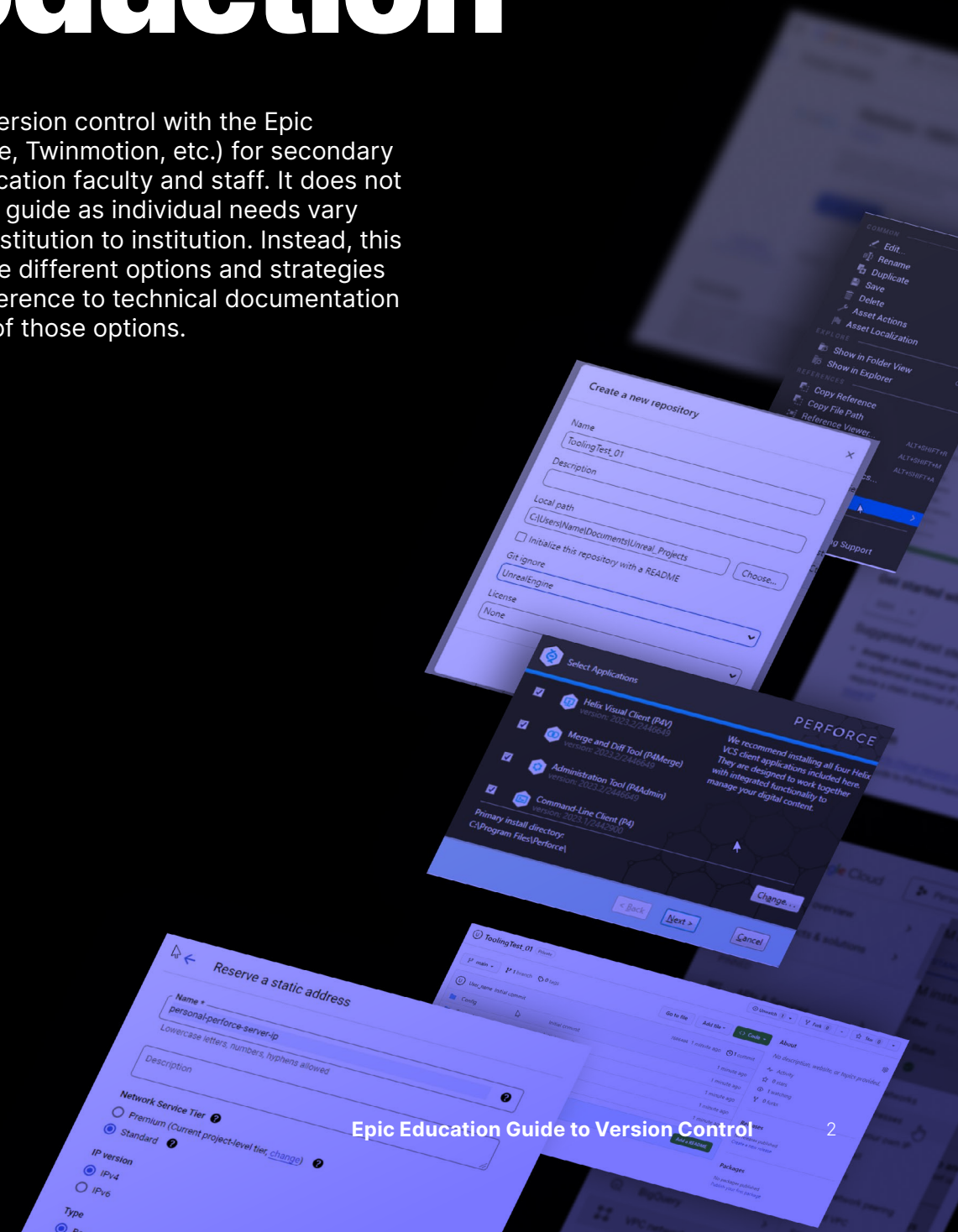
Epic Education Guide to Version Control

Introduction

This is a guide to using version control with the Epic ecosystem (Unreal Engine, Twinmotion, etc.) for secondary and post-secondary education faculty and staff. It does not purport to be a complete guide as individual needs vary from course to course, institution to institution. Instead, this document will present the different options and strategies available and provide reference to technical documentation for the install and setup of those options.



What is version control?

Software and game development are team-based activities, often with many people working in similar roles. In your class projects, for example, you might have three students working on code, one on technical art, two on game design, and six on artwork. Each of those team members need to be able to contribute to the project without damaging or deleting existing work. Over the history of software development, this problem has been tackled using **version control**.

Version control is a software system, which once applied to a project, allows for each team member to:

- View the existing state of the project
- Contribute work to the project
- Have contribution authorship tracked by user
- Return to a previous version of the project on an ad-hoc basis

Why have version control?

Version control is not strictly necessary when developing games and software. There are times when its benefits are disproportionate to the setup and maintenance labor—on shorter individual/small team projects, often just using USB drives with sufficient discipline is fine. With larger teams, however, passing data around can get problematic very quickly—files can be overwritten, corrupted, or lost.

Ultimately, the use of version control is recommended simply because it's what is used in industry. By enabling your students to learn how to use it, make mistakes, and understand its benefits, you are preparing them to enter the industry ready to work.

There is an added benefit from an educator's standpoint: by using version control, you are able to see exactly who has contributed what to the project. Because each change is marked by an author, reviewing the version control database enables you to see how your student team works.

Version control options

There are several version control options available, each with their own idiosyncrasies and benefits. Some, like SVN, are older legacy systems that while still in use in some studios, generally are no longer recommended as a starting point. For the purposes of this introduction, we will be looking at two systems: **Perforce** and **Git.**

The following are quick descriptions of each software, including key terms. This is intended as an introduction. For more detailed information on each, please see Appendix A.

Perforce

While Perforce is the name of the company that makes the software, it is common to also refer to the software itself by that name. Technically, the software itself is called **Helix** but throughout this guide we will refer to it as Perforce.

Perforce is a centralized system for version control: there is a central server that stores the project data and the version control data in a depot and users connect to it using client software.

The Perforce model can be easily compared to a library, where a user can check out a book or resource. This resource can be edited and revised, but as long as that user has that resource it is locked from anyone else editing it. When finished, the user checks in the resource and the software updates its version and makes it available to the team.

Benefits

- Each resource in the system has a record of changes and in Perforce these changes are saved as *deltas*—that is, it only saves the changes made since the last version. This cuts down on data space as often even in large files those changes can be quite small.
- Perforce handles *binary* files (images, sounds, other non-text-based media) well.

Complications

- Because the resource is locked while editing, no other user can make changes; but issues can arise when that user neglects to check the resource back in.
- Setup can be complicated and requires knowledge of command-line systems like UNIX.

How to get it

Perforce has a free version for academic institutions and one for student teams with five or fewer members.

Git

Git is open-source software designed by the same person who originally created the Linux operating system to help with the unique challenges of version control for open-source projects. By this nature, it has a decentralized or *distributed* system model. While you can make your own Git server, it is very common to host projects at services like **GitHub** or **BitBucket**.

Unlike centralized systems where single resources can be checked out, with Git the entire project, called a *repo*, is duplicated locally on the client computer, a process called cloning. Then, the user can make a new copy of the project, called a *fork*. The user is then free to push as many changes as they like to the fork, with the knowledge that other users might be making similar or conflicting changes on their own local computers. Once a given user has a series of changes that are ready to go, they propose those changes to the owner of the original repo in an action called a *pull request*. The owner of the repo then reviews the code, handles any conflicts by *merging* the code, and then closing the pull request.

If we were to apply the library metaphor from above, it would look like this: a user sees a book on the shelves and photocopies the whole thing. They then make any changes they like to the photocopy and when done, they send the new pages back to the library. The author of the original book then reads the new pages and decides to include them in their book or not.

This may seem very complicated! Again, this system was designed for open-source software development, where the fundamental premise is that anyone can make their own versions of any open-source project from anywhere, so there needs to be more safeguards and processes in place to protect your code.

Benefits

- Each team member will have their own copy of the project at all times. If any repo fails, there are multiple redundant copies.
- Setup is very easy and fast.

Complications

- The fork-push-pull-merge pipeline can be quite complex and require discipline from the main owner of the project.
- Because Git was designed for coding projects, it handles text files very well. It does have some issues with binary files and services like GitHub often have a file size limit.

How to get it

GitHub accounts are free for all, with the condition that the repos are public. Educational institutions can apply for an institutional account, which makes related student repos private.

For a more detailed discussion of the Perforce and Git pipelines, please see appendix A.

Comparison table

	Perforce	Git
Application process	Submit a form to Perforce for approval.	Create an account with a provider like GitHub.
Server setup	Either local servers or cloud-based servers. Process can be complicated and requires command line experience.	Easiest with service providers like GitHub and BitBucket. Very easy set up.
Binary support	Built-in	Requires external support
Industry acceptance	Very high in game development.	Very high in general programming, light use in game development.

Guide to Perforce

Requirements

Perforce is a server-based solution and as such you will need server space to host the server software and the storage space.

- **Server space**
    - Local Windows or Linux server, managed by your IT department, or
    - Cloud-based services
        - AWS
        - Azure
        - Google
        - Digital Ocean
    - A static IP address, usually available through your cloud provider or IT department
- **Perforce Helix server software**
- **Perforce Helix visual client**

Server

Helix Server is the executable on your server that manages the data and version control. The executable itself is called **P4D**, short for Perforce Daemon. The server runs continuously, handling any requests from clients. Clients can be one of any of the following:

- The command-line client **P4**. This is the most robust and powerful way to interact with the server but also has the fewest safeguards. It is very possible to delete data without protection using this tool.
- The Admin tool **P4Admin**. This is a visual client for administering the server including adding new users, groups, and projects. It does not provide all of the administrative tools needed—for that, you will still need to use P4.
- The visual client **P4V**. This is a visual client for end users—your students and staff working on projects. More information is below.

For small teams

For the free-use Perforce for teams of five of fewer, you can use cloud-based subscription services. Prices for the server and storage space starts around $10 - $15 per month.

Currently, Perforce recommends the following service providers:

- AWS* **(documentation)**
- Azure* **(documentation)**
- Google **(documentation)**

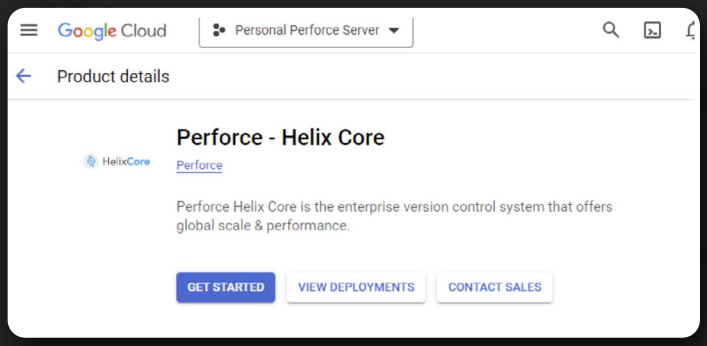
AWS and Azure have the option of the Enhanced Studio Pack, which includes Perforce's Hansoft project management tools and other features. This is usually not needed for educational purposes but you can install the pack without having to use the extra tools.

Setup

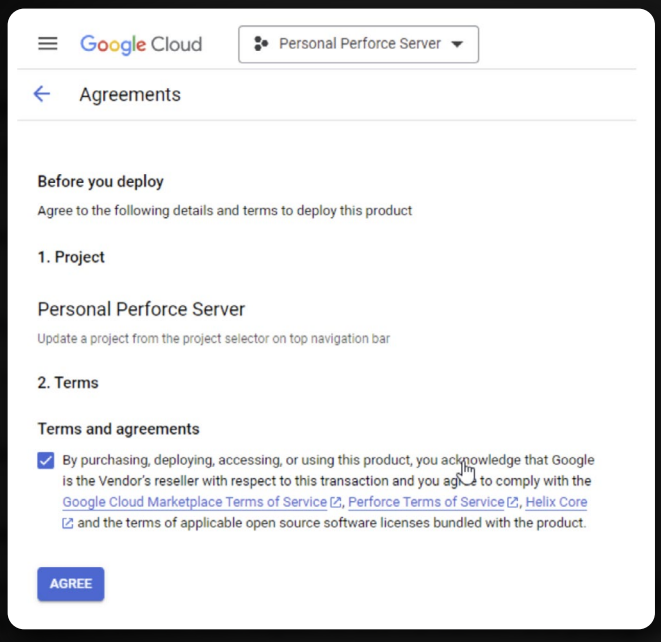
For the purposes of this document, we will go through the process using Google. But please make sure to research available options for suitability with your particular ecosystem.

The first step is to install the Helix Core server. Most of the cloud providers have a preformatted install you can use. With Google Cloud, you can access the package through the **Google Cloud Marketplace**. Search for 'Perforce', and click on the result link.

Click on 'Get Started' to start the installation process.



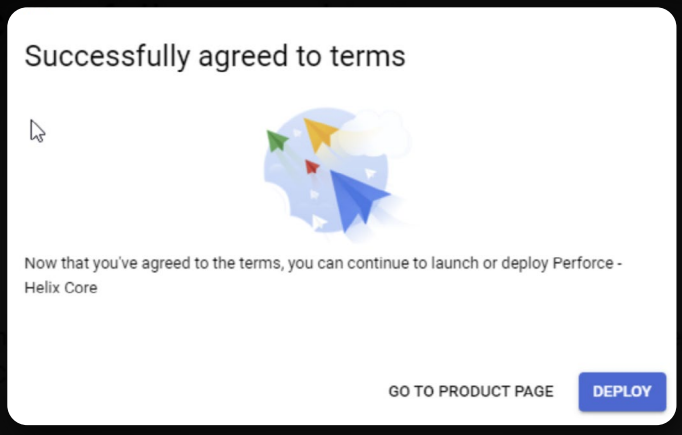
The following dialog presents the terms to agree. However, this dialog also is where you set the project you're installing the server to. If you have multiple existing projects, make sure to check you're installing on the correct one. If you have no existing projects, click on the project dropdown in the menubar to create a new project.



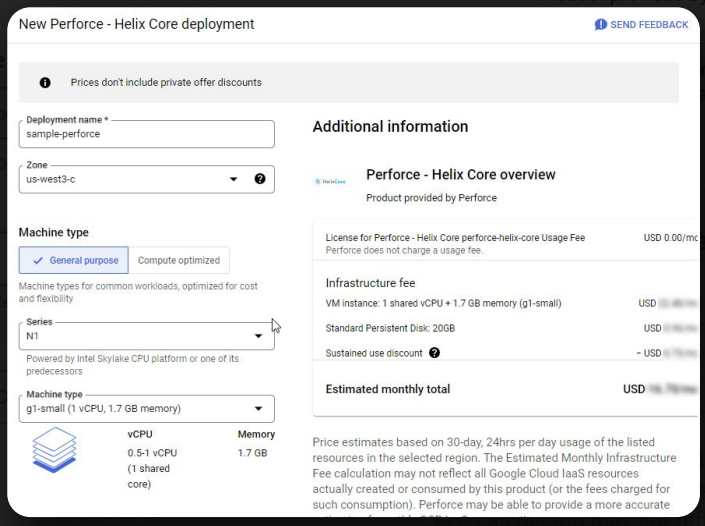
When you create a new project, ensure you give it a readable and descriptive name.

Click Deploy.

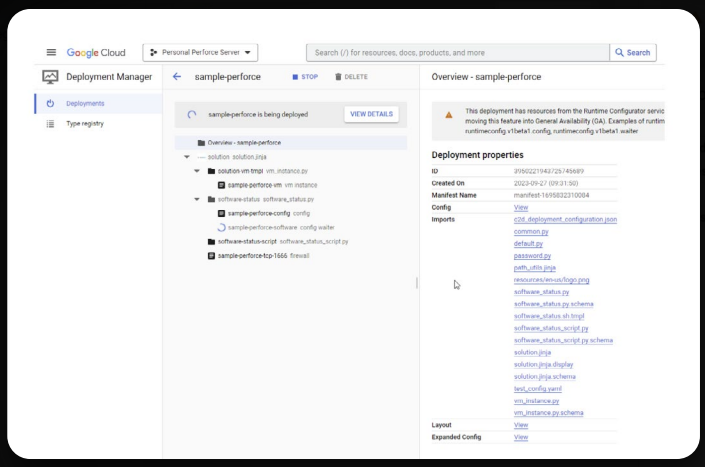
When the installation is complete, you will need to configure your server for your needs. On the overview page, you can further specify your server configuration, including geolocation of the server, how much memory is allocated to the project, and your expected monthly charges.



Click Deploy at the bottom of the page.

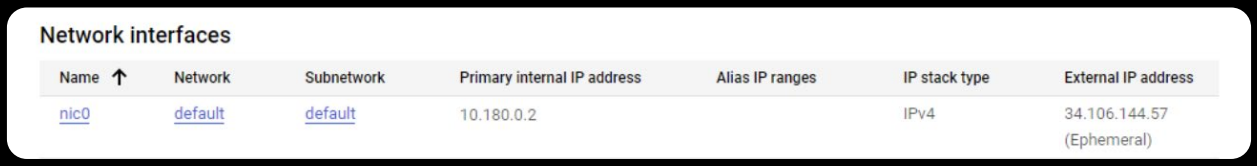


Wait while all the services start up—this may take several minutes. When everything is up and running, you can then access the Perforce server through SSH.



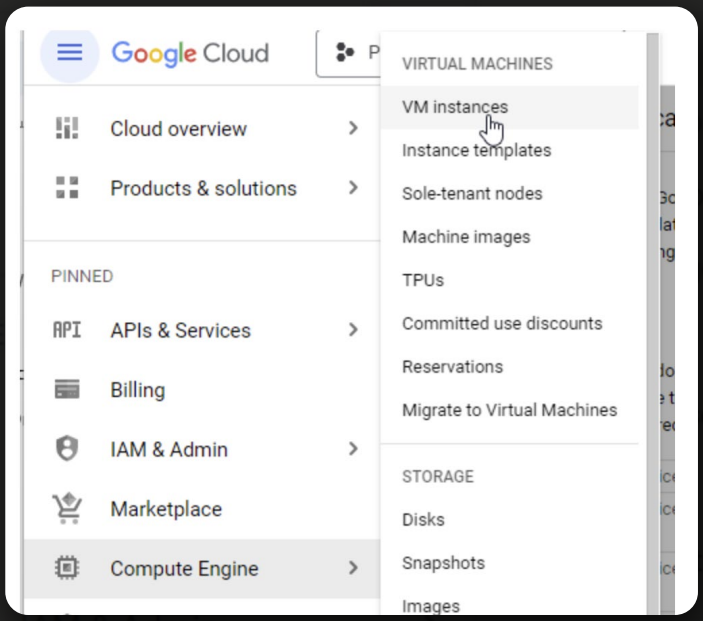
Setting the Static IP Address

The server IP address is the internet address used to locate your server. When initially created, the server will have an *ephemeral* IP address, which means that it might change at any time based upon the cloud provider's optimizations. To keep your Perforce server working, you will need a *static* IP address. Your cloud server provider will have instructions on how to promote your ephemeral IP address to a static IP.

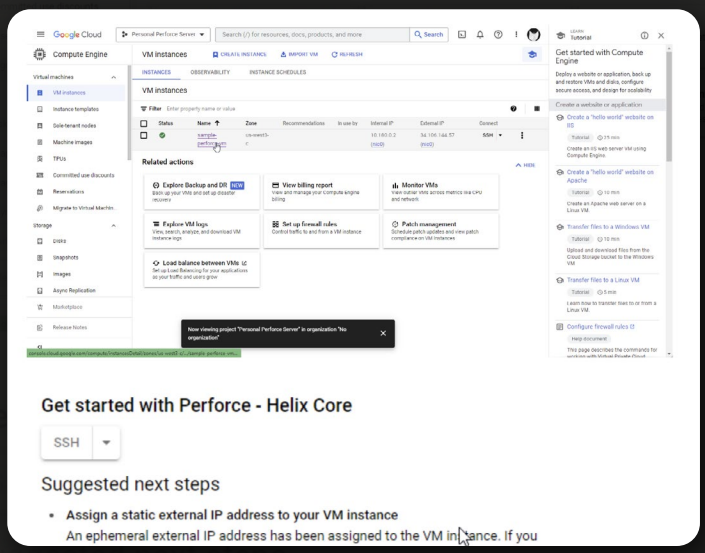


To check your IP status in Google Cloud, in the Google Cloud console click on the menu burger at the top left of the menubar and select Compute Engine > VM Instances.

Click on your instance.

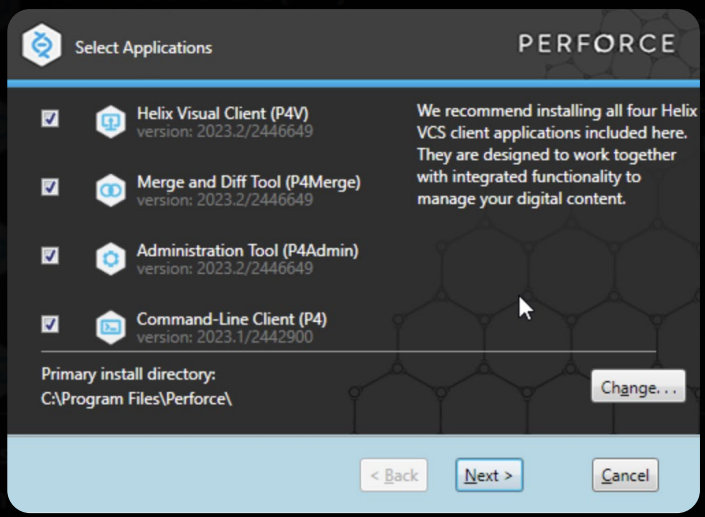


The process of promoting your IP address to a static address tends to change frequently. Please check your cloud provider documentation to see how to perform this promotion.



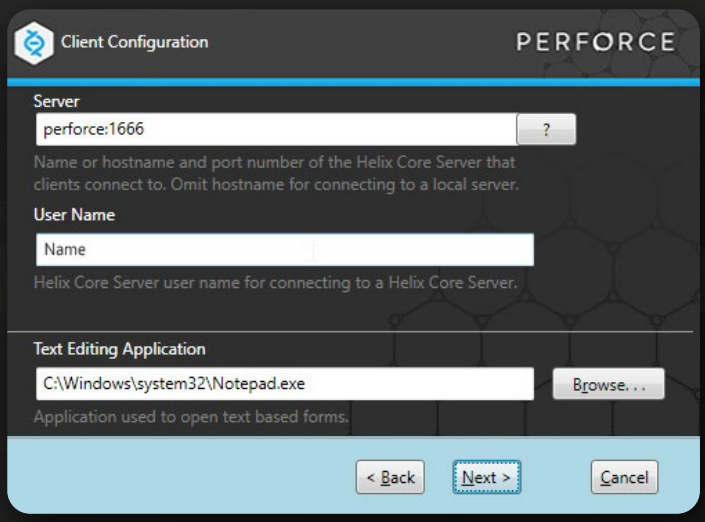
Client install

The Perforce client install is straightforward. Once the install package is downloaded, open it on your client computer.



Make sure to change the address to the IP address of your cloud server.

Hot tip: Keep the install executable as it is also the way you can repair or uninstall the software.



Log in as super

The first time you log in, you have to use the special super account `perforce`. The password is generated automatically, but can be a little tricky to figure out, as you need to know your server **Instance ID**. To get that value, go to the **Google Cloud Console.** Select your Perforce server from the dropdown menu at the top.

Scroll down to the `Compute Engine` button and click it. You will be presented with the Compute Engine dashboard, which has a list of your Virtual Machines (VMs).

Click on the name of your VM to get the basic information about it. In the list, you will find an Instance ID value. Make a note of this number.

You can now log in using the superuser `perforce`. Open P4V on your client, and enter your credentials. The password for the `perforce` user is “i-<instance id>”

For institutions

Application process

For institution-wide use across multiple teams and projects, you will need to apply for a free educational license from Perforce. The process is simple, and starts at **How to get your educational licenses**. You will need to sign an user agreement and provide data to Perforce. The data you'll need at hand are:

- How many users are expected
- A static IP address and port for the server. Perforce defaults to port 1666.
- Contact information

After you submit, the application will be reviewed, and if successful, a .lic license file will be emailed to you. Once your server is active, this license file is uploaded to the server using the Admin tool.

Set up

Depending on the type of server you have, information about installation can be found here:

- **Windows**
- **Linux**

There is also a setup **workflow document** that is very useful in laying out all the steps along the way.

Notes

When presented with a dialog to create the initial user, make sure you choose a memorable username that, for security reasons, isn't `admin`.

This user will be the superuser for the server—the one account that has access to change anything on the server. As such, the superuser credentials should be protected.

In addition, you may have to prepend `ssl:` to your server address in order to connect securely.

After installing the server, you should download and install on a local computer the visual client (see below). The first time you use P4V, you will log in using the superuser account. It will then ask you to provide user information for the superuser. Again, make sure your details are correct.

Once installed, the P4V client gives you access to the administration tool, P4Admin, via the `Tools > Administration` menu. In the admin tool, you can set up your security for the server. For more information, see **Securing the server**.

Understanding the server

Unfortunately the Perforce server isn't a one-step install and go process. Once you have installed the server software, it is critical that you configure and set up the server to your needs. Configuring your server is probably the most important step of getting your Perforce version control working correctly.

Server protection

Server files

Perforce uses three main types of files:

- **Metadata.** These files define the database itself and have names that start with "db." They should be on the same drive as the server software.
- **Journaling.** The journal file is a record of all transactions performed by the server—the records of when, who, what. There can be many of these files. It's best to put these on a different drive than the metadata files as they are critical in restoring data in the case of server or storage failure.
- **Versioned.** These are the actual project files. For speed, it's best to put them on the same drive as the server, but they can be elsewhere. See below for more information.

Storage space

Game projects can take up a lot of drive space—up to several gigabytes depending on your development pipeline. Even with many gigabytes of storage available, it is relatively easy to run out of storage space.

There are several potential solutions to this problem:

- Provide a lot of storage on the same server that hosts the server software. This is the default behavior and is easy to set up from the start of deployment. Resizing the storage to accommodate more room can be tricky.
- Connect the server software to external storage solutions. This takes more set up and technical knowledge but enables you to resize or distribute the storage space quite easily. You may notice a small delay on version changes.

Depots

It is generally a good idea to have one separate depot per project. You can create new depots from P4Admin. When you are first setting up the server, consider your naming convention for the depot—for example, you might want to start with the year, course number, or other prefixes. Having some sort of organization in your server can only help.

Example:

```
2023_289301_
EndlessClimber
2023_289301_RunNBun
2024_289101_
LevelDesign_01
```

Typemaps

The typemap is Perforce's way of determining what kinds of files are saved on the server, how they are saved, and what permissions they have. Getting your typemap correct is an essential step for getting your Perforce server working properly and is the most common source of error.

To set the typemap, you can use the **typemap command.**

Each line in the typemap assigns permissions to a filetype. Each line consists of the following information:

1. **Data type.** Usually either binary or text.
2. **Lock permissions.** Usually either locked (l) or writable (w). When locked, only the user who has the asset checked out can edit it. When writable, a user who hasn't checked out the asset can still edit it locally. Text data type files do not need lock permissions as text files can be merged.
3. **File path.** The `//...` specifies that the rule applies to everything in the depot and the `.uasset` is the corresponding file type.

Unreal Engine typemap example:

```
Typemap:
binary+l //....uasset
        binary+l //....umap
        binary+w //....3ds
        binary+w //....abc
        binary+w //....ai
        binary+w //....aiff
        binary+w //....blend
        binary+w //....bgeo
        binary+w //....bmp
        binary+w //....dae
        binary+w //....dds
        binary+w //....dxf
        binary+w //....exr
        binary+w //....fbx
        binary+w //....flac
        binary+w //....geo
        binary+w //....gif
        binary+w //....glb
        binary+w //....gltf
        binary+w //....hip
        binary+w //....hipnc
        binary+w //....hmv
        binary+w //....jpg
        binary+w //....jpeg
        binary+w //....ma
        binary+w //....mb
        binary+w //....mp3
        binary+w //....mp4
        binary+w //....obj
        binary+w //....ogg
        binary+w //....pic
        binary+w //....picnc
        binary+w //....png
        binary+w //....psb
        binary+w //....psd
        binary+w //....sbsar
        binary+w //....skp
        binary+w //....spp
        binary+w //....tga
        binary+w //....tif
        binary+w //....tiff
        binary+w //....usd
        binary+w //....wav
        binary+w //....wmv
        binary+w //....vfl
        binary+w //....zpr
        binary+w //....ztl
        binary+w //....exe
        binary+w //....dll
        binary+w //....lib
        binary+w //....app
        binary+w //....dylib
        binary+w //....stub
        binary+w //....ipa
        binary+Sw //....pdb
        text //....c
        text //....config
        text //....cpp
        text //....cs
        text //....h
        text //....ini
        text //....m
        text //....mm
        text //....py
        text //....txt
        text+w //....DotSettings
        text+w //....modules
        text+w //....target
        text+w //....version
Setting up users
```

Client

The client is how your students and staff will interact with version control. You can log in to the server, check out resources, create changelists, upload new versions, and rollback to prior versions.

You can download the client for any platform here: **Download Helix Visual Client (P4V)**. Each client computer will need the client installed.

Once installed, you can follow the **Client Quickstart** instructions.

Configuring the client

Ignore files

You can specify file types that Perforce should ignore using the `P4Ignore` file. This file tells the client which file types to ignore so that they are not even seen by the client. This makes it clear which files can and cannot be added to the database.

To create an ignore file, follow the **Using P4IGNORE with P4V instructions.**

Unreal Engine P4Ignore example

```
Saved/
LocalBuilds/
*.csproj.*
.vs/*
*.pdb
*.suo
*.opensdf
*.sdf
*.tmp
*.mdb
obj/
*.vcxproj
*.sln
*-Debug.*
FileOpenOrder/
*.xcworkspace
*.xcodeproj
./Makefile
./CMakeLists.txt
.ue4dependencies
ipch\*

# These files add clutter
Samples/*
FeaturePacks/*
Engine/Documentation/*

# Engine intermediates
Engine/Intermediate/*
Intermediate/

# Intermediate folders for programs should not be checked in
Engine\Programs\*\Intermediate\*

# Intermediate folders created for various C# programs
Engine\Source\Programs\*\obj\*

# Saved folders for programs should not be checked in
Engine\Programs\*\Saved\*
Engine\Programs\UnrealBuildTool\*

# Derived data cache should never be checked in
Engine/DerivedDataCache/*

# Ignore any build receipts
Engine/Build/Receipts/*

# Ignore personal workspace vars
p4config.txt

# Ignore Unix backup files
*~

# Ignore Mac desktop services store files
.DS_Store

# Ignore crash reports
crashinfo--*

# Ignore linux project files
*.user
*.pro
*.pri
*.kdev4
```

```
# Obj-C/Swift specific
*.hmap
*.ipa
*.dSYM.zip
*.dSYM

# Ignore documentation generated for C# tools
Engine/Binaries/DotNET/UnrealBuildTool.xml
Engine/Binaries/DotNET/AutomationScripts/BuildGraph.Automation.xml

# Ignore version files in the Engine/Binaries directory created by UBT
/Engine/Binaries/**/*.version

# Ignore exp files in the the Engine/Binaries directory as they aren't C/C++ source files
/Engine/Binaries/**/*.exp

# Ignore Swarm local save files
Engine/Binaries/DotNET/SwarmAgent.DeveloperOptions.xml
Engine/Binaries/DotNET/SwarmAgent.Options.xml

# Intermediary Files
*.target.xml
*.exe.config
*.exe.manifest

#
# Setup.bat Ignores
#
# There already might be some files pushed into the repo that would otherwise be
# filtered: those are from the initial commit, coming directly from UE4-GitHub.
# After running Setup.bat, the folders below will be populated with ~46GB of
# files. Hence we'll be blanked ignoring some directories such as ThirdParty,
# Media, Content, etc. If we want to change anything in those blanked-ignored
# folders, we should unignore the specific thing we're adding.
#
.git\*
Samples\**\Content\*
Samples\**\Media\*
Templates\**\Content\*
Templates\**\Media\*
Templates\Media\*
Engine\Platforms\**\Content\*
Engine\Documentation\*
Engine\Extras\*
Engine\Content\*
Engine\Build\*
Engine\Source\ThirdParty\*
Engine\Source\Programs\*
Engine\Source\Developer\*
Engine\Plugins\*


# Ignore project-specific files
*/Build/Receipts/*
*/DerivedDataCache/*
*/Binaries/*
*/Intermediate/*

# idea
# Default ignored files
/shelf/
/workspace.xml

# vs code
/.vs/
.vs/
.vs/*
```

Resources

Complete Guide to Cloud Version Control

Helix Core Cloud Deployment Using VM Images Guide

Google Cloud Marketplace

How to Get Your Educational Licenses

Setting Up Perforce Helix Core: How to Get Started Fast

Planning a Helix Core Server Installation

Installation and Setup Workflow Documentation

Setting the Typemap

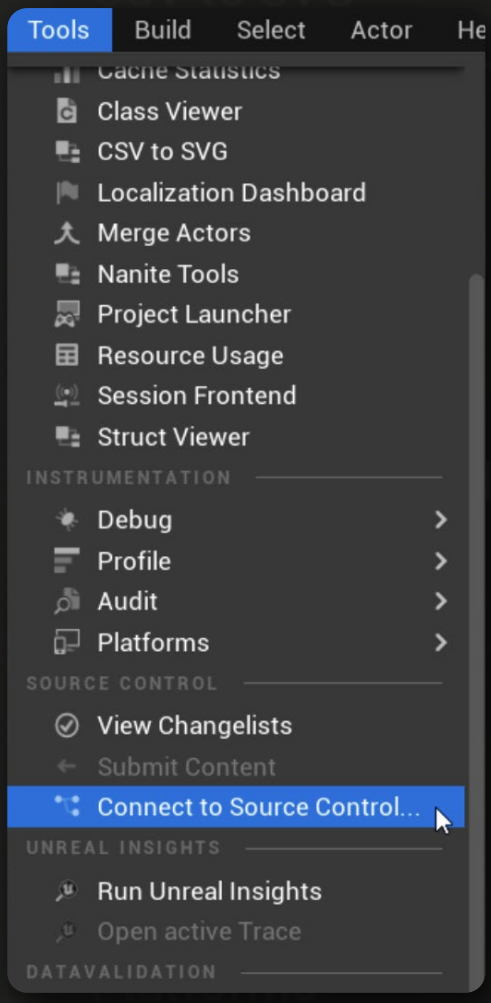
Perforce Cheatsheet

Virtual Production Training

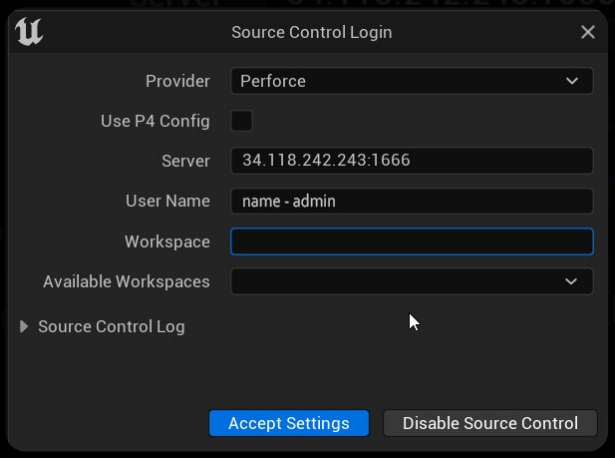
Using Perforce with Unreal Engine

You can use the P4V Visual Client with Unreal Engine projects but you can also connect directly to Perforce from within Unreal Engine. To do so, follow these steps:

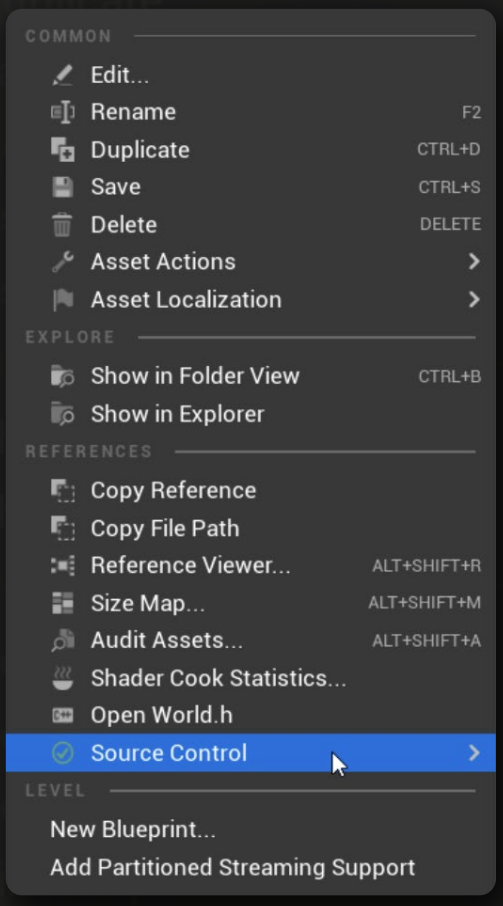
Select
`Tools > Connect to Version Control`



Fill in the dialog with your server information:



Once connected, the Content Browser has a contextual menu for version control:



Guide to GitHub

Requirements

Git is free software that you can install on any server. If you're concerned about data and privacy, you can install on a local server—but most people use the GitHub service. It's free, and offers excellent service and the whole thing is set up and ready to go. Setting up a Git server is beyond the scope of this document, but the **Pro Git book** is available online for free.

For individuals

To use GitHub, you need to **create a GitHub account.** Once you have your account, make sure you enroll in the **Global Campus** program, where you can get a GitHub Pro account for free.

For institutions

With a GitHub account, you can **create Organizations.** These operate as ways to collect users and projects under one umbrella. So you could have an organization for your department, your faculty, or on a project-by-project basis.

Additionally, your institution can apply for the **GitHub Campus Program**. This program offers free education accounts for all students, enterprise-level GitHub access, and often a lot of free swag. To join the program, your school needs to:

1. Offer GitHub to all of your technical departments
2. Add your school's logo to the GitHub Campus Program partner school website
3. Agree to receive regular announcements from GitHub Education
4. For each department using GitHub, appoint an administrator to complete the teacher-training program

Setup

Server

Because GitHub is entirely web-based, there is no server software to configure.

If you are planning on hosting your own instance of Git, refer to the **server documentation**.

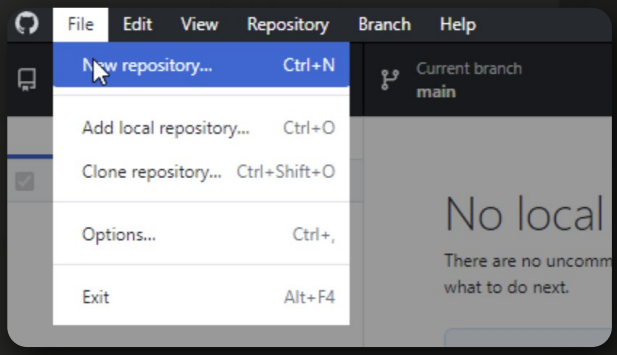
Client

GitHub Desktop is the standard GitHub client and is a simple install. There are other client alternatives, such as **Tower**, that offer expanded functionality.

Creating a repo

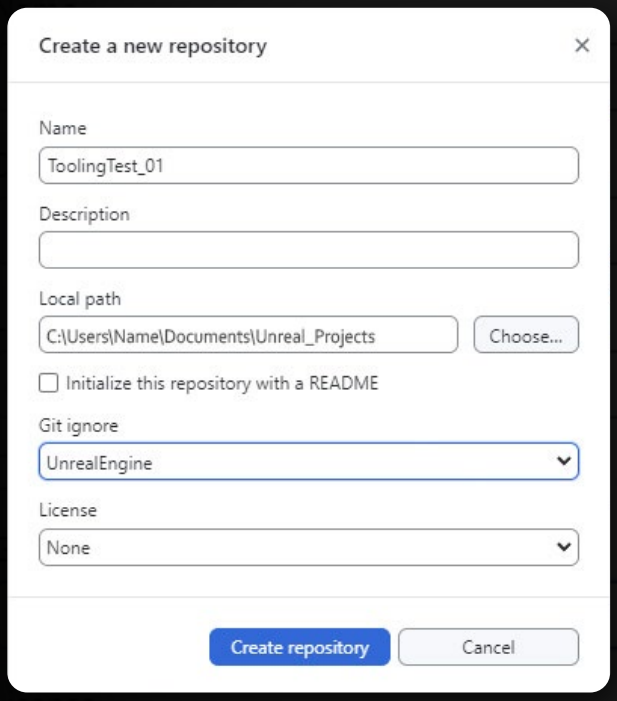
You can make a new repository from the website or from the desktop client. The following will be from the GitHub desktop, and will assume you have an existing or blank Unreal Engine project on your local computer.

Click on *File* > *New Repository*

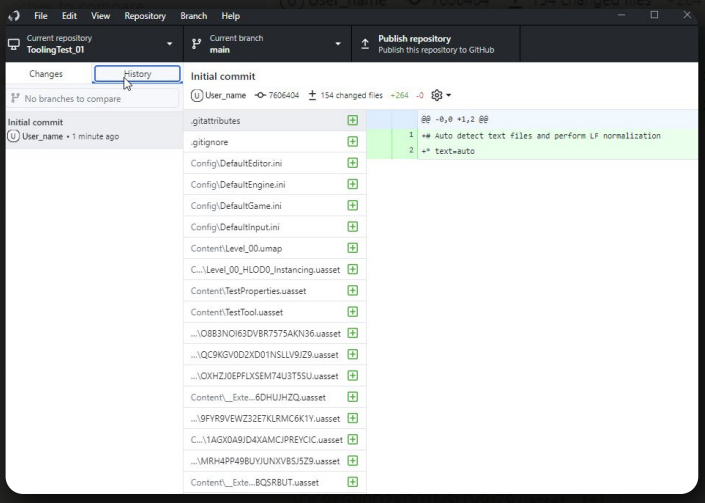


Complete the dialog. For an existing project, put the local path to the directory that contains your project, not the project itself. Then, name the repository the exact same name as the project.

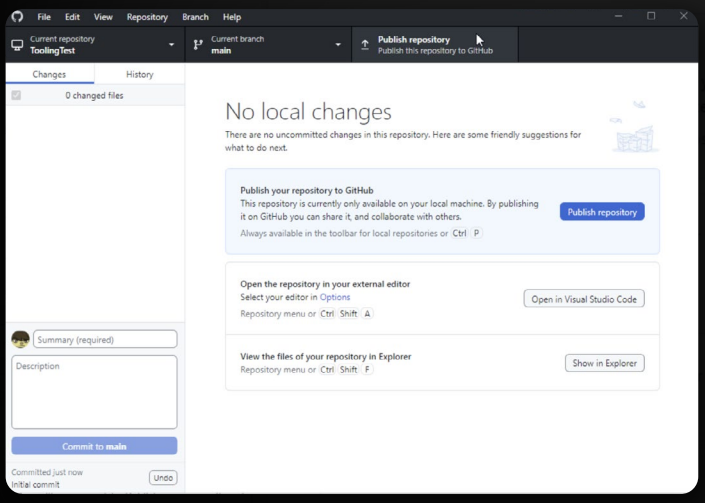
Make sure to choose "UnrealEngine" for your Git ignore.



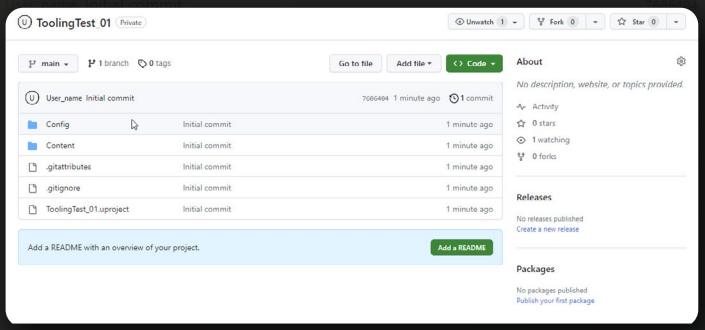
This creates a new local repository. You can double-check the contents are correct by clicking on the History tab.



You will now need to Publish your repository to GitHub.



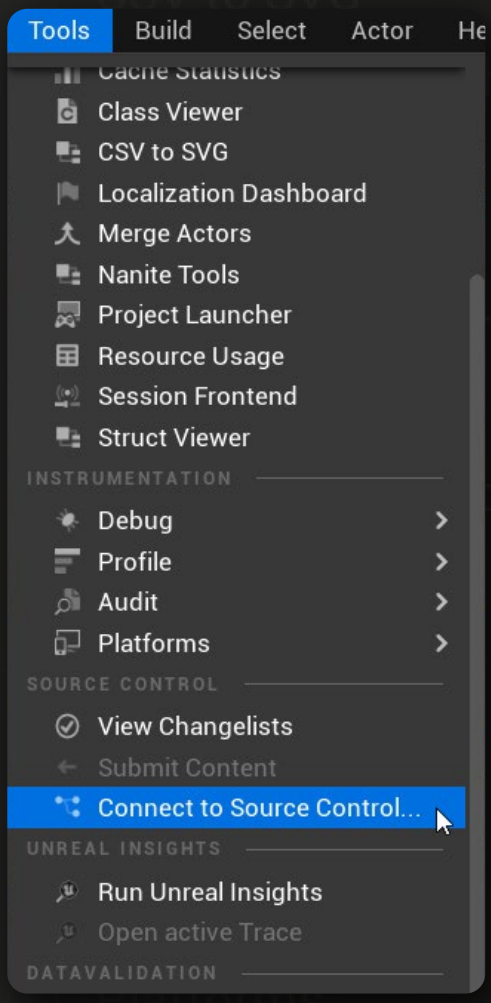
Your project is now in GitHub and ready to collaborate.



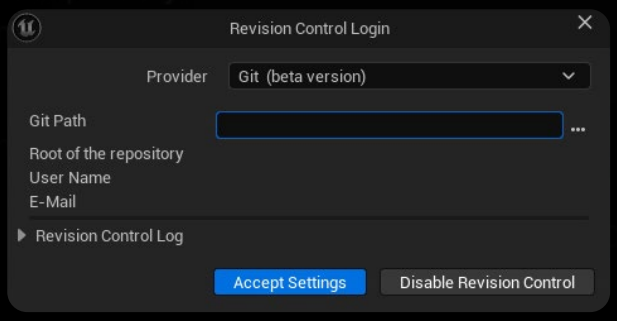
Using Git with Unreal Engine

Much like Perforce, you can also connect Git to Unreal Engine. To do so, follow these steps:

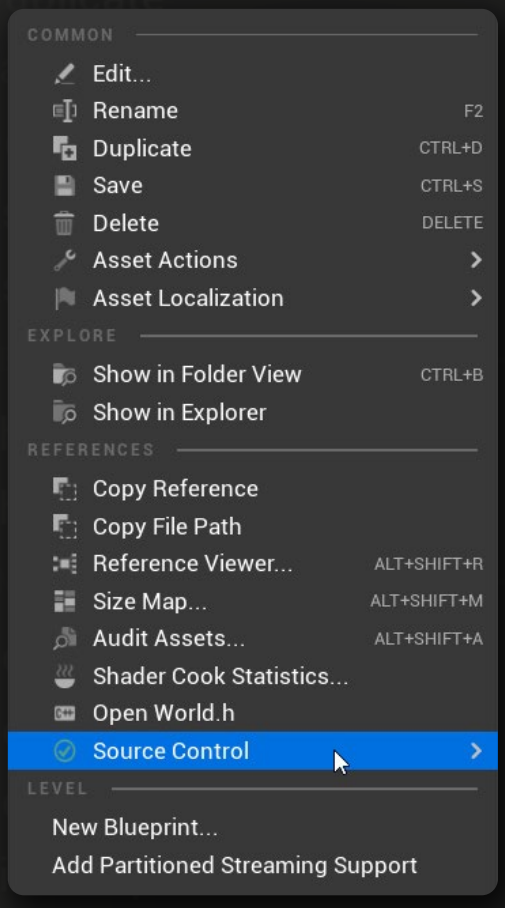
Select
`Tools > Connect to Version Control`



Fill in the dialog with your server information:



Once connected, the Content Browser has a contextual menu for version control:



Resources

About collaborative development models

Installing Git on a server

About GitHub Organizations

Global Campus (Student) Program

GitHub Campus (Institution) Program

Appendix

Appendix A: Version control pipelines in depth

Centralized version control

In a centralized VCS, all of the project resources are located on a central server. When working on files, a user is expected to sync their project with the server, work on the most recent version of the file, create a changelist, and submit those changes to the server. The user only ever has the most recent version of whatever project they are working with.

Working offline with a Centralized VCS is impossible as you need a connection to the server to make new versions. However this means they take less space overall because the versions are only kept in a single place, and with Perforce because only the differences in files are saved and not entire copies of the file, they can be very space efficient.

Distributed version control

In a distributed VCS, every user has a copy of the project and it's versions. There's still a central server that holds a copy of the versions for users to sync to but one of the biggest differences to centralized is the ability to create change lists on your local machine. Much of the workflow is the same as centralized, users will still want to make sure they're on the same page as everyone else with the latest versions of what they're working on, but they may need to be more mindful of it.

Because they have the option of offline work, they may not know about the most recent changes made, and with Git, as with many distributed systems, file locking isn't an option meaning overwriting changes other people made is a much easier thing to do. Users will want to make sure they communicate their intentions earlier to make sure nothing gets corrupted. However because the versions are local it means that making change lists are faster and local branching is a possibility. The downside of that is that the space it can take up is far larger and will often save entire copies of the files in question.

Appendix B:
Best of both worlds?

Recognizing the popularity of the Git workflow for coders, Perforce has developed an extra tool called **Helix4Git.** This tool extends an existing Perforce server and adds on a built-in Git layer. This enables coders to use the Git workflow but all of the code they create is still available in the main Perforce depot for any of the team to access.

This tool is automatically included in the Enhanced Studio Pack available on the AWS and Azure platforms.