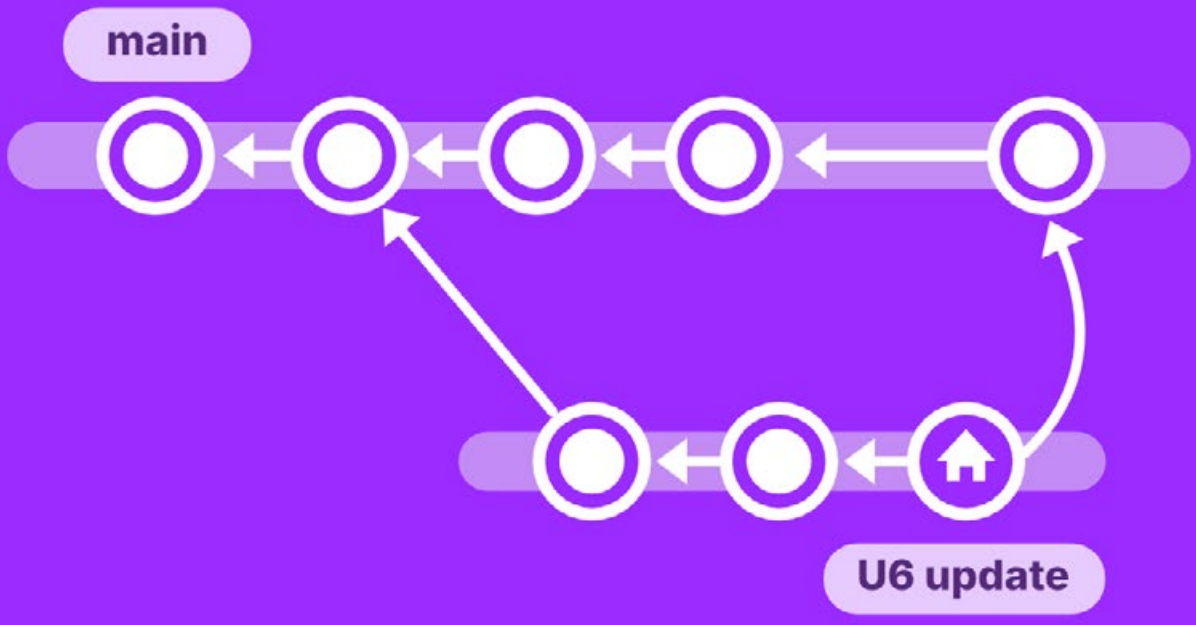
→ E-BOOK

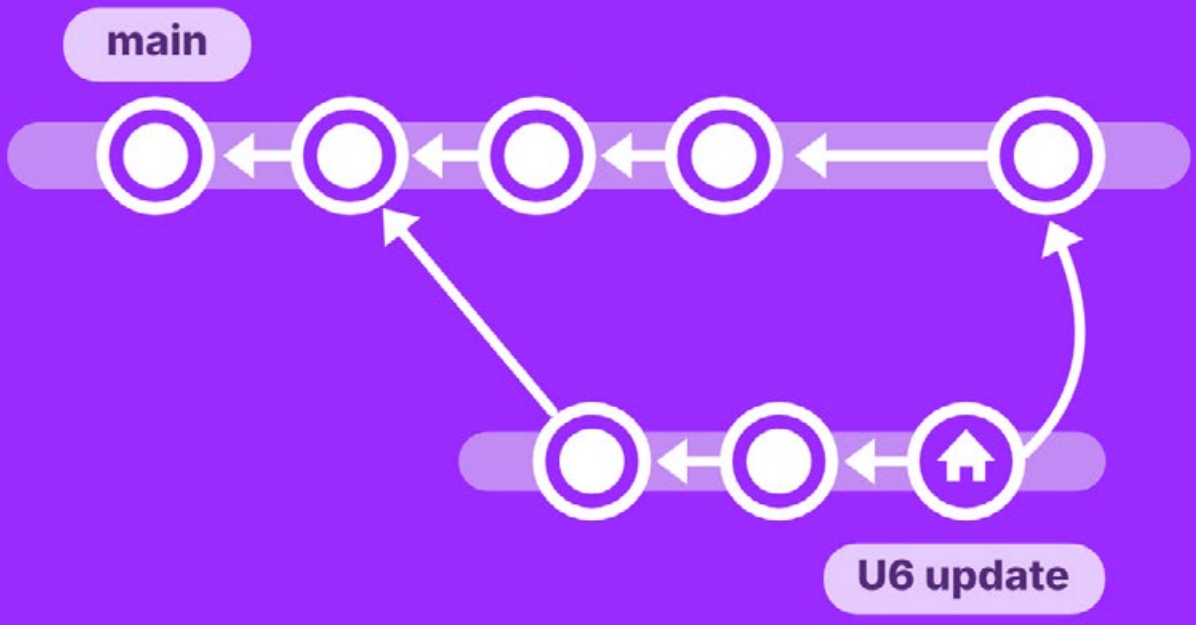
Best practices for project organization and version control



Unity®

→ E-BOOK

项目组织的最佳实践和版本控制



Unity®

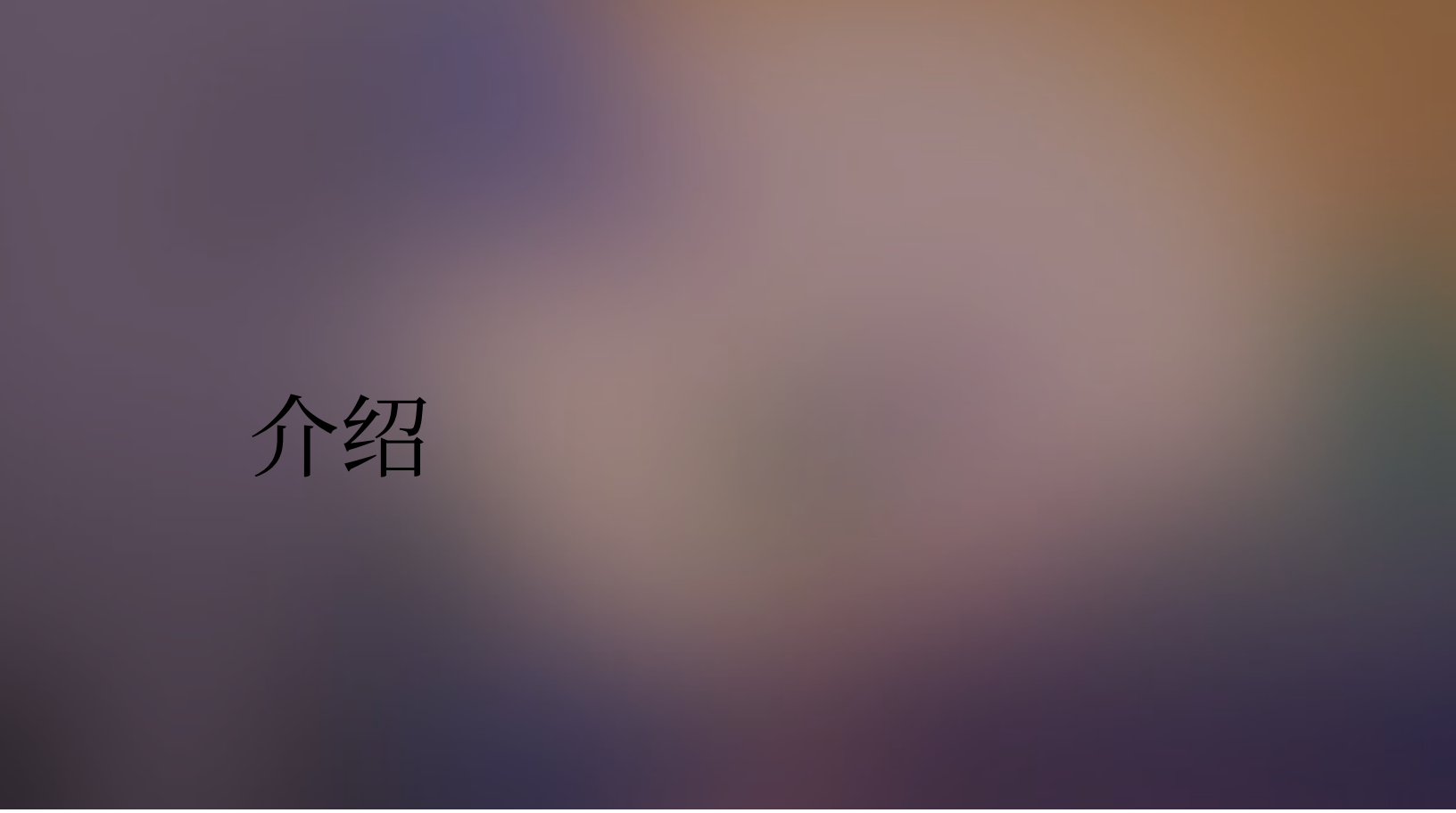
Contents

Introduction

Software development becomes a different beast when you move from working on your own to with a team. Where do you store the project so that every team member has access to it? What happens if more than one person works on the same file at the same time? Programmers often understand the concepts behind source control, but what about artists and other non-technical team members? How can you minimize the amount of support they need from programmers, so they don't have to worry about doing something wrong?

Source control, or version control, can be a daunting topic for game developers, especially if you're not from a technical background. But it doesn't need to be that way. There are a number of tools that integrate with Unity to help your team work effectively with versioning.

This guide explains the key concepts of version control, compares some of the different version control systems (VCS) available, and provides an introduction to additional Unity DevOps tools like Unity Asset Manager, engagement and analytics, game economy, multiplayer services, and more. It provides tips and tricks you can use when setting up your Unity project to help ensure team collaboration is smooth and efficient. Finally, you'll pick up some version control best practices for working successfully in a team.

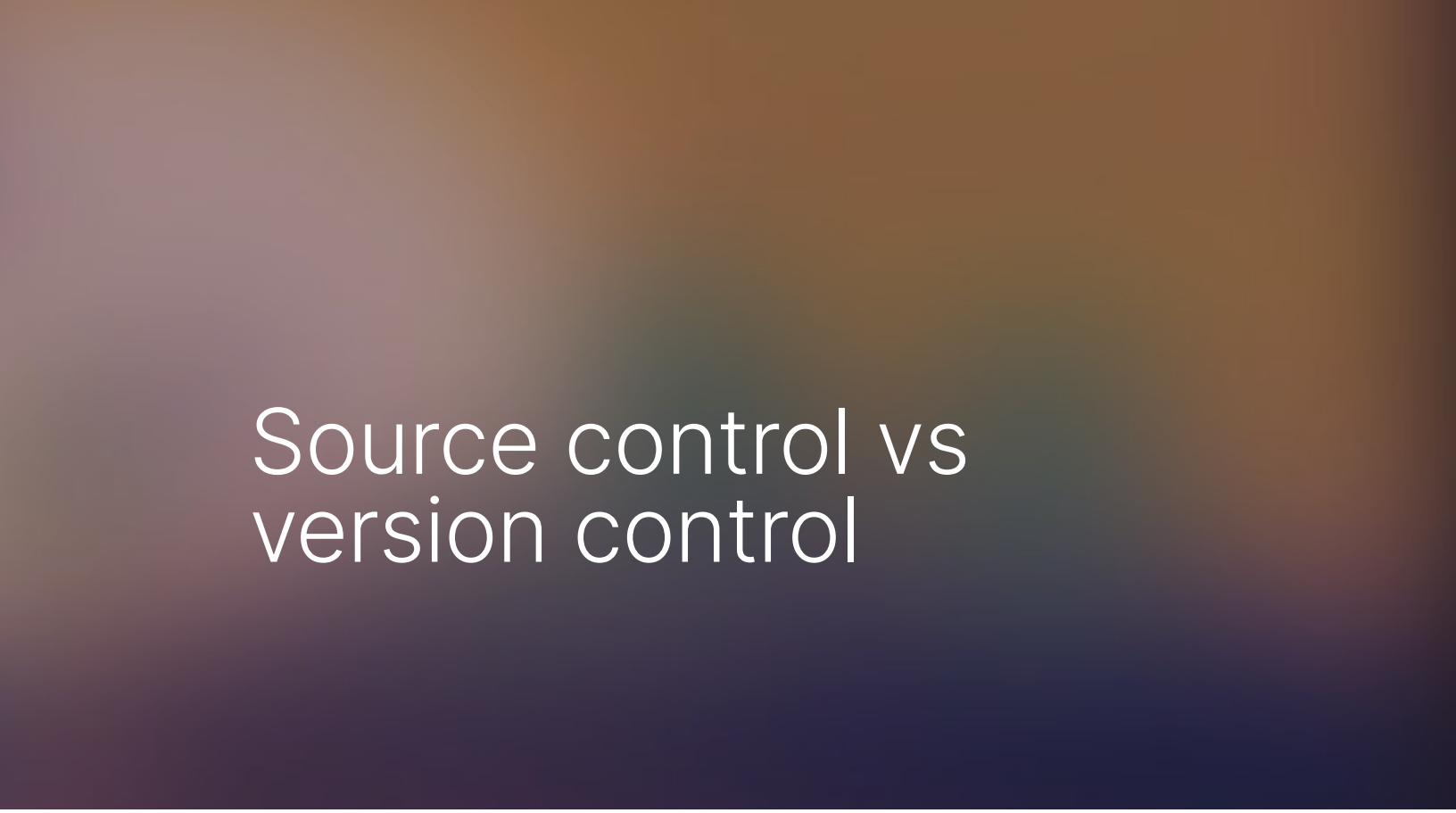


介绍

当您从独立工作转变为与团队合作时，软件开发就会变得与众不同。您将项目存储在哪里，以便每个团队成员都可以访问它？ 如果多人同时处理同一个文件，会发生什么情况？程序员通常了解源代码控制背后的概念，但美术师和其他非技术团队成员呢？您如何最大限度地减少他们需要程序员提供的支持量，以便他们不必担心做错事？

对于游戏开发人员来说，源代码控制或版本控制可能是一个令人生畏的话题，尤其是当您没有技术背景时。但事情并不一定非得是这样的。有许多工具可与 Unity 集成，帮助您的团队有效地进行版本控制。

本指南解释了版本控制的关键概念，比较了一些可用的不同版本控制系统 （VCS），并介绍了其他 Unity DevOps 工具，如 Unity Asset Manager、参与和分析、游戏经济、多人游戏服务等。它提供了在设置 Unity 项目时可以使用的提示和技巧，以帮助确保团队协作顺畅高效。最后，您将掌握一些在团队中成功工作的版本控制最佳实践。



Source control vs version control

In the beginning of computing, all software development was pure code. Even as 3D graphics evolved, everything was still described as code. As such, the term “source control” was used to describe the systems in place to manage the project’s contents, while the term source code management, or SCM, was given as a label for those tools.

Moving into the modern era of software and game development, we now work with a lot more than just the source code. 3D model formats, such as FBX, textures, materials, audio files, and more, mean that SCMs now have to handle more than just text file changes. The term “source control” no longer covers what we need, and thus “version control system” or VCS, became a more apt description and is now the common label given to the tools used.

The terms can still be used interchangeably. However, when talking about Unity projects that often deal with large binary assets, version control and VCS are most accurate, so that’s how they’ll be referred to throughout the rest of the guide.

Three of the main version control systems that work best with Unity are Unity Version Control (UVCS), (formerly known as Plastic SCM),[1] Git, and Perforce Helix Core. This guide presents the benefits and shortcomings of these systems when working as a team on a Unity project.

[1] Plastic SCM joined the Unity family in 2020, which means that these tools are closely integrated into the Unity Editor.

源代码控制与版本控制

在计算的开始，所有的软件开发都是纯代码。即使 3D 图形不断发展，一切都仍然被描述为代码。因此，术语“源代码控制”被用来描述用于管理项目内容的系统，而术语源代码管理 （SCM） 则被作为这些工具的标签。

进入软件和游戏开发的现代时代，我们现在处理的不仅仅是源代码。3D 模型格式（如 FBX、纹理、材质、音频文件等）意味着 SCM 现在必须处理的不仅仅是文本文件更改。术语 “源代码控制” 不再涵盖我们需要的内容，因此 “版本控制系统” 或 VCS 成为一个更贴切的描述，现在是所用工具的通用标签。

这两个术语仍然可以互换使用。但是，当谈论经常处理大型二进制资源的 Unity 项目时，版本控制和 VCS 是最准确的，因此在本指南的其余部分将这样引用它们。

最适合 Unity 的三个主要版本控制系统是 Unity 版本控制 （UVCS）（以前称为 Plastic SCM）、[1] Git 和 Perforce Helix Core。本指南介绍了在团队协作处理 Unity 项目时这些系统的优点和缺点。

[1] Plastic SCM joined the Unity family in 2020, which means that these tools are closely integrated into the Unity Editor.

Foundational concepts

This section covers some of the core concepts of version control. If you don’t know your “commit” from your “push”, this section will help you understand version control terminology.

How version control works

Version control allows you to keep a historical record of your entire project. It brings organization to your work and enables teams to iterate efficiently.

Project files are stored in a shared database called a repository, or “repo.” You backup your project at regular intervals to the repo, and if something goes wrong, you can revert back to an earlier version of the project.

With a VCS, you can make multiple individual changes and commit them as a single group for versioning. This commit sits as a point on the timeline of your project, so that if you need to revert back to a previous version, everything from that commit is undone and restored to the state it was at the time. You can review and modify each change grouped within a commit or undo the commit entirely.

With access to the project’s entire history, it’s easier to identify which changes introduced bugs, restore previously removed features, and easily document changes between your game or product releases.

What’s more, because version control is typically stored in the cloud or on a distributed server, it supports your development team’s collaboration from wherever they’re working – an increasingly important benefit as remote work becomes commonplace.

基本概念

本节介绍版本控制的一些核心概念。如果你不知道你的 “commit” 和 “push” 是什么，本节将帮助你理解版本控制术语。

版本控制的工作原理

版本控制允许您保留整个项目的历史记录。它为您的工作带来组织性，并使团队能够高效迭代。

项目文件存储在称为存储库或“存储库”的共享数据库中。您可以定期将项目备份到存储库，如果出现问题，您可以恢复到项目的早期版本。

使用 VCS，您可以进行多个单独的更改，并将它们作为单个组提交以进行版本控制。此提交位于项目时间轴上的一个点，因此，如果您需要恢复到以前的版本，该提交中的所有内容都会被撤消并恢复到当时的状态。您可以查看和修改提交中分组的每项更改，也可以完全撤消提交。

通过访问项目的整个历史记录，可以更轻松地识别哪些更改引入了错误，恢复之前删除的功能，并轻松记录游戏或产品版本之间的更改。

此外，由于版本控制通常存储在云中或分布式服务器上，因此它支持开发团队在任何地方进行协作 - 随着远程工作变得司空见惯，这是一个越来越重要的好处。

Why use version control?

Aside from the reasons mentioned above, version control is useful for making experimental changes. You can add a new feature in your local version of the project, and if things don't work out, you just revert your changes to go back to working on a clean, functional version of the project.

You can iterate on experimental ideas, and if you need to help out on a major issue in the main project, version control allows you to save your changes for a later date. Then you can get your local version back to the main branch to help out with whatever needs to be worked on. Once you're done, you can restore and carry on with the experimental work.

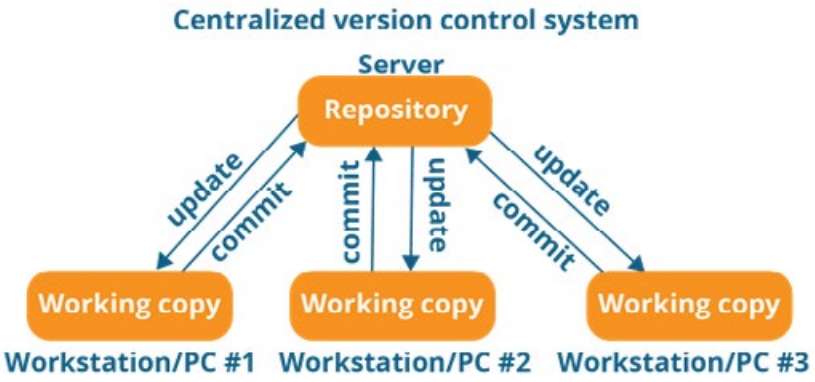
Most version control systems prevent you from accidentally overwriting work that someone else in your team has done. As you commit your work to the repository, you will also need to "pull" the latest updates from the repository. This allows you to check that someone else hasn't been working on the same file as you. This is also known as the dreaded "merge conflict," one of the things that can be scary to people who are not used to version control. However, merge conflicts can usually be resolved easily once you understand the tools. Unity Version Control (UVCS) uses smart locks which helps to reduce the risk of merge conflicts by checking to ensure that locked files on any branch are the latest revision of that file.

Centralized vs distributed version control

For the most part, version control systems fall into one of two categories: centralized or distributed. Depending on which kind of version control system you work with, some of the terms outlined below will apply, some won't, and some may even have a different meaning. Let's take a look at the differences between these two categories.

Centralized

The first key difference between centralized and distributed systems is where the repo resides. Many companies choose the centralized option to keep the servers hosting their proprietary software on-site. Source control security is often an important factor in choosing this kind of system. A centralized system doesn't have to mean on-site servers since the repo can still be hosted in the cloud, but this setup is less common than in distributed systems.



为什么要使用版本控制?

除了上述原因之外，版本控制对于进行实验性更改也很有用。您可以在项目的本地版本中添加新功能，如果事情不顺利，您只需恢复您的更改以返回使用干净、功能齐全的项目版本。

你可以迭代实验性的想法，如果你需要帮助解决主项目中的重大问题，版本控制允许你保存更改以备后用。然后，你可以将本地版本返回主分支，以帮助处理任何需要处理的事情。完成后，你可以恢复并继续实验工作。

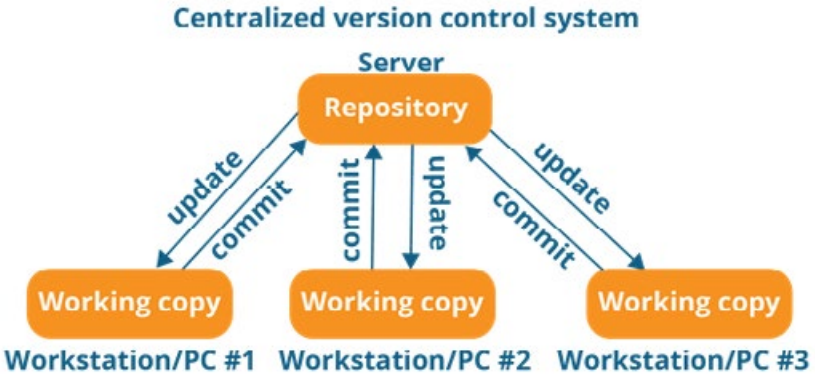
大多数版本控制系统可以防止您意外覆盖团队中其他人已完成的工作。当您将工作提交到存储库时，您还需要从存储库中“拉取”最新更新。这允许您检查其他人是否没有与您处理同一个文件。这也被称为可怕的 “合并冲突”，对于不习惯版本控制的人来说，这可能是可怕的事情之一。 但是，一旦您了解了这些工具，通常可以轻松解决合并冲突。Unity 版本控制 （UVCS） 使用智能锁，通过检查以确保任何分支上的锁定文件是该文件的最新版本，从而帮助降低合并冲突的风险。

集中式与分布式版本控制

在大多数情况下，版本控制系统分为两类之一：集中式或分布式。根据您使用的版本控制系统类型，下面概述的一些术语将适用，一些不适用，有些甚至可能具有不同的含义。 让我们来看看这两个类别之间的区别。

集中

集中式系统和分布式系统之间的第一个关键区别是存储库所在的位置。许多公司选择集中式选项，以将托管其专有软件的服务器保留在现场。源代码控制安全性通常是选择此类系统的重要因素。集中式系统不一定意味着现场服务器，因为存储库仍然可以托管在云中，但这种设置比分布式系统少见。



The other key difference between the two approaches is *how* users deploy their changes to the repo. Centralized version control is often seen as the more straightforward option. When working with a centralized repo, changes are fetched from and sent to the repository directly. This process is called updating from and committing to the repo.

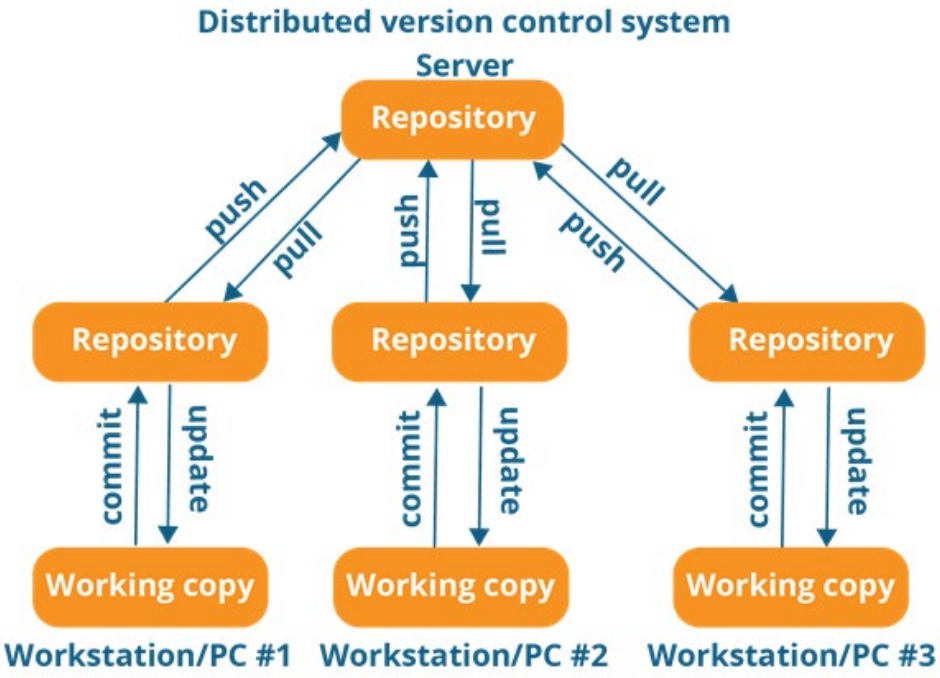
The downside to this setup is that users must be connected to the server to submit any work. To avoid conflicts, users can lock files for modification. This is known as checking out the file, and it prevents anyone else from committing changes until the file is checked back in.

In a centralized workflow, a user only ever has the latest version of the project files on their workstation, and the server holds the project's entire history.

Distributed

In a distributed workflow, there is still a single location where the repo lives, usually on a cloud service such as GitHub, but users clone the entire project history to their workstation. This allows users to work on their own local copy and commit changes quickly since they don't need to be connected to a central server. To send those changes so others can access them later, the user needs to push them to the server and pull any other changes down. However, they don't need to be always working with the latest files like on a centralized system.

Working this way allows you to create a group of changesets that perhaps equate to a larger feature before pushing them up for the rest of your team. In fact, it's encouraged to commit little and often, but we will get to those best practices later on.



这两种方法之间的另一个主要区别是 *how* 用户将其更改部署到存储库。集中式版本控制通常被视为更直接的选项。使用集中式存储库时，将从存储库获取更改并将其直接发送到存储库。 此过程称为 updating from and committing to the repo。

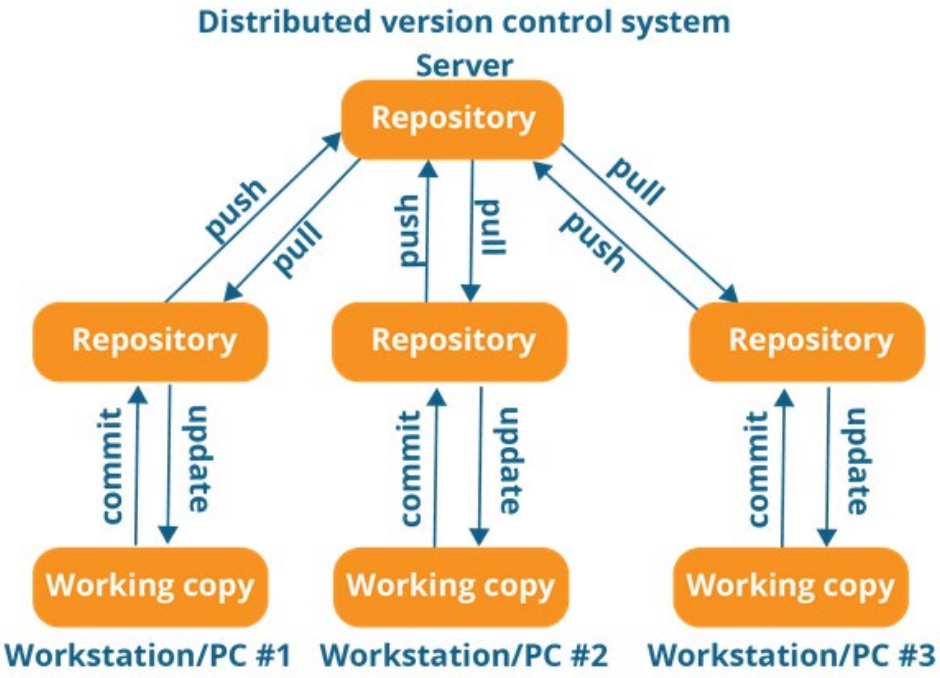
此设置的缺点是用户必须连接到服务器才能提交任何工作。为避免冲突，用户可以锁定文件以进行修改。这称为签出文件，它可以防止其他人提交更改，直到文件被签回。

在集中式工作流程中，用户的工作站上只有最新版本的项目文件，并且服务器保存项目的整个历史记录。

分散式

在分布式工作流中，存储库仍然位于单个位置，通常在 GitHub 等云服务上，但用户将整个项目历史记录克隆到他们的工作站。这允许用户快速处理自己的本地复制和提交更改，因为他们不需要连接到中央服务器。要发送这些更改以便其他人以后可以访问它们，用户需要将它们推送到服务器并拉取任何其他更改。但是，他们不需要像在集中式系统上那样始终使用最新文件。

以这种方式工作允许你创建一组变更集，这些变更集可能等同于一个更大的功能，然后再将它们推给团队的其他成员。事实上，我们鼓励少承诺和经常承诺，但我们稍后会介绍这些最佳实践。



File locking is still available in some distributed workflows, however, it's less common since you can handle merges more easily. By pulling the latest changes from the server to your local project, you can compare anyone else's changes to your own to be sure there are no conflicts before pushing your changes to the repo.

While the distributed approach is often preferred, it also has a few disadvantages. Firstly, having the entire project history on local machines takes up a lot of space, especially for teams working with binary file types. Git has an option called Large File Storage (LFS), which converts the history of certain files to text pointers, offloading some of the weight. However, other files have the entire history, and repos can end up with a load of old or stale test data. Studios working with small M2 drives may then find the size of the repo gets bloated with old versions, overloading their drives.

Secondly, as developers don't have to stay in contact with a central server, they can end up working in isolation for long periods. Their local version can become quite detached from the main repository, and when it comes time to merge their changes back in, this may be more work than they bargained for.

Typical workflow

In brief, here are the steps in a workflow for a centralized system and a distributed one:

Centralized

1. Update your working copy with changes from the server.
2. Make your changes.
3. Commit your changes to the central server.

Distributed

1. Pull any remote changes into your local repo.
2. Make your changes.
3. Commit your changes.
4. Perform steps 2 and 3 as many times as you like.
5. Push all commits back to the remote repo.

This guide focuses on three main version control systems: UVCS, Git, and Perforce Helix Core. A detailed walkthrough of each VCS starts in the section "Version control systems", but here's a brief introduction to each one and the workflow it supports:

— Unity Version Control or UVCS (both): UVCS is a flexible version control system with unique interfaces to support programmers and artists alike. It excels at handling large repos and binary files, and as both a file-based and changeset-based solution, it gives you the capability to download only the specific files you're working on, rather than the entire project build.

— Git (distributed): This is one of the most popular version control systems around. Git is open source, free, and flexible. As a platform Git is a command line-only tool. But many different GUIs have been developed for it, making the system more accessible to users.

文件锁定在某些分布式工作流程中仍然可用，但是，它不太常见，因为您可以更轻松地处理合并。通过将最新更改从服务器拉取到本地项目，您可以将其他人的更改与您自己的更改进行比较，以确保在将更改推送到存储库之前没有冲突。

虽然分布式方法通常是首选，但它也有一些缺点。首先，在本地计算机上拥有整个项目历史记录会占用大量空间，尤其是对于处理二进制文件类型的团队。Git 有一个称为大文件存储 （LFS） 的选项，它可以将某些文件的历史记录转换为文本指针，从而减轻一些权重。但是，其他文件具有完整的历史记录，并且存储库最终可能会加载旧的或过时的测试数据。 使用小型 M2 驱动器的工作室可能会发现存储库的大小会因旧版本而膨胀，从而使驱动器过载。

其次，由于开发人员不必与中央服务器保持联系，因此他们最终可以长时间孤立工作。他们的本地版本可能会与主存储库完全分离，当需要将他们的更改合并回来时，这可能比他们预期的要多。

典型工作流程

简而言之，以下是集中式系统和分布式系统的工作流程中的步骤：

集中	分散式
1. 使用服务器中的更改更新您的工作副本。2. 进行更改.3. 将更改提交到中央服务器.	1. 将任何远程更改拉取到本地存储库中。2. 进行更改. 3. 提交更改。4. 根据需要多次执行步骤 2 和 3. 5. 将所有提交推送回远程存储库

本指南重点介绍三个主要的版本控制系统：UVCS、Git 和 Perforce Helix Core。 每个 VCS 的详细演练从 “版本控制系统” 部分开始，但以下是每个 VCS 及其支持的工作流程的简要介绍：

— Unity 版本控制或 UVCS（两者兼而有之）： UVCS 是一个灵活的版本控制系统，具有独特的界面，可为程序员和艺术家提供支持。它擅长处理大型存储库和二进制文件，并且作为基于文件和基于变更集的解决方案，它使您能够仅下载您正在处理的特定文件，而不是整个项目构建。

— Git （分布式）： 这是最流行的版本控制系统之一。Git 是开源、免费且灵活的。作为一个平台，Git 是一个仅限命令行的工具。但是已经为它开发了许多不同的 GUI，使用户更容易访问该系统。

There can often be some confusion between Git and GitHub. GitHub is a hosting service for Git repositories, but you can use Git without using GitHub. That said, as any experienced developer reading this will know, GitHub is a very popular service because there is a free version (with some limitations), and it doesn't require any custom server setups.

— Perforce Helix Core (centralized): This is an enterprise-level version control system generally used by large game studios because it features centralized repos that are most often hosted on their own servers.

Key terms

Here are the terms for some of the key features and processes in a VCS:

Term	Explanation
Repository/repo	This is the database of all the changes and edits to your project. Stored on a server, either on-site or in the cloud, it holds the full history of the project.
Working copy	This is your local version of the project. Sometimes also called a checkout or workspace. You make changes to your working copy, and, when you're happy with them, commit them to the repository.
Commit/check in	A commit encodes file modifications. A centralized workflow sends those changes to the server and is more commonly called checking in. In a distributed workflow, it adds them to the changeset that needs to later be pushed to the server.
Pull/update/check out	Pulling or updating retrieves the latest changes available on the server. Check out is the more common term when working in a centralized workflow.
Locking	Locking a file prevents it from being edited by another user. You are telling the server, "I'm working on this; please don't make any other changes." Locking is generally not supported in distributed workflows.
Clone	In a distributed workflow, cloning a repo is how you initially get a copy of the project and its entire history onto your local machine.
Tags	Tags are special notes that can be added to a commit. They are often used to mark a point in time where a build was made.
Branch	A branch creates a new copy of the codeline, which can then be worked on in parallel. This allows someone to work on parts of the project in isolation, for example a new feature, without affecting the main line of development.

Git 和 GitHub 之间经常会有一些混淆。GitHub 是 Git 存储库的托管服务，但您可以在不使用 GitHub 的情况下使用 Git。也就是说，正如任何阅读本文的经验丰富的开发人员都知道的那样，GitHub 是一项非常受欢迎的服务，因为有一个免费版本（有一些限制），并且它不需要任何自定义服务器设置。

— Perforce Helix Core（集中式）：这是大型游戏工作室通常使用的企业级版本控制系统，因为它具有集中式存储库，这些存储库通常托管在他们自己的服务器上。

关键术语

以下是 VCS 中一些关键功能和流程的术语：

Term	Explanation
Repository/repo	This is the database of all the changes and edits to your project. Stored on a server, either on-site or in the cloud, it holds the full history of the project.
Working copy	This is your local version of the project. Sometimes also called a checkout or workspace. You make changes to your working copy, and, when you're happy with them, commit them to the repository.
Commit/check in	A commit encodes file modifications. A centralized workflow sends those changes to the server and is more commonly called checking in. In a distributed workflow, it adds them to the changeset that needs to later be pushed to the server.
Pull/update/check out	Pulling or updating retrieves the latest changes available on the server. Check out is the more common term when working in a centralized workflow.
Locking	Locking a file prevents it from being edited by another user. You are telling the server, "I'm working on this; please don't make any other changes." Locking is generally not supported in distributed workflows.
Clone	In a distributed workflow, cloning a repo is how you initially get a copy of the project and its entire history onto your local machine.
Tags	Tags are special notes that can be added to a commit. They are often used to mark a point in time where a build was made.
Branch	A branch creates a new copy of the codeline, which can then be worked on in parallel. This allows someone to work on parts of the project in isolation, for example a new feature, without affecting the main line of development.

Merge	Merging can happen either when a branch is finished and needs to be merged back into the main line, or even just when two people make changes around the same time. The two changesets will need to be compared and merged together to create the new working copy. Most merges can be handled automatically.
Conflict	A conflict is what happens when merges cannot be handled automatically. This usually occurs when two people have made changes to the same lines of code or the same binary file. Code conflicts can usually be resolved by comparing the text and working out which changes should be accepted, or even whether both can be brought together in a way. For binary files, such as Unity scenes or Prefabs, merging a conflict becomes a lot trickier. However, sometimes a quick conversation with the other contributor is the easiest way to resolve what changes make the most sense to keep.
Pull request	When work on a branch is complete, it's good practice to open a pull request. This signals to the rest of your team that work on that branch is complete and ready to be merged back into the main line. This system gives team leads and/or seniors a chance to review the changes before accepting them back into the main branch.
Head	Head refers to the latest commit on your working copy.
Reset/revert	Depending on your VCS, reset or revert can be used to discard all your local changes back to their state at the head.
Index	The Git index is a file that describes all the current changes you have in your working copy. These changes sit in what's known as the staging area, where you can select which changes you want to add to your next commit.
Git stash	If you have some changes that aren't ready yet for a commit, but you need to move onto some different work, you can use a stash to save those changes in a temporary file and reset your working copy back to head.

Merge	Merging can happen either when a branch is finished and needs to be merged back into the main line, or even just when two people make changes around the same time. The two changesets will need to be compared and merged together to create the new working copy. Most merges can be handled automatically.
Conflict	A conflict is what happens when merges cannot be handled automatically. This usually occurs when two people have made changes to the same lines of code or the same binary file. Code conflicts can usually be resolved by comparing the text and working out which changes should be accepted, or even whether both can be brought together in a way. For binary files, such as Unity scenes or Prefabs, merging a conflict becomes a lot trickier. However, sometimes a quick conversation with the other contributor is the easiest way to resolve what changes make the most sense to keep.
Pull request	When work on a branch is complete, it's good practice to open a pull request. This signals to the rest of your team that work on that branch is complete and ready to be merged back into the main line. This system gives team leads and/or seniors a chance to review the changes before accepting them back into the main branch.
Head	Head refers to the latest commit on your working copy.
Reset/revert	Depending on your VCS, reset or revert can be used to discard all your local changes back to their state at the head.
Index	The Git index is a file that describes all the current changes you have in your working copy. These changes sit in what's known as the staging area, where you can select which changes you want to add to your next commit.
Git stash	If you have some changes that aren't ready yet for a commit, but you need to move onto some different work, you can use a stash to save those changes in a temporary file and reset your working copy back to head.

Best practices for organizing a Unity project

Regardless of which VCS you choose, there are some generally recommended practices that will help streamline your version control workflow when working in Unity. First, let's take a look at some of the different ways your team can work together effectively.

Project organization

Folder structure

Although there is no single way to organize your project, in general, follow these recommendations.

— Define and document your naming conventions and folder structure as a team. A style guide and/or project template makes your files easier to find and organize. Pick what works for your team, and make sure everyone is on board with it.

— Be consistent with your naming convention. Don't deviate from your chosen style guide or template. If you do need to amend your naming rules, parse and rename your affected assets all at once. In cases where the changes affect a large number of files, consider automating the update using a script.

— Don't use spaces in file and folder names. Use CamelCase as an alternative for spaces.

— Separate testing or sandbox areas. Create a separate folder for non-production scenes and experimentation. Subfolders with usernames can divide your work area by team member.

— Avoid extra folders at the root level. In general, store your content files within the Assets folder. Don't create additional folders at the project's root level unless it's absolutely necessary.

组织 Unity 项目的最佳实践

无论您选择哪种 VCS，在 Unity 中工作时，都有一些通常推荐的做法有助于简化版本控制工作流程。首先，让我们看一下您的团队可以有效协作的一些不同方式。

项目组织

文件夹结构

尽管没有单一的方法来组织您的项目，但通常请遵循这些建议。

— 以团队的形式定义并记录您的命名约定和文件夹结构。样式指南和/或项目模板使您的文件更易于查找和组织。选择适合您团队的方法，并确保每个人都参与其中。 — 与您的命名约定保持一致。不要偏离您选择的风格指南或模板。如果您确实需要修改命名规则，请一次性解析和重命名受影响的资源。如果更改影响大量文件，请考虑使用脚本自动执行更新。 — 不要在文件和文件夹名称中使用空格。使用 CamelCase 作为空格的替代项。 — 单独的测试区域或沙盒区域。为非生产场景和实验创建一个单独的文件夹。带有用户名的子文件夹可以按团队成员划分您的工作区域。 — 避免在根级别使用额外的文件夹。通常，将内容文件存储在 Assets 文件夹中。不要在项目的根级别创建其他文件夹，除非它是 abs

— Keep your internal assets separate from third-party ones. If you are using assets from the Asset Store or other plug-ins, odds are they have their own project structure. Keep your assets separate.

If you find yourself modifying a third-party asset or plug-in for your project, then version control can really help you out when you need to get the latest update for the plug-in. Once the update is imported, you can look through the diff to find where your modifications may have been overwritten and reimplement them.

While there is no set folder structure, here are a couple of examples of how you might set up your Unity project. These structures are based on splitting up your project by asset type. The Asset Types manual page describes the most common assets in greater detail. You can use the Template or Learn projects as an example of organizing your folder structure. While you're not limited to these folder names, they should give you a good starting point.

Example 1

```
Assets
+---Art
| +---Materials
| +---Models
| +---Textures
+---Audio
| +---Music
| \---Sound
+---Code
| +---Scripts  # C# scripts
| \---Shaders  # Shader files and shader graphs
+---Docs   # Wiki, concept art, marketing material
+---Level   # Anything related to game design in Unity
| +---Prefabs
| +---Scenes
| \---UI
```

— 将您的内部资产与第三方资产分开。如果您使用的是 Asset Store 或其他插件中的资源，则它们很可能具有自己的项目结构。将您的资产分开。

如果您发现自己正在修改项目的第三方资源或插件，那么当您需要获取插件的最新更新时，版本控制可以真正为您提供帮助。导入更新后，您可以浏览差异以查找您的修改可能被覆盖的位置并重新实现它们。

虽然没有设置文件夹结构，但以下是如何设置 Unity 项目的几个示例。这些结构基于按资产类型拆分项目。资产类型手册页更详细地介绍了最常见的资产。您可以使用 Template 或 Learn 项目作为组织文件夹结构的示例。虽然您不仅限于这些文件夹名称，但它们应该为您提供一个很好的起点。

Example 1

```
Assets
+---Art
| +---Materials
| +---Models
| +---Textures
+---Audio
| +---Music
| \---Sound
+---Code
| +---Scripts  # C# scripts
| \---Shaders  # Shader files and shader graphs
+---Docs   # Wiki, concept art, marketing material
+---Level   # Anything related to game design in Unity
| +---Prefabs
| +---Scenes
| \---UI
```

Example 2

```
Assets
+---Art
| +---Materials
| +---Models
| +---Music
| +---Prefabs
| +---Sound
| +---Textures
| +---UI
+---Levels
+---Src
| +---Framework
| \---Shaders
```

If you download one of the template or starter projects from the Unity Hub, you'll find that those projects have their subfolders split up based on asset type, as seen in the image below.

Templates available to download in the Unity Hub

Depending on which template you've chosen, you should see subfolders that represent several common assets. Here is one way to organize by type:

Animations	Animations contain animated motion clips and their controller files. These can also contain Timeline assets for in-game cinematics or rigging information for procedural animation.
Audio	Sound assets include audio clips as well as the mixers used for blending the effects and music.
Editor	Here you'll find the scripted tools made for use with the Unity Editor but not appearing in a target build.
Fonts	This folder contains the fonts used in the game.
Materials	These assets describe surface shading properties.

Example 2

```
Assets
+---Art
| +---Materials
| +---Models
| +---Music
| +---Prefabs
| +---Sound
| +---Textures
| +---UI
+---Levels
+---Src
| +---Framework
| \---Shaders
```

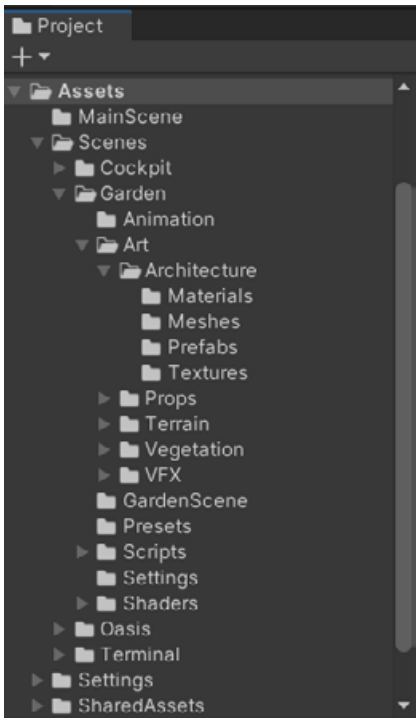
如果从 Unity Hub 下载其中一个模板或初学者项目，您会发现这些项目的子文件夹根据资源类型进行了拆分，如下图所示。

可在 Unity Hub 中下载的模板

根据您选择的模板，您应该会看到代表几个常见资源的子文件夹。以下是按类型组织的一种方法：

Animations	Animations contain animated motion clips and their controller files. These can also contain Timeline assets for in-game cinematics or rigging information for procedural animation.
Audio	Sound assets include audio clips as well as the mixers used for blending the effects and music.
Editor	Here you'll find the scripted tools made for use with the Unity Editor but not appearing in a target build.
Fonts	This folder contains the fonts used in the game.
Materials	These assets describe surface shading properties.

Meshes	Store models created in an external digital content creation (DCC) application here.
Particles	The particle simulations in Unity, created either with the Built-In Particle System or Visual Effect Graph.
Prefabs	These are reusable GameObjects with prebuilt Components.
Scripts	All user-developed code for gameplay appears here.
Scenes	Unity stores small, functional portions of your project into Scene assets. They often correspond to game levels or part of a level.
Settings	This can be used for storing render pipeline settings, such as for the High Definition Render Pipeline (HDRP) and Universal Render Pipeline (URP).
Shaders	These programs run on the GPU as part of the graphics pipeline.
Textures	Image files can consist of texture files for materials and surfacing, UI overlay elements for user interface, and lightmaps to store lighting information.
ThirdParty	If you have assets from an external source like the Asset Store, keep them separated from the rest of your project here. This makes updating your third-party assets and scripts easier. Third-party assets may have a set structure that cannot be altered.
UI	If you're using UI Toolkit, your UXML and USS files are stored here.

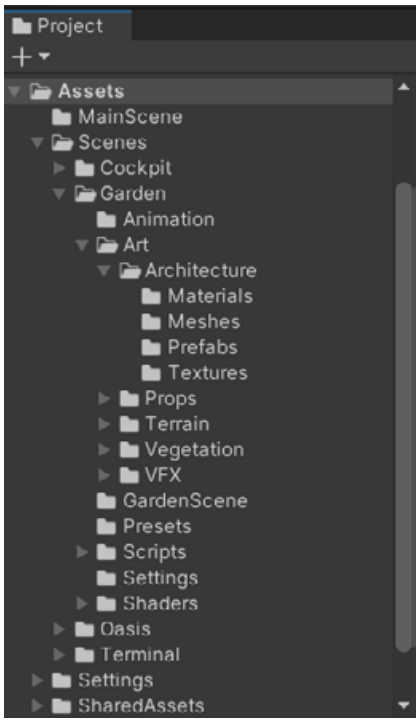


The sample scene with the URP template includes a number of asset folders.

Defining a good project structure in the beginning will avoid version control issues later. If you move assets from one folder to another, many VCS will see that as just deleting one file and adding another, rather than the file being moved. This loses the history of the original file.

UVCS can handle file moves within Unity and maintains the history of any file that's moved. However, it's essential that when you move a file, you do it in the Unity Editor so that the .meta file moves with the asset file.

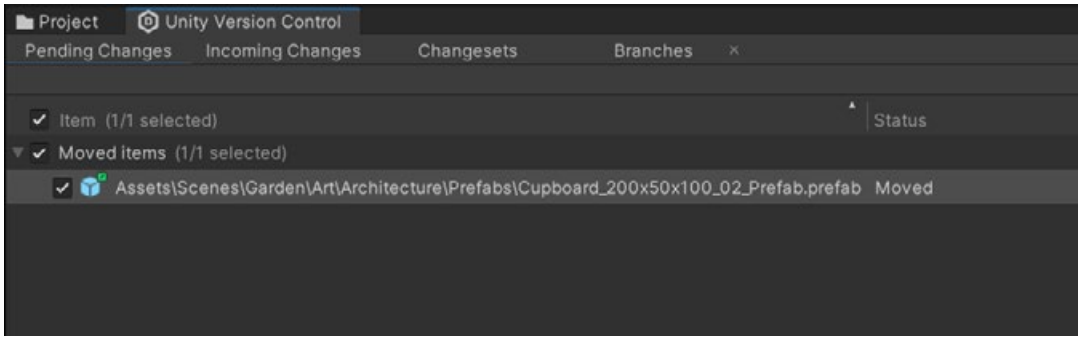
Meshes	Store models created in an external digital content creation (DCC) application here.
Particles	The particle simulations in Unity, created either with the Built-In Particle System or Visual Effect Graph.
Prefabs	These are reusable GameObjects with prebuilt Components.
Scripts	All user-developed code for gameplay appears here.
Scenes	Unity stores small, functional portions of your project into Scene assets. They often correspond to game levels or part of a level.
Settings	This can be used for storing render pipeline settings, such as for the High Definition Render Pipeline (HDRP) and Universal Render Pipeline (URP).
Shaders	These programs run on the GPU as part of the graphics pipeline.
Textures	Image files can consist of texture files for materials and surfacing, UI overlay elements for user interface, and lightmaps to store lighting information.
ThirdParty	If you have assets from an external source like the Asset Store, keep them separated from the rest of your project here. This makes updating your third-party assets and scripts easier. Third-party assets may have a set structure that cannot be altered.
UI	If you're using UI Toolkit, your UXML and USS files are stored here.



带有 URP 模板的示例场景包括许多资源文件夹。

在开始时定义一个好的项目结构将避免以后的版本控制问题。如果您将资产从一个文件夹移动到另一个文件夹，许多 VCS 会将其视为只是删除一个文件并添加另一个文件，而不是正在移动的文件。这将丢失原始文件的历史记录。

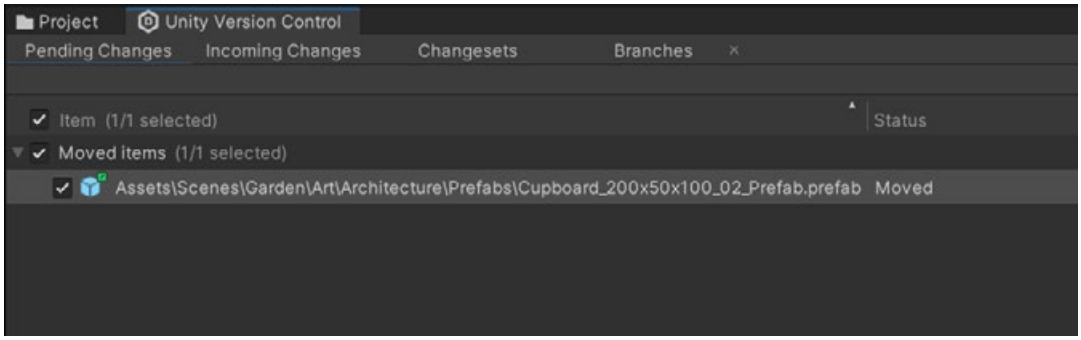
UVCS 可以在 Unity 中处理文件移动，并维护任何移动文件的历史记录。 但是，在移动文件时，必须在 Unity Editor 中执行此作，以便 .meta 文件与资源文件一起移动。



Tracking file movements

Once you've decided on a folder structure for your projects, use an Editor script to reuse the template and create the same folder structure for all projects moving forward. When it's placed in an Editor folder, the script below will create a root folder in Assets matching the "PROJECT_NAME" variable. Doing this keeps your own work separate from third-party packages.

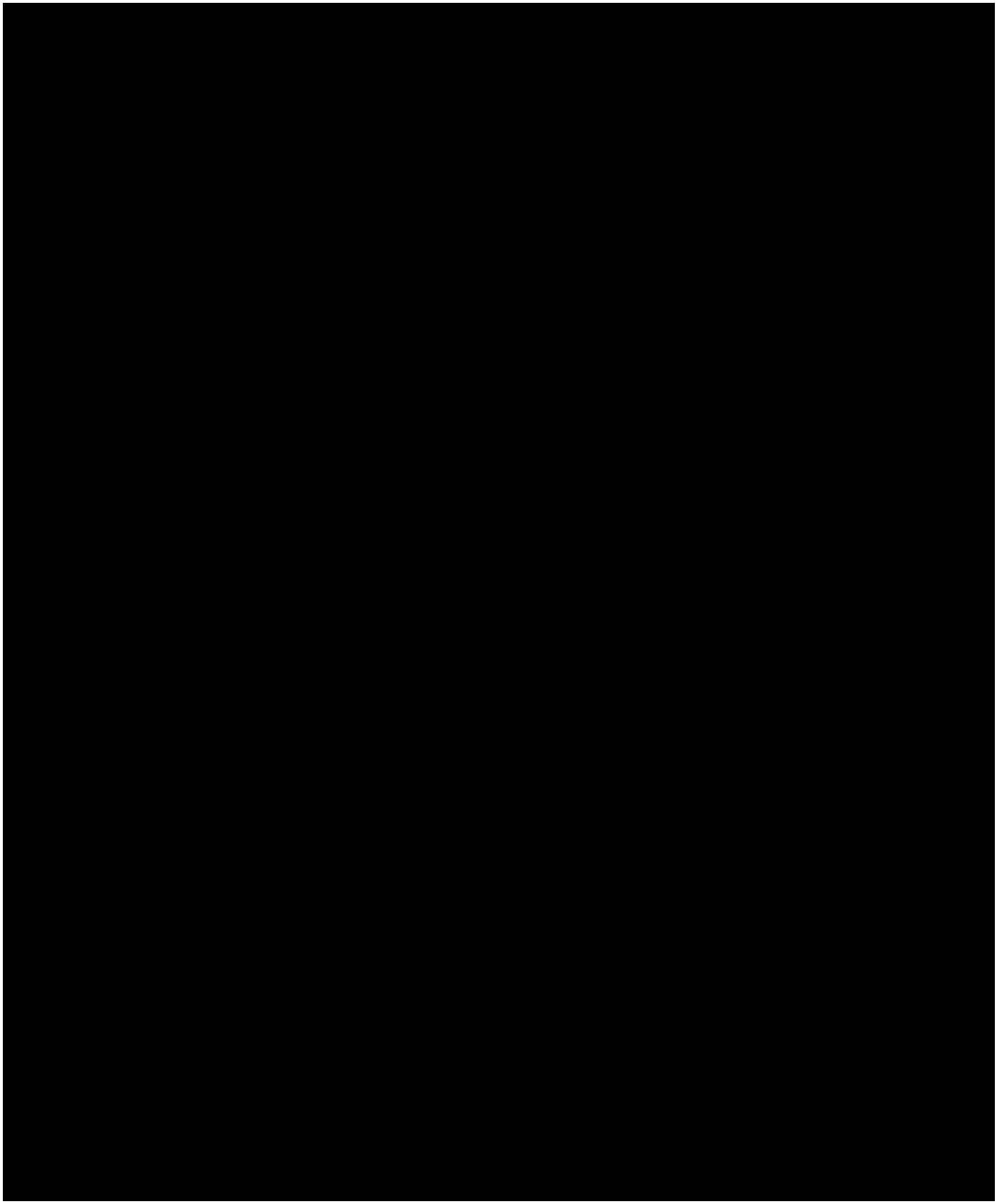
```
using UnityEditor;
using UnityEngine;
using System.Collections.Generic;
using System.IO;

public class CreateFolders : EditorWindow {

 private static string projectName = "PROJECT_NAME";
  [MenuItem("Assets/Create Default Folders")]
 private static void SetUpFolders()
  {
  CreateFolders window =
ScriptableObject.CreateInstance<CreateFolders>();
  window.position = new Rect(Screen.width/2, Screen.height/2, 400, 150);
  window.ShowPopup();
 }
 private static void CreateAllFolders()
 {
  List<string> folders = new List<string>
  {
   "Animations",
   "Audio",
   "Editor",
   "Materials",
   "Meshes",
   "Prefabs",
   "Scripts",
```

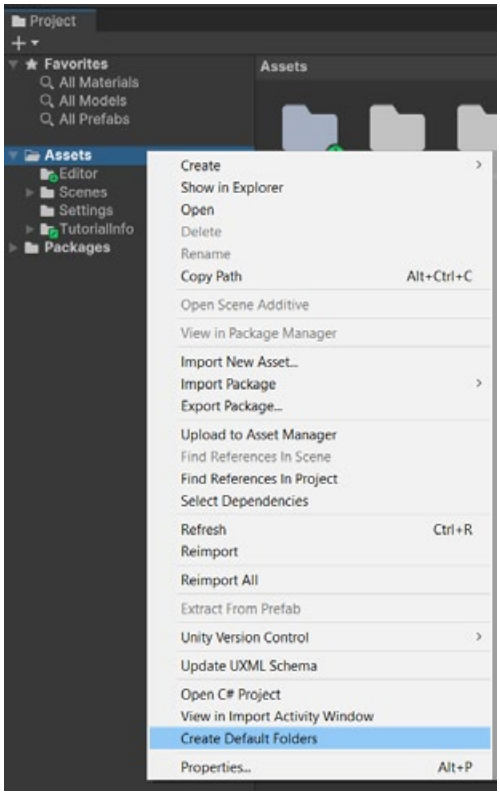


跟踪文件移动

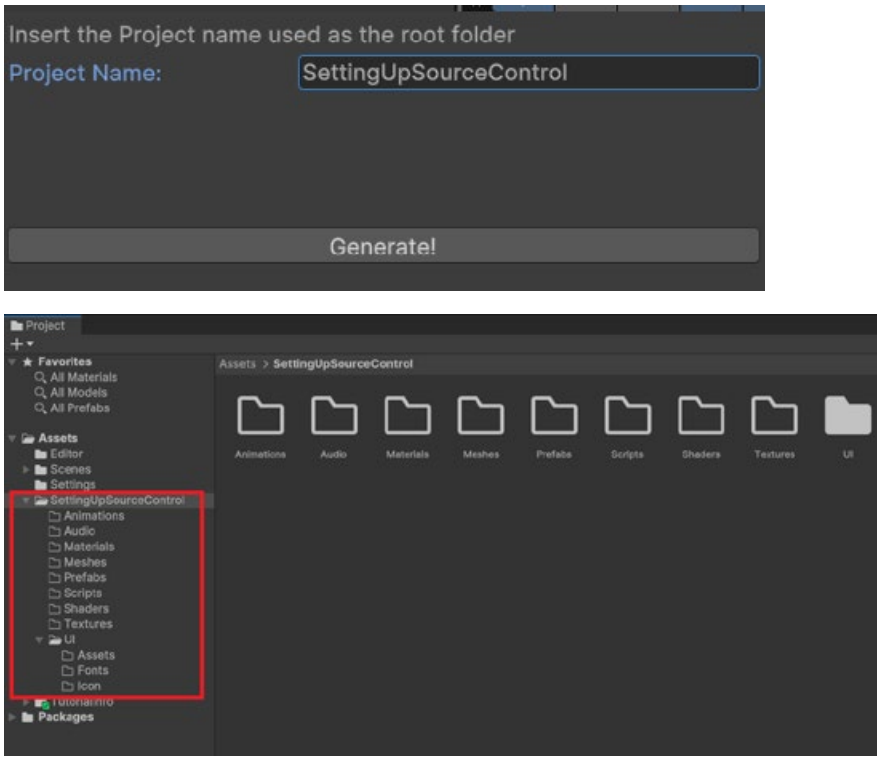
确定项目的文件夹结构后，请使用 Editor 脚本重复使用模板，并为以后的所有项目创建相同的文件夹结构。当它被放置在 Editor 文件夹中时，下面的脚本将在 Assets 中创建一个与 “PROJECT_ NAME” 变量匹配的根文件夹。这样做可以将您自己的工作与第三方软件包分开。

```
    “Scenes”,
    “Shaders”,
    “Textures”,
    “UI”
   };

   foreach (string folder in folders)
   {
    if (!Directory.Exists(“Assets/” + folder))
    {
     Directory.CreateDirectory(“Assets/” + projectName + “/” + folder);
    }
   }

   List<string> uiFolders = new List<string>
   {
    “Assets”,
    “Fonts”,
    “Icon”
   };

   foreach (string subfolder in uiFolders)
   {
    if (!Directory.Exists(“Assets/” + projectName + “/UI/” + subfolder))
    {
     Directory.CreateDirectory(“Assets/” + projectName + “/UI/” + subfolder);
    }
   }

   AssetDatabase.Refresh();
  }

  void OnGUI()
  {
   EditorGUILayout.LabelField(“Insert the Project name used as the root folder”);
   projectName = EditorGUILayout.TextField(“Project Name: “,projectName);
   this.Repaint();
   GUILayout.Space(70);
   if (GUILayout.Button(“Generate!”)) {
    CreateAllFolders();
    this.Close();
   }
  }
}
```





Go to menu > Assets > Create Default Folders.



Creating empty folders at the start of your project will help keep your teamwork organized and efficient.

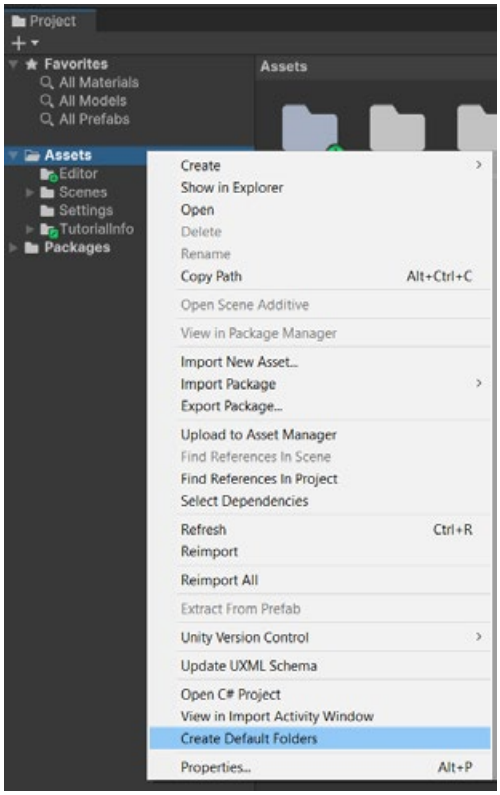
Empty folders

Empty folders like those shown in the previous images can present a bit of an issue in version control – so only create the folders for what you need. With Git and Perforce, empty folders are ignored by default. If these project folders are set up and someone attempts to commit them, they'll be unable to until something is placed in the folder.

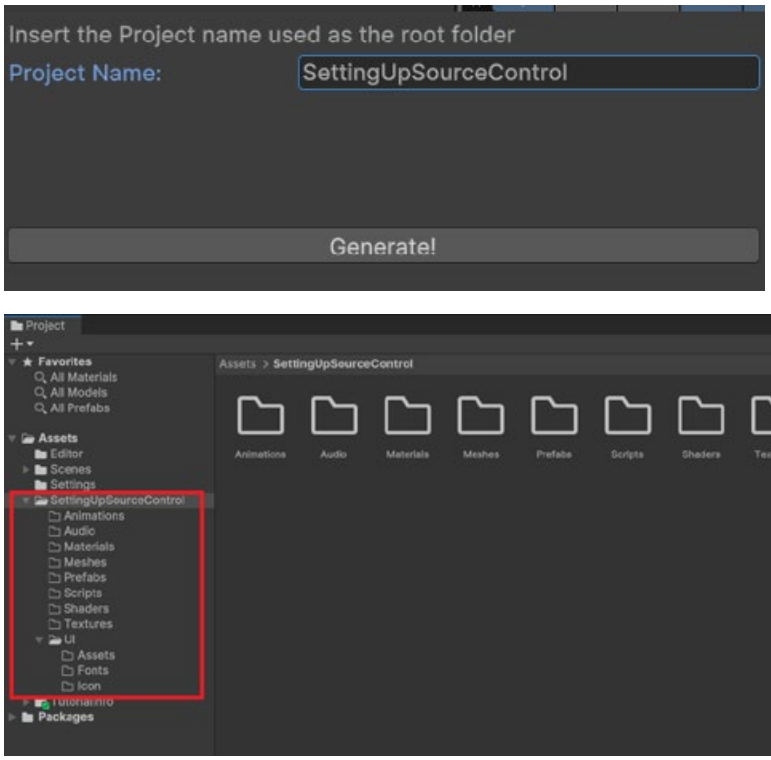
Note: A common workaround is to place a ".keep" file inside an empty folder. This is enough for the folder to then be committed to the repository.

UVCS can handle empty folders. Directories are treated as entities and have a version history associated with them.

This is a point to note when working in Unity. Unity generates a .meta file for every file in the project, including folders. With Git and Perforce, a user can easily commit the .meta file for an empty folder, but the folder itself won't end up under version control. When another user gets the latest changes, there will be a .meta file for a folder that doesn't exist on their machine, and Unity will then delete the .meta file. UVCS avoids this issue by including empty folders under version control.



转到菜单 > 资产 > 创建默认文件夹。



在项目开始时创建空文件夹将有助于保持您的团队合作井井有条和高效。

空文件夹

如上图所示的空文件夹可能会在版本控制中带来一些问题 - 因此请仅为您需要的内容创建文件夹。使用 Git 和 Perforce 时，默认情况下会忽略空文件夹。如果这些项目文件夹已设置，并且有人尝试提交它们，则在将某些内容放入该文件夹之前，他们将无法提交。

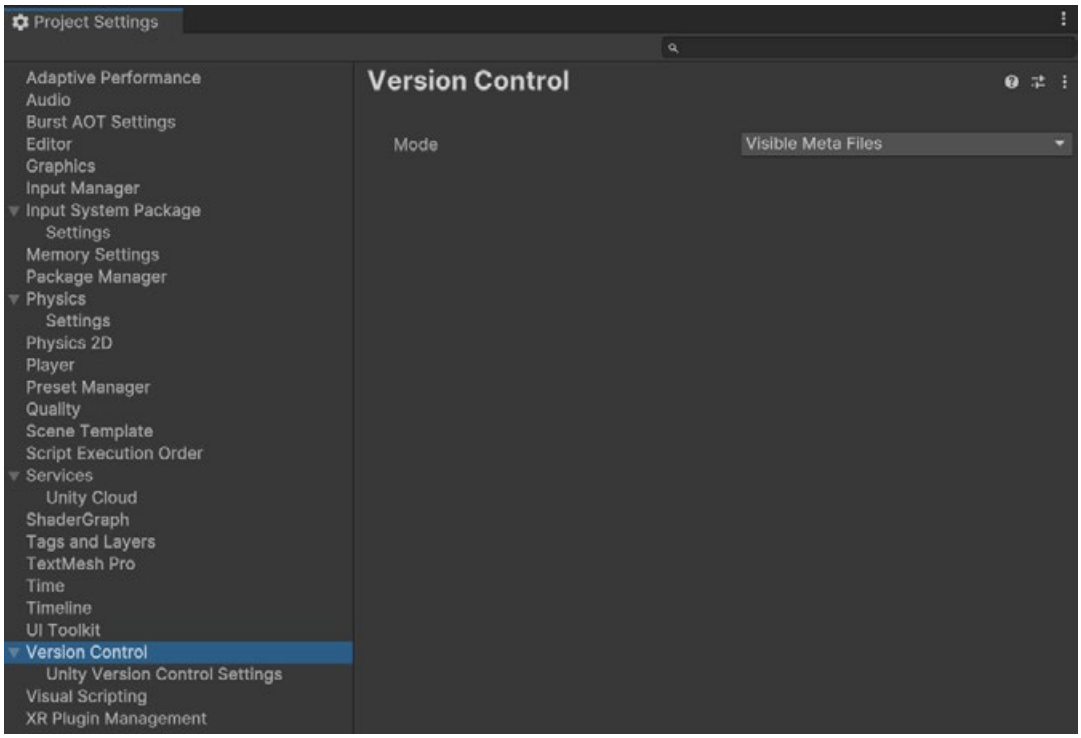
注意：一种常见的解决方法是将 “.keep” 文件放在空文件夹中。这足以将文件夹提交到存储库。

UVCS 可以处理空文件夹。目录被视为实体，并具有与之关联的版本历史记录。

这是在 Unity 中工作时需要注意的一点。Unity 为项目中的每个文件（包括文件夹）生成一个 .meta 文件。使用 Git 和 Perforce，用户可以轻松地为空文件夹提交 .meta 文件，但文件夹本身最终不会受到版本控制。当其他用户获得最新更改时，其计算机上不存在的文件夹将有一个 .meta 文件，然后 Unity 将删除该 .meta 文件。UVCS 通过在版本控制下包含空文件夹来避免此问题.

The .meta file

Unity generates a .meta file for every other file inside the project, and while it's typically inadvisable to include auto-generated files in version control, the .meta file is a little different. Visible Meta Files mode should be turned on in the Version Control window (unless you're using UVCS or Perforce modes).

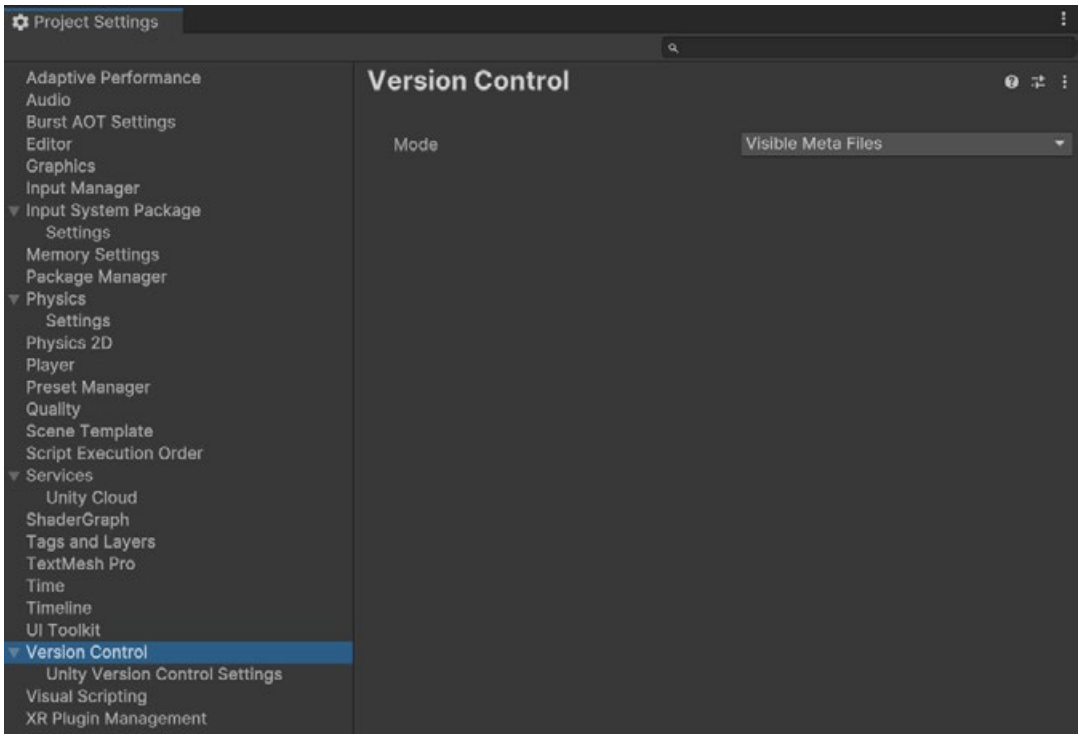


Turn on Visible Meta Files when working with Git.

While the .meta file is auto-generated, it also holds a lot of information about the file with which it's associated. This is common with assets that have import settings, such as Textures, meshes, audio clips, etc. When you change any import settings on these files, the changes are written into the .meta file, not the asset file. This is why you commit the .meta files to your repository, so everyone works with the same file settings.

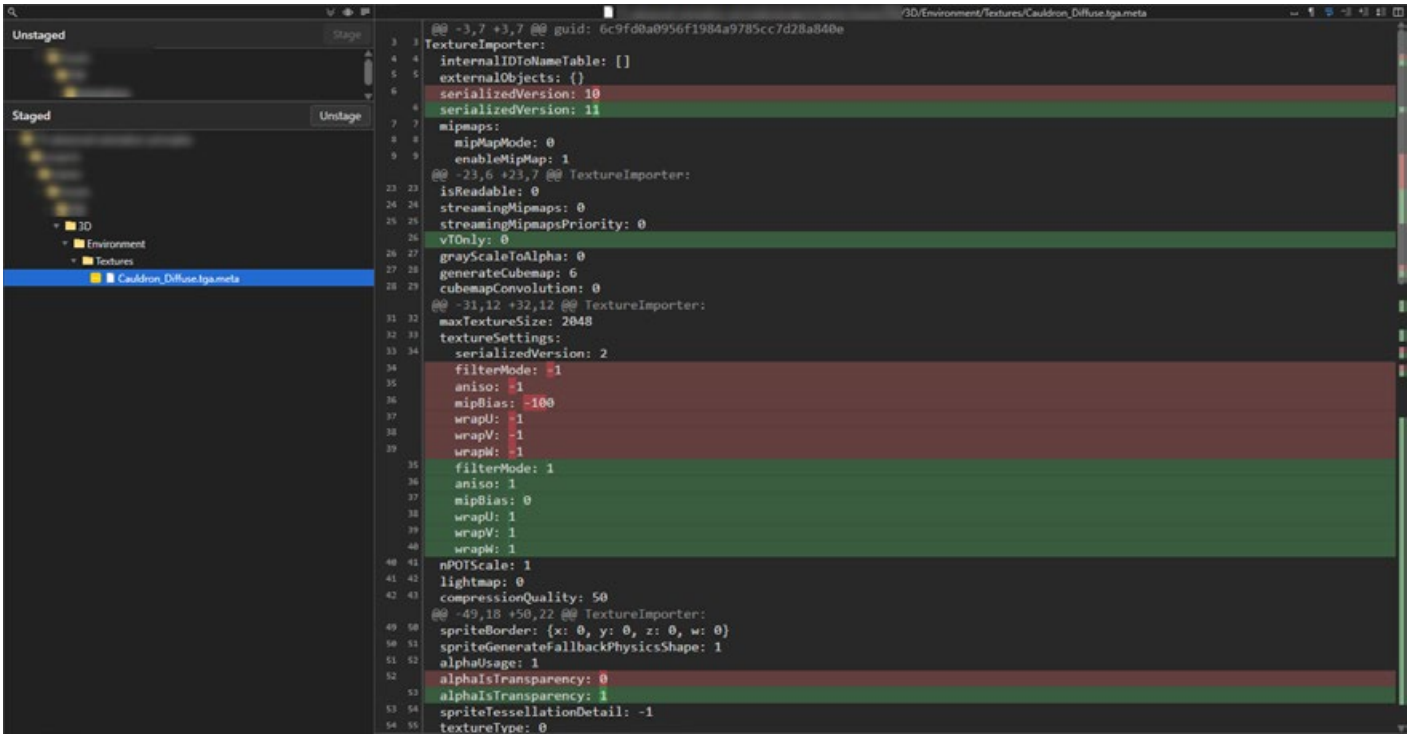
.meta 文件

Unity 会为项目中的每个其他文件生成一个 .meta 文件，虽然通常不建议在版本控制中包含自动生成的文件，但 .meta 文件略有不同。 应在 Version Control （版本控制） 窗口中打开 Visible Meta Files （可见元文件） 模式（除非您使用的是 UVCS 或 Perforce 模式）。



在使用 Git. 时打开 Visible Meta Files

虽然 .meta 文件是自动生成的，但它也包含有关与其关联的文件的大量信息。这在具有导入设置的资源（如纹理、网格、音频剪辑等）中很常见。当您更改这些文件的任何导入设置时，更改将写入 .meta 文件，而不是资源文件。这就是您将 .meta 文件提交到存储库的原因，以便每个人都使用相同的文件设置。



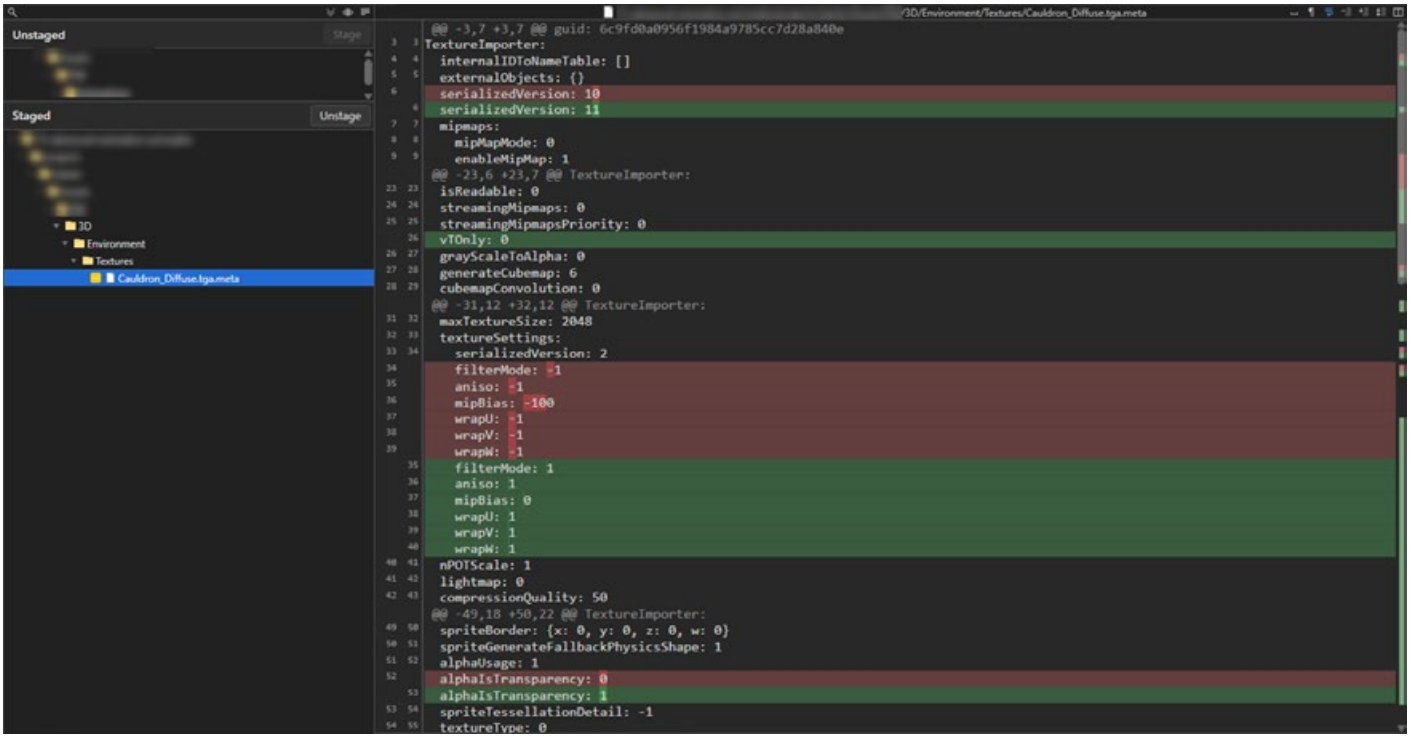
Changes to a .meta file when import settings were adjusted on a file

Naming standards

Agreeing on standards doesn't stop with project folder structure. Setting a naming standard for GameObjects in a scene or prefabs inside project folders can make things easier for your team to understand when you end up working in one another's files.

Though there is no definitive naming standard for GameObjects, consider the following.

Standard	Example
Use descriptive names and don't abbreviate. Use names that you will remember several months from now. Consider whether another person will understand your notation, and choose names that you can pronounce and remember. For this reason abbreviations are generally not recommended.	largeButton, LargeButton, or leftButton NOT: lButton
Use Camel case/Pascal case. Avoid spaces in your object names. Camel case or Pascal case improve readability (and typing accuracy according to this study).	OutOfMemoryException, dateTimeFormat, NOT: Outofmemoryexception, datetimeformat



调整文件的导入设置时对 .meta 文件的更改

命名标准

就标准达成一致并不止于项目文件夹结构。为场景中的游戏对象或工程文件夹中的预制件设置命名标准，可以使您的团队在最终处理彼此的文件时更容易理解。

虽然游戏对象没有明确的命名标准，但请考虑以下事项。

Standard	Example
Use descriptive names and don't abbreviate. Use names that you will remember several months from now. Consider whether another person will understand your notation, and choose names that you can pronounce and remember. For this reason abbreviations are generally not recommended.	largeButton, LargeButton, or leftButton NOT: lButton
Use Camel case/Pascal case. Avoid spaces in your object names. Camel case or Pascal case improve readability (and typing accuracy according to this study).	OutOfMemoryException, dateTimeFormat, NOT: Outofmemoryexception, datetimeformat

Use underscores (or hyphens) sparingly. Avoid underscores and hyphens in general. However, they can be useful in certain circumstances. Prefixing a name with an underscore puts it alphabetically first. You can also use underscores to denote variants of a specific object.	Active States: EnterButton_Active, EnterButton_ Inactive Texture Maps: Foliage_Diffuse, Foliage_Normalmap Level of Detail: Building_LOD1, Building_LOD0
Use number suffixes to denote a sequence. Likewise, don't suffix with a number if it's not part of a list.	For a path, name the nodes: Node0, Node1, Node2, etc.
Follow the design document naming.	If your design document names locations like HighSpellTower or RedDragonLair, use those exact spellings.

Agreeing on a consistent style across your team can result in a cleaner, more readable and scalable project. The Unity e-book, *Create a C# style guide: Write cleaner code that scales*, provides tips and best practices for naming conventions, formatting, classes, methods, comments and more. Overall the guide follows Microsoft C# style standards 2, providing a Unity-specific subset of that, but there is no one "true" method; the Google C# 2 guide is also a great resource for defining guidelines around naming, formatting, and commenting conventions.

Workflow optimization

Aside from how and where you keep your assets inside the Assets folder, there are several design and development choices you can make to help speed up your workflow, especially when you're using version control.

Split up your assets

Large, single Unity scenes do not lend themselves well to collaboration. Break your levels into many smaller scenes so that artists and designers can collaborate better on a single level while minimizing the risk of conflicts.

At runtime, your project can load scenes additively using SceneManager.LoadSceneAsync passing the LoadSceneMode.Additive parameter mode.

Additionally, break work up into Prefabs where possible. If you need to make changes later, you can change the Prefab rather than the scene it's used in to avoid conflicts with anyone working on the scene. Prefab changes can often be easier to read when doing a diff under version control.

Use underscores (or hyphens) sparingly. Avoid underscores and hyphens in general. However, they can be useful in certain circumstances. Prefixing a name with an underscore puts it alphabetically first. You can also use underscores to denote variants of a specific object.	Active States: EnterButton_Active, EnterButton_ Inactive Texture Maps: Foliage_Diffuse, Foliage_Normalmap Level of Detail: Building_LOD1, Building_LOD0
Use number suffixes to denote a sequence. Likewise, don't suffix with a number if it's not part of a list.	For a path, name the nodes: Node0, Node1, Node2, etc.
Follow the design document naming.	If your design document names locations like HighSpellTower or RedDragonLair, use those exact spellings.

在整个团队中就一致的风格达成一致，可以使项目更简洁、更具可读性和可扩展性。Unity 电子书 *Create a C# style guide: Write cleaner code that scales* 提供了有关命名约定、格式、类、方法、注释等的提示和最佳实践。总体而言，该指南遵循 Microsoft C# 样式标准 2，提供了特定于 Unity 的子集，但没有一个“真正的”方法;Google C# 2 指南也是定义有关命名、格式和注释约定的准则的绝佳资源.

工作流程优化

除了在 Assets 文件夹中保存资源的方式和位置之外，您还可以做出多种设计和开发选择来帮助加快工作流程，尤其是在使用版本控制时。

拆分您的资产

大型单个 Unity 场景不适合协作。将关卡分解为许多较小的场景，以便艺术家和设计师可以在单个关卡上更好地协作，同时最大限度地降低冲突风险。

在运行时，您的项目可以使用 SceneManager.LoadSceneAsync 传递 LoadSceneMode.Additive 参数模式以附加方式加载场景。

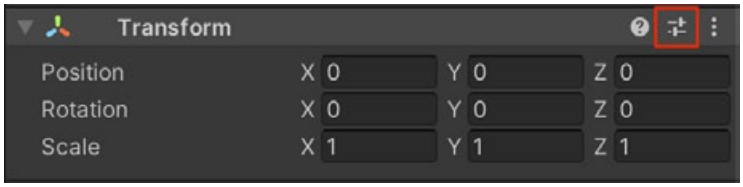
此外，尽可能将工作分解为预制件。如果以后需要进行更改，可以更改预制件而不是使用它的场景，以避免与在场景中工作的任何人发生冲突。在版本控制下执行差异时，预制件更改通常更容易阅读.

And if you end up with a scene conflict, Unity also has a built-in YAML (a human-readable, data-serialization language) tool specifically for merging scenes and Prefabs. For more information, see Smart merge in the Unity documentation.

Presets

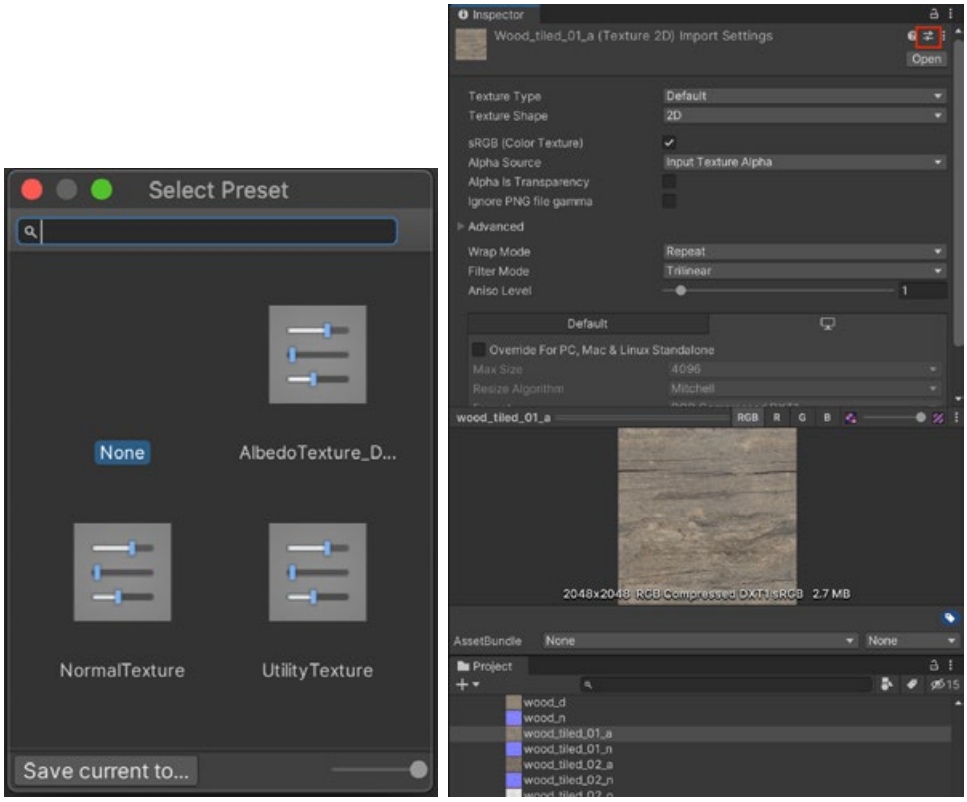
Unity presets are predefined configurations that allow you to save and reuse specific settings across different components, assets, or tools within a project. Creating a Preset lets you copy the settings of a component or asset, save it as an asset, then apply the same settings to another item later.

Use Presets to enforce standards or to apply reasonable defaults to new assets. This ensures consistent standards across your team, so commonly overlooked settings don't impact your project's performance.



The Preset icon is highlighted here in red.

Click the Preset icon to the top right of the component. Click **Save current to...** to save the Preset as an asset. Click one of the available Presets to load a set of values.



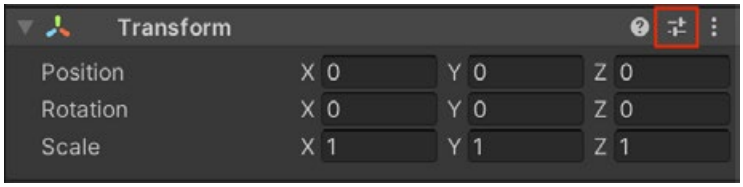
In this example, the Presets contain different Import Settings for 2D textures depending on usage (albedo, normal, or utility).

如果最终出现场景冲突，Unity 还有一个内置的 YAML（一种人类可读的数据序列化语言）工具，专门用于合并场景和预制件。有关更多信息，请参阅 Unity 文档中的智能合并。

预设

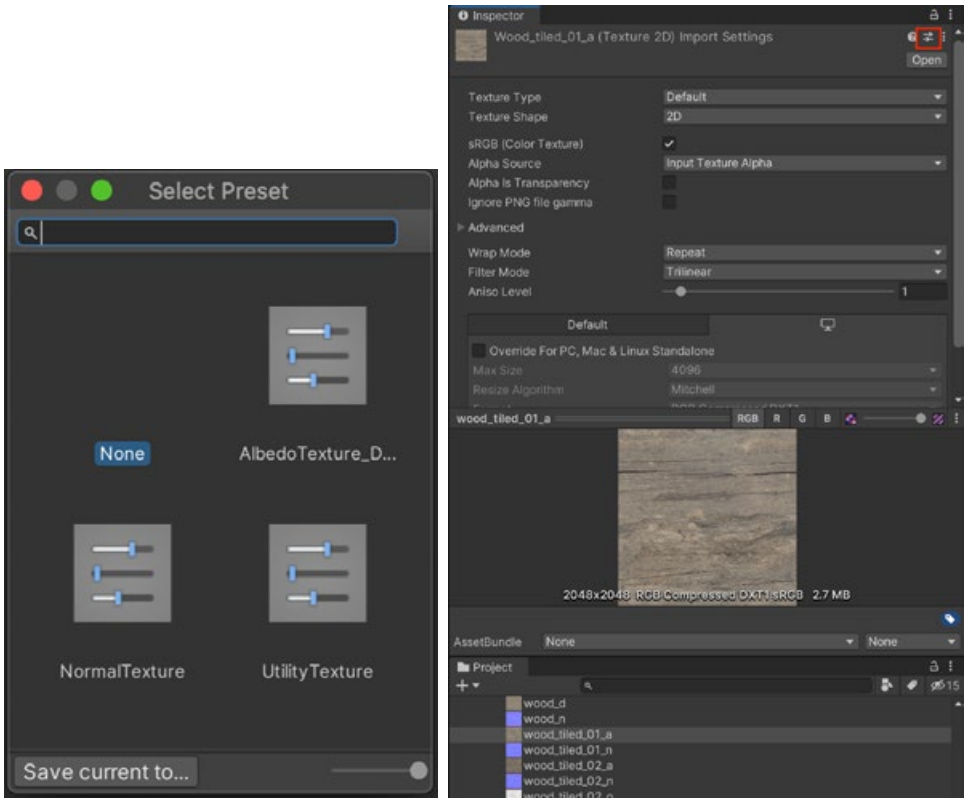
Unity 预设是预定义的配置，允许您在项目中的不同组件、资源或工具中保存和重复使用特定设置。通过创建预设，您可以复制组件或资源的设置，将其保存为资源，然后稍后将相同的设置应用于另一个项目。

使用预设可强制实施标准或将合理的默认值应用于新资源。这可确保整个团队的标准一致，因此通常被忽视的设置不会影响项目的性能。



预设图标在此处以红色. 高亮显示

单击组件右上角的 Preset 图标。单击 Save current to...将预设保存为资产。单击其中一个可用的预设以加载一组值。



在此示例中，预设包含 2D 纹理的不同导入设置，具体取决于用途（反照率、法线或实用程序）。

Other handy ways to use Presets include:

- **Create a GameObject with defaults:** Drag and drop a Preset asset into the Hierarchy to create a new GameObject with the corresponding component that includes Preset values.
- **Associate a specific Type with a Preset:** In the Preset Manager (**Project Settings > Preset Manager**), specify one or more Presets per type. Creating a new component will then default to the specified Preset values.
    - Pro tip: Create multiple Presets per type, and rely on the filter to associate the correct Preset by name.
- **Save and load manager settings:** Use Presets for a Manager window so the settings can be reused. For example, if you plan to reapply the same tags and layers or physics settings, Presets can reduce setup time for your next project.

Code standards

Coding standards will also help keep your team's work consistent and make it easier for developers to swap between different areas of your project. Again, there are no set-in-stone rules here. You need to decide what is best for your team – but once you've decided, stick with it.

As an example, namespaces can help organize your code better. They allow you to separate modules inside your project and avoid conflicts with third-party assets where class names may end up repeating.

When using namespaces in your code, break your folder structure up by the namespace for better organization.

A standard header is also a good practice. Including a standard header in your code template will help to document the purpose of a class, the date it was created, and even *who* created it. All of this is information that could easily get lost in the long history of a project, even when using version control.

Unity employs a template script to read from whenever you create a new Monobehaviour in the project. Every time you create a new script or shader, Unity uses a template stored in **%EDITOR_PATH%\Data\Resources\ScriptTemplates**:

- Windows: *C:\Program Files\Unity\Editor\Data\Resources\ScriptTemplates*
- Mac: */Applications/Hub/Editor/[version]/Unity/Unity.app/Contents/Resources/ScriptTemplates*

The default Monobehaviour template is this one: **81-C# Script-NewBehaviourScript.cs.txt**

使用预设的其他便捷方法包括：

— 创建具有默认值的游戏对象：将 Preset 资源拖放到 Hierarchy 中，以使用包含 Preset 值的相应组件创建新的游戏对象.

— 将特定类型与预设关联：在预设管理器（项目设置 **>** 预设管理器）中，为每个类型指定一个或多个预设。然后，创建新组件将默认为指定的 Preset 值。

— 专业提示：为每个类型创建多个预设，并依靠过滤器按名称关联正确的预设。

— 保存和加载管理器设置：对管理器窗口使用预设，以便可以重复使用设置。例如，如果您计划重新应用相同的标签和图层或物理设置，预设可以缩短下一个项目的设置时间。

代码标准

编码标准还将有助于保持团队工作的一致性，并使开发人员更容易在项目的不同领域之间切换。同样，这里没有一成不变的规则。您需要决定什么最适合您的团队——但一旦你决定了，就坚持下去。

例如，命名空间可以帮助更好地组织您的代码。它们允许您在项目中分隔模块，并避免与第三方资源发生冲突，因为类名称最终可能会重复。

在代码中使用命名空间时，请按命名空间拆分文件夹结构，以便更好地组织.

标准标头也是一种很好的做法。在代码模板中包含标准标头将有助于记录类的用途、创建日期，甚至 *who* 创建类。所有这些都很容易在项目的漫长历史中丢失，即使在使用版本控制时也是如此.

Unity 使用模板脚本，每当您在项目中创建新的 Monobehaviour 时，都会从中读取。每次创建新脚本或着色器时，Unity 都会使用存储在 %EDITOR_PATH%\Data\Resources\ScriptTemplates 中的模板：

— Windows：*C:\Program Files\Unity\Editor\Data\Resources\ScriptTemplates* — Mac：*/Applications/Hub/Editor/[version]/Unity/Unity.app/Contents/Resources/ScriptTemplates*

默认的 Monobehaviour 模板是这样的：81-C# Script-NewBehaviourScript.cs.txt

There are also templates for shaders, other behavior scripts, and assembly definitions.

For project-specific script templates, create an Assets/ScriptTemplates folder, and copy the script templates into this folder to override the defaults.

You can also modify the default script templates directly for all projects, but make sure you backup the originals before making any changes. Each version of Unity has its own template folder, so when you update to a new version, you need to replace the templates again.

The original 81-C# Script-NewBehaviourScript.cs.txt file looks like this:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

#ROOTNAMESPACEBEGIN#
public class #SCRIPTNAME# : MonoBehaviour
{
 // Start is called before the first frame update
 void Start()
 {
  #NOTRIM#
 }

 // Update is called once per frame
 void Update()
 {
  #NOTRIM#
 }
}
#ROOTNAMESPACEEND#
```

There are two keywords that may be helpful:

— **#SCRIPTNAME#** indicates the filename entered or the default filename (for example, NewBehaviourScript).

— **#NOTRIM#** ensures that the brackets contain a line of whitespace.

You can also use your own keywords and replace them with an Editor script implementing the OnWillCreateAsset method.

还有用于着色器、其他行为脚本和程序集定义的模板。

对于特定于项目的脚本模板，请创建一个 Assets/ScriptTemplates 文件夹，并将脚本模板复制到此文件夹中以覆盖默认值。

您还可以直接修改所有项目的默认脚本模板，但请确保在进行任何更改之前备份原始脚本模板。Unity 的每个版本都有自己的模板文件夹，因此当您更新到新版本时，需要再次替换模板。

原始的 81-C# Script-NewBehaviourScript.cs.txt 文件如下所示：

有两个关键字可能会有所帮助：

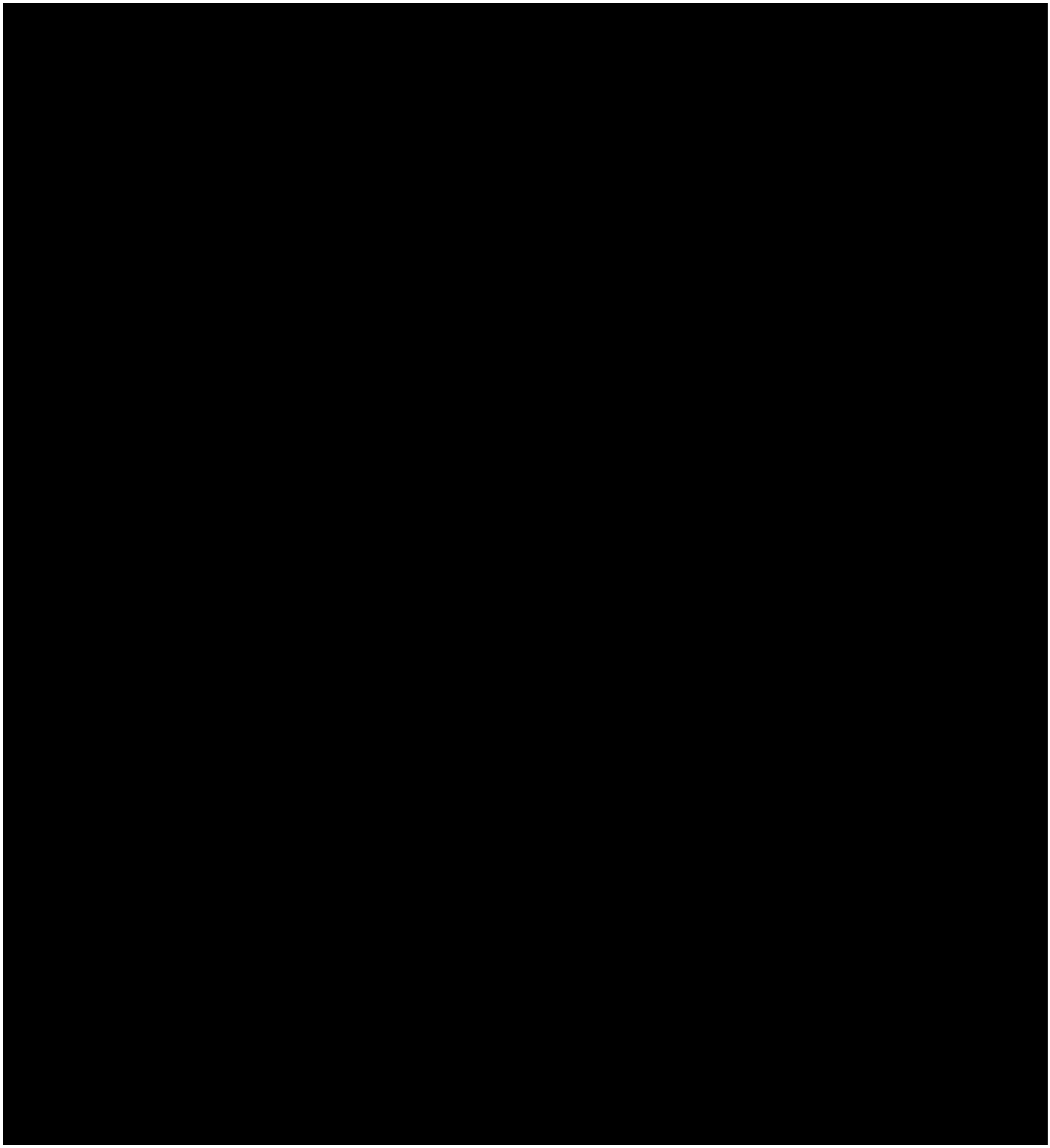
— #SCRIPTNAME# 表示输入的文件名或默认文件名（例如，NewBehaviourScript）。

— #NOTRIM# 确保括号中包含一行空格。

您还可以使用自己的关键字，并将其替换为实现 OnWillCreateAsset 方法的 Editor 脚本。

```
// /*-----------------------------------------
// -----------------------------------------
// Creation Date: #DATETIME#
// Author: #DEVELOPER#
// Description: #PROJECTNAME#
// -----------------------------------------
// -----------------------------------------*/

using UnityEngine;
using UnityEditor;

public class KeywordReplace : UnityEditor.AssetModificationProcessor {

 public static void OnWillCreateAsset (string path)
 {
  path = path.Replace(“.meta”, “”);
  int index = path.LastIndexOf(“.”);
  if (index < 0)
   return;

  string file = path.Substring(index);
  if (file != “.cs” && file != “.js” && file != “.boo”)
   return;

  index = Application.dataPath.LastIndexOf(“Assets”);
  path = Application.dataPath.Substring(0, index) + path;
  if (!System.IO.File.Exists(path))
   return;

  string fileContent = System.IO.File.ReadAllText(path);

  fileContent = fileContent.Replace(“#CREATIONDATE#”, System.DateTime.Today.ToString(“dd/MM/
yy”) + “”);
  fileContent = fileContent.Replace(“#PROJECTNAME#”, PlayerSettings.productName);
  fileContent = fileContent.Replace(“#DEVELOPER#”, System.Environment.UserName);

  System.IO.File.WriteAllText(path, fileContent);
  AssetDatabase.Refresh();
 }
}
```

You can also use your own keywords and replace them with an Editor script implementing the OnWillCreateAsset method.



您还可以使用自己的关键字，并将其替换为实现 OnWillCreateAsset 方法的 Editor 脚本。

UI Toolkit formatting conventions

UI Toolkit uses UXML code. As it's inspired by standard web technologies, UI Toolkit uses the kebab-case (also known as dash-case or lisp-case) for naming classes, the same convention used in CSS-related styling systems.

Just as with classes, HTML (and by extension UXML) typically uses kebab-case (lowercase with dashes) for ID names. Therefore, names like my-element-id would be more standard.

However, it's recommended that the ID names be specific and descriptive to easily identify the element's purpose. For example, submit-button or main-nav-container, logo-image etc.

	Recommended	Not recommended
Classes	.menu-bar-blue	ButtonBlue
Visual Element IDs	main-nav-image	

Recommended naming conventions in UXML

The main reasons for using kebab style include:

— CSS selectors are case-insensitive, so using Camel case or Pascal case (like .buttonBlue or .ButtonBlue) can lead to confusion.

— The HTML specification recommends making attribute names all lowercase, meaning .button-blue and not .buttonBlue or .ButtonBlue.

— Kebab case is easy to read and write when multiple words are involved.

Whatever naming convention you choose to go with, it's important to be consistent across your codebase. Consistent naming conventions across all types of scripts leads to cleaner code and makes it easier to access and modify by all team members.

Services for project organization

Asset Manager

The Unity Asset Manager is Unity's extensible, cloud-based digital asset management (DAM) solution that lets you increase discoverability, reuse, and ROI of content across your organization. Asset Manager lets you index content from Unity projects using cloud storage, making asset discovery easy for entire teams even when based in different locations. It's key features include:

— Better asset management

— Intuitive browsing and discovery

— Essential integrations with creative tools

— Lifecycle management

— Role-based permissions

— Flexibility and extensibility

UI Toolkit 格式约定

UI Toolkit 使用 UXML 代码。由于受到标准 Web 技术的启发，UI Toolkit 使用 kebab-case（也称为破折号或 lisp-case）来命名类，与 CSS 相关样式系统中使用的约定相同。

就像类一样，HTML（以及扩展的 UXML）通常使用 kebab-case（小写带破折号）作为 ID 名称。因此，像 my-element-id 这样的名称会更标准。

但是，建议 ID 名称具体且具有描述性，以便轻松识别元素的用途。例如，submit-button 或 main-nav-container、logo-image 等。

	Recommended	Not recommended
Classes	.menu-bar-blue	ButtonBlue
Visual Element IDs	main-nav-image	

UXML 中建议的命名约定

使用 kebab 风格的主要原因包括：

— CSS 选择器不区分大小写，因此使用 Camel 大小写或 Pascal 大小写（如 .buttonBlue 或。 ButtonBlue） 可能会导致混淆。— HTML 规范建议将属性名称全部小写，即 .button-blue，而不是 .buttonBlue 或 .ButtonBlue。— 当涉及多个单词时，Kebab 大小写很容易阅读和写入。

无论您选择使用哪种命名约定，在代码库中保持一致都很重要。所有类型的脚本的一致命名约定可以使代码更简洁，并使所有团队成员更容易访问和修改。

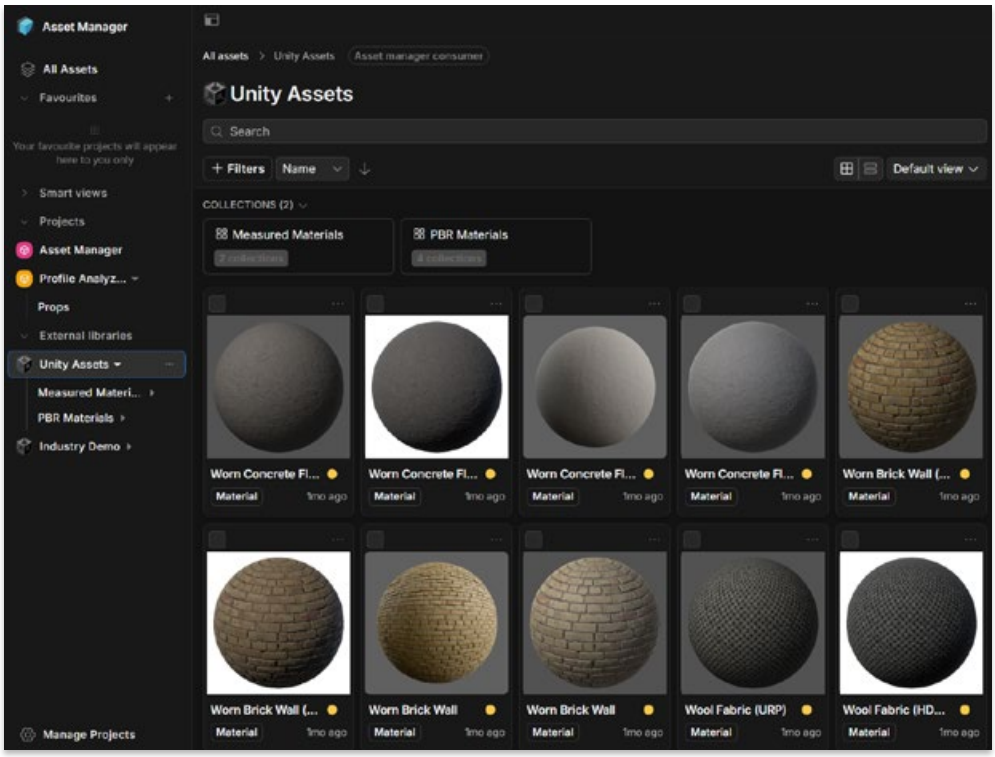
项目组织服务

资产管理器

Unity Asset Manager 是 Unity 基于云的可扩展数字资产管理 （DAM） 解决方案，可让您提高整个组织中内容的可发现性、重用性和投资回报率。Asset Manager 允许您使用云存储为 Unity 项目中的内容编制索引，使整个团队即使位于不同位置也能轻松发现资产。它的主要功能包括：

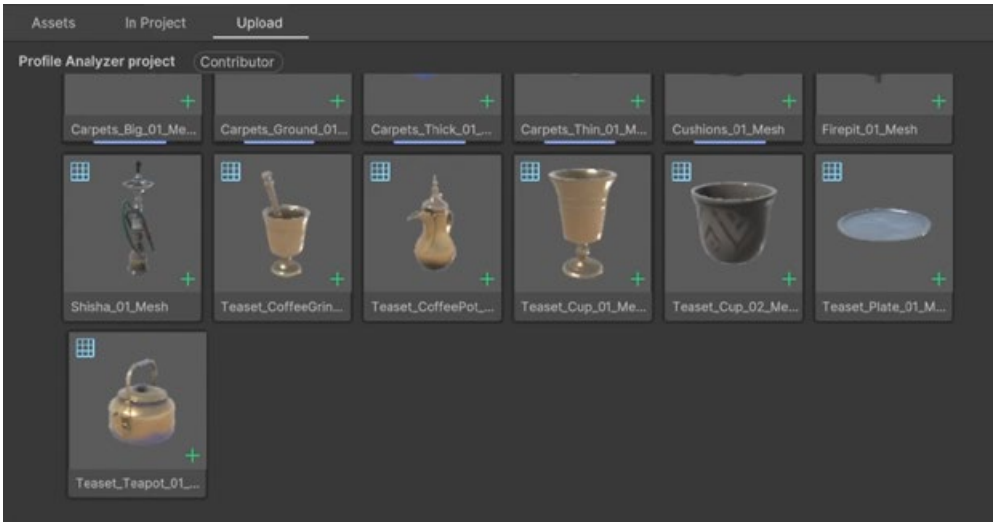
— 更好的资源管理 — 直观的浏览和发现 — 与创意工具的基本集成 — 生命周期管理 — 基于角色的权限 — 灵活性和可扩展性

You will find a collection of free materials and textures and demos available through the dashboard when logging into Unity Cloud. Open the **Unity Assets** sub section to download these into your projects.



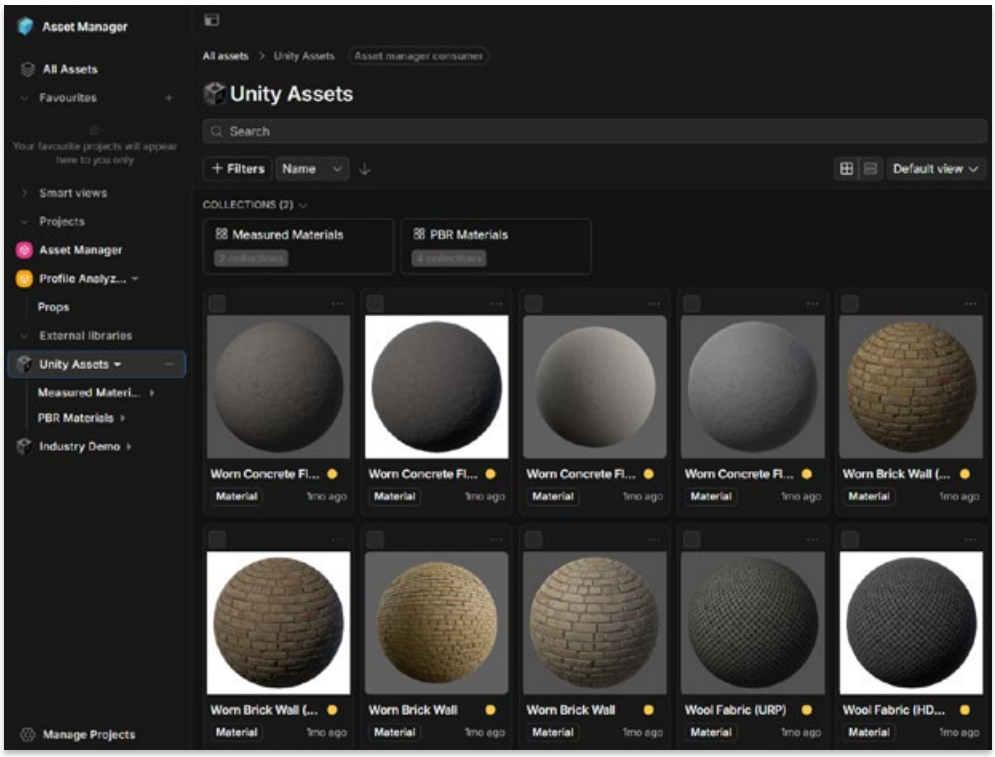
Free materials and textures on Unity Cloud in the Unity Assets section

Inside Unity, you can drag files from your current project into the **Asset Manager** window to upload them to the cloud. Those files are then available to any of your other projects that use Asset Manager as well as to team members connected to your project using UVCS.



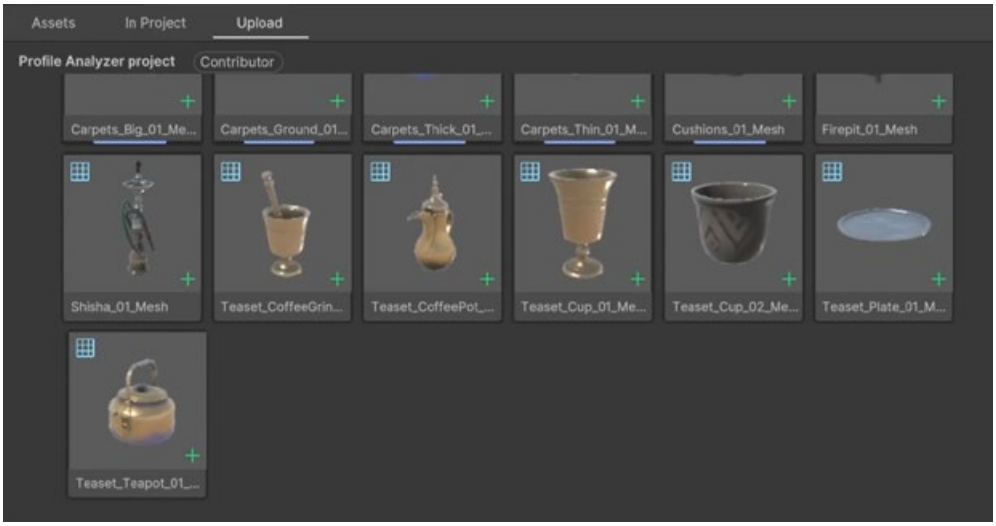
Drag assets into the Upload tab to upload to the cloud.

登录 Unity Cloud 时，您可以通过仪表板找到一系列免费的材质、纹理和演示。打开 Unity Assets 子部分，将这些内容下载到您的项目中。



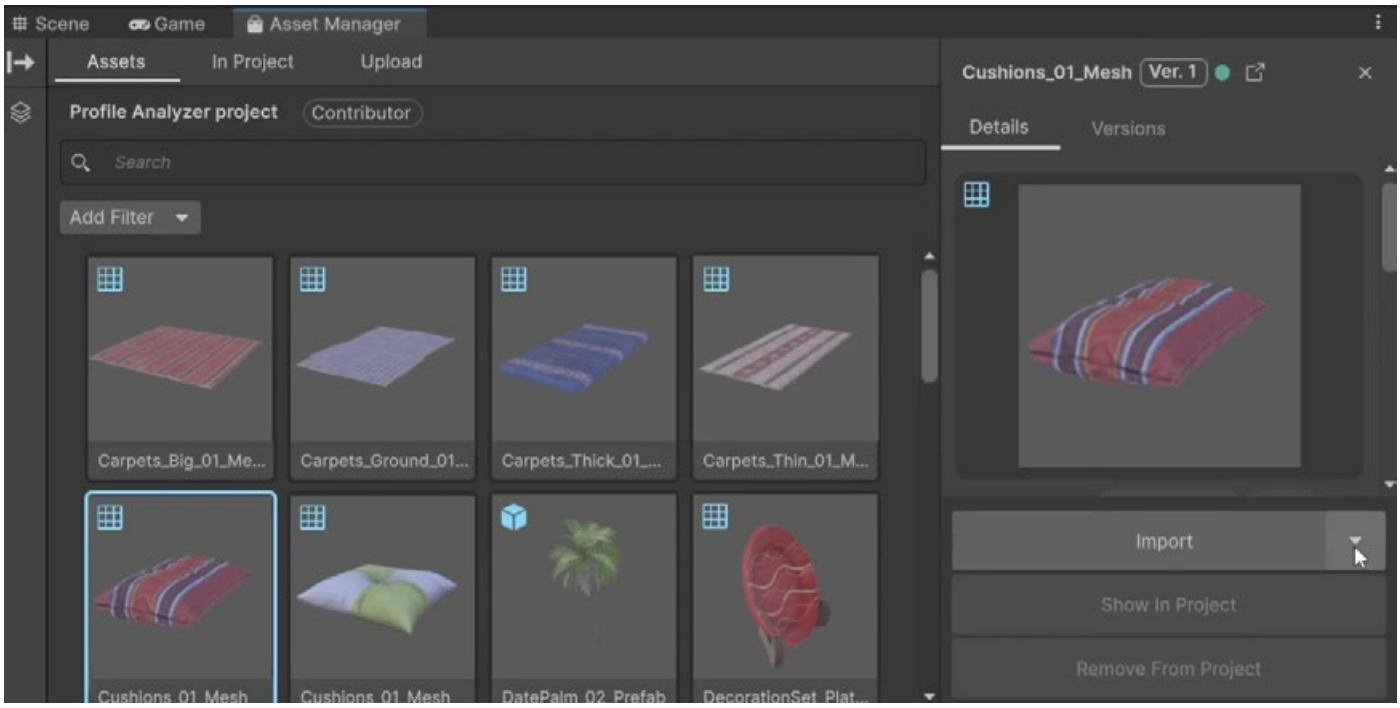
Unity Cloud 上 Unity Assets 部分中的免费材质和纹理

在 Unity 中，您可以将文件从当前项目拖动到 Asset Manager 窗口中，以将其上传到云中。然后，这些文件可供使用 Asset Manager 的任何其他项目以及使用 UVCS 连接到项目的团队成员使用。



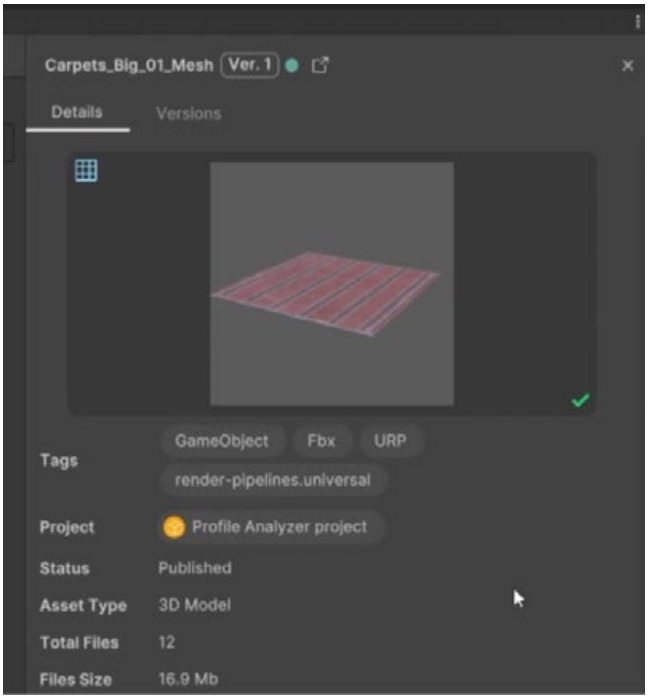
将资产拖动到 Upload 选项卡以上传到云。

You can then import files into other projects directly from the Asset Manager window, making this system easy to use and a comprehensive archive of all your project assets.



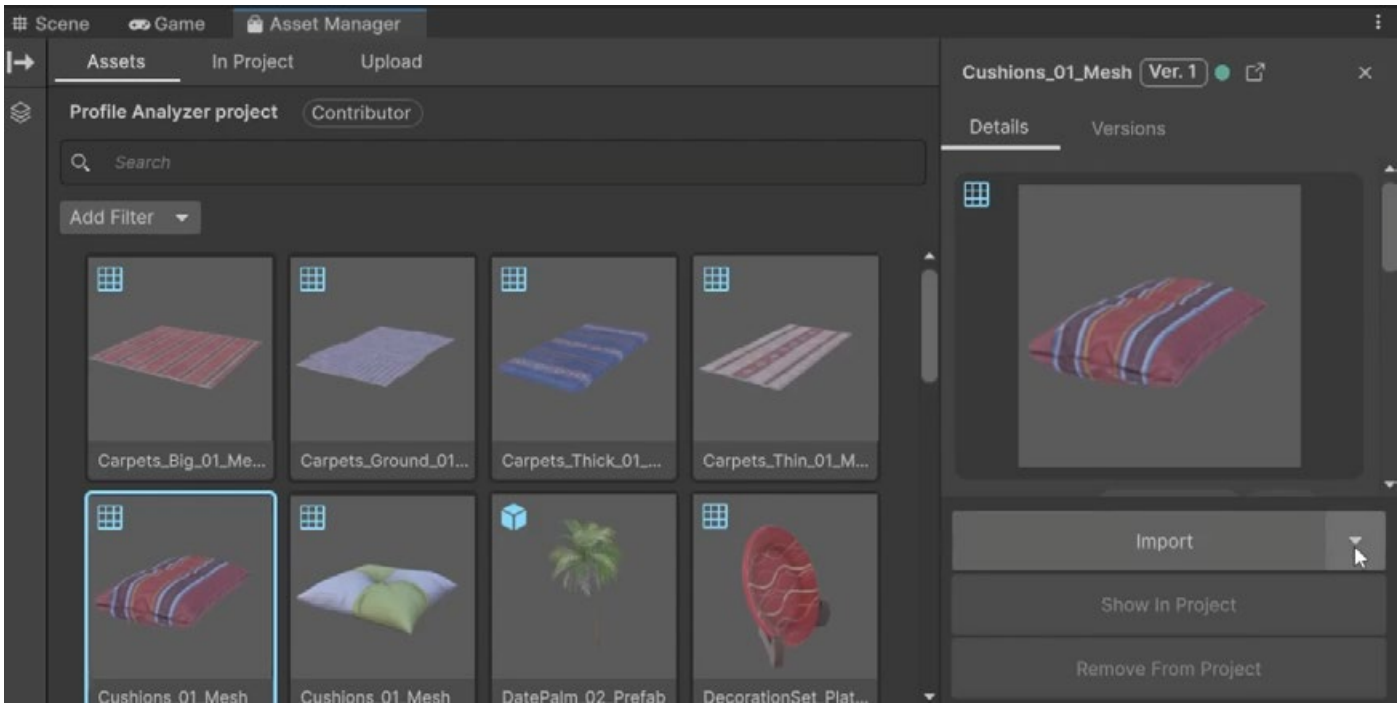
Click import to download and import the files into your project.

Tags are auto generated to enable quick search when you have lots of assets. The assets are also linked to projects, so you can limit your search to assets that belong to a specific project or artist.



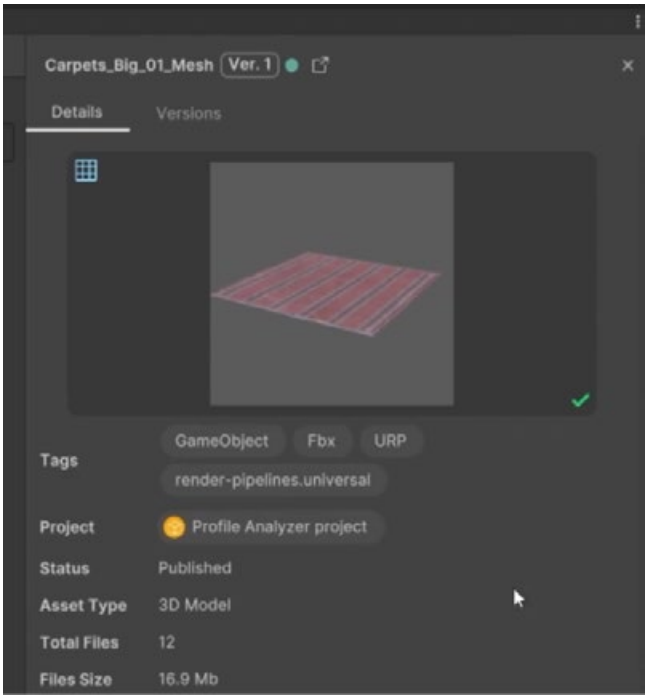
Auto-generated tags make searching for assets quick and easy.

然后，您可以直接从 Asset Manager 窗口将文件导入到其他项目中，使该系统易于使用并全面存档所有项目资源。



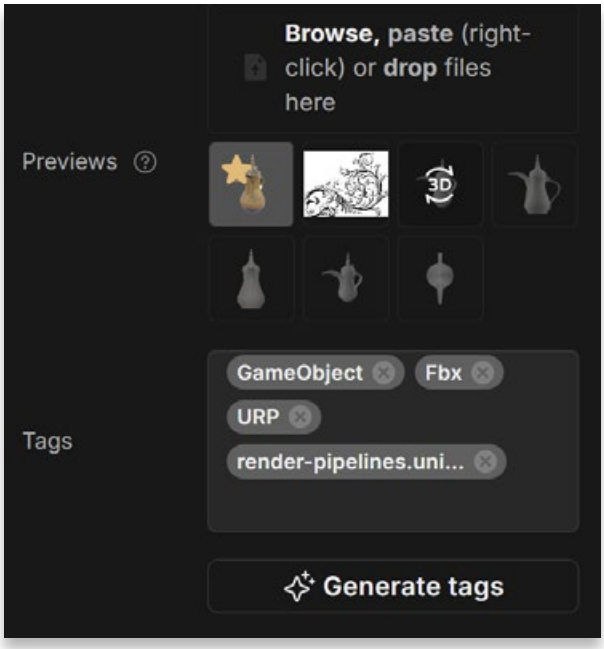
单击 import （导入） 以下载文件并将其导入到您的项目中。

标记是自动生成的，以便在您拥有大量资源时启用快速搜索.资源也链接到项目，因此您可以将搜索限制为属于特定项目或艺术家的资源。



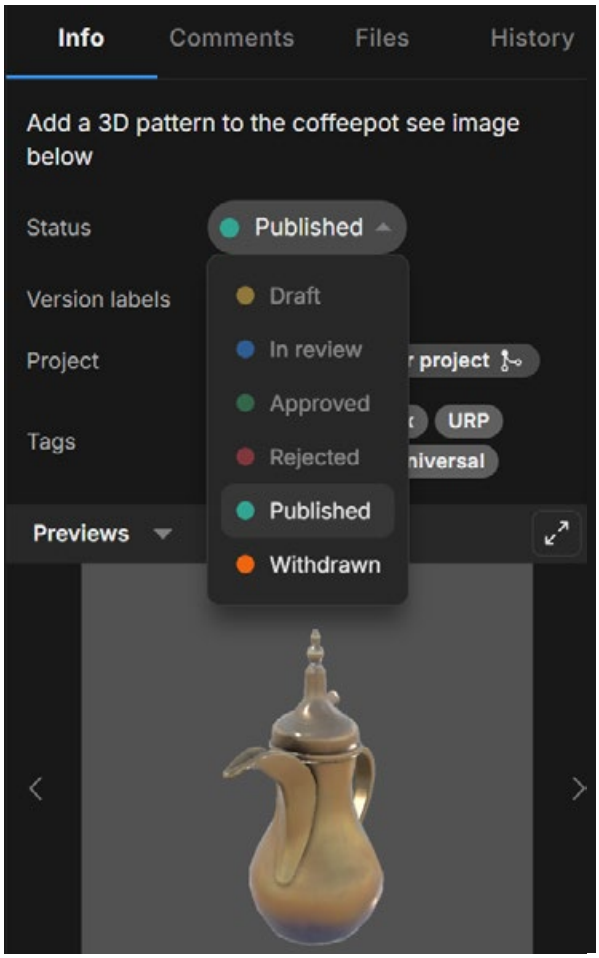
自动生成的标签使搜索资产变得快速而简单。

Assets can be edited in the Unity Cloud. Add custom thumbnails, extra image previews, more tags, and comments.



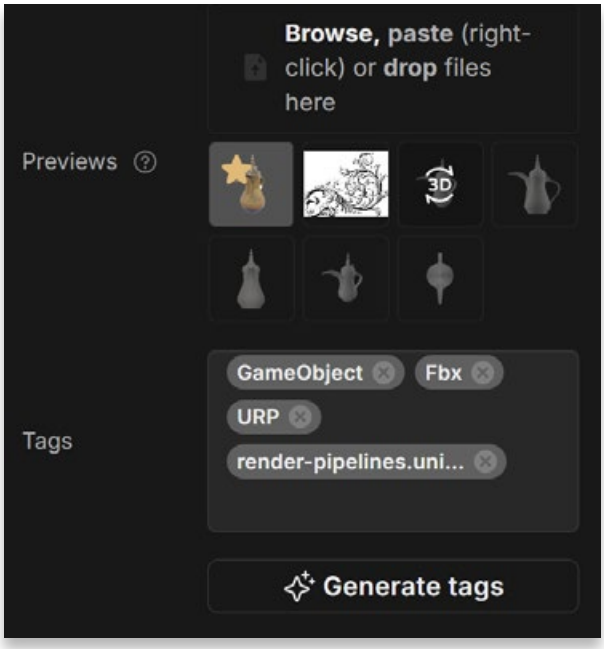
Edit assets to add extra image previews and tags.

The status of the model can be changed on the Cloud to let others know if the asset is in draft, in review, approved, rejected, published, or withdrawn. Adding a comment allows you to include further details about changes. The status of the asset will then be visible in the Asset Manager window in Unity.



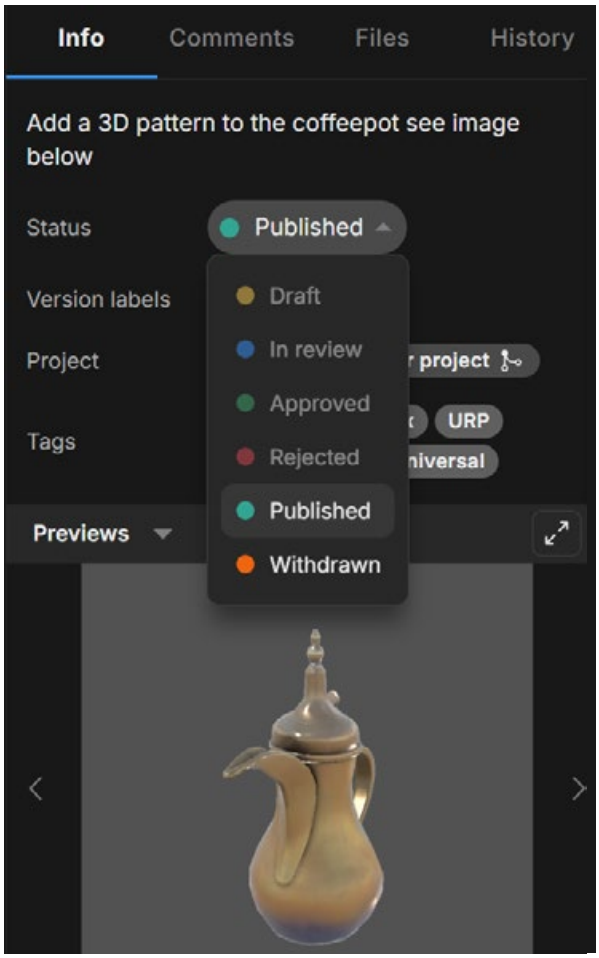
Change an asset's status.

可以在 Unity Cloud 中编辑资源。添加自定义缩略图、额外的图像预览、更多标签和评论。



编辑资源以添加额外的图像预览和标记.

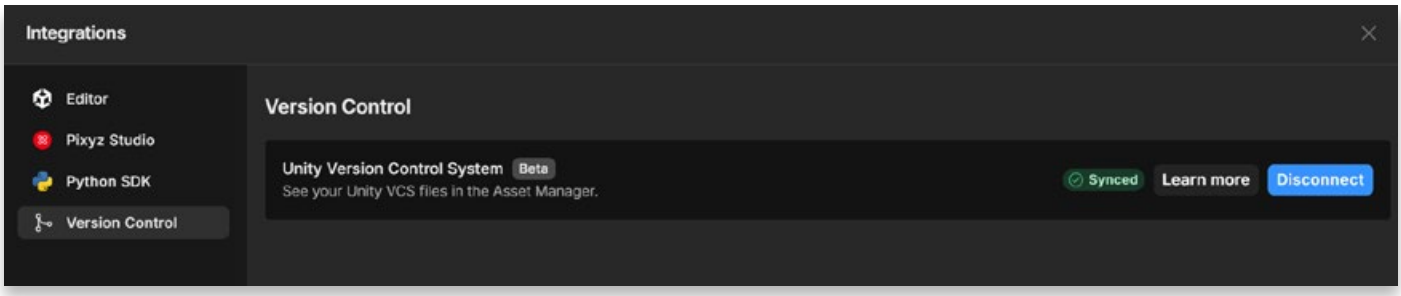
可以在云上更改模型的状态，让其他人知道资产是处于草稿状态、审核中、已批准、已拒绝、已发布还是已撤回。通过添加注释，您可以包含有关更改的更多详细信息。然后，资源的状态将显示在 Unity. 的 Asset Manager 窗口中



更改资产的状态。

Team members always have access to the latest version, reducing issues when working with asset iterations.

Asset Manager works with UVCS by enabling the integration on the Cloud.



Integrate Asset Manager to UVCS to synchronize the data.

Any modifications made through UVCS will automatically synchronize with Unity Asset Manager, preventing redundant content creation within your team.

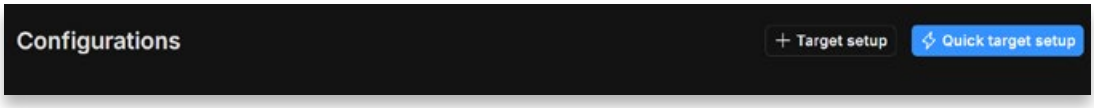
Learn more about the benefits of Unity Asset Manager here.

You can also watch this Asset Manager tutorial on Unity Learn.

Build Automation

Build automation is an integral part of any DevOps strategy. You can run builds on the cloud as well as run builds simultaneously for all your target platforms. Instead of waiting for builds to complete sequentially on your device you can spend that time creating your project.

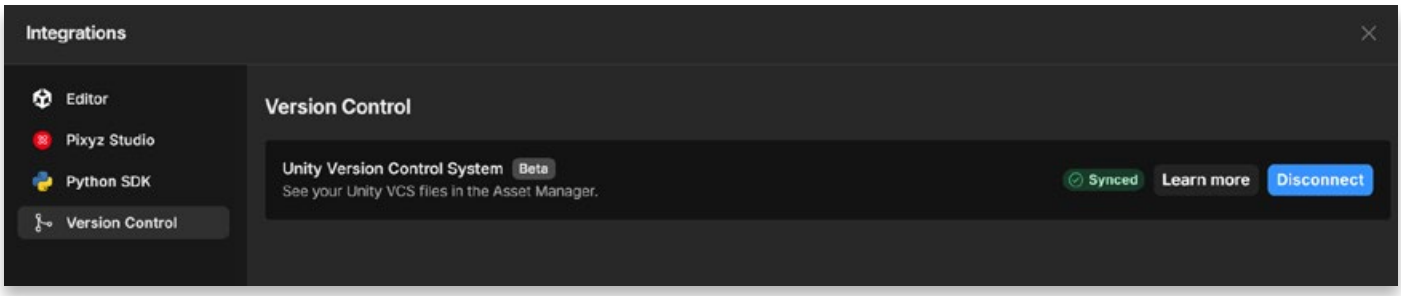
In the Unity Cloud dashboard you can set up as many build targets as you like from the **Configurations** section.



Click **Quick target setup** to create a Build target (alternatively, you can click Target setup if you want to configure more advanced options).

团队成员始终可以访问最新版本，从而减少使用资产迭代时的问题.

Asset Manager 通过在 Cloud. 上启用集成来与 UVCS 配合使用



将 Asset Manager 集成到 UVCS 以同步数据.

通过 UVCS 所做的任何修改都将自动与 Unity Asset Manager 同步，从而防止在团队中创建冗余内容。

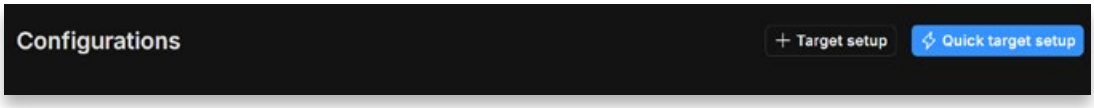
详细了解 Unity Asset Manager her 的优势 e.

您还可以在 Unity Learn 上观看此 Asset Manager 教程.

构建自动化

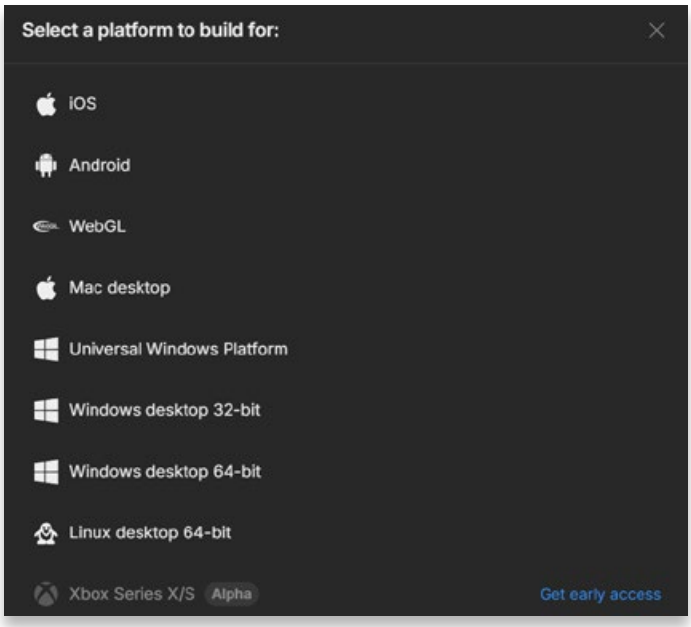
构建自动化是任何 DevOps 策略不可或缺的一部分。您可以在云上运行构建，也可以为所有目标平台同时运行构建。您无需等待构建在您的设备上按顺序完成，而是可以将这些时间花在创建项目上。

在 Unity Cloud 控制面板中，您可以从 Configurations 部分. 设置任意数量的构建目标



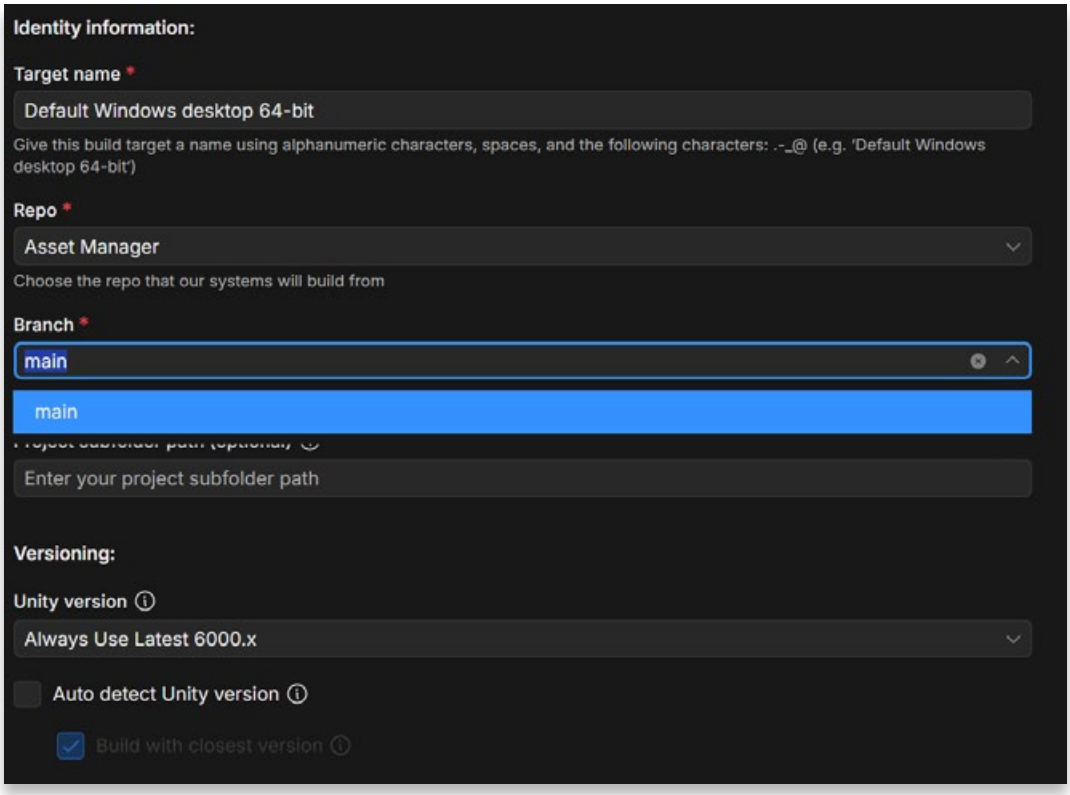
单击 Quick target setup 以创建 Build 目标（或者，也可以单击 Target setup （如果要配置更高级的选项），也可以单击 Target setup）。

Choose a platform to build for. You can set up numerous build targets for each platform.



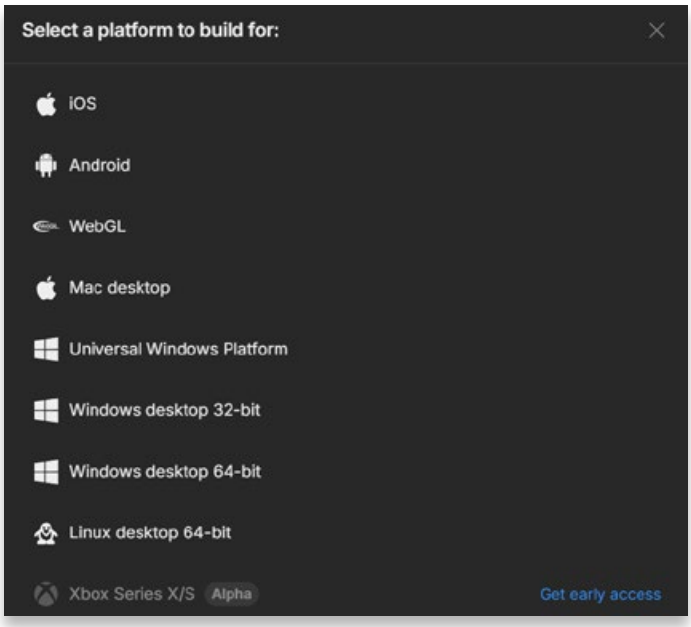
Target platforms to build for

You can build from any branch, and set up multiple targets for each branch. You can also select specific versions of Unity.



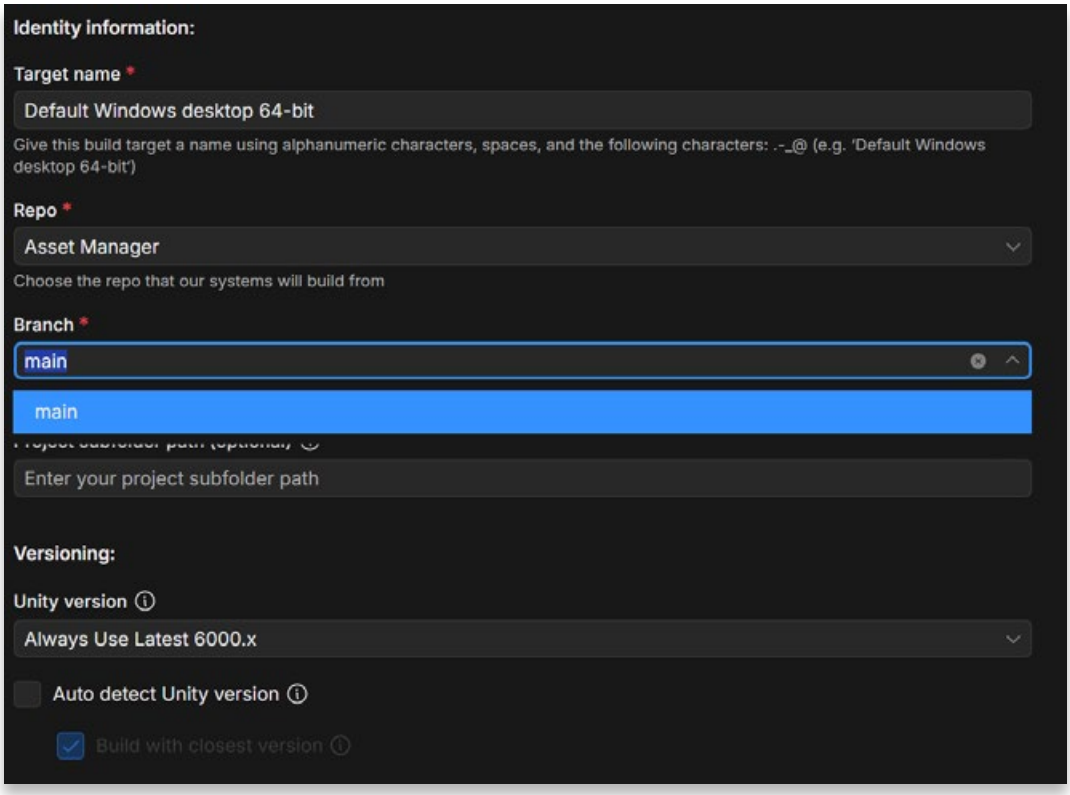
Choose a branch to create the build from.

选择要为其构建的平台。您可以为每个平台设置多个构建目标。



要为其构建的目标平台

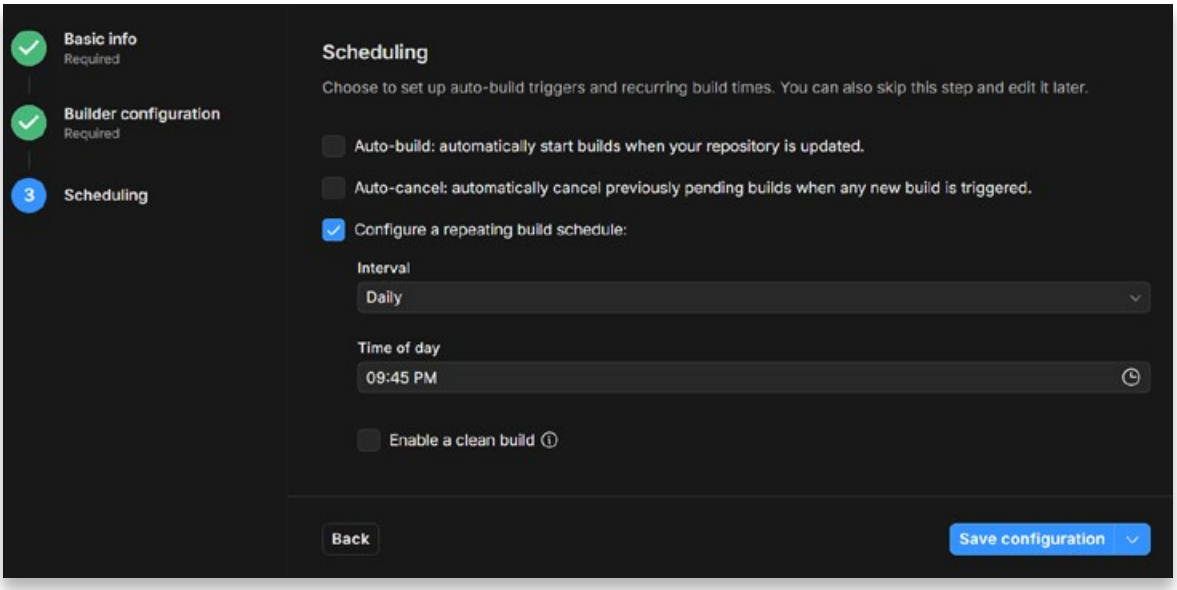
您可以从任何分支构建，并为每个分支设置多个目标。您还可以选择特定版本的 Unity。



选择要从中创建生成的分支。

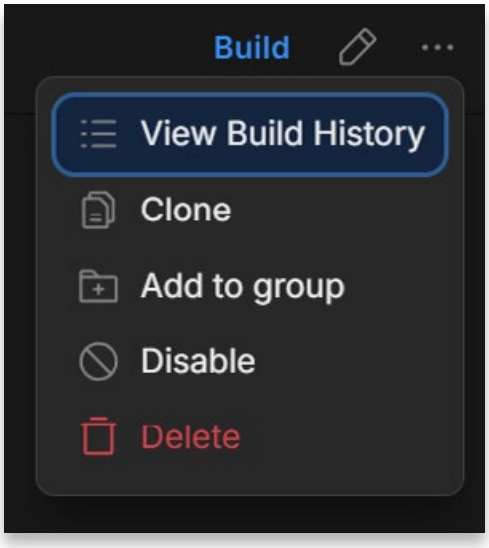
Note: Windows 11 in the Builder operating system, at the time of writing, offers 200 free minutes of build time per month.

You can then set up a schedule. You can either build once to test your progress or set up a repeating schedule. The example below shows a daily schedule that will produce a build at 9.45 pm so that the latest game build is available every morning to check for any issues or bugs. You will receive an email message after completion of the build to indicate its status, whether it was a success or a fail.



Create a schedule for your builds

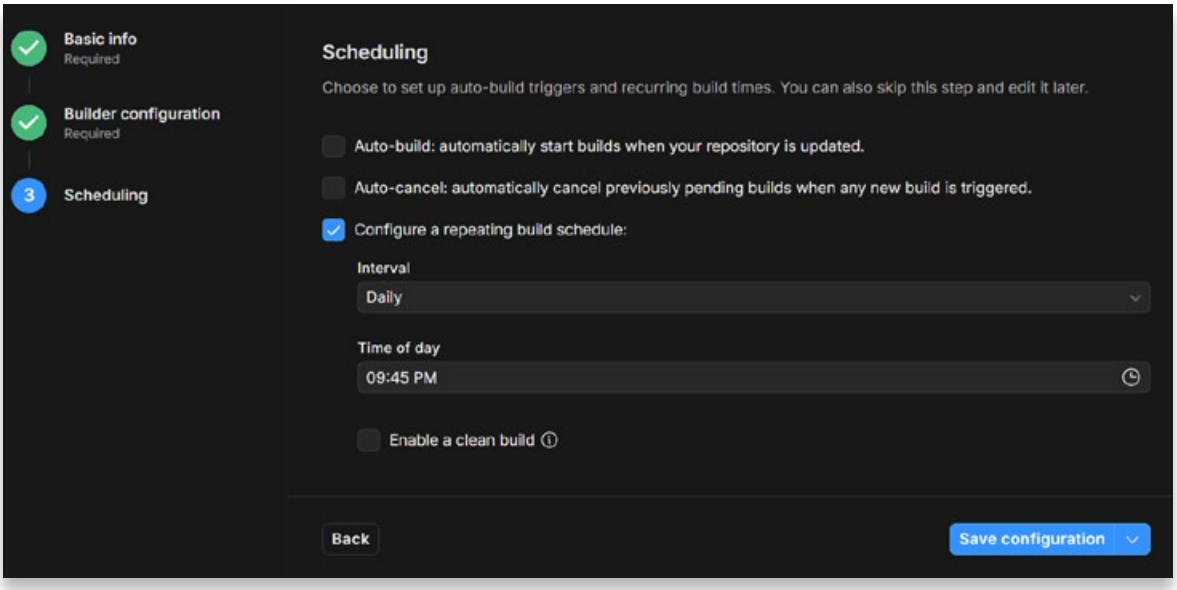
From the configurations section you can click **Build** next to your setup to build immediately. You can also pause or delete a build configuration.



Disable or delete a build schedule.

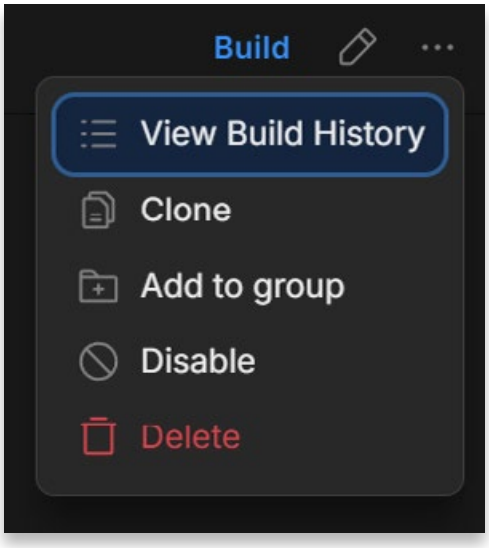
注意：在撰写本文时，Builder作系统中的 Windows 11 每月提供 200 分钟的免费构建时间。

然后，您可以设置计划。您可以构建一次来测试您的进度，也可以设置一个重复的计划。以下示例显示了一个每日计划，该计划将在晚上 9.45 生成生成版本，以便每天早上提供最新的游戏生成版本以检查是否存在任何问题或错误。构建完成后，您将收到一封电子邮件，以指示其状态，无论是成功还是失败。



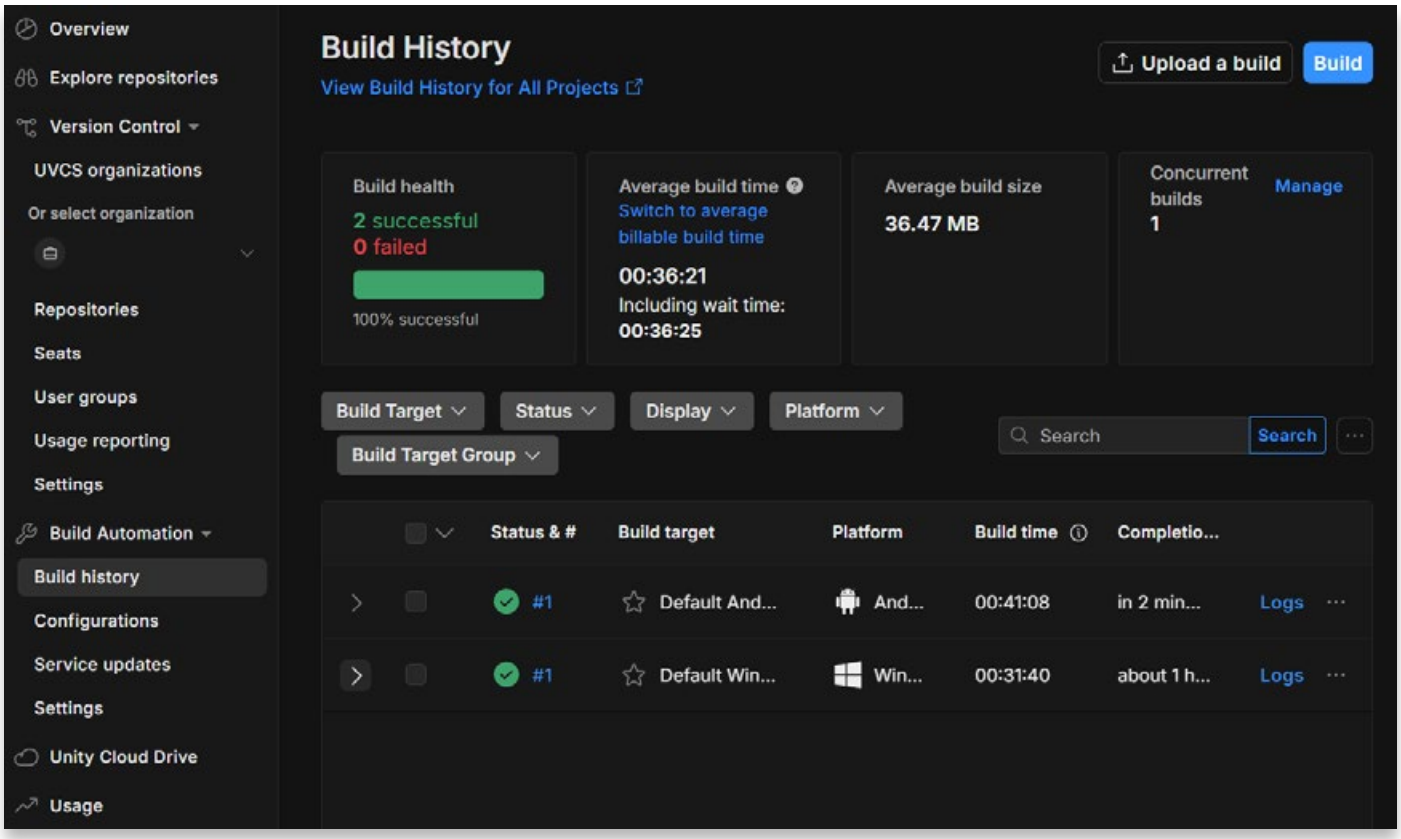
为您的构建创建计划

在 configurations （配置） 部分中，您可以单击 Build （构建） 旁边的设置以立即构建。 您还可以暂停或删除构建配置。



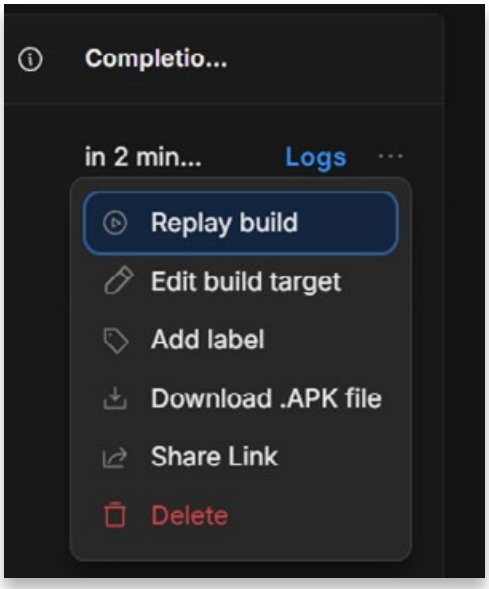
禁用或删除构建计划.

In the **Build History** section on the Build Automation dashboard, you can check the results of your builds. If any of them have failed, you can troubleshoot the failure by checking the logs.



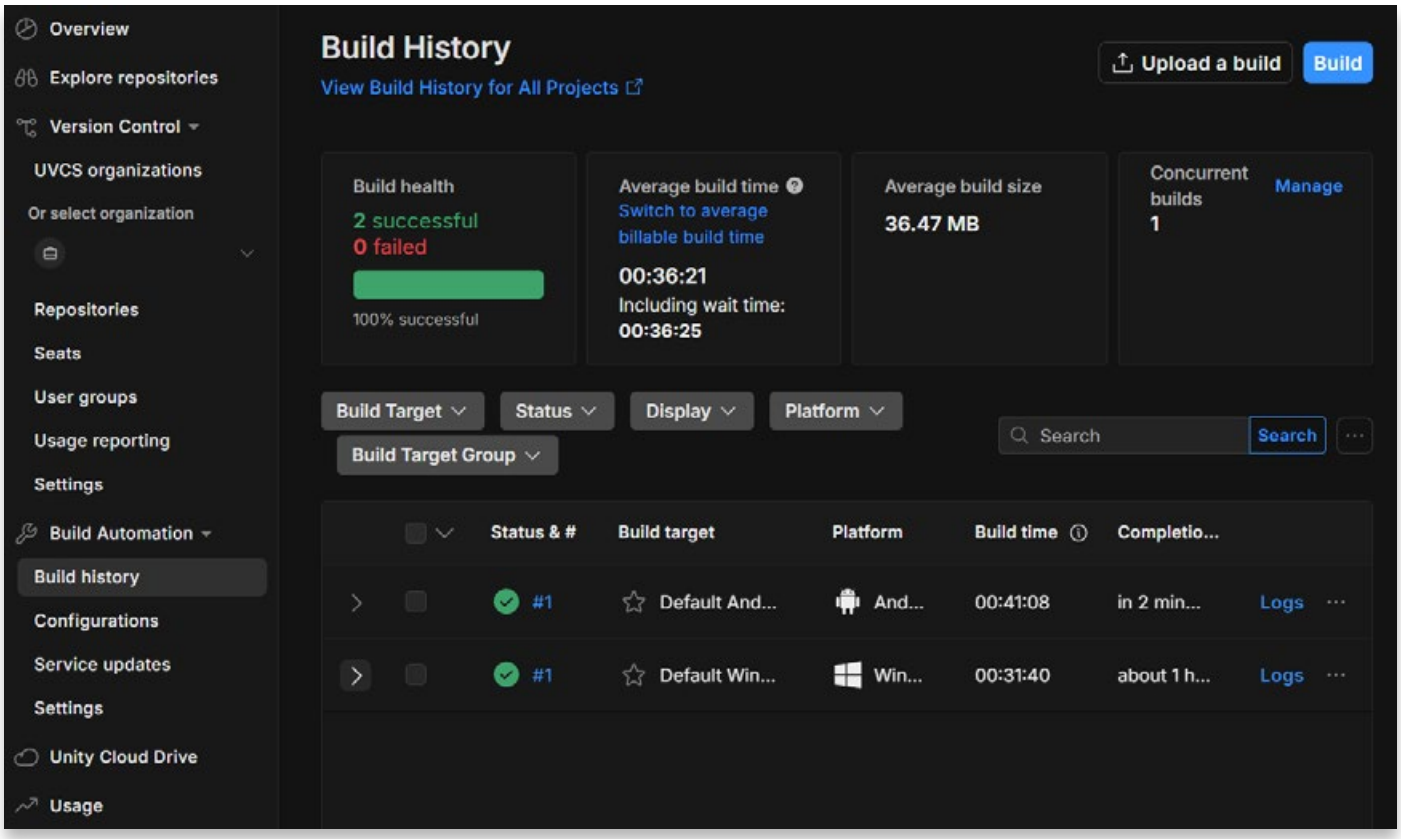
Build history in the Cloud

You can download, share, or delete builds from the cloud.



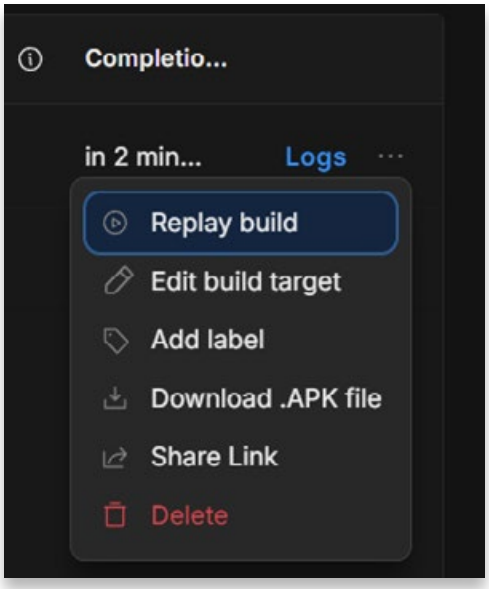
Download, share or delete builds.

在 Build Automation 控制面板的 Build History （构建历史记录） 部分中，您可以检查构建的结果。如果其中任何一个失败，您可以通过检查日志来排查故障。



在云中构建历史记录

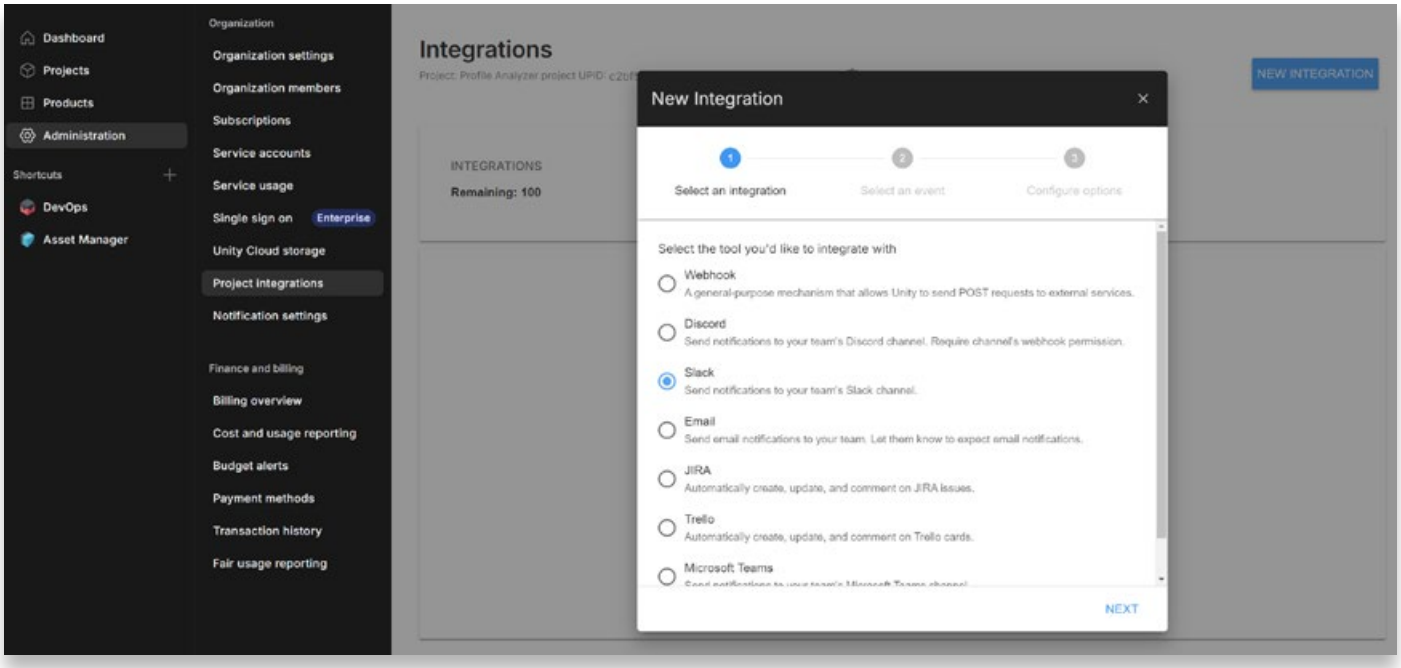
您可以从云中下载、共享或删除内部版本。



下载、共享或删除内部版本。

Set up integrations if you would like messages to be sent to other applications such as Discord or Slack after the completion of a build.

Go to **Administration** then **Project Integrations** on the Cloud and click on **New Integration**.



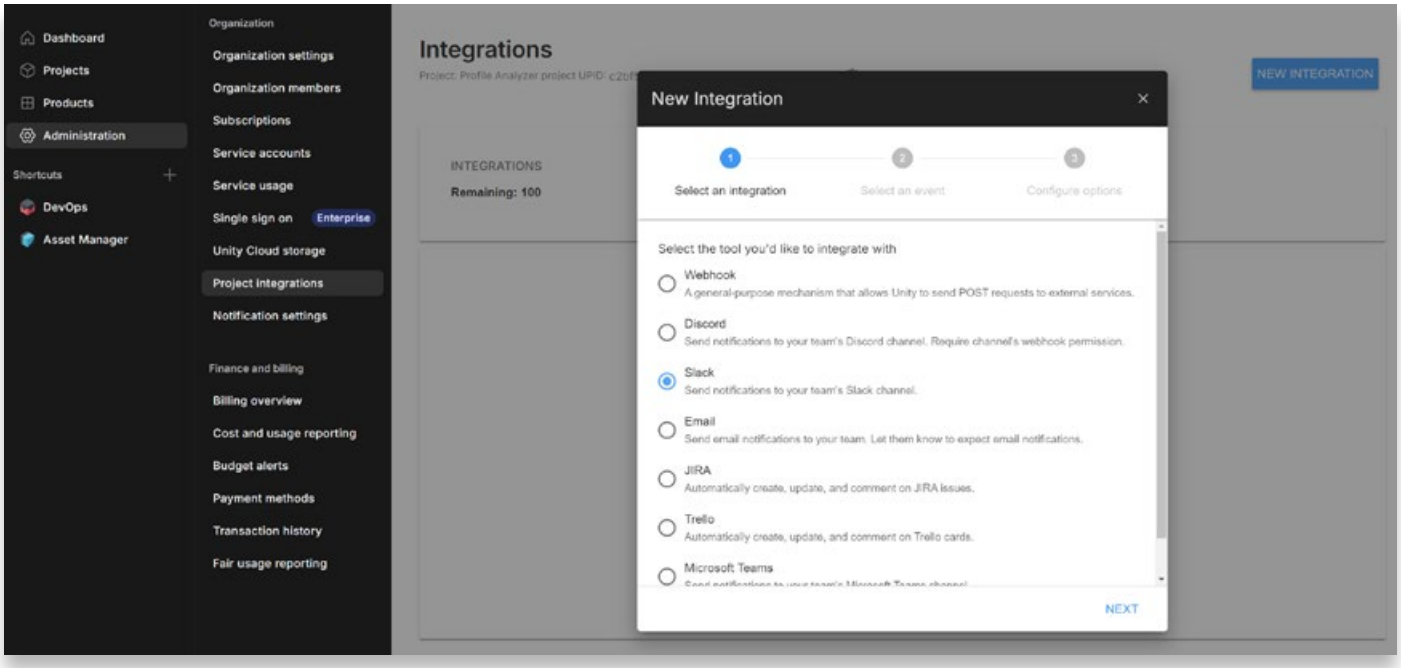
Add new integrations via Administration > Project Integrations.

Choose the application/s you would like to use. Now messages will be sent via those applications to inform team members.

Learn more about Unity Build Automation.

如果您希望在构建完成后将消息发送到其他应用程序，例如 Discord 或 Slack，请设置集成。

转到 Administration 然后转到 Project Integrations on the Cloud，然后单击 New Integration.



通过管理 > 项目集成添加新的集成。

选择您要使用的应用程序。现在，将通过这些应用程序发送消息以通知团队成员。

Learn more about Unity Build Automation.

Version control systems

Now that you're familiar with some of the key terms and concepts in version control, project organization, and naming conventions best practices, it's time to introduce some of the key players. Of course, no one solution is best for everyone. There are many things to consider when choosing which VCS to use in your team. Hopefully, by the end of this book, you'll have all the information you need to make that decision.

Git

Open source, free, and flexible, Git is one of the most popular version control systems around. However, as a distributed setup it can be daunting to non-technical users.

Developed in 2005 by Linus Torvalds to control the Linux kernel development, it's remained well-maintained and open source since. Git as a platform is a command line-only tool. But many different GUIs have been developed for it, making the system more accessible to users.

There can often be some confusion between Git and GitHub. GitHub is a hosting service for Git repositories, but you can use Git without using GitHub. That said, GitHub is a very popular service because there is a free version (with some limitations), and it doesn't require any custom server setups.

版本控制系统

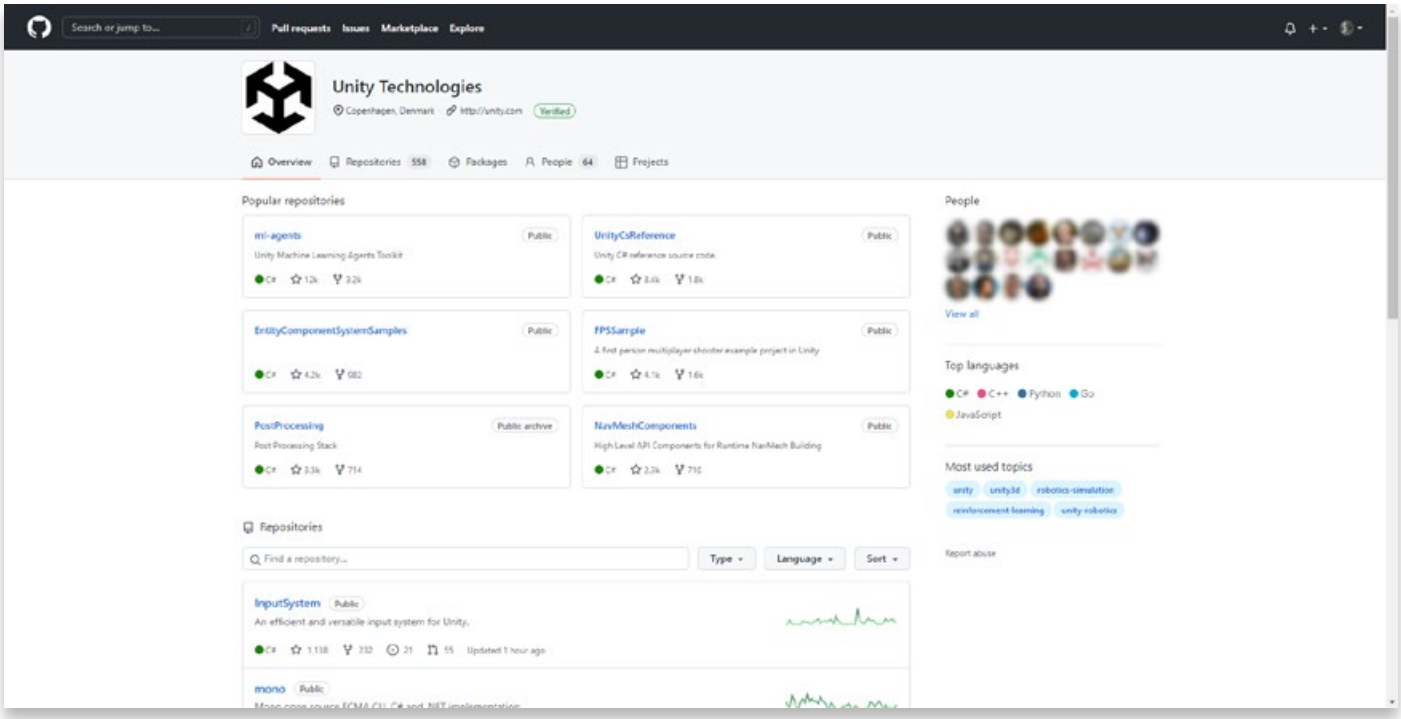
现在，您已经熟悉了版本控制、项目组织和命名约定最佳实践中的一些关键术语和概念，是时候介绍一些关键参与者了。当然，没有一种解决方案适合所有人。在选择在团队中使用哪种 VCS 时，需要考虑很多事情。希望在本书结束时，您将拥有做出该决定所需的所有信息。

Git 公司

Git 开源、免费且灵活，是最受欢迎的版本控制系统之一。 但是，作为分布式设置，对于非技术用户来说，它可能会令人生畏。

由 Linus Torvalds 于 2005 年开发以控制 Linux 内核开发，此后一直保持良好维护和开源。Git 作为平台是仅限命令行的工具。但是已经为它开发了许多不同的 GUI，使用户更容易访问该系统。

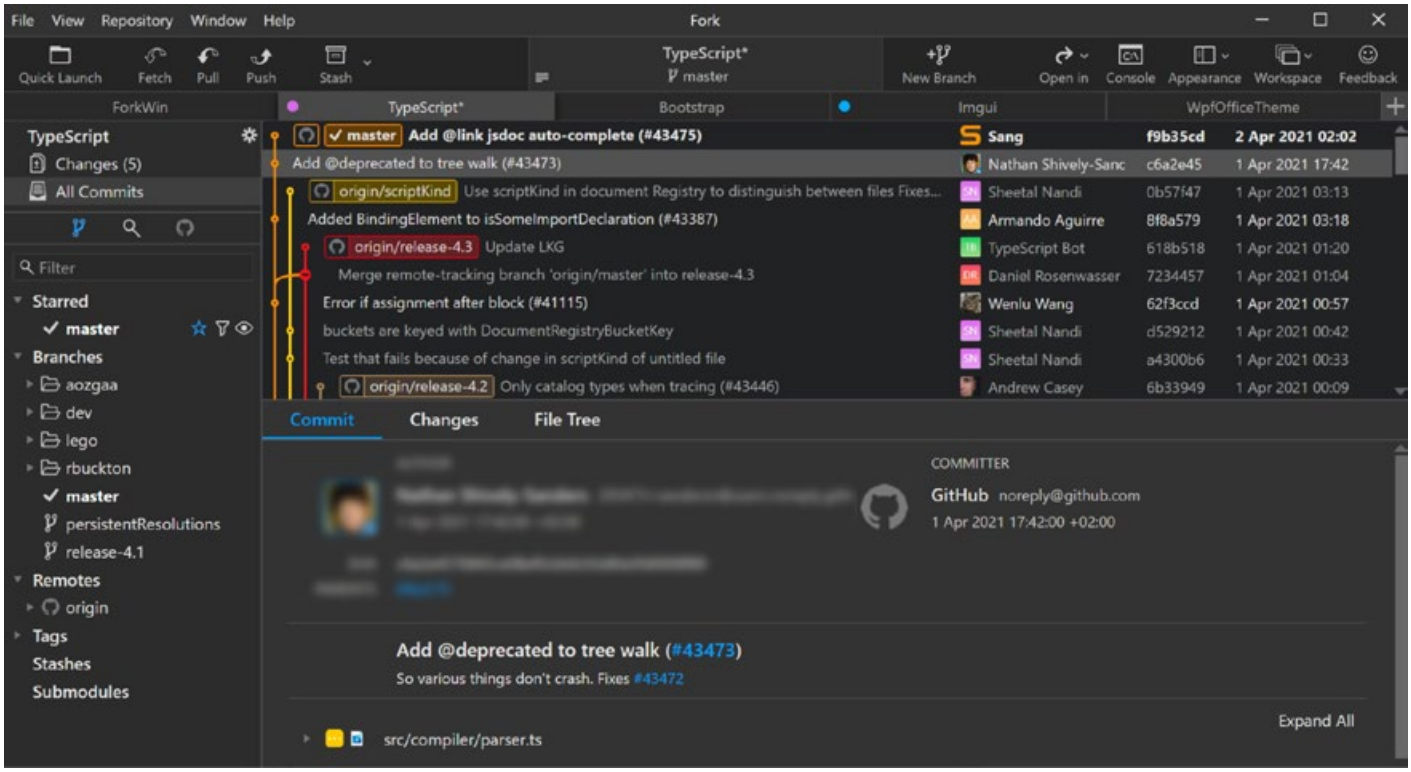
Git 和 GitHub 之间经常会有一些混淆。GitHub 是 Git 存储库的托管服务，但您可以在不使用 GitHub 的情况下使用 Git。也就是说，GitHub 是一项非常受欢迎的服务，因为有一个免费版本（有一些限制），并且它不需要任何自定义服务器设置。



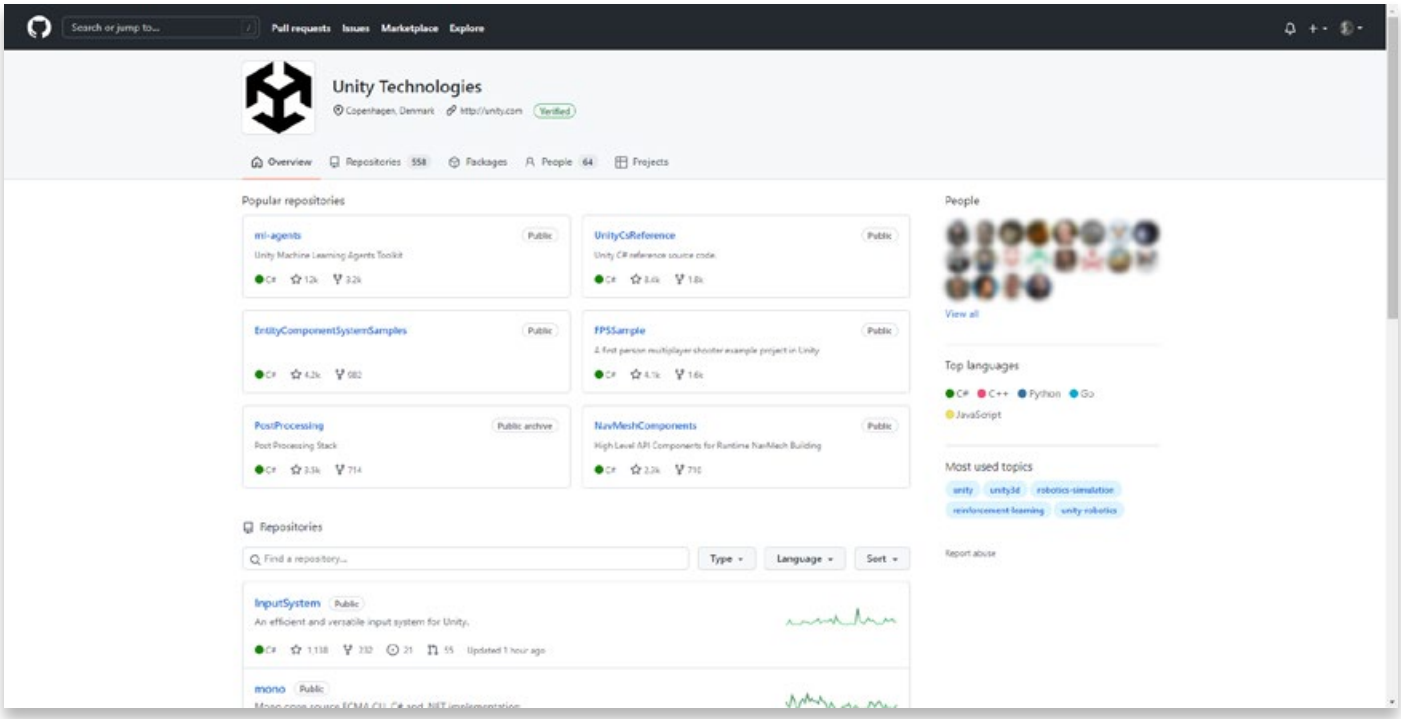
GitHub

Some popular Git GUI clients include:

Fork: It's a fast and friendly GUI that you can download for a free evaluation.



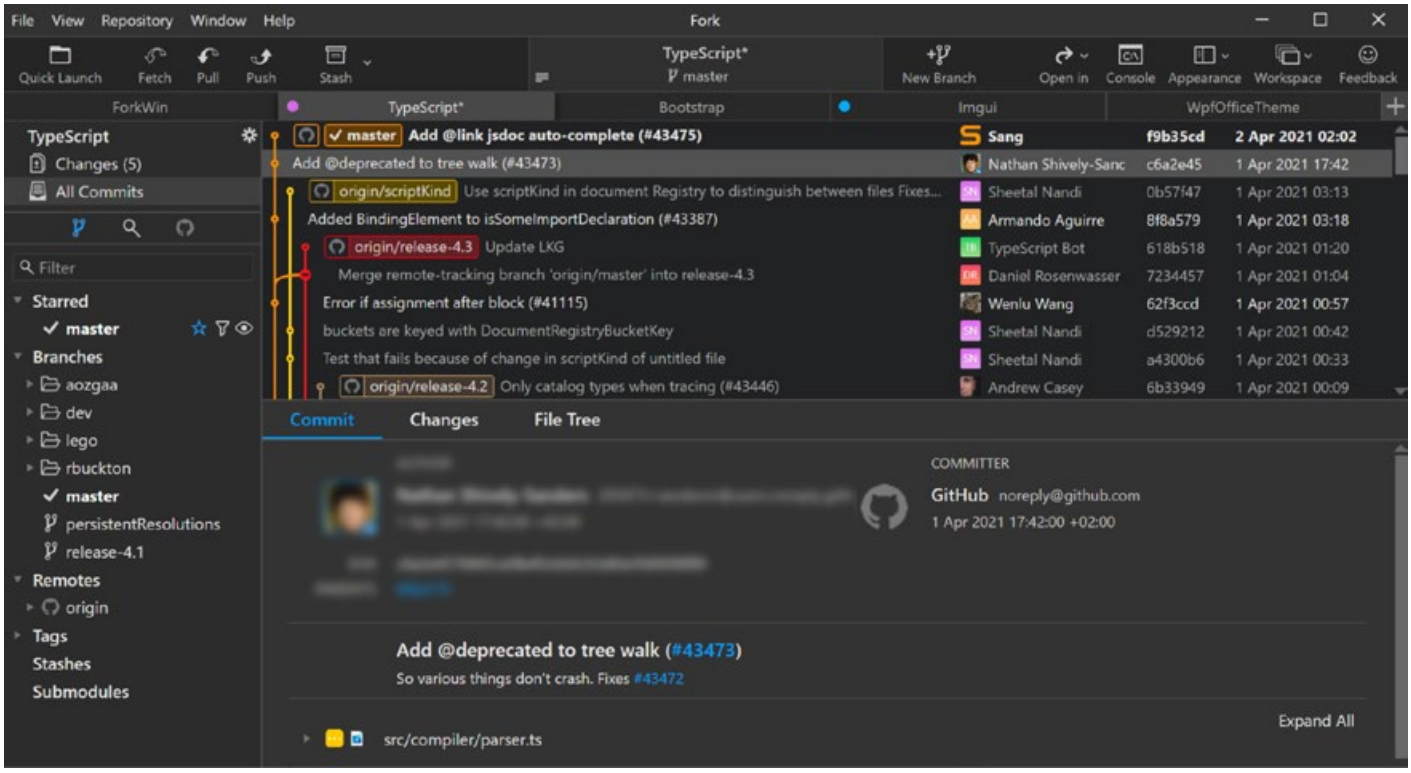
Fork



GitHub

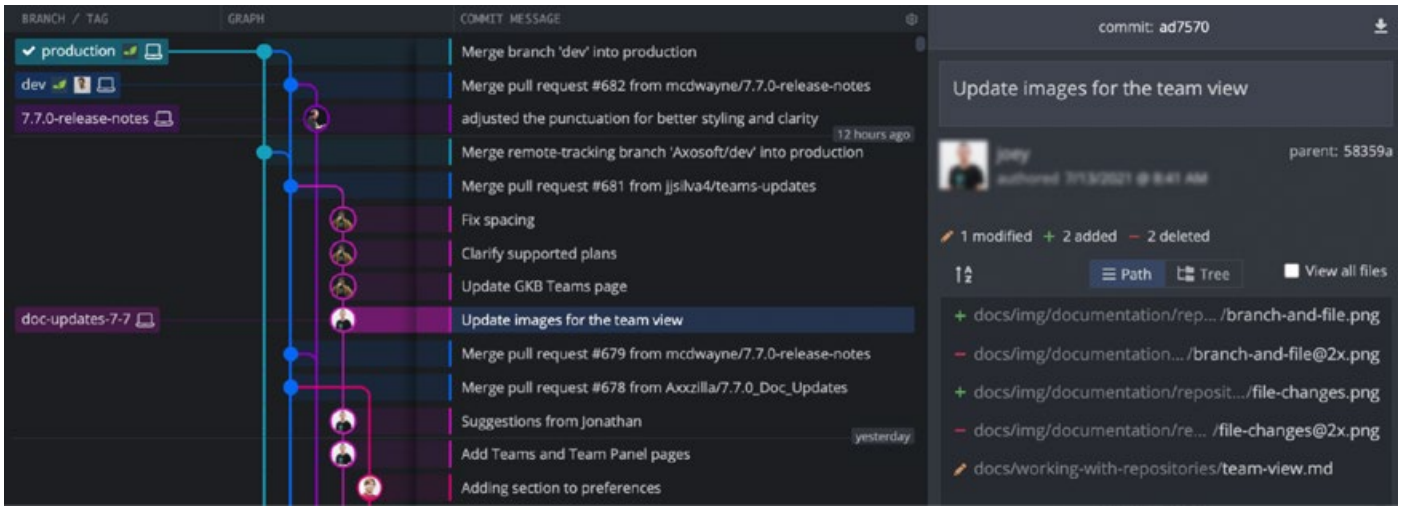
一些流行的 Git GUI 客户端包括：

Fork：这是一个快速友好的 GUI，您可以下载以进行免费评估。



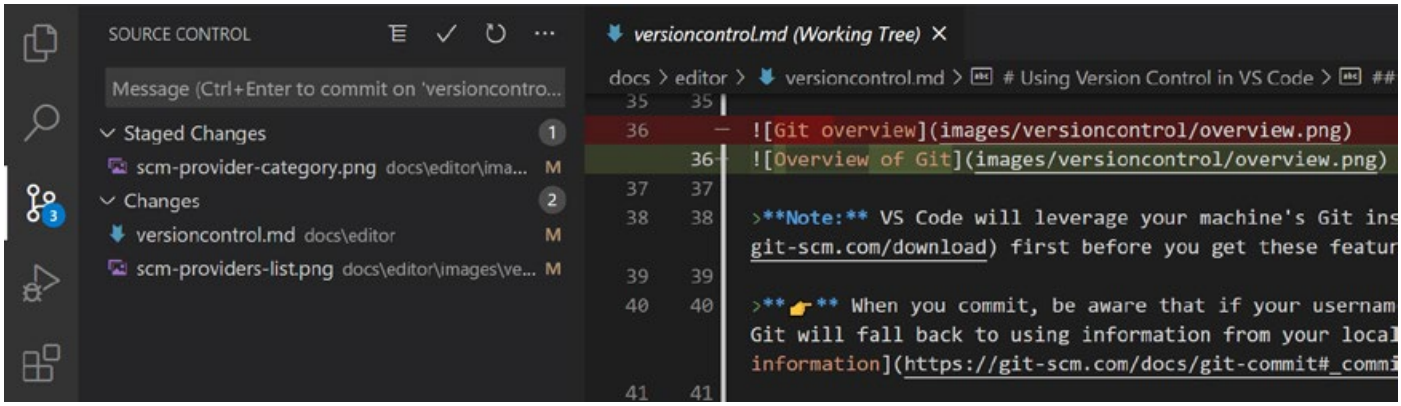
Fork

GitKraken: This offers a more visual and accessible way of working with Git with an intuitive UI as well as the flexibility to switch between a GUI or a CLI terminal.



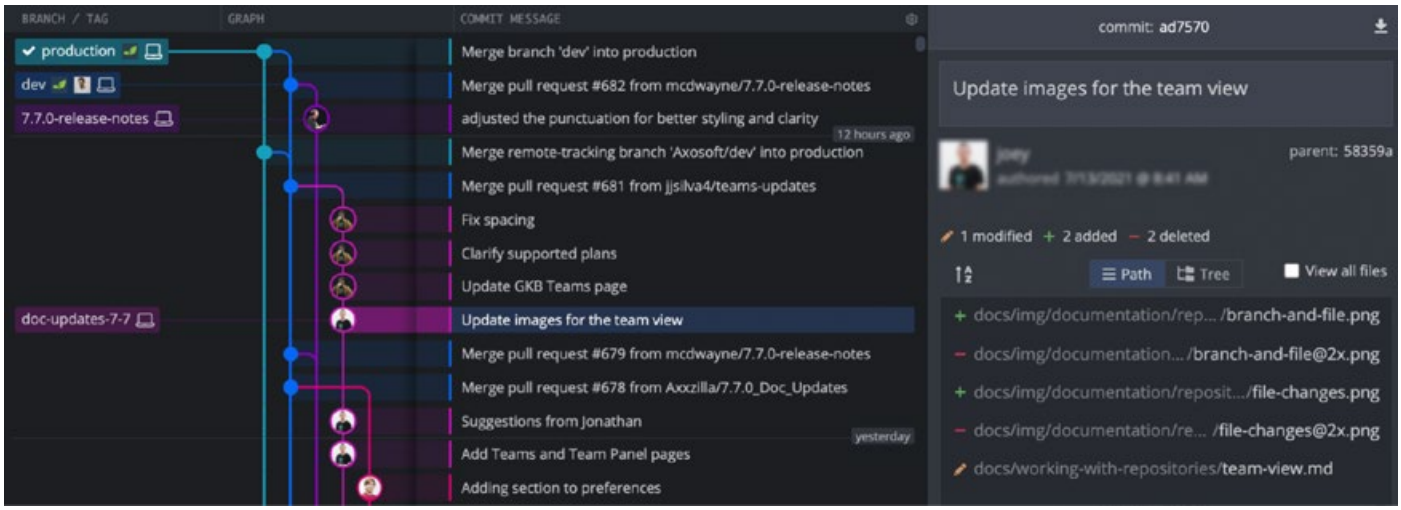
GitKraken

Microsoft Visual Studio Code: VS Code has source control integration built in, and with all the extensions available, you can avoid using a separate program altogether.



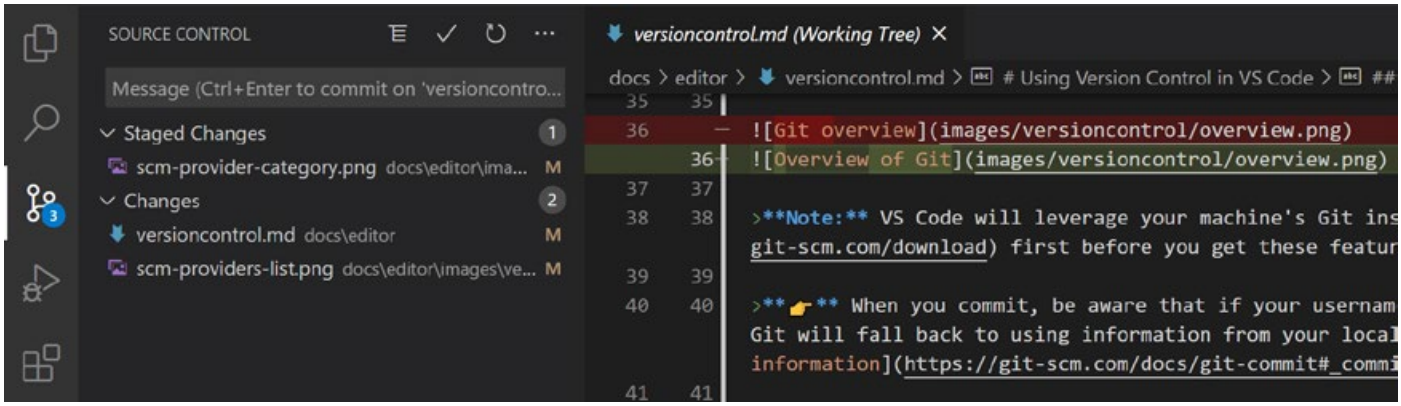
Visual Studio Code

GitKraken：这提供了一种更直观、更易于访问的 Git 使用方式，具有直观的 UI，并且可以灵活地在 GUI 或 CLI 终端之间切换。



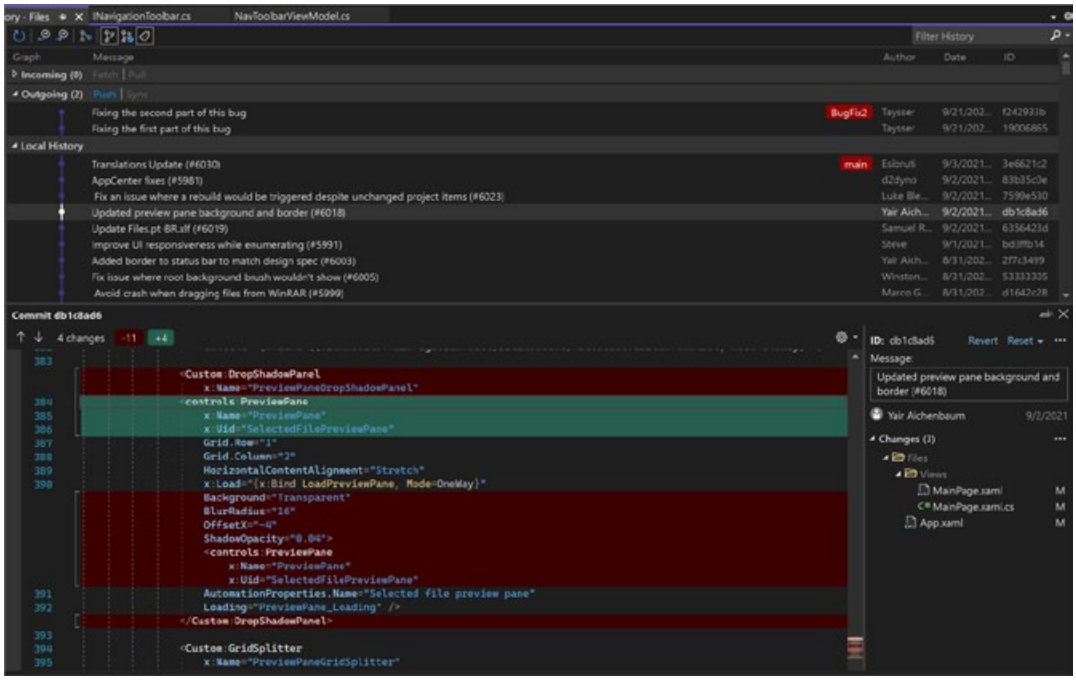
GitKraken

Microsoft Visual Studio Code：VS Code 内置了源代码控制集成，并且具有所有可用的扩展，您可以完全避免使用单独的程序。



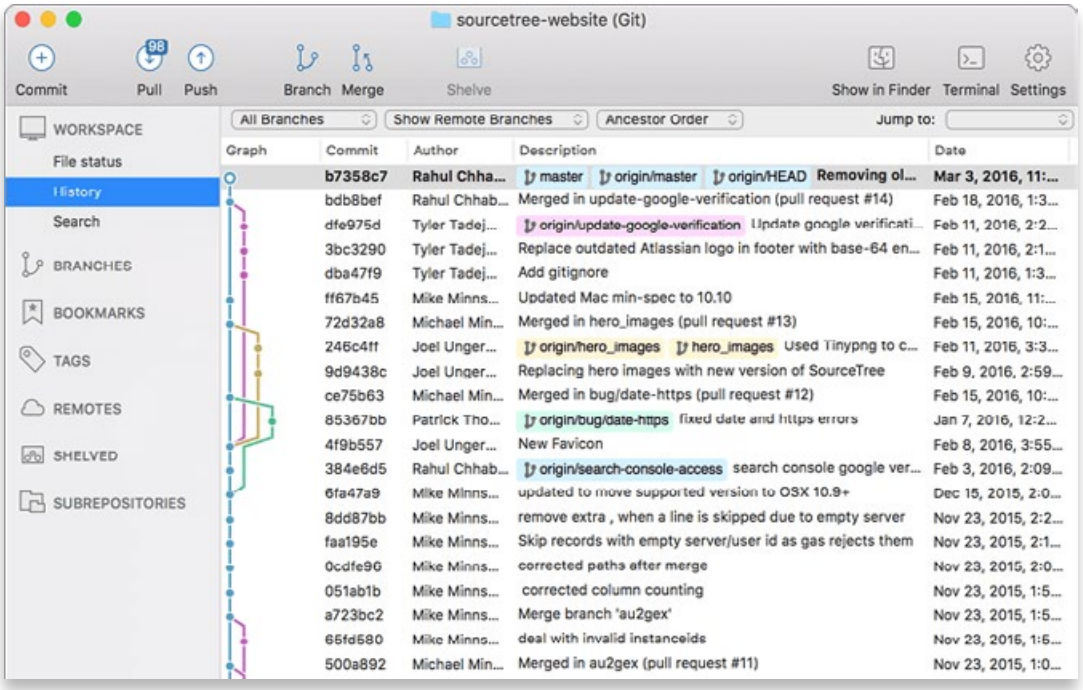
Visual Studio 代码

Microsoft Visual Studio: As with VS Code, Visual Studio also has Git controls built in and includes a GitHub extension.



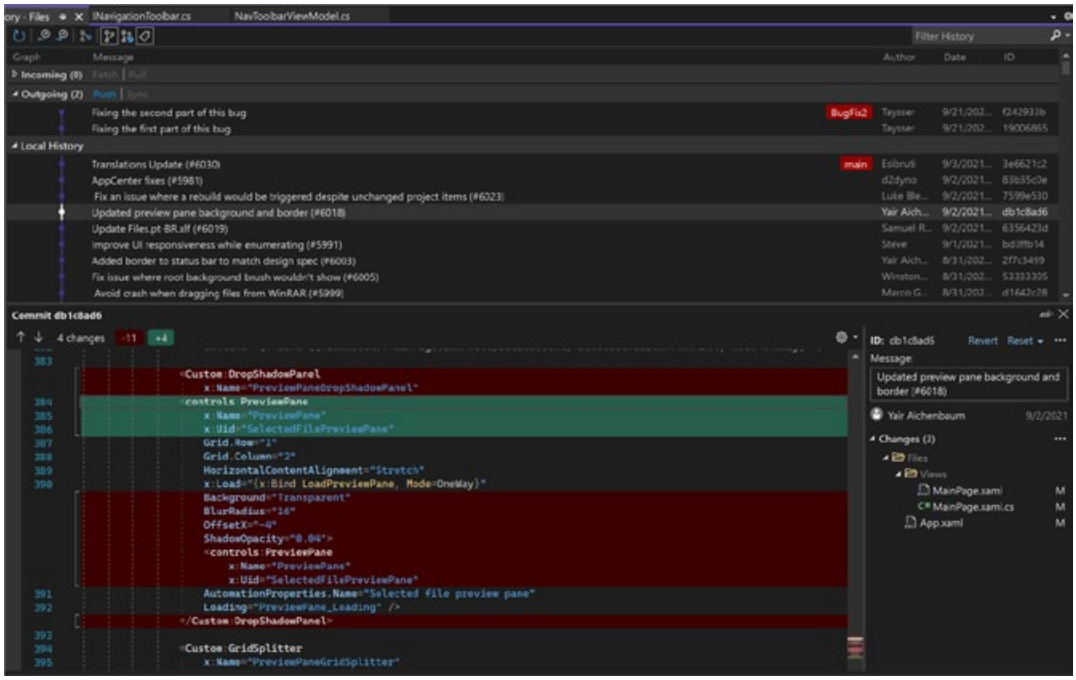
Visual Studio

SourceTree: Part of the Atlassian product group SourceTree is a free Git client for Windows and Mac that can also help you visualize and manage your Git repositories easily.



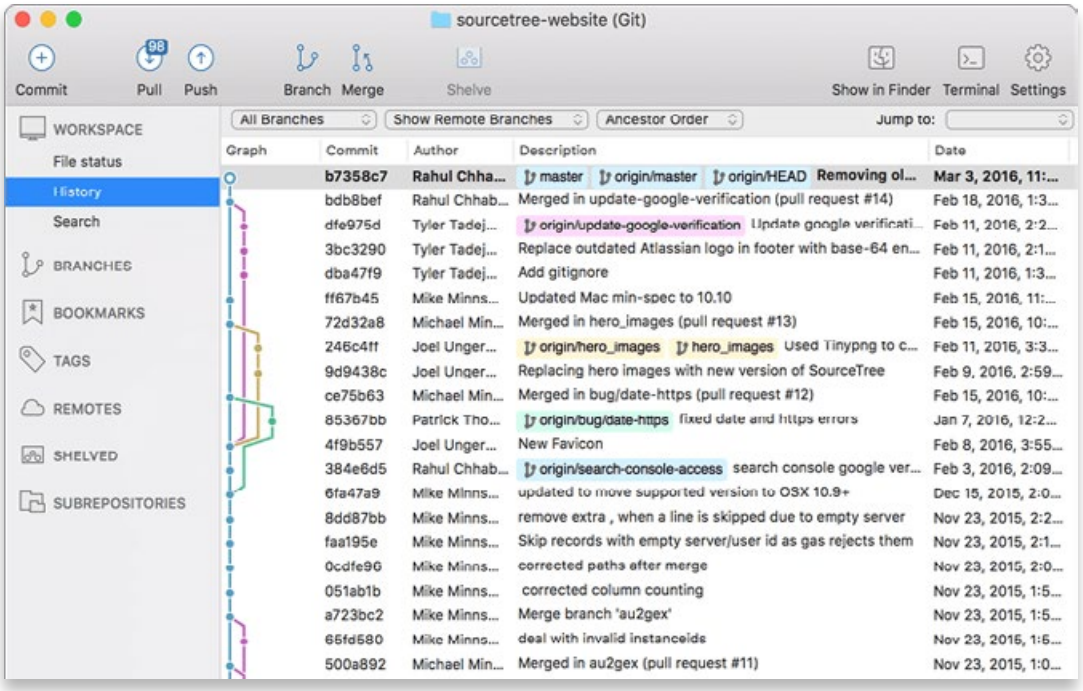
SourceTree

Microsoft Visual Studio：与 VS Code 一样，Visual Studio 也内置了 Git 控件，并包含一个 GitHub 扩展.



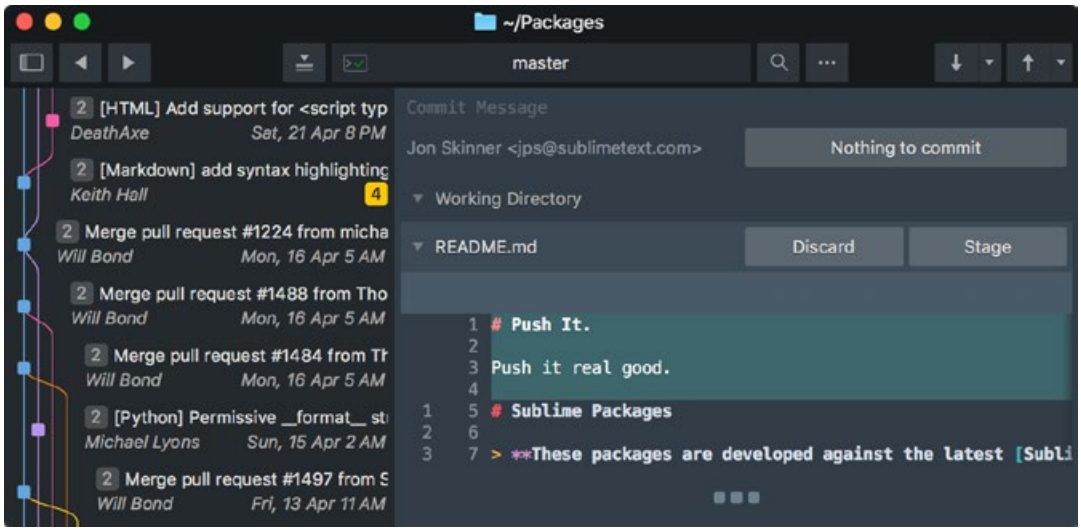
Visual Studio的

SourceTree：SourceTree 是 Atlassian 产品组的一部分，是一款适用于 Windows 和 Mac 的免费 Git 客户端，还可以帮助您轻松可视化和管理 Git 存储库。



SourceTree （源树）

Sublime Merge: This system offers tools for speeding up code reviews with side-by-side diffs and syntax highlighting. It's a lightweight, high-performance client.



Sublime Merge

Git is generally considered strong in branching and merging capabilities, but it can't handle large binary files as effectively as other solutions on the market. Git Large File Storage (LFS) goes some way to rectifying this.

Since Git is a distributed client, the entire repository and complete history is on the developer's machine. This makes actions such as switching branches or reverting back to a point in history extremely quick. If you're working on a large project with multiple features and release branches, a Git workflow can save countless hours.

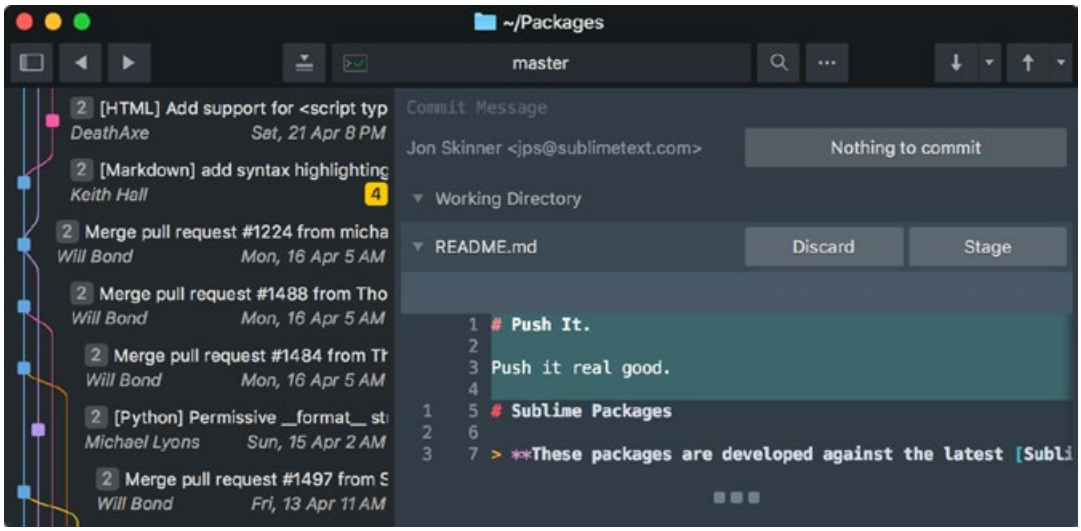
Unity has released their C# editor and engine code to the public on GitHub. This is incredibly useful when you need to know how some functions work or how to replicate a feature of the Editor inside your own project.

GitHub also has its own Git GUI, GitHub Desktop. When working in Unity, you can also use the GitHub for Unity package to bring the Git tools directly into the Unity Editor.

As mentioned, GitHub isn't the only hosting service available for your Git projects. You can also use Bitbucket (from Atlassian) or GitLab, which have many more DevOps features available to them, or one of the many other hosting services available.

See this talk from Unite Now 2020 on how to get started with Github, GitKraken and Unity.

Sublime Merge：该系统提供了通过并排差异和语法突出显示来加速代码审查的工具。它是一个轻量级、高性能的客户端。



崇高合并

Git 通常被认为在分支和合并功能方面很强大，但它无法像市场上的其他解决方案那样有效地处理大型二进制文件。Git Large File Storage （LFS） 在一定程度上解决了这个问题。

由于 Git 是分布式客户端，因此整个存储库和完整历史记录都在开发人员的计算机上。这使得切换分支或恢复到历史记录中的某个时间点等作变得非常快速。如果您正在处理具有多个功能和发布分支的大型项目，Git 工作流程可以节省无数小时。

Unity 已在 GitHub. 上向公众发布了他们的 C# 编辑器和引擎代码，当您需要了解某些函数的工作原理或如何在自己的项目中复制编辑器的功能时，这非常有用。

GitHub 也有自己的 Git GUI，即 GitHub Desktop。在 Unity 中工作时，您还可以使用 GitHub for Unity 包将 Git 工具直接引入 Unity 编辑器。

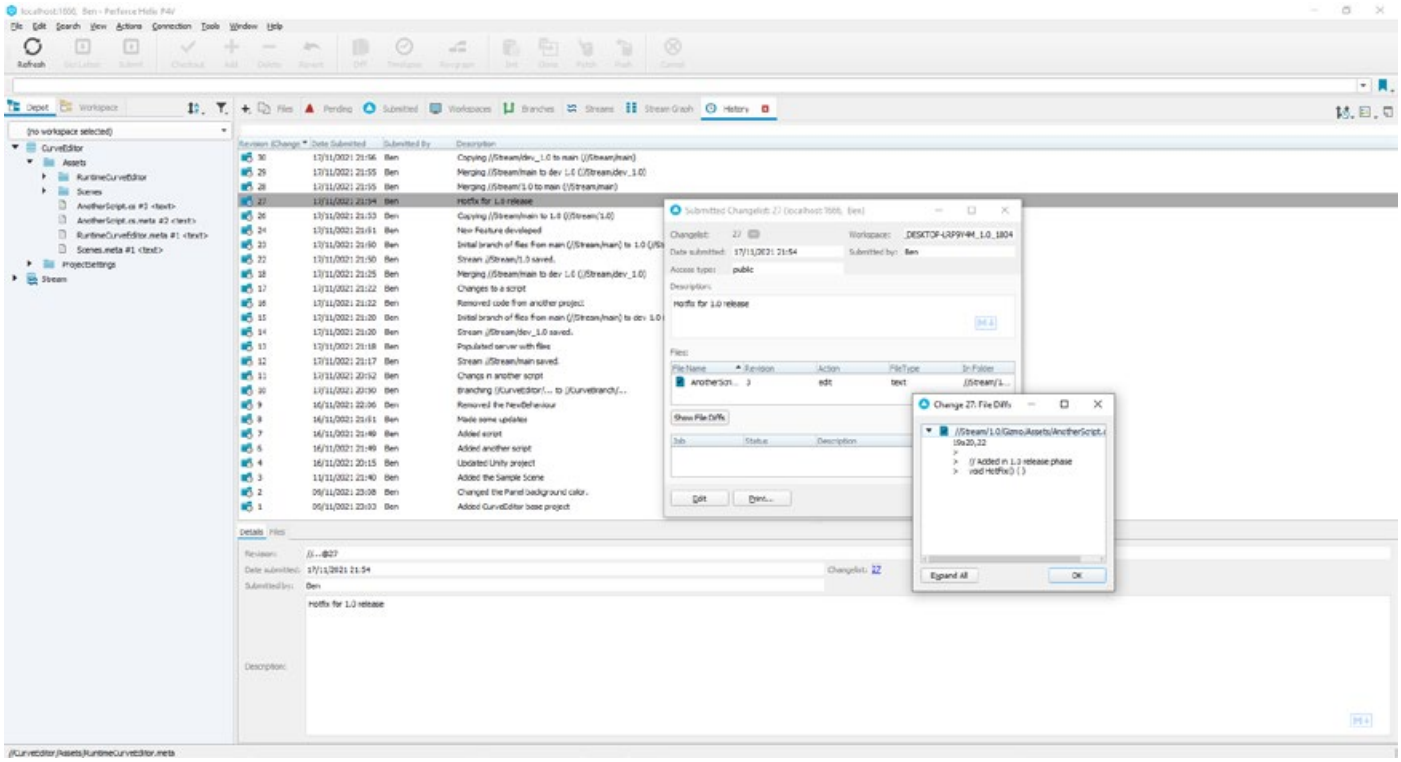
如前所述，GitHub 并不是可用于 Git 项目的唯一托管服务。您还可以使用 Bitbucket（来自 Atlassian）或 GitLab，它们有更多的 DevOps 功能可供他们使用，或者许多其他可用的托管服务之一。

请参阅 Unite Now 2020 的此演讲，了解如何开始使用 Github、GitKraken 和 Unity。

Perforce (Helix Core)

Helix Core is an enterprise-level version control system generally used by large game studios. These studios use Perforce because it features centralized repos that are most often hosted on their own servers. It does not feature visual repos, so its adoption might be more challenging for non-technical developers, but in larger studios there will be DevOps and Release Engineers to help manage the code base. Plus, as an enterprise solution, it includes a global support team.

Helix Core can also be used by small teams and you can still deploy to the cloud through solutions like Amazon AWS or Microsoft Azure.



Helix Core P4V interface

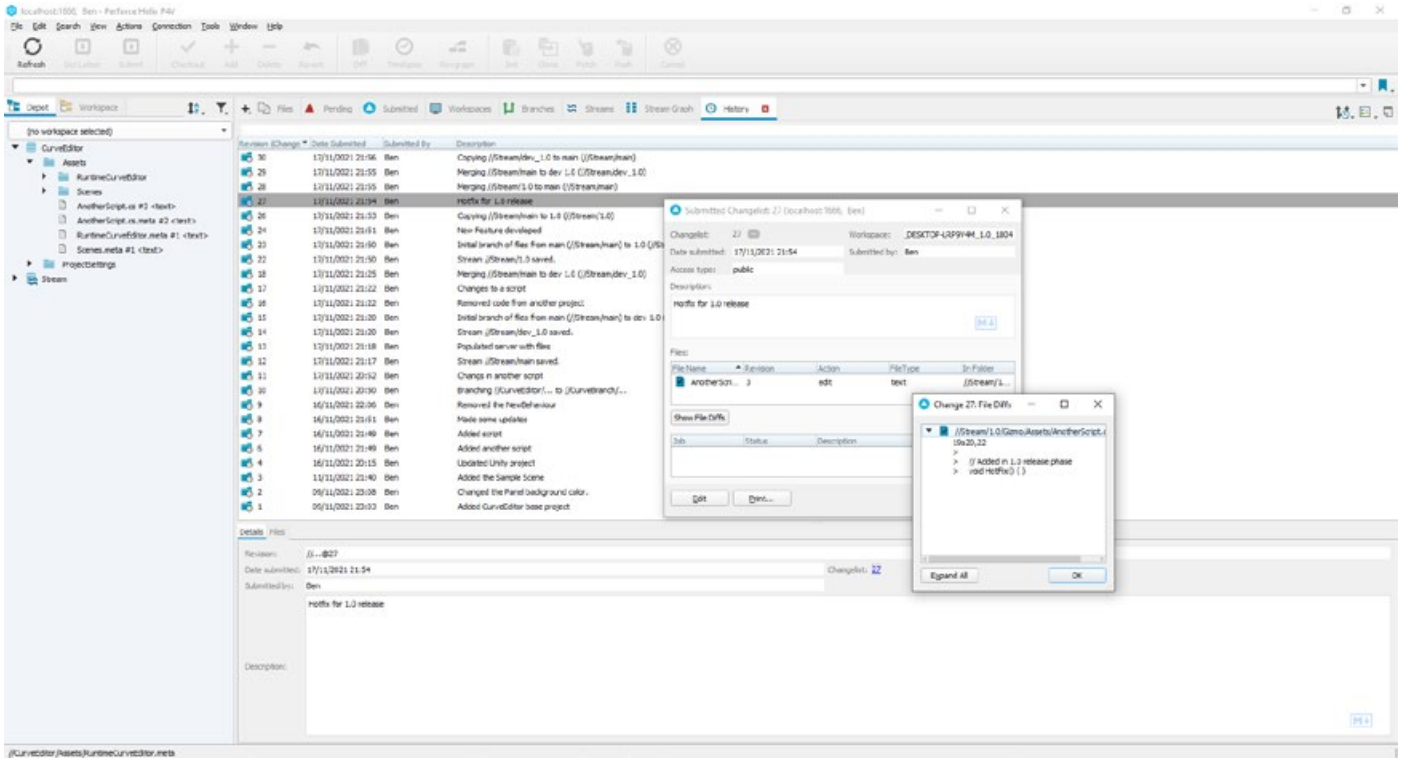
Helix Core can handle large files and there's a built-in Unity Editor integration that's covered in a later section.

To learn more about integrating your Unity workflow with Helix Core, check out this Perforce blog post.

Perforce（螺旋核心）

Helix Core 是大型游戏工作室通常使用的企业级版本控制系统。这些工作室使用 Perforce，因为它具有集中式存储库，这些存储库通常托管在他们自己的服务器上。它没有可视化存储库，因此对于非技术开发人员来说，采用它可能更具挑战性，但在大型工作室中，将有 DevOps 和发布工程师帮助管理代码库。此外，作为企业解决方案，它包括一个全球支持团队。

Helix Core 也可以由小型团队使用，您仍然可以通过 Amazon AWS 或 Microsoft Azure. 等解决方案部署到云中



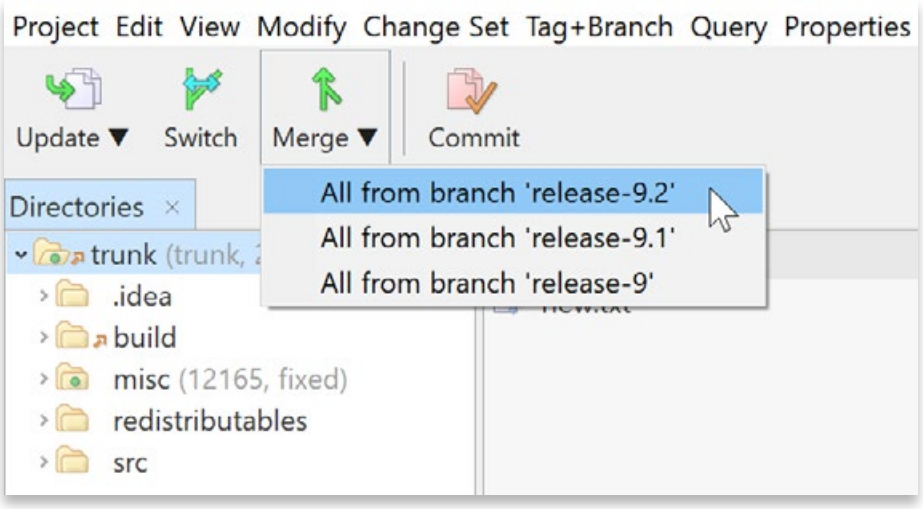
Helix Core P4V 接口

Helix Core 可以处理大型文件，并且有一个内置的 Unity 编辑器集成，将在后面的章节. 中介绍。

要了解有关将 Unity 工作流与 Helix Core 集成的更多信息，请查看此 Perforce 博客文章.

Apache Subversion

Like Git, Apache Subversion (known as SVN) is a free and open-source version control system. Unlike Git, it's a centralized VCS that can handle large binary files. However, it's still a command line system that requires one of the many third-party GUI clients to be a bit more user friendly. One such client is SmartSVN.



SmartSVN GUI

Before Git LFS, SVN was a popular choice when working in Unity. As a centralized solution, it was simpler to work with and, as mentioned, better for working with large files. Where SVN falls behind the other tools is when you start to use branches and need to merge between them. Merging in SVN has many pains, especially when it comes to conflicts – or even false conflicts – between files. A merge operation that would take minutes in another VCS may take hours to go through manually in SVN.

For more information on setting up Unity to work with SVN, check out the Unity documentation.

Unity Version Control

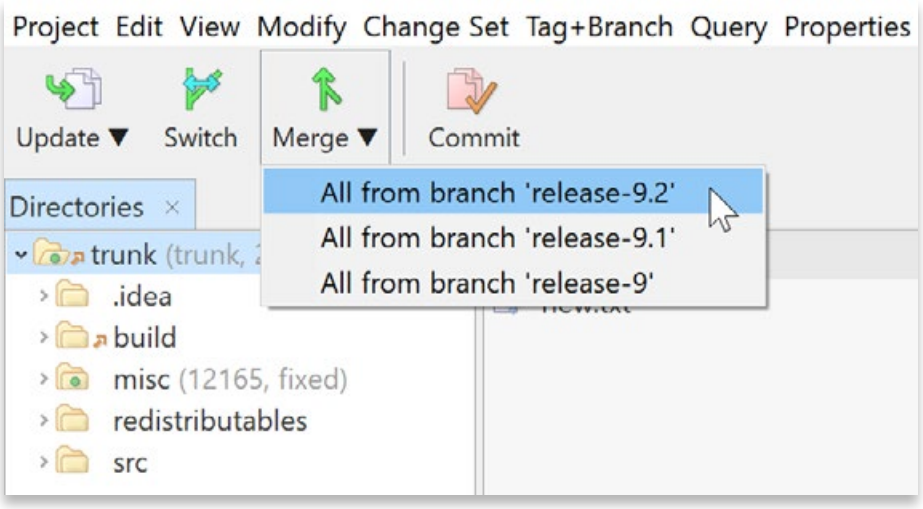
Unity Version Control (UVCS) is a flexible version control system with unique interfaces to support programmers and artists alike. It excels at handling large repos and binary files, and as both a file-based and changeset-based solution, it gives you the capability to download only the specific files you're working on, rather than the entire project build.

There are three ways to access UVCS: via multiple applications and repositories through the UVCS desktop client, by adding it to your projects through the Unity Hub, or accessing the repository on Unity cloud via your web browser. See the section "Get started with UVCS in Unity 6" for more information on how to set it up.

Small teams of up to three users can sign up for free and at the time of writing, get up to 5GB of cloud storage, along with access to the Version Control software, including Gluon.

亚派克子版本

与 Git 一样，Apache Subversion（称为 SVN）是一个免费的开源版本控制系统。与 Git 不同，它是一个集中式 VCS，可以处理大型二进制文件。但是，它仍然是一个命令行系统，需要众多第三方 GUI 客户端之一更加用户友好。SmartSVN. 就是这样一个客户端



SmartSVN 图形用户界面

在 Git LFS 之前，SVN 是在 Unity 中工作时的常见选择。作为集中式解决方案，它更易于使用，并且如前所述，更适合处理大文件。SVN 落后于其他工具的地方是当你开始使用分支并需要在它们之间进行合并时。在 SVN 中合并有很多痛苦，尤其是当涉及到文件之间的冲突 - 甚至是错误的冲突 - 时。在另一个 VCS 中需要几分钟的合并作可能需要数小时才能在 SVN 中手动完成。

有关设置 Unity 以使用 SVN 的更多信息，请查看 Unity 文档。

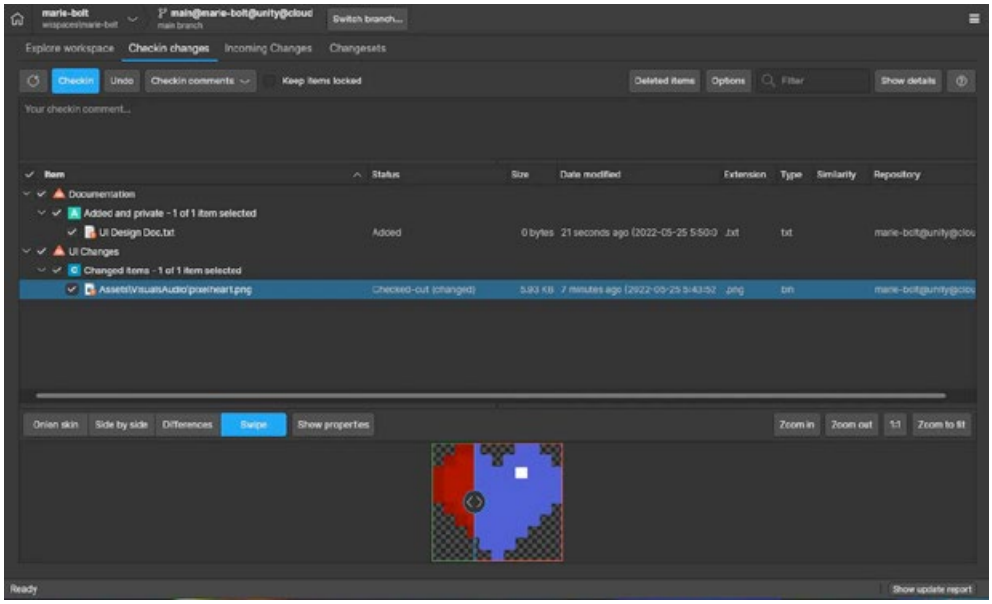
Unity 版本控制

Unity 版本控制 （UVCS） 是一个灵活的版本控制系统，具有独特的界面，可为程序员和艺术家提供支持。它擅长处理大型存储库和二进制文件，并且作为基于文件和基于变更集的解决方案，它使您能够仅下载您正在处理的特定文件，而不是整个项目构建。

有三种方法可以访问 UVCS：通过 UVCS 桌面客户端通过多个应用程序和存储库，通过 Unity Hub 将其添加到项目中，或通过 Web 浏览器访问 Unity 云上的存储库。有关如何设置的更多信息，请参阅“Unity 6 中的 UVCS 入门”部分。

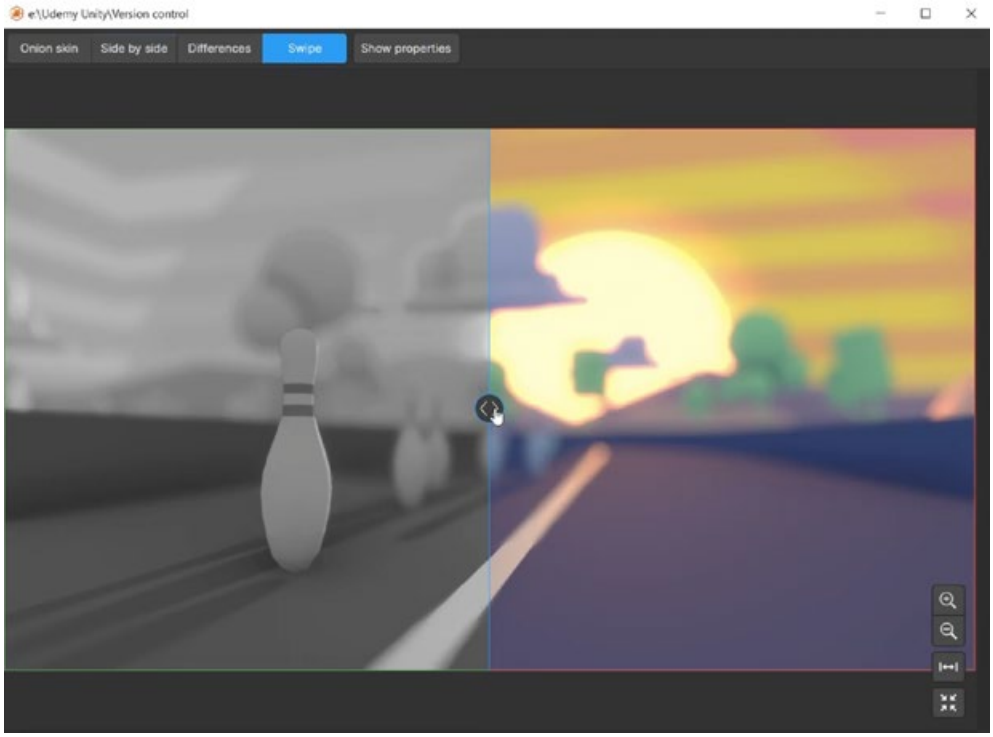
最多三个用户可以免费注册，在撰写本文时，最多可获得 5GB 的云存储空间，以及版本控制软件的访问权限，包括 Gluon.

Gluon is a slimline client designed to let artists work like artists, not programmers. It allows you to pick only the files that you're going to work on and check them out from the server, locking them from being modified by anyone else. Once you complete your work, you check the files back in. The Gluon GUI removes the more complex concepts that work better for programmers than for other, less technical users.



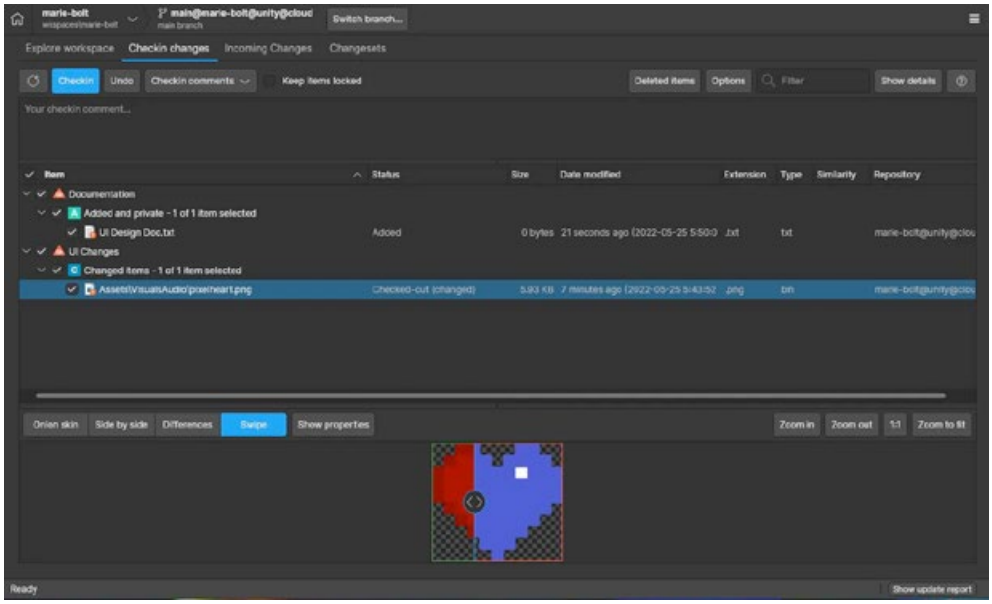
Gluon offers a workflow especially designed for artists, making it easy to preview files and history as well as to check in changes.

For artists, both UVCS and Gluon include ways to diff images. The image diff tool lets you compare two versions of the same file visually, a feature that many other systems don't offer.



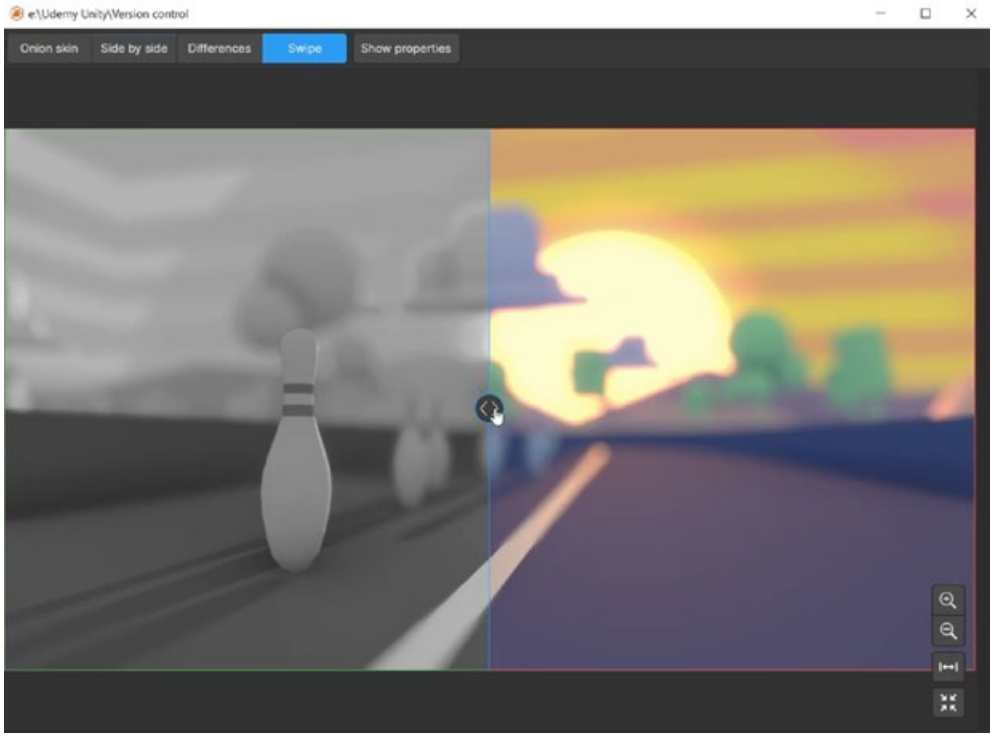
A diff viewer in Swipe mode: Go from one version to the other by dragging the swipe control, a useful feature for tracking image evolution

Gluon 是一个精简的客户端，旨在让艺术家像艺术家一样工作，而不是程序员。它允许您只选择要处理的文件并从服务器中签出它们，从而锁定它们不被其他任何人修改。完成工作后，将文件签回。Gluon GUI 删除了更复杂的概念，这些概念更适合程序员，而不是其他技术含量较低的用户。



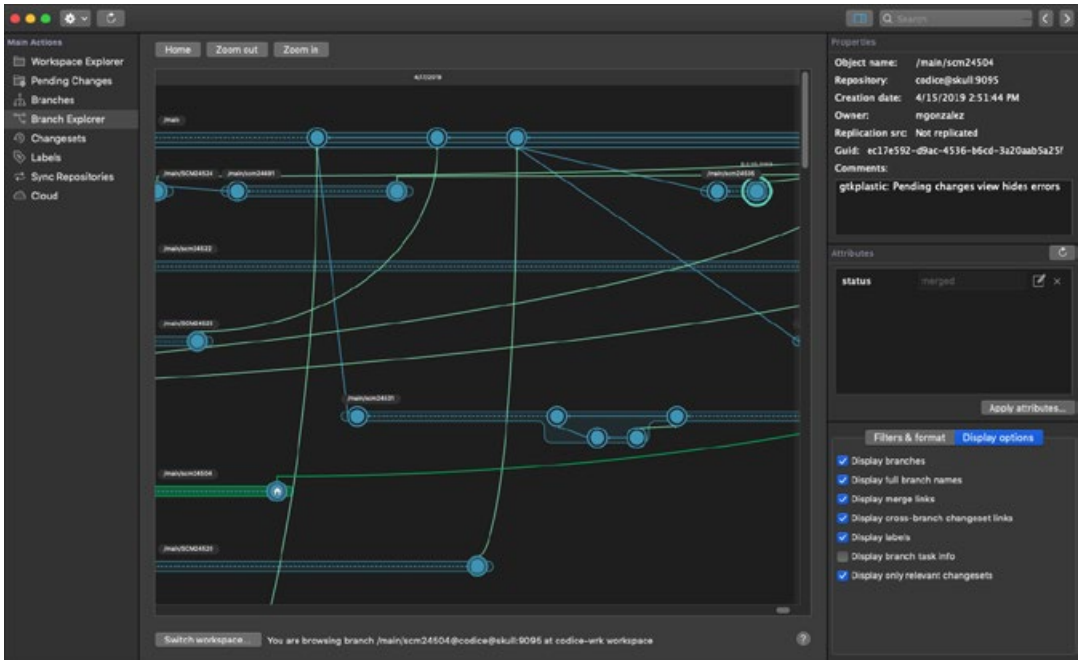
Gluon 提供了专为艺术家设计的工作流程，使预览文件和历史记录以及签入更改变得容易。

对于艺术家，UVCS 和 Gluon 都包含对图像进行比较的方法。图像比较工具允许您直观地比较同一文件的两个版本，这是许多其他系统不提供的功能。



Swipe m 中的差异查看器 ode：通过拖动滑动控制从一个版本转到另一个版本，这是一个有用的功能。 R 跟踪图像演变

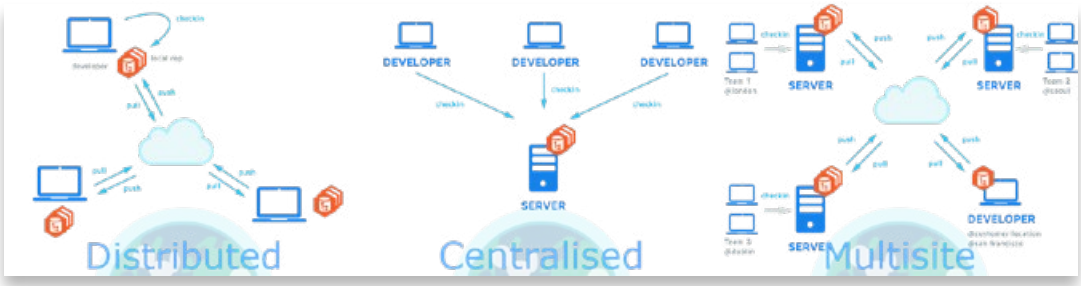
The standard UVCS GUI client has all the features they would be looking for and more for the programming team. The GUI has an interactive visual Branch Explorer that shows the true relationships of all the branches in a project. There is also a built-in Code Review system that you can use to request the review of your work from other groups or team members.



The branch explorer visualizes the merge structure of the project. It evolves horizontally from left to right.

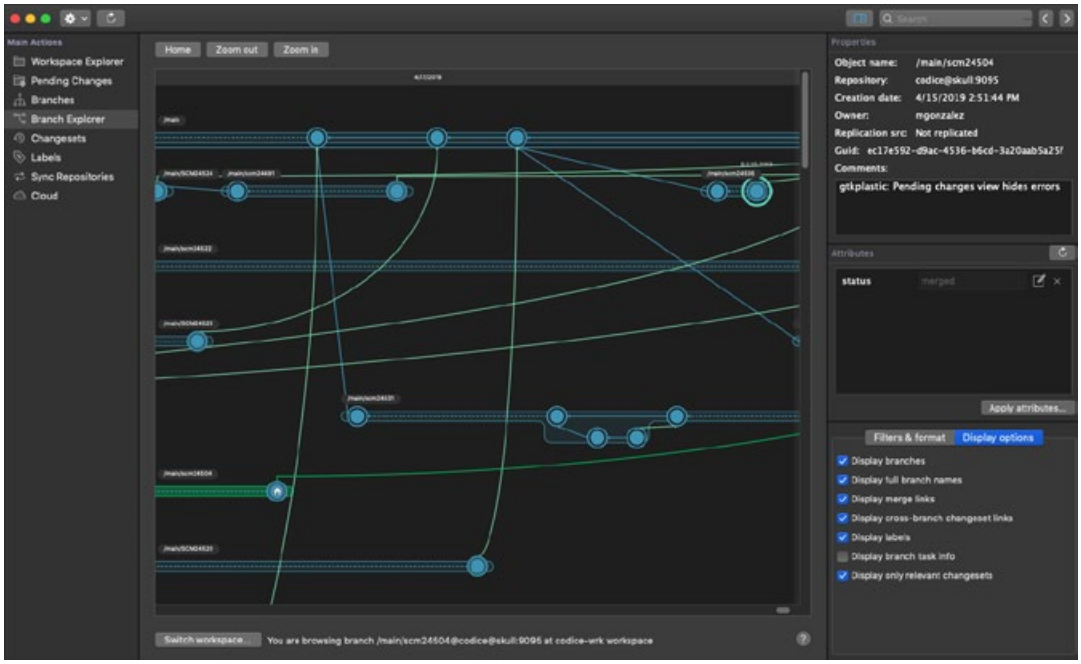
One of the key strengths of UVCS is that it has the flexibility to be configured for a distributed or centralized workflow. In fully distributed mode, developers work with a repository on their local machine, checking in, branching, and merging with ease. Developers will then push and pull changes to the server to share them when ready.

In centralized mode, users check out and check in their changes directly to the server so everyone is working on the latest changes. However, as development teams have grown into global organizations, everyone communicating with one central server isn't always beneficial. UVCS can also be configured to work in a multi-site system. In this system, servers are set up at each site, so teams can check in to their local server, keeping their workflow fast and hard drives happy. Then, the distributed servers communicate with each other to a central or cloud server.



See the section Get started with UVCS in Unity 6, along with this Unity Learn quick-start tutorial, for the steps to setting up UVCS in Unity.

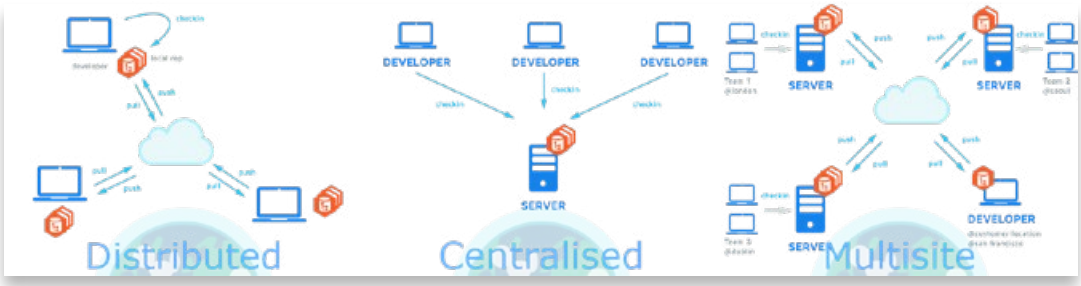
标准 UVCS GUI 客户端具有他们正在寻找的所有功能，并为编程团队提供更多功能。GUI 具有交互式可视化 Branch Explorer，可显示项目中所有分支的真实关系。还有一个内置的代码审查系统，您可以使用它来请求其他组或团队成员对您的工作进行审查。



分支资源管理器可视化项目的合并结构。它从左到右水平演变。

UVCS 的主要优势之一是它可以灵活地配置为分布式或集中式工作流程。在完全分布式模式下，开发人员在本地计算机上使用存储库，轻松签入、分支和合并。然后，开发人员会将更改推送和拉取到服务器，以便在准备就绪时共享它们。

在集中式模式下，用户将他们的更改直接签入并签入服务器，以便每个人都在处理最新的更改。然而，随着开发团队发展成为全球性组织，每个人都与一台中央服务器通信并不总是有益的。UVCS 也可以配置为在多站点系统中工作。在这个系统中，服务器在每个站点都设置好，因此团队可以签入他们的本地服务器，保持他们的工作流程快速和硬盘驱动器满意。然后，分布式服务器通过中央或云服务器相互通信。



有关在 Unity 中设置 UVCS 的步骤，请参阅 Unity 6 中的 UVCS 入门部分以及此 Unity Learn 快速入门教程。

VCS comparison

		UVCS	Git	Perforce	Subversion
Flexibility	Work centralized Checkin only, no push/ pull				
	Work distributed Push/pull + local repo				
Binaries	Large repos				
	Large files				
	Lock files to avoid merging				
GUI	Visualizes your repos (so you don't need a PhD in branching)				
	User-friendly GUIs				
	Artist-friendly GUI and workflow				
Workflow	Creates effective task branches				
Merge	Detects merges between branches				
	Provides diff and three-way merge tools				
	Tools help you understand the merge				
	Good merging renames, moved files, directories, refactors				

VCS 比较

		UVCS	Git	Perforce	Subversion
Flexibility	Work centralized Checkin only, no push/ pull				
	Work distributed Push/pull + local repo				
Binaries	Large repos				
	Large files				
	Lock files to avoid merging				
GUI	Visualizes your repos (so you don't need a PhD in branching)				
	User-friendly GUIs				
	Artist-friendly GUI and workflow				
Workflow	Creates effective task branches				
Merge	Detects merges between branches				
	Provides diff and three-way merge tools				
	Tools help you understand the merge				
	Good merging renames, moved files, directories, refactors				

Cloud	Can host repos in the cloud				
	Cloud hosting is good with large repos				
DIFF	Can diff code moved across files				
	Shows you the history of a method				
	Enterprise Support				

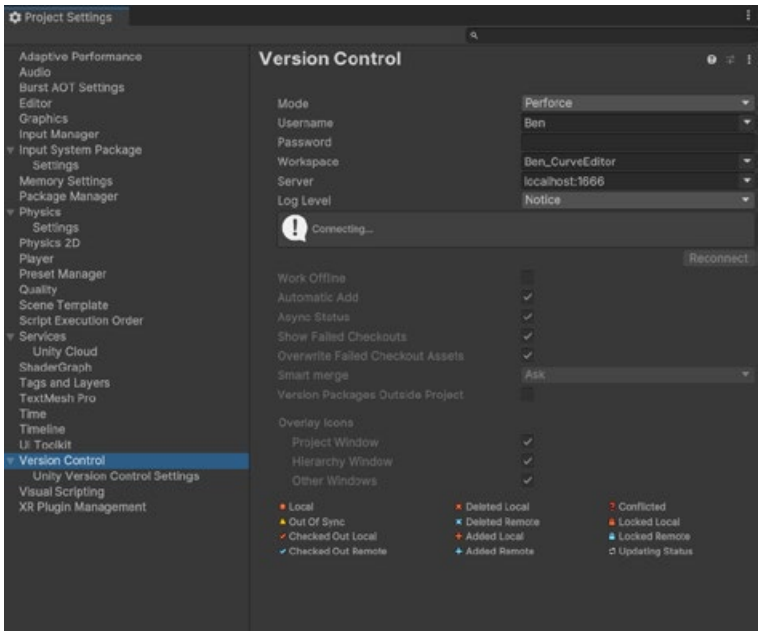
Setting up Unity to work with version control

This section will help you set up Unity to work with Git, Perforce, or UVCS, depending on your preference. By understanding some of the key workflows for each solution, you can make an informed decision about which system will best suit your team.

Editor project settings

Perforce Helix Core

Unity Editor integration is available with most version control systems, and Perforce Helix Core integration is built into the Editor. You only need to enable it via **Edit** > **Project Settings** > **Version Control**. Set the Mode to Perforce, and fill in the information of your workspace and server settings.



Setting up Perforce Helix Control for a project

Cloud	Can host repos in the cloud				
	Cloud hosting is good with large repos				
DIFF	Can diff code moved across files				
	Shows you the history of a method				
	Enterprise Support				

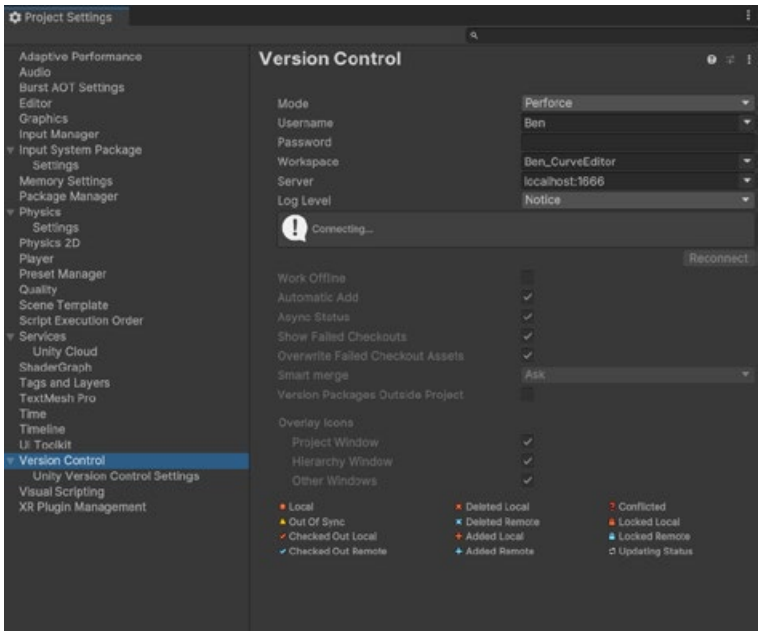
设置 Unity 以使用版本控制

本部分将帮助您根据自己的偏好设置 Unity 以使用 Git、Perforce 或 UVCS。通过了解每个解决方案的一些关键工作流程，您可以就哪种系统最适合您的团队做出明智的决定。

Editor 工程设置

Perforce 螺旋核心

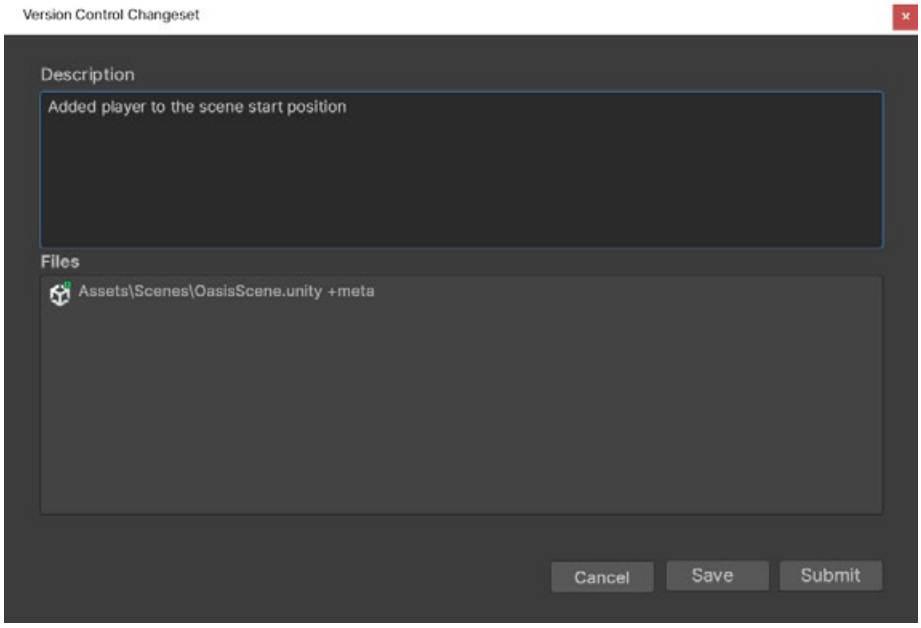
大多数版本控制系统都提供 Unity 编辑器集成，并且 Perforce Helix Core 集成内置于编辑器中。您只需通过编辑 > 项目设置 > 版本控制. 将模式设置为 Perforce，并填写工作区和服务器设置的信息.



为项目设置 Perforce Helix Control

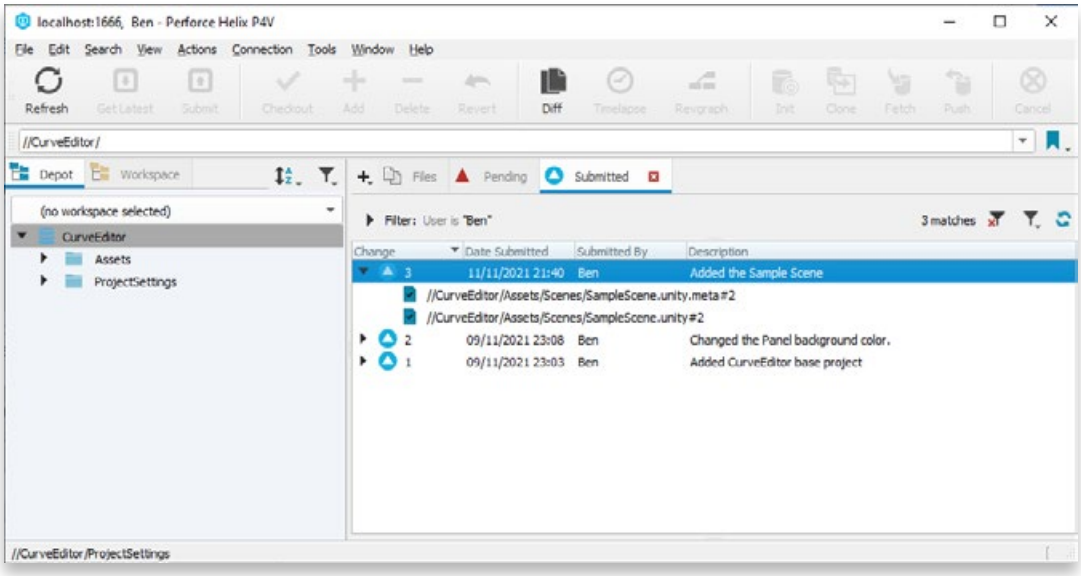
Once this is enabled, you will see that files are now considered “Under Version Control,” with the option to check them out.

Once a file is checked out, you can lock, unlock, submit, or revert the file. Choosing to submit will bring up a changeset dialog for you to add your commit message before submitting it into the repository.



Changeset dialog box

Use the Helix P4V interface to view the project history.

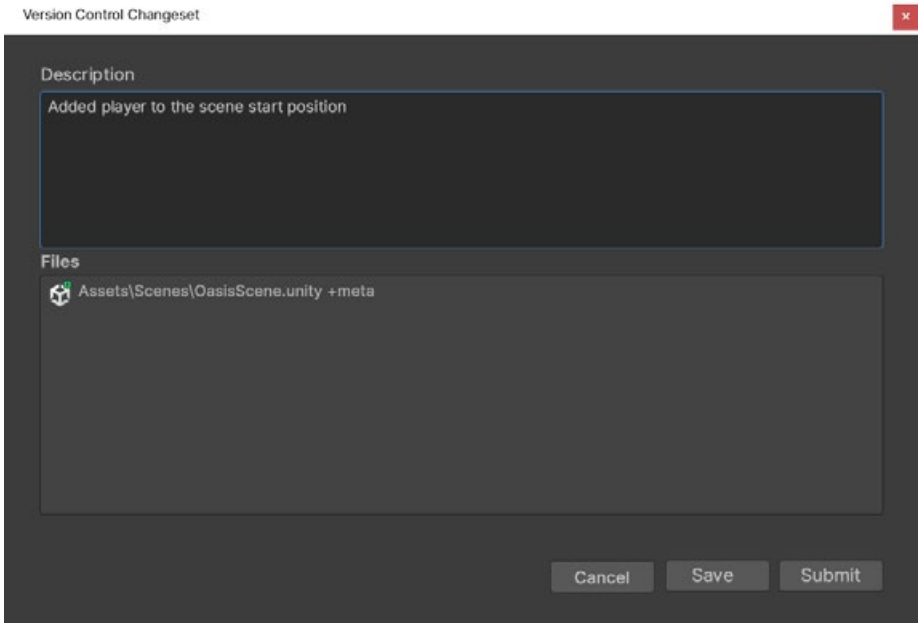


View the project history.

For more on getting started with Perforce Helix Core and Unity, check out the Perforce blog.

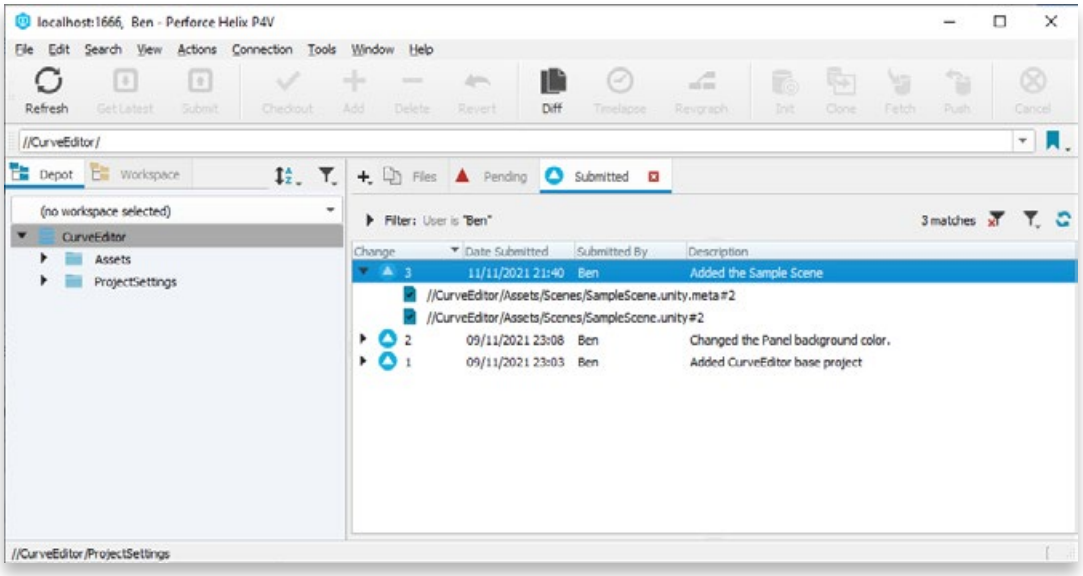
启用此功能后，您将看到文件现在被视为“Under Version Control”，并有签出选项。

签出文件后，您可以锁定、解锁、提交或还原文件。选择 submit 将弹出一个变更集对话框，供您在将提交消息提交到存储库之前添加提交消息。



“变更集”对话框

使用 Helix P4V 界面查看项目历史记录。



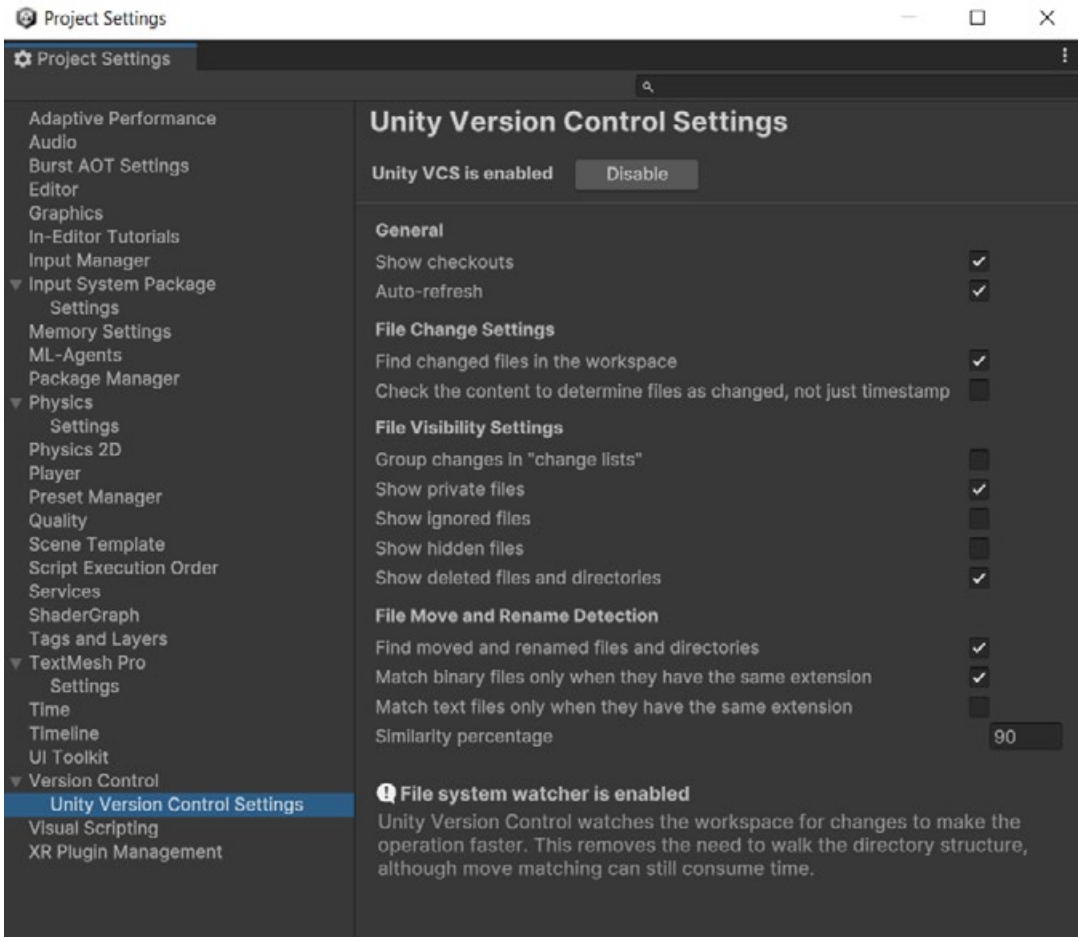
查看项目历史记录。

有关开始使用 Perforce Helix Core 和 Unity 的更多信息，请查看 Perforce 博客.

UVCS

UVCS is integrated with the Unity Editor, making it easy to get started and simplify workflows. When you create a new Unity project in the Hub, you can immediately integrate Unity VCS by selecting the **Use Unity Version Control** checkbox.

To connect UVCS to an existing project in the Editor, open **Window > Unity Version Control.** The tab will appear in the Project window.



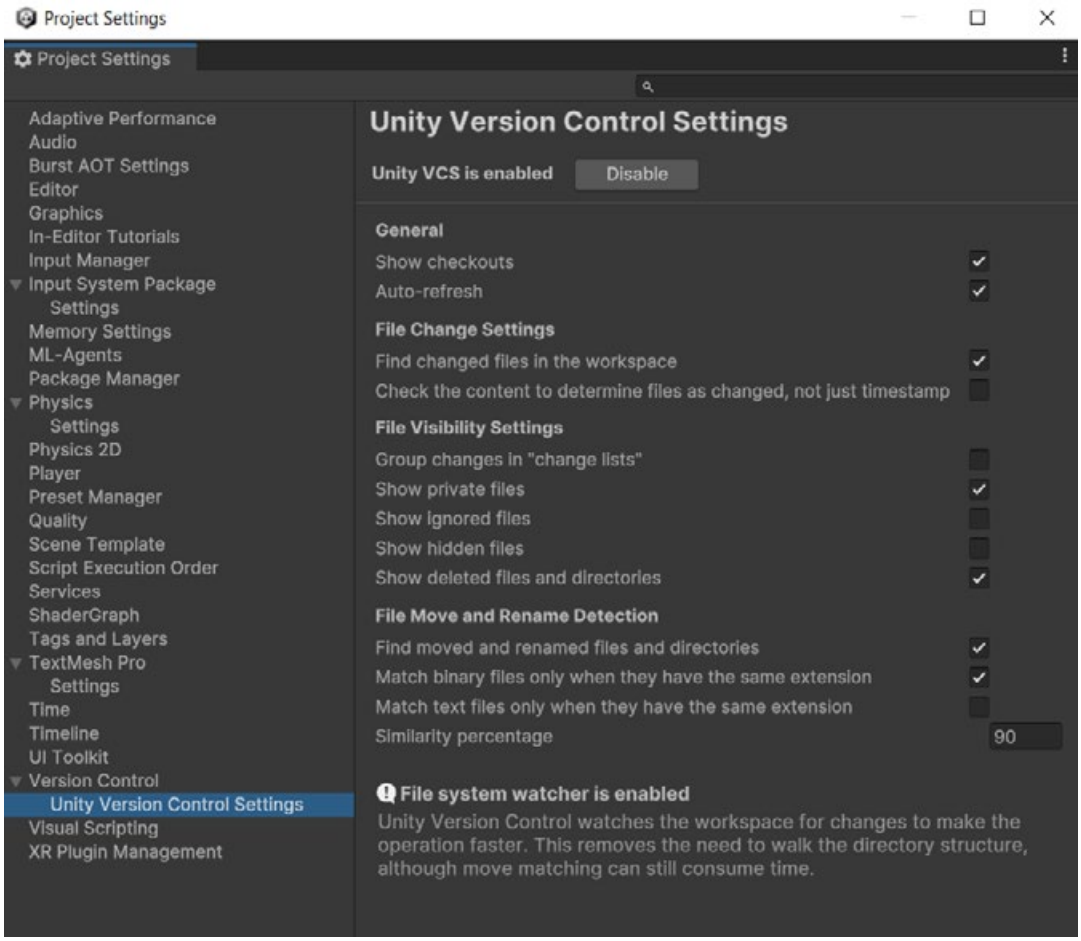
UVCS in the Project Settings

UVCS will feel familiar to Unity users, and simplifies workflows by eliminating the need for an additional client. Files can be added, checked out, reverted, checked in, or submitted, directly from the Editor.

UVCS 抗体

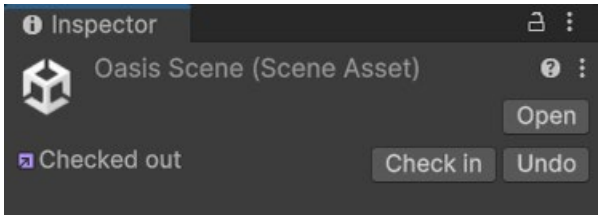
UVCS 与 Unity 编辑器集成，便于入门和简化工作流程。在 Hub 中创建新的 Unity 项目时，可以通过选中 Use Unity Version Control 复选框立即集成 Unity VCS。

要将 UVCS 连接到 Editor 中的现有项目，请打开 Window **>** Unity Version Control。 该选项卡将显示在 Project 窗口中。

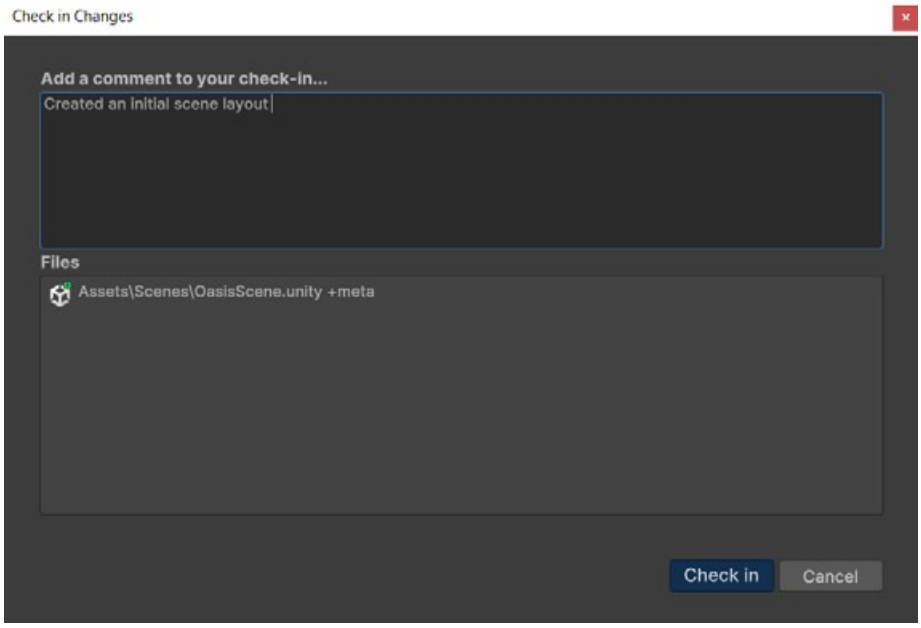


Project Settings 中的 UVCS

UVCS 对 Unity 用户来说会很熟悉，并且无需额外的客户端，从而简化了工作流程。可以直接从 Editor 添加、签出、还原、签入或提交文件。

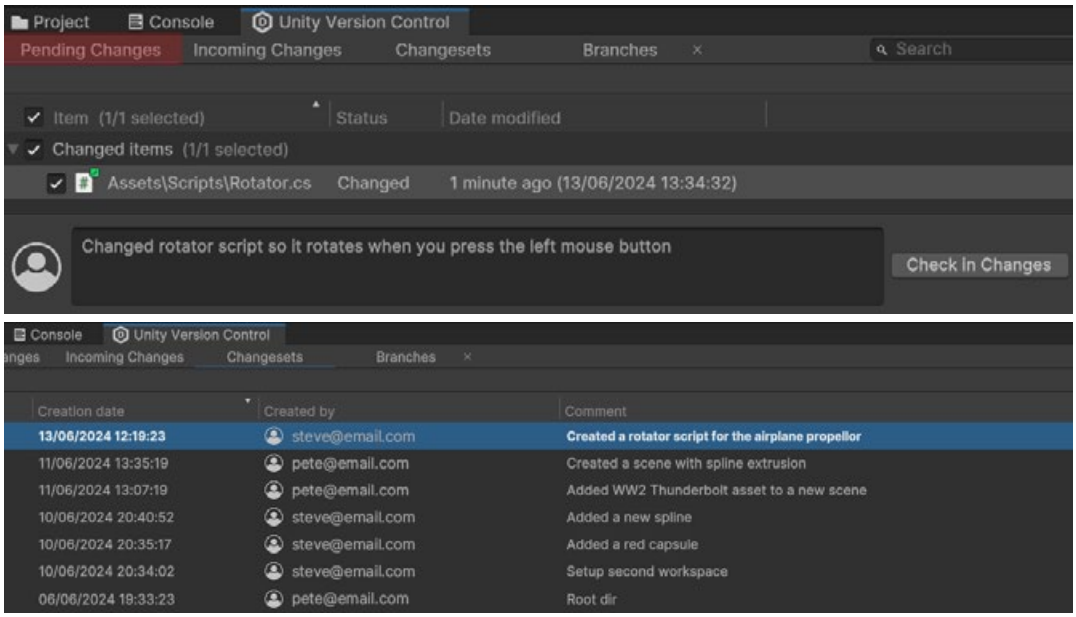


Working with files in UVCS from the Unity Editor

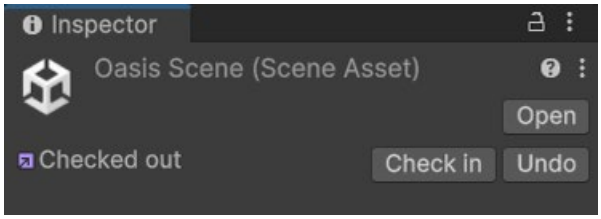


Checking in a file

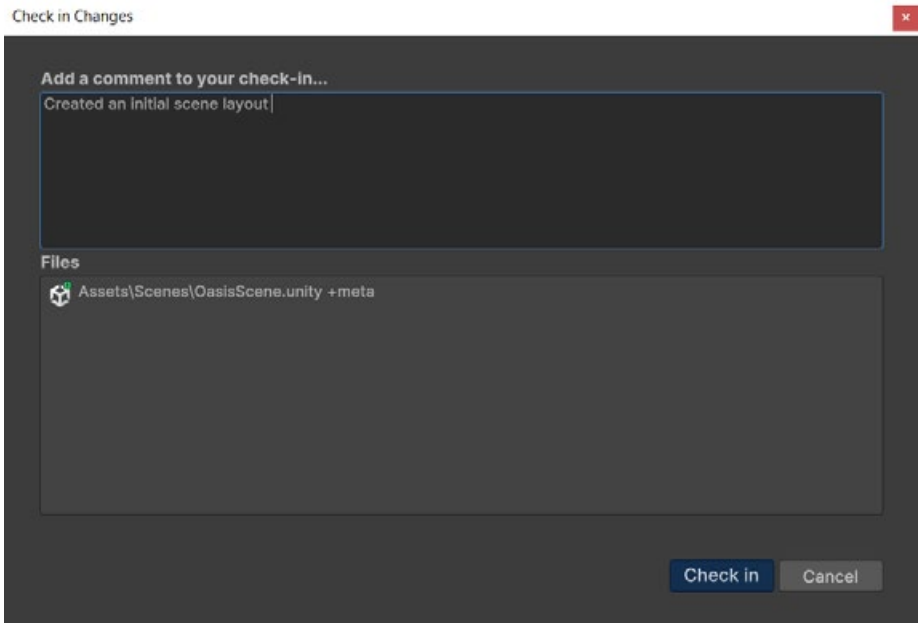
UVCS also has the advantage of having a Changesets tab available in the Unity Editor via **Window** > **Unity Version Control**.



Pending changes and Changesets tabs

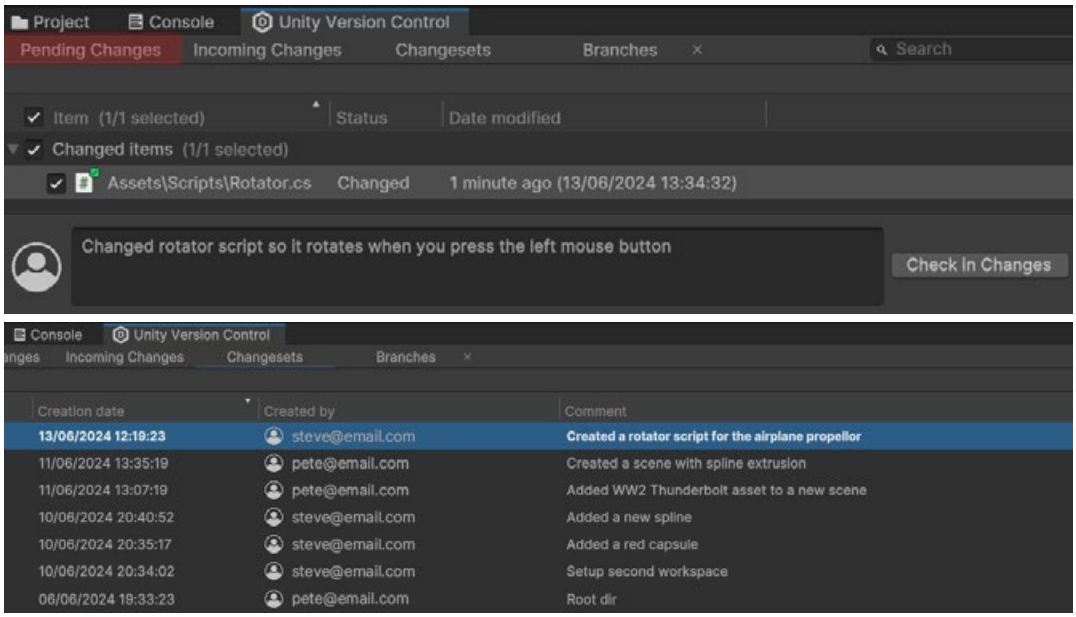


从 Unity Editor 处理 UVCS 中的文件



签入文件

UVCS 还有一个优势，即通过窗口 > 在 Unity 编辑器中提供 Changesets 选项卡Unity Version Control.



Pending changes 和 Changesets 选项卡

For more information on setting up Version Control in Unity, check out the documentation.

Git and other solutions

For all other VCS, open the **Edit** > **Project Settings** > **Version Control** window, and select Visible Meta Files from the dropdown menu. There are no other options here, but meta files must be visible in order for version control systems to detect them (see previous chapter on Meta Files).

What to ignore

When working with a Unity project, or any project for that matter, only files that cannot be generated should be placed under version control.

For Unity projects, that means only files in the Assets and Project Settings folders should be committed to your repository. Unity can automatically recreate all the other folders. Under no circumstance should you commit the Library folder, since this folder can get very large and Unity will recreate it when launching the Editor if it doesn't exist.

— UVCS automatically selects the appropriate folders and files to place under version control when set up from the Unity Editor. There is a list that is saved in the `ignore.conf` file at the root of the project that describes which files are ignored. To learn more about setting up the "ignore.conf" file, check out this blog post.

— With Perforce, you need to explicitly add the Assets and Project Settings folders to your depot.

— Git requires a **.gitignore** file to indicate what files should never be included. Depending on your Git GUI client, you can select a template when creating a repository, or this can be done through GitHub if you set the hosting up first. Alternatively, a template can be downloaded here.

You should also avoid committing things like .exe or .apk files. Additionally, gradle and xcode projects built from your Unity project should not be added to the repository.

A small exception to this rule is if you were to set up automated build processes for your Gradle or Xcode projects, but then they would be typically committed to a repository of their own.

有关在 Unity 中设置版本控制的更多信息，请查看文档.

Git 和其他解决方案

对于所有其他版本控制系统，打开 Edit > Project Settings > Version Control 窗口，然后从下拉菜单中选择 Visible Meta Files。这里没有其他选项，但元文件必须可见，版本控制系统才能检测到它们（参见上一章关于 Meta Files）。

要忽略的内容

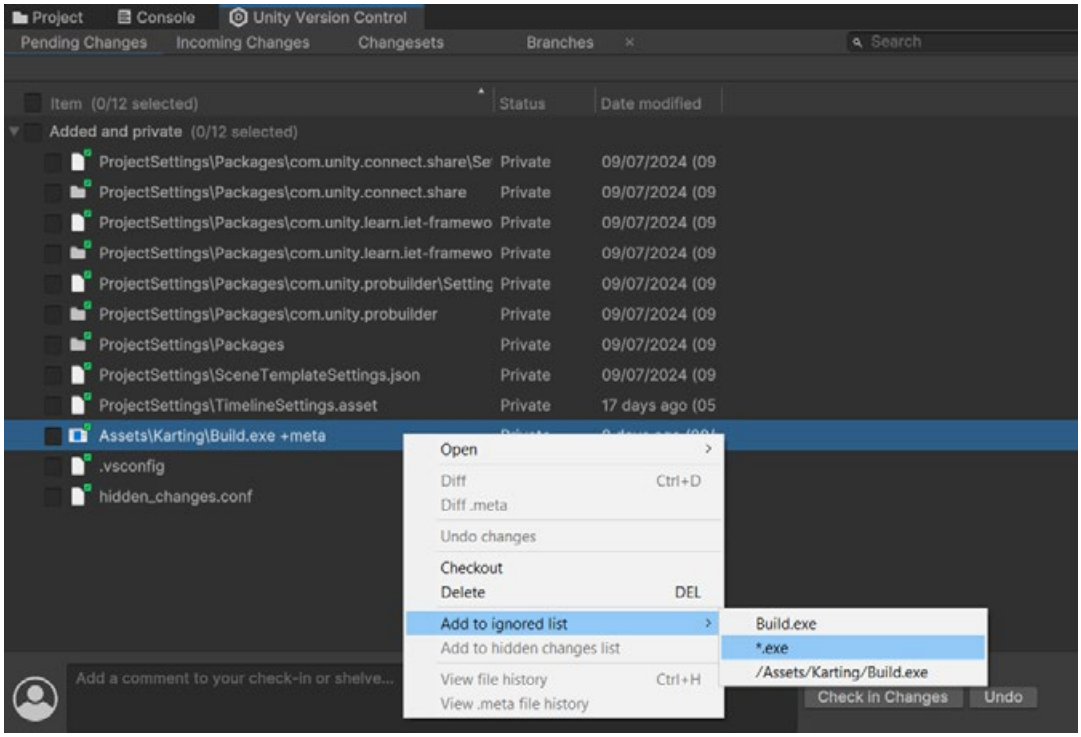
使用 Unity 项目或任何相关项目时，应仅将无法生成的文件置于版本控制之下。

对于 Unity 项目，这意味着只应将 Assets 和 Project Settings 文件夹中的文件提交到存储库。Unity 可以自动重新创建所有其他文件夹。在任何情况下都不应提交 Library 文件夹，因为此文件夹可能会变得非常大，如果不存在，Unity 将在启动 Editor 时重新创建它。

— 从 Unity 编辑器进行设置时，UVCS 会自动选择适当的文件夹和文件以置于版本控制之下。有一个列表保存在项目根目录下的 ìgnore.conf 文件中，用于描述要忽略哪些文件。要了解有关设置“ignore.conf”文件的更多信息，请查看此博客文章.— 使用 Perforce，您需要将 Assets 和 Project Settings 文件夹显式添加到您的 depot。— Git 需要一个 .gitignore 文件来指示哪些文件不应包含在内。根据您的 Git GUI 客户端，您可以在创建存储库时选择模板，或者如果您先设置托管，可以通过 GitHub 完成此作。或者，也可以在此处下载模板.

您还应该避免提交 .exe 或 .apk 文件等内容。此外，不应将从您的 Unity 项目构建的 gradle 和 xcode 项目添加到存储库中。

此规则的一个小例外是，如果您要为 Gradle 或 Xcode 项目设置自动化构建流程，但随后它们通常会提交到自己的存储库.



Files can be added to the ignored list directly from the Unity Editor when using UVCS.

Working with large files

Unity projects are made up of a lot more than just code. In fact, scripts can often be heavily outnumbered by other asset files in a Unity project. These assets are stored as binary files: Textures, models, Prefabs, audio clips, timelines, and so on. This results in two things:

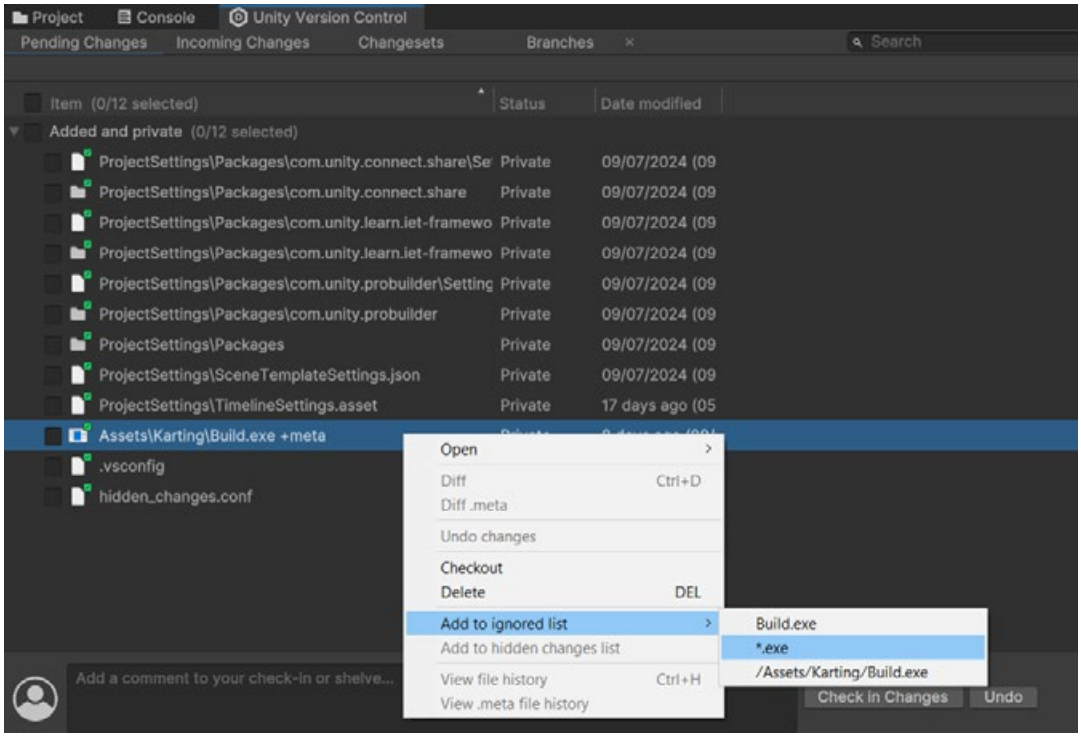
— They can be hard to compare between revisions.

— The diff cannot be described, so the whole file is written when a change is pushed to the repo.

Again, in a distributed environment, the entire project history is available on a user's local machine. Now if you have a history of large files that have had many changes over a long time, then you will have that many copies of the file stored on your machine. This can quickly consume a large portion of your hard drive space!

This is one of main reasons that historically, teams preferred a centralized workflow. This way, large historical versions of binary files would only live on a central server, with individual users only accessing the latest version on their machines.

Both Perforce and UVCS are centralized systems that can handle large files well. UVCS also gives you the option to work in a distributed pipeline, and large file sizes is the tradeoff that you need to consider when choosing between these options.



使用 UVCS 时，可以直接从 Unity Editor 将文件添加到忽略列表中。

使用大型文件

Unity 项目由许多代码组成。事实上，在 Unity 项目中，脚本的数量通常远远超过其他资源文件。这些资源存储为二进制文件：纹理、模型、预制件、音频剪辑、时间轴等。这会导致两件事：

— 它们可能很难在修订版之间进行比较。

— 无法描述 diff，因此在将更改推送到存储库时写入整个文件。

同样，在分布式环境中，整个项目历史记录在用户的本地计算机上可用。现在，如果您有一个大文件的历史记录，这些文件在很长一段时间内发生了许多更改，那么您的计算机上将存储该文件的多个副本。这会很快占用您的大部分硬盘空间！

这是历史上团队更喜欢集中式工作流程的主要原因之一。这样，二进制文件的大型历史版本将仅存在于中央服务器上，单个用户只能在其计算机上访问最新版本。

Perforce 和 UVCS 都是集中式系统，可以很好地处理大文件。UVCS 还为您提供了在分布式管道中工作的选项，而大文件大小是您在这些选项之间进行选择时需要考虑的权衡。

Another feature of UVCS is the Dynamic Workspace, which relies on a virtual filesystem. This means that the Dynamic Workspace downloads files on demand – so, while you see everything in your workspace, in reality not everything is downloaded.

Git, being distributed, can struggle with large files. Be sure to also include Git LFS if you will be working with large files. Git LFS replaces your large files in the .git folder with text pointers while storing the actual asset on a server such as GitHub.

UVCS 的另一个功能是 Dynamic Workspace，它依赖于虚拟文件系统。这意味着 Dynamic Workspace 按需下载文件 – 因此，虽然您可以看到工作区中的所有内容，但实际上并非所有内容都已下载。

分发的 Git 可能会难以处理大文件。如果您要处理大文件，请务必还包括 Git LFS。Git LFS 将 .git 文件夹中的大文件替换为文本指针，同时将实际资产存储在 GitHub. 等服务器上

Best practices for version control

Regardless of which VCS you use, many best practices can help your team work more effectively. Every team has different needs, so every practice won't fit every team.

These tips come from the Unity Enterprise Support Team, who are helping to optimize real-world projects for some of the biggest studios out there.

Commit little, commit often

This is by far the easiest change you can make to your workflow, yet it's the one that some developers struggle with the most. When working with other project management tools, it's likely you have already broken down the work into small, manageable tasks. Commits should be exactly the same.

A single commit should only relate to one task or ticket, unless a single line of code magically fixes several bugs. If you are working on a larger feature, break it down into smaller tasks, and make commits for those tasks. We'll dive into feature branches later.

The biggest advantage of using smaller commits is that when something does go wrong, you will find the change much more easily and can revert the negative change without affecting any other positive changes.

版本控制的最佳实践

无论您使用哪种 VCS，许多最佳实践都可以帮助您的团队更有效地工作。每个团队都有不同的需求，因此每次练习都不适合每个团队。

这些提示来自 Unity 企业支持团队，他们正在帮助一些最大的工作室优化实际项目。

少投入，经常投入

这是迄今为止您可以对工作流程进行的最简单的更改，但也是一些开发人员最难以应对的更改。使用其他项目管理工具时，您可能已经将工作分解为小的、可管理的任务。提交应完全相同。

单个提交应该只与一个任务或工单相关，除非一行代码神奇地修复了几个错误。如果你正在开发一个更大的功能，请将其分解为较小的任务，并为这些任务进行提交。我们稍后将深入研究功能分支。

使用较小提交的最大优点是，当出现问题时，你将更容易找到更改，并且可以在不影响任何其他积极更改的情况下还原负面更改。

Keep commit messages clean

Commit messages describe the history of your project. It's much easier to find the change that added high-score tables to your game if its commit message says "Added high score tables to the menu" and not "Menu updated!"

When working with a task ticketing system like JIRA or GitLab, it's even better to include a ticket number in your commit. Many systems can be set up to work together with smart commits, in which you can actually reference tickets and change their status from your commit message.

For example, the commit "JRA-123 #close #comment task completed" would set JIRA ticket JRA-123 to closed, leaving the comment "task completed" on the ticket.

For more on setting this workflow up, see the documentation in JIRA or the Pivotal Tracker service in GitLab.

Avoid indiscriminate commits

The only time "commit -a" (the git command for "commit all changes") or any of its counterparts should be used is with the first commit of a project. Usually, this is when the only files in the project are README.md.

A commit should only ever include files that are related to the change you are committing to the repo. Particular care should be taken when working with Unity projects, and some changes may result in several files being marked as changed, such as scenes, prefabs, or sprite atlases, even though you didn't intend to make any changes to them.

If you accidentally commit a change to a scene that someone else is working on, that could cause a headache for them when they go to commit their changes and find they need to merge your changes first.

This is one of the most common mistakes that people who are new to version control will make. It's important to understand that you should only commit what *you* have changed in the project. To learn more, check out this blog post on how to speed up your workflow.

保持提交消息干净

提交消息描述了您的项目的历史。如果游戏的提交消息显示“Added high score tables to the menu”而不是“Menu updated”，那么找到为游戏添加高分表的更改要容易得多。

当使用 JIRA 或 GitLab 等任务工单系统时，最好在提交中包含工单编号。许多系统可以设置为与智能提交一起工作，在这些系统中，您实际上可以引用工单并从提交消息中更改它们的状态。

例如，提交 “JRA-123 #close #comment task completed” 会将 JIRA 工单 JRA-123 设置为 closed，在工单上留下注释 “task completed”。

有关设置此工作流的更多信息，请参阅 JIRA 中的文档或 GitLab. 中的 Pivotal Tracker 服务

避免不分青红皂白的提交

唯一应该使用 “commit -a”（用于 “commit all changes” 的 git 命令）或其任何对应项的时间是项目的第一次提交。通常，这是项目中唯一 README.md 文件时。

提交应仅包含与你提交到存储库的更改相关的文件。使用 Unity 项目时应特别小心，某些更改可能会导致多个文件被标记为已更改，例如场景、预制件或 sprite 图集，即使您不打算对它们进行任何更改。

如果您不小心将更改提交到其他人正在处理的场景，这可能会让他们头疼，因为他们去提交更改并发现他们需要先合并您的更改。

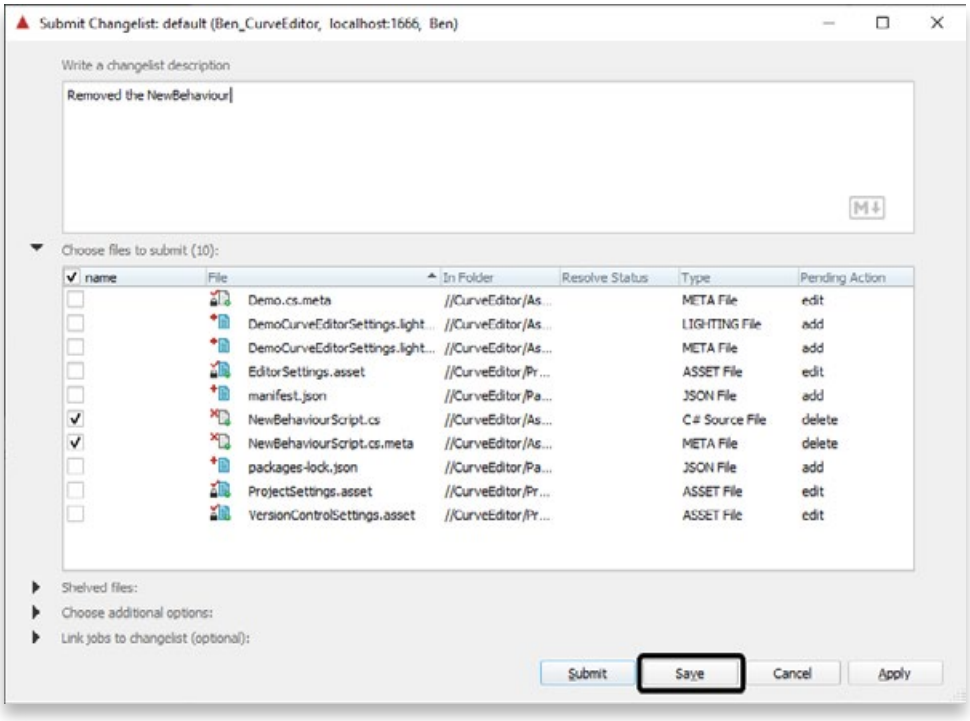
这是版本控制新手最常犯的错误之一。重要的是要了解您应该只提交项目中 *you* 更改的内容。要了解更多信息，请查看这篇关于如何加快工作流程的博客文章。

Get the latest

As often as it makes sense, pull the latest changes from the repo into your working copy. It's not good to work off in isolation, as this only increases the likelihood of merge conflicts. A typical daily workflow in each system would be something like this.

Git	Perforce

— git pull
— Then as many times as you like:
  — Make edits in your working copy.
  — git commit your changes.
  — git pull the latest changes.
— Once you are happy with your changeset of commits:
  — git pull once more.
  — git push to send your commits to the repo.
	— Get latest
	— Check out files to work on
	— Make edits
	— Submit changes



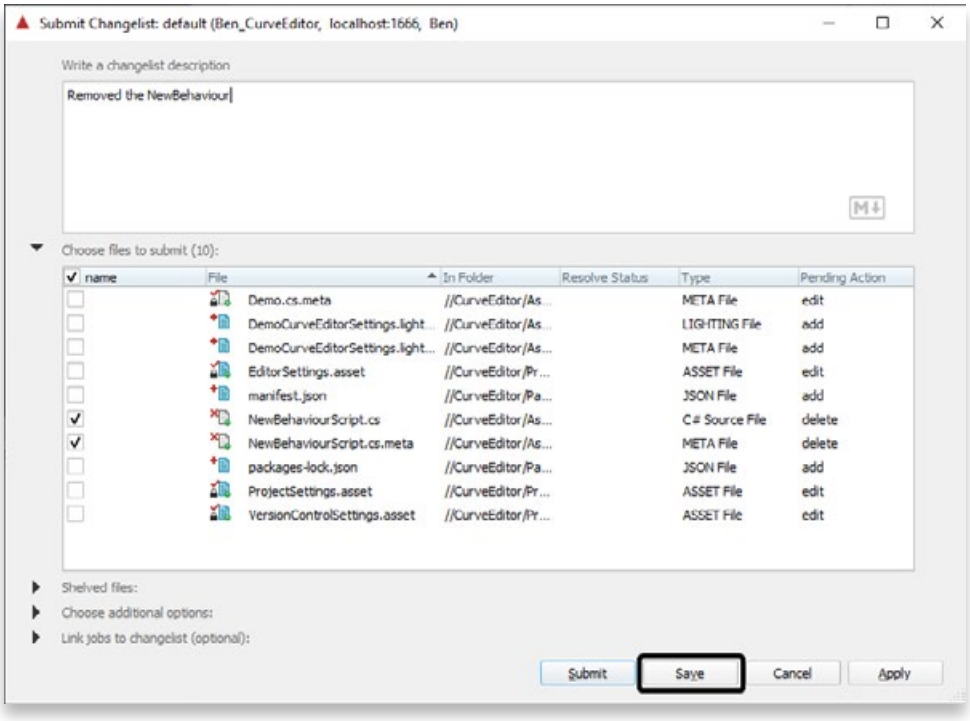
Saving changes to a new changelist in P4V

获取最新版本

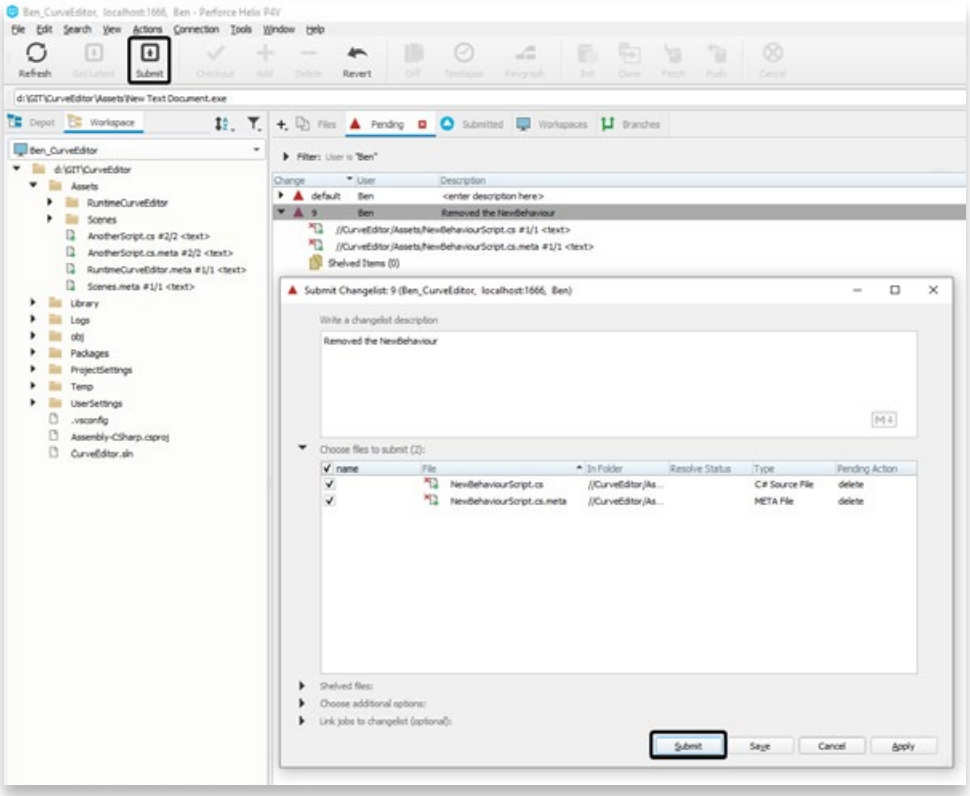
只要有意义，就把最新的更改从 repo 拉到你的工作副本中。孤立地工作是不好的，因为这只会增加合并冲突的可能性。每个系统中典型的日常工作流程是这样的。

Git	Perforce

— git pull
— Then as many times as you like:
  — Make edits in your working copy.
  — git commit your changes.
  — git pull the latest changes.
— Once you are happy with your changeset of commits:
  — git pull once more.
  — git push to send your commits to the repo.
	— Get latest
	— Check out files to work on
	— Make edits
	— Submit changes



在 P4V 中保存对新变更列表的更改

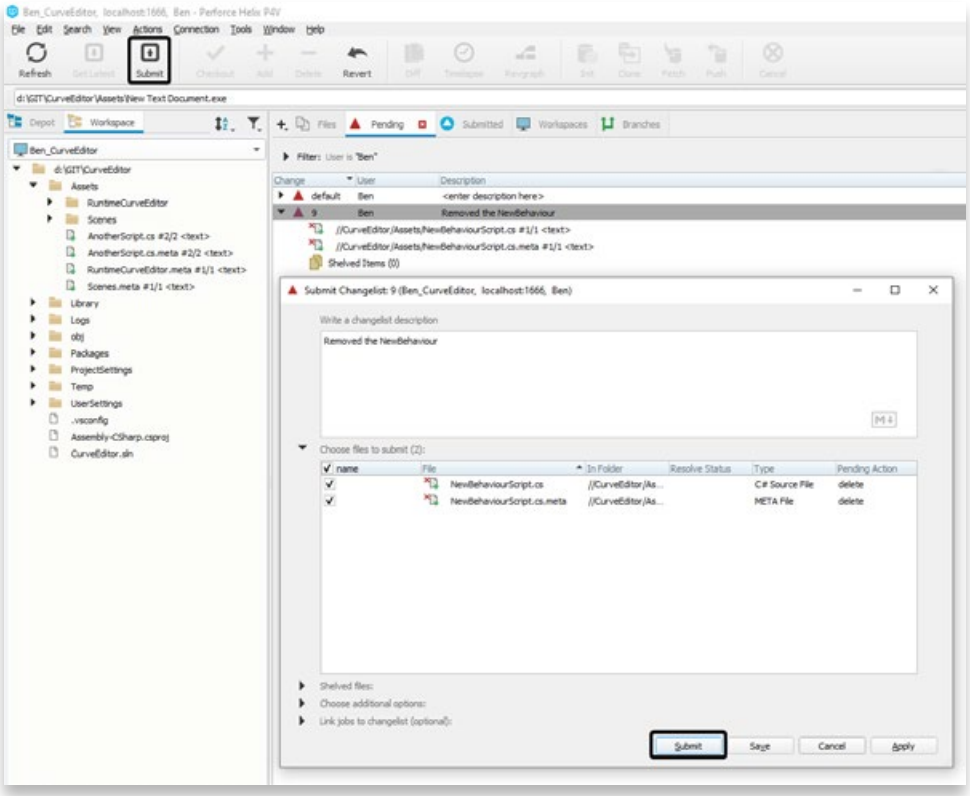


Submitting a changeset in P4V

UVCS workflows are a little different because you can work in centralized, distributed, or multi-site configurations.

UVCS (centralized)	UVCS (distributed)	UVCS (multi-site)
— Sync repositories — Pull visible — Check out files to work on — Make edits — Check in changes — Sync repositories — Push visible	— Pull changes from the server — Check in changes to your local copy — Pull any new changes — Push your changes back up to the server	— A hybrid of the two, depending on your setup

Multi-site configurations can be tailored to custom needs, with each user working in either a centralized or distributed workflow.



在 P4V 中提交变更集

UVCS 工作流程略有不同，因为您可以在集中式、分布式或多站点配置中工作。

UVCS (centralized)	UVCS (distributed)	UVCS (multi-site)
— Sync repositories — Pull visible — Check out files to work on — Make edits — Check in changes — Sync repositories — Push visible	— Pull changes from the server — Check in changes to your local copy — Pull any new changes — Push your changes back up to the server	— A hybrid of the two, depending on your setup

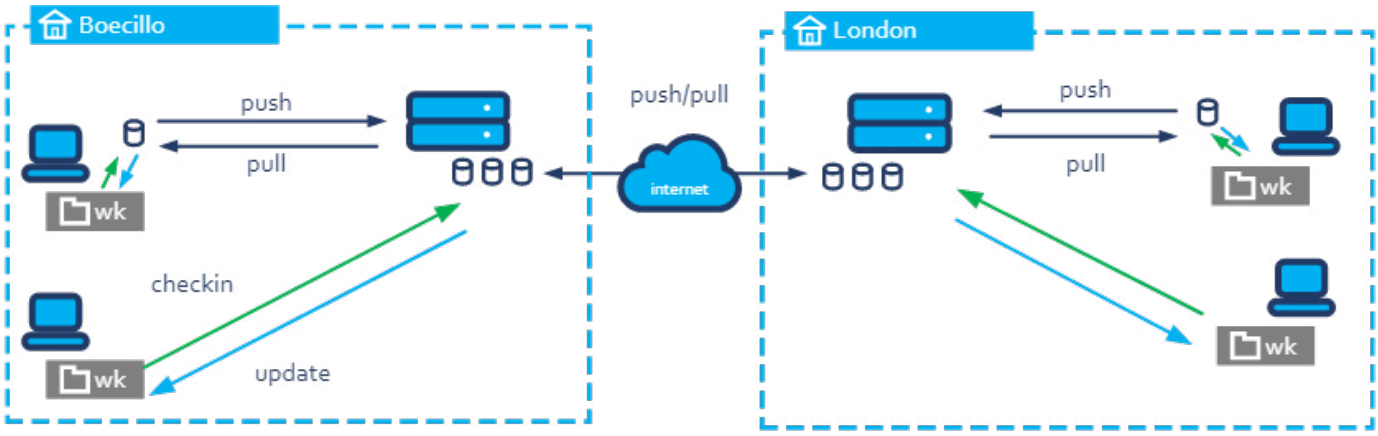
多站点配置可以根据自定义需求进行定制，每个用户在集中式或分布式工作流程中工作。

Consider the following example of two teams:

— Each team has an on-site server.

— Team members at both sites check in locally or distributed but benefit from the speed of a close on site server.

— Servers push/pull between one another to keep fully or partially in sync.



Multi-site UVCS configuration

Know your toolset

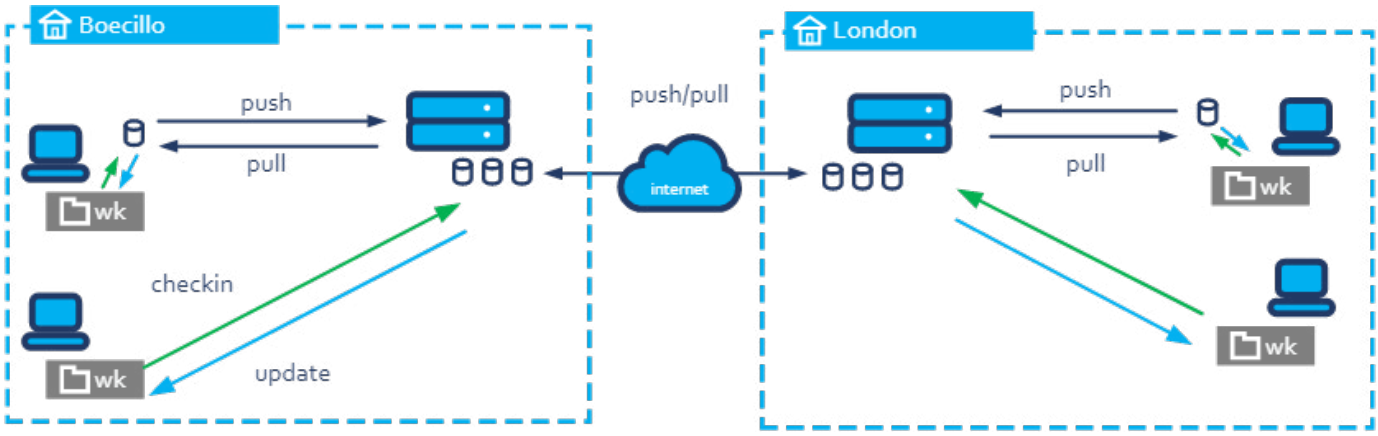
Whichever VCS your team chooses to work with, make sure that the team is comfortable using it and understands the tools at their disposal including visual clients.

If you're working with Git, not everyone needs to use the same GUI client. But make sure that everyone is comfortable with the commit > pull > push workflow, and that they know how to commit only the files they need.

If you're working with UVCS, let your artists get comfortable using Gluon to simplify their workflow. Gluon lets you decide which files you want to work on and only download those, removing the need to download and manage the entire project. It allows you to lock a file to prevent others from working on it, and, once you're finished, users can submit files back to the repository and unlock them again.

请考虑以下两个团队的示例：

— 每个团队都有一个现场服务器。 — 两个站点的团队成员在本地或分布式签到，但受益于现场关闭服务器的速度.— 服务器在彼此之间推送/拉取以保持完全或部分同步。



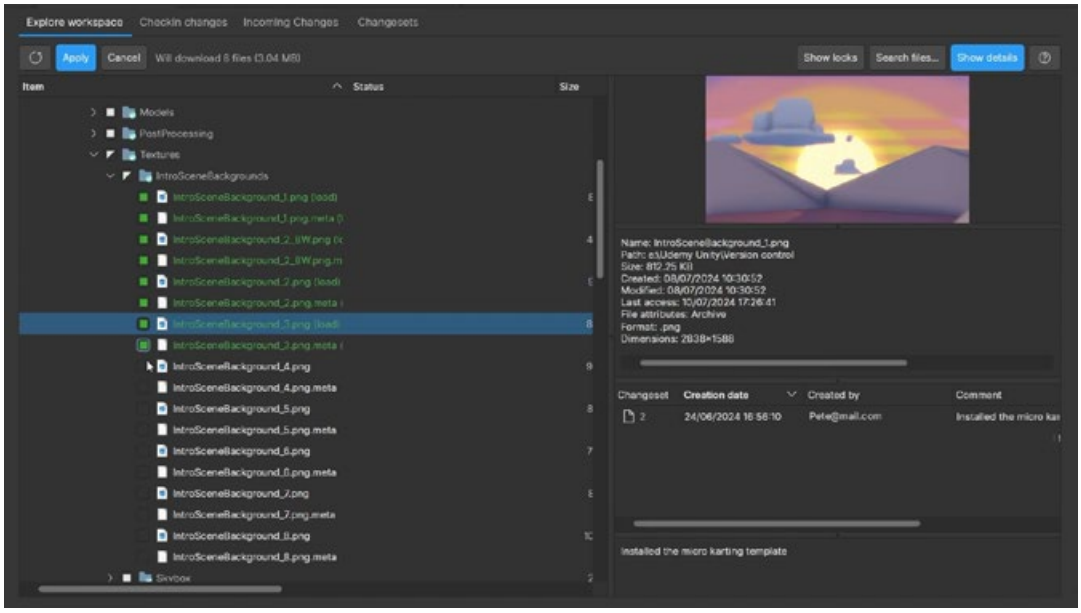
多站点 UVCS 配置

了解您的工具集

无论您的团队选择使用哪种 VCS，请确保团队能够舒适地使用它并了解他们可以使用的工具，包括视觉客户端。

如果您正在使用 Git，则并非每个人都需要使用相同的 GUI 客户端。但请确保每个人都熟悉提交 > 拉取 > 推送工作流程，并且他们知道如何仅提交他们需要的文件。

如果您正在使用 UVCS，请让您的艺术家熟悉使用 Gluon 来简化他们的工作流程。Gluon 允许您决定要处理哪些文件，并且只下载这些文件，无需下载和管理整个项目。它允许您锁定文件以防止其他人处理该文件，并且一旦您完成，用户可以将文件提交回存储库并再次解锁它们。

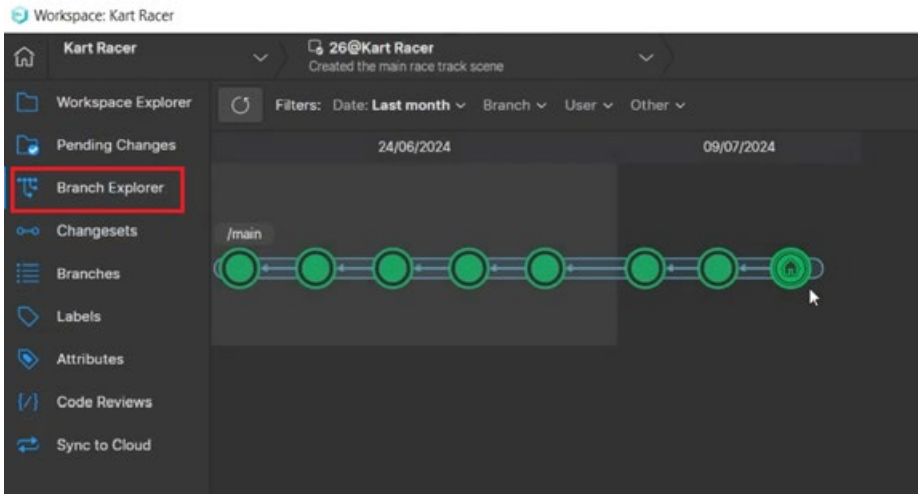


Gluon in UVCS

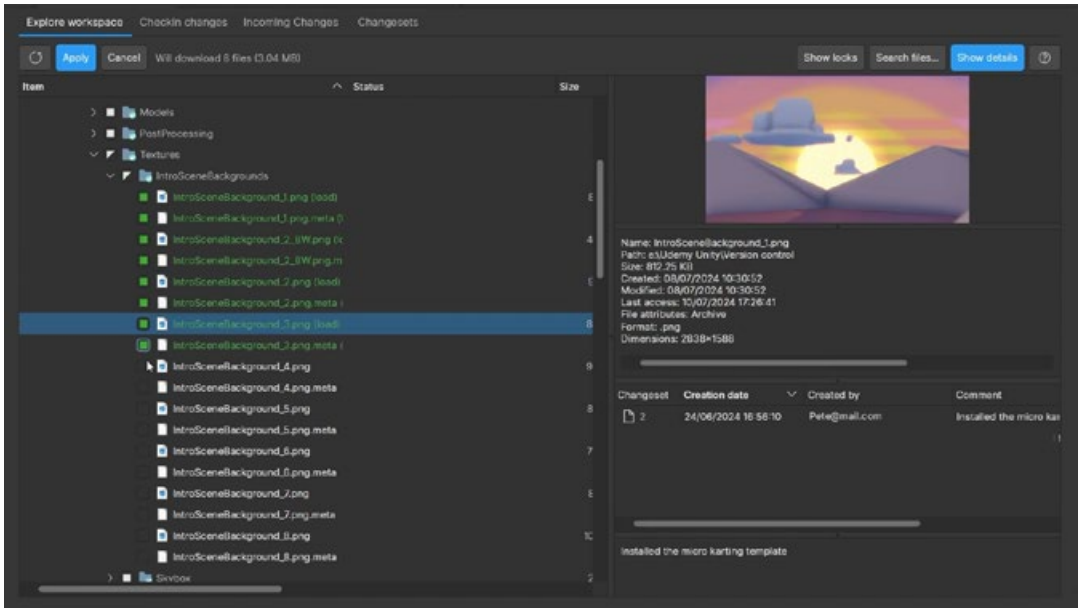
If you are working with Perforce Helix Core, use the built-in Unity Editor tools for managing version control directly from the Editor. This is incredibly useful, both for artists or for general handling of Unity asset files such as scenes, Prefabs, and so on. You can check out assets for modification in the Editor, make your changes, and then check them back in without even leaving Unity.

Feature branches and Git Flow

When you're working on a long-standing project with multiple release cycles, feature branching is hugely beneficial to your workflow. Often, teams work out of the same branch of a repo that would likely be called trunk, master, or main. When you do this, your entire project moves along the same timeline.



Development along the main branch in UVCS

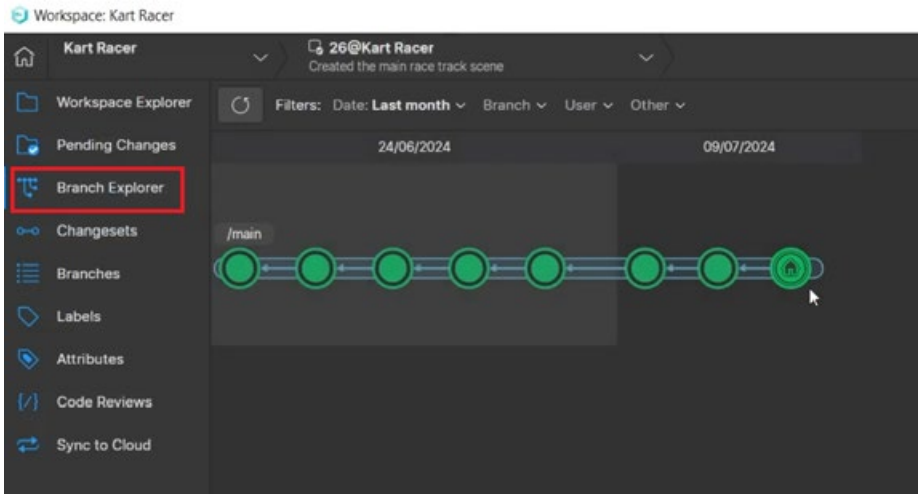


UVCS 中的胶子

如果您使用的是 Perforce Helix Core，请使用内置的 Unity 编辑器工具直接从编辑器管理版本控制。这对于艺术家或 Unity 资源文件（如场景、预制件等）的一般处理都非常有用。您可以在 Editor 中签出要修改的资源，进行更改，然后重新签入它们，甚至无需离开 Unity.

功能分支和 Git 流

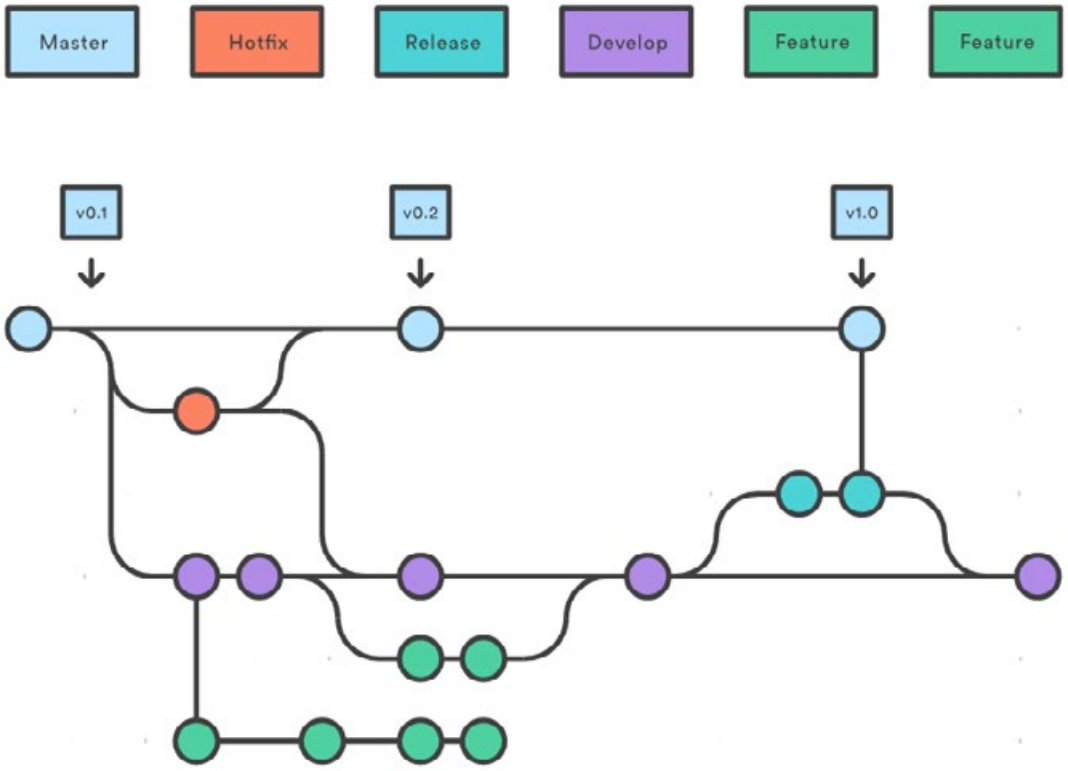
当您处理具有多个发布周期的长期项目时，功能分支对您的工作流程非常有益。通常，团队在存储库的同一分支中工作，该分支可能称为 trunk、master 或 main。执行此作时，整个项目将沿同一时间轴移动。



UVCS 中沿主分支的发展

However, it can be beneficial to split the work off into several branches to work more effectively as a team.

In Git, a specific workflow called Git Flow focuses on using different branches for features, bug fixes, and releases. A developer starts out work on a new feature inside an isolated branch, and when they're finished, it's merged back into the main branch. Meanwhile, someone else may have had to do a hotfix on the previous release, fixed a bug, and released a new version safely, without any of the features still under development being included.

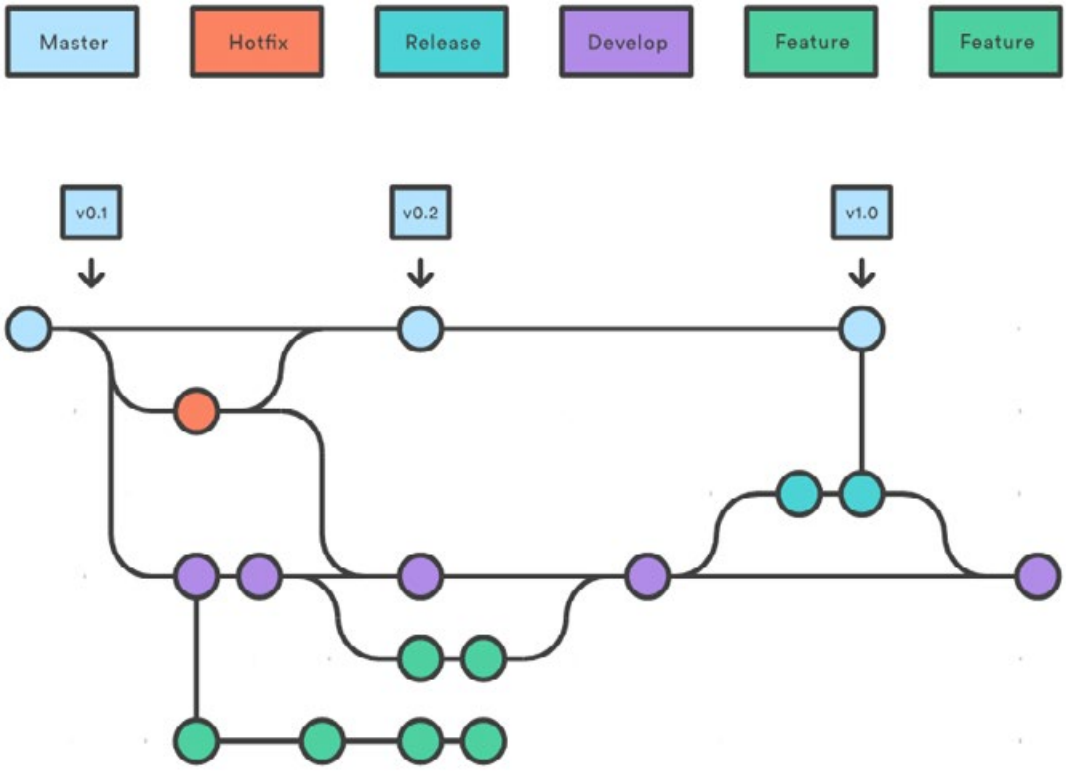


A Git Flow workflow allows for easier release management.

UVCS also features task branches. For this pattern, you create a new branch for every task that you track. While in Git Flow, we use feature branches to develop complete, sometimes large, features, task branches in UVCS are meant to be short-lived. If a task takes more than a handful of commits to implement, odds are it could be broken down into smaller tasks.

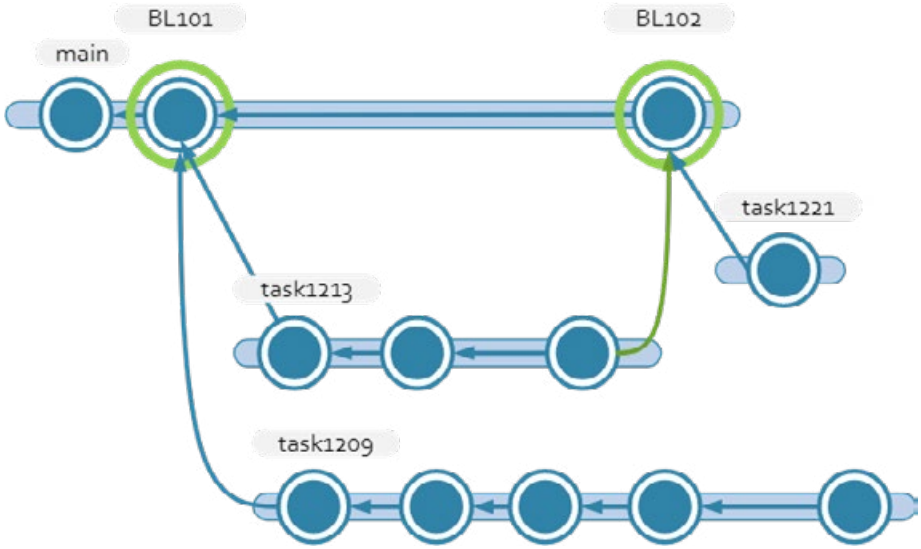
但是，将工作拆分为多个分支以更有效地作为一个团队工作可能是有益的。

在 Git 中，一个名为 Git Flow 的特定工作流专注于使用不同的分支来提供功能、错误修复和发布。开发人员在独立分支中开始开发新功能，完成后，它会合并回主分支。同时，其他人可能不得不对以前的版本进行热修复，修复一个错误，并安全地发布新版本，而不包含任何仍在开发中的功能。



Git Flow 工作流可以更轻松地进行发布管理。

UVCS 还具有任务分支。对于此模式，您将为跟踪的每个任务创建一个新分支。在 Git Flow 中，我们使用功能分支来开发完整的（有时是大型的）功能，而 UVCS 中的任务分支是短暂的。如果一个任务需要大量的提交来实现，那么它很可能被分解成更小的任务。

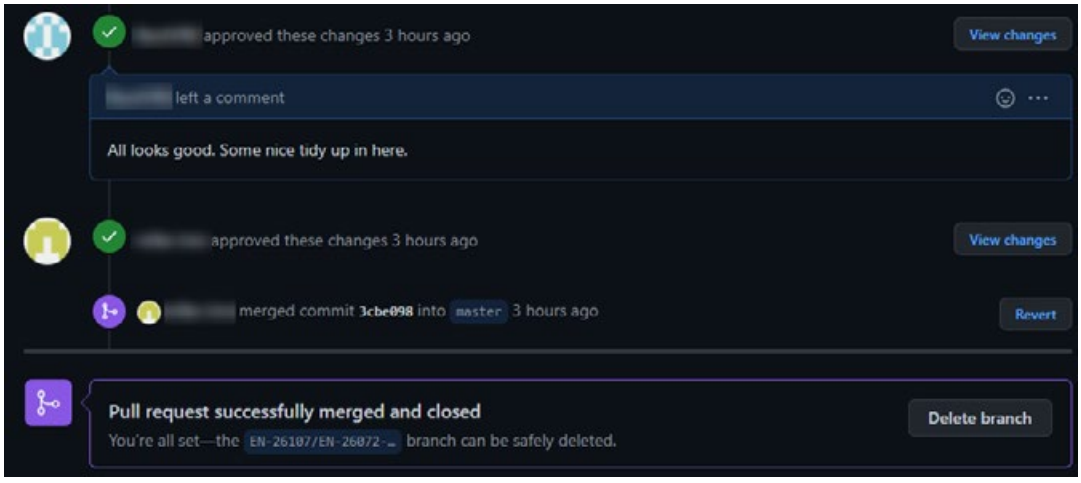


UVCS branch per task pattern

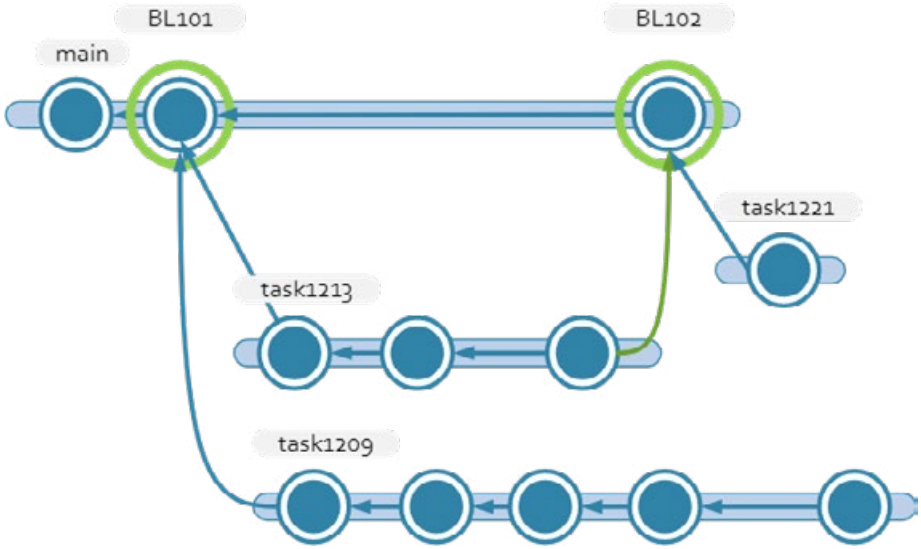
Perforce Helix Core uses a system called Streams to facilitate this style of workflow. When creating a depot to work in, you need to set it up as a stream depot type. Then, you can use the Stream Graph view to create new streams. Every stream other than the mainline stream will need to have a parent stream, so changes can be copied back up-stream.

Pull requests

Once you've completed work on a feature branch, it's a good practice to use pull requests to get your changes back into the main stream of the repo. Pull requests are created by the developers of the feature or task, and it's usually the responsibility of a senior developer or DevOps to review the changes before accepting them into the mainline.



A closed pull request on GitHub

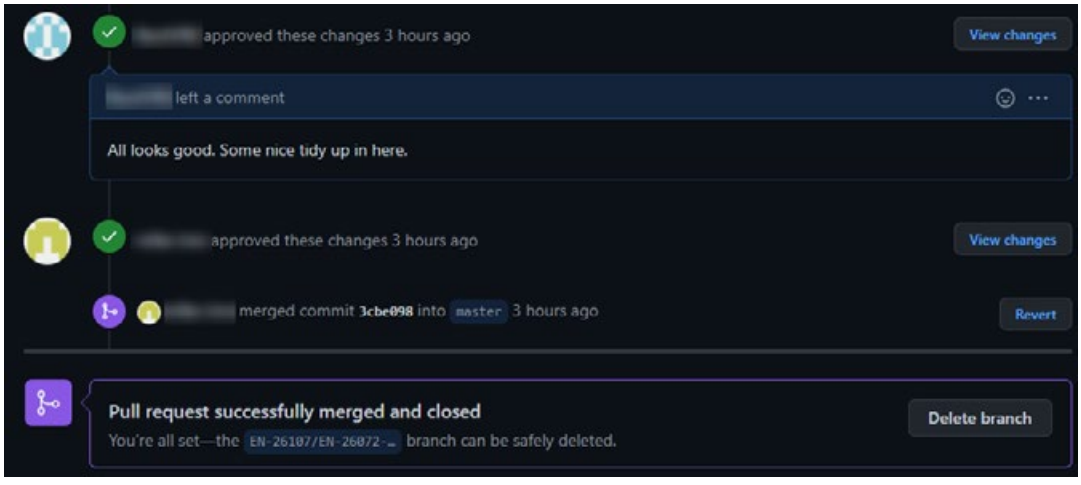


每个任务的 UVCS 分支模式

Perforce Helix Core 使用一个名为 Streams 的系统来促进这种风格的工作流程。创建要使用的站点时，需要将其设置为河流站点类型。然后，您可以使用 Stream Graph （流图） 视图创建新流。除主线流之外的每个流都需要有一个父流，以便可以将更改复制回上游。

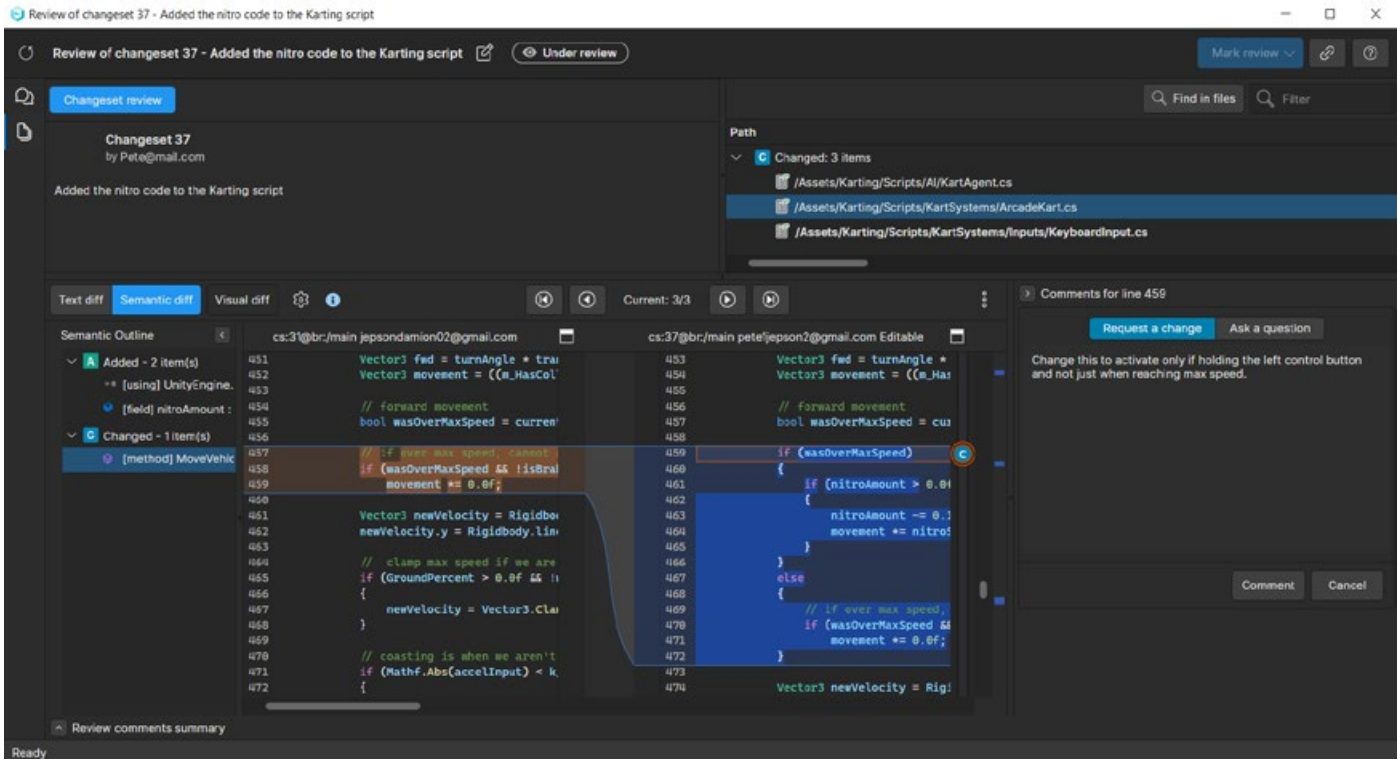
拉取请求

完成功能分支的工作后，最好使用拉取请求将更改重新纳入存储库的主流。拉取请求由功能或任务的开发人员创建，通常高级开发人员或 DevOps 有责任在将更改接受到主线之前对其进行审查。



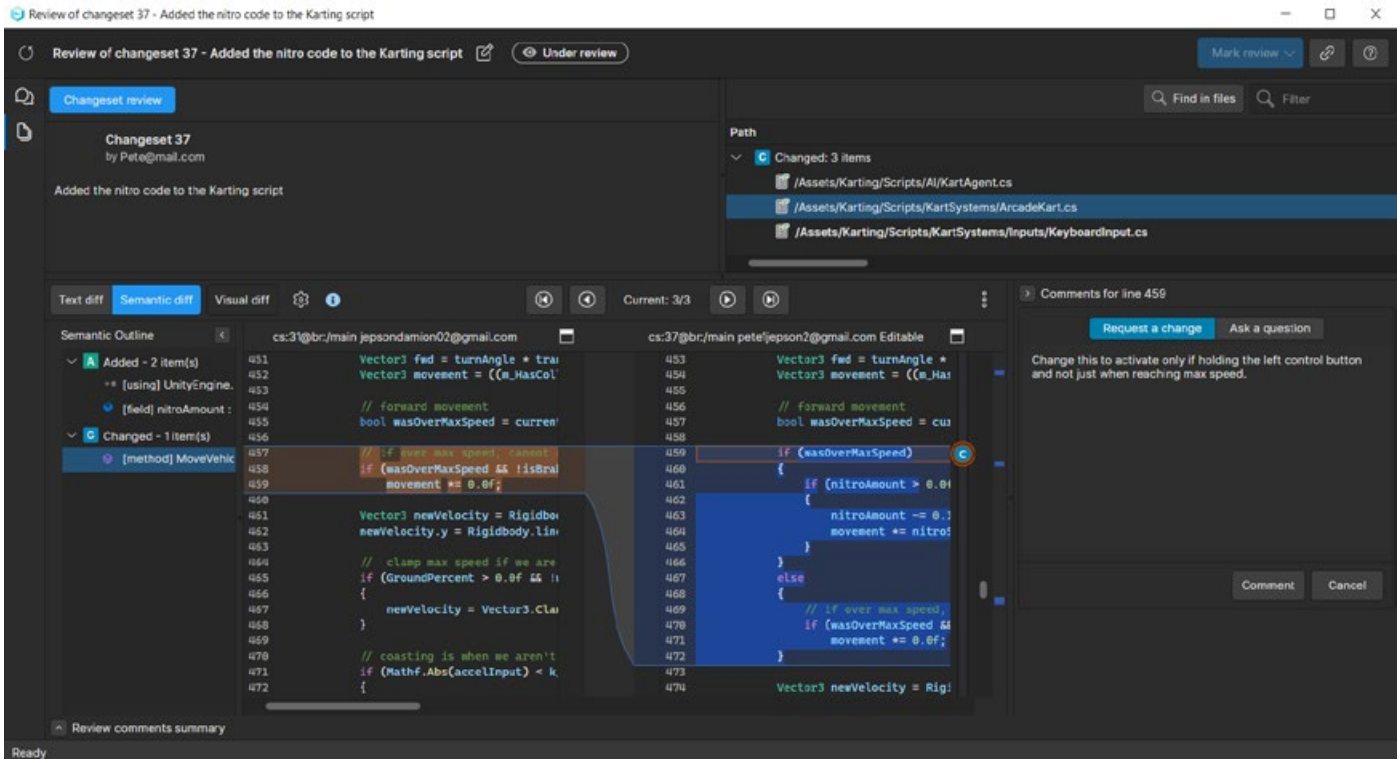
GitHub 上的已关闭拉取请求

UVCS and Perforce both have automated tools to help manage merging branches back into the mainline. UVCS does this with the help of Mergebot, which automatically merges branches of a repo once they've been reviewed and passed validation. Perforce has an additional platform, Helix Swarm, for managing code reviews that can also be set up with automated testing.



UVCS code reviews are included in the GUI.

UVCS 和 Perforce 都有自动化工具来帮助管理将分支合并回主线。UVCS 在 Mergebot 的帮助下实现了这一点，一旦 Mergebot 的分支经过审查并通过验证，它就会自动合并仓库的分支。Perforce 还有一个额外的平台 Helix Swarm，用于管理代码审查，也可以通过自动化测试进行设置.



UVCS 代码审查包含在 GUI. 中

Get started with UVCS in Unity 6

The Unity DevOps offering comprises UVCS and Build Automation. UVCS is free for up to three (3) team members (seats) and up to 5GB of data per month. After that, pricing depends on your monthly active users and total cloud storage. UVCS sends warning emails to the UVCS organization owner when you reach 50%, 75%, and 90% of your usage allowance for any of the UVCS services so it's easy for you to keep track of your usage.

See the Unity Cloud pricing plans page for more information or, sign into the Unity Cloud dashboard and go to the DevOps dashboard "About" page.

There are three ways to access UVCS: via multiple applications and repositories through the UVCS desktop client, by adding it to your projects through the Unity Hub, or accessing the repository on Unity cloud via your web browser.

Unity 6 中的 UVCS 入门

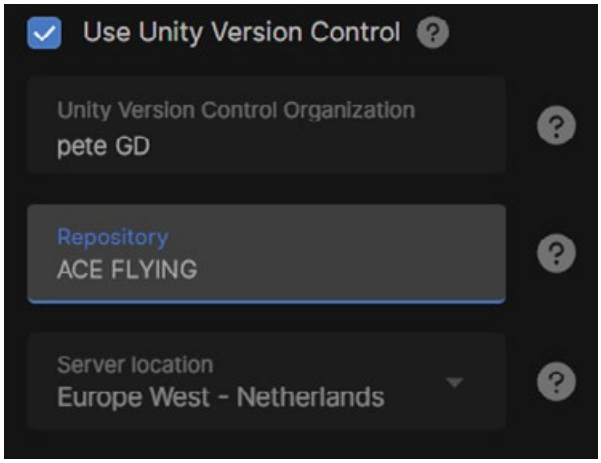
Unity DevOps 产品包括 UVCS 和 Build Automation。UVCS 对最多三 （3） 名团队成员（席位）免费，每月最多可使用 5GB 数据。之后，定价取决于您的每月活跃用户和总云存储。当您达到任何 UVCS 服务的使用限额的 50%、75% 和 90% 时，UVCS 会向 UVCS 组织所有者发送警告电子邮件，以便您轻松跟踪您的使用情况。

有关更多信息，请参阅 Unity Cloud 定价计划页面，或者登录 Unity Cloud 控制面板并转到 DevOps 控制面板的“关于”页面.

有三种方法可以访问 UVCS：通过 UVCS 桌面客户端通过多个应用程序和存储库，通过 Unity Hub 将其添加到项目中，或通过 Web 浏览器访问 Unity 云上的存储库。

Use UVCS in a Unity project

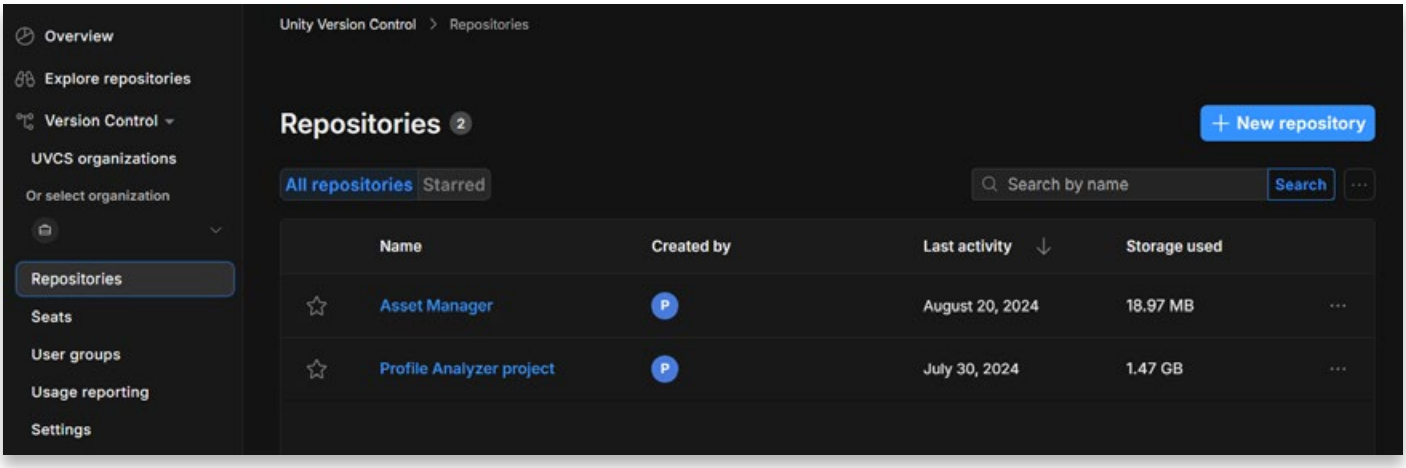
Unity will automatically set up a repository (repo) when you create a project using the Unity Hub and check the **Version Control** checkbox.



Setting up Version Control in Unity Hub

This means that your project's data and files will be stored both locally on your machine, but also on the cloud, in order to help with collaboration and act as a safety backup. The name of your project becomes the name of the repo, and you can choose the server that is closest to you for high speed connections.

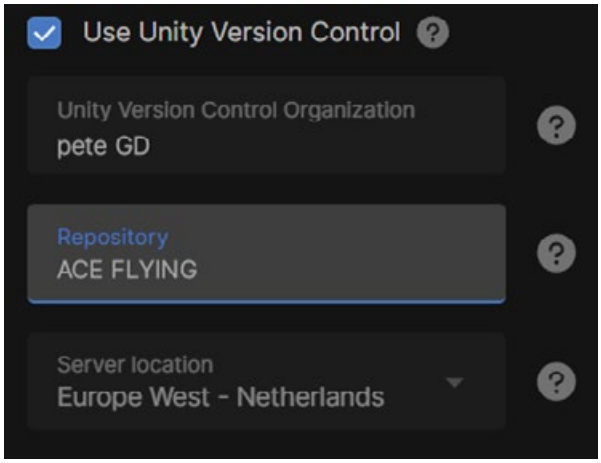
The Assets, Packages, and Project Settings folders are backed up onto the server on first use. You can access these files by logging into the Unity Cloud and in the repository select **File Explorer**. This shows you all the files that exist on the cloud repo.



The File Explorer on the Unity Cloud dashboard shows the structure of files and folders in the repo.

在 Unity 项目中使用 UVCS

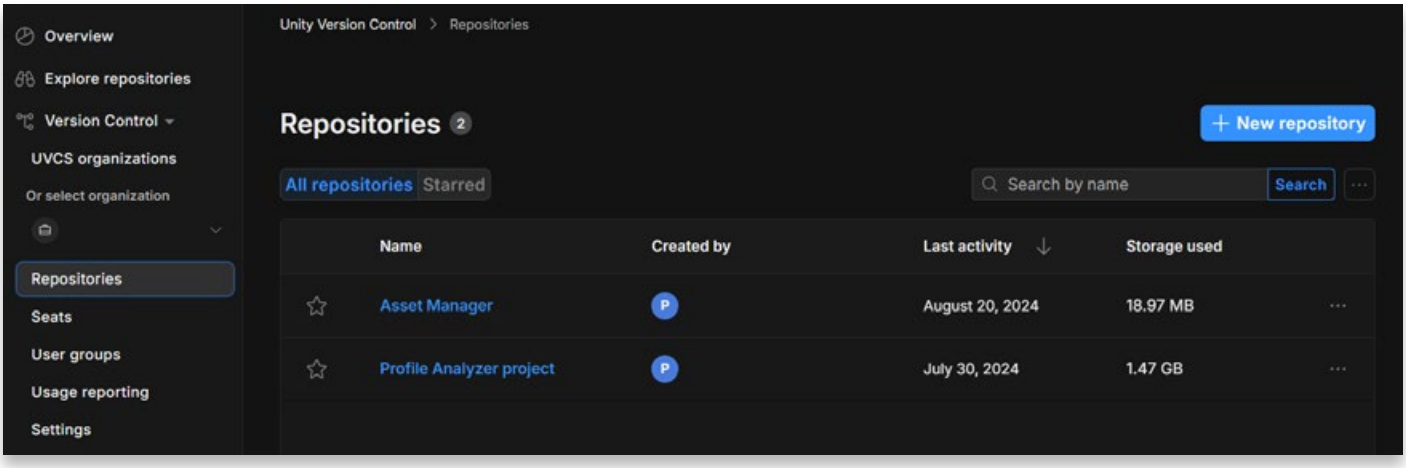
当您使用 Unity Hub 创建项目时，Unity 将自动设置存储库 （repo） 并选中 Version Control 复选框
。



在 Unity Hub 中设置版本控制

这意味着您的项目数据和文件将既存储在本地计算机上，也可以存储在云上，以帮助协作并充当安全备份。您的项目名称将成为存储库的名称，您可以选择离您最近的服务器进行高速连接。

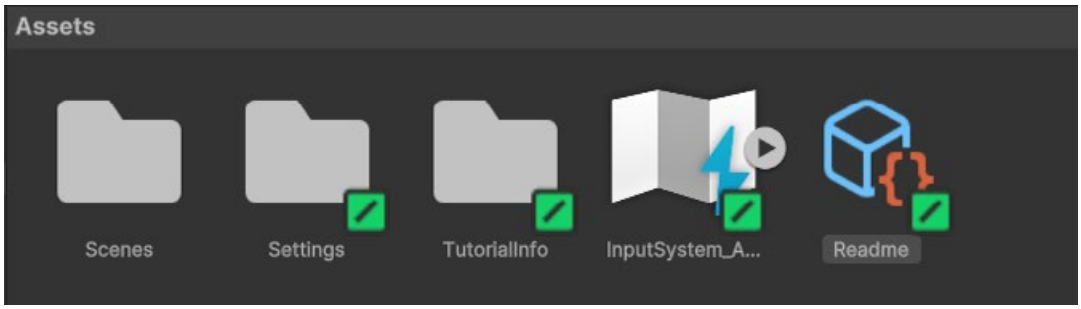
Assets、Packages 和 Project Settings 文件夹在首次使用时备份到服务器上。 您可以通过登录 Unity Clou d 并在存储库中选择 File Explorer 来访问这些文件。这将显示 Cloud 存储库中存在的所有文件。



Unity Cloud 控制面板上的 File Explorer 显示存储库中文件和文件夹的结构。

UVCS ignores the Library, Logs and User Settings folders, as these are auto-generated on your computer and take up a lot of space.

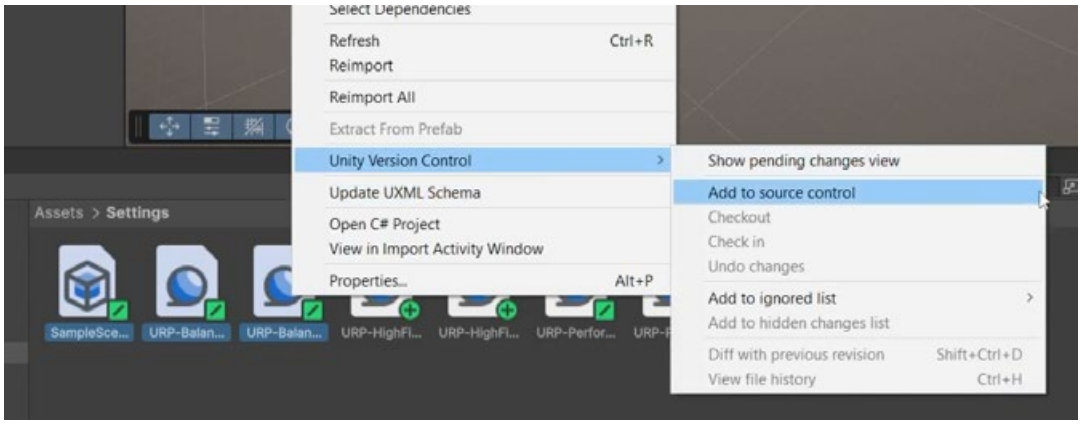
Other folders that are ignored by UVCS will have a green box with a forward slash icon below the folder or file in the Project window.



Ignored files that are not backed up on the cloud are shown with a green box containing a forward slash.

Be sure to include the Settings folder in the **Pending Changes** section of the Version Control window if the project uses URP or HDRP. If the Settings folder has a forward slash in a green box, select it and right-click. Choose **Version Control> Add to source control**.

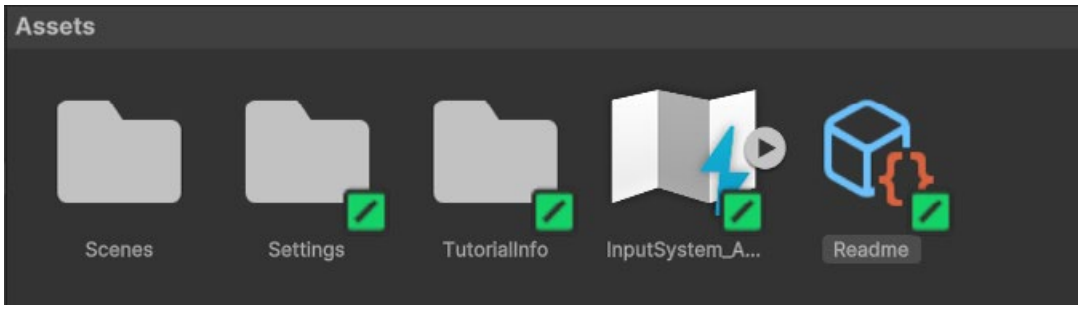
The forward slash will become a plus icon in a green circle and these files will now be added to the pending changes list, ready to be uploaded to the cloud.



Including settings files to the pending changes view

UVCS 会忽略 Library、Logs 和 User Settings 文件夹，因为这些文件夹是在计算机上自动生成的，会占用大量空间。

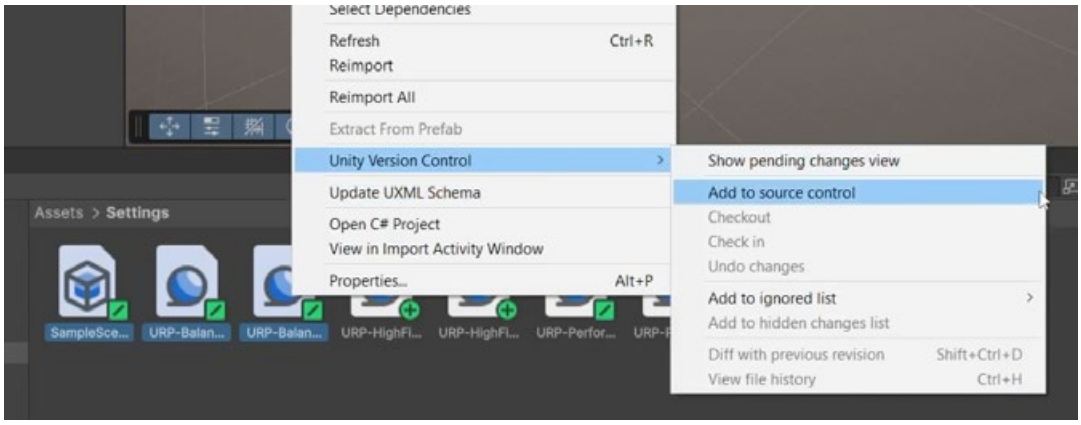
UVCS 忽略的其他文件夹将在 Project 窗口中的文件夹或文件下方有一个带有正斜杠图标的绿色框。



未在云中备份的被忽略的文件将显示一个包含正斜杠的绿色框。

如果项目使用 URP 或 HDRP，请务必将 Settings 文件夹包含在 Version Control 窗口的 Pending Changes 部分中。如果 Settings 文件夹的绿色框中有正斜杠，请选择该文件夹并右键单击。选择 Version Control**>**Add to source control.

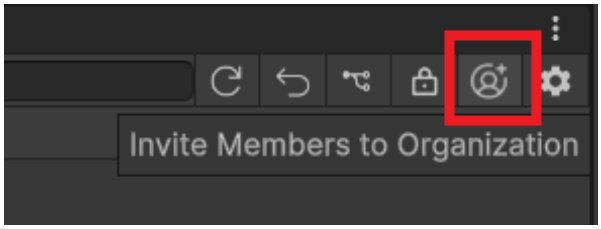
正斜杠将变为绿色圆圈中的加号图标，这些文件现在将被添加到待处理更改列表中，准备上传到云端。



将设置文件包含在待处理更改视图中

Inviting other team members

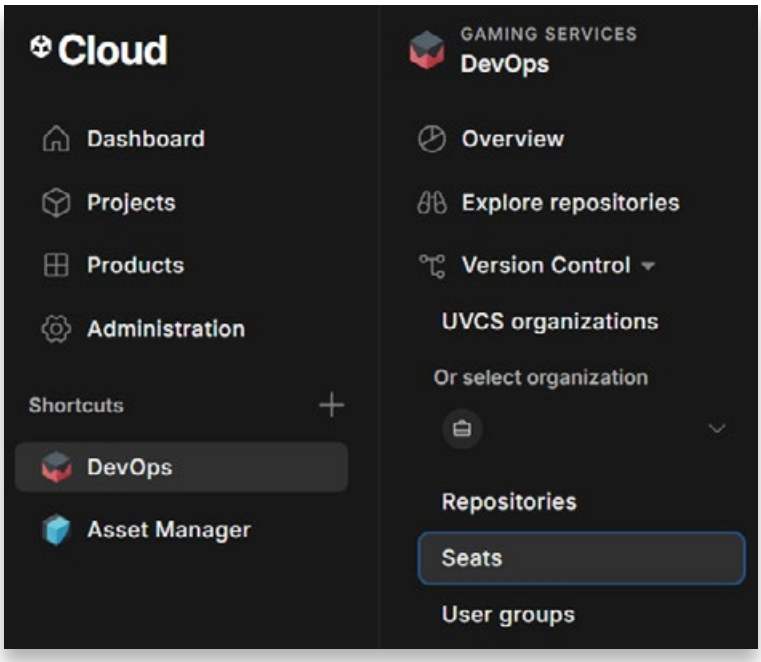
In the Version Control window there is a button on the right of the window that you can use to invite team members.



The **Invite Members to Organization** option in the Version Control window

This will take you to the Unity Cloud login page. Use your Unity ID and password to login.

DevOps will open from the shortcuts on the left and the seats section will then also open, allowing you to add extra seats to your current project. At the time of writing you can't add more than three seats in the free tier. You can upgrade to Cloud Pro to buy additional seats. See the Unity Cloud pricing plans page for more information or, sign into the Unity Cloud dashboard and go to the DevOps dashboard "About" page.

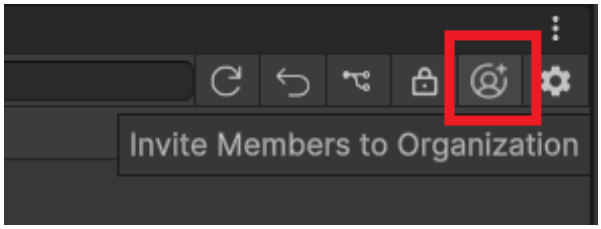


Add extra seats from the DevOps section on the Unity Cloud.

Assign the new members a seat in your project. They should also be invited to become organization members. This gives them full access to all data on the dashboard for the cloud including usage statistics.

邀请其他团队成员

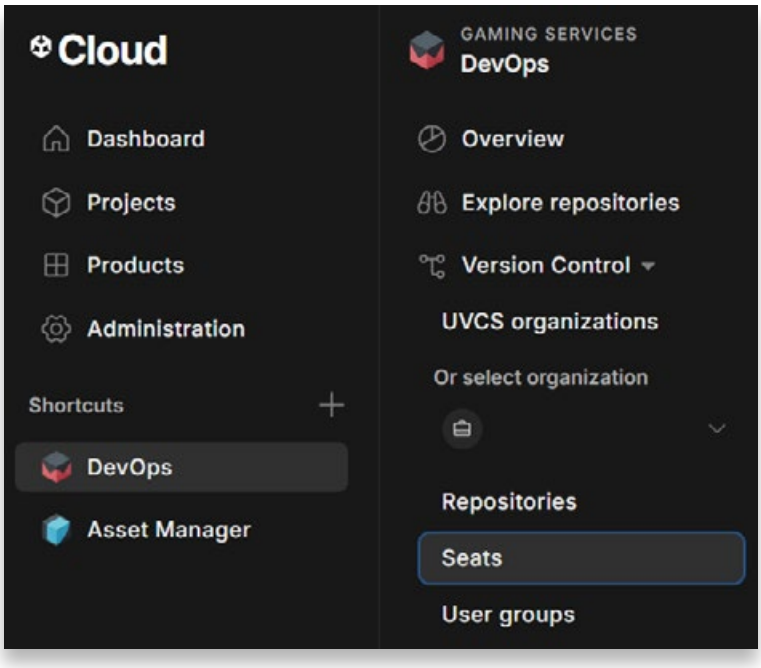
在 Version Control （版本控制） 窗口中，窗口右侧有一个按钮，可用于邀请团队成员。



“版本控制”（Version Control） 窗口中的“邀请成员加入组织”（Invite Members to Organization） 选项

这将带您进入 Unity Cloud 登录页面。使用您的 Unity ID 和密码登录。

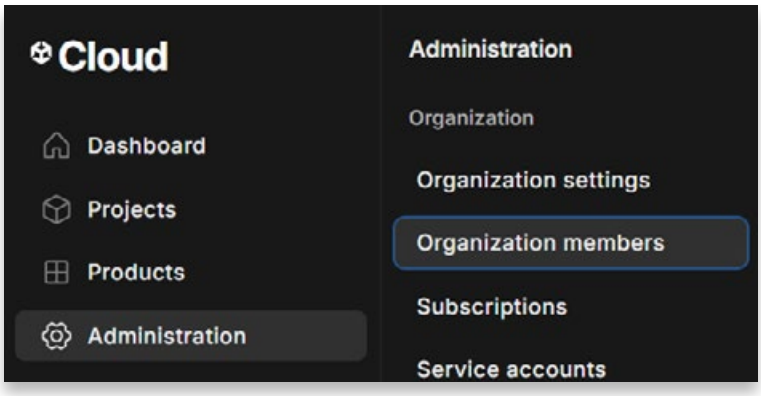
DevOps 将从左侧的快捷方式打开，然后席位部分也将打开，允许您向当前项目添加额外的席位。在撰写本文时，您不能在免费套餐中添加超过三个席位。您可以升级到 Cloud Pro 以购买更多席位。有关更多信息，请参阅 Unity Cloud 定价计划页面，或者登录 Unity Cloud 控制面板并转到 DevOps 控制面板的“关于”页面.



从 Unity Cloud 上的 DevOps 部分添加额外的席位。

在项目中为新成员分配席位。还应邀请他们成为组织成员。这使他们能够完全访问云控制面板上的所有数据，包括使用情况统计信息.

Go to **Administration>Organization members** and click the **invite organization members** button to add extra people to your organization.



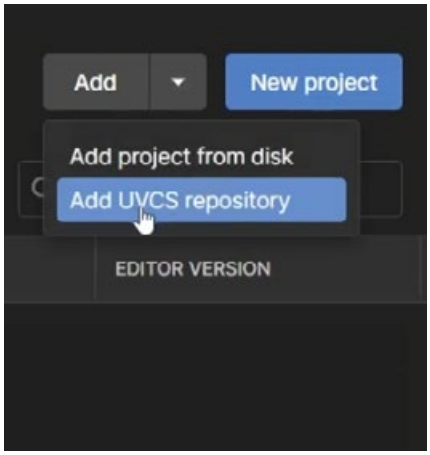
Add extra organization members from the **Administration** option.

Assign each member an appropriate role. Choose from guest, user, or admin. Users and admin members can see all projects and you can assign roles in the project members section to give them the appropriate level of access to the project data.

This allows them to access the project and contribute to it based on their role. Users with admin status will have full access to the options available on the cloud.

An email is sent to the other members that you invite.

The invited member should then open Unity Hub and click on the drop down next to the **Add** button and choose **Add UVCS repository**:

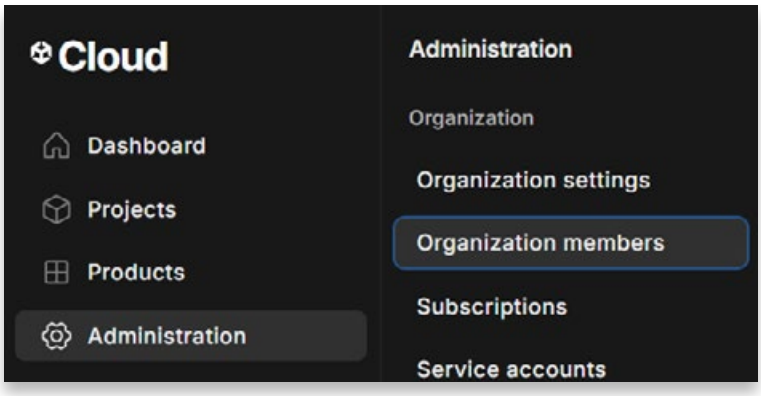


Click **Add UVCS repository** to open a project on the Unity Cloud.

Note: It's important that all team members are using the same version of Unity, otherwise it can cause issues when converting the project between versions.

This will then download the project from the cloud repository onto the user's computer.

转到 Administration**>**Organization members（组织成员），然后单击 invite organization members（邀请组织成员）按钮以向组织添加额外的人员。



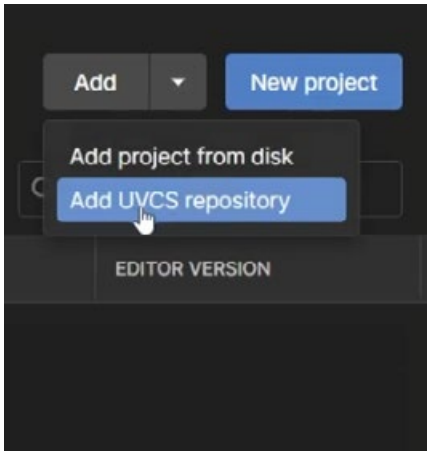
从 Administration 选项添加额外的组织成员。

为每个成员分配适当的角色。从 guest、user 或 admin 中进行选择。用户和管理员成员可以查看所有项目，您可以在项目成员部分中分配角色，以授予他们对项目数据的适当级别的访问权限。

这允许他们访问项目并根据其角色为项目做出贡献。具有管理员身份的用户将拥有对云上可用选项的完全访问权限。

系统会向您邀请的其他成员发送一封电子邮件。

然后，受邀成员应打开 Unity Hub，然后单击 Add 按钮旁边的下拉菜单，然后选择 Add UVCS repository：



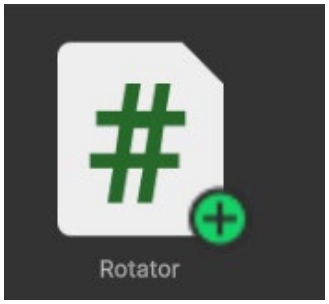
单击 Add UVCS repository 以在 Unity Cloud 上打开一个项目。

注意：所有团队成员都必须使用相同版本的 Unity，否则在版本之间转换项目时可能会导致问题。

然后，这会将项目从云存储库下载到用户的计算机上。

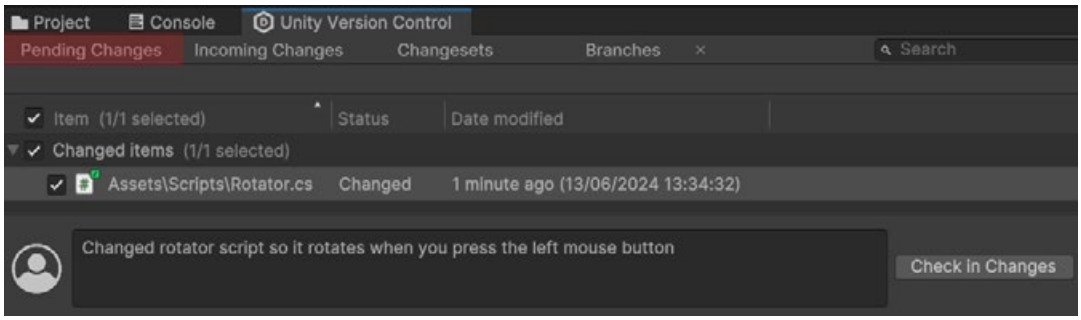
Check in changes

As you make changes to your project you will see a green **+** icon on the files that are changed. This shows you which files to check into the server for update.



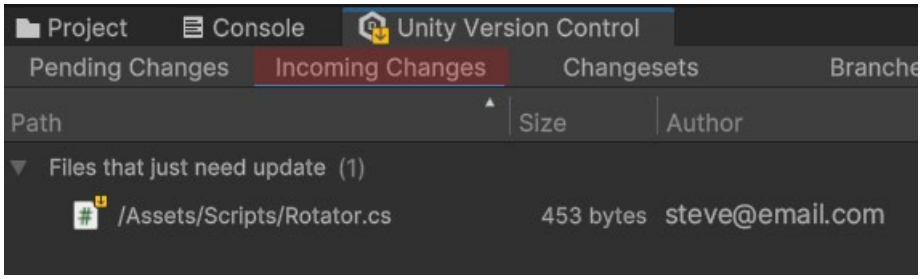
A green circle with a plus icon indicates a file has been changed and needs to be checked in.

Update your project often to make rolling back easier. Updated files are shown in the **Pending Changes** section of the Version Control window. Add a short, concise description of the changes you have made and click the **Check in Changes** button.



The Pending Changes section allows you to check in changes you make to your project.

Other users who are working on the same branch will see a yellow download icon appear in the Version Control window, under the **Incoming Changes** tab. This displays the changes that are made by other users.

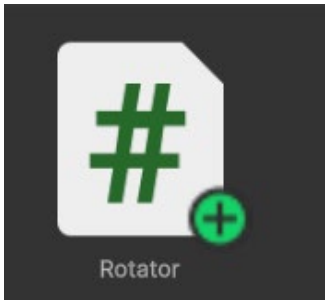


Other users can see your checked-in changes and can update their workspace to incorporate them.

They can click the **Update** workspace button to update their workspace and incorporate your changes.

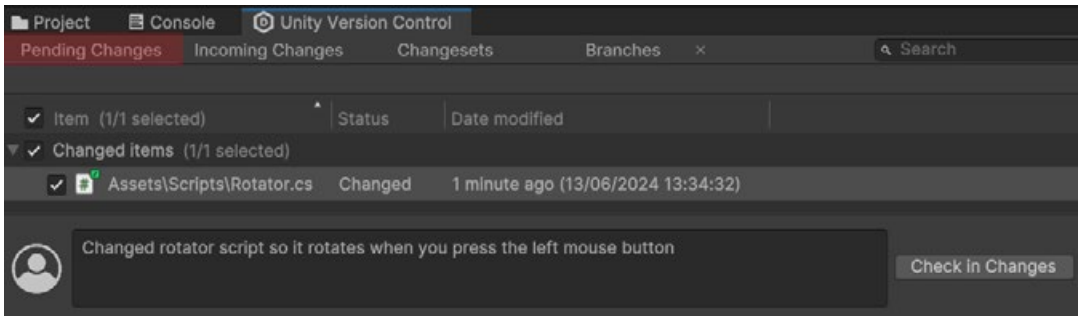
签到更改

当您对项目进行更改时，您将在更改的文件上看到一个绿色的 ✚ 图标。 这将显示要检入服务器以进行更新的文件。



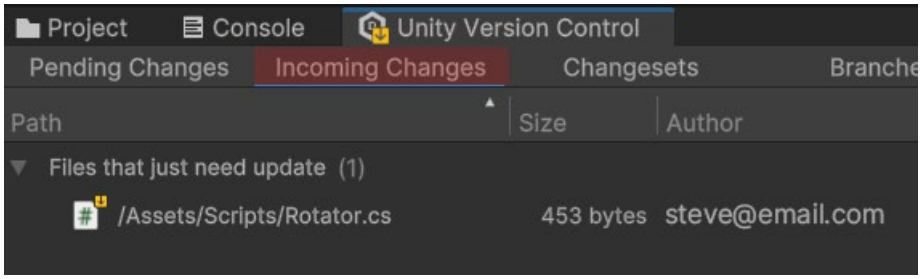
带有加号图标的绿色圆圈表示文件已更改，需要签入。

经常更新您的项目，以便更轻松地回滚。更新的文件显示在 Version Control 窗口的 Pending Changes 部分中。添加您所做的更改的简短描述，然后单击 Check in Changes 按钮.



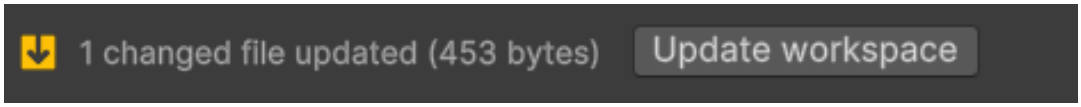
Pending Changes 部分允许您签入对项目所做的更改。

正在处理同一分支的其他用户将在 Version Control 窗口的 Incoming Changes 选项卡下看到一个黄色的下载图标。这将显示其他用户所做的更改。

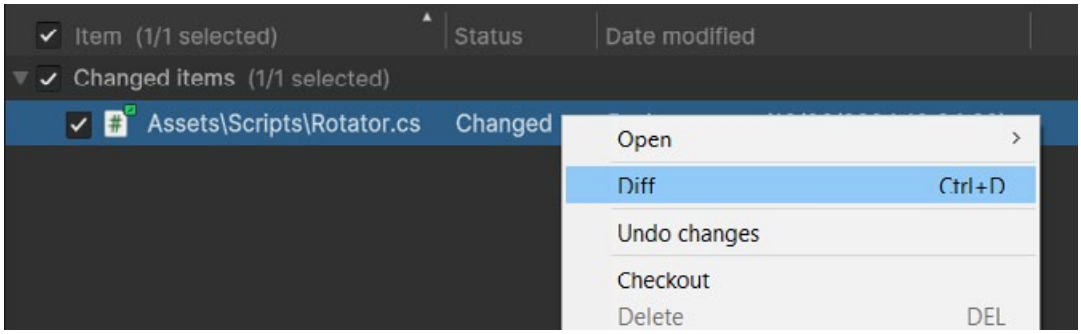


其他用户可以看到您签入的更改，并可以更新其工作区以合并这些更改。

他们可以单击 Update workspace（更新工作区）按钮来更新其工作区并合并您的更改.

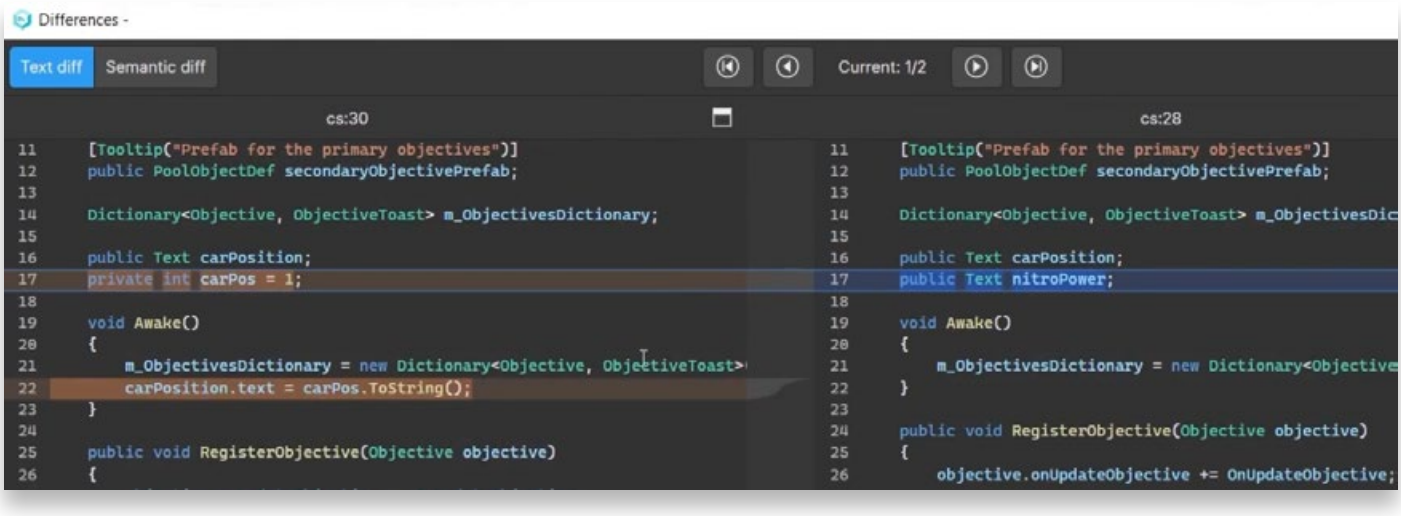


Or they can choose to see the differences from before and after if it's an existing file that has been modified by right-clicking on the changed file and selecting **Diff**.



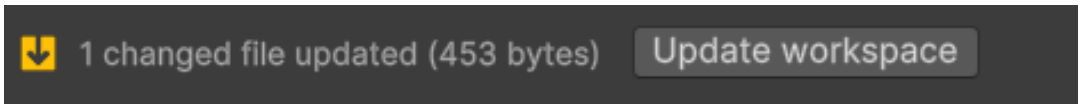
Check the difference between the changed file and the original file.

In the Diff viewer they can now see the original version on the left and the incoming changes on the right. This is useful for C# scripts. Unity's code-aware merge tech, Semantic Merge, tracks moved code to help you focus only on the relevant changes.

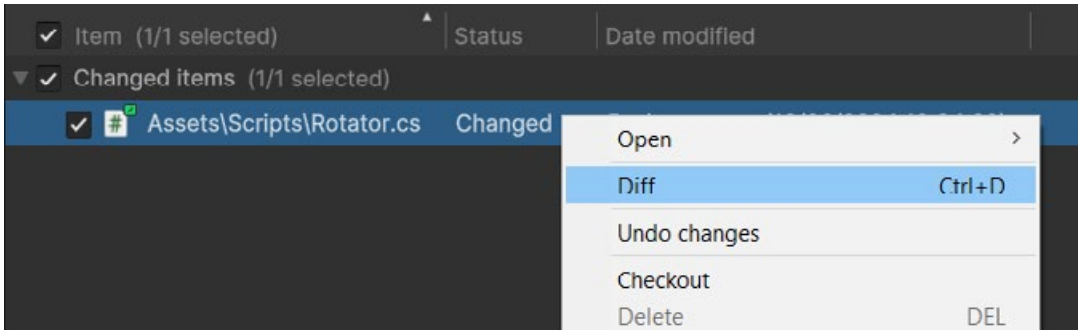


The Diff viewer displays the original on the top, in orange, and the changed file below, in blue.

All changesets can be seen in a list in the **Changesets** section, showing who made the changes on what day and time. The description of the change and the files that have been updated are included in the Changeset list.

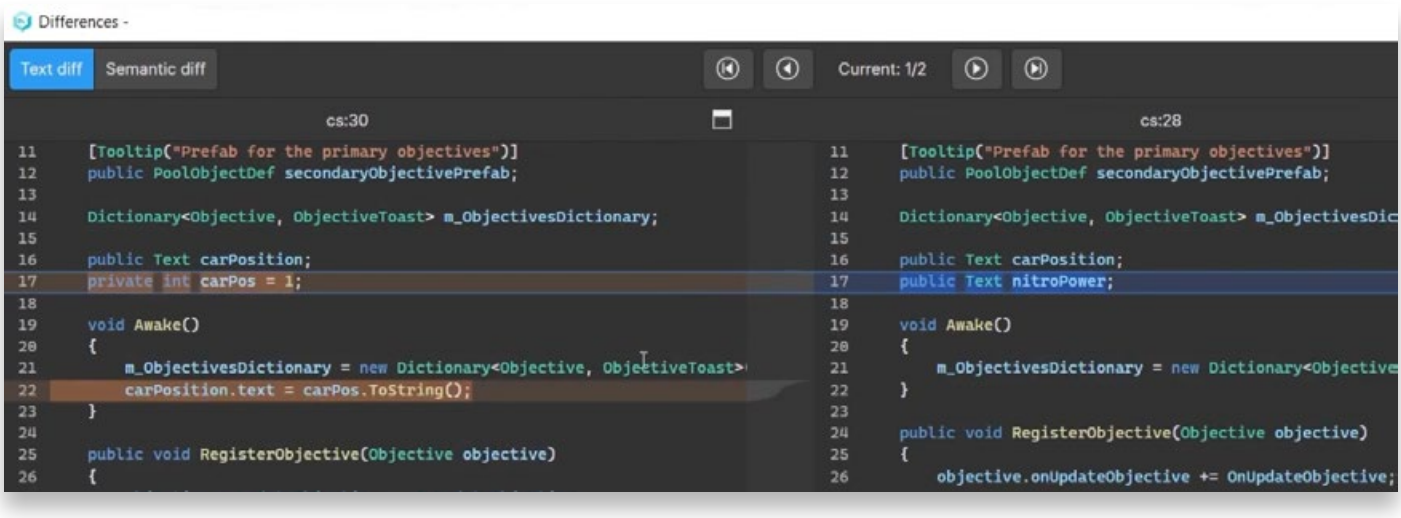


或者，如果它是已修改的现有文件，他们可以通过右键单击已更改的文件并选择 Diff. 来选择查看之前和之后的差异



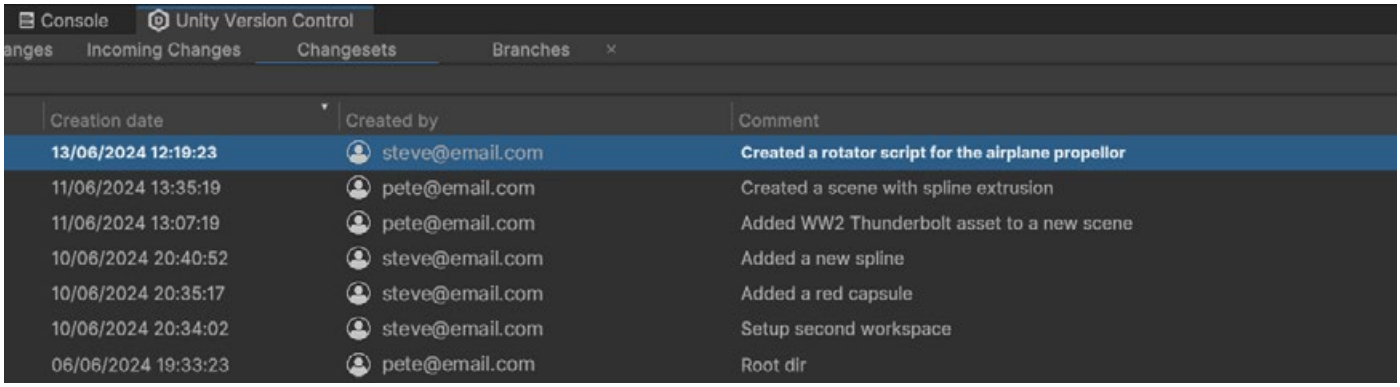
检查更改的文件与原始文件之间的差异。

在 Diff 查看器中，他们现在可以在左侧看到原始版本，在右侧看到传入的更改。这对 C# 脚本非常有用。Unity 的代码感知合并技术 Semantic Merge 会跟踪移动的代码，以帮助您仅关注相关更改。



Diff 查看器在顶部以橙色显示原始文件，在下面以 蓝。

所有变更集都可以在 Changesets （变更集） 部分的列表中看到，显示谁在什么日期和时间进行了更改。更改的描述和已更新的文件包含在 Changeset 列表中。

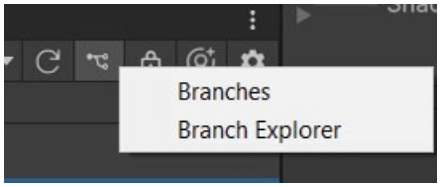


A display of all the changesets made to the project; you can roll back to any changeset

The UVCS desktop client

The Version Control window in the Unity Editor gives you the standard features you'll need to push and pull data from the cloud. The desktop client provides additional features.

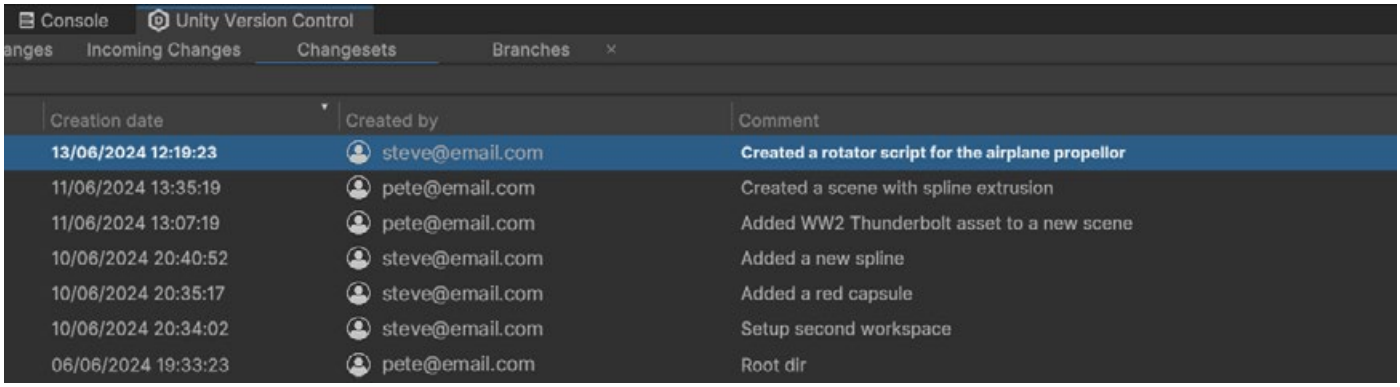
To launch the desktop client click on **Branch Explorer** when working in the expanded mode. If you have not yet installed the client this will prompt you to do so.



Click the Branch Explorer button to launch the UVCS desktop app.

The version control apps, including Gluon and Unity DevOps Version Control, the latter being the UVCS desktop app

Gluon is the more beginner-friendly software while UVCS is the full-featured option. By default clicking Branch Explorer will open UVCS software. You can switch between Gluon and UVCS by opening them from the installed software on your computer. Unity will auto detect which version you are using and switch to using that system.

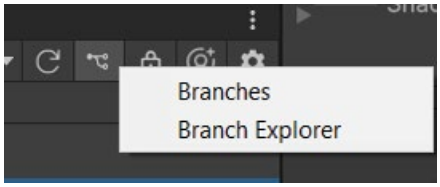


对项目所做的所有变更集的显示;您可以回滚到任何变更集

UVCS 桌面客户端

Unity Editor 中的 Version Control 窗口提供了从云端推送和拉取数据所需的标准功能。桌面客户端提供了其他功能。

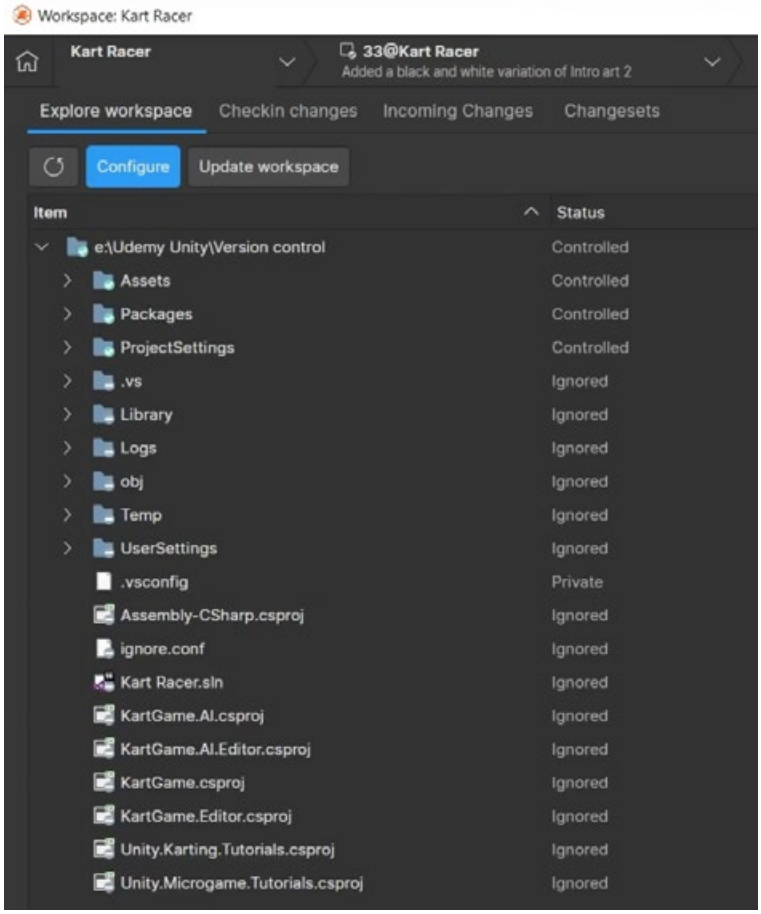
要启动桌面客户端，请在展开模式下工作时单击 Branch Explorer。如果您尚未安装客户端，这将提示您安装。



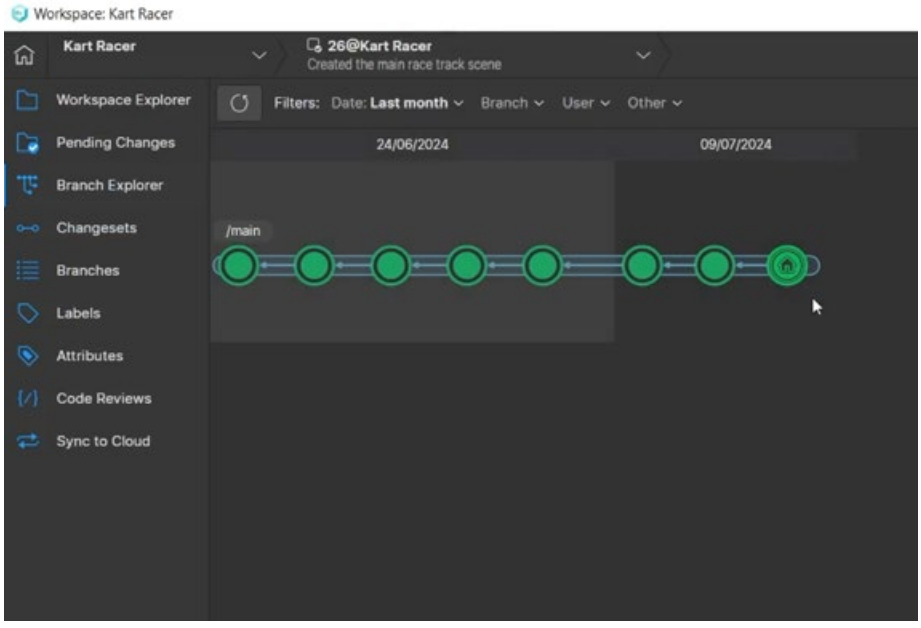
单击 Branch Explorer 按钮以启动 UVCS 桌面应用程序。

版本控制应用程序，包括 Gluon 和 Unity DevOps 版本控制，后者是 UVCS 桌面应用程序

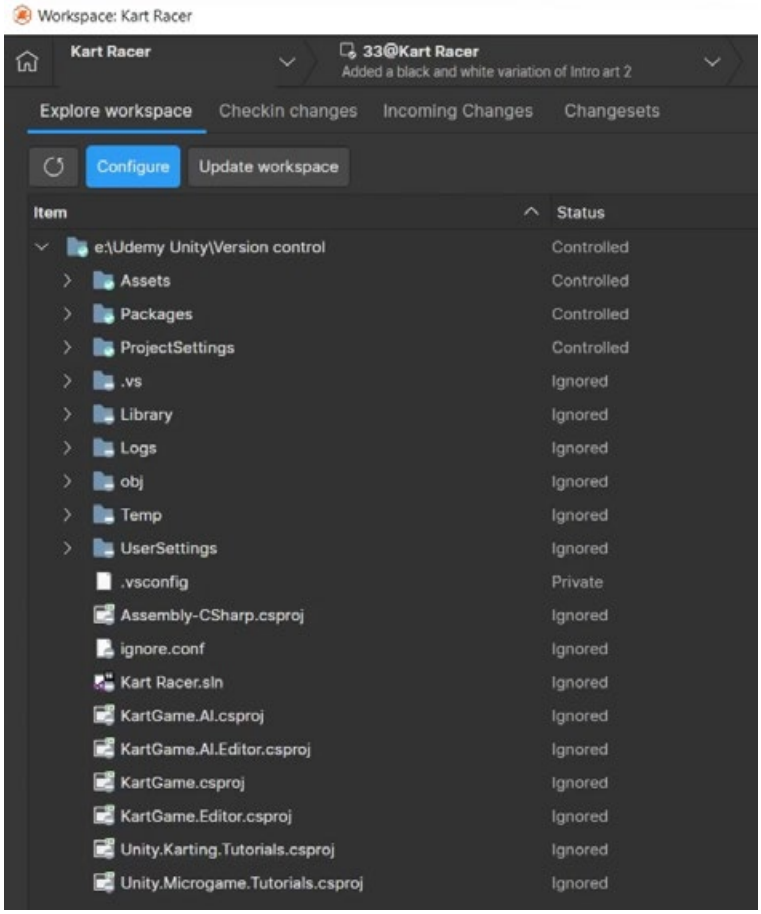
Gluon 是更适合初学者的软件，而 UVCS 是功能齐全的选项。默认情况下，单击 Branch Explorer 将打开 UVCS 软件。您可以通过从计算机上安装的软件中打开 Gluon 和 UVCS 来在 Gluon 和 UVCS 之间切换。Unity 将自动检测您正在使用的版本并切换到使用该系统。



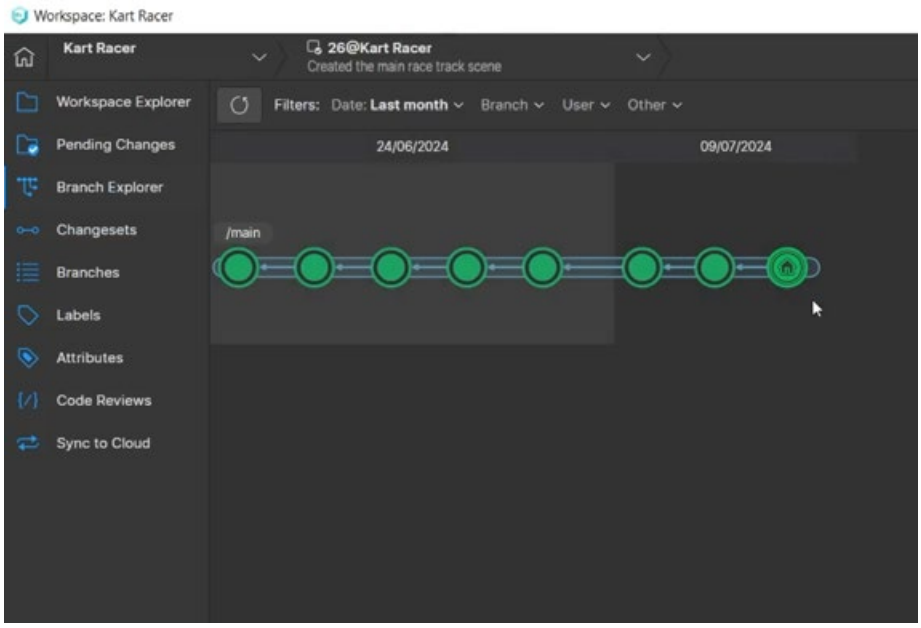
Gluon is a version control app that's artist-friendly.



UVCS is the full software.



Gluon 是一个对艺术家友好的版本控制应用程序。



UVCS 是完整的软件。

Using Gluon

As explained in the section that introduces UVCS, Gluon is a slim client designed specifically with artists in mind. It allows you to pick only the files that you're going to work on and check them out from the server, locking them from being modified by anyone else. Once you complete your work, you check the files back in. The Gluon GUI removes the more complex concepts that work better for programmers than for less technical users.

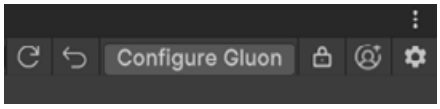
Open Gluon from the installed Unity DevOps Version Control applications.

The Gluon desktop app can be found from your installed Unity DevOps software.

It will prompt you to change from full mode to partial mode, the mode which Gluon runs in.

This warning allows you to change to a partial mode when working in Gluon by clicking **Run an update**.

Inside Unity, a **Configure Gluon** button replaces the **branches** icon. This will open the Gluon app every time you click on it.



Team members using Gluon will now be able to open the Gluon app directly from the Version Control window in Unity.

You can enable the dark theme by clicking on the options button at the top right of the Gluon app.

使用 Gluon

如介绍 UVCS 的部分所述，Gluon 是一个专为艺术家设计的精简客户端。它允许您只选择要处理的文件并从服务器中签出它们，从而锁定它们不被其他任何人修改。完成工作后，将文件签回。Gluon GUI 删除了更复杂的概念，这些概念更适合程序员而不是技术水平较低的用户。

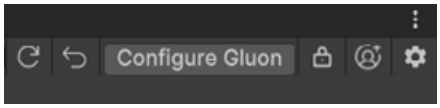
从已安装的 Unity DevOps 版本控制应用程序中打开 Gluon。

可以从已安装的 Unity DevOps 软件中找到 Gluon 桌面应用程序。

它将提示您从 full 模式更改为 partial 模式，即 Gluon 运行的模式。

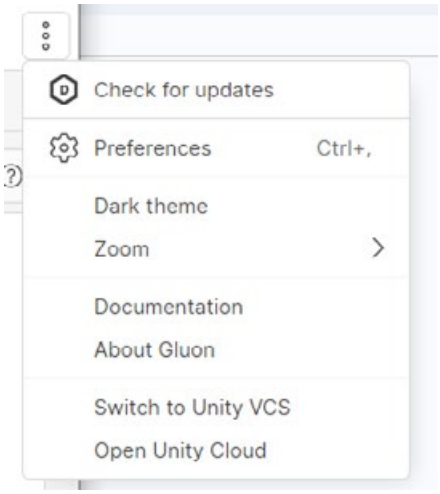
此警告允许您在 Gluon 中工作时通过单击运行 update. 更改为部分模式

在 Unity 中，Configure Gluon 按钮取代了 branches 图标。这将在您每次单击 Gluon 应用程序时打开它。



团队成员u sing Gluon 现在将能够直接从版本控制风中打开 Gluon 应用程序 ow 在 Unity 中。

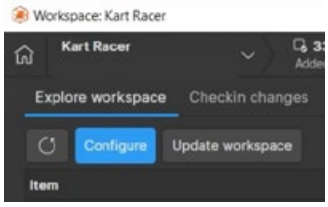
您可以通过单击 Gluon 应用程序右上角的选项按钮来启用深色主题。



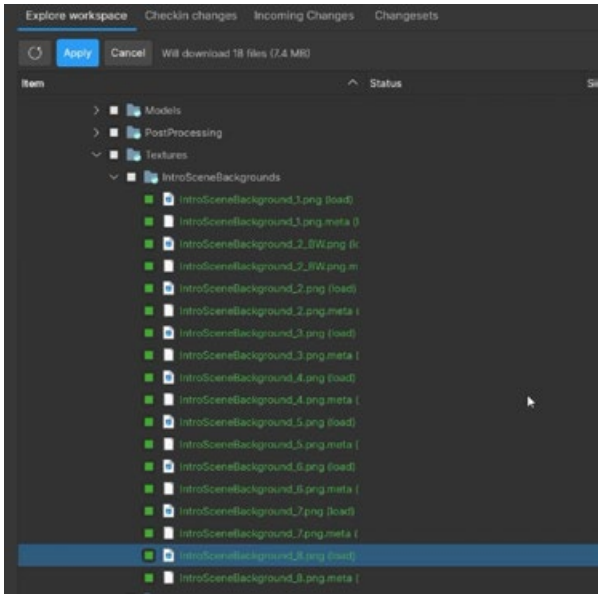
In Gluon you can use the options drop down at the top right to enable the dark theme.

Gluon uses the main branch only, providing a visual, artist-friendly workspace. You can use Gluon to easily access the **Explore workspace** feature to upload and download files as needed.

In the Explore workspace section click the **Configure** button to make changes to the files on the repo.



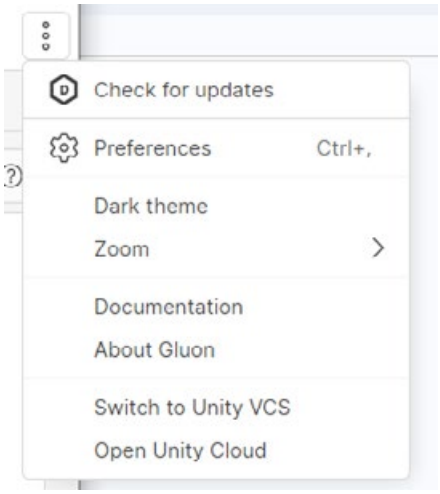
Click the **Configure** button to make changes to files on the repo.



Select the files you need to download and click Apply in Gluon.

Click on the checkboxes next to files to either download or remove from the Explore workspace view. Files that will download to the Unity Editor turn green and files that you want to remove from Unity will turn red. Click **Apply** when done and then **Update workspace** so the files will be available in your version of Unity.

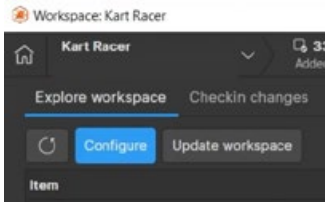
Even if you delete files inside of Gluon they only relate to this current changeset and any future changesets you create. The files will still be on the cloud as they will be referenced by previous changesets.



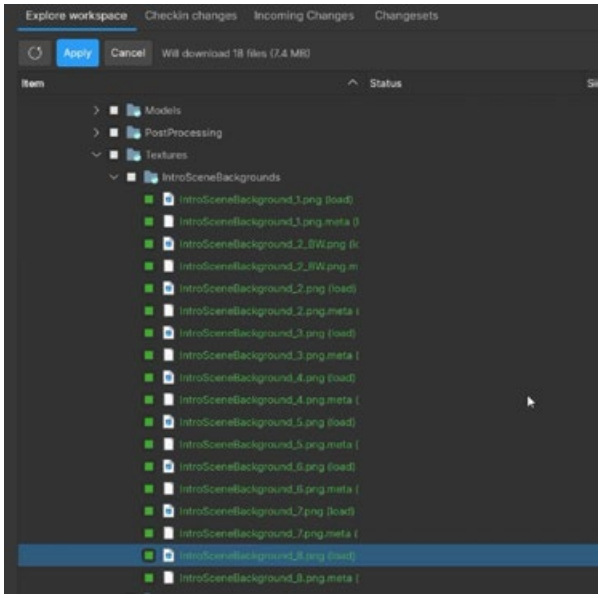
在 Gluon 中，您可以使用右上角的选项下拉菜单来启用深色主题。

Gluon 仅使用 main 分支，提供对艺术家友好的可视化工作区。您可以使用 Gluon 轻松访问“探索”工作区功能，以根据需要上传和下载文件.

在 Explore workspace （浏览工作区） 部分中，单击 Configure （配置） 按钮以更改存储库中的文件。



单击 Configure 按钮以更改存储库中的文件。

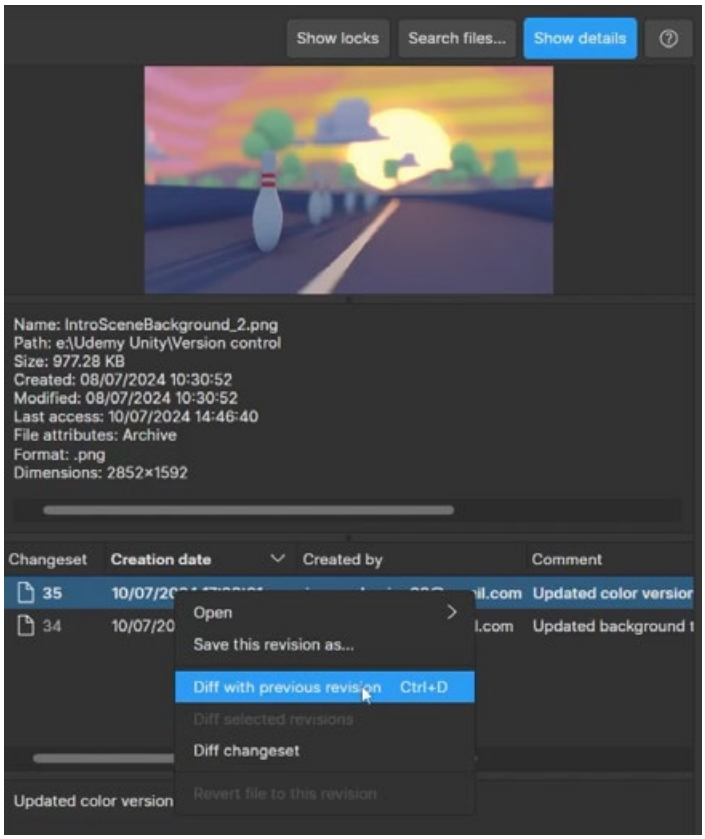


选择您需要下载的文件，然后单击 Apply in Gluon。

单击文件旁边的复选框，以下载或从 Explore 工作区视图中删除。将下载到 Unity Editor 的文件将变为绿色，要从 Unity 中删除的文件将变为红色.完成后单击 Apply，然后单击 Update workspace，以便文件在您的 Unity 版本中可用。

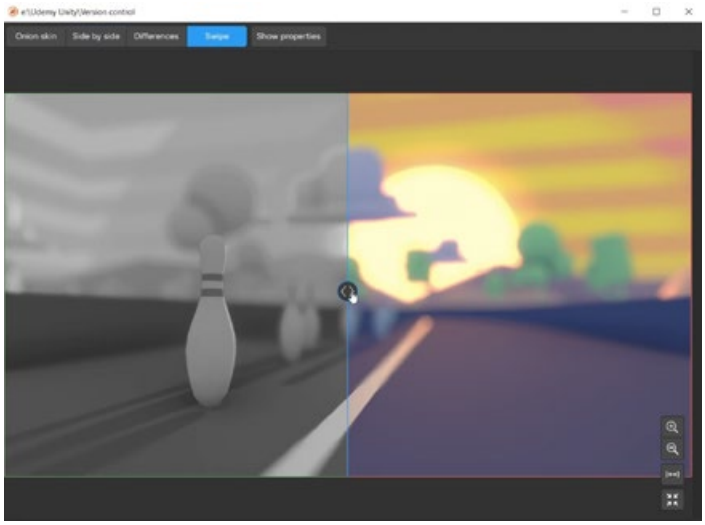
即使你删除了 Gluon 中的文件，它们也只与当前变更集和你创建的任何未来变更集相关。这些文件仍将位于云中，因为它们将被以前的变更集引用。

You can see the visual change of a texture that has numerous revisions by right-clicking on a revision in the options section on the right side of the Gluon app and selecting **Diff with previous revision**.



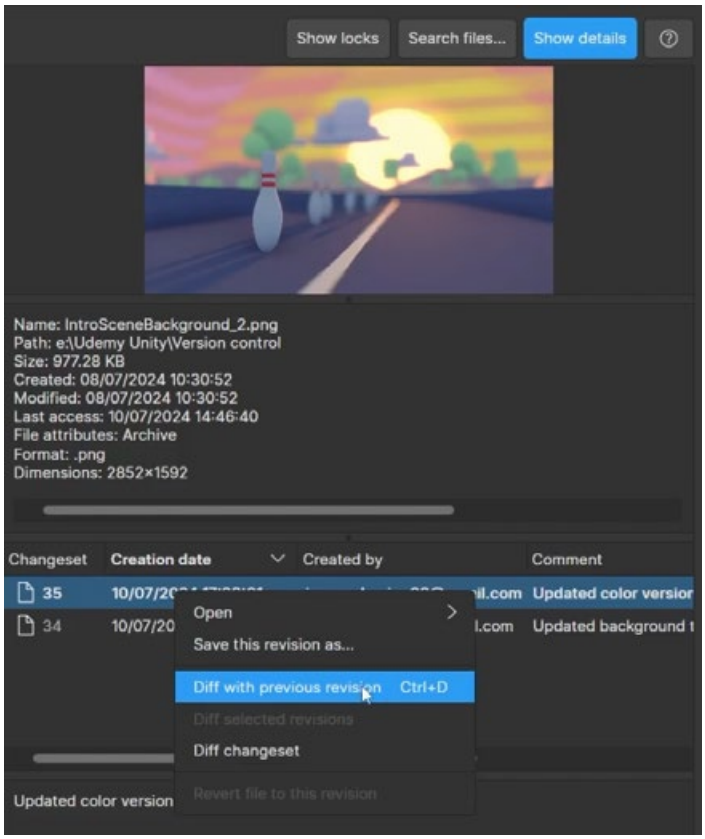
See a visual change between revisions by selecting **Diff with previous revision**.

This will show the differences between the previous revision and the current one.You have a few options in which to view this including swipe, which lets you drag a swipe slider along the image to view before and after.



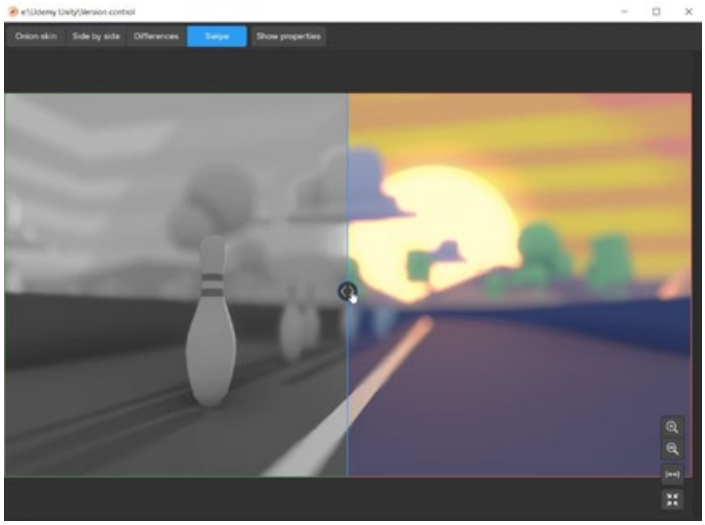
Swipe option in the Diff viewer

您可以通过右键单击 Gluon 应用程序右侧选项部分中的修订版，然后选择 Diff with previous revision ., 来查看具有大量修订版的纹理的视觉变化



通过选择 Diff with previous revision. 来查看修订之间的视觉更改

这将显示上一个修订版和当前修订版之间的差异。您有几个选项可以查看此版本，包括滑动，它允许您沿图像拖动滑动滑块以查看之前和之后。



差异查看器中的滑动选项

UVCS desktop app

The UVCS desktop app is the full, complete version control software suitable for team members who want access to branching. You can open it from the installed Unity DevOps Version Control applications.

If you used Gluon previously, it will prompt you to change from partial mode to full mode, the mode which UVCS runs in.

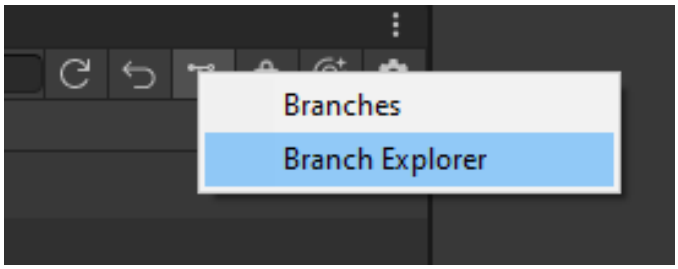
Change from partial mode to full mode for UVCS desktop version control software.

Branches

Branches allow different team members to work on the same project independently of each other. For example, a team member might create a new feature on a separate branch, while other team members work on the main branch. Features can be things like creating the code for achievements, adding the mechanics for different types of pickups, etc.

When they complete their task they can merge their branches back into the main branch. All team members working on the project will then be able to update their workspace to incorporate all the work that has been done.

When you select the Branch Explorer within the Version Control window in the Unity Editor, it opens the UVCS app and takes you to the Branches section.



You can open the Branch Explorer section of the UVCS desktop app.

UVCS 桌面应用程序

UVCS 桌面应用程序是完整、完整的版本控制软件，适用于希望访问分支的团队成员。您可以从已安装的 Unity DevOps 版本控制应用程序中打开它。

如果您以前使用过 Gluon，它会提示您从部分模式更改为完整模式，即 UVCS 运行的模式。

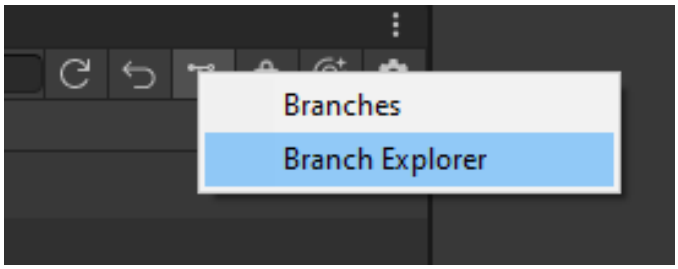
将 UVCS 桌面版本控制软件从部分模式更改为完整模式。

分支

分支允许不同的团队成员彼此独立地处理同一个项目。例如，团队成员可能在单独的分支上创建新功能，而其他团队成员则在主分支上工作。功能可以是创建成就代码、为不同类型的拾取物添加机制等。

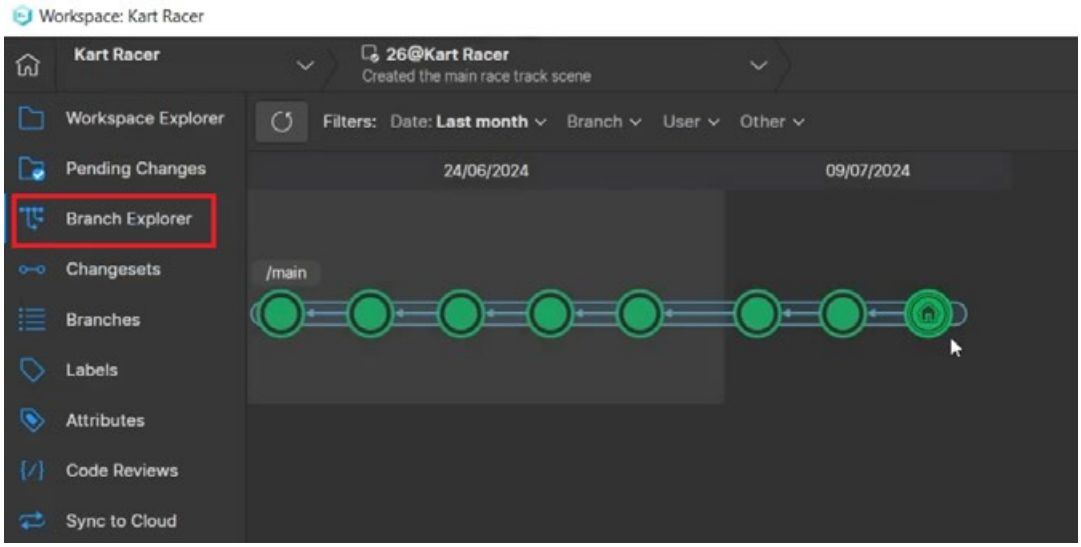
当他们完成任务时，他们可以将他们的分支合并回 main 分支。 然后，所有参与该项目的团队成员将能够更新他们的工作区以包含已完成的所有工作。

在 Unity Editor 的 Version Control 窗口中选择 Branch Explorer 时，它会打开 UVCS 应用程序并转到 Branches 部分。

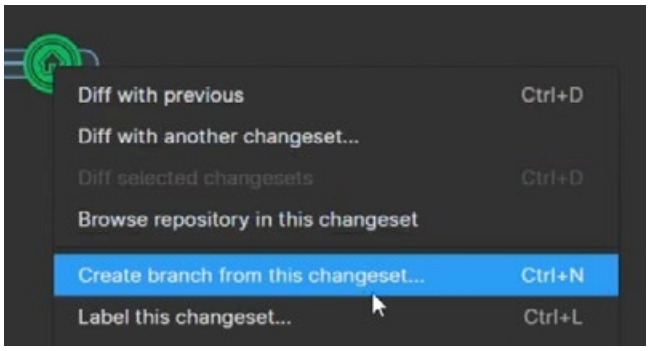


您可以打开 UVCS 桌面应用程序的 Branch Explorer 部分。

Each of the circles represents a changeset. Your current changeset has a house icon.



Right-click on a changeset and choose **Create branch from this changeset.**

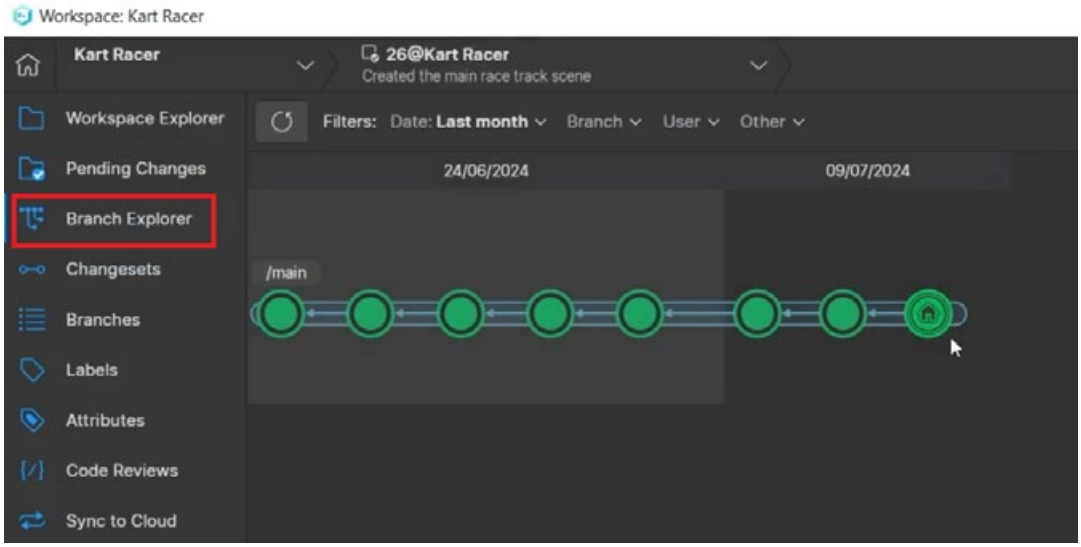


The Create a branch from a changeset option

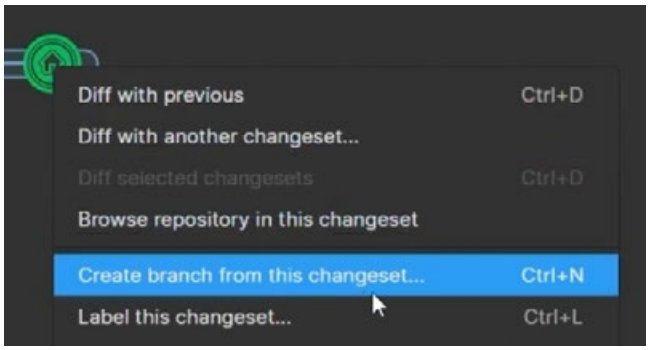
You will now be working on a separate branch. Any changes made by team members on the main branch or other feature branches will not be sent to you. This reduces the risk of someone else's changes conflicting with your progress. You can update their changes later after merging.

Branches can have multiple changesets marked by circles just like the main branch.

每个圆圈代表一个变更集。您当前的变更集有一个 house 图标。



右键单击变更集，然后选择 Create branch from this changeset.

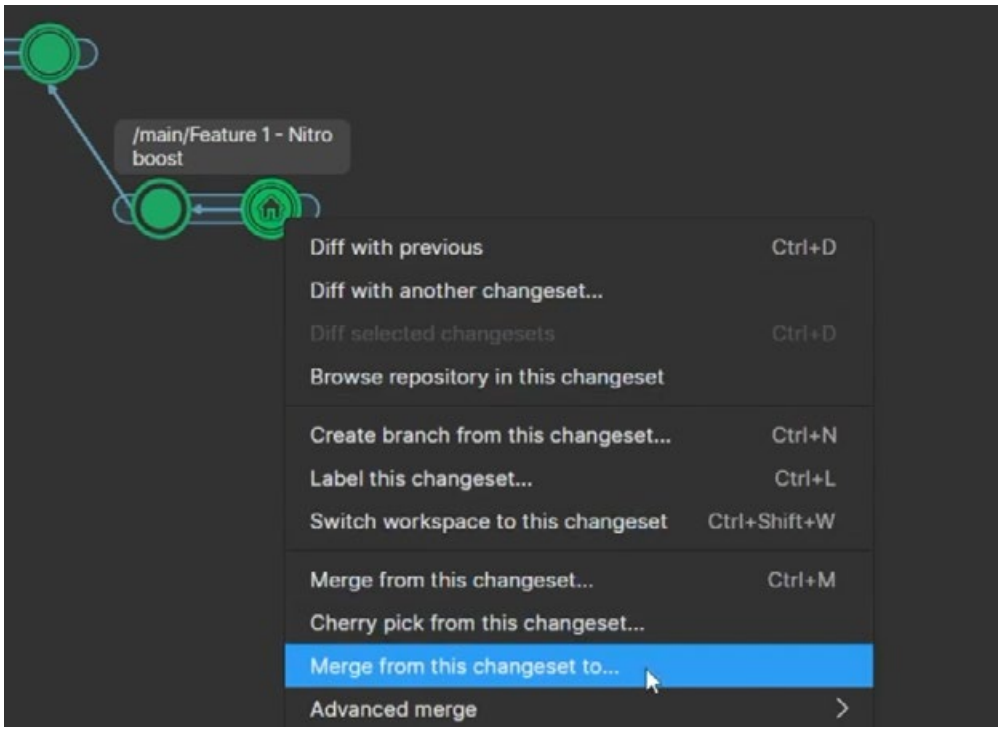


Create a branch from a changeset 选项

您现在将处理一个单独的分支。团队成员在 main 分支或其他功能分支上所做的任何更改都不会发送给您。这降低了其他人的更改与您的进度冲突的风险。您可以在合并后更新其更改。

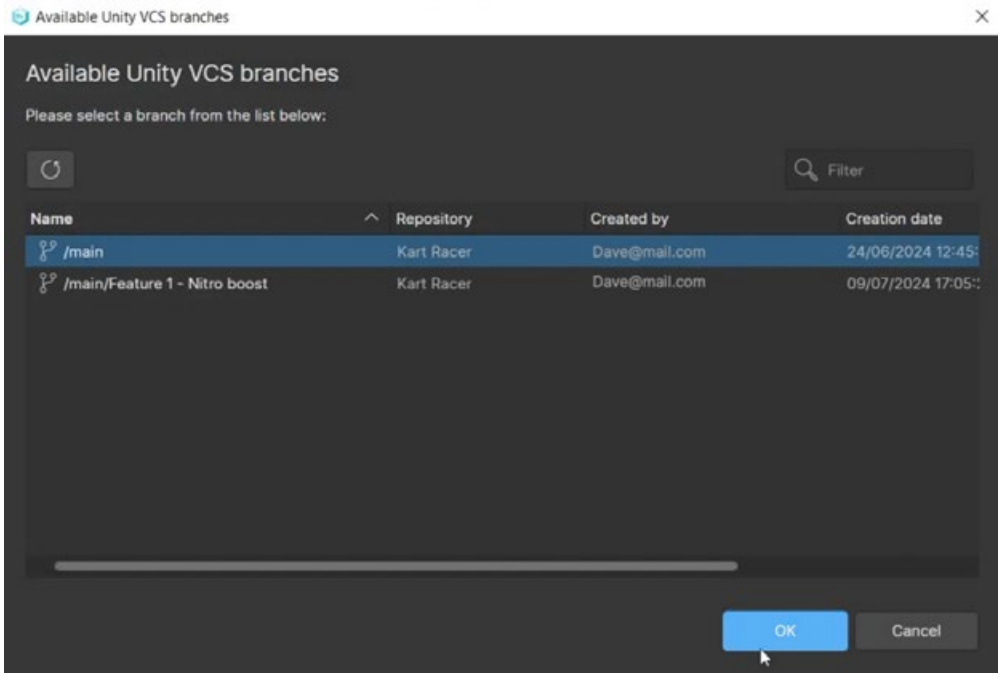
分支可以有多个由圆圈标记的变更集，就像主分支一样。

When you complete your work on the branch you can merge your branch back to the main one by right-clicking and choosing **Merge from this changeset to...**.



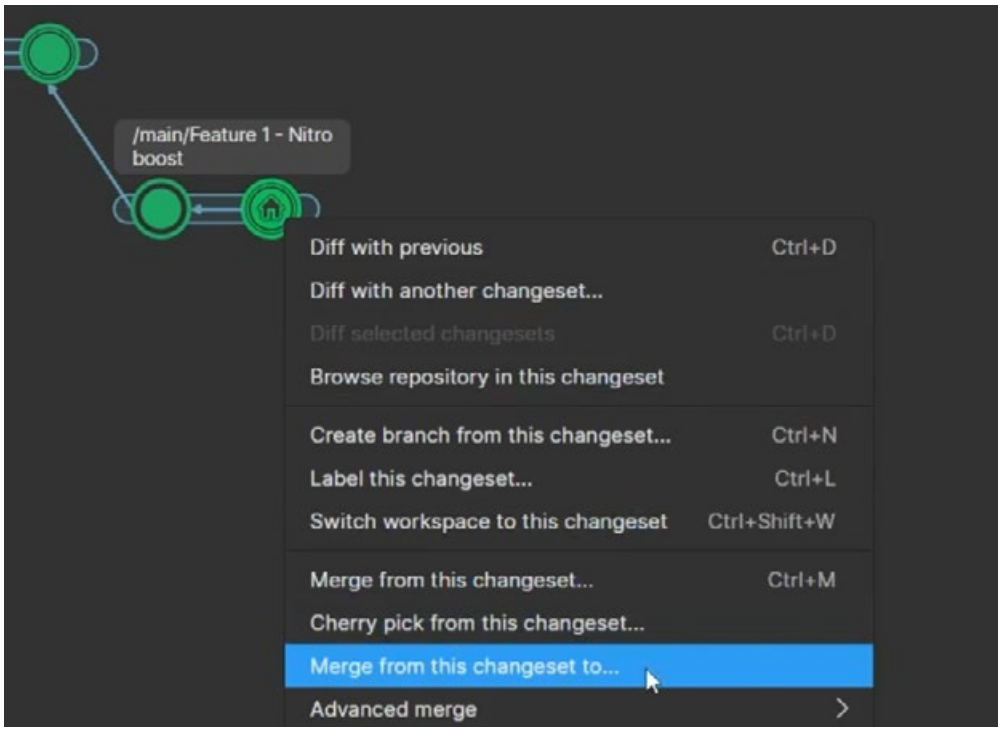
Merge a branch back into another branch such as Main.

It will display available branches. Select the branch you want to merge to.



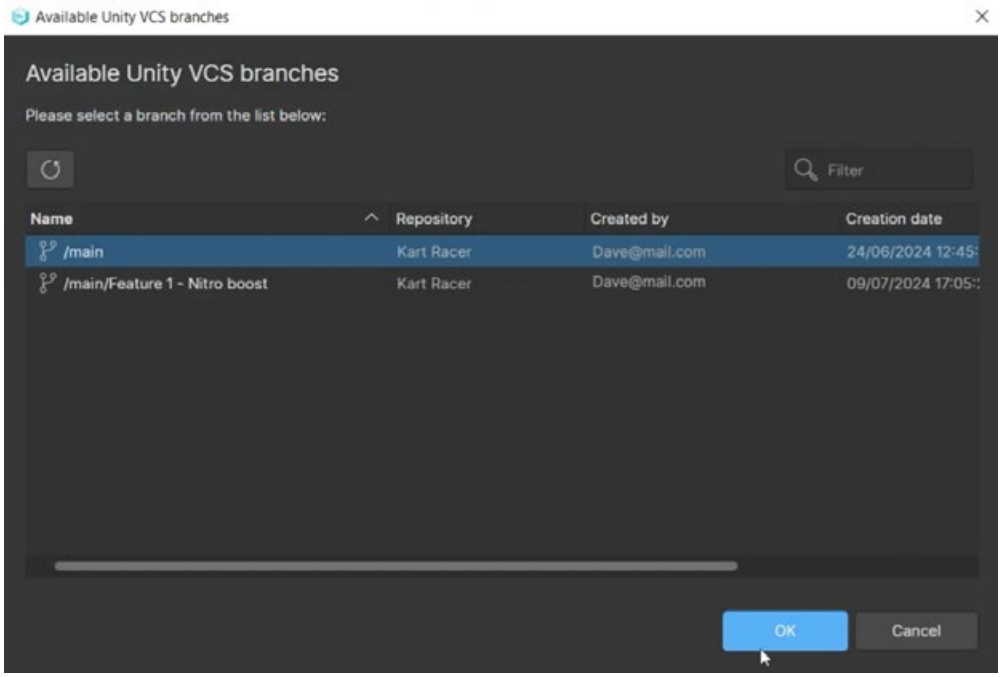
Available branches to merge to

完成分支上的工作后，可以通过右键单击并选择 Merge from this changeset to....



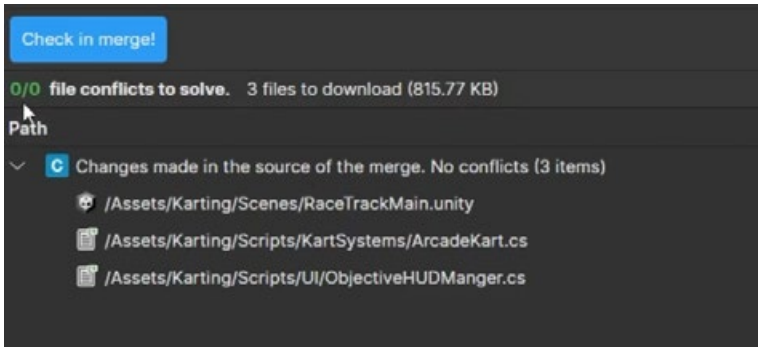
将一个分支合并回另一个分支，例如 Main.

它将显示可用的分支。选择要合并到的分支。



可合并到的分支

This opens the merge section and shows if there are any conflicts. If there are no conflicts, click the **Check in merge** button.

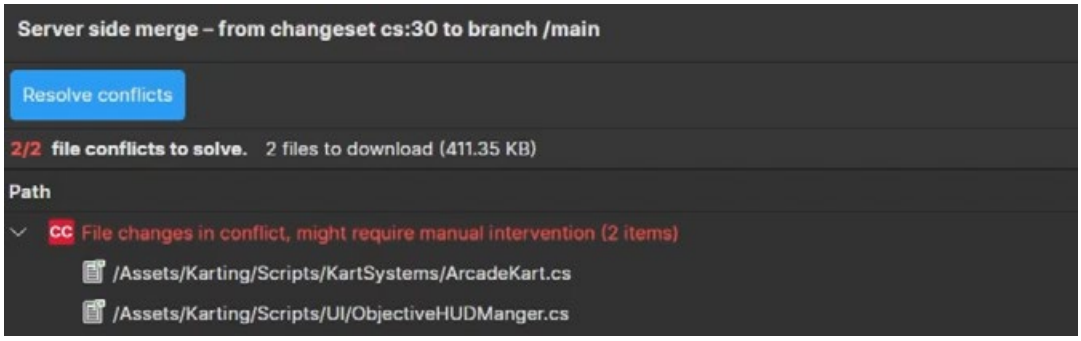


UVCS checks all files on your branch for any conflicts with the main branch.

If someone on the main branch has made changes to the same file or files that you have, this might cause conflicts when trying to merge.

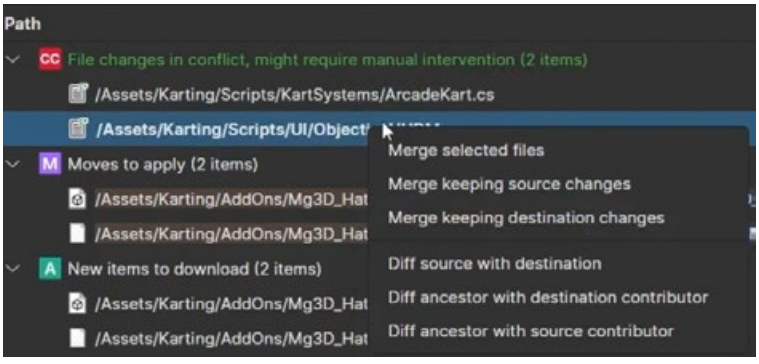
Handling conflicts

A script or a scene file might be changed by two or more people on different branches. Trying to merge them together will trigger a merge conflict.



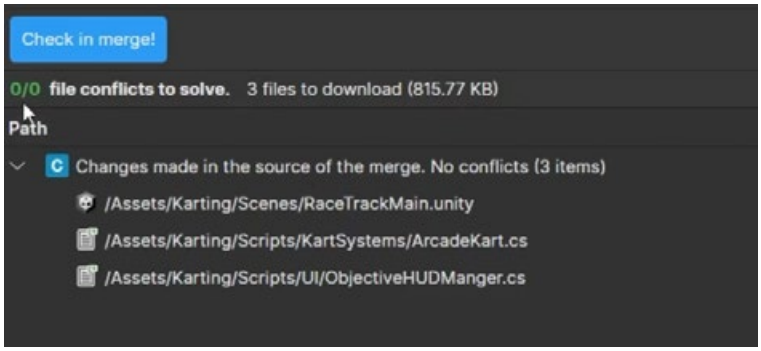
Two conflicts have been identified showing the files containing the conflicts.

Right-click on the files that have conflicts to show the differences and then decide if you want to keep or remove the changes.



Right-click menu showing the options available to handle a merge conflict

这将打开 merge 部分并显示是否存在任何冲突。如果没有冲突，请单击 Check in merge 按钮.

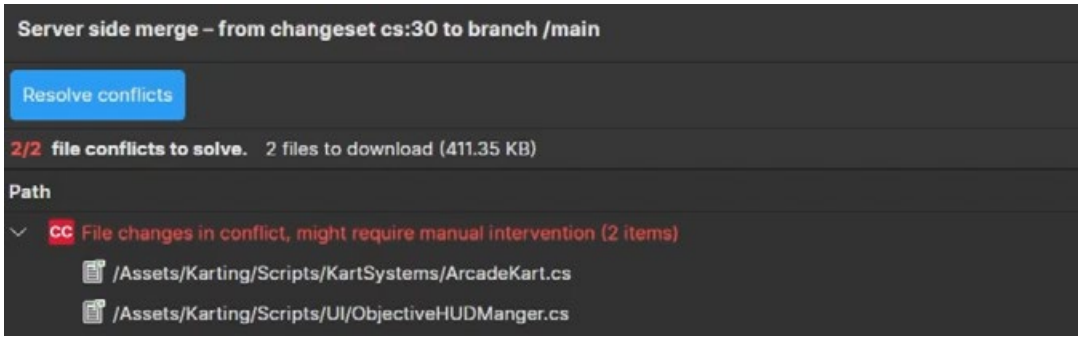


UVCS 会检查分支上的所有文件是否与 main 分支发生冲突。

如果 main 分支上的某人对您拥有的一个或多个文件进行了更改，则可能会在尝试合并时导致冲突。

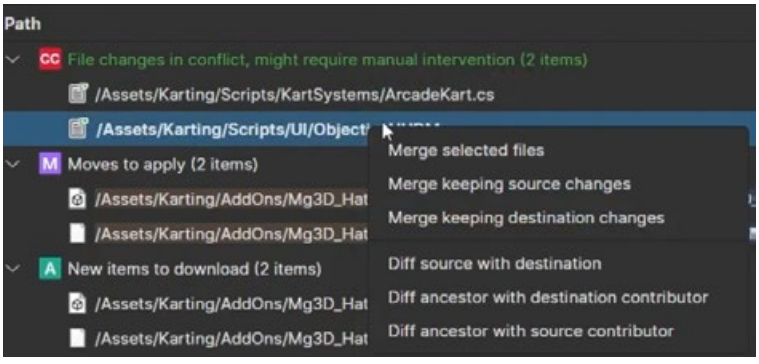
处理冲突

脚本或场景文件可能由不同分支上的两个或多个人更改。尝试将它们合并在一起将触发合并冲突。



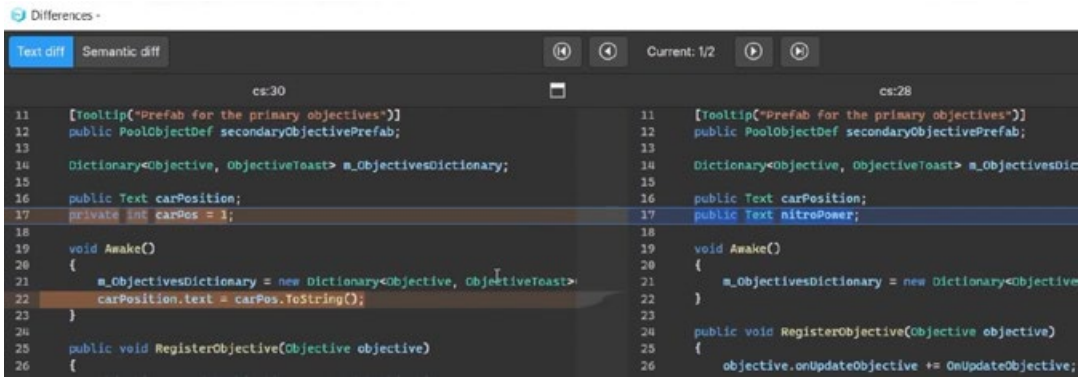
已确定两个冲突，其中显示了包含冲突的文件。

右键单击存在冲突的文件以显示差异，然后决定是要保留还是删除更改。



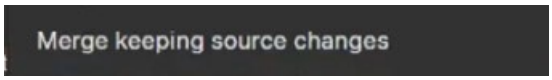
显示可用于处理合并冲突的选项的右键单击菜单

With a script you can choose **Diff source with destination**. This will open the Diff viewer so you can see how your file is different to the one on the branch you want to merge with.

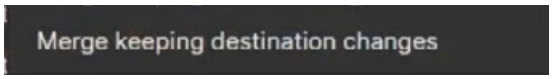


The Diff viewer shows the differences in the code between your file and the one on the branch you want to merge with.

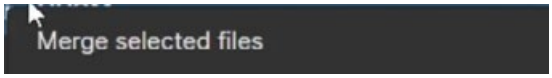
This can help you to decide whether to overwrite the file on the other branch with your file. To do this choose **Merge keeping source changes**:



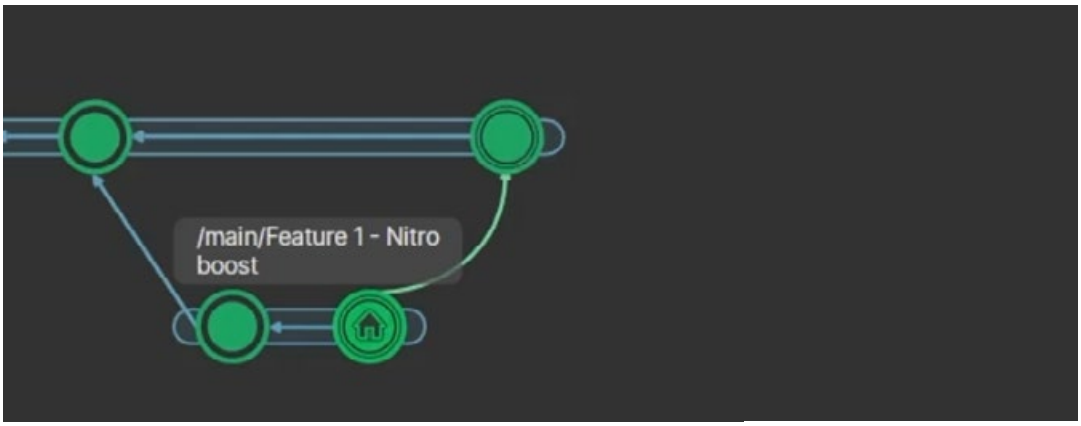
Or you can discard your changes and keep the original file on the other branch. To do this choose **Merge keeping destination changes**:



Finally, you can choose **Merge selected files** and UVCS will merge the two files together attempting to keep both sets of changes, although you should check the script after the merge to ensure it has done this correctly:

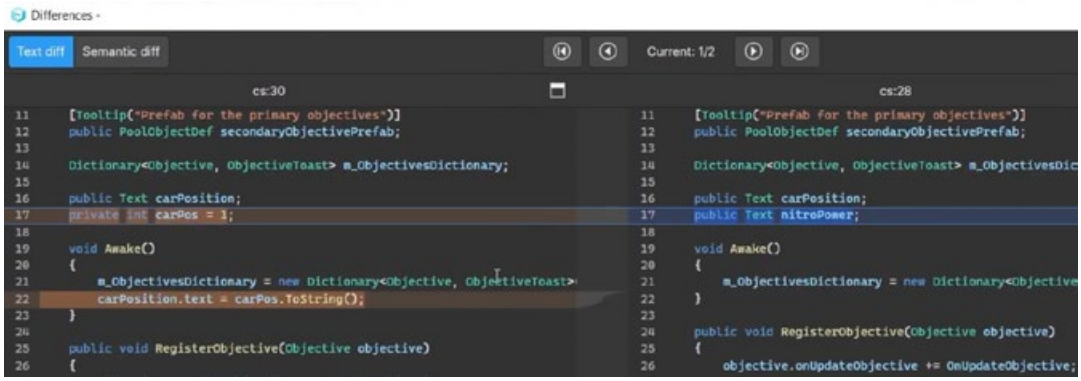


The Branch Explorer will now show which changesets have inherited merges from other branches.



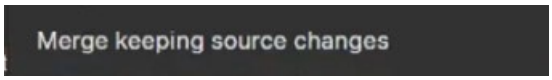
Branch merged back into Main

使用脚本，您可以选择 Diff source with destination （将源与目标进行比较）。这将打开 Diff 查看器，以便你可以看到你的文件与你想要合并的分支上的文件有何不同。

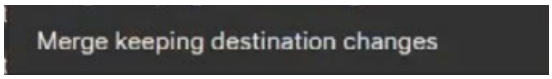


Diff 查看器 s 您的文件与所需分支上的代码之间的差异 以合并。

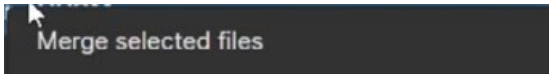
这可以帮助您决定是否使用您的文件覆盖另一个分支上的文件。为此，请选择 Merge keep source changes（合并保留源更改）：



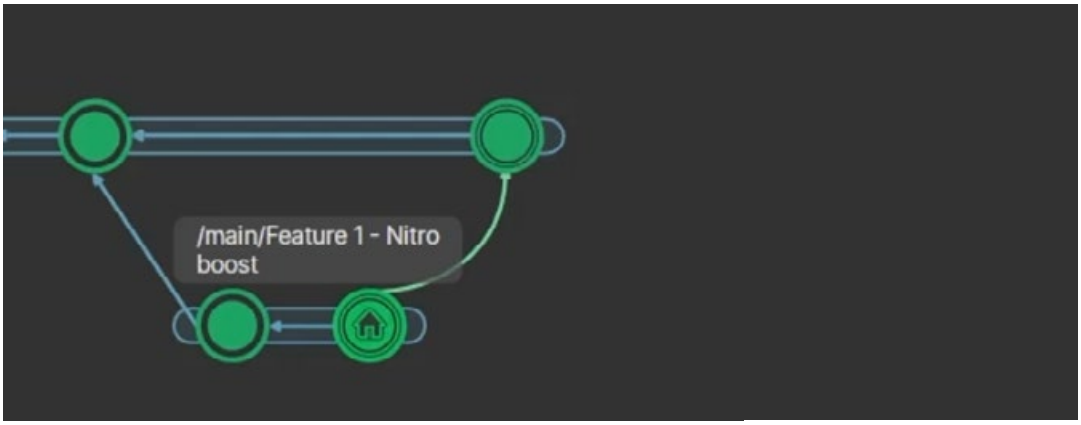
或者，您可以放弃更改并将原始文件保留在另一个分支上。为此，请选择 Merge keep destination changes（合并保留目标更改）：



最后，您可以选择 Merge selected files（合并所选文件），UVCS 会将两个文件合并在一起，尝试保留两组更改，但您应该在合并后检查脚本以确保它已正确完成此作：

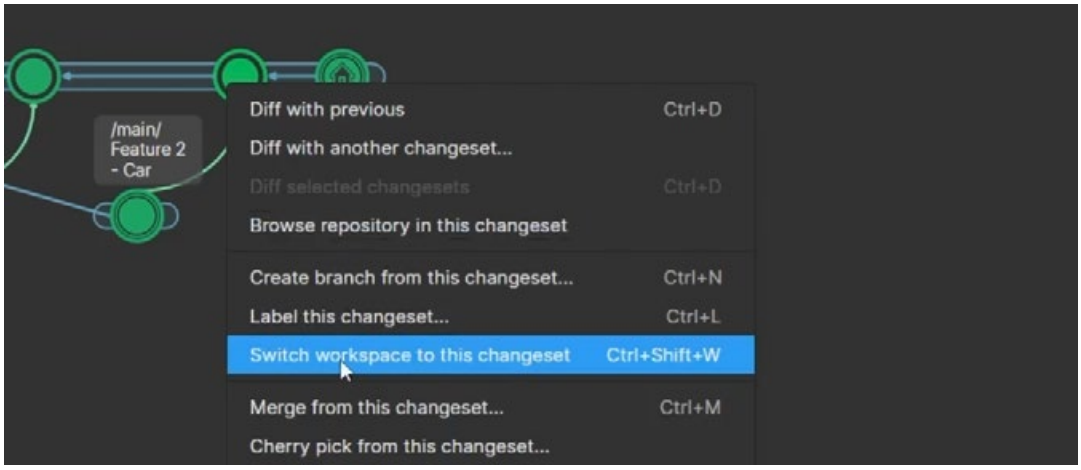


Branch Explorer 现在将显示哪些变更集继承了其他分支的合并.



Branch 合并回 Main

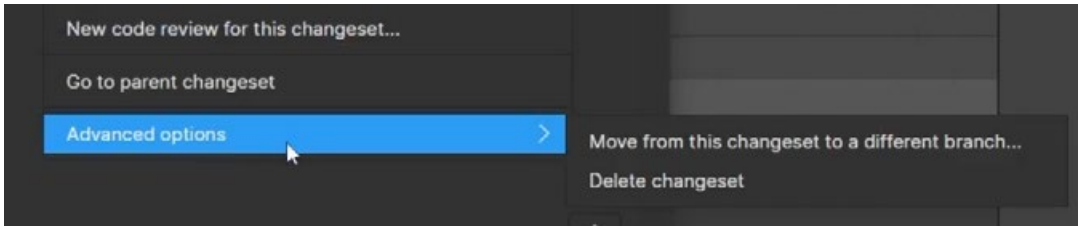
You can also right-click on any of the previous changesets and choose **Switch workspace to this changeset** to revert back to that point. This may be necessary if you added items or scripts that disrupted the game or caused issues.



You can switch back to any previous changeset by right-clicking on the changeset of choice.

Changesets that cause serious errors or bugs can be deleted. These should not have any changesets leading forward from them, or labels or sub branches.

Right-click on a changeset with errors and go to **Advanced options > Delete changeset**.

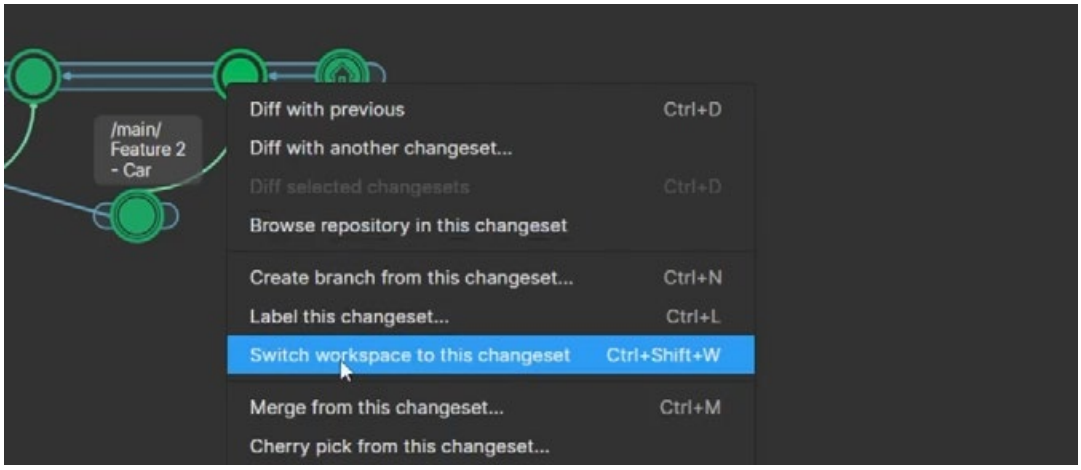


Delete a changeset from the right-click menu.

Merge rules

A good idea is to implement merge rules to prevent merge conflicts from disrupting your team's workflow. This can be used in larger teams where team managers want to approve a branch before allowing it to be merged into the main branch. This prevents introducing errors into the main workflow.

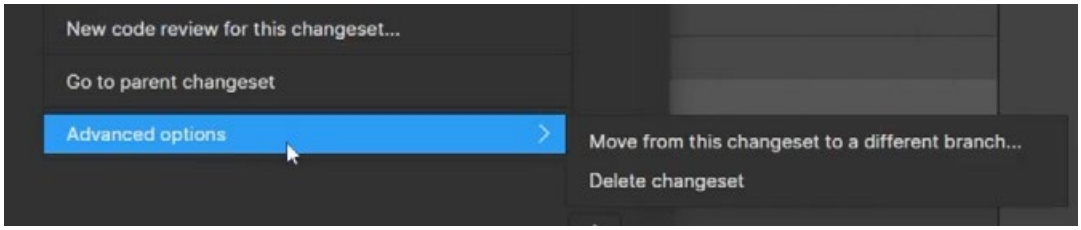
您还可以右键单击任何以前的更改集，然后选择 Switch workspace to this changeset （将工作区切换到此更改集） 以恢复到该点。如果您添加的项目或脚本会破坏游戏或导致问题，则可能需要这样做。



您可以通过右键单击所选变更集来切换回任何以前的变更集。

导致严重错误或 bug 的变更集可以删除. 这些变更集不应有任何从它们向前的变更集，也不应有标签或子分支。

右键单击有错误的变更集，然后转到高级选项 **>** 删除变更集.

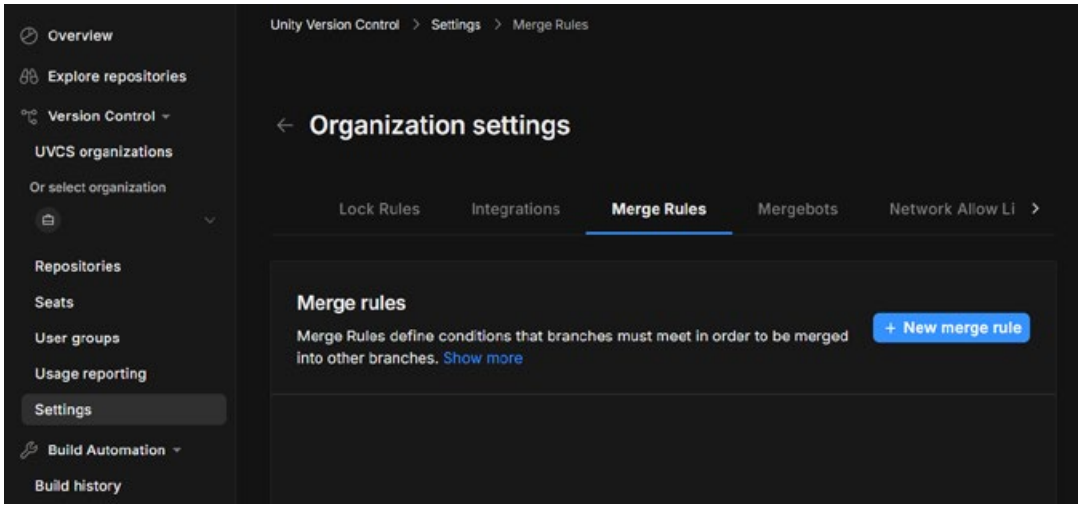


从右键单击菜单中删除变更集。

合并规则

一个好主意是实施合并规则，以防止合并冲突中断团队的工作流程。这可用于团队经理希望在允许将分支合并到主分支之前批准分支的大型团队中。这可以防止将错误引入主工作流程。

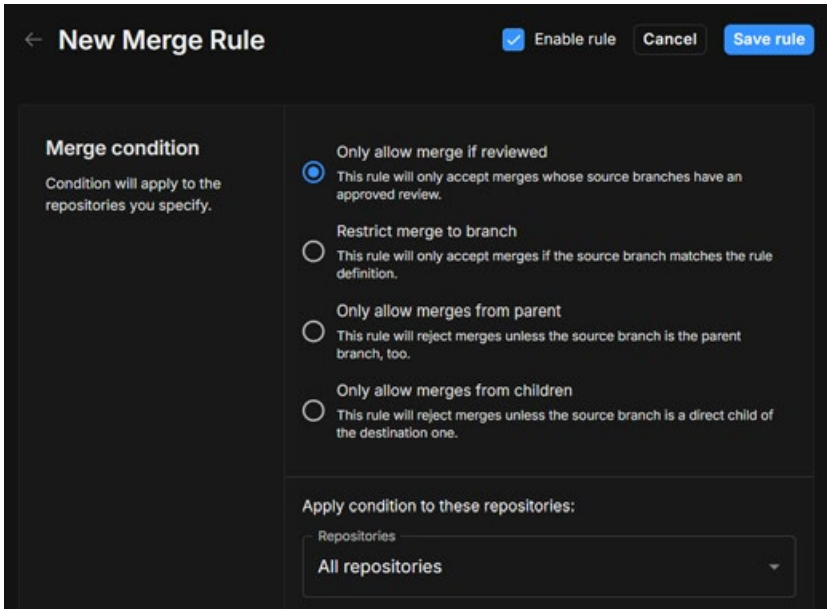
Log into the Unity Cloud dashboard, go to **settings**, then **Merge rules**.



Create a new merge rule from the Unity Cloud settings.

Click the **New merge rule** button and select from one of the four options. **Only allow merge if reviewed** will ensure that a merge can only take place after being reviewed and approved by other team members, or team managers.

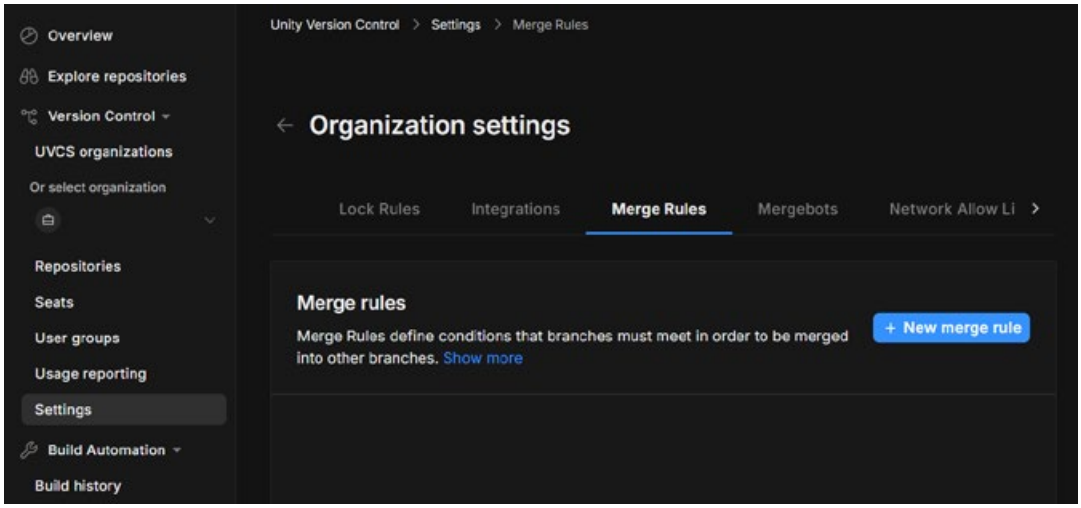
You can also apply this to all repos if you have multiple that you are working on, or specify a specific repo from the drop down list.



The option **Only allow merge if reviewed** requires other team members to review your branch before merge.

Specify which branches you would like the merge rules to apply to. It needs a source branch, which is the branch you are working on, and a destination branch, the branch you want to merge to. Click the **Save rule** button and it will now be active. At any point you can disable or delete a rule from the Unity Cloud.

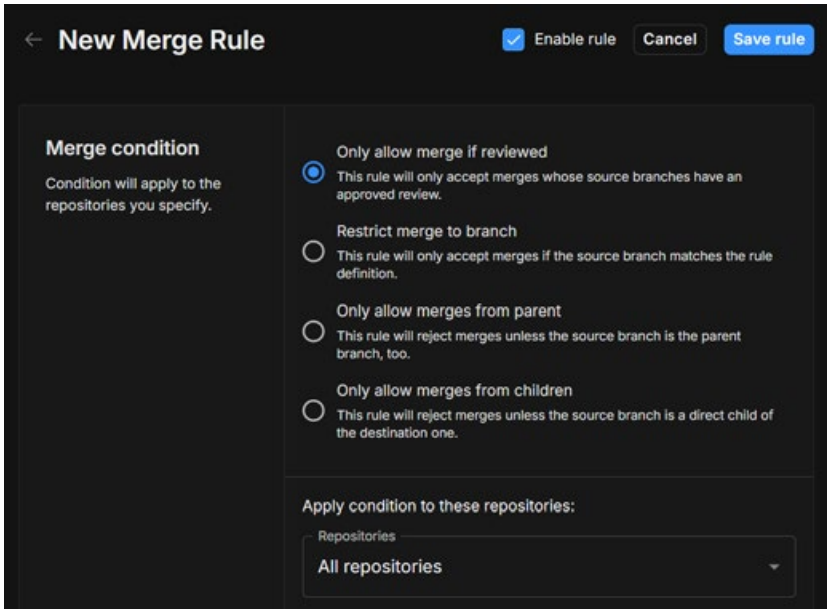
登录 Unity Cloud 控制面板，转到 settings，然后 Merge rules.



从 Unity Cloud 设置创建新的合并规则。

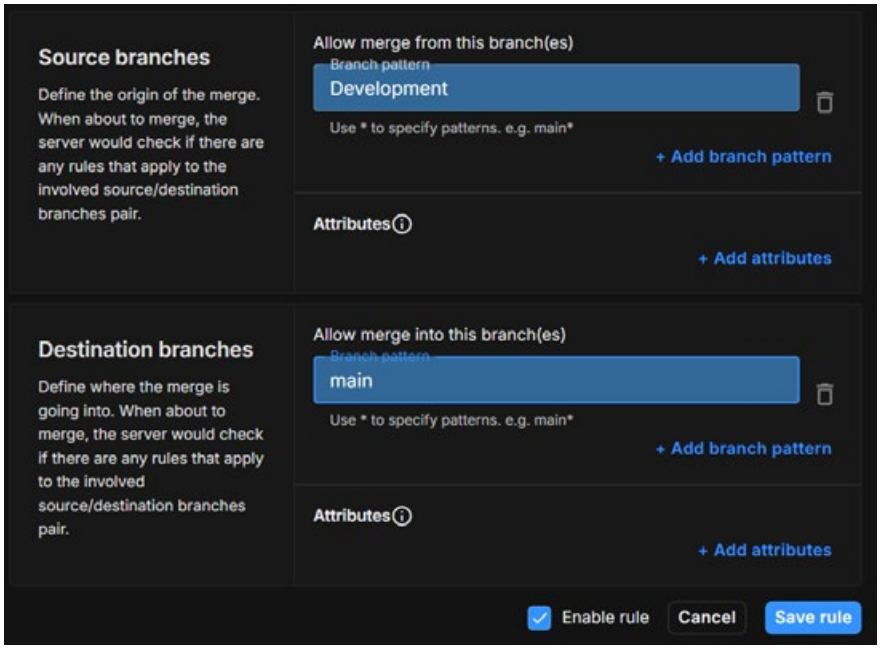
单击 New merge rule 按钮，然后从四个选项中选择一个。Only Allow merge if reviewed 将确保合并只能在其他团队成员或团队经理审核和批准后进行。

如果您正在处理多个存储库，您还可以将其应用于所有存储库，或者从下拉列表中指定特定存储库。



选项 Only allow merge if reviewed 要求其他团队成员在合并前查看您的分支。

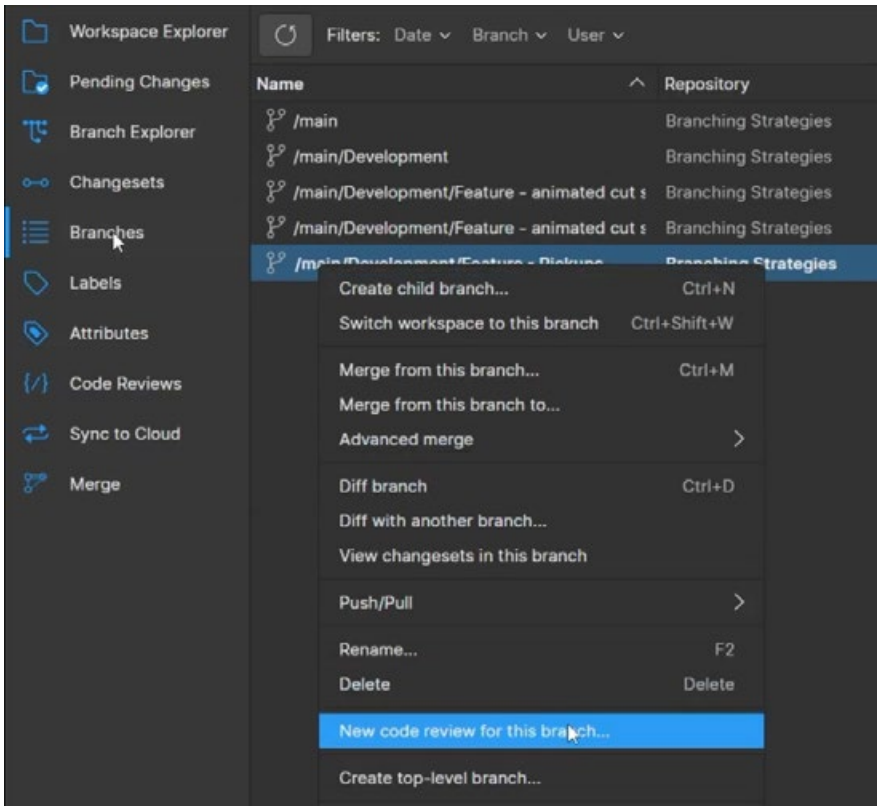
指定要将合并规则应用于的分支。它需要一个源分支，即您正在处理的分支，以及一个目标分支，即您要合并到的分支。点击 Save rule 按钮，它现在将处于活动状态。您可以随时从 Unity Cloud 中禁用或删除规则。



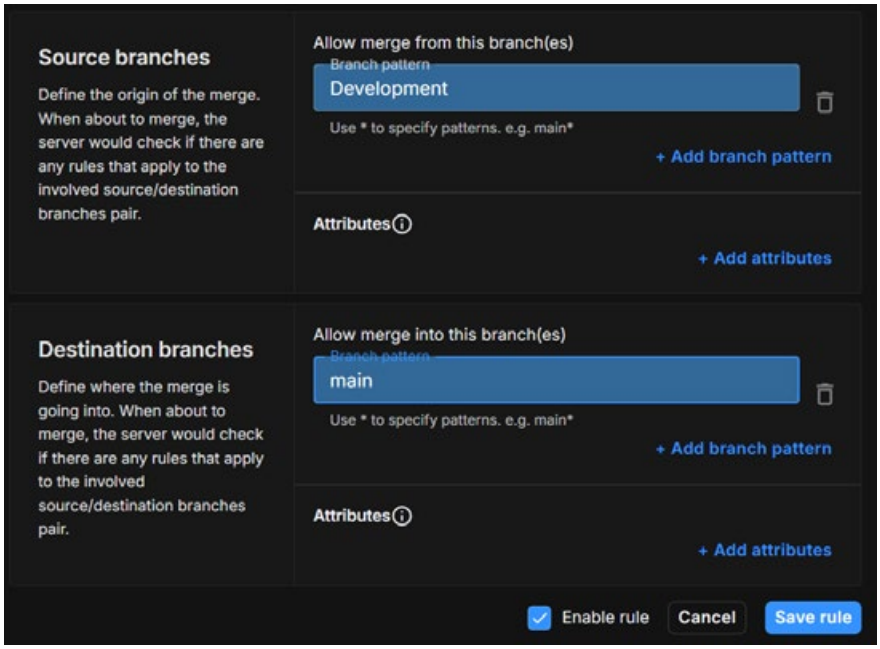
Include which branches the merge rule will apply to.

Team members wanting to merge their branch will have to request a branch review.

Go to the Branches section and right-click on the branch you want to merge. Choose the option **New code review for this branch.**



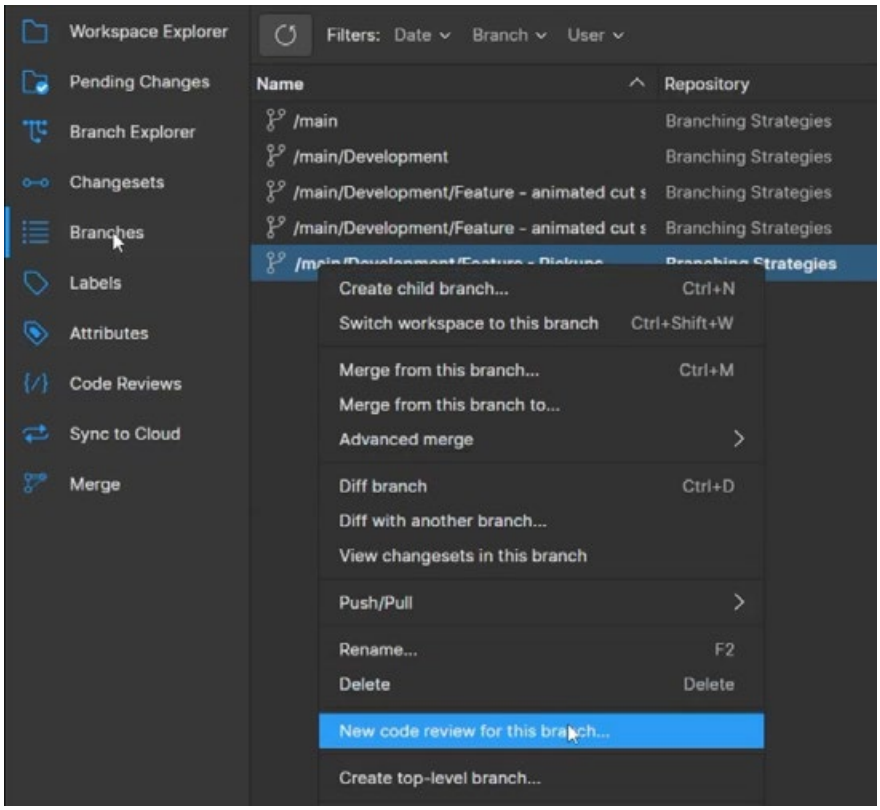
Request a code review for a branch you want to merge.



包括合并规则将应用于的分支。

想要合并其分支的团队成员必须请求分支审核。

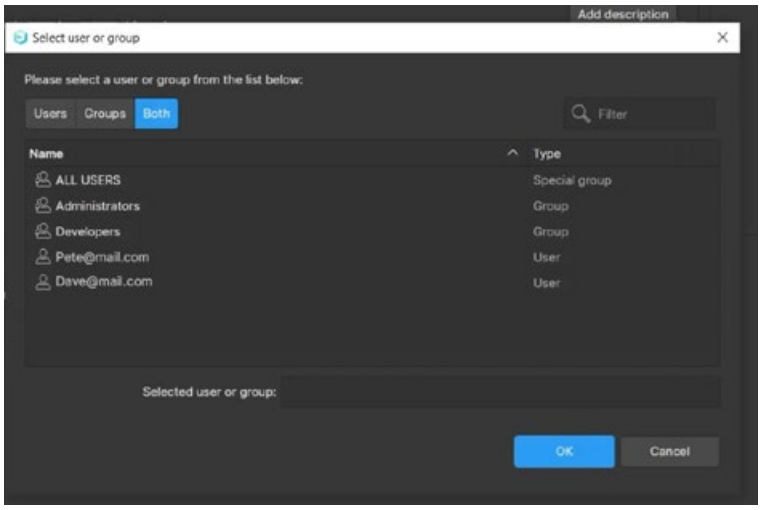
转到 Branches 部分并右键单击要合并的分支。选择选项 New code review for this branch.



请求对要合并的分支进行代码审查.

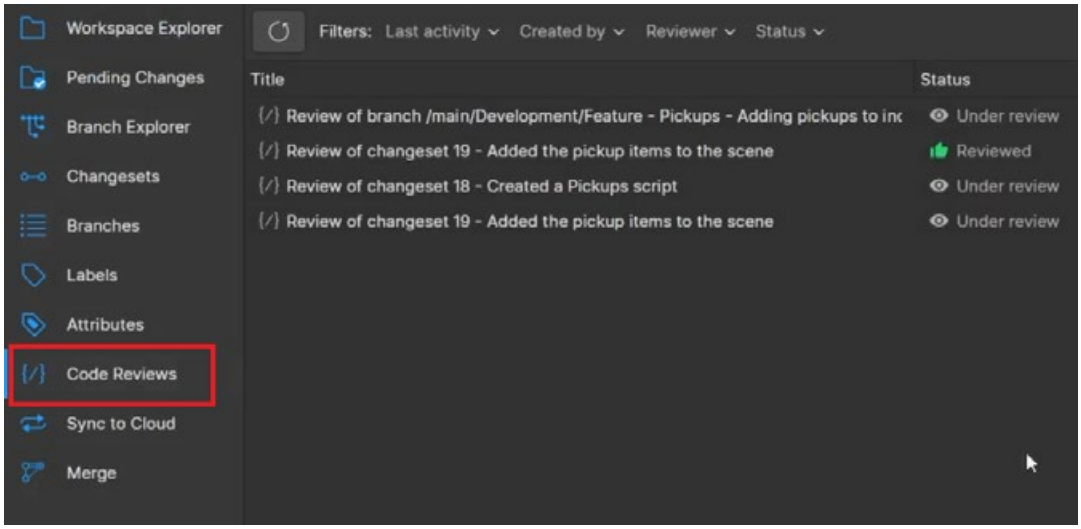
This will contain all the files that have been changed over the course of the entire branch.

Click on **conversations** to leave a message and invite other team members to review your work. Note that each team member must review the branch before you can merge, so ensure to include only the relevant people. They will then receive an email message to review your code.



Choose individuals or groups members to review your branch.

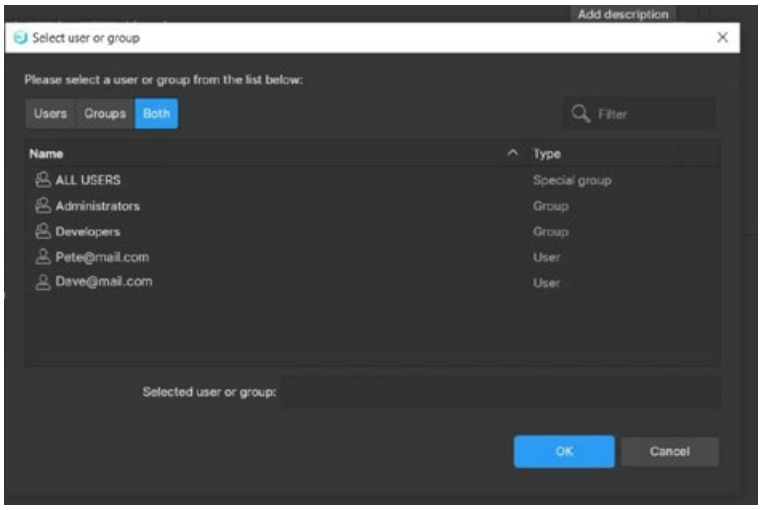
Go to **Code Reviews** to see the progress of the review. Double-click to open it to see if any messages have been left for you.



Code reviews show the ongoing progress of requested reviews.

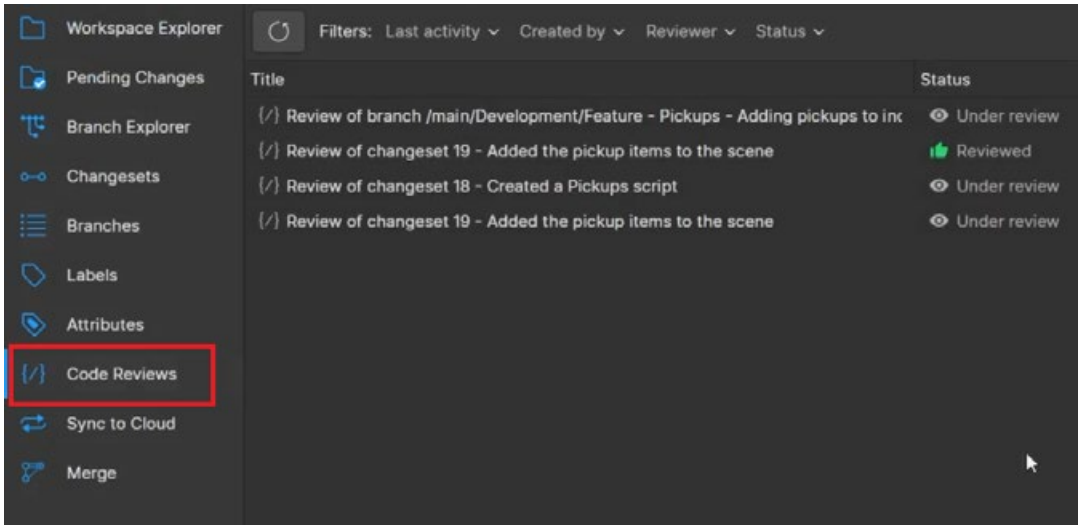
这将包含在整个分支过程中更改的所有文件。

单击 对话 留言并邀请其他团队成员查看您的工作。请注意，每个团队成员必须先审核分支，然后才能合并，因此请确保仅包含相关人员。然后，他们将收到一封电子邮件，以检查您的代码。



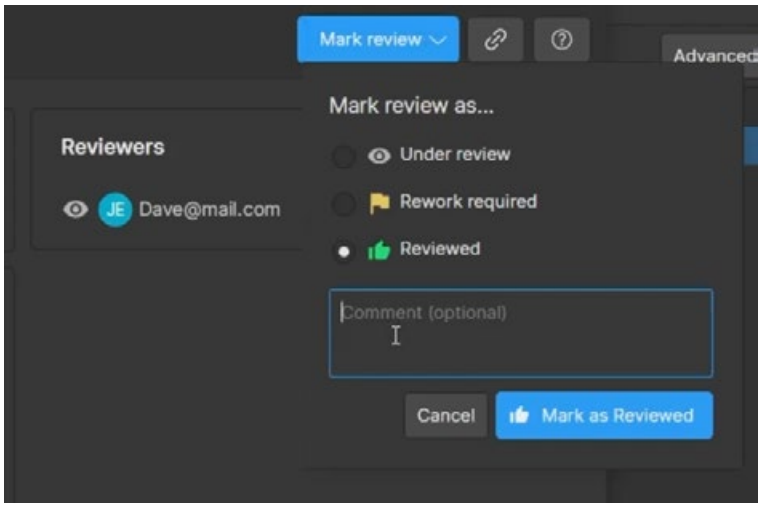
选择个人或组成员以查看您的分支。

转到 Code Reviews（代码审查）以查看审查的进度。双击将其打开，查看是否为您留下了任何消息。



代码评审显示请求的评审的持续进度。

The invited team members can then mark the branch as either **Rework required** or **Reviewed** and leave a message.

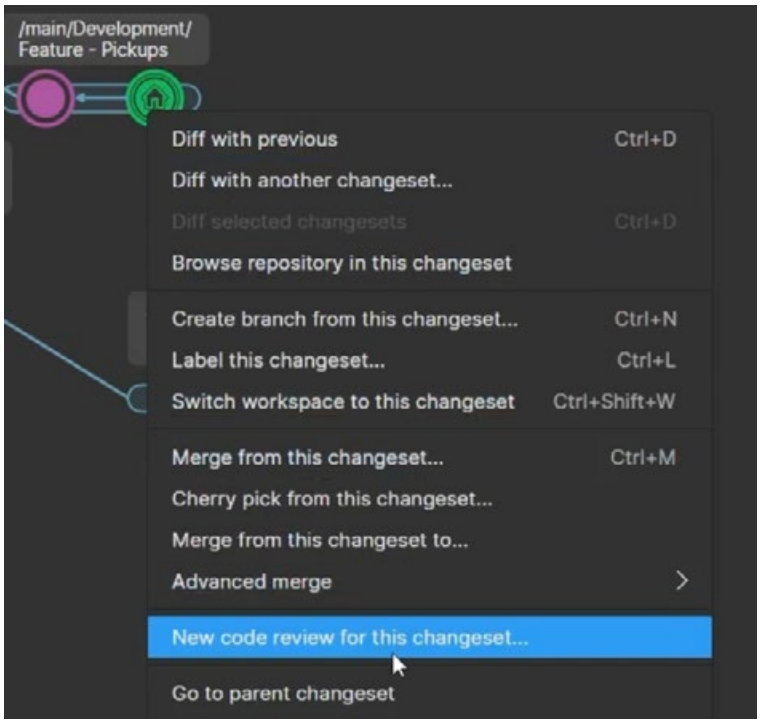


Reviewers can click the **Mark Review** button to mark it as rework required or reviewed.

Once the branch is reviewed it can then be merged. This adds extra security to your project.

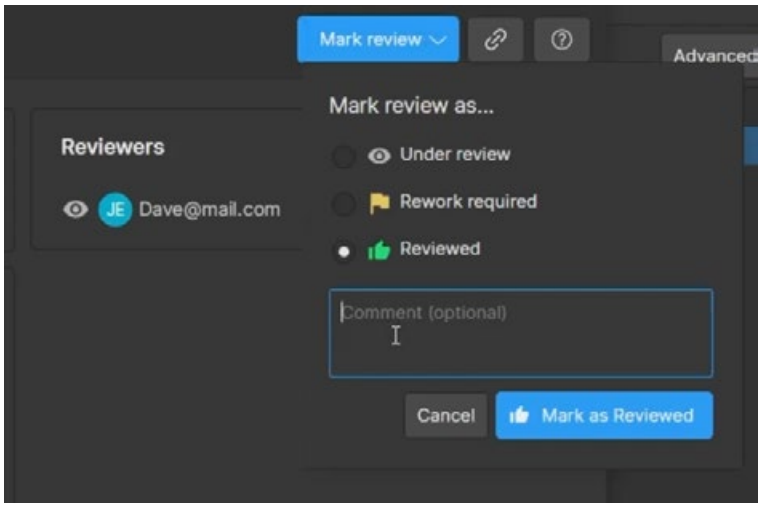
You can also do the same procedure for code reviews on a specific changeset. This might be useful when you want other team members to review your code for bugs or issues, or perhaps suggest ways to improve the code.

To do this, right-click on a changeset and choose **New code review for this changeset**. You can then invite team members to check your files and give recommendations.



Requesting a code review for a changeset

然后，受邀的团队成员可以将分支标记为 Rework required （需要返工） 或 Reviewed （已审核） 并留言。

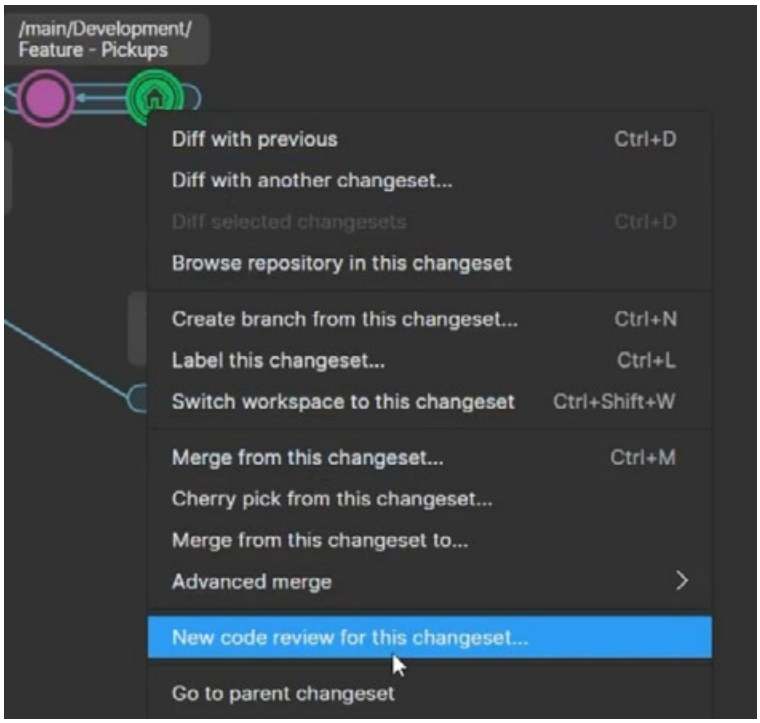


审阅者可以单击 Mark Review 按钮将其标记为 rework required 或 reviewed。

一旦 Branch 被审查，就可以合并它。这为您的项目增加了额外的安全性。

您还可以对特定变更集的代码审查执行相同的过程。当您希望其他团队成员检查您的代码中是否存在 bug 或问题，或者建议改进代码的方法时，这可能很有用。

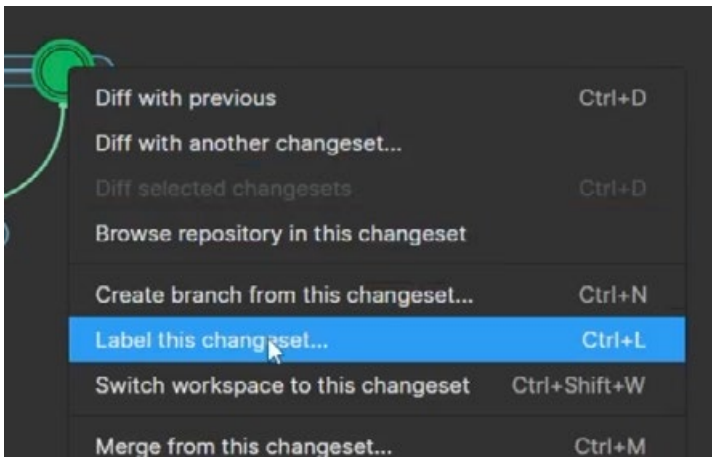
为此，请右键单击变更集，然后选择 New code review for this changeset （此变更集的 New code review）。然后，您可以邀请团队成员检查您的文件并提供建议。



请求对变更集进行代码审查

You can also add labels to a changeset. This might be useful if the changeset is an approved version e.g., version 1.0, or if it's a stable changeset and can be used for branching.

Right-click on the changeset and choose **Label this changeset**. Labels prevent the changeset from being deleted. You can remove a label from the labels section if necessary.



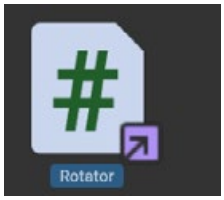
Label a changeset from the right-click menu.

Labels give other users important information about this changeset.

Locking files

Locking a file while you are working on it can prevent major merge conflicts. This file will then be locked on the cloud, preventing other team members from making changes to it while it's locked (they can still access the file in a read-only format). Once the file is checked back in, the lock is removed and other team members can make changes to it.

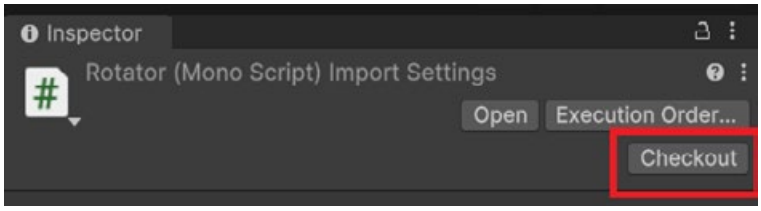
Locks are not automatically applied. By default, when you save your progress in Unity, any files you are working on will be marked with a checkout icon.



A purple checkout icon appears on the file icon in Unity and also on everyone else's versions, letting them know that somebody is working on the file.

The checkout icon in a purple box is displayed on the file on every team member's computer.

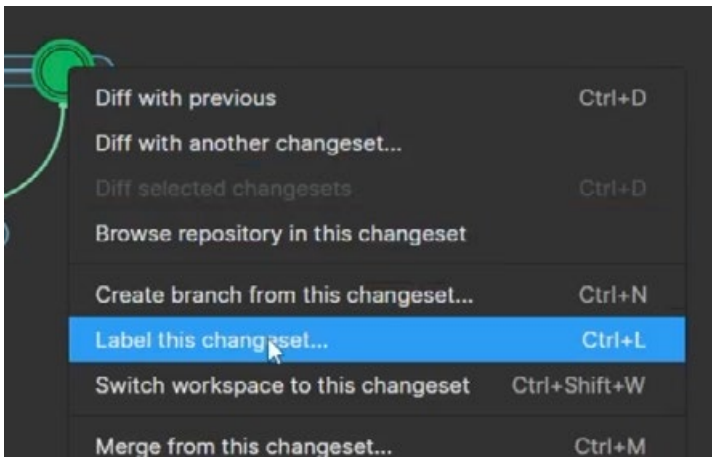
You can manually checkout a file before you begin working on it. Click the **Checkout** button in the Inspector. Other team members will see that you are working on the file, however, this is not a file lock, so other team members can still make their own changes to the same file. This is not as effective as using file locking.



Checkout a file when you are making changes to warn other team members that you are working on it.

您还可以向变更集添加标签。如果变更集是已批准的版本（例如 1.0 版），或者它是一个稳定的变更集并且可以用于分支，这可能很有用。

右键单击变更集，然后选择 Label this changeset （标记此变更集）。标签可防止删除变更集。如有必要，您可以从 labels 部分删除标签。



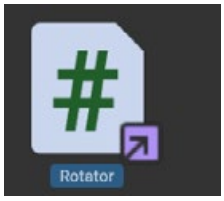
从右键单击菜单中标记变更集。

标签为其他用户提供重要信息
关于此 changeset.

锁定文件

在处理文件时锁定文件可以防止重大合并冲突。然后，此文件将被锁定在云中，以防止其他团队成员在锁定时对其进行更改（他们仍然可以以只读格式访问文件）。重新签入文件后，锁定将被删除，其他团队成员可以对其进行更改。

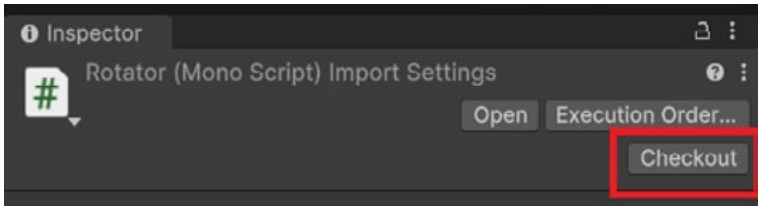
锁不会自动应用。默认情况下，当您在 Unity 中保存进度时，您正在处理的任何文件都将标有签出图标。



一个紫色的签出图标出现在 Unity 中的文件图标上，也出现在其他所有人的版本上，让他们知道有人正在处理该文件。

紫色框中的结帐图标显示在每个团队成员的 comp 上的文件上　　子宫

您可以在开始处理文件之前手动签出文件。单击 Inspector 中的 Checkout 按钮。其他团队成员将看到您正在处理该文件，但是，这不是文件锁定，因此其他团队成员仍然可以对同一文件进行自己的更改。这不如使用文件锁定有效。

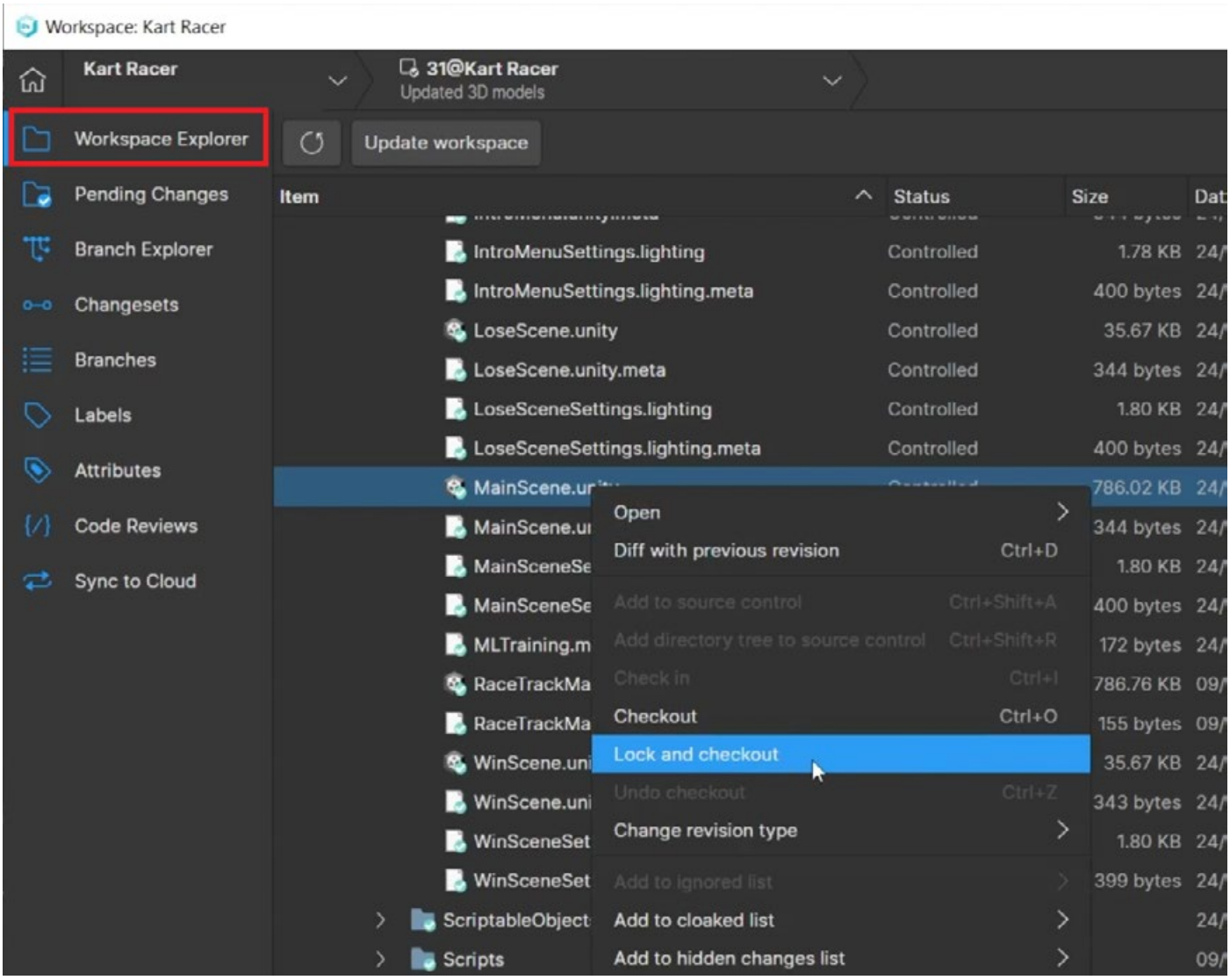


中岩您进行更改时，请 eckout 一个文件以警告其他团队成员您正在　　在上面嬉戏。

UVCS uses smart locks. This ensures that other team members who want to make changes to this file will be forced to use the latest revision of the file. This is important, as you may be working on a branch that is using an older version of that file. Smart locks checks all branches including the main to find the latest revision of this file and ensures you are working from that version and not an outdated older version.

To lock files, go into the Branch Explorer.

In the **Workspace Explorer** view find the file you want to lock, right-click on it and choose **Lock and checkout**. This only works as long as no other team member has checked the file out.

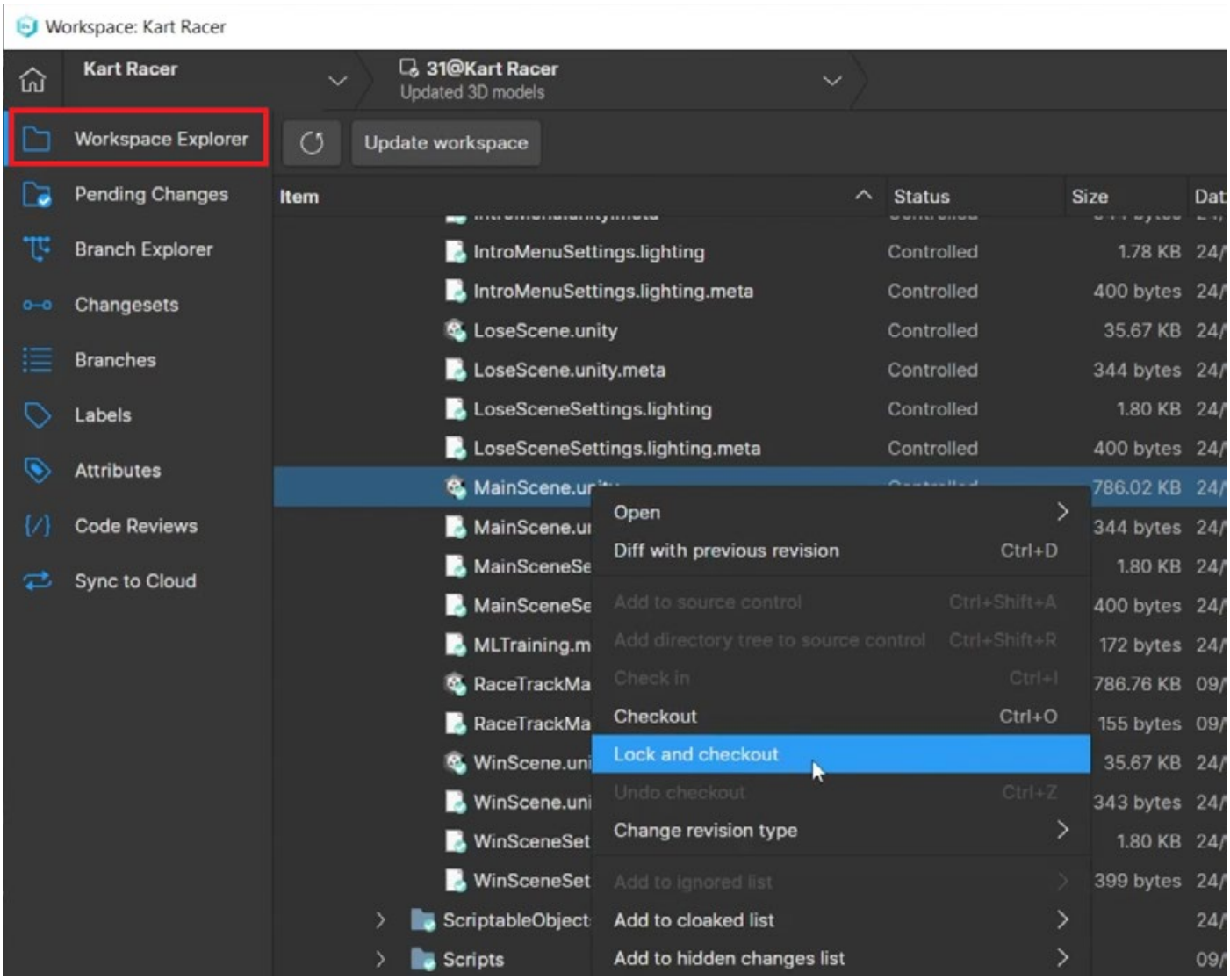


Lock files from the UVCS desktop app in the Workspace Explorer.

UVCS 使用智能锁。这可确保要更改此文件的其他团队成员将被迫使用该文件的最新版本。这很重要，因为您可能正在处理使用该文件的旧版本的分支。智能锁会检查包括主分支在内的所有分支以查找此文件的最新版本，并确保您使用的是该版本，而不是过时的旧版本.

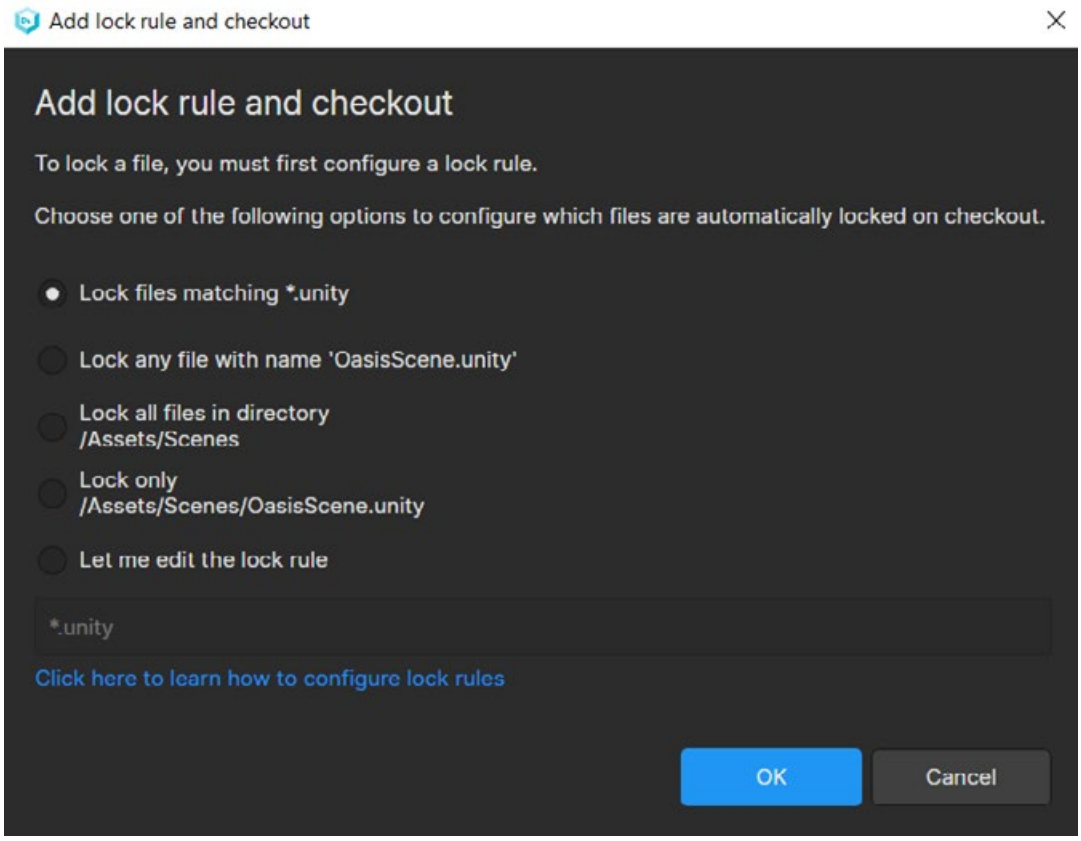
要锁定文件，请进入 Branch Explorer。

在 Workspace Explorer 视图中，找到要锁定的文件，右键单击它，然后选择 Lock and checkout。仅当没有其他团队成员签出文件时，此选项才有效。



在 Workspace Explorer 中从 UVCS 桌面应用程序锁定文件。

You can define locking rules for this type of file. To just lock this specific file, choose **Lock only** (The fourth option in the list below).



Define rules for locking this file or filetype.

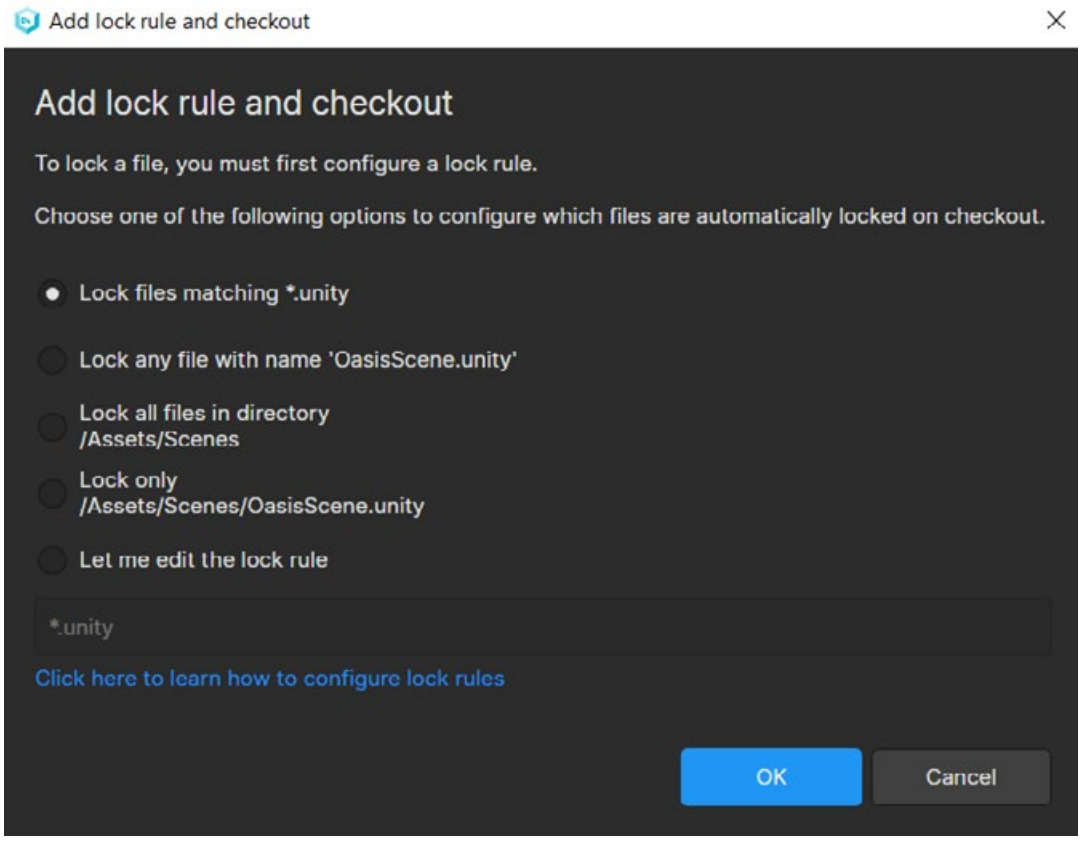
The file will now display a purple lock icon in each team member's version of Unity, letting others know that you have locked the file for editing.

The purple lock icon shows other team members that this file is locked.

Team members can still view the file as read-only, but cannot make any changes until it is checked back in by you.

After completing your changes, check in the file and it removes the lock.

您可以为此类文件定义锁定规则。要仅锁定此特定文件，请选择 Lock only （仅锁定）（下面列表中的第四个选项）。



定义用于锁定此文件或文件类型的规则。

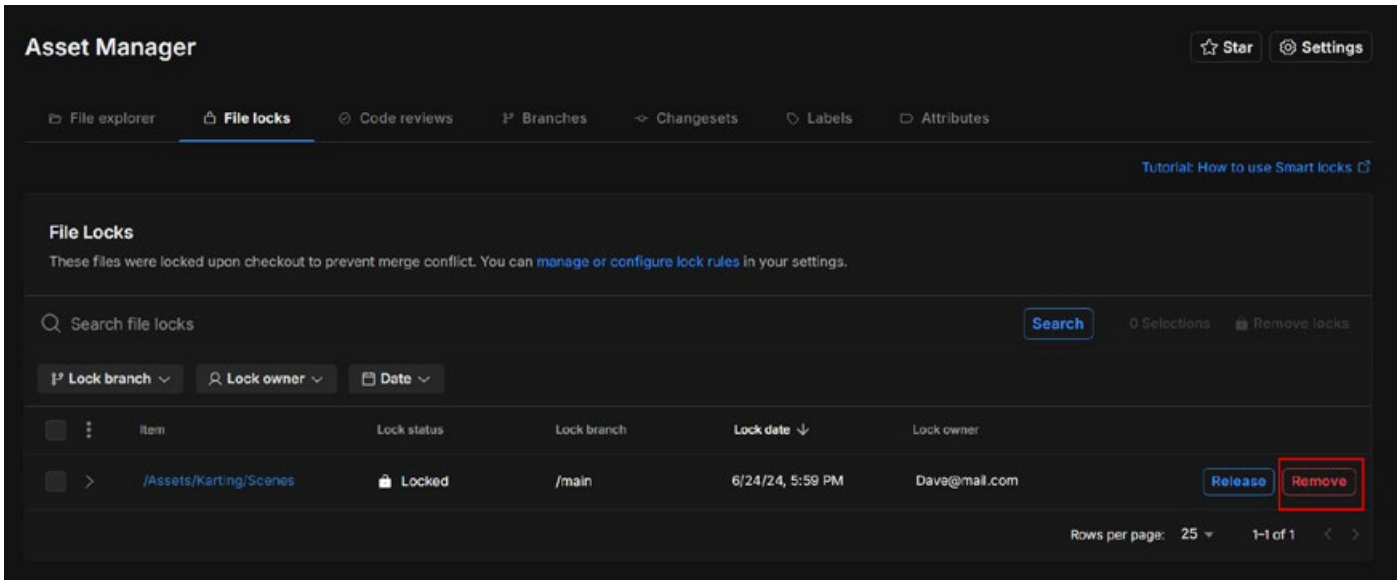
现在，该文件将在每个团队成员的 Unity 版本中显示一个紫色锁图标，让其他人知道您已锁定文件以进行编辑。

紫色锁图标向其他团队成员显示此文件已锁定。

团队成员仍可以只读方式查看文件，但在您重新签入文件之前无法进行任何更改.

完成更改后，签入文件，它会删除锁。

Team members with admin status can log into the cloud and go to **Repositories** and then **File locks** to manually remove a lock.

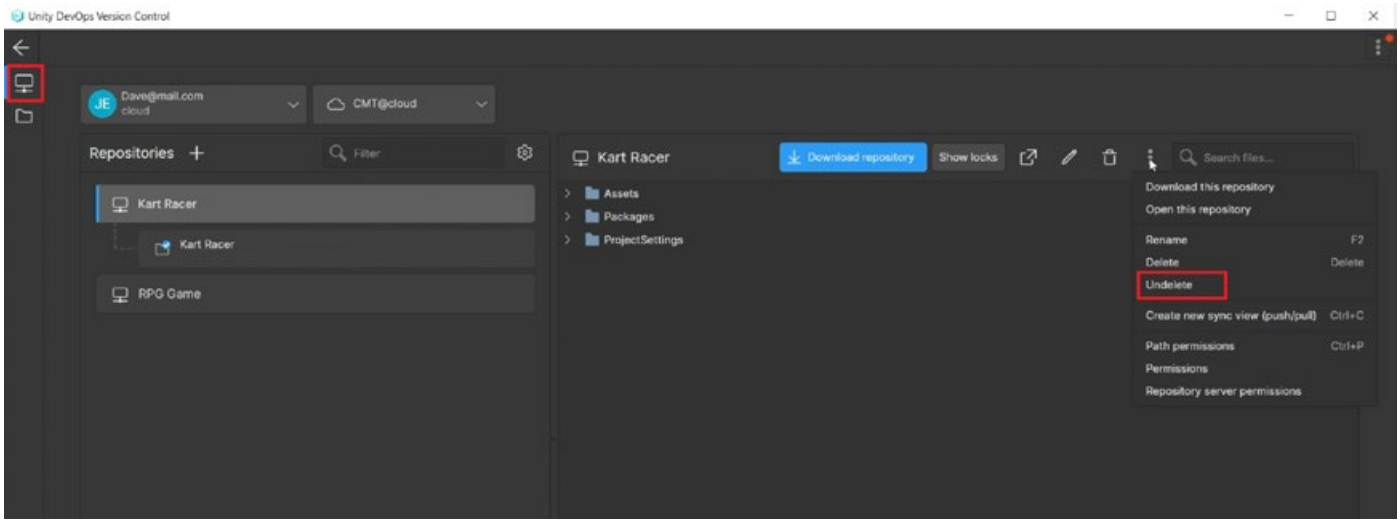


Removing file locks on the Cloud

Monitoring or removing a repository

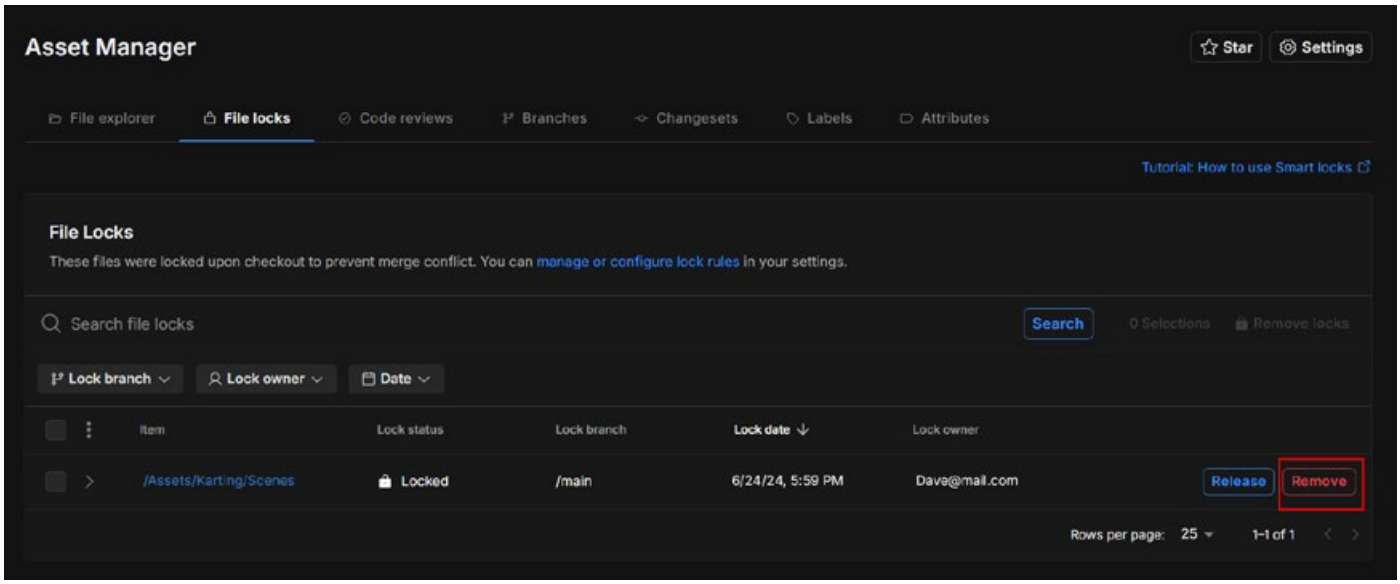
You can monitor the Repository in the desktop app. Click on the **Home** icon at the top left and select the top icon to view the current Repository.

From here you can download the entire repo to your computer, open it in the desktop app, rename it, or delete it. Repos stay on the cloud for a further 10 days before being deleted, in case you change your mind. You can retrieve a deleted repo by choosing **Undelete**.



Downloading, deleting, or undeleting a repo

具有管理员身份的团队成员可以登录云并转到 Repositories （存储库），然后转到 File locks （文件锁定） 以手动删除锁定。

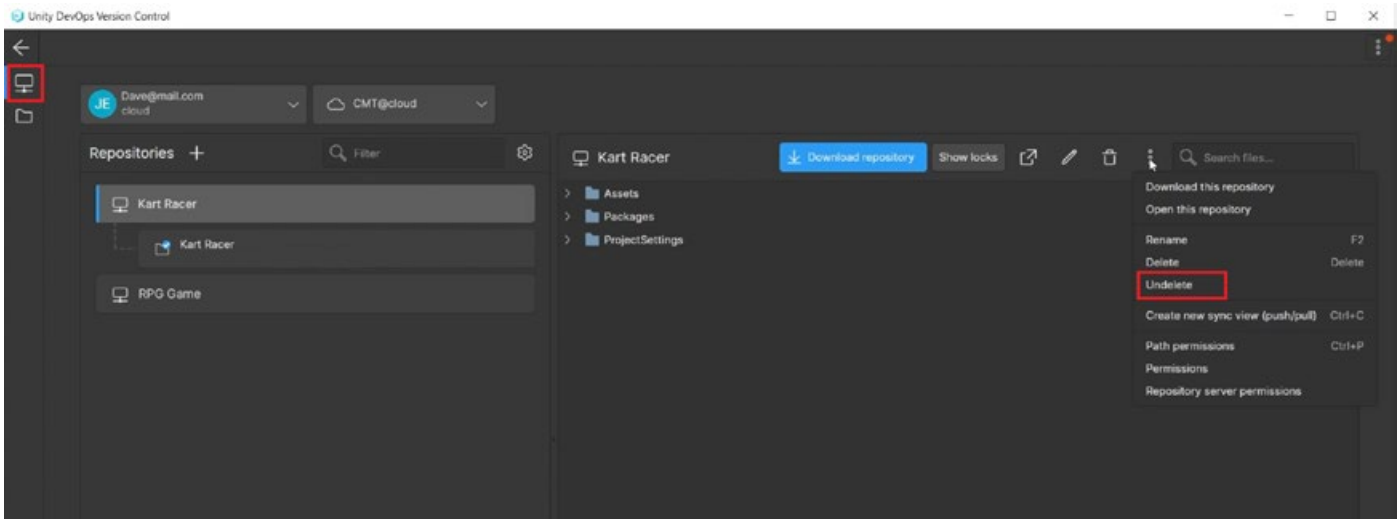


删除云上的文件锁定

监控或删除存储库

您可以在桌面应用程序中监控存储库。单击左上角的 Home 图标，然后选择顶部图标以查看当前存储库。

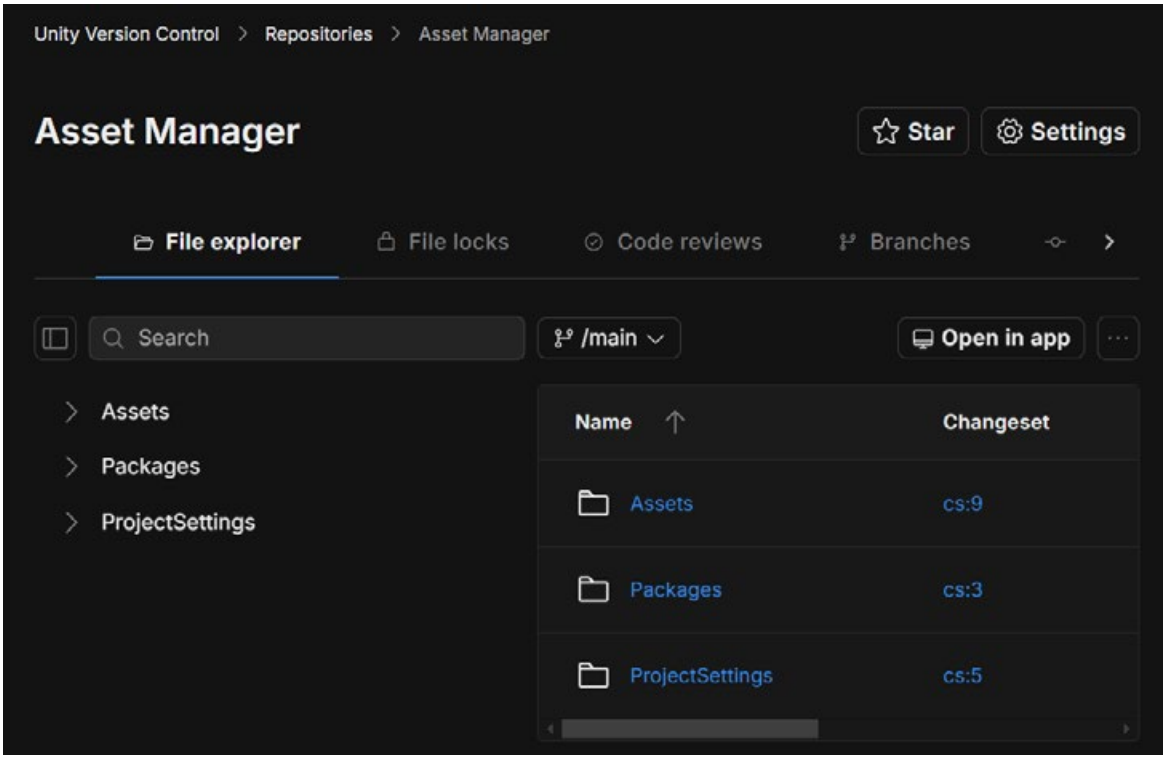
在这里，您可以将整个存储库下载到您的计算机上，在桌面应用程序中打开它，重命名或删除它。存储库在被删除之前会在云上再保留 10 天，以防你改变主意。您可以通过选择 Undelete. 来检索已删除的存储库



下载、删除或取消删除存储库

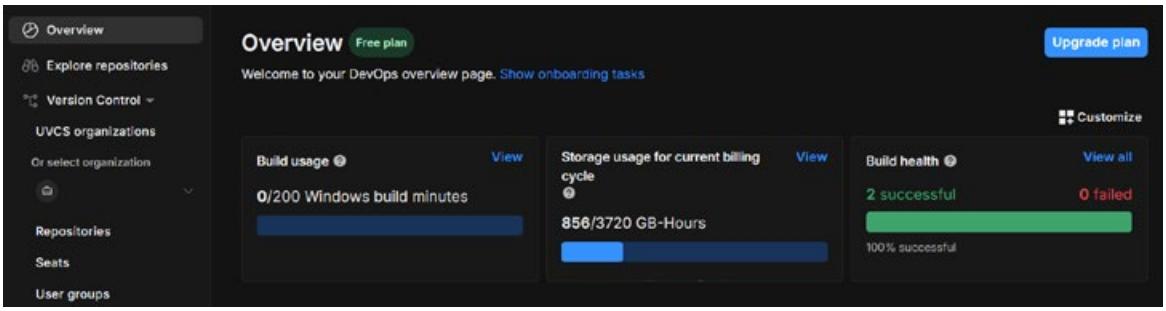
You can view any repositories on the Cloud in the DevOps section. This will show you the actual storage used, which should be much smaller than the project on your computer.

Here you can also add or remove repositories. The DevOps section will give you more information regarding your DevOps account including usage reports and settings.



View the storage of your repositories by logging into the Unity Cloud DevOps section.

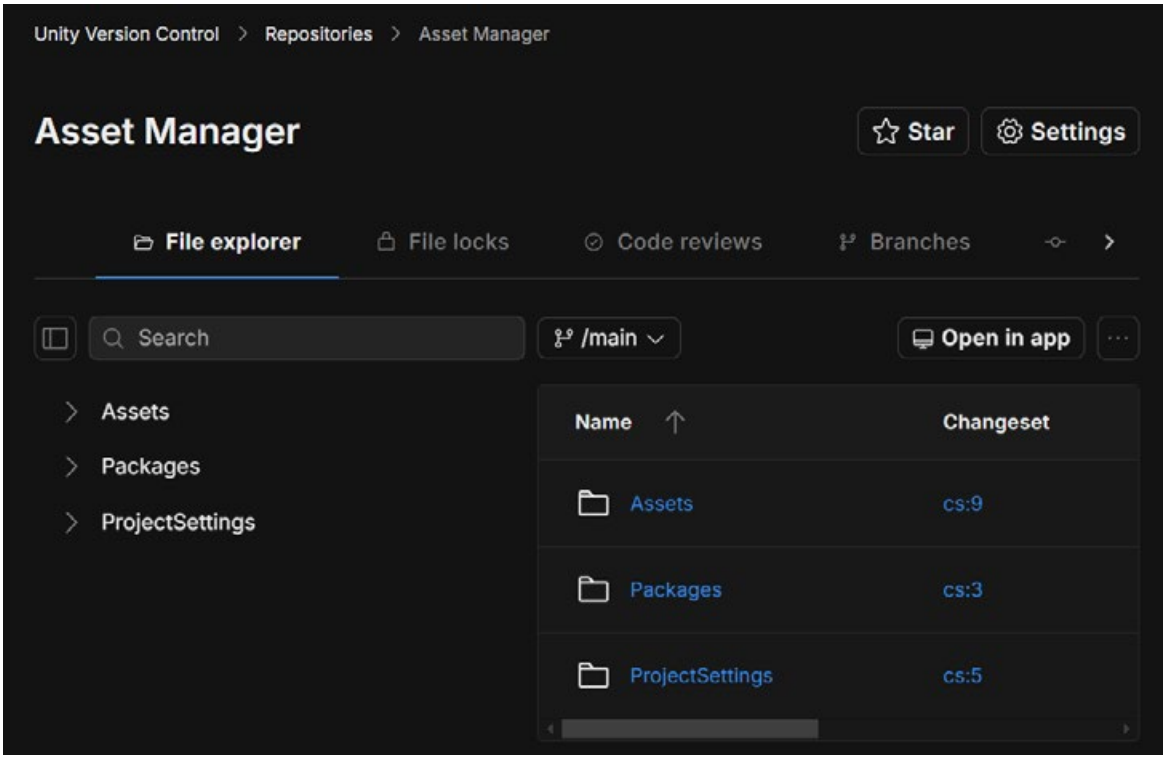
You can view your current data usage by checking the **Overview** section in the cloud. This will show your monthly GB hours. See the Unity Cloud pricing plans page for more information or sign into the Unity Cloud dashboard and go to the DevOps dashboard “About” page.



The Overview section on the Cloud shows your monthly storage.

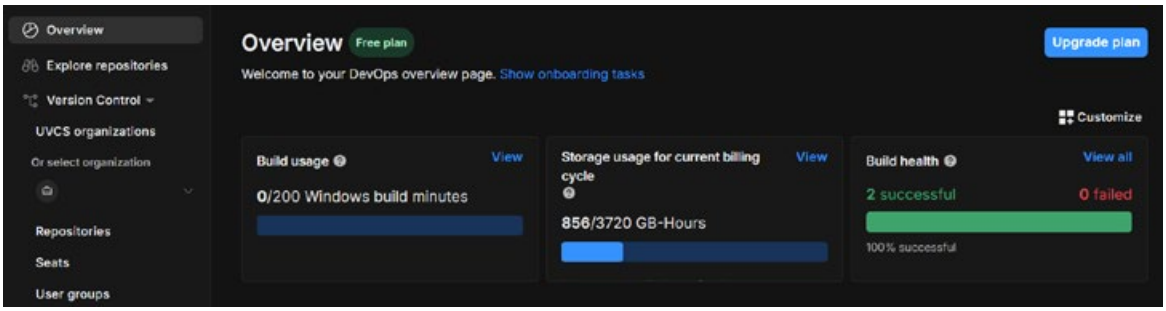
您可以在 DevOps 部分查看云上的任何存储库。这将显示实际使用的存储空间，该存储空间应比计算机上的项目小得多。

您还可以在此处添加或删除存储库。DevOps 部分将为您提供有关 DevOps 帐户的更多信息，包括使用情况报告和设置。



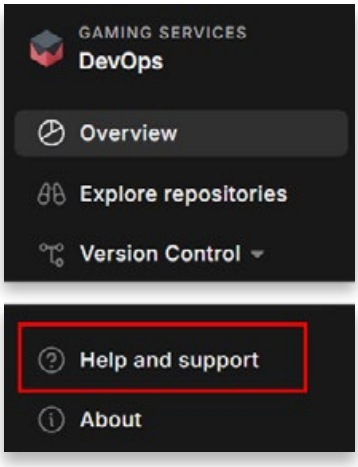
通过登录 Unity Cloud DevOps 部分查看存储库的存储。

您可以通过检查 cloud. 中的“概述”部分来查看当前数据使用情况，这将显示您每月的 GB 小时数。有关更多信息，请参阅 Unity Cloud 定价计划页面，或登录 Unity Cloud 控制面板并转到 DevOps 控制面板的“关于”页面.



Cloud 上的 Overview （概述） 部分显示您的每月存储量。

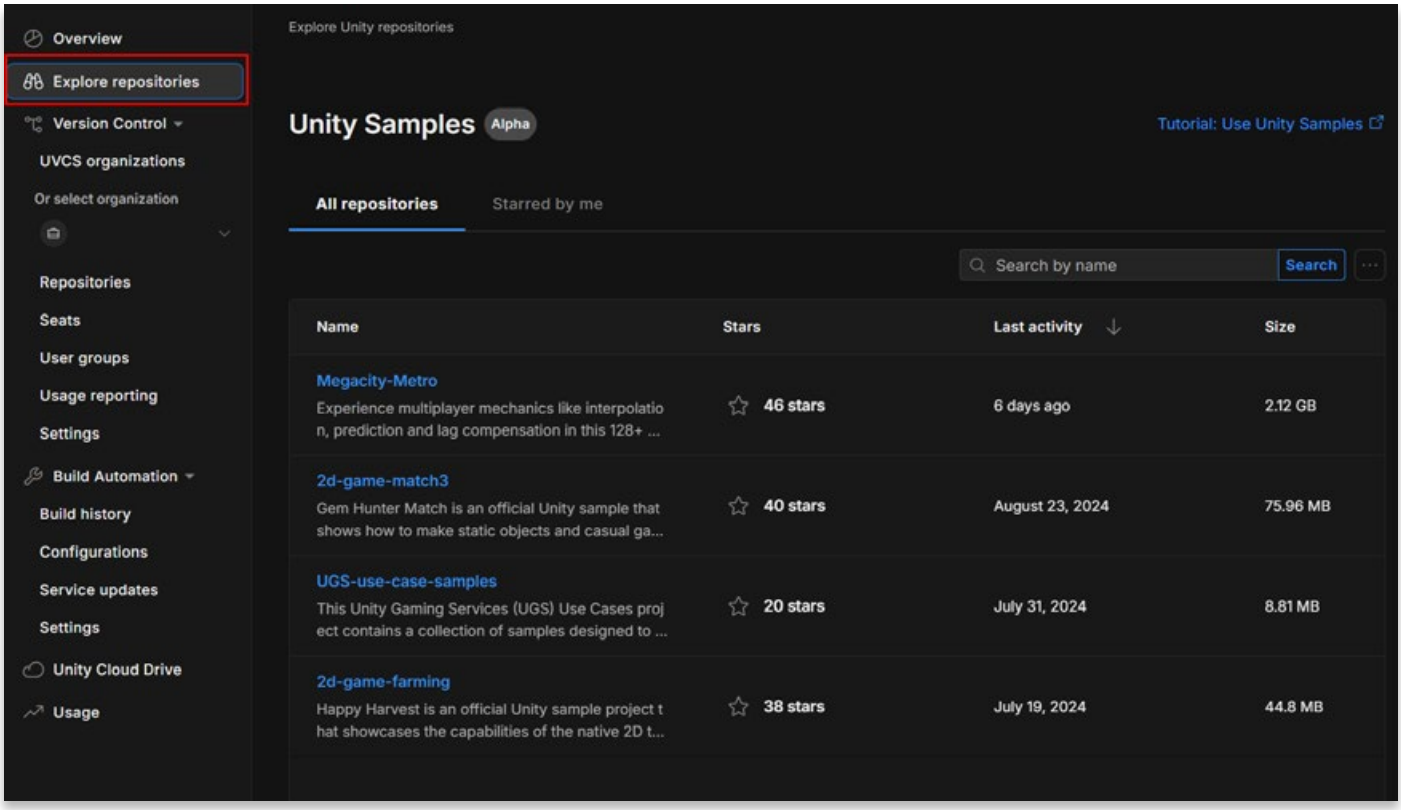
Unity support



In the DevOps section of the Unity Cloud dashboard, click on **Help and support**.

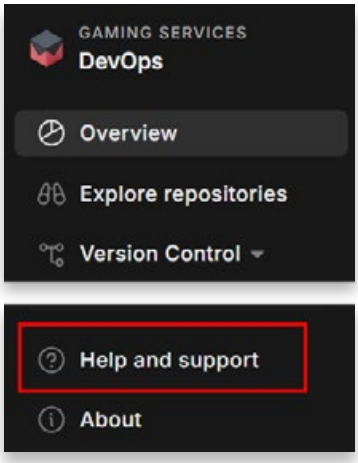
From the cloud dashboard, click on the **Help and support** button to view FAQs, forums and blog posts, or generate a support ticket for help with issues that you might encounter.

You will also find sample repo projects in the **DevOps** category, under **Explore repositories**. You can download these public repos to your computer as template projects to work on.



Some of the public sample repositories you can download

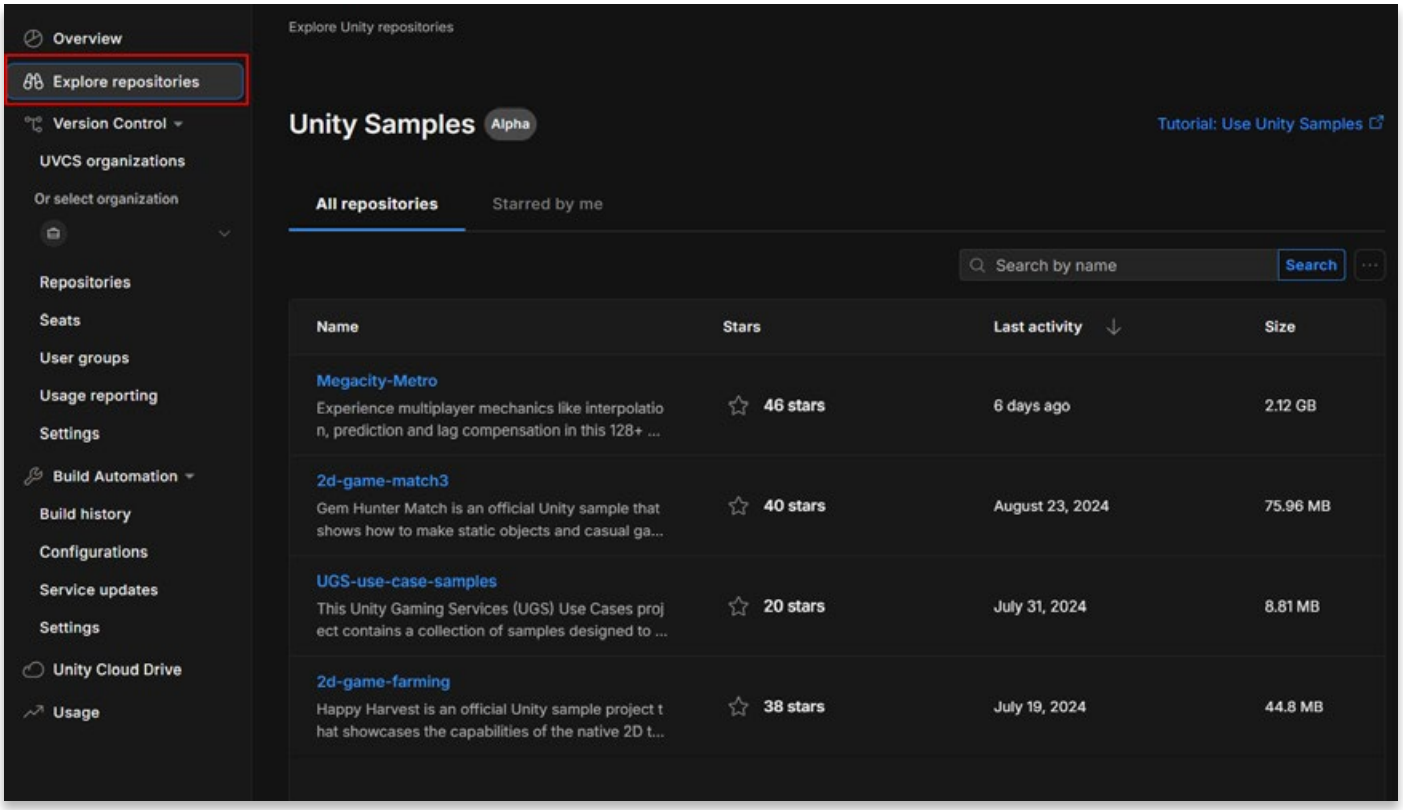
Unity 支持



在 Unity Cloud 控制面板的 DevOps 部分中，单击 Help and support.

在云控制面板中，单击 Help and support 按钮以查看常见问题解答、论坛和博客文章，或生成支持票证以帮助解决您可能遇到的问题。

您还可以在 DevOps 类别的 Explore repositories. 下找到示例存储库项目您可以将这些公共存储库作为模板项目下载到您的计算机上进行处理。



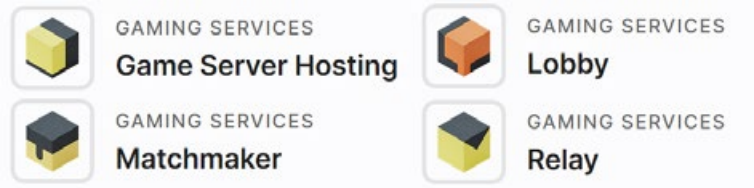
您可以下载的一些公共示例存储库

Build the foundation for your live game

Outside of devops Unity also provides a complete ecosystem for managing live games with Unity Gaming Services (UGS) that can help you acquire and connect your players at scale across multiple platforms, and manage your back end infrastructure. Here is a brief overview of some of the services.

Unity Gaming Services

Multiplayer



Unity offers two networking stacks: Netcode for GameObjects and Netcode for Entities, plus multiplayer tools from Unity Gaming Services like Game Server Hosting and Vivox Voice Chat. You can build within Unity's ecosystem, or mix and match your preferred tools and services depending on your needs.

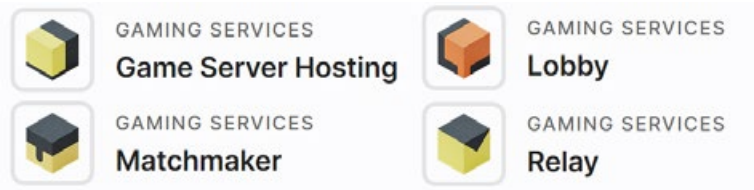
Learn more about Unity Multiplayer

为您的实时游戏奠定基础

在 DevOps 之外，Unity 还提供了一个完整的生态系统，用于通过 Unity Gaming Services （UGS） 管理实时游戏，这可以帮助您跨多个平台大规模获取和连接玩家，并管理您的后端基础设施。以下是一些服务的简要概述。

Unity 游戏服务

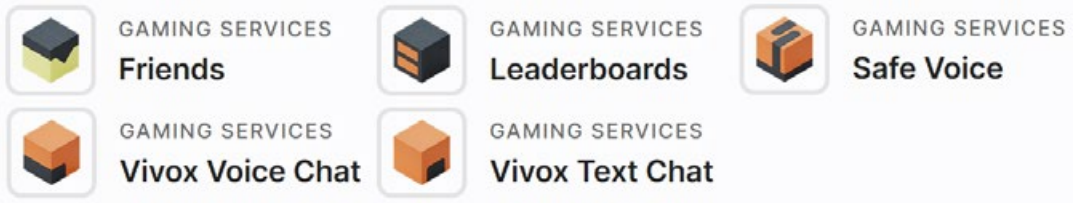
多人游戏



Unity 提供两个网络堆栈：用于游戏对象的 Netcode 和用于实体的 Netcode，以及来自 Unity Gaming Services 的多人游戏工具，例如 Game Server Hosting 和 Vivox Voice Chat。您可以在 Unity 的生态系统中构建，也可以根据需要混合和匹配您喜欢的工具和服务。

了解有关 Unity Multiplayer 的更多信息

Community



Access a suite of community gaming services for games of all sizes, from a two-person project to millions of simultaneous users, with engine-agnostic voice and text chat. Add social elements like friends and leaderboards to your game to increase player engagement and retention.

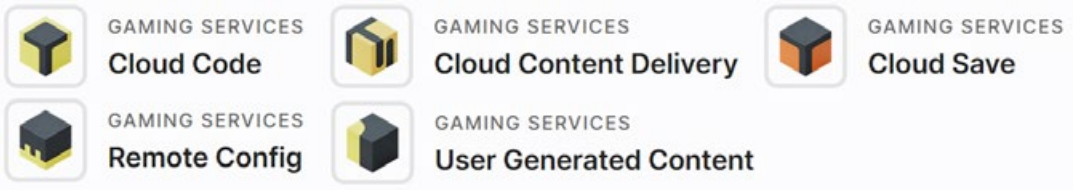
Learn more about Unity game Community solutions

Accounts

Unity Authentication and Unity Player Accounts offer you the flexibility to handle your players' identities by providing mix-and-match features suited to your game and target platforms.

Learn more about Unity Authentication and Player Accounts.

Content management

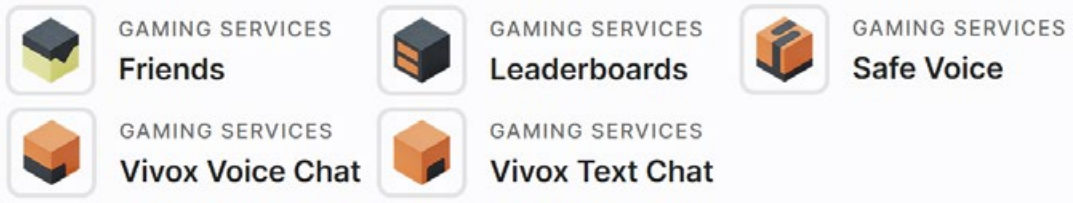


Cloud Save makes it efficient to create a better player experience where players can store game data to the cloud. With Cloud Content Delivery, you can build and release game updates with powerful asset management and content delivery via the cloud and the first 50 GB of bandwidth are free every month.

The Remote Config service makes it efficient to run A/B experiments that target specific player segments by using Game Overrides. You can change your game's difficulty, ad frequency, or general attributes for all players or a subset of players before making any code changes or app updates.

Learn more about Unity Content Management

社区



访问一套适用于各种规模游戏的社区游戏服务，从两人项目到数百万并发用户，以及与引擎无关的语音和文本聊天。在游戏中添加好友和排行榜等社交元素，以提高玩家的参与度和留存率.

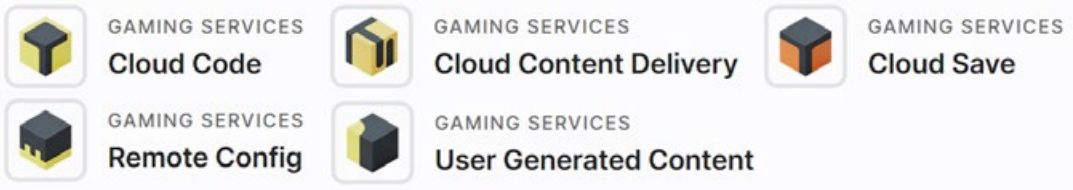
了解有关 Unity 游戏社区解决方案的更多信息

帐户

Unity Authentication 和 Unity Player Accounts 通过提供适合您的游戏和目标平台的混合搭配功能，让您可以灵活地处理玩家的身份。

了解有关 Unity 身份验证和玩家帐户的更多信息 s.

内容管理

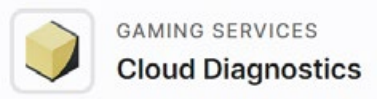


Cloud Save 可以高效地创造更好的玩家体验，玩家可以将游戏数据存储到云中。借助 Cloud Content Delivery，您可以通过云构建和发布具有强大资产管理和内容交付功能的游戏更新，并且每月前 50 GB 的带宽是免费的。

Remote Config 服务通过使用 Game Overrides（游戏覆盖）可以高效地运行针对特定玩家细分的 A/B 实验。在进行任何代码更改或应用更新之前，您可以更改所有玩家或部分玩家的游戏难度、广告频率或常规属性。

Learn more about Unity Content Management

Crash reporting



The Cloud Diagnostics service can help improve stability and reduce churn. It provides real-time crash data so your team can fix bugs faster and help prevent player churn. Unity offers crash and error reporting solutions that are free to start using and can be scaled up as your project grows.

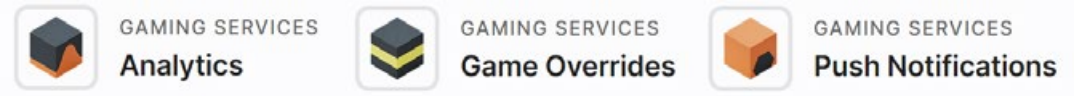
Learn more about Unity Cloud Diagnostics

Game economy

Game Economy services provide you with features to build a customized in-game economy and offer your players seamless purchases, currency conversion, inventory management, and more. Once you have integrated a fully-featured economy into your game you can manage it efficiently through a streamlined dashboard.

Learn more about Unity's Economy service

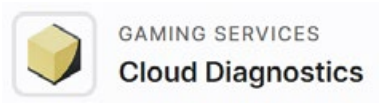
Engagement and analytics



Understand your game and engage your players. These services can help you efficiently refine your game experience, retain players, and grow your game.

Learn more about Unity Analytics

崩溃报告



Cloud Diagnostics 服务有助于提高稳定性并减少客户流失。它提供实时崩溃数据，以便您的团队可以更快地修复错误并帮助防止玩家流失。Unity 提供崩溃和错误报告解决方案，这些解决方案可以免费开始使用，并且可以随着项目的增长而扩展。

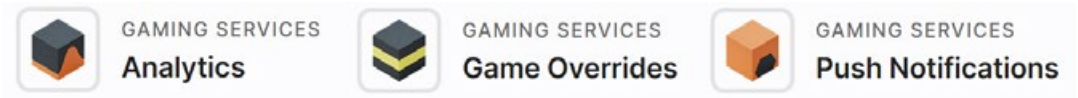
了解有关 Unity Cloud Diagnostics 的更多信息

游戏经济

Game Economy 服务为您提供构建自定义游戏内经济的功能，并为玩家提供无缝购买、货币转换、库存管理等功能。将功能齐全的经济系统集成到游戏中后，您可以通过简化的仪表板有效地管理它。

了解有关 Unity 的 Economy 服务的更多信息

参与度和分析



了解您的游戏并吸引您的玩家。这些服务可以帮助您有效地优化游戏体验、留住玩家并发展您的游戏。

Learn more about Unity Analytics

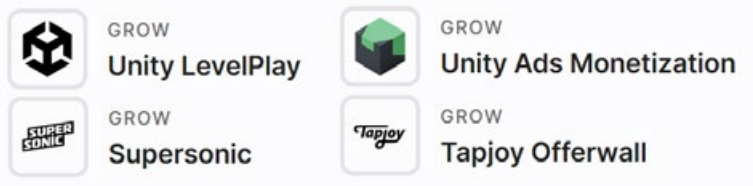
Unity Grow

User acquisition

Unity's suite of user acquisition services can help your game acquire the right mobile users. Tap into the extensive global supply of Unity Ads to optimize your campaign goals for return on ad spend (ROAS), retention, or scale.

Learn about Unity user acquisition

Monetization



Reach your app's full revenue potential with the most advanced technology and robust toolset in the market for monetizing your game effectively.

Learn about Unity monetization

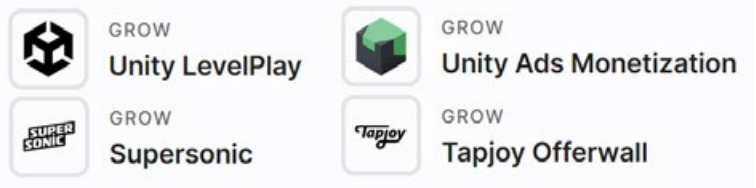
Unity 成长

用户获取

Unity 的用户获取服务套件可以帮助您的游戏获得合适的移动用户。利用 Unity Ads 的广泛全球供应来优化您的广告活动目标，以实现广告支出回报率 （ROAS）、留存率或规模。

了解 Unity 用户获取

货币



利用市场上最先进的技术和强大的工具集，有效地实现游戏变现，充分发挥应用的收入潜力。

了解 Unity 变现

Conclusion

Hopefully this guide will help you to feel more comfortable working with version control as part of a team in Unity. Even if you're working on a solo project, the principles of organizing your project and using version control can be really useful.

The biggest takeaway is the importance of clear team communication and agreement on tools, processes, structure and code style. As a team, you need to agree on your guidelines: how you should structure your project, which version control system to use, and how your workflow in that system looks. Then, when you start integrating other tools such as JIRA, GitLab, build tools, or automated testing, the work you've already done structuring your project and workflow will really come into its own.

Finally, check out the following resources to find a wealth of information on the various version control systems discussed in the book, plus more tips on setting up your Unity project for success.

结论

希望本指南能帮助您在 Unity 中作为团队的一员更自如地使用版本控制。即使您正在处理一个单独的项目，组织项目和使用版本控制的原则也非常有用。

最大的收获是清晰的团队沟通以及就工具、流程、结构和代码风格达成一致的重要性。作为一个团队，您需要就您的指导方针达成一致：您应该如何构建您的项目、使用哪个版本控制系统以及您在该系统中的工作流程是什么样子的。然后，当您开始集成其他工具（如 JIRA、GitLab、构建工具或自动化测试）时，您已经完成的构建项目和工作流程的工作将真正发挥作用。

最后，请查看以下资源，以查找有关书中讨论的各种版本控制系统的大量信息，以及有关设置 Unity 项目以取得成功的更多提示.

Additional resources

- Eight factors to consider when choosing a version control system
- Introduction to version control, Unite Now 2020
- Git Apprentice, by Chris Belanger and Bhagat Singh
- Version Control for Games with Unity's Plastic SCM
- Plastic SCM product documentation
- Mergebot in Plastic SCM
- Unity open project with version control
- How KO_OP uses Plastic SCM to accelerate production
- The hidden productivity costs disrupting your release timelines

Perforce setup

- How to Configure Helix Core and Game Engine
- Helix Core documentation

其他资源

— 选择版本控制系统时要考虑的八个因素 — 版本控制简介，Unite Now 2020 — Git Apprentice，作者：Chris Belanger 和 Bhagat Singh — 使用 Unity 的 Plastic SCM 对游戏进行版本控制 — Plastic SCM 产品文档 — Plastic SCM 中的 Mergebot — 具有版本控制的 Unity 开放项目 — KO_OP 如何使用 Plastic SCM 加速生产 — 扰乱发布时间表的隐藏生产力成本

Perforce 设置

— 如何配置 Helix Core 和游戏引擎— Helix Core 文档


Unity®


Unity®