository

Note: Refer to the demo document for detailed steps:
06_Merging_Branches_in_Git

Checking the Status of a File

Duration: 20 Min.

Problem statement:

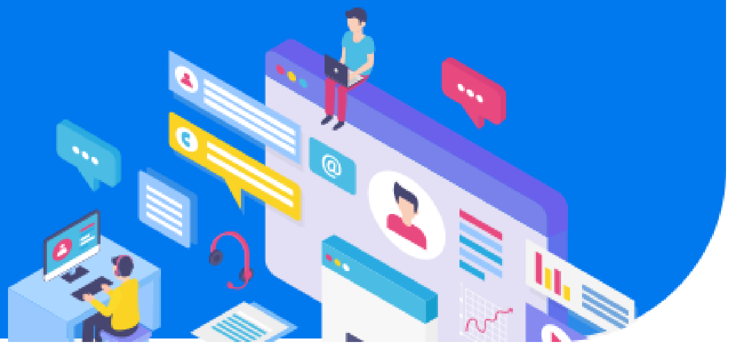
You have been assigned a task to check the status of a file in Git to track changes, identify uncommitted modifications, and ensure version control integrity.

Outcome:

By the end of this demo, you will be able to check the status of files in a Git repository using the Git status command.

Note: Refer to the demo document for detailed steps: 07_Checking_the_Status_of_a_File

Checking the Status of a File



ASSISTED PRACTICE

Duration: 20 Min.

Steps to be followed:

1. Create a GitHub repository
2. Create a directory to check the status of the file

Note: Refer to the demo document for detailed steps:
07_Checking_the_Status_of_a_File

Key Takeaways

- Software configuration management (SCM) is a set of processes, policies, and tools that organize the development process.
- Git is a version control system for tracking changes in computer files.
- GitHub provides a web-based Git repository hosting service that provides a web interface to upload files.
- The different operations of branching are to create a branch, merge a branch, delete a branch, and checkout a branch.

Creating and Managing a Git Repository for HTML Project Deployment

Project Agenda: To create and manage a Git repository for a basic HTML project, simulating a real-world version control workflow. The process includes initializing a local repository, staging and committing project files, linking the repository to a remote GitHub repository, and pushing the code to the remote server.

Description: This project includes setting up a Git-based version control workflow for a simple HTML website. You are tasked with creating a local project directory, adding HTML files (index.html and schedule_meeting.html), writing a ReadMe.md file for documentation, and initializing a Git repository. The repository should then be connected to a remote GitHub repository, where all project files are committed and pushed.

LESSON-END PROJECT

Creating and Managing a Git Repository for HTML Project Deployment

Steps to be performed:

1. Create the project directory
2. Add HTML files to the project directory
3. Create a ReadMe.md file
4. Initialize a local Git Repository
5. Create a remote Repository on GitHub
6. Add the remote origin
7. Stage files for commit
8. Push the code to GitHub

Expected deliverables: A structured project directory and a fully configured Git repository for a basic HTML website, including a ReadMe.md file with descriptive documentation. The deliverables will demonstrate a complete Git workflow, starting from local repository initialization, staging and committing files, to configuring a remote origin and pushing code to GitHub.

LESSON-END PROJECT

Thank You