

# COMP1531

🏐 Teamwork

1.2 - Git - Solo Usage

# Problem

To effectively work on large projects in groups of engineers we need a more complex method of managing our code that:

1. **Version control:** Tracks changes to our code over time in a detailed and systematic way
2. **Concurrent programming:** Effectively allows multiple people to work on the same files or series of files and seamlessly integrate changes together

Programs like "Dropbox" or "Onedrive" maintain a history of files and allow syncing between multiple sources. However, they are too simple for our needs.

# Solution - Git

Git is a **version control** tool that enables to people to work **concurrently on the same codebase**.

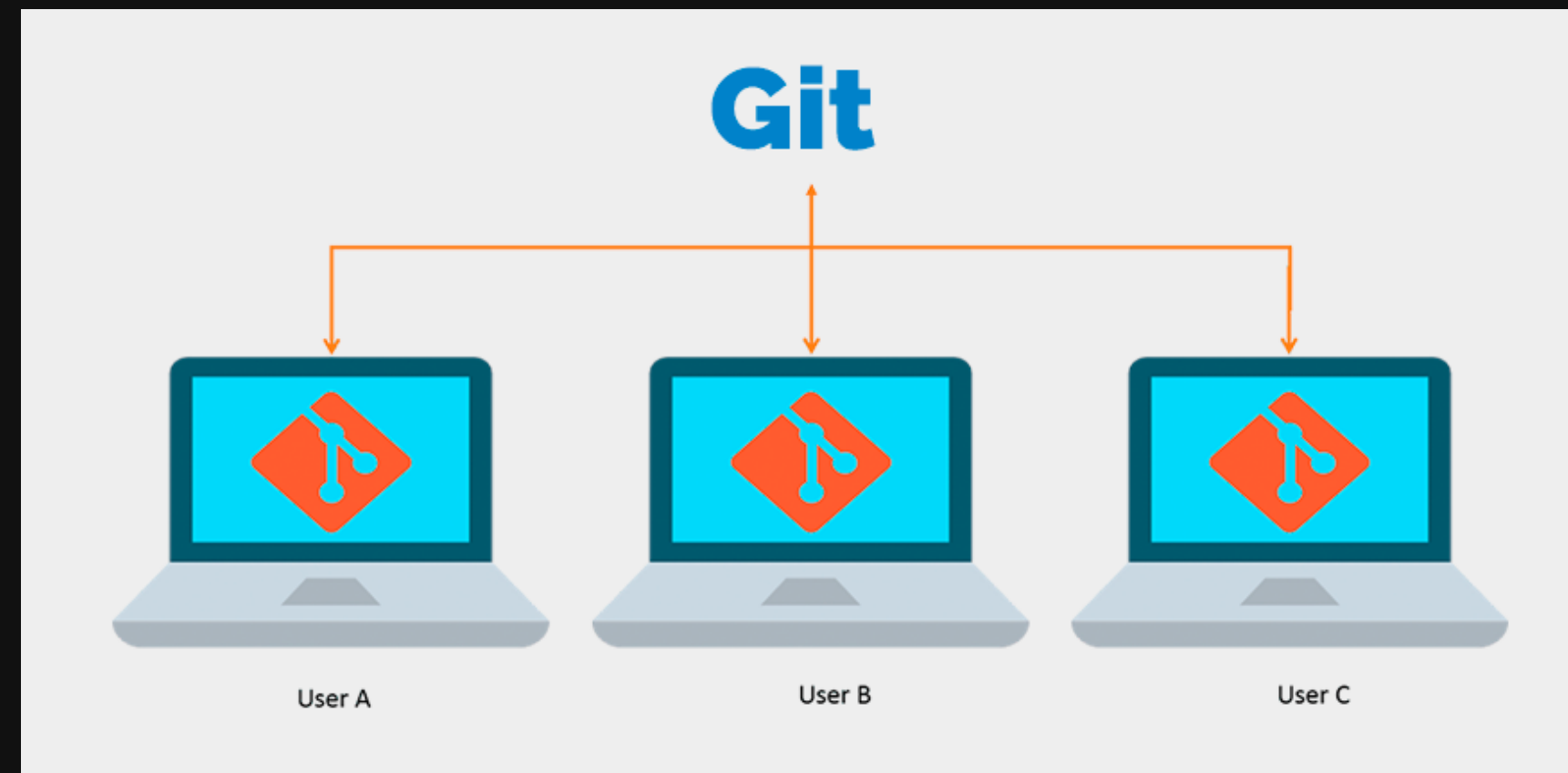
It's built for programmers, designed for managing code across lots of people with a detailed history.

There are **other solutions** to the same problem, but git continues to be popular in both industry and open source work.



# Git

Git is a **distributed version control** software. Whilst many users share work via a central cloud, each user has a full copy of the work and therefore each user has a full backup of the work.



(Source: <https://codexitos.com/what-is-git-and-why-you-should-use-it/>)

# Git Tools

Git is just a programming language, but we use specific tools to interact with it. There are 3 major git software tools that **implement the git language via an easy-to-use web app.**

They all do the same thing. Just like how Chrome, Safari, Firefox are different ways to interacting with the exact same internet.

In this course we will be using Gitlab.



# Learning Git

We're going to learn git in 3 key stages:

1. Version control on a single machine
2. Version control across multiple of your machines
3. Version control across a team of engineers

These will be practical demos. If you want to follow a written guide, then please checkout [Atlassian's git guide](#).

# Git - Single Machine

## Stage 1. Version control on a single machine

### Getting Setup

- SSH Keys
- git clone

### Status of work

- git status
- git log

### Doing work

- git add
- git diff
- git commit
- git push

# Git - Multiple of your machines

## Stage 2. Version control across multiple of your machines

### Multiple Machines

- git pull
- merge conflicts



# Git Commands Summary

The following commands are what we learn before we worry about "branches"

Command	Description	Example
git clone	Clones from a cloud repository to a local repository	git clone
git status	Tells you information about the "state" of your repo	git clone
git log	Gives you a commit history of commits made	git clone
git add	Adds a particular untracked file to your repo ready for commit, or stages a tracked file ready for commit	git add --all git add file.py
git diff	Shows the difference between the last commit and the work you've done since then	git diff
git commit	Commits changes ("takes a snapshot") of your work	git commit -m "Message name"
git push	Syncs the commit history locally with the commit history on the cloud	git push git push origin master
git pull	Syncs the commit history on the cloud with the commit history locally	git pull git pull origin master

# Feedback

