OOP Lab 01

Version Control with Git and GitHub

What is Version Control?

A version control system (VCS), allows you to track changes to your code over time. It's essential for developers who often need to work on the same codebase concurrently. It gives them a way to undo mistakes, track changes, manage conflicts, and collaborate with others on code. However, anyone who regularly works with files that change can benefit from using version control.

If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

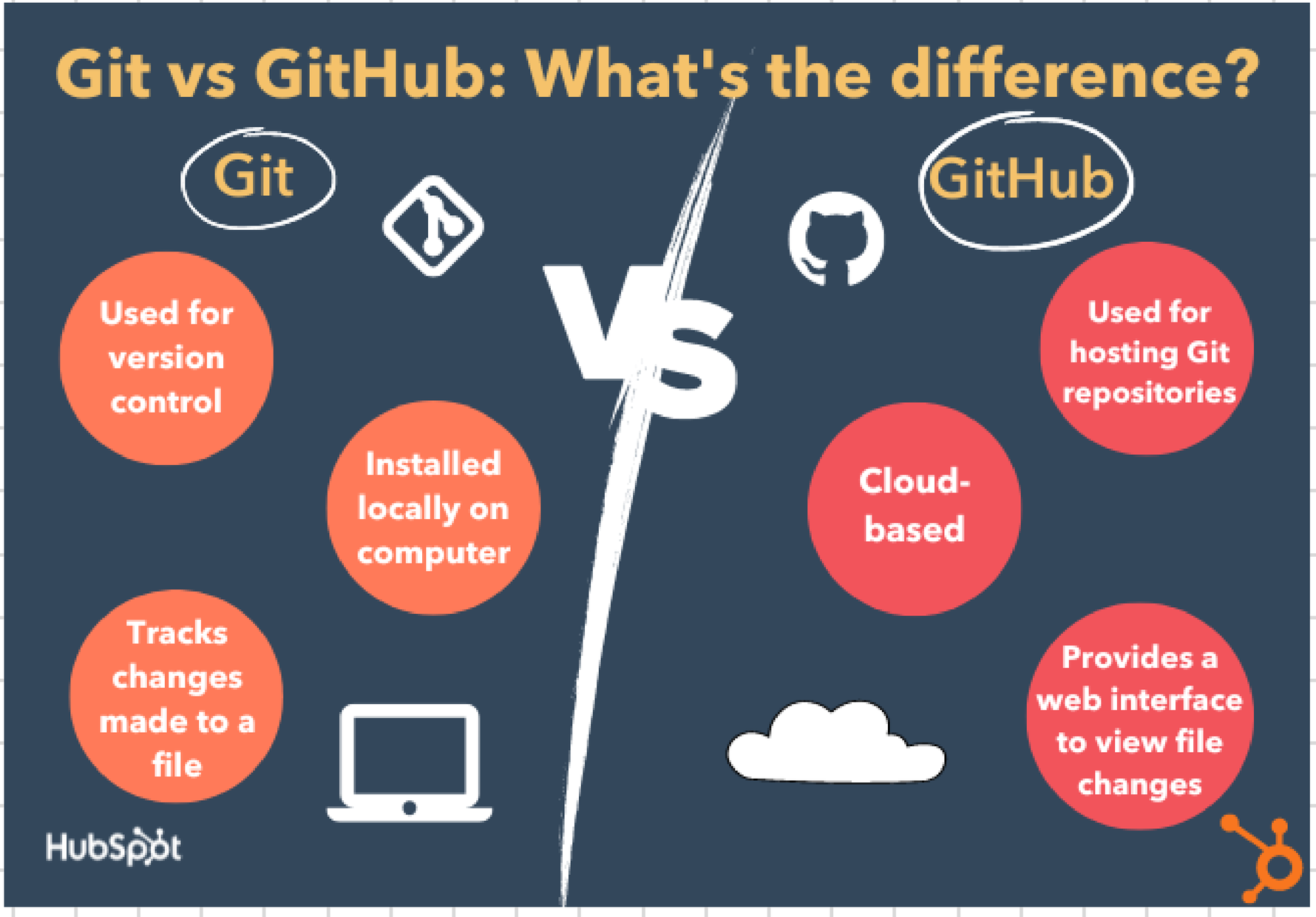
Without version control, issues like file overwrites and lost work are common. Version control provides a history of changes and makes collaboration more efficient.

Git vs GitHub

By far, the most widely used modern version control system in the world today is Git.

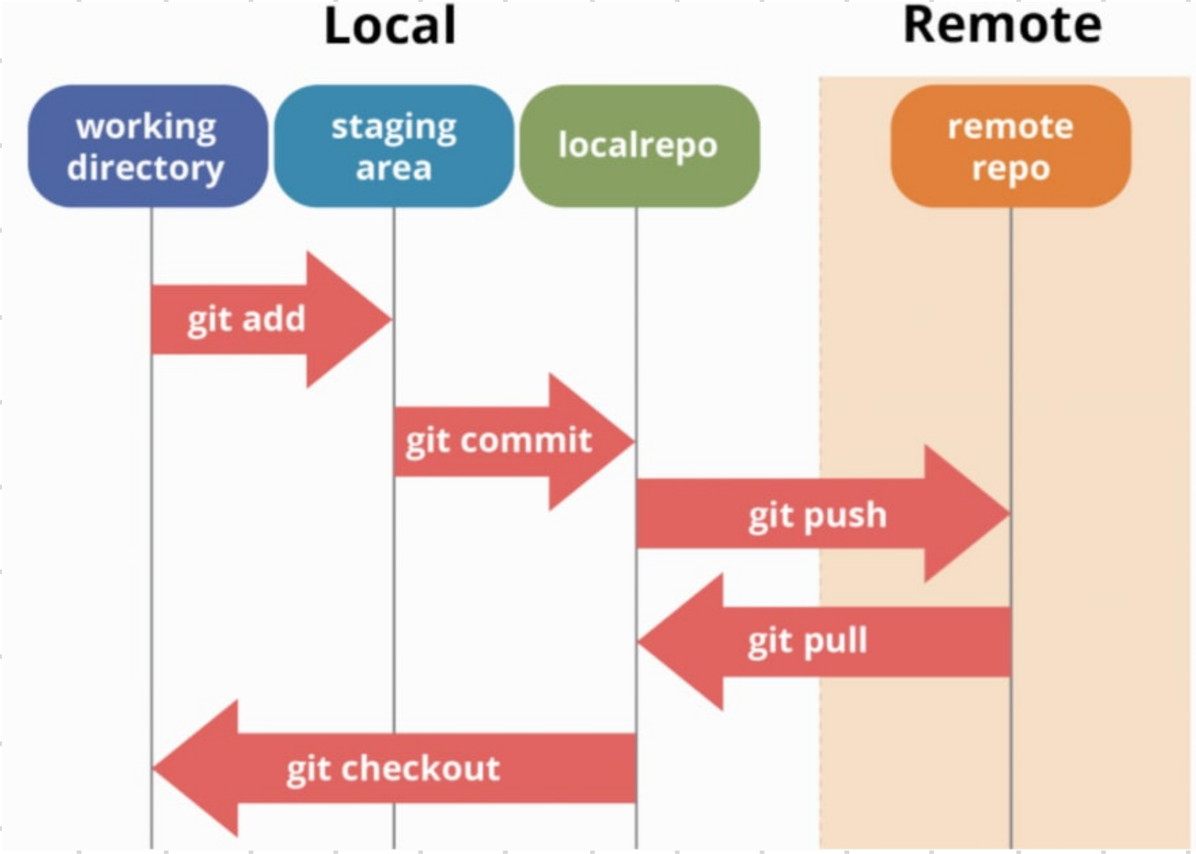
GitHub is a web-based hosting service for Git repositories. It makes Git more user-friendly and also provides a platform for developers to share code with others. In addition, GitHub makes it easy for others to contribute to projects.

Git is a version control system that allows developers to track changes in their code. GitHub is a web-based hosting service for git repositories. In simple terms, you can use git without Github, but you cannot use GitHub without Git.


Git vs GitHub: What's the difference?
Git
GitHub
VS
Used for version control
Installed locally on computer
Tracks changes made to a file
Used for hosting Git repositories
Cloud-based
Provides a web interface to view file changes
HubSpot

How Does Git Work?

Git tracks changes in files and directories.
It has local and remote repositories.



1. Initialize a Git Repository: Start a new Git repository in a project folder with **git init** (or clone a remote repo with **git clone**)

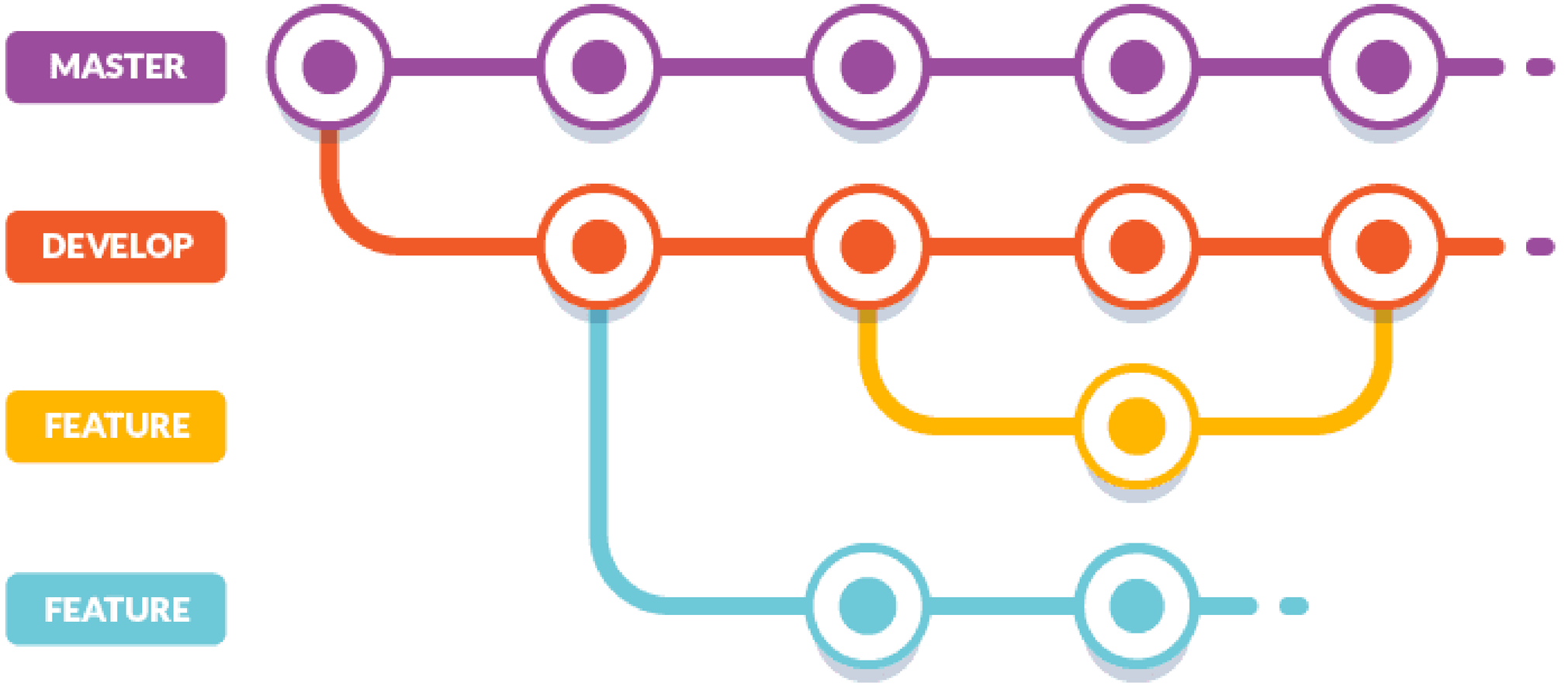
2. Stage Changes: Use **git add <file>** to stage changes for the next commit; Use **git add .** to stage all changes in the current directory; When you stage changes, you are telling Git that these changes are ready to be part of your project's history, and they will be saved in the next commit.

3. Commit Changes: Commit staged changes with **git commit -m "Your commit message"**

4. Check the Status: Use **git status** to see the current status of your repository, including changes to be committed and untracked files

5. Create a New Branch: Create a new branch with **git branch <branchname>**; switch to the new branch with **git checkout <branchname>**

6. Use **git pull** to fetch and merge changes from a remote repository into your current branch, and **git push** to push your local changes to a remote repository with


MASTER
DEVELOP
FEATURE
FEATURE

Learn

BASIC GIT WORKFLOW

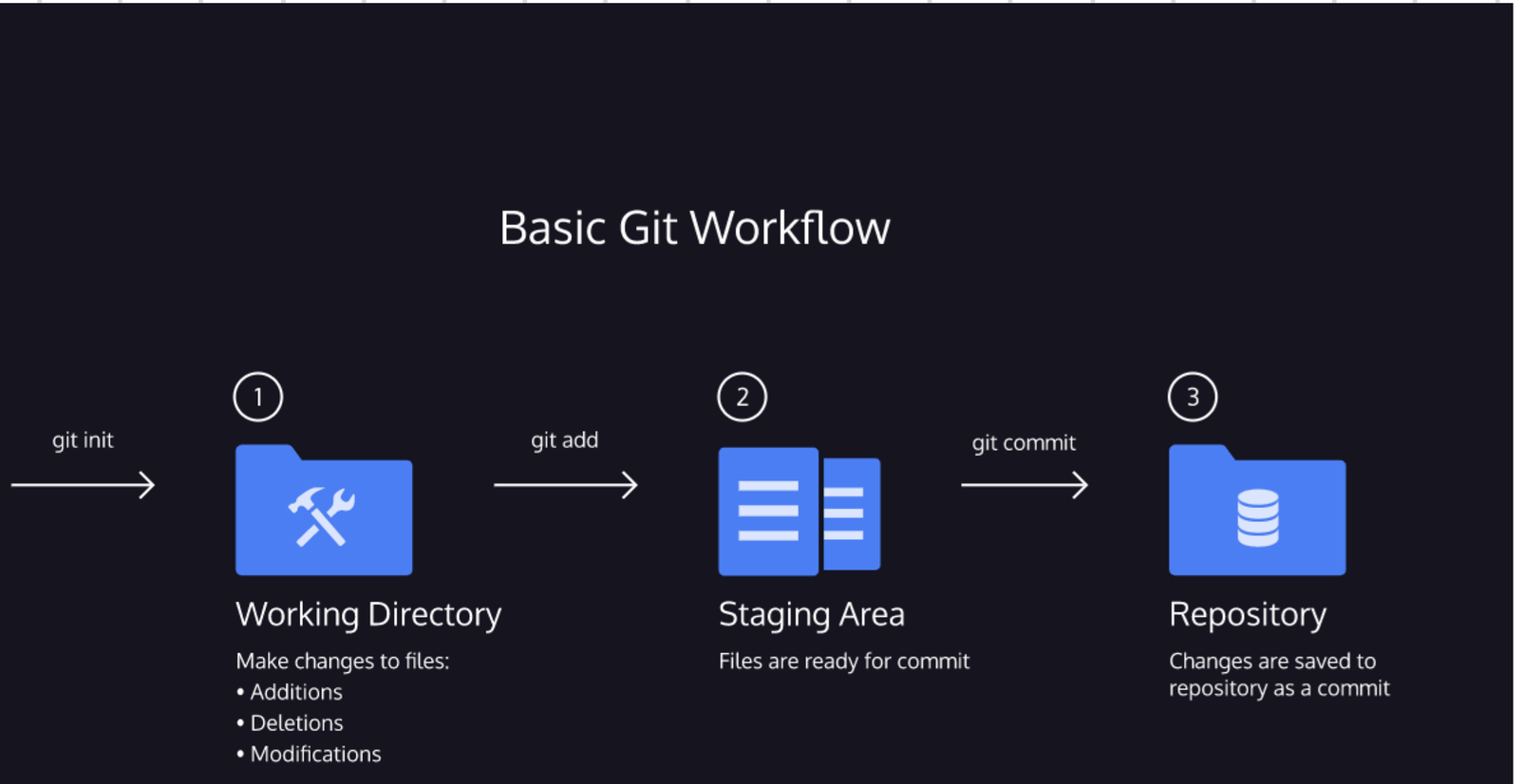
Git Workflow

Nice! We have a Git project. A Git project can be thought of as having three parts:

1. A *Working Directory*: where you'll be doing all the work: creating, editing, deleting and organizing files
2. A *Staging Area*: where you'll list changes you make to the working directory
3. A *Repository*: where Git permanently stores those changes as different *versions* of the project

The Git workflow consists of editing files in the working directory, adding files to the staging area, and saving changes to a Git repository. In Git, we save changes with a *commit*, which we will learn more about in this lesson.

Instructions



Back Next

Q&A