Week 8: Version Control

Welcome to Week 8!

Version Control

8.1—Introduction

Version control is a system that tracks and manages changes to a project's files over time, allowing multiple people to work on the same project without overwriting each other's work. The purpose of version control is to organize development, maintain a history of changes, and make it easy to revert to previous versions if needed.

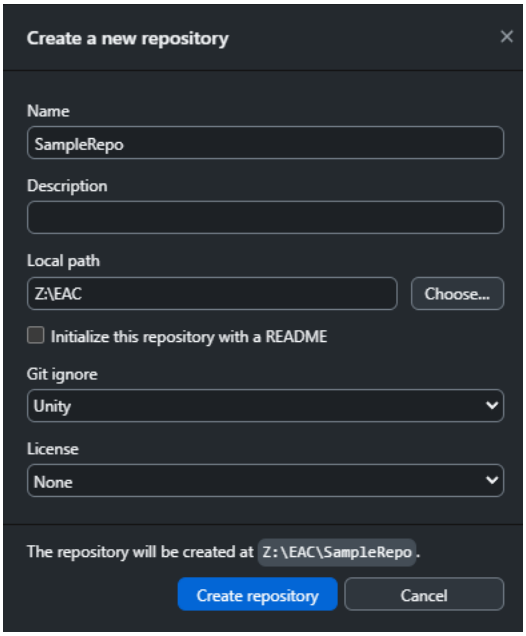
While there are several version control systems available, this course will focus on using GitHub and GitHub Desktop to manage projects and collaborate efficiently.

8.2—Repo

A repository, often referred to as a repo, is a central place where project's files, history, and changes are stored and managed. It keeps track of every update made to the project over time, allowing users to see what has changed, when it changed, and who made the changes.

The main purpose of a repository is to organize and safeguard a project while making it easy for individuals or teams to collaborate without overwriting each other's work. Using a repository has several advantages: it provides a complete history of the project, makes it easy to revert to earlier versions if something breaks, supports team work through branching and merging, and helps keep code backed up and accessible from different locations.

To create a repository using GitHub Desktop, on the main screen, click “Create a New Repository” under the File. A new dialog box will pop up to enter the name for the repository, choose the location where project files will be stored, and a description. Repositories can also optionally initialize a repo with README, Git ignore, and license. Once everything has been filled out, click “Create Repository.” After the repository is created, files can start being added to the repository, commits can be made to save changes to the repo, and publishing the repository online so others can access or collaborate on the project.



8.3—Basic commands

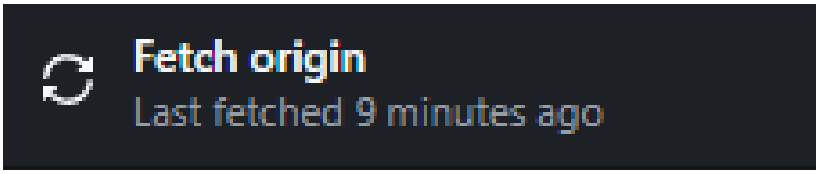
The push, commit, and pull commands are basic but essential commands in version control. The commit command saves a snapshot of your changes locally, the push command sends those changes to the remote repository, and the pull commands brings updates from the remote repository into your local copy. A simple and effective workflow is to first pull to get the latest changes, then commit your own updates, and finally push them so others can access your work. This cycle keeps projects synchronized, organized, and collaborative.

Pull

The pull command is used to fetch and integrate changes from a remote repo into your local repo. Pulling keeps the local copy of the repo up to date with the latest changes made by other collaborators. Regularly pulling changes helps prevent conflicts by synchronizing updates before making changes, maintaining consistency across the team, and making it easier to collaborate without overwriting someone’s work.

After the creating and publishing the repo, make sure that the correct repo is selected. At the top right of the application window, click “Fetch origin” button. If updates are available, the button

will change to “Pull origin.” Select “pull origin” to download and merge changes from the remote repo into the local copy of the project on the computer.

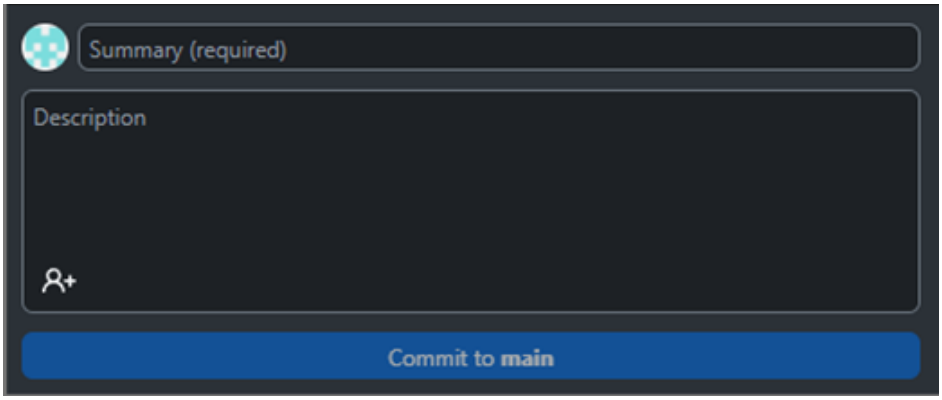


Commit

The commit command is to record the staged changes as a snapshot in your local repository’s history. Commits create a permanent record of file changes, updates project history, and every commit includes metadata for every file change with the repo. It’s important to frequently make commits of changes made to a project.

Before making a commit, it’s important to save and close the Unity project that is inside the repo. If Unity is left open while committing, it can create conflicts when pulling updates from the remote repository later. This happens because Unity continuously updates certain project files in the background as the Unity application runs.

Once changes have been made to the project files within a repo, all of the changes will be reflected in the “Changes” tab as a list. Files that are to be included within the commit are selected, then a short summary of changes are added to the “Summary” field. Depending on the complexity of the changes, a detailed description is added. Then the “Commit to (current branch’s name)” button is selected, and the commit is saved to the repo’s history. To confirm the commit, the Summary should be listed under the “History” tab.



Push

The push command transfers committed changes (commits) from the local repository to a remote repository. Pushing updates the online version of a project so that other collaborators can see updates with a project and stay in sync. Additionally, pushing changes protects your project from local data loss. Regularly pushing changes ensures that both your local and remote repositories are aligned and up to date.

Once there is at least one commit that's ready to be merged into the remote repository, there should be a "Push origin" button in the top right of the application window. After pushing changes to the remote repo, the "Push origin" button typically will change back to a "Pull origin" button.

8.4—Advanced Commands

Branch

Creating a branch is creating an isolated development within a project, allows multiple contributors to work on new features, bug fixes, or experiments simultaneously without disrupting the stable main codebase. Branching allows multiple people to work on different tasks at the same time, reduce the risk of introducing errors to the main codebase, and make it easier to test and review changes before they are merged.

The main branch is the default branch in a repository where the stable and official version of the project is stored. In the past, this branch was commonly called the "master" branch, but many platforms, including GitHub, have moved away from using that term and now use "main" instead.

At the top of the GitHub application window, there is a "Current Branch" button. Click the dropdown, then select "New Branch." A "Create a branch" dialog box will open to enter the name of the branch that describes the feature or change that's being worked on, as well as, the branch template that the branch is based on. Once that's information has been entered, click "Create branch".

Merge

The merge command in version control is used to combine changes from one branch into another, bringing together different branches of development back together in a single branch. It integrates work that has been done separately, like a feature or bug fixes, into a single, unified codebase. When a merge is performed, the version control system compares the changes differences between the branches and applies any changes to the target branch. This comparison is used to also identify if there are any conflicts between the two branches that need to be resolved before the merge can be completed.

Using the merge command, helps keep the main project up to date with completed work, preserve the full history of changes, and allow for smooth collaboration between team members by allowing developers to work independently without interfering with each other's progress.

To merge branches in a repo, confirm that all changes to a branch have been committed and pushed. Switch to the branch that the changes made in another branch are to be merged into. Click "Branch" in the top menu and choose "Merge into current branch", then select the branch whose changes are going to be added from. If there are no conflicts, the merge will complete

automatically. If there are conflicts, there will be a prompt to resolve the conflicts before completing the merge.

8.5—GitIgnore

A gitignore file is a special file in a repository that tell Git which files or folders should be ignored and not tracked. Its purpose is to prevent unnecessary or sensitive files from being added to the repository. Including a gitignore file keeps repositories clean and organized, reduces clutter from files that don't need to be shared, protects sensitive information, and prevents commits of large unnecessary files.

When creating a new repository, GitHub and GitHub Desktop allow gitignore templates that are based on the type of project that's going to be added to a repo.

To modify a gitignore file, open the gitignore file using a text editor, like Notepad or VS Code. Your gitignore file should be located in the root of your project folder. Inside the file, file names or folders can be added for Git to automatically ignore. After saving changes to the gitignore file, the gitignore file should be committed and pushed to the remote repo.

8.6—Exercises

Reinforcement

The reinforcement section is used to check the understanding of the information that was presented in the PowerPoint/document. Please answer the questions in your own words.

R—8.1 What is version control, and why is it important for managing projects?

R—8.2 How does a repository help individuals or teams collaborate?

R—8.3 What are the three basic commands in version control?

R—8.4 What is difference between a main branch and a master branch?

R—8.5 What is purpose of the merge command in version control?

R—8.6 What is a gitignore file, and what is it's purpose in a repository?

R—8.7 How can you modify a gitignore file, and what must be done after editing it to ensure the changes are applied?

Project

The project section's tasks are used in work toward creating the final Flying Bird game project for the end of the course.

P—8.1 Make a repo with the name “Unity101”, a README file, Unity gitignore template.