# GitHub Training Agenda

(4-week program)

Author: Shaun Joel

* Week1:
  + Day1:
    - Welcome and Course Goals
    - Course Overview
    - About Instructor
    - About Audience
    - What is VCS?
  + Day2:
    - Git vs GitHub
    - Git Architecture
    - Create an account on GitHub
    - Key Git Terminology/GitHub Flow
    - Installation Overview
    - Installation on Windows/Mac
    - Hello World
  + Day3:
    - Setup Git
    - Authenticating with GitHub from Git
    - Set username
    - Caching credentials
    - Repeated credentials prompts
    - About remote repositories
    - Manage remote repositories
    - Associating text editors with Git
    - Handle line endings
    - Ignoring files
  + Day4:
    - Getting started with Git
    - Setting GitHub Desktop
    - Creating a local repository
    - Creating remote repository
    - Setting space to make changes (Branches)
    - Saving snapshot
    - Using Command line
  + Day5:
    - Working with repositories (Create, Clone & Fork)
    - Managing repository settings
    - Managing contributors
    - GitHub Issues & Projects
    - Wiki Pages
* Week2:
  + Day1:
    - Git
    - Basic Git Commands
      * Git Init
      * Git clone
      * Git add
      * Git commit
      * Git status
      * Git branch
      * Git merge
      * Git push
      * Git pull
    - Push commits to remote
      * Git push
      * Renaming branches
      * Dealing with non-fastforward errors
      * Resolving blocked commits
      * Pushing tags
      * Deleting a remote branch/tag
      * Remotes/forks
  + Day2:
    - Getting changes from remote repository
      * Options for getting changes
      * Cloning a repository
      * Fetching changes from a remote repository
      * Merging changes into your local branch
      * Pulling changes from a remote repository
      * Dealing with non-fast-forward errors
    - Splitting subfolder into a new repository
  + Day3:
    - About git subtree merges
    - Setting up the empty repository for subtree merge
    - Adding a new repository as a subtree
    - Synchronizing with updates and changes
  + Day4:
    - Git rebase
      * Rebasing commits against a branch
      * Rebasing commits against a point in time
      * Commands available with rebasing
      * Resolve conflicts after rebase
    - Dealing with special characters in branch and tag names
      * Why you need to escape special characters
      * How to escape special characters
      * Naming branches and tags
      * Restrictions on names in GitHub
  + Day5:
    - Maximum push limit
      * Troubleshooting the 2GB push limit
      * Splitting up a large push
      * Starting from scratch
* Week3:
  + Day1:
    - Integrations
      * Building integrations
      * Featured GitHub integrations
        + Editor Tools (VS, VS Code)
      * Project Management Tools
        + Jira
        + Slack
        + MS Teams
  + Day2:
    - Stash & Tags
    - Stashing untracked files & pops
    - Managing multiple stashes
    - Stashing into a branch
  + Day3:
    - GitOps
    - Example
  + Day4:
  + Day5:
* Week4:
  + Day1:
    - Coding best practices
    - Code reviews
    - Secret Management
    - Securing dependencies
  + Day2:
    - GitHub Actions
  + Day3:
    - Migrations from TFS to GitHub
  + Day4:
    - Live Project
  + Day5:
    - Open Discussion

👋 Welcome Note

**Welcome to GitHub Training for Corporate Teams!**

We're thrilled to have you join us for this hands-on session designed to empower your development workflows and collaboration practices u

🎯 Course Objectives

By the end of this training, participants will be able to:

* ✅ Understand the fundamentals of Git and GitHub, including repositories, commits, branches, and pull requests
* 🔐 Apply best practices for secure and efficient collaboration in corporate environments
* 🛠️ Use GitHub features like Issues, Projects, Actions, and Wikis to manage workflows and documentation
* 📦 Integrate GitHub with other tools and platforms commonly used in enterprise settings
* 🚀 Contribute confidently to shared codebases and participate in code reviews

Would you like help customizing this for a specific audience—like developers, project managers, or QA teams? I can also help you build a full training agenda or slide deck.

**About Audience:**

**🗣️ Friendly & Inclusive Approach**

**"Now that you know a bit about me, I’d love to hear from you! Could you briefly share your name, your role, and what you're hoping to get out of this GitHub training?"**

**🎯 Goal-Oriented Approach**

**"Before we dive in, let’s do a quick round of introductions. Please share your name, your department or team, and one thing you’d like to learn or improve using GitHub."**

**💬 Interactive Icebreaker Style**

**"Let’s break the ice! Tell us your name, your current role, and—just for fun—your favorite tech tool or coding snack. GitHub counts too!"**

**🧭 Structured Corporate Style**

**"To help tailor the session to your needs, I’d appreciate a quick intro from each of you. Please mention:**

* **Your name**
* **Your role or team**
* **Your experience level with Git/GitHub (Beginner / Intermediate / Advanced)"**

What is VCS?( https://www.geeksforgeeks.org/git/version-control-systems/)

A **Version Control System (VCS)** is a tool that helps track and manage changes to files—especially source code—over time. It's essential for collaborative software development and maintaining a reliable history of a project’s evolution.

🧠 Key Concepts of VCS:

* **Track Changes:** Every modification to a file is recorded, allowing you to see what changed, who changed it, and when.
* **Collaboration:** Multiple developers can work on the same project without overwriting each other’s work.
* **Rollback:** You can revert to previous versions of files if something breaks or needs to be undone.
* **Branching & Merging:** Developers can create separate branches to work on features or fixes independently, then merge them back into the main codebase.

🧰 Types of Version Control Systems:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

Local VCS - Stores version history on a single machine. Simple but limited to one user.

Centralized VCS (CVCS) - Uses a central server to store all versions. Developers pull and push changes to this server.

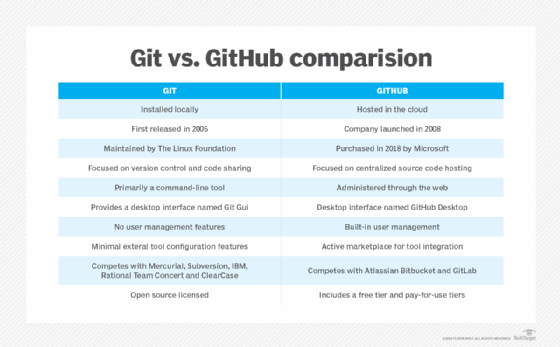
Distributed VCS (DVCS) - Each user has a full copy of the repository. Enables offline work and better redundancy. Git is a popular DVCS.

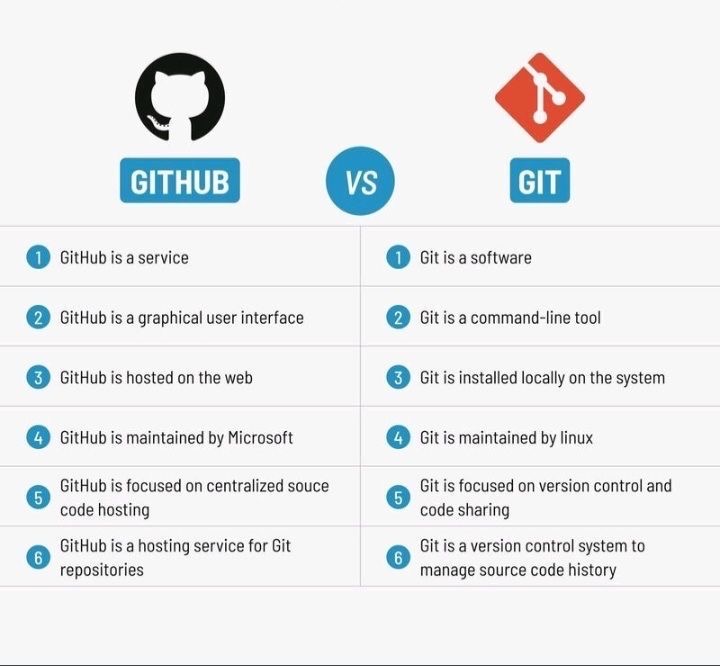
🚀 Why It Matters in Corporate Environments:

* Prevents accidental loss of code
* Enables code reviews and audits
* Supports agile and DevOps workflows
* Improves team productivity and accountability

GitHub, GitLab, and Bitbucket are popular platforms built on VCS principles—especially Git.

Git Vs GitHub



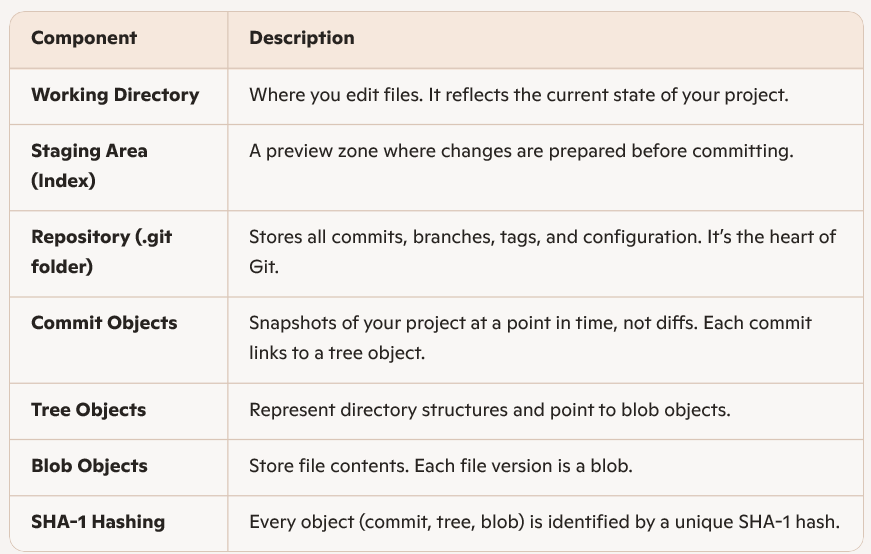


Git Architecture:

🏗️ Git Architecture Overview

Git is a **Distributed Version Control System (DVCS)**, which means every developer has a full copy of the project history. Its architecture is built around speed, data integrity, and support for distributed workflows.

🔧 Core Components of Git Architecture



|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

🔁 Git Workflow States

Git operates in **three main states**:

1. **Modified** – You’ve changed a file but haven’t staged it.
2. **Staged** – You’ve marked a file for inclusion in the next commit.
3. **Committed** – The file is safely stored in your local repository.

🧠 How Git Differs from Other VCS

* Git stores **snapshots**, not diffs.
* It’s **distributed**, so every user has the full history.
* It uses **content-addressable storage** via SHA-1 hashes for integrity.

A diagram of a remote work flow

AI-generated content may be incorrect.

Here are some helpful diagrams that illustrate Git architecture, including its three-stage workflow and internal data structures. These visuals show how Git manages your project through the **Working Directory**, **Staging Area**, and **Repository**, and how it uses objects like commits, trees, and blobs to track changes.

No, the physical location on your computer is not the same for all three stages.

Here is where each stage physically resides:

* **Working Directory:** Your working directory is the main project folder you see and interact with directly. This is where you create, edit, and delete files.
* **Staging Area:** The staging area is a conceptual "index" that lives as a file within the hidden .git directory. It tracks information about what changes are to be included in the next commit, not copies of the files themselves.
* **Local Repository:** This is the hidden .git subdirectory located at the root of your project folder. It is a database that contains the entire history of your project, including all past commits, branches, and objects.

Create an account on GitHub

[**Signing up for a new personal account**](https://docs.github.com/en/get-started/start-your-journey/creating-an-account-on-github?versionId=free-pro-team%40latest&productId=get-started&restPage=archiving-your-github-personal-account-and-public-repositories%2Copting-into-or-out-of-the-github-archive-program-for-your-public-repository#signing-up-for-a-new-personal-account)

1. Navigate to <https://github.com/>.
2. Click **Sign up**.
3. Alternatively, click on **Continue with Google** to sign up using social login.
4. Follow the prompts to create your personal account.

During sign up, you'll be asked to verify your email address. Without a verified email address, you won't be able to complete some basic GitHub tasks, such as creating a repository.

Some enterprises create managed user accounts for their users. You can't sign up for a personal account with an email address that's already verified for a managed user account.

If you're having problems verifying your email address, there are some troubleshooting steps you can take. For more information, see [Verifying your email address](https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/verifying-your-email-address#troubleshooting-email-verification).

Creating a repository with existing code:

To convert your existing local code into a [GitHub](https://github.com/) repository, you will first initialize a local Git repository, commit your code, and then link and push it to a newly created repository on GitHub.

**Step 1: Create a GitHub repository**

1. Go to GitHub and log in to your account.
2. Click the **+** sign in the top-right corner and select **New repository**.
3. Enter a name for your repository and an optional description.
4. Choose the visibility (Public or Private).
5. **Do not** initialize the repository with a README.md or .gitignore file, as you will add your existing files later.
6. Click **Create repository**.
7. After the repository is created, copy the URL provided in the "Quick Setup" box. You will need this for a later step.

**Step 2: Initialize Git in your project folder**

1. Open your terminal or command prompt.
2. Navigate to the root directory of your existing project using the cd command.
3. Initialize a new local Git repository by running git init.

**Step 3: Add and commit your code**

1. Add all of your project's files to the staging area with the command git add ..
2. Commit the files to your local repository with a descriptive message using git commit -m "Initial commit".

**Step 4: Link and push to GitHub**

1. Connect your local repository to the remote GitHub repository by running git remote add origin <URL>, replacing <URL> with the repository URL you copied earlier.
2. Verify that the remote was added correctly with git remote -v.
3. Push your code to GitHub with the command git push -u origin main (or master if that is your default branch name).

Your code is now converted and pushed to your GitHub repository.

Steps to create a local repository and upload to GitHub

* Go to your source code folder(parent folder)
* Type “git init”. It creates “.git” folder in the parent directory. With this local repo is ready.
* history
* 1 git
* 2 pwd
* 3 mkdir github\_exercises
* 4 cd github\_exercises/
* 5 mkdir repo1
* 6 cd repo1
* 7 touch file1.txt
* 8 vi file1.txt
* 9 ls
* 10 git init
* 11 ls
* 12 ls -alf
* 13 cd .git
* 14 ls
* 15 cat HEAD
* 16 ls -alf
* 17 ls -ltr
* 18 cat description
* 19 git status
* 20 cd ..
* 21 git status
* 22 git add file1.txt
* 23 git status
* 24 git rm --cached file1.txt
* 25 git status
* 26 git add .
* 27 git status
* 28 git branch
* 29 git commit -m "first commit"
* 30 git config --global user.email "shaunjoel525@gmail.com"
* 31 git config --global user.name "shaun"
* 32 git commit -m "first commit"
* 33 git status
* 34 git remote -v
* 35 git remote add origin https://github.com/shaunjoel/firstrepo.git
* 36 git branch
* 37 git push origin master
* 38 git diff
* 39 git fetch
* 40 ls
* 41 git pull
* 42 git branch --set-upstream-to=origin/master master
* 43 ls
* 44 git pull
* 45 ls
* 46 history