

Using Git & GitHub

Version control systems

- A **version control system** is a piece of software for controlling and tracking the version history of files – typically source code
- Viewing previous versions of files, and reverting to them, is really important in software development
- Version control helps with collaboration
 - Developers can **branch** off and work in separate places
 - When they are finished, their changes can be **merged**

Examples of VCS

- Git
 - By far the most popular
 - Several platforms use Git to provide version control features to developers
 - GitHub
 - GitLab
 - Bitbucket

Examples of VCS

- Git
 - By far the most popular
 - Several platforms use Git to provide version control features to developers
 - GitHub
 - GitLab
 - Bitbucket
- Mercurial
 - Less popular, but still used in some large organisations
 - Meta

Examples of VCS

- Git
 - By far the most popular
 - Several platforms use Git to provide version control features to developers
 - GitHub
 - GitLab
 - Bitbucket
- Mercurial
 - Less popular, but still used in some large organisations
 - Meta
- Apache Subversion
 - Not widely used anymore

Git and GitHub

- So far, we have been asking you to store your code in Google Drive so that you can access it outside of lessons
- Git is a more appropriate tool for this – but it also has a steeper learning curve, so we don't expect everyone to use it
 - In Git, our projects are stored in **repositories** (like a big folder)
- GitHub is a Git provider
- Let's create GitHub accounts now...

Using Git and GitHub

- Git is decentralised: we have a full clone of the repository everywhere it is used
 - on college PCs
 - on your PC at home
 - on GitHub's servers – this version is known as the **remote repository**
- When we make changes we **commit** those changes to the repository
- We can then **push** our commits to the remote repository
 - Until we push, we have only changed our local clone, not the remote repository
- Then, on other devices, we **pull** any changes that have been made

Commits

- Commits do not contain the state of the project at a given point
- Instead, they contain a set of changes (a **diff**)
- The current state of the project is produced by applying every diff up to, and including, that of the current commit

Working between college and home

1. Start the project at college
2. Commit your changes and push to GitHub
3. At home, pull your changes
4. Continue your work
5. Commit your changes and push to GitHub
6. At college, pull your changes
7. Continue your work

Writing commit messages

- Every commit has a message
- Typically, we describe what the commit *does* in the present tense, imperative mood
- Examples
 - Add system to get user input
 - Fix bug with integer parsing
 - Delete the whole project out of frustration

Branches & merging

- We always work on a branch
 - Typically the default branch is called `main`
- We can branch off to work on different things, and then merge our changes into `main`
- This is useful for collaboration and separating development workflows
- Collaboration can lead to **merge conflicts**
 - Two branches have edited the same file; how do we decide which changes to keep?
- Resolving merge conflicts can be trivial, but isn't always