Terminal, Git and Github Manual

www.anchorsoftacademy.com

PART 2: Git

Table of Contents

1. Introduction to Version Control

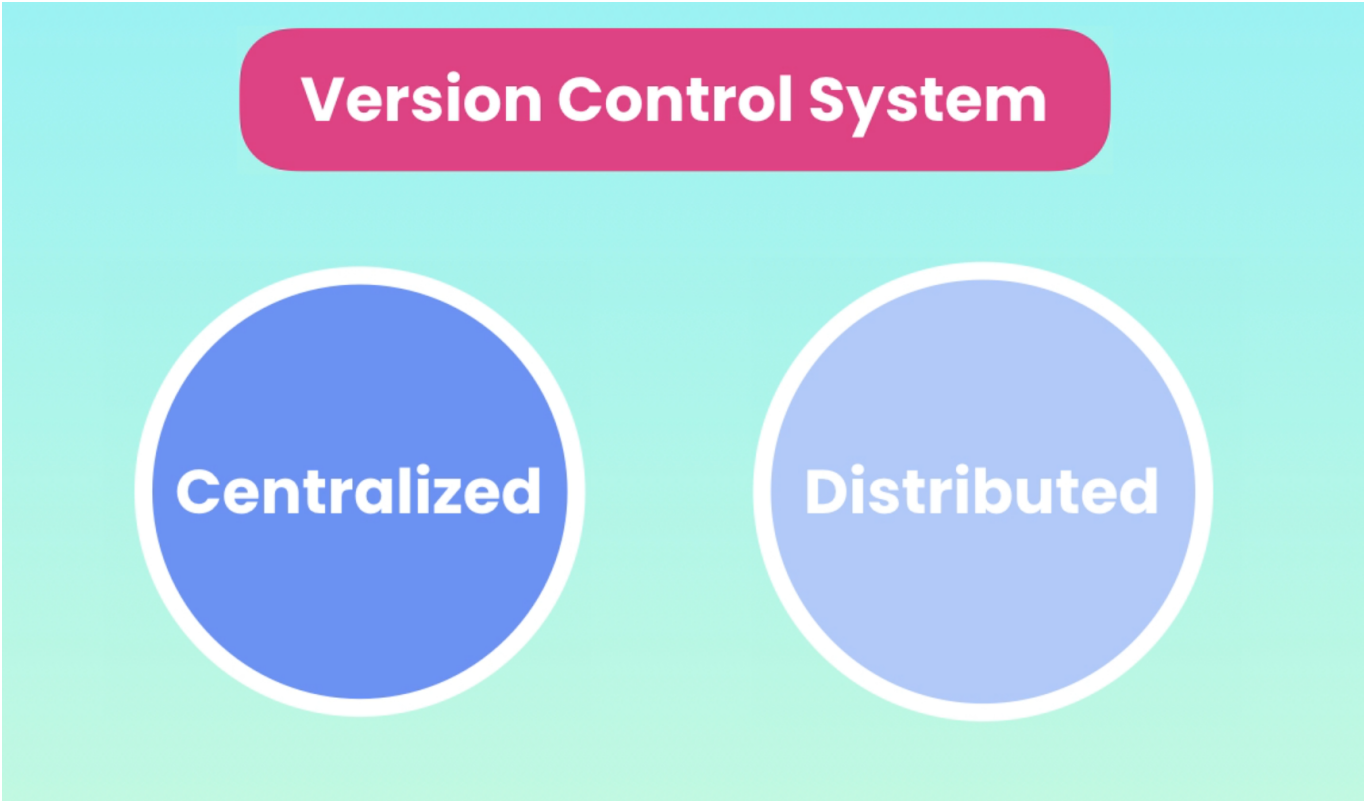
Version control is a system that manages changes to files and documents over time. It is used to track modifications, revisions, and updates made to a project's source code or any other set of files. With version control, you can keep a detailed history of changes, collaborate with others seamlessly, and easily revert back to previous versions if needed. Version control systems allow developers to work on different aspects of a project simultaneously without interfering with each other's work.

Why Use Version Control?

Version control offers several significant advantages for developers and teams working on software projects:

- **History and Auditability:** You can view the entire history of changes made to a file or project, including who made the changes and when. This helps in understanding why specific decisions were made and simplifies debugging.
- **Collaboration:** Multiple developers can work on the same project simultaneously, and the version control system manages the merging of their changes automatically. This reduces conflicts and improves teamwork.
- **Reproducibility:** You can recreate any previous state of the project, making it easier to reproduce bugs and issues reported by users.
- **Branching and Parallel Development:** Version control allows developers to create branches, enabling them to work on new features or bug fixes independently without impacting the main codebase. This promotes experimentation and parallel development.
- **Backup and Disaster Recovery:** Version control systems act as a backup, protecting your code from accidental data loss and providing an additional layer of security.
- **Code Reviews:** Version control facilitates code reviews, allowing peers to provide feedback on changes before they are integrated into the main codebase.

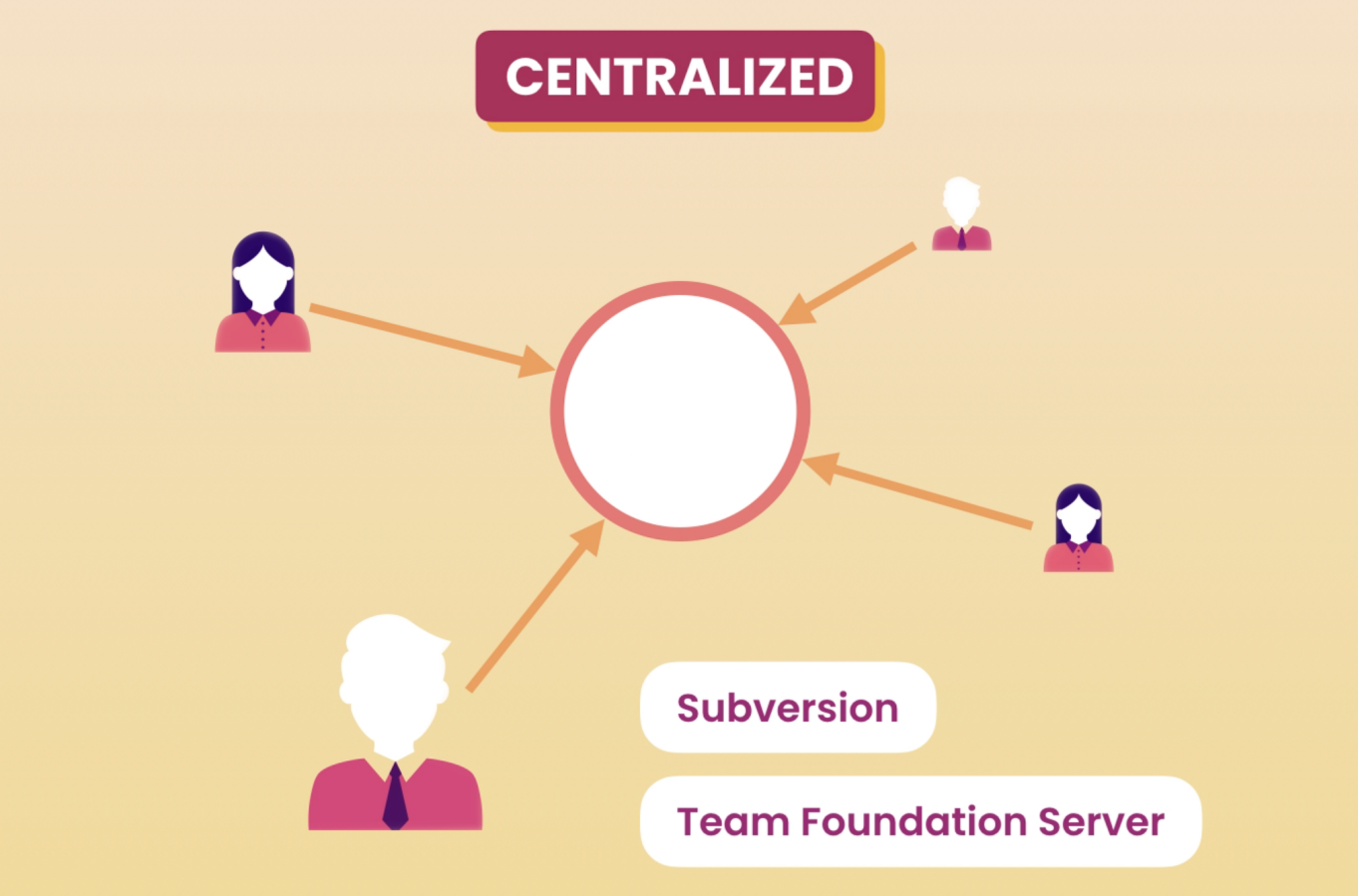
Types of Version Control Systems



There are mainly two types of version control systems:

Centralized Version Control System (CVCS)

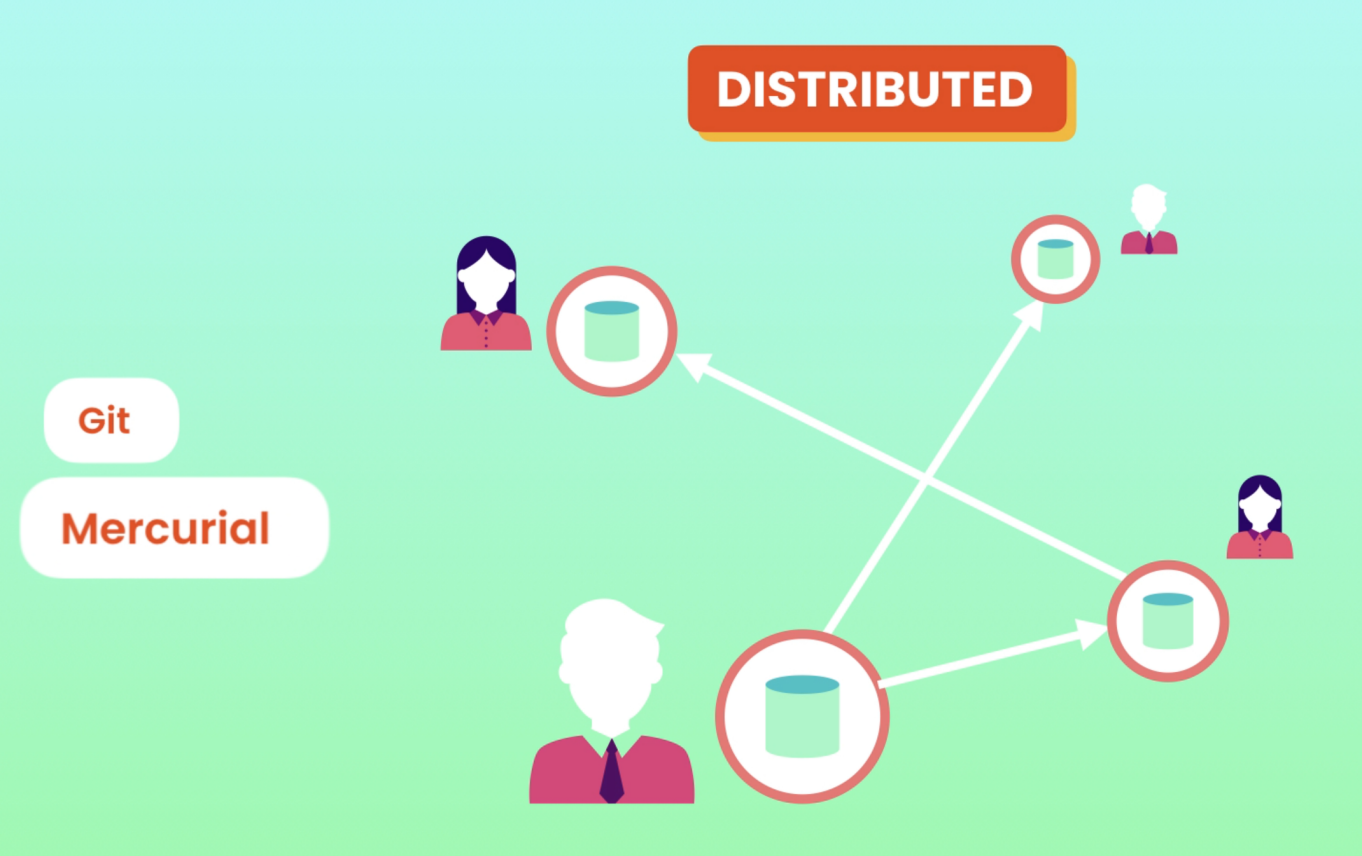
In a CVCS, there is a central server that stores the entire history of the project, and developers check out files from that central repository to work on them locally. Changes are made locally and then pushed back to the central server. Examples of CVCS include Subversion (SVN) and Perforce.



Distributed Version Control System (DVCS)

In a DVCS, each developer has a complete copy of the entire repository, including its history, on their local machine. This allows developers to work independently and commit changes locally without requiring a constant connection to a central server. Common DVCS examples include Git, Mercurial, and Bazaar.

While both types have their pros and cons, Distributed Version Control Systems like Git have become more popular due to their flexibility, robustness, and support for decentralized collaboration. Git, in particular, has become the de facto standard for version control in the software development industry.



2. Git Fundamentals

Installing Git

To use Git, you need to install it on your local machine. Here's how you can install Git:

- **Windows:** Download the Git installer from the official Git website (https://git-scm.com/) and run the executable file. Follow the installation wizard, and Git will be installed on your system.
- **macOS:** Git usually comes pre-installed on macOS. If it's not already installed or you want to get the latest version, you can use Homebrew (`brew install git`) or download the installer from the official Git website.
- **Linux:** On most Linux distributions, you can install Git using the package manager. For example, on Ubuntu, you can use `apt-get` with the command `sudo apt-get install git`.

Git Configuration

Configuring Git for the first time after installation involves setting up your identity and some basic settings. Here are the steps to configure Git:

1. Set Your Username and Email:

Open a terminal or command prompt, and run the following commands to set your name and email:

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

Replace "Your Name" with your actual name and "your.email@example.com" with your email address. The `--global` flag ensures that these settings apply globally to all repositories on your machine.

2. Set Default Text Editor (Optional):

By default, Git uses the system's default text editor for commit messages. If you want to use a specific text editor, you can set it using the following command (replace "editor_name" with your preferred editor's command):

```
git config --global core.editor "editor_name"
```

For example, to use Notepad as the editor on Windows:

```
git config --global core.editor "notepad"
```

3. Check Configuration:

You can check your Git configuration by running:

```
git config --list
```

This will display a list of all the settings that Git currently has, including your name, email, and other configurations.

4. Set Default Branch Name (Optional, Git 2.28+):

Starting from Git version 2.28, you can configure the default branch name. If you want to change the default branch name from "master" to something else (e.g., "main"), use the following command:

```
git config --global init.defaultBranch main
```

Replace "main" with your desired default branch name.

Basic Git Commands:**

Git commands are used to interact with the version control system. Here are some essential Git commands:

- `git init`: Initializes a new Git repository in the current directory.

- `git clone <repository_url>`: Copies a remote repository to your local machine.
- `git add <file>`: Stages changes for a specific file to be included in the next commit.
- `git commit -m "commit message"`: Commits the staged changes with a brief description.
- `git status`: Shows the current status of your working directory, including changes and staged files.
- `git log`: Displays the commit history, showing the author, date, and commit message for each commit.
- `git push`: Pushes your local commits to a remote repository.
- `git pull`: Pulls the latest changes from a remote repository and merges them into your local branch.
- `git branch`: Lists all branches in the repository.
- `git checkout <branch_name>`: Switches to the specified branch.
- `git merge <branch_name>`: Merges the changes from the specified branch into the current branch.

3. Git Workflow:

The typical Git workflow involves the following steps:

- **Initializing a Repository:** Create a new repository or clone an existing one to your local machine using `git init` or `git clone`.
- **Working on the Code:** Modify the files in your working directory according to the changes you want to make.
- **Staging Changes:** Use `git add` to stage the changes you want to include in the next commit.
- **Committing Changes:** Create a new commit using `git commit -m "commit message"` to save the staged changes with a descriptive message.
- **Pushing Changes:** If you're working with a remote repository, use `git push` to send your commits to the remote server.
- **Pulling Changes:** Before making new changes, use `git pull` to fetch and merge the latest changes from the remote repository into your local branch.
- **Branching and Merging:** Use branches (`git branch`) to work on new features or bug fixes without affecting the main codebase. Merge the changes into the main branch using `git merge`.
- **Review and Collaboration:** Git allows code review and collaboration by sharing repositories among team members. Changes can be reviewed using pull requests and discussions on the code.