An introduction to version control

In this train, we take a look at version control and how it allows us to track code changes, both locally and remotely, and effectively collaborate.

Learning objectives

By the end of this train, you should be able to:

- Discuss what version control is and why it is used.
- Discuss the difference between version history and version control.
- Describe what a Distributed Version Control System (DVCS) is and list popular examples thereof.
- Distinguish between Git and GitHub and discuss how they are used in conjunction.

Outline

What is version control and why is it used?

> Version control is a system that allows us to **track the state of files as they change over time**, enabling multiple people to work on the same project simultaneously, without conflicts.

We use version control for the following reasons:

1. **Traceability:** Enables us to **track code changes**, especially if there are multiple collaborators.
2. **Accountability:** **Links code** to decisions, contributions, contributors, and time.
3. **Clarity:** We can easily **distinguish the latest version of code**.
4. **Reduces errors:** We can easily **identify and fix errors** in code with **minimal disruption** to collaborators.
5. **High availability:** We can immediately **switch to an earlier version of code** without interrupting the work of others.

Version history != version control

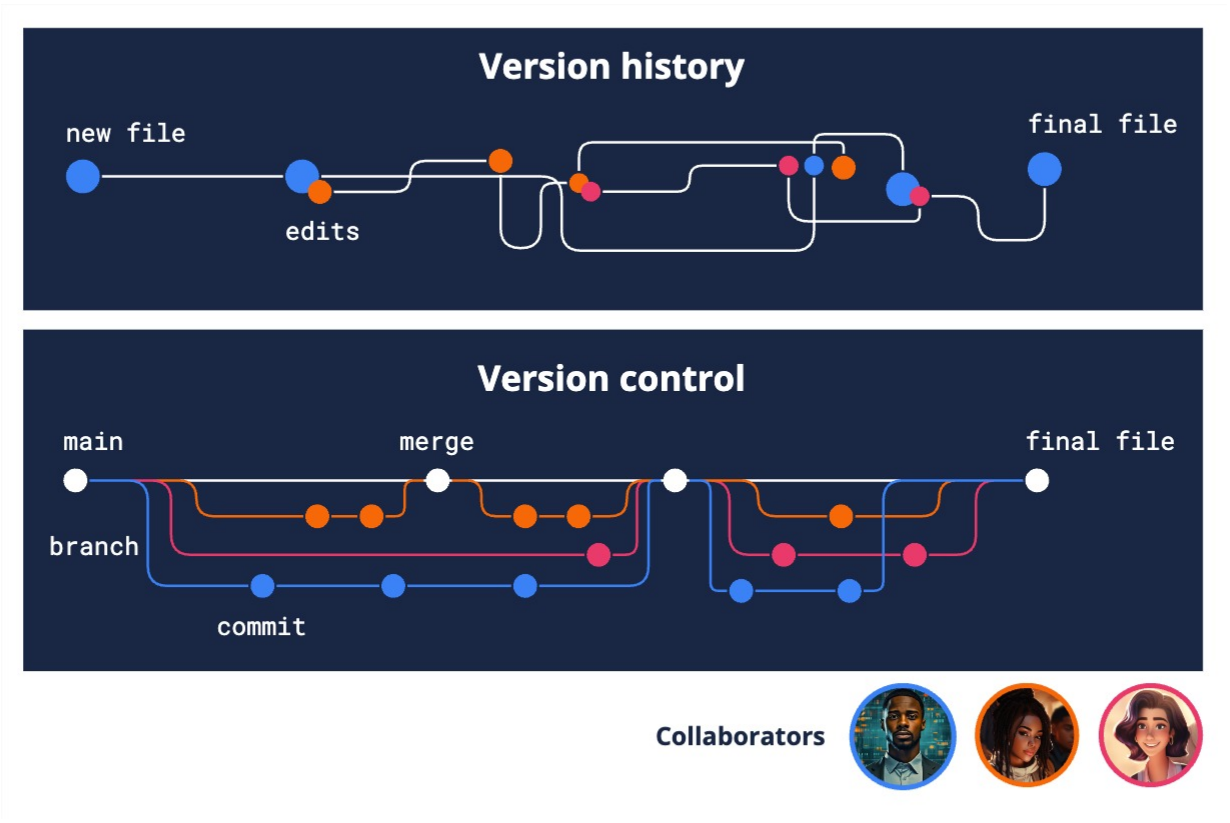

Figure 1: The difference between version history and version control.

Version history is simply the *chronological record of changes* made to a project or a file over time. **Version control**, on the other hand, is a system that records changes to a file or set of files over time with the primary purpose of facilitating collaboration among multiple people by **managing and tracking changes**.

What is Git?

> Git is a specific **Distributed Version Control System (DVCS).**

A **Distributed Version Control System (DVCS)** is a type of version control system that allows **multiple users to collaborate** on code. In a DVCS, the entire repository, including the complete version history of the project, is **mirrored on the local machines** of each contributor. This decentralisation enables us to work independently on our local copies of the code, making it possible to commit changes, create branches, and merge modifications without requiring constant access to a central server.

Git is a popular DVCS due to its flexibility, speed, and powerful branching and merging capabilities. It provides us with a rich set of features such as allowing many individuals to collaborate *remotely* on a software-based project, with the full power of a version control system.

Alternatives to Git include Mercurial, Bazaar, and Darcs. While Git is widely adopted, the choice between version control systems often depends on factors such as personal preference, project requirements, and the specific needs of the development team.

Basic Git terminology

- **Repository:** A container **where all the files of a project are stored** along with their history. Also commonly called a "repo".
- **Commit:** A **save point** of the project that records the changes made at a specific time.
- **Branch:** A **separate copy of the project** that allows us to work on new features without affecting the main version of the project.
- **Merge:** The process of integrating changes from one branch into another.

How does Git work?

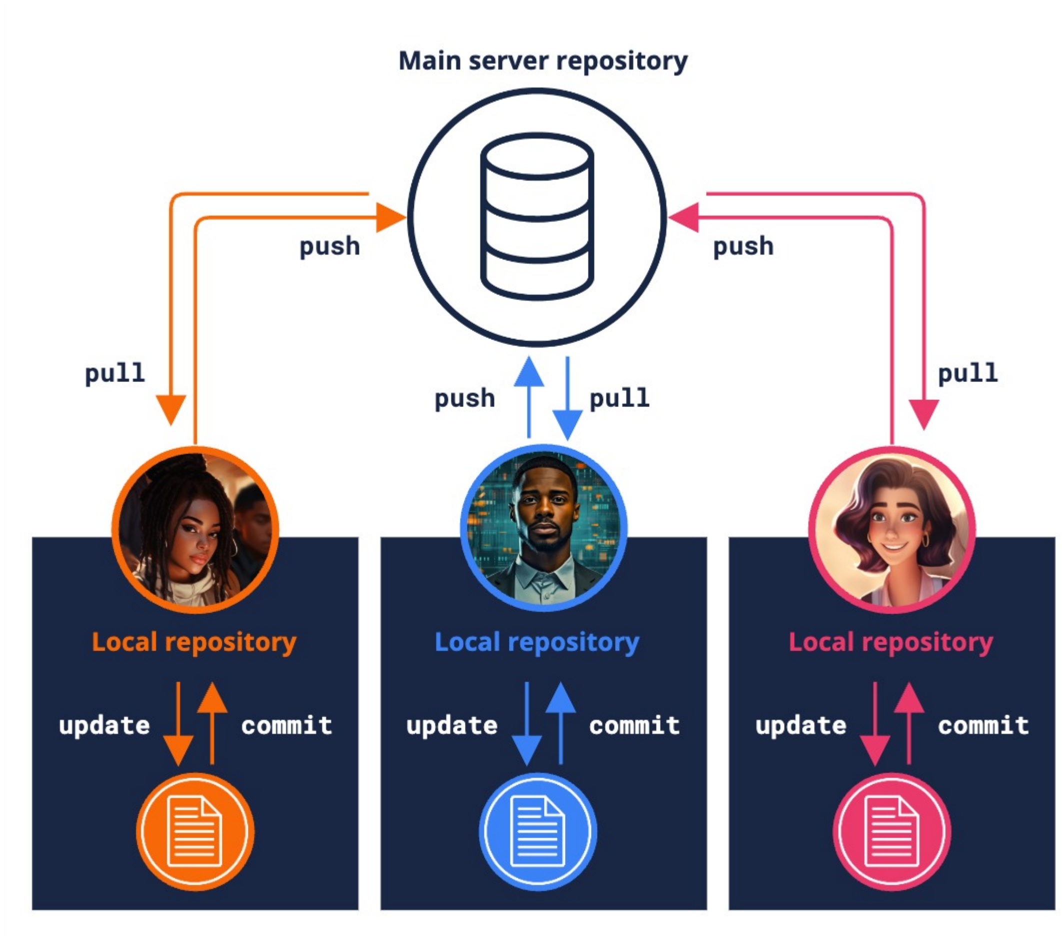

Figure 2: An example of how three collaborators would contribute to a single main repository.

In a Git-based project, a collaborator will typically work on their own local machine using a full copy of the project known as a **local repository**. As changes are made to the collaboratorâ€™s version of the project, Git commands are used to store and send these changes to a single **remote repository** to which every other collaborator in the project also has access.

Git uses a sophisticated process to **merge** these changes with those of other collaborators, enabling a single reputable and updated version of the project to exist.

What is GitHub?

GitHub is a web-based platform that **provides hosting for version control** using Git.

GitHub serves as a **collaborative platform** for software development, allowing multiple contributors to work on projects simultaneously. GitHub offers a **range of features** that facilitate collaboration, code management, and project organisation.

GitHub is built on Git, so we can **use Git without GitHub**, but we cannot use GitHub without Git.

Some key aspects of GitHub include:

- **Version control:** It is built on top of the Git version control system. It allows us to **track changes** in our code, **manage different versions** of our projects, and **collaborate effectively** with others.
- **Repository hosting:** It provides a **hosting service for Git repositories**. We can create repositories to store and organise our code, making it accessible to collaborators.
- **Collaboration features:** It facilitates **collaboration** through features such as **pull requests**, **issues**, and **code reviews**. Pull requests enable us to propose changes, and issues are used for tracking tasks, enhancements, and bugs.
- **Branching and merging:** It supports branching, enabling us to **work on different features or fixes simultaneously** without affecting the main codebase. Changes made in branches can be merged back into the main branch when ready.
- **Web-based interface:** It provides a **user-friendly web interface** that allows us to navigate repositories, view code, track issues, and manage project settings. This interface makes it accessible for both technical and non-technical users.
- **Social coding:** It has a social aspect, allowing us to **follow each other**, **star repositories**, and **contribute to open-source projects**. This social element fosters community engagement and collaboration.
- **Continuous integration:** It integrates with various continuous integration (CI) tools, allowing **automated testing and build processes** to be triggered whenever changes are pushed to a repository.
- **Hosting for documents:** Besides code, GitHub repositories can **host documentation for projects**. This documentation can include README files, wikis, and other resources to help users understand and contribute to a project.

- **Access control:** GitHub provides robust access control mechanisms, allowing repository owners to **manage permissions for collaborators**. This ensures that only authorised individuals can make changes to a project.

GitHub has become a central hub for many open-source and private software development projects, serving as a platform that brings together coders, facilitates collaboration, and provides tools for effective version control and project management.

Collaborative workflow with Git and GitHub

Here is a summary of how different users interact with GitHub through Git:

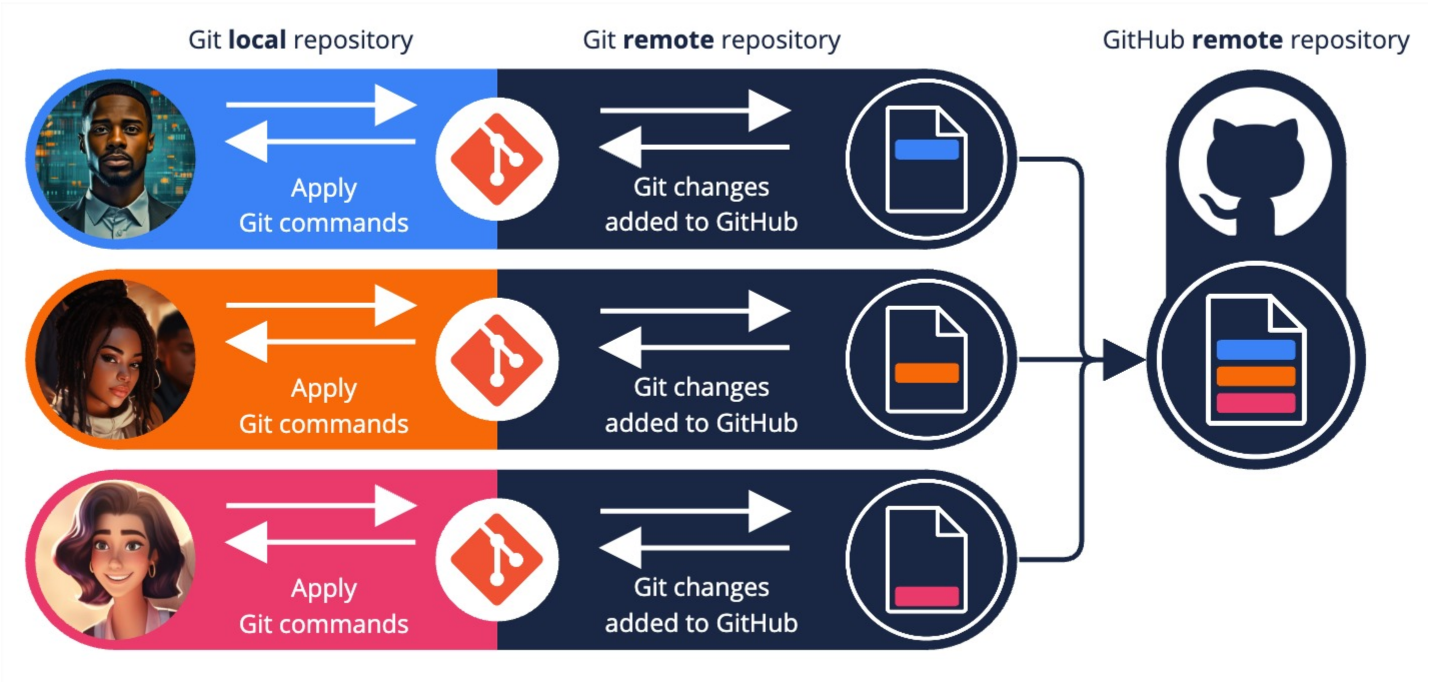

Figure 3: How we use Git and GitHub in a workflow.

Let's take a look at a high-level overview of how this would work:

1. **Start a project**
   If it doesnâ€™t exist yet, a **new remote GitHub repository** is created for the project. Each person then begins by initiating this remote repo on their local computer using Git, i.e. a **Git local repository**. This sets up a version control system that tracks local changes to the project made by that specific person.
2. **Collaborate with branches**
   When working on specific features or fixes, team members can **create branches in their Git local repository**. These branches allow for isolated development without affecting the main project.
3. **Make changes locally**
   Each person **works independently on their local copy** of the project. They can create, modify, or delete files as needed.
4. **Stage and save changes**
   After making changes, each individual **uses Git to stage and commit those changes to their Git local repository** ensuring a record of their work.
5. **Connect to GitHub**
   The team **creates a shared GitHub remote repository**, serving as a central hub for the project.
6. **Share changes on GitHub**
   Team members **push their committed changes from their Git local repository to the shared GitHub remote repository** This action makes their work accessible to others.
7. **Stay updated**
   Team members regularly **pull from the GitHub remote repository to stay updated** with each other's changes. This helps to avoid conflicts and keeps everyone on the same page.
8. **Submit changes for review**
   After completing a task, a team member can **submit their changes for review by creating a pull request on the GitHub remote repository** This proposes the changes to be merged into the main project.
9. **Review and merge**
   Other team members review the **proposed changes on the GitHub remote repository**. They can provide feedback, ask questions, and once satisfied, merge the changes into the main project.
10. **Repeat and iterate**
    The team continues this collaborative cycle. Everyone contributes, **shares their progress on GitHub**, reviews each other's work, and collaboratively improves the project.

alx