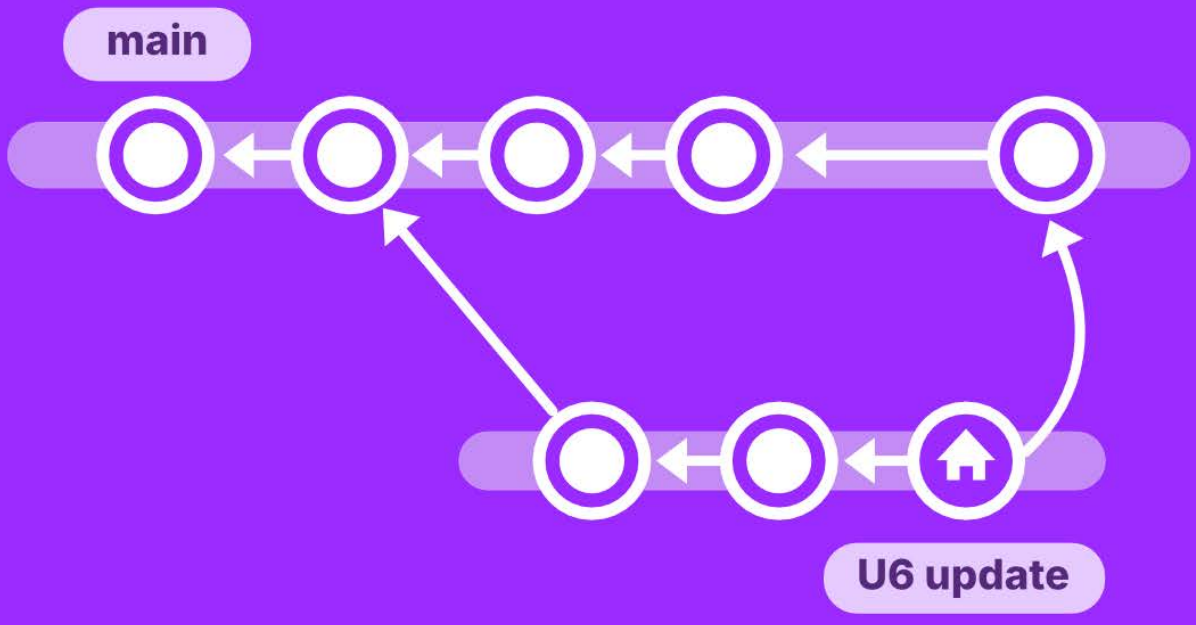
→ E ブック

プロジェクト整理とバージョン管理のベストプラクティス



Unity®

Contents

はじめに **5**

ソース管理とバージョン管理の比較 **6**

基本的な概念 **7**

バージョン管理の仕組み 7

バージョン管理を使うべき理由 8

集中型バージョン管理と分散型バージョン管理 8

集中型 8

分散型 9

集中型 10

分散型 10

主要な用語 11

Unity プロジェクト管理のベストプラクティス **13**

プロジェクト整理 13

フォルダー構造 13

空のフォルダー 19

.meta ファイル 20

命名基準 21

ワークフローの最適化 22

アセットの分割 22

プリセット 23

コーディング基準 24
UI Toolkit の形式規則 27
プロジェクト整理用のサービス 27
Asset Manager 27
Build Automation 31
バージョン管理システム 36
Git 36
Perforce（Helix Core） 41
Apache Subversion 42
Unity Version Control 42
VCS の比較 45
Unity をバージョン管理システムに連携する 46
エディターのプロジェクト設定 46
Perforce Helix Core 46
UVCS 48
Git やその他のソリューション 50
無視するファイル 50
大きなファイルの管理 51
バージョン管理のベストプラクティス 53
コミットは小さく頻繁に 53
コミットメッセージは簡潔に 54
無計画なコミットを避ける 54
最新の変更を取得 55
ツールセットを理解する 57
機能ブランチと Git Flow 58
プルリクエスト 60
Unity 6 で UVCS を始める 62

Unity プロジェクトで UVCS を使用する 63
他のチームメンバーを招待する 65
変更をチェックイン. 67
UVCS デスクトップクライアント 69
Gluon を使用する 71
UVCS デスクトップアプリ. 74
ブランチ. 74
競合の処理 77
ルールをマージする 79
ファイルのロック 84
リポジトリを監視または削除する 87
Unity サポート. 89
ライブゲームの基盤を構築する 90
Unity Gaming Services 90
マルチプレイヤー 90
コミュニティ 91
アカウント. 91
コンテンツ管理. 91
クラッシュレポート 92
Game Economy 92
エンゲージメントと分析 92
Unity Grow. 93
ユーザー獲得 93
Monetization. 93
まとめ 94
追加リソース 95

はじめに

ソフトウェア開発は、ソロプロジェクトとチームプロジェクトでは、また違ったものになります。チームメンバー全員がアクセスできるようにするには、どこにプロジェクトを保管すべきなのでしょうか？同じファイルで複数人が同時に作業を行った場合はどうなるのでしょうか？プログラマーはソース管理の背後にある概念を理解していることが多いですが、アーティストや他の非技術的なチームメンバーはどうでしょう？プログラマーからのサポートを最小限に抑えつつ、彼らが間違った操作をする心配をなくすにはどうすれば良いのでしょうか？

ソース管理、またはバージョン管理は、特に技術的なバックグラウンドがない場合、ゲーム開発者の頭を悩ませるトピックになり得ます。しかし、そんなに難しく考える必要はありません。チーム開発での効率的なバージョン管理を支援する、Unity との統合が可能なツールは数多く存在します。

このガイドでは、バージョン管理の主な概念を説明し、利用可能なさまざまなバージョン管理システム (VCS) をいくつか比較し、Unity Asset Manager、エンゲージメントと分析、ゲームエコノミー、マルチプレイヤーサービスなどの Unity DevOps 追加ツールを紹介します。チームコラボレーションをスムーズかつ効率的に行えるようにするために、Unity プロジェクトをセットアップする際に使用できるヒントやテクニックを提供します。最後に、チームでの作業を成功させるためのバージョン管理のベストプラクティスも学ぶことができます。

ソース管理とバージョン管理の比較

コンピューティングの黎明期では、ソフトウェア開発はすべて純粋なコードで行われていました。3D グラフィックスが進化し始めてからも、すべてがコードで記述されていたのです。そのため、プロジェクトの内容を管理するシステムは「ソース管理」と呼ばれ、それらのツールには「ソースコード管理」（SCM）という名称が付けられていました。

しかし、現代のソフトウェアやゲーム開発では、ソースコード以外にも多くのものを扱います。FBX などの 3D モデル形式、テクスチャ、マテリアル、オーディオファイルなど、SCM はテキストファイルの変更以上のものを意味するようになりました。いまや「ソース管理」という単語では必要なものを網羅しきれないため、より適切な説明として、「バージョン管理システム（VCS）」という名前が使用されるようになりました。

これらの単語は、今でも同じ意味で使われることがあります。しかし、大規模なバイナリアセットを扱うことが多い Unity プロジェクトについて言及する場合は「バージョン管理（VCS）」という表現がより正確なので、当ガイドではそのような表現を使用します。

Unity にもっとも適した主なバージョン管理システムは、Unity Version Control（UVCS）（旧 Plastic SCM）、[1]Git、Perforce Helix Core の 3 つです。このガイドでは、複数人で Unity プロジェクトでの作業を行う際の、これらのシステムの利点と欠点を紹介します。

[1] Plastic SCM は 2020 年に Unity ファミリーに追加されたため、これらのツールは Unity エディターと密接に統合されています。

基本的な概念

このセクションでは、バージョン管理の核となる概念のいくつかを取り上げます。「コミット」と「プッシュ」の違いがわからない場合、このセクションを読むことでバージョン管理用語を理解できるようになるでしょう。

バージョン管理の仕組み

バージョン管理を行うことによって、プロジェクト全体の履歴を残すことができます。作業内容を整理し、チームが効率的にイテレーションを行うことも可能にします。

プロジェクトファイルは、リポジトリと呼ばれる共有データベースに保存されます。定期的にプロジェクトをリポジトリにバックアップすることで、何か問題が発生した際、プロジェクトを以前のバージョンに戻すことが可能になります。

VCS を用いると、個別の変更を複数行い、バージョニング用にそれらを 1 つのグループとしてコミットできます。このコミットは、プロジェクトのタイムライン上の一点として位置づけられます。以前のバージョンに戻す必要がある場合、そのコミット以降の変更はすべて元に戻され、プロジェクトがその時点の状態に復元されます。コミット内でグループ化された各変更をレビューして修正することも、コミットを完全に取り消すことも可能です。

プロジェクト全体の履歴にアクセスできるため、どの変更によってバグが発生したかを特定したり、過去に削除された機能を復元したり、ゲームや製品のリリース間の変更を簡単にドキュメントにまとめたりすることができるようになります。

さらに、バージョン管理はクラウドまたは分散サーバーに保存されることが多いため、開発チームがどこで作業していてもコラボレーションをサポートすることができます。リモートワークが一般化している今、このメリットはより重要なものになっています。

バージョン管理を使うべき理由

上記の理由に加えて、バージョン管理は実験的な変更を行うのにも便利です。ローカルバージョンのプロジェクトに新機能を追加し、うまくいかなかった場合は変更を元に戻すだけで、プロジェクトをクリーンで機能的なバージョンに戻すことができます。

これによって実験的なアイデアのイテレーションができるようになりますし、メインのプロジェクトで発生した大きな問題の解決を手伝わなくてはならなくなった場合でも、バージョン管理を活用していれば後日に向けて変更を保存することが可能です。その後、手元のローカルバージョンをメインのブランチに戻し、必要な作業を手伝うことができます。完了したら復元を行い、実験的な作業を続けられます。

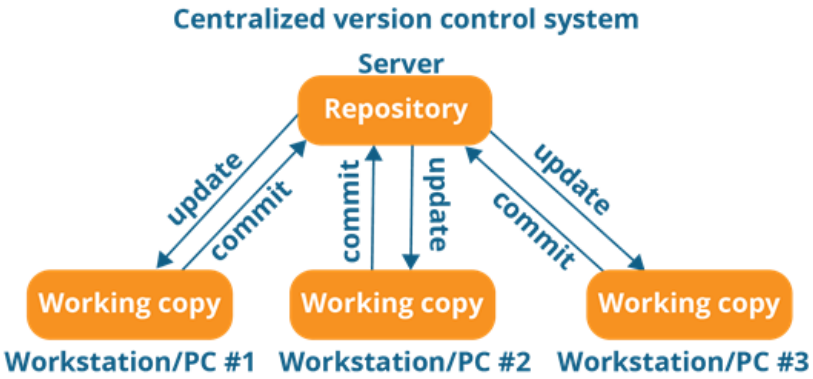
多くのバージョン管理システムは、チーム内の誰かが行った作業を誤って上書きしてしまうことを防いでくれます。リポジトリに作業をコミットする際、リポジトリから最新の更新情報を「引っ張ってくる」必要があります。これにより、他の誰かが同じファイルで作業していないかどうかを確認することが可能です。このような状況は「マージ競合」としても知られており、バージョン管理に慣れていない人にとっては怖いもののように思えるかもしれません。しかし、マージ競合は、ツールを理解すれば簡単に解決できます。Unity Version Control（UVCS）はスマートロックを採用しており、あらゆるブランチのロックされたファイルが最新のリビジョンかどうか確認することで、マージ競合のリスクを減らす手助けをしています。

集中型バージョン管理と分散型バージョン管理

バージョン管理システムの大半は、集中型または分散型のいずれかのカテゴリーに分類されます。取り扱うバージョン管理システムの種類によっては、以下に概説する用語が適用されたり、適用されなかったりします。用語によっては意味がまったく異なるものになるかもしれません。これらの 2 つのカテゴリーの違いを見てみましょう。

集中型

集中型と分散型の最初の重要な違いは、リポジトリがどこに存在するかということです。多くの企業は、プロプライエタリソフトウェアをホストしているサーバーをオンサイトで維持するために、集中型のバージョン管理システムを選択します。ソース管理のセキュリティは、しばしば集中型システムの採用につながる重要な要素となります。集中型システムにおいて、リポジトリをオンサイトのサーバーではなくクラウド上でホスティングすることも可能ですが、このような設定は分散型システムほど一般的ではありません。



2 つのアプローチのもう 1 つの重要な違いは、ユーザーがどのように変更をリポジトリにデプロイするかです。集中型バージョン管理は、多くの場合、よりシンプルなオプションとみなされます。集中型のリポジトリでは、変更のフェッチおよびプッシュは、リポジトリに対して直接行われます。このプロセスは「リポジトリからの更新」と「リポジトリへのコミット」と呼ばれます。

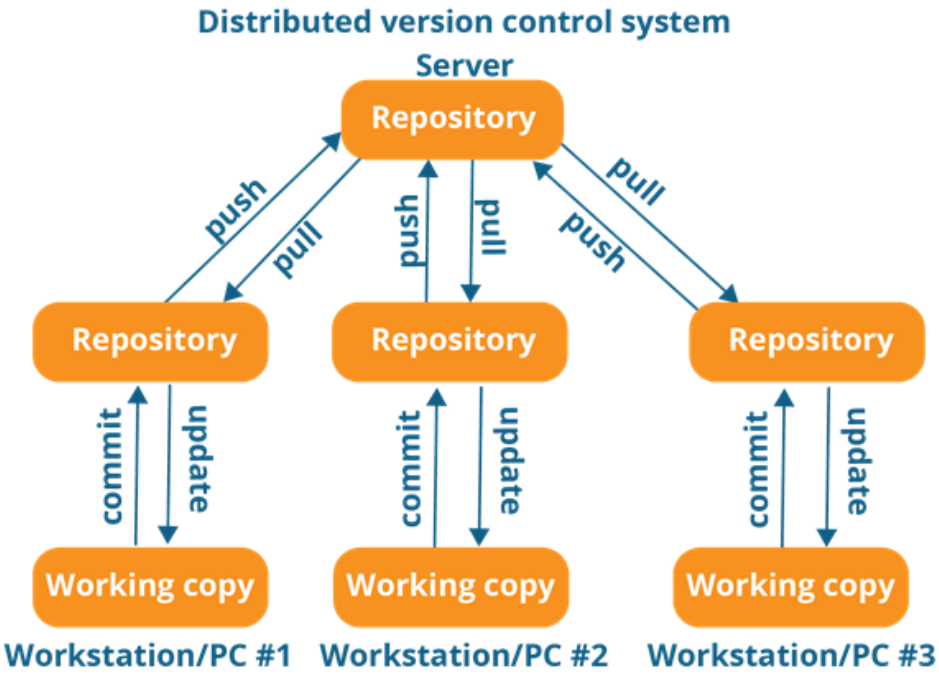
この設定の欠点は、ユーザーがサーバーに接続していないと作業内容を送信できないことです。競合を避けるため、ユーザーはファイルをロックして変更を行います。これはファイルのチェックアウトと呼ばれ、ファイルがチェックインされるまで、他の誰も変更をコミットできなくなります。

集中型のワークフローにおいて、ユーザー側は自分のワークステーションに最新バージョンのプロジェクトファイルしか持たず、サーバー側にはプロジェクトの全履歴が保存されます。

分散型

分散型のワークフローでは、通常 GitHub などのクラウドサービス上に 1 箇所のリポジトリが存在しますが、ユーザーはプロジェクト履歴全体を自分のワークステーションにクローンします。これにより、ユーザーは自分のローカルコピーで作業でき、中央サーバーに接続する必要がないため、変更を素早くコミットできます。これらの変更を送信し、後で他の人がアクセスできるようにするには、ユーザーがサーバーにプッシュし、他の変更をプルする必要があります。しかし、集中型のシステムとは異なり、常に最新版のファイルで作業する必要はありません。

この方法で作業すれば、より大きな機能に相当するチェンジセットのグループを作成してから、チーム全体に共有することができます。実際、小さな変更を頻繁にコミットすることが推奨されていますが、そのようなベストプラクティスについては後ほどご紹介しましょう。



ファイルロックは一部の分散型ワークフローでも利用可能ですが、マージを簡単に行えるため、使用頻度はそれほど高くありません。最新の変更をサーバーからローカルプロジェクトにプルすることで、変更をリポジトリにプッシュする前に他の人の変更と自分の変更を比較し、競合がないかを確認することができます。

分散型のアプローチのほうが好まれやすい一方、欠点もいくつかあります。まず、特にチームでバイナリファイルを扱う場合、プロジェクト履歴全体をローカルマシンに置くために多くのスペースが必要になります。Git には Large File Storage（LFS）というオプションがあり、特定のファイルの履歴をテキストポインタに変換して、容量を削減することができます。しかし、他のファイルには履歴全体が含まれている場合もあり、リポジトリに古いテストデータが蓄積されてしまう場合もあります。小さな M2 ドライブを使用しているスタジオの場合、リポジトリが古いバージョンによって肥大化し、ドライブに負荷がかかることがあるかもしれません。

次に、開発者は中央サーバーと常に接続する必要がないため、長期間にわたって孤立して作業することになりかねません。ローカルバージョンがメインリポジトリから大きく乖離してしまい、変更内容をマージする際に想像以上の手間がかかる可能性があります。

一般的なワークフロー

簡単にまとめると、集中型システムと分散型システムのワークフローの手順は以下の通りです。

集中型

1. サーバーからの変更で作業用コピーを更新する。
2. 自分の変更を行う。
3. 変更を中央サーバーにコミットする。

分散型

1. リモートの変更をローカルリポジトリにプルする。
2. 自分の変更を行う。
3. 変更をコミットする。
4. ステップ 2 と ステップ 3 を好きなだけ繰り返す。
5. すべてのコミットをリモートリポジトリにプッシュする。

このガイドでは、UVCS、Git、Perforce Helix Core の 3 つの主要なバージョン管理システムに焦点を当てています。それぞれの VCS の詳細な説明は「バージョン管理システム」のセクションで行いますが、ここではそれぞれの簡単な概要とサポートされているワークフローをご紹介します。

- Unity Version Control または UVCS（両方）：UVCS は、プログラマーやアーティストをサポートする独自のインターフェースを備えた柔軟なバージョン管理システムです。大規模なリポジトリやバイナリファイルの扱いに優れており、ファイルベースでも変更セットベースでもあるソリューションとして、プロジェクト全体ではなく、作業中の特定のファイルのみをダウンロードする機能を提供します。
- Git（分散型）：もっとも人気のあるバージョン管理システムのひとつです。Git はオープンソースで、無料で使用でき、高い柔軟性を備えています。プラットフォームとしての Git は、コマンドライン専用のツールです。しかし、Git 向けに様々な GUI が開発されており、ユーザーにとってより使いやすいシステムになっています。

Git と GitHub は、混同されることがよくあります。GitHub は Git リポジトリのホスティングサービスですが、GitHub を使わずに Git を利用することも可能です。とはいえ、本書を読んでいる経験豊富な開発者ならご存知のように、GitHub は（いくつかの制限はありますが）無料版があり、カスタムサーバーのセットアップも不要なため、非常に人気のあるサービスです。

— Perforce Helix Core（集中型）：自社サーバーでホストされることが多い集中型のリポジトリを特徴とするため、大規模なゲームスタジオで使用されることが多いエンタープライズ向けのバージョン管理システムです。

主要な用語

以下は、VCS における主な機能やプロセスの用語です。

用語	説明
リポジトリ	プロジェクトのすべての変更および編集を記録するデータベースです。オンサイトまたはクラウド上のサーバーに保存され、プロジェクトの全履歴を含みます。
作業用コピー	プロジェクトのローカルバージョンです。「チェックアウト」または「ワークスペース」とも呼ばれます。作業用コピーに変更を加え、問題がなければリポジトリにコミットします。
コミット / チェックイン	コミットを行うと、ファイルへの変更がエンコードされます。集中型ワークフローではこれらの変更がサーバーに送信され、より一般的にはチェックインと呼ばれます。分散型ワークフローでは、変更セットに追加され、これを後でサーバーにプッシュすることになります。
プル / 更新 / チェックアウト	プルまたは更新を行うと、利用可能な最新の変更がサーバーから取得されます。集中型ワークフローでは、チェックアウトと呼ばれることが多いです。
ロック	ファイルをロックすると、他のユーザーによる編集が阻止されます。「今このファイルで作業をしているので、他の変更を加えないでください」とサーバーに伝えているのと同義です。一般的に、分散型ワークフローでロックはサポートされていません。
クローン	分散型ワークフローでは、リポジトリをクローンすることで、プロジェクトとその全履歴をローカルマシンにコピーできます。
タグ	タグとは、コミットに追加できる特別なメモです。ビルドが作成された時点をマークするのによく使われます。
ブランチ	ブランチはコードラインの新しいコピーを作成し、並行して作業することを可能にします。これにより、例えば新機能の開発時など、プロジェクトの一部を切り離して作業しても、メインの開発ラインに影響を与えません。

マージ	マージは、ブランチが完了してメインラインに統合する必要があるとき、または 2 人が同時に変更を加えたときに発生します。2 つの変更セットを比較してマージし、新たな作業用コピーを作成する必要があります。マージの大半は自動的に処理できます。
競合	競合とは、マージを自動的に行えない場合に発生します。通常、2 人が同じコード行や同じバイナリファイルに変更を加えたときに起こります。 コードの競合は通常、テキストを比較して、どの変更を採用するか、または両方を統合できるかを判断することで解決できます。 Unity シーンやプレハブなどのバイナリファイルでは、競合マージがかなり難しくなります。ただし、競合箇所の作業者と軽く話し合うだけで、どちらの変更を採用するのが最適か簡単に決められることもあります。
プルリクエスト	ブランチでの作業が完了した際は、プルリクエストをオープンするのが良い習慣です。これにより、ブランチでの作業が完了し、メインラインにマージする準備が整ったことをチームの他のメンバーに通知できます。このシステムを使うと、チームリーダーや上級開発者が、メインブランチに反映する前に変更内容をレビューする機会を持てます。
HEAD	HEAD とは、作業用コピーの最新のコミットを指します。
リセット / リバート	VCS によりますが、リセットまたはリバートは、ローカルでの変更をすべて破棄し、HEAD の状態に戻すのに使われます。
インデックス	Git のインデックスとは、作業用コピーの現在の変更状況をすべて記述したファイルです。これらの変更はステージングエリアと呼ばれる場所に置かれ、次のコミットに追加したい変更を選択することができます。
Git スタッシュ	まだ変更をコミットする準備ができていないが、別の作業に移らなくてはならない場合、スタッシュを使用してそれらの変更を一時ファイルに保存し、作業用コピーをリセットして HEAD に戻すことができます。

Unityプロジェクト管理のベストプラクティス

どの VCS を選択するにしても、Unity で作業する際にバージョン管理のワークフローを合理化するのに役立つ、一般的な推奨事項がいくつかあります。まず、チームが効果的に共同作業を行うためのいくつかの方法を見てみましょう。

プロジェクト整理

フォルダー構造

プロジェクトを整理する方法はひとつとは限りませんが、以下の一般的な推奨事項に従うと良いでしょう。

- チーム全体で命名規則およびフォルダー構造を定義し、文書化する。スタイルガイドやプロジェクトテンプレートを使用して、ファイルの検索や整理を効率化。チームに合った方法を選び、全員がそれに賛同していることを確認する。
- 命名規則に一貫性をもたせる。選択したスタイルガイドまたはテンプレートから逸脱しない。命名規則を修正しなくてはならない場合、影響を受けるすべてのアセットを一度に解析し、名前を変更する。変更が多数のファイルに影響する場合、スクリプトを使って更新を自動化することを検討する。
- ファイル名およびフォルダー名にスペースを要しない。スペースの代わりにキャメルケースを使用する。
- テスト用エリアやサンドボックスエリアを分ける。本番以外のシーンや実験用に別のフォルダーを作成する。ユーザー名を含むサブフォルダーで、チームメンバー別に作業領域を分ける。
- ルート階層に余分なフォルダーを作成しないようにする。一般的に、コンテンツファイルは Assets フォルダーに保存する。どうしても必要な場合を除き、プロジェクトのルート階層に追加フォルダーを作成しないようにする。

— 社内で制作したアセットは、サードパーティ製のものとは分けておく。Asset Store のアセットや他のプラグインを使用する場合、それぞれに個別のプロジェクト構造がある可能性が高い。アセットを分けておく。

プロジェクトでサードパーティのアセットやプラグインをカスタマイズする場合、最新のプラグインアップデートが必要になった際に、バージョン管理が非常に役立ちます。アップデートをインポートした後、差分を確認することで、自分のカスタム変更がどこで上書きされたのか特定し、それを再実装できます。

Unity プロジェクトに決まったフォルダー構造はありませんが、例としていくつかのセットアップを紹介します。これらの構造は、アセットの種類ごとにプロジェクトを分割する原則に基づいています。アセットの種類のマニュアルページでは、もっとも一般的なアセットについて詳しく説明しています。フォルダー構造の管理方法の例として、テンプレートまたは Learn プロジェクトを参考にできます。フォルダー名は必ずしもこれらに限定されるわけではありませんが、良い出発点になるはずです。

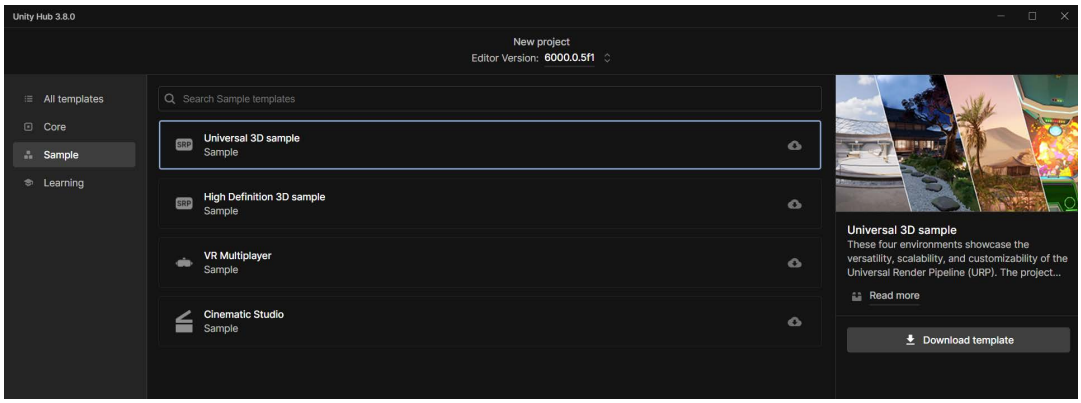
例 1

```
Assets
+---Art
| +---Materials
| +---Models
| +---Textures
+---Audio
| +---Music
| \---Sound
+---Code
| +---Scripts  # C# スクリプト
| \---Shaders  # シェーダーファイルとシェーダーグラフ
+---Docs   # Wiki、コンセプトアート、マーケティングマテリアル
+---Level   # Unity でのゲームデザインに関係するものすべて
| +---Prefabs
| +---Scenes
| \---UI
```

例 2

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

Unity Hub からテンプレートやスタータープロジェクトをダウンロードすると、下の画像のようにアセットの種類によってサブフォルダーが分けられていることがわかります。

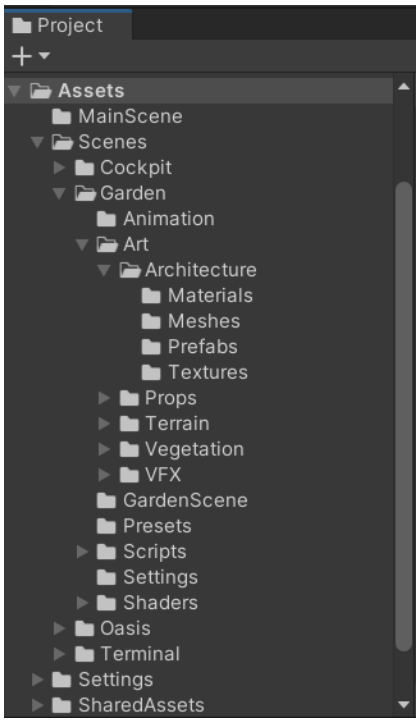


Unity Hub でダウンロード可能なテンプレート

どのテンプレートを選んだかによりますが、いくつかの一般的なアセットを表すサブフォルダーが表示されるはずです。種類別に整理する一例をご紹介します。

Animations	アニメーションには、アニメーションのモーションクリップとそのコントローラーファイルが含まれます。ゲーム内のシネマティクス用の Timeline アセットや、プロシージャルアニメーション用のリギング情報が含まれる場合もあります。
Audio	サウンドアセットには、オーディオクリップやエフェクトと音楽をブレンドするためのミキサーが含まれます。
Editor	ここには、Unity エディターで使用するために作られた、ターゲットビルドには含まれないスクリプトツールが含まれます。
Fonts	このフォルダーには、ゲーム内で使用されるフォントが含まれます。
Materials	これらのアセットには、サーフェスのシェーディングプロパティが記述されています。

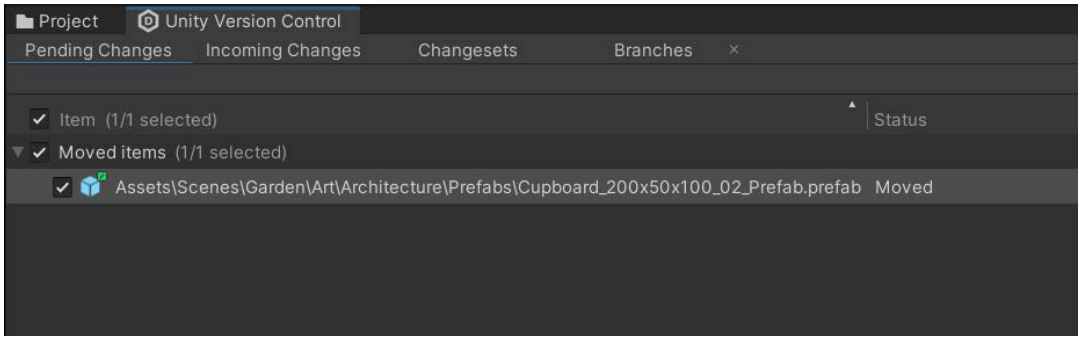
Meshes	外部のデジタルコンテンツ作成（DCC）アプリケーションで作成したモデルをここに保存します。
Particles	ビルトインパーティクルシステムまたは Visual Effect Graph で作成された、Unity のパーティクルシミュレーションです。
Prefabs	事前に作成されたコンポーネントを持つ、再利用可能なゲームオブジェクトです。
Scripts	ユーザーが開発したゲームプレイ用のコードはすべてここに配置します。
Scenes	Unity は、プロジェクトの機能的な部分を小分けにして Scene アセットに保存します。これらはしばしば、ゲームのレベルやその一部に対応します。
Settings	HD レンダーパイプライン (HDRP) やユニバーサルレンダーパイプライン (URP) などの、レンダーパイプラインの設定を保存するのに使用します。
Shaders	これらのプログラムは、グラフィックスパイプラインの一部として GPU で実行されます。
Textures	画像ファイルには、マテリアルやサーフェス用のテクスチャファイル、ユーザーインターフェース用の UI オーバーレイ要素、ライティング情報を保存するためのライトマップなどが含まれます。
ThirdParty	アセットストアなど外部ソースから取得したアセットは、プロジェクト内の他のファイルとは区別してここに保存しましょう。これにより、サードパーティ製のアセットやスクリプトのアップデートが容易になります。 サードパーティ製のアセットは、変更できない特定の構造を持っている場合があります。
UI	UI Toolkit を使用する場合、UXML および USS ファイルをここに保存します。



URP テンプレートを使用したサンプルシーンには、多くのアセットフォルダーが含まれる。

最初にきちんとプロジェクト構成を定義しておくことで、後からバージョン管理の問題が発生するのを防ぐことができます。アセットをあるフォルダーから別のフォルダーに移動した場合、多くの VCS はファイルを移動したのではなく、あるファイルを削除して新しいファイルを追加したと認識します。これにより、元のファイルの履歴が失われます。

UVCS は Unity 内でのファイル移動に対応しており、移動したファイルの履歴を維持します。ただし、ファイルを移動する際は、Unity エディター内で行い、アセットファイルと一緒に .meta ファイルも移動させることが不可欠です。

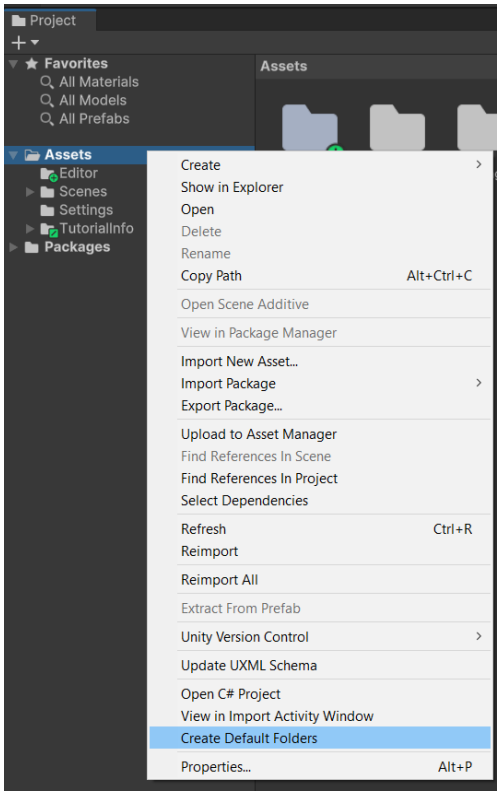


ファイル移動の追跡

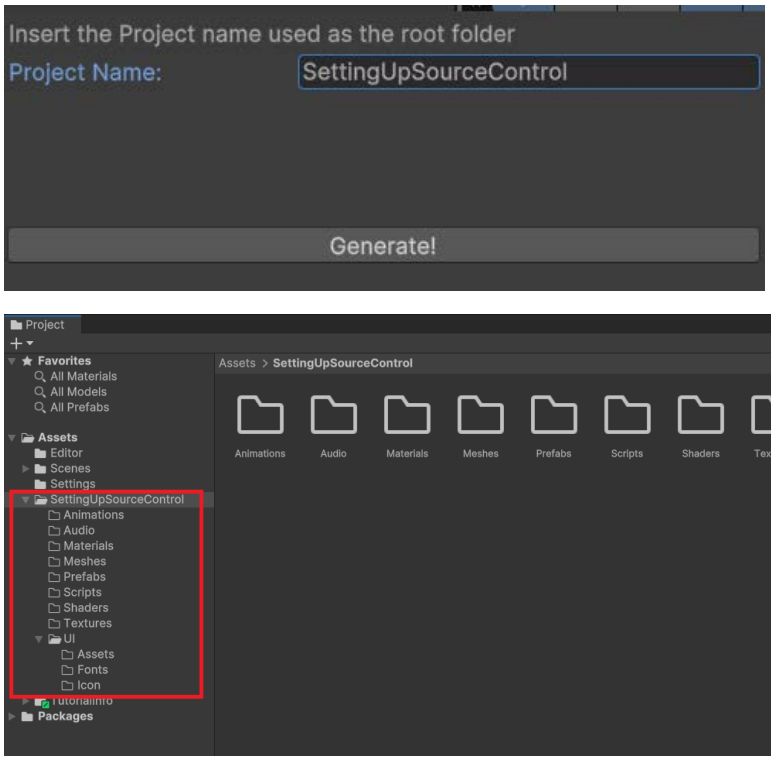
プロジェクトのフォルダー構造を決めたら、エディタースクリプトを使用してテンプレートを再利用し、今後のすべてのプロジェクトで同じフォルダー構造を作成しましょう。以下のスクリプトを Editor フォルダーに配置すると、Assets フォルダー内に「PROJECT_NAME」変数と一致するルートフォルダーが作成されます。こうすることで、サードパーティ製のパッケージと自分の作業ファイルを分けて管理できます。

```
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”,
```

```
    “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(“ルートフォルダーとして使用されるプロジェクト名を挿入してください”);
   projectName = EditorGUILayout.TextField(“プロジェクト名: “,projectName);
   this.Repaint();
   GUILayout.Space(70);
   if (GUILayout.Button(“生成!”)) {
    CreateAllFolders();
    this.Close();
   }
  }
}
```



「Menu」>「Assets」>「Create Default Folders」に移動します。



プロジェクト開始時に空のフォルダーを作成しておくことで、チームの作業を効率的かつ整理された状態に保つことができます。

空のフォルダー

前の画像にあるような空のフォルダーは、バージョン管理で問題を引き起こす可能性があるため、必要なフォルダーのみ作成してください。Git や Perforce では、空のフォルダーはデフォルトで無視されます。プロジェクトフォルダーが作成されても、フォルダー内に何かが入っていなければ、コミットすることはできません。

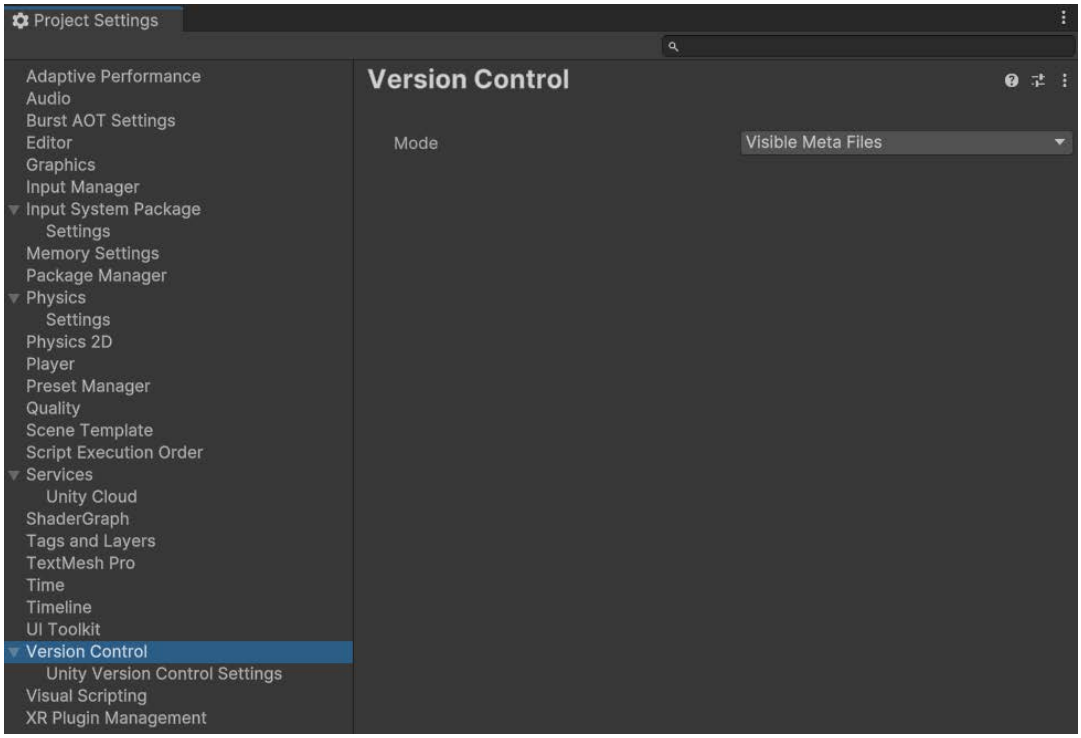
注:一般的な対策として、空のフォルダー内に「.keep」ファイルを置く方法があります。こうすることで、そのフォルダーをリポジトリにコミットできます。

UVCS は空のフォルダーを扱うことができます。ディレクトリはエンティティとして扱われ、紐付けられたバージョン履歴を持ちます。

これは、Unity で作業する際に留意すべきポイントです。Unity は、フォルダーを含むプロジェクト内のすべてのファイルに対して .meta ファイルを生成します。Git や Perforce では、ユーザーは空のフォルダーの .meta ファイルを簡単にコミットできますが、フォルダー自体はバージョン管理下に置かれません。他のユーザーが最新の変更を取得すると、そのユーザーのマシンに存在しないフォルダーの .meta ファイルが生成されるため、Unity はその .meta ファイルを削除します。UVCS は、バージョン管理に空のフォルダーを含めることで、この問題を回避します。

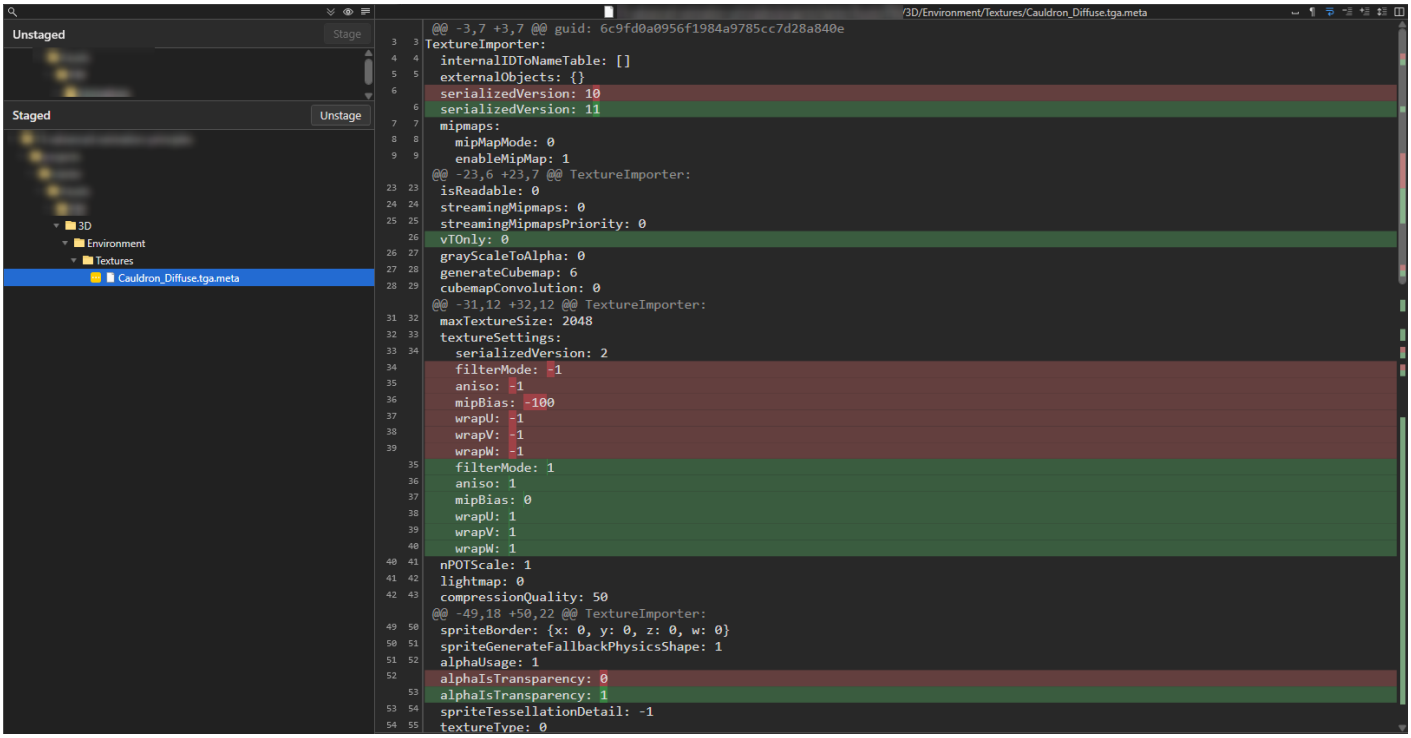
.meta ファイル

Unity はプロジェクト内のすべてのファイルに対して .meta ファイルを生成します。通常、自動生成されたファイルをバージョン管理に含めることは望ましくありませんが、.meta ファイルは少し異なります。UVCS や Perforce モードを使用していない限り、バージョン管理ウィンドウで Visible Meta Files モードをオンにする必要があります。



Git で作業をする際、Visible Meta Files をオンにすること。

.meta ファイルは自動生成されますが、対応するファイルに関する多くの情報を保持しています。これはテクスチャ、メッシュ、オーディオクリップなど、インポート設定があるアセットでよく見られます。これらのファイルのインポート設定を変更すると、変更内容はアセットファイルではなく .meta ファイルに書き込まれます。そのため、.meta ファイルをリポジトリにコミットし、全員が同じファイル設定で作業できるようにします。



ファイルのインポート設定が調整された時に .meta ファイルに加えられる変更

命名基準

プロジェクトのフォルダー構造だけでなく、他の部分でも基準を決定する必要があります。シーン内のゲームオブジェクトやプロジェクトフォルダー内のプレハブに命名基準を設定すると、チーム内で他のメンバーのファイルを扱う際に内容を理解しやすくなります。

ゲームオブジェクトの明確な命名規則はありませんが、以下を検討してください。

基準	例
説明的な名前を使い、省略は避けましょう。数か月後でも覚えていられるような名前を使いましょう。他の人があなたの命名を理解できるかどうかを考慮し、発音しやすく覚えやすい名前を選ぶべきです。このため、一般的に省略した名前は推奨されません。	largeButton、LargeButton または leftButton 悪い例： lButton
キャメルケースまたはパスカルケースを使いましょう。オブジェクト名にスペースを含めないようにしてください。キャメルケースやパスカルケースを使用すると、可読性が向上し、この調査によれば、タイピングの正確性も向上します。	OutOfMemoryException、dateTimeFormat 悪い例： Outofmemoryexception、datetimeformat

アンダースコア（またはハイフン）の使用は控えめにしましょう。一般的に、アンダースコアやハイフンの使用は避けるべきです。しかし、特定の状況では有用です。名前の先頭にアンダースコアを付けると、アルファベット順で最初に表示されます。また、アンダースコアを使って、ファイルが特定のオブジェクトのバリアントだと示すこともできます。	アクティブステート： EnterButton_Active、EnterButton_Inactive テクスチャマップ： Foliage_Diffuse、Foliage_Normalmap Level of Detail： Building_LOD1、Building_LOD0
数字のサフィックスを使って連番を表しましょう。同様に、リストの一部でない場合は数字のサフィックスを使用しないようにします。	パスの場合、ノードに以下のような名前をつけましょう。 Node0、Node1、Node2、など
デザインドキュメントの命名に従いましょう。	デザインドキュメントに HighSpellTower や RedDragonLair のような地名が記載されている場合、その記載通りの綴りを使用してください。

チーム全体で一貫したスタイルを用いることで、よりクリーンで読みやすく、スケーラブルなプロジェクトを実現できます。Unity e ブック「C#スタイルガイドを作成する：拡張性のある、よりクリーンなコードを書く」では、命名規則、形式、クラス、メソッド、コメントなどに関するヒントやベストプラクティスを紹介しています。このガイドは全体的に Microsoft C# スタイル標準 2 に従っており、Unity 固有のサブセットを提供していますが、1 つの「正確」があるわけではありません。Google C# 2 ガイドも、命名、形式、コメント規則に関するガイドラインを定義するための優れたリソースです。

ワークフローの最適化

Assets フォルダー内でのアセットの配置や管理方法に加え、特にバージョン管理を使用している場合、ワークフローを効率化するためのデザインや開発上の工夫がいくつかあります。

アセットの分割

大規模な単一の Unity シーンは、コラボレーションに適していません。アーティストやデザイナーが 1 つのレベルでより効果的に協力し、競合のリスクを最小限に抑えることができるよう、レベルをより小さなシーンに分割しましょう。

実行時には、SceneManager.LoadSceneAsync を使用し、LoadSceneMode.Additive パラメータを渡すことで、シーンを加算的にロードできます。

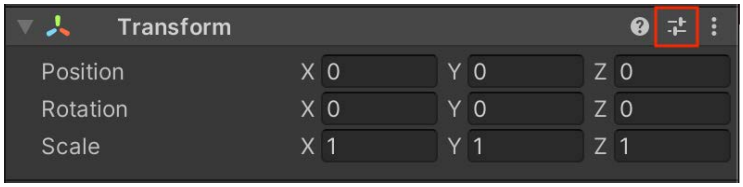
さらに、可能な限り作業内容をプレハブに分割しましょう。後で変更が必要になった場合、シーンではなくプレハブを変更することで、シーンで作業している他の人との競合を避けることができます。プレハブの変更は、バージョン管理下で差分を確認するときに、より読みやすい場合が多いです。

そして、万一シーンの競合が発生した場合でも、Unity にはシーンやプレハブをマージするためのビルトイン YAML（人間にも解読可能なデータシリアライズ言語）ツールが用意されています。詳しくは、Unity ドキュメントのスマートマージを参照してください。

プリセット

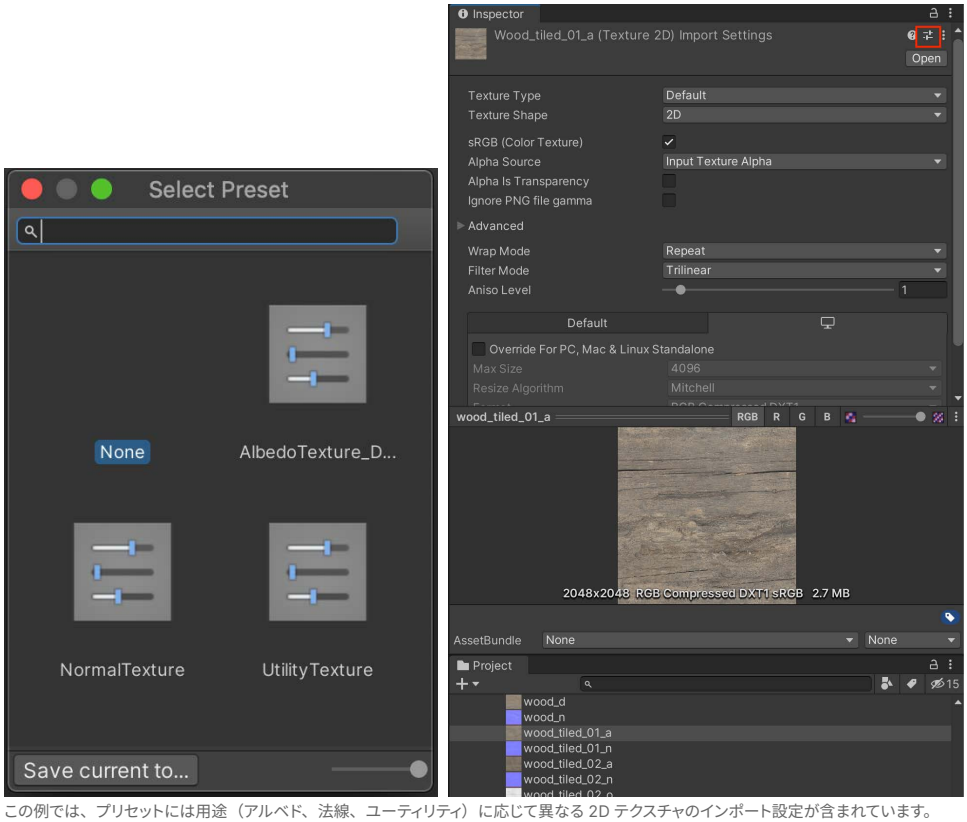
Unity プリセットとは、プロジェクト内の異なるコンポーネント、アセット、またはツール間で特定の設定を保存して再利用することを可能にする、あらかじめ定義された設定です。プリセットを作成すると、コンポーネントやアセットの設定をコピーしてアセットとして保存し、後で同じ設定を別のアイテムに適用できます。

プリセットを使用して基準を適用したり、新しいアセットに適切なデフォルトを設定したりできます。これにより、チーム全体で一貫した基準が確保され、一般的に見落とされがちな設定がプロジェクトのパフォーマンスに影響を与えることがなくなります。



プリセットアイコンは、ここでは赤でハイライトされています。

コンポーネントの右上にあるプリセットアイコンをクリックします。「**Save current to…**」をクリックし、プリセットをアセットとして保存します。利用可能なプリセットのいずれかをクリックし、値のセットをロードします。



この例では、プリセットには用途（アルベド、法線、ユーティリティ）に応じて異なる 2D テクスチャのインポート設定が含まれています。

プリセットの他の便利な使い方には、以下のようなものがあります。

- **デフォルトでゲームオブジェクトを作成する：**プリセットアセットを階層にドラッグ&ドロップすると、プリセット値を含む対応するコンポーネントを持つ新しいゲームオブジェクトが作成されます。
- **特定のタイプをプリセットに関連付ける：**Preset Manager（**「Project Settings」>「Preset Manager」**）で、タイプごとに 1 つ以上のプリセットを指定します。新しいコンポーネントを作成すると、指定されたプリセット値がデフォルトになります。
    - ヒント：タイプごとに複数のプリセットを作成し、フィルターを活用することで名前によって正しいプリセットを関連付けましょう。
- **マネージャー設定をセーブまたはロードする：**マネージャーウィンドウにプリセットを使用して、設定を再利用できるようにしましょう。例えば、同じタグやレイヤー、物理設定を再適用する予定がある場合、プリセットを使用すれば次のプロジェクトの設定にかかる時間を短縮できます。

コーディング基準

また、コーディング基準を設けることで、チームの作業の一貫性を保ち、開発者がプロジェクト内の異なる領域をスムーズに切り替えることが容易になります。ただし、ここに絶対的なルールがあるわけではありません。チームにとって最適な方法を決め、それを守り続けることが重要です。

例えば、名前空間はコードを整理するのに役立ちます。これによってプロジェクト内でモジュールを分離することが可能になり、クラス名が重複する可能性のあるサードパーティ製アセットとの競合を避けることができます。

コードに名前空間を使用する場合は、フォルダー構造を名前空間ごとに分割すると整理しやすくなります。

標準的なヘッダーを使用するのも良い習慣です。コードテンプレートに標準的なヘッダーを含めると、クラスの目的、作成日、さらには*作成者*を記録するのに役立ちます。これらはバージョン管理を使っていても、プロジェクトの長期間にわたる開発の中で簡単に失われてしまう情報です。

Unity では、プロジェクトで新しい Monobehaviour を作成するたびに、テンプレートスクリプトを使用します。新しいスクリプトまたはシェーダーを作成するたびに、Unity は **%EDITOR_PATH%\Data\Resources\ScriptTemplates** に保存されているテンプレートを使用します：

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

デフォルトの MonoBehaviour テンプレートはこちらです。**81-C# Script-NewBehaviourScript.cs.txt**

また、シェーダーやその他のビヘイビアスクリプト、アセンブリ定義のテンプレートも用意されています。

プロジェクト固有のスクリプトテンプレートに関しては、Assets/ScriptTemplates フォルダーを作成し、スクリプトテンプレートをこのフォルダーにコピーしてデフォルトのものを上書きします。

また、すべてのプロジェクトのデフォルトのスクリプトテンプレートを直接変更することもできますが、変更する前に必ずオリジナルをバックアップしてください。Unity の各バージョンには独自のテンプレートフォルダーがあるため、新しいバージョンにアップデートした場合は、テンプレートを再度置き換える必要があります。

オリジナルの 81-C# Script-NewBehaviourScript.cs.txt ファイルは以下のようになっています。

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

#ROOTNAMESPACEBEGIN#
public class #SCRIPTNAME# :MonoBehaviour
{
 // 最初のフレームが更新される前に Start が呼び出される
 void Start()
 {
  #NOTRIM#
 }

 // フレーム毎に 1 回 Update が呼び出される
 void Update()
 {
  #NOTRIM#
 }
}
#ROOTNAMESPACEEND#
```

役に立つ可能性のあるキーワードが 2 つあります。

— **#SCRIPTNAME#** は、入力されたファイル名またはデフォルトのファイル名（例えば NewBehaviourScript など）を示します。

— **#NOTRIM#** は、かっこ内に空白行を残すための指定です。

独自のキーワードを使用し、OnWillCreateAsset メソッドを実装したエディタースクリプトで置き換えることもできます。

```
// /*-------------------------------------------
// -------------------------------------------
// 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();
 }
}
```

独自のキーワードを使用し、OnWillCreateAsset メソッドを実装したエディタースクリプトで置き換えることもできます。

UI Toolkit の形式規則

UI Toolkit は、UXML コードを使用します。標準的なウェブ技術に着想を得た UI Toolkit は、CSS 関連のスタイリングシステムと同じく、クラスの命名にケバブケース（ダッシュケースまたはリスプケースとも呼ばれる）を使用します。

クラスと同様に、HTML（および UXML）では ID 名にケバブケース（ダッシュ付きの小文字）を使うのが一般的です。そのため、my-element-id のような名前がより標準的です。

しかし、ID 名は要素の目的を簡単に特定できるよう、具体的で説明的なものにすることが推奨されています。たとえば、submit-button、main-nav-container、logo-image などです。

	推奨	非推奨
クラス	.menu-bar-blue	ButtonBlue
ビジュアル要素の ID	main-nav-image	

UXML で推奨される命名規則

ケバブスタイルを使うべき主な理由には、以下のようなものが含まれます。

- CSS セレクターは大文字小文字を区別しないため、（.buttonBlue や .ButtonBlue など）キャメルケースやパスカルケースを使用すると混乱を招く可能性があります。
- HTML の仕様では、属性名はすべて小文字にすることが推奨されており、.buttonBlue や .ButtonBlue ではなく .button-blue とします。
- 複数の単語が含まれる場合、ケバブケースのほうが読み書きしやすいです。

どのような命名規則を使用するにせよ、コードベース全体で一貫性を保つことが重要です。あらゆる種類のスクリプトに一貫した命名規則があると、コードがすっきりし、アクセスや修正がチームメンバー全員にとって容易になります。

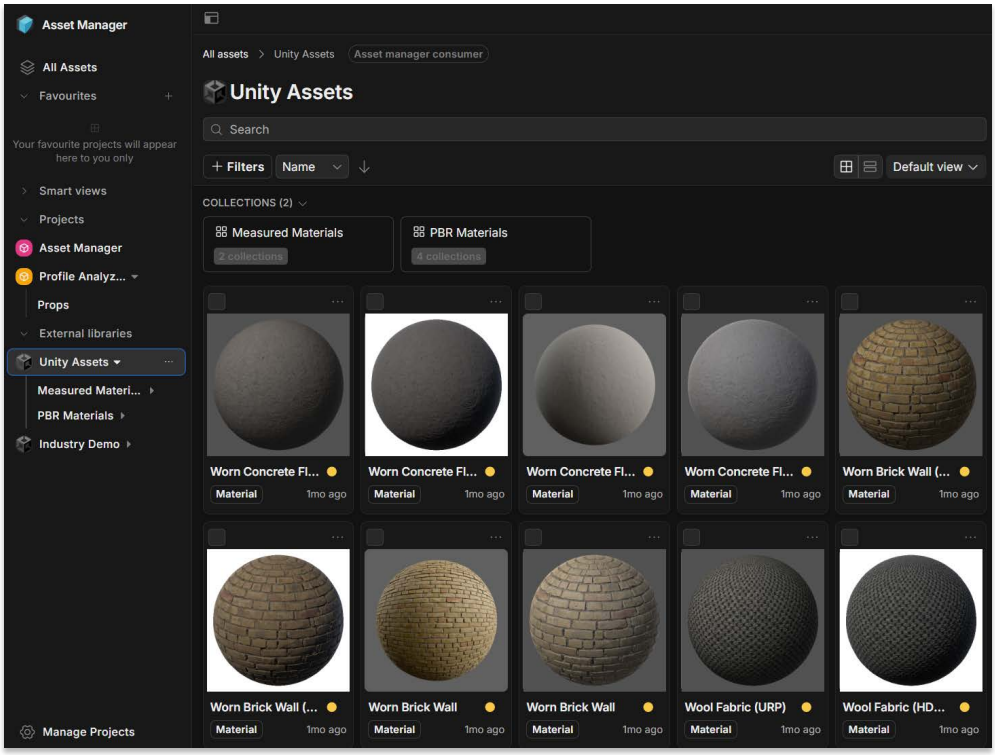
プロジェクト整理用のサービス

Asset Manager

Unity Asset Manager は Unity の拡張可能なクラウドベースのデジタルアセット管理（DAM）ソリューションで、組織全体においてコンテンツの検索性、再利用性、ROI を高めることができます。Asset Manager では、クラウドストレージを使用して Unity プロジェクトのコンテンツをインデックス化できるため、拠点が異なる場合でもチーム全体で簡単にアセットを検索できます。主な機能は以下の通りです。

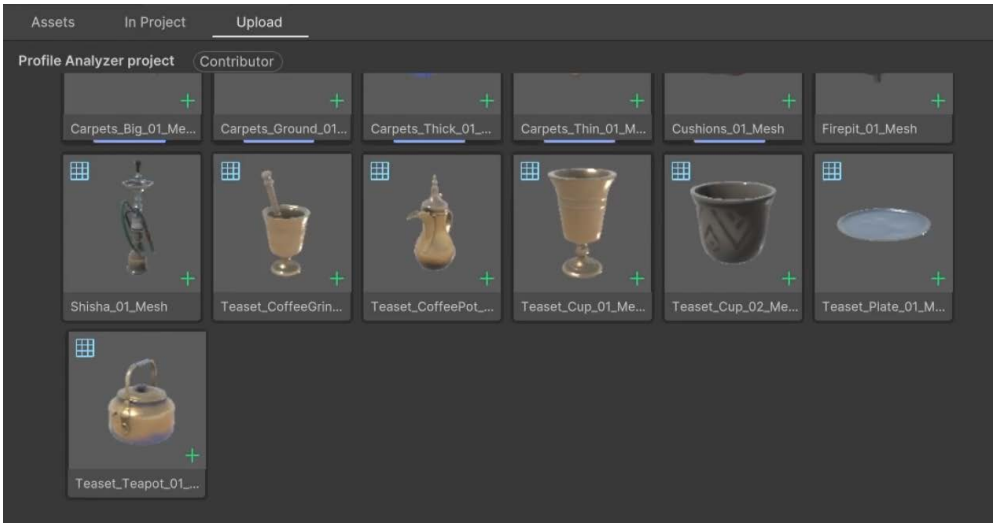
- より効果的なアセット管理
- 直感的なブラウジングおよび検索
- クリエイティブツールとの不可欠なインテグレーション
- ライフサイクル管理
- ロールベースの権限
- 柔軟性および拡張性

Unity Cloud にログインすると、ダッシュボードから無料のマテリアル、テクスチャ、デモのコレクションを利用できます。**Unity Assets** サブセクションを開き、これらをプロジェクト内にダウンロードしましょう。



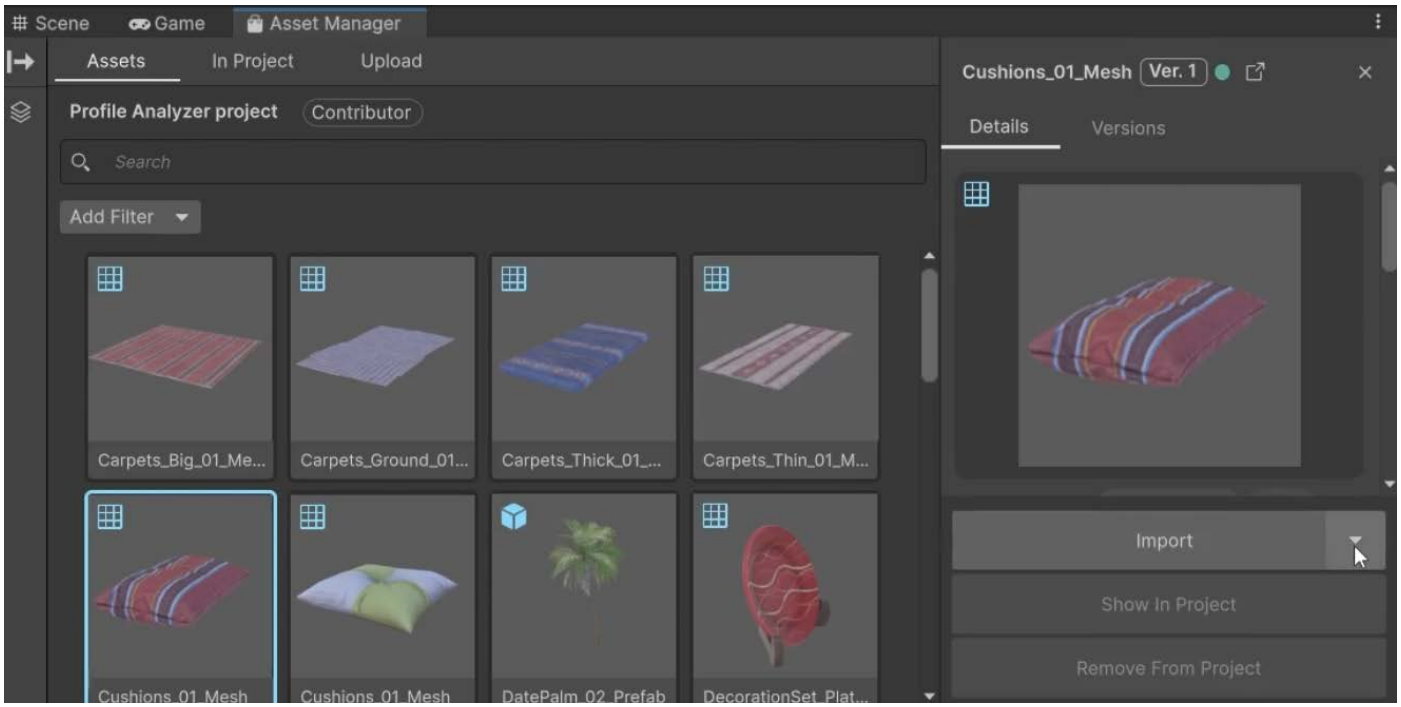
Unity Cloud の「Unity Assets」セクションにある無料のマテリアルやテクスチャ

Unity 内では、現在のプロジェクトから **Asset Manager** ウィンドウにファイルをドラッグすることで、クラウドにアップロードできます。そうすると、Asset Manager を使用している他のプロジェクトや、UVCS を使用してプロジェクトに接続しているチームメンバーもこのファイルにアクセスできるようになります。



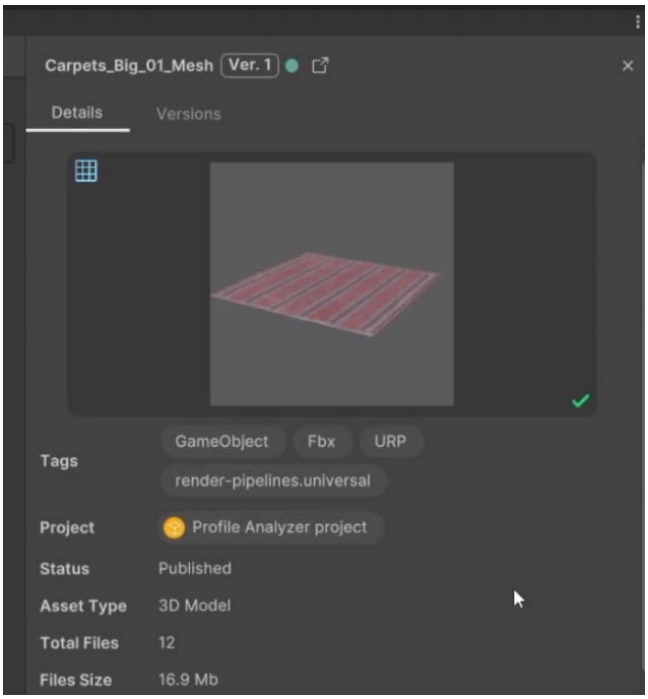
アセットを「Upload」タブにドラッグしてクラウドにアップロードします。

その後、Asset Manager ウィンドウから直接、他のプロジェクトにファイルをインポートできるので、簡単にすべてのプロジェクトアセットの包括的なアーカイブを作成することが可能です。



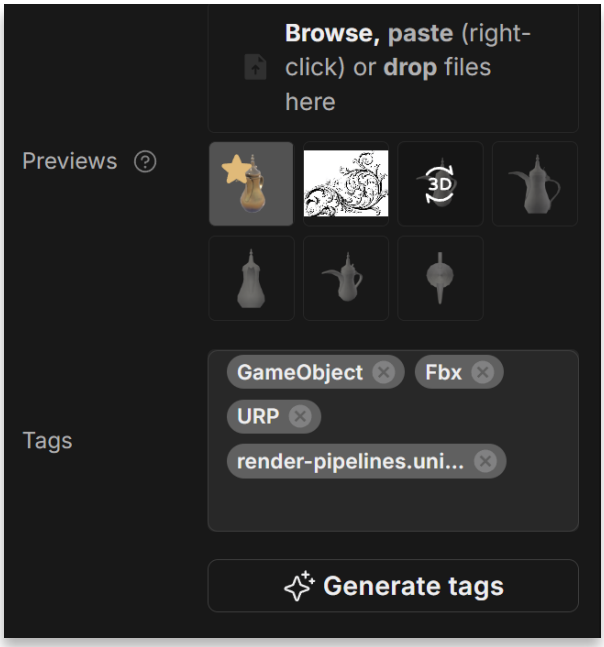
「Import」をクリックしてファイルをダウンロードし、プロジェクトにインポートします。

タグが自動生成されるため、多くのアセットがある場合でも素早く検索することができます。アセットはプロジェクトにも紐付けられているので、特定のプロジェクトやアーティストに属するアセットに限定して検索することも可能です。



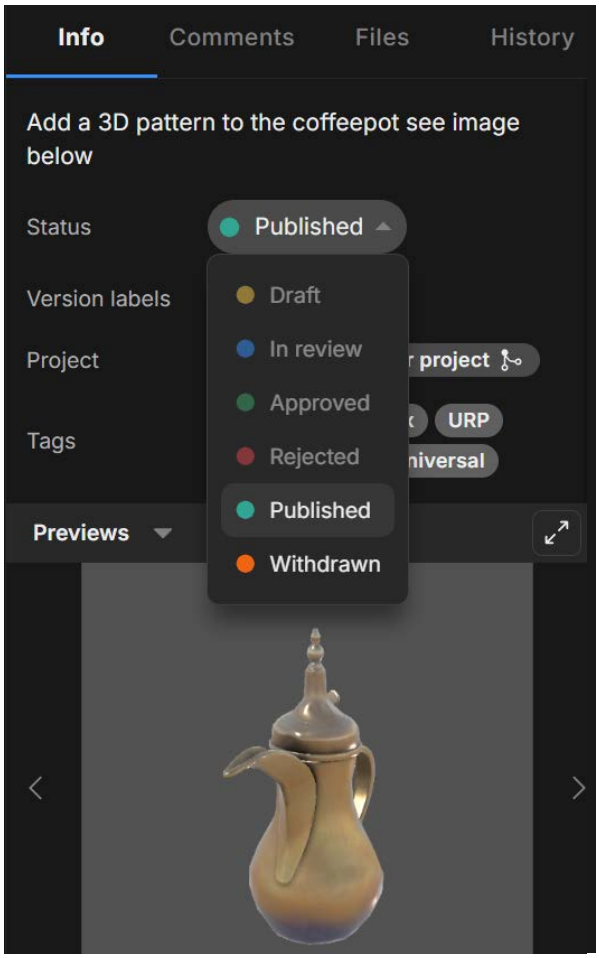
自動生成されたタグにより、アセットをすばやく簡単に検索できます。

アセットは Unity Cloud で編集することができます。カスタムサムネイル、画像プレビュー、タグ、コメントを追加できます。



アセットを編集して画像プレビューやタグを追加する。

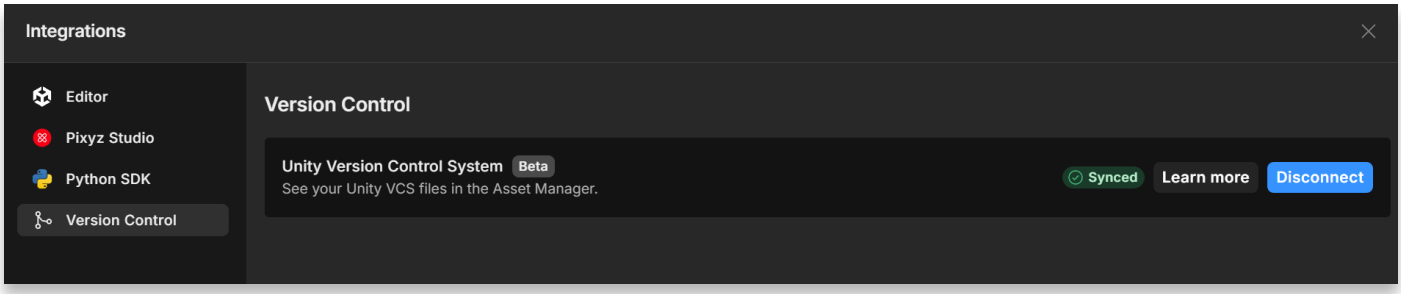
モデルのステータスはクラウド上で変更可能で、アセットがドラフト中、レビュー中、承認済み、却下済み、公開済み、または取り下げ済みであることを他の人に知らせることができます。コメントを追加すると、変更内容についての詳細を含めることができます。その後、アセットのステータスは Unity の Asset Manager ウィンドウに表示されます。



アセットのステータスを変更する。

チームメンバーは常に最新バージョンにアクセスできるため、アセットのイテレーション作業時の問題が軽減されます。

Asset Manager は、クラウド上で統合を有効にすることで UVCS と連携します。



データを同期するには、Asset Manager を UVCS に統合します。

UVCS で行った変更は自動的に Unity Asset Manager と同期され、チーム内での重複したコンテンツ作成を防ぎます。

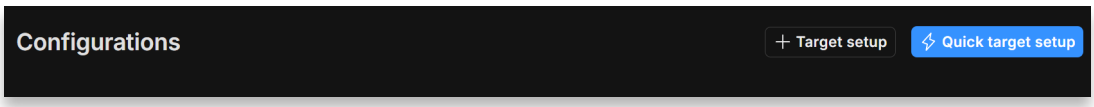
Unity Asset Manager の利点についての詳細はこちらからご覧ください。

Unity Learn でこちらの Asset Manager チュートリアルをご覧いただくことも可能です。

Build Automation

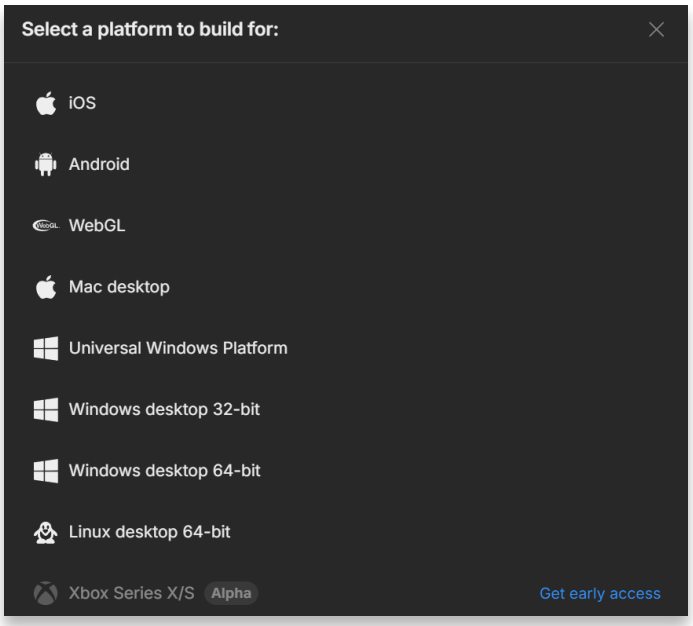
ビルドの自動化は、あらゆる DevOps 戦略において重要です。クラウド上でビルドを実行するだけでなく、すべてのターゲットプラットフォームに対して同時にビルドを実行することもできます。デバイス上でビルドが順次完了するのを待つ必要がなくなるため、その時間をプロジェクトの作成に費やすことが可能になります。

Unity Cloud ダッシュボードの「**Configurations**」セクションで、必要なだけビルドターゲットを設定できます。



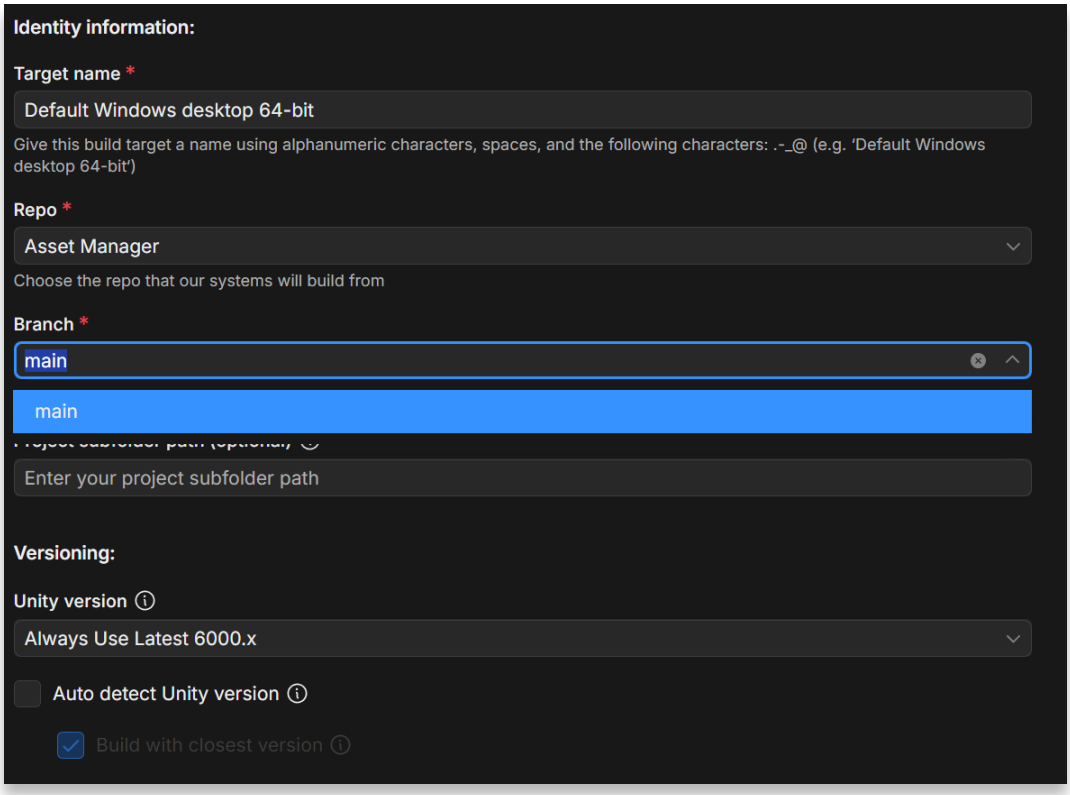
「**Quick target setup**」をクリックしてビルドターゲットを作成します（より詳細なオプションを設定したい場合は「Target setup」をクリックしてください）。

どのプラットフォーム向けにビルドするかを選択します。各プラットフォームに対し、多数のビルドターゲットを設定できます。



ビルドするターゲットプラットフォーム

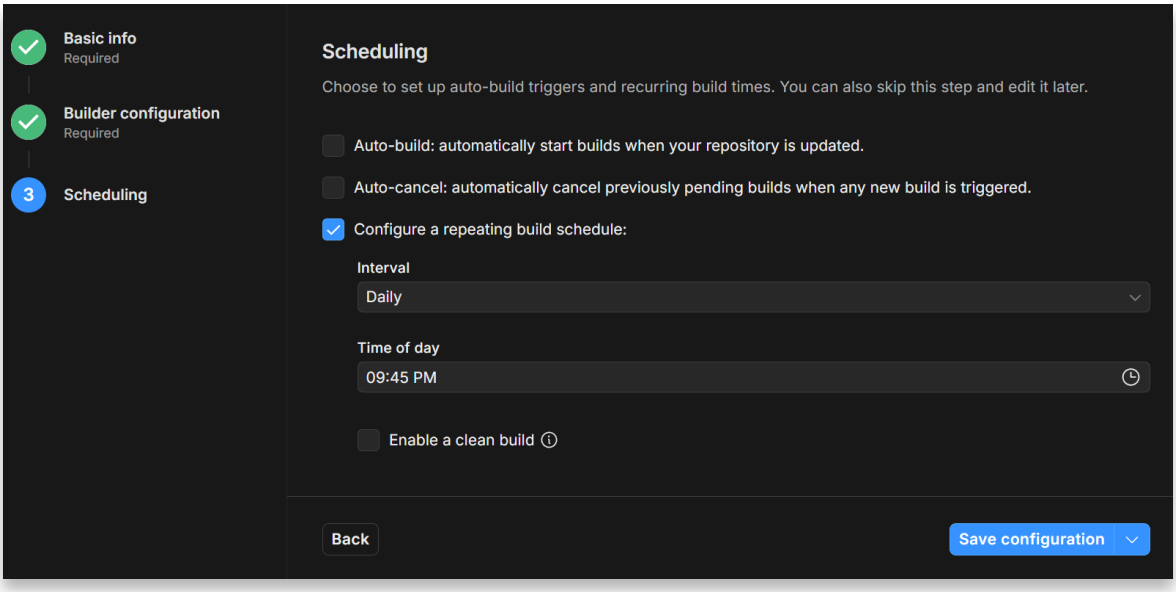
どのブランチからでもビルドでき、ブランチごとに複数のターゲットを設定できます。また、Unity の特定のバージョンを選択することもできます。



ビルドを作成するブランチを選択します。

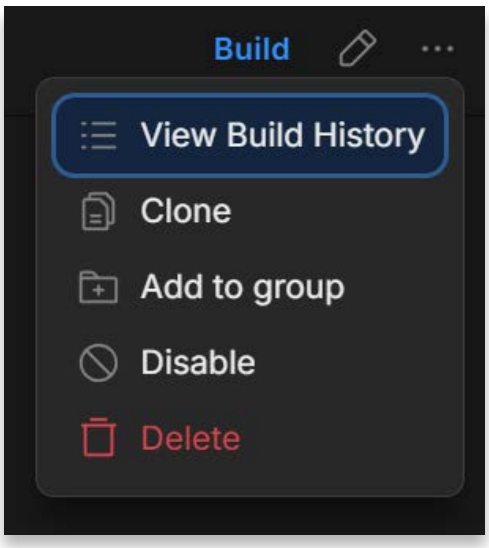
注：Builder オペレーティングシステムの Windows 11 では、本稿執筆時点で、月 200 分のビルド時間が無料で提供されています。

その後、スケジュールを設定します。現バージョンをテストするために一度だけビルドすることも、繰り返しのスケジュールを設定することも可能です。以下の例は、毎朝最新のゲームビルドを確認して問題やバグをチェックできるよう、午後 9 時 45 分にビルドを生成する日次スケジュールを示しています。ビルドが完了すると、成功か失敗かのステータスを知らせるメールが届きます。



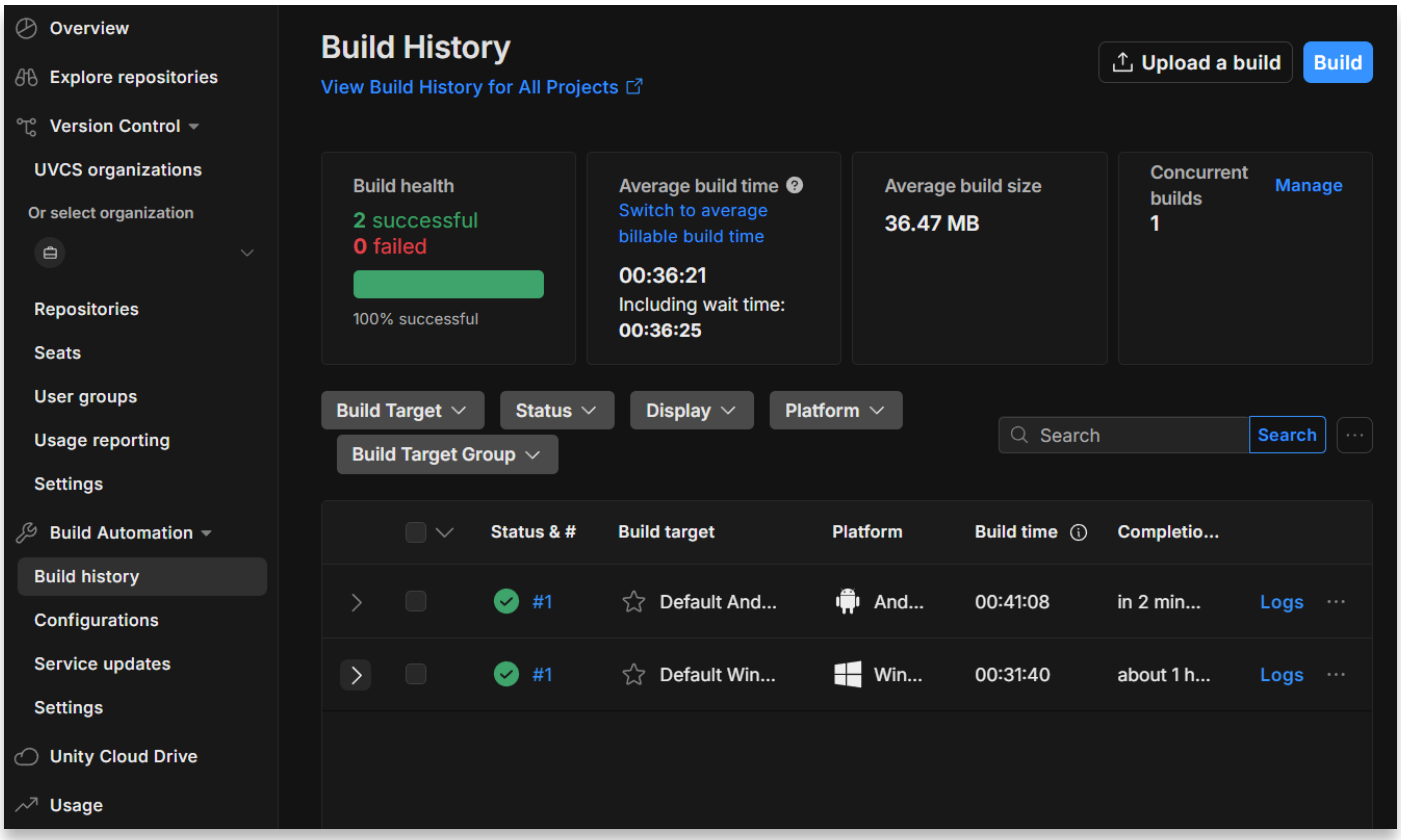
ビルドのスケジュール作成

設定のセクションで、セットアップの横の「**Build**」をクリックすると、すぐにビルドを開始できます。ビルド設定を一時停止または削除することも可能です。



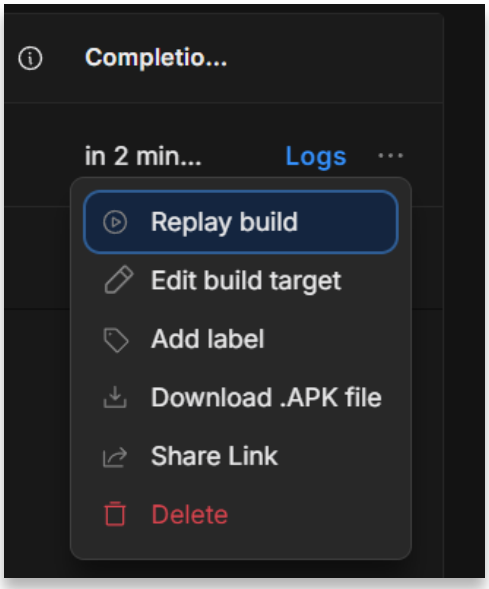
ビルドスケジュールを無効化または削除する

Build Automation ダッシュボードの「**Build History**」セクションで、ビルドの結果を確認できます。いずれかのビルドが失敗した場合は、ログを確認してトラブルシューティングを行うことができます。



クラウドのビルド履歴

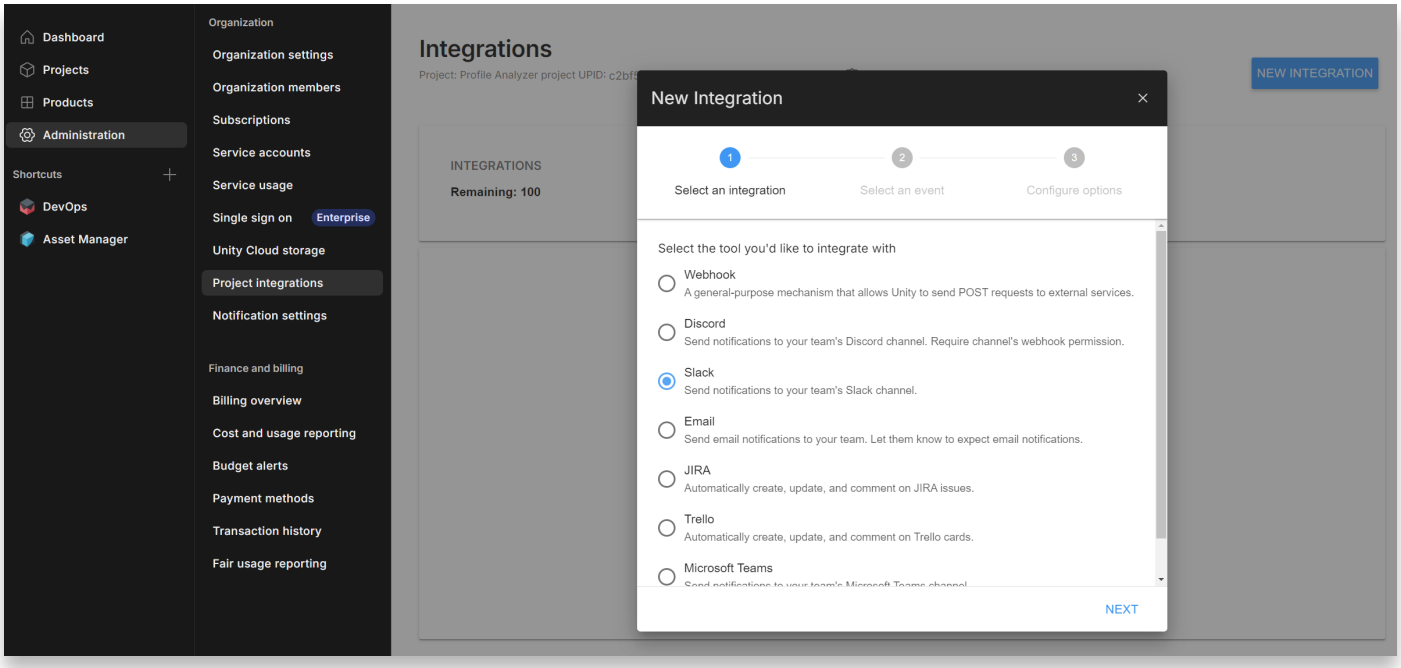
クラウドからビルドをダウンロード、共有、または削除できます。



ビルドのダウンロード、共有、削除

ビルド完了後に Discord や Slack などの他のアプリケーションにメッセージを送信したい場合、インテグレーションを設定します。

クラウドで「**Administration**」に移動し、「**Project Integrations**」を選択して、「**New Integration**」をクリックします。



「Administration」>「Project Integrations」から新たなインテグレーションを追加します。

使用したいアプリケーションを選択します。これで、該当するアプリケーションを通じて、チームメンバーに通知が送信されるようになります。

Unity Build Automation の詳細はこちら。

バージョン管理システム

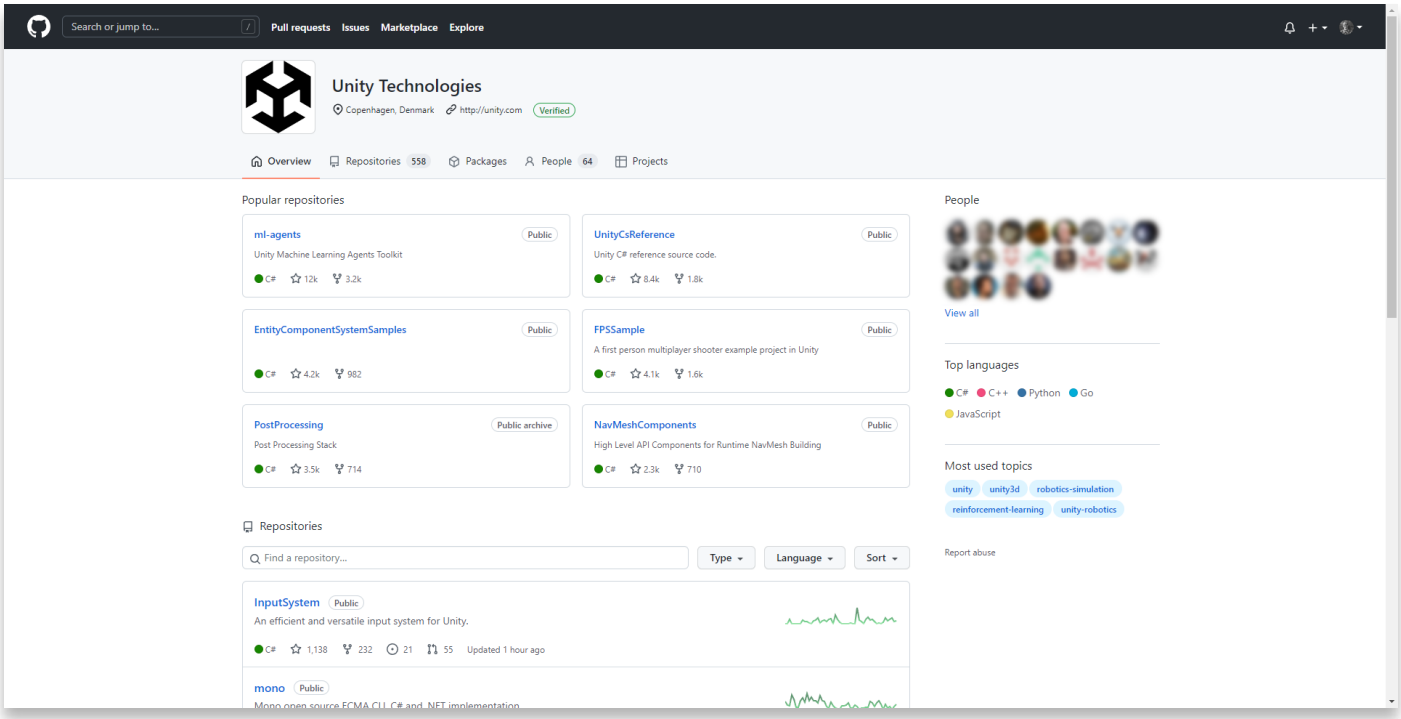
バージョン管理、プロジェクト編成、命名規則のベストプラクティスにおける重要な用語や概念を学んだところで、主要なバージョン管理システムをご紹介します。もちろん、すべての人にとって最適なソリューションは存在しません。チーム内でどの VCS を採用するか選択する際に、検討すべきことはたくさんあります。本書を通じて、その決断に必要な情報をすべて手に入れていただければ幸いです。

Git

オープンソースで無料、かつ柔軟性の高い Git は、もっとも人気のあるバージョン管理システムのひとつです。しかし、分散型であるため、技術的な知識がないユーザーには難しく感じるかもしれません。

2005 年に Linus Torvalds によって Linux カーネル開発をコントロールするために開発されて以降、適切にメンテナンスされ、オープンソースとして提供され続けています。プラットフォームとしての Git は、コマンドライン専用のツールです。しかし、Git 向けに様々な GUI が開発されており、ユーザーにとってより使いやすいシステムになっています。

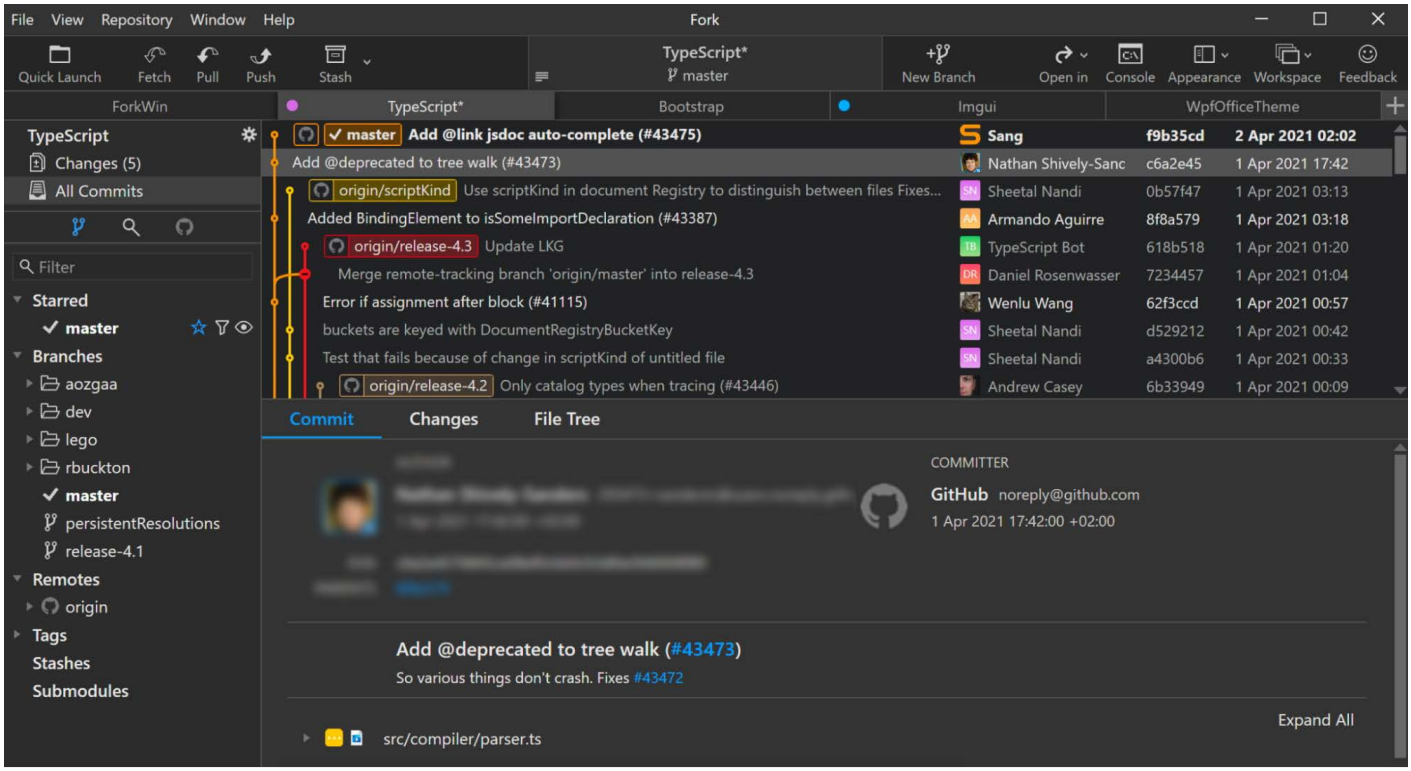
Git と GitHub は、混同されることがよくあります。GitHub は Git リポジトリのホスティングサービスですが、GitHub を使わずに Git を利用することも可能です。とはいえ、GitHub は（いくつかの制限はありますが）無料版があり、カスタムサーバーのセットアップも不要なため、非常に人気のあるサービスです。



GitHub

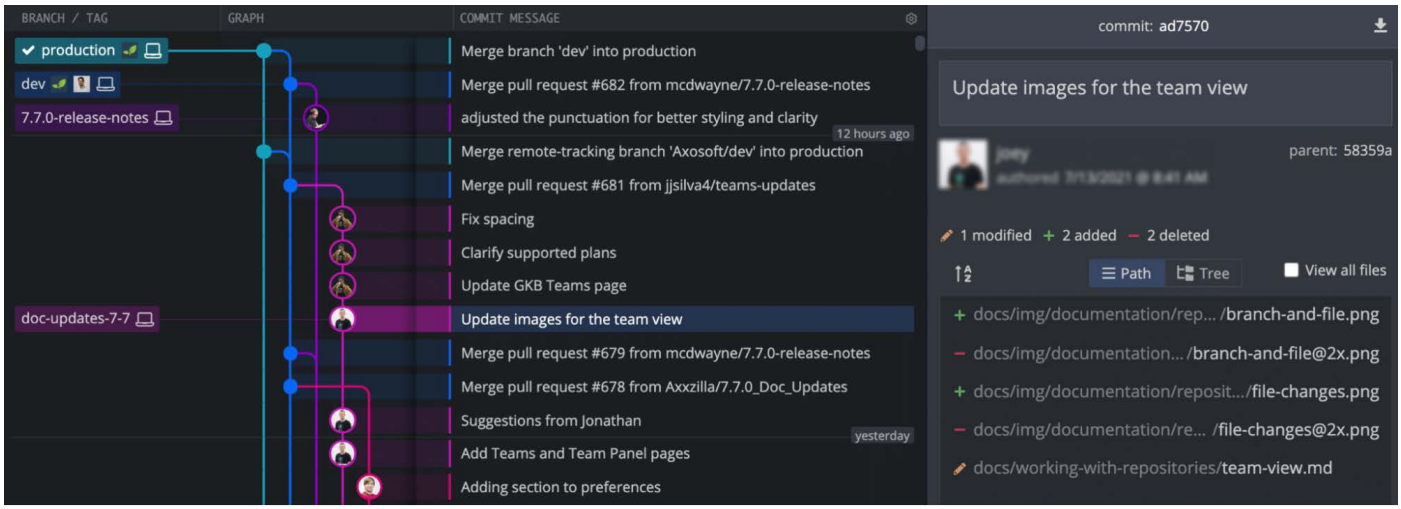
人気のある Git GUI クライアントには、以下のものがあります。

Fork：お試し版を無料でダウンロードできる、高速で使いやすい GUI です。



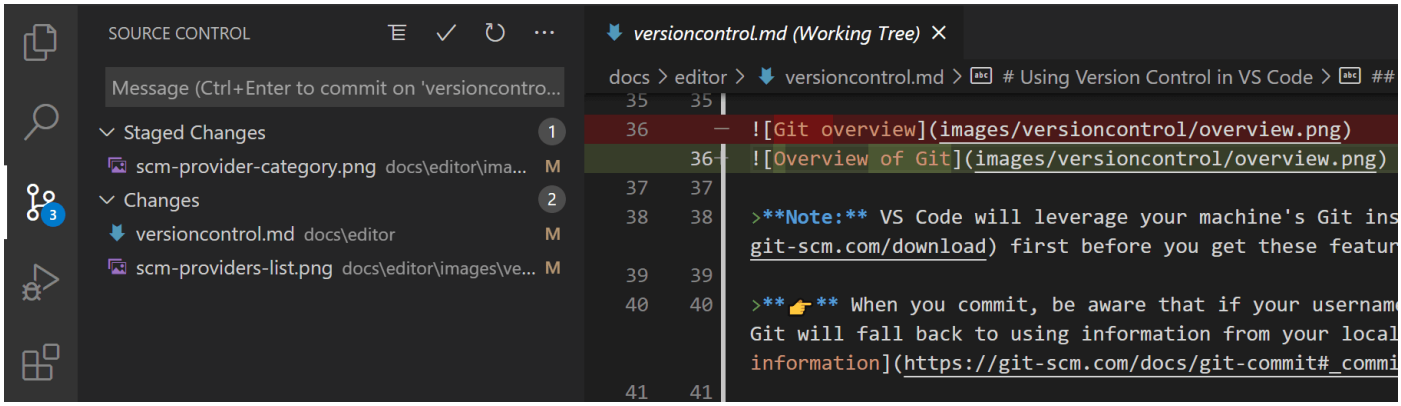
Fork

GitKraken：直感的な UI を備え、GUI と CLI ターミナルを切り替えられる柔軟性があり、Git の操作をより視覚的で使いやすくします。



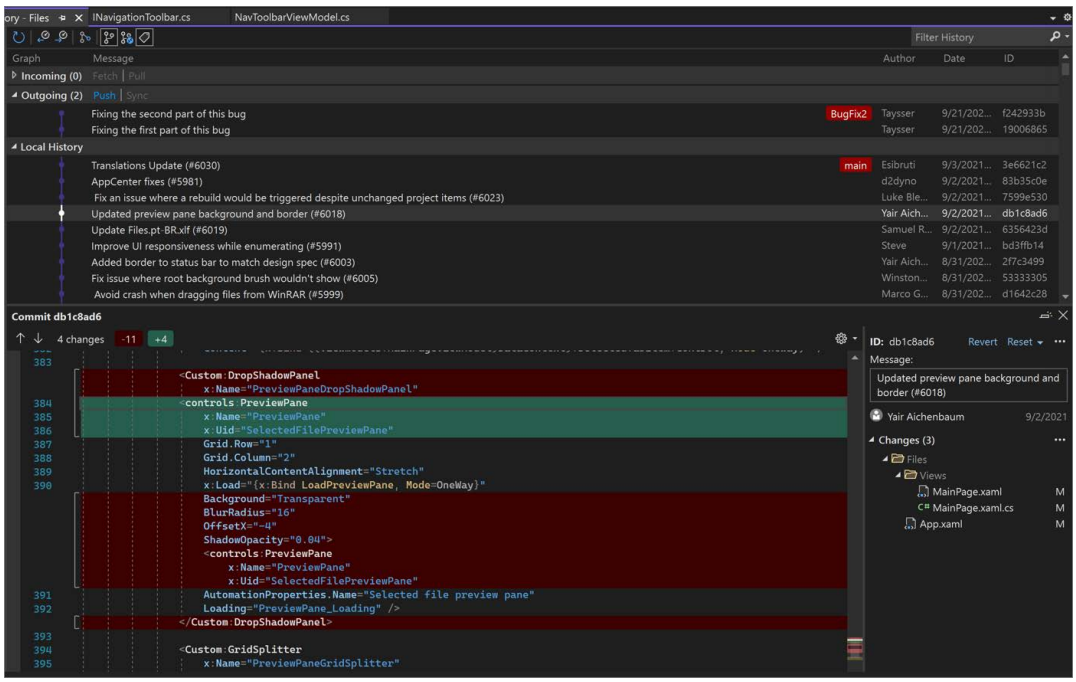
GitKraken

Microsoft Visual Studio Code：VS Code にはソース管理のインテグレーションが組み込まれており、豊富な拡張機能を活用することで、別のプログラムを使う必要がなくなります。



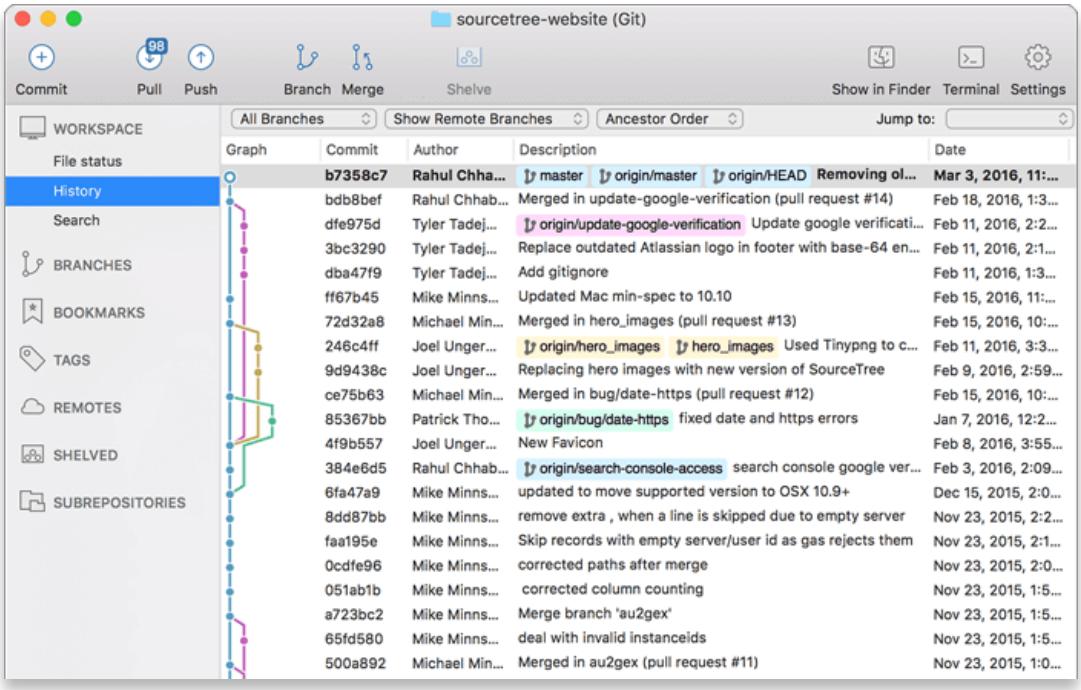
Visual Studio Code

Microsoft Visual Studio：VS Code 同様、Visual Studio にも Git コントロールが組み込まれており、GitHub の拡張機能も含まれています。



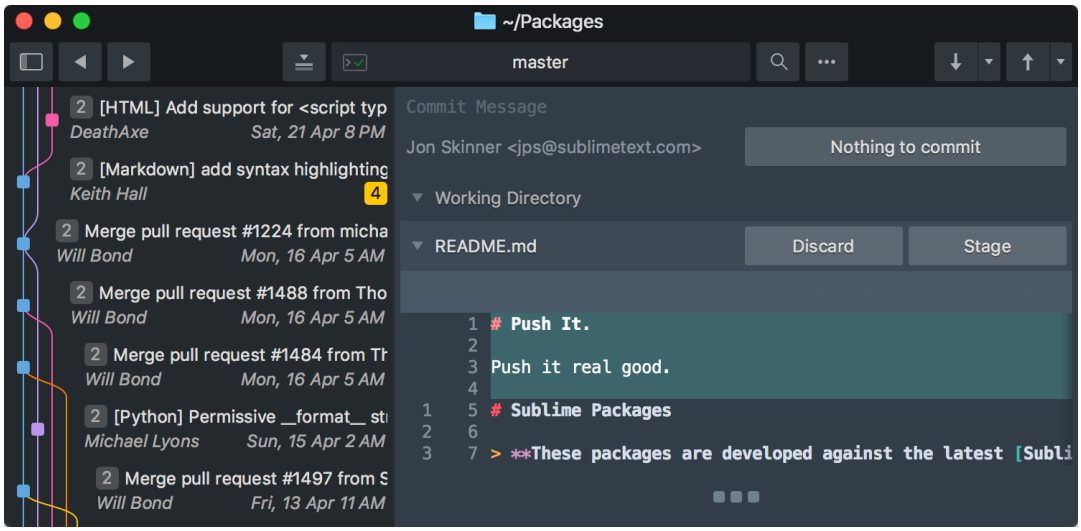
Visual Studio

SourceTree：アトラシアン製品グループの一部である SourceTree は、Windows および Mac 用の無料の Git クライアントで、Git リポジトリを簡単に視覚化して管理できます。



SourceTree

Sublime Merge：このシステムは、差分を並べて表示する機能や構文ハイライト機能を提供し、コードレビューの効率化に役立ちます。軽量かつ高性能なクライアントです。



Sublime Merge

Git は一般的にブランチやマージの機能に優れていると考えられていますが、市場に出ている他のソリューションほど大きなバイナリファイルを効率的に扱うことはできません。Git Large File Storage（LFS）は、この問題をある程度改善します。

Git は分散型クライアントなので、リポジトリ全体と完全な履歴が開発者のマシンに保存されます。このため、ブランチを切り替えたり、過去のある時点に戻ったりといった操作を非常に素早く行うことができます。複数の機能やリリースブランチを持つ大規模なプロジェクトに取り組んでいる場合、Git ワークフローを使用することで多くの時間を節約できます。

Unity は、自社の C# エディターおよびエンジンのコードを GitHub で一般公開しています。これは、特定の機能がどのように動作するかや、エディターの機能を自分のプロジェクト内で再現する方法を知りたいときに非常に役立ちます。

GitHub には、独自の Git GUI、GitHub Desktop もあります。Unity で作業する場合、GitHub for Unity パッケージを使うことで、Git ツールを Unity エディターに直接組み込むこともできます。

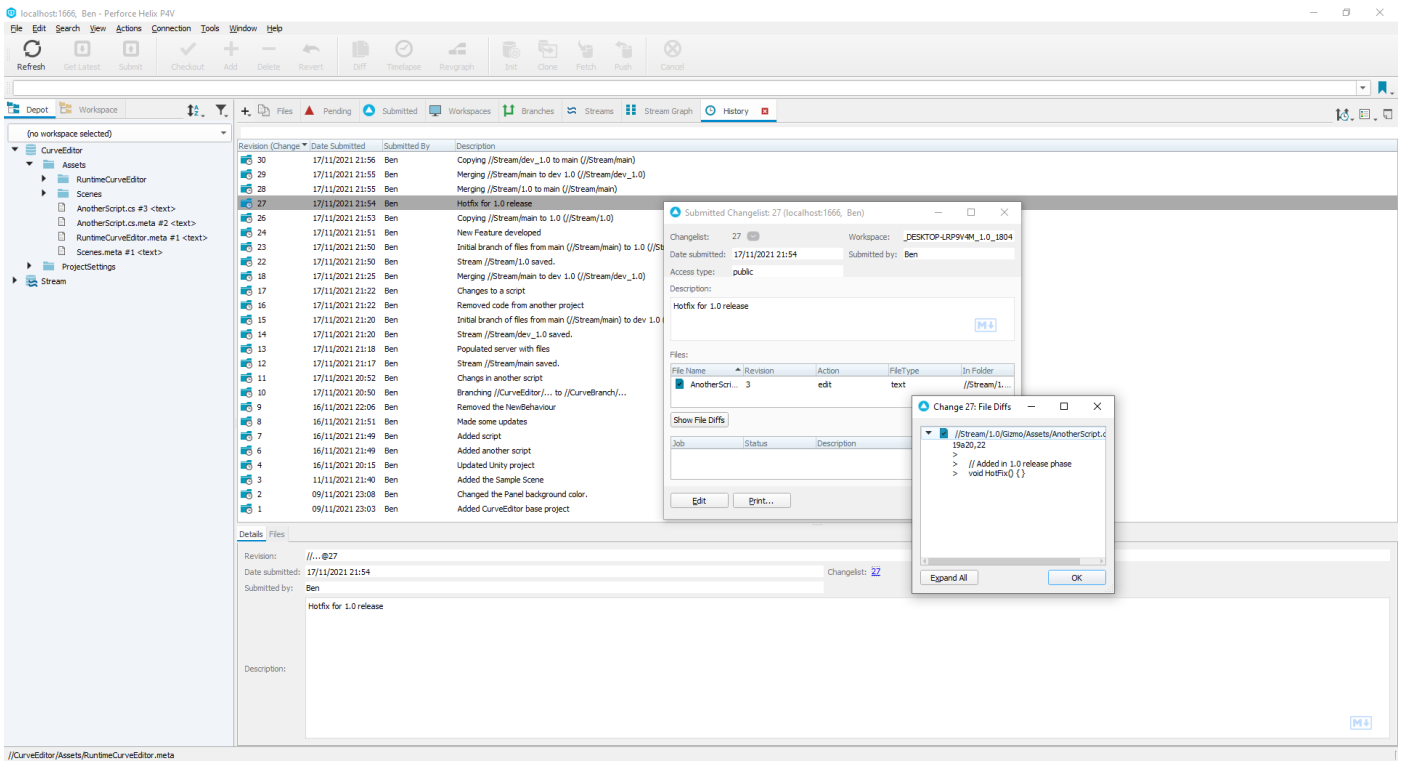
前述のとおり、Git プロジェクトで使用できるホスティングサービスは GitHub 以外にも存在します。より多くの DevOps 機能を備えたアトラシアン社の Bitbucket や GitLab、その他多数のホスティングサービスから選ぶこともできます。

GitHub、GitKraken、Unity を使い始める方法については、Unite Now 2020 のこちらの講演をご覧ください。

Perforce（Helix Core）

Helix Core は大規模なゲームスタジオで使用されることが多い、エンタープライズレベルのバージョン管理システムです。これらのスタジオが Perforce を使用するのは、集中型リポジトリを提供しており、それらは多くの場合、自社のサーバーでホストされるからです。視覚的なリポジトリを持たないため、非技術的な開発者にとっては扱いづらいかもしれませんが、大規模なスタジオにはコードベース管理を支援する DevOps やリリースエンジニアがいます。また、Helix Core はエンタープライズソリューションとして、グローバルなサポートチームも備えています。

Helix Core は小規模なチームでも使用可能で、Amazon AWS や Microsoft Azure などのソリューションを通じてクラウドにデプロイすることもできます。



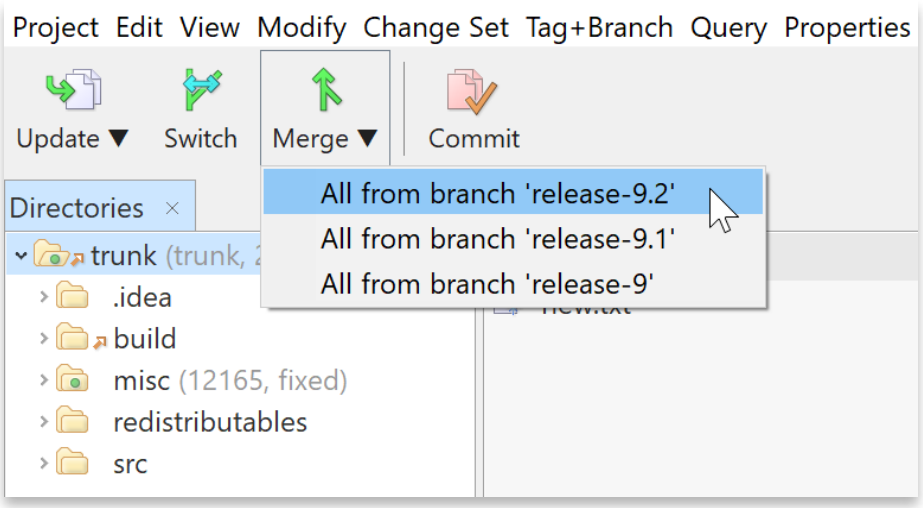
Helix Core P4V インターフェース

Helix Core は大きなファイルを扱うことができ、Unity エディターとの統合も組み込まれています。

Unity ワークフローと Helix Core の統合について、詳しくは Perforce のブログ記事をご覧ください。

Apache Subversion

Git 同様、Apache Subversion（SVN）は無料でオープンソースのバージョン管理システムです。Git とは異なり、こちらは大きなバイナリファイルを扱える集中型の VCS です。しかし、コマンドラインシステムであるため、使いやすくするにはサードパーティ製の GUI クライアントが必要です。その一例が SmartSVN です。



SmartSVN GUI

Git LFS が登場する前、Unity で作業する際には SVN がよく使われていました。集中型ソリューションであるため取り扱いが容易で、前述のとおり大きなファイルを扱うのに適していました。SVN が他のツールに劣るのは、ブランチを使用してそれらをマージする必要が出てきたときです。SVN でのマージには多くの問題があり、特にファイル間の競合（あるいは誤った競合）に関して困難が伴います。他の VCS では数分で終わるマージ作業でも、SVN では手作業で何時間もかかることがあります。

Unity での SVN の設定方法について、詳しくは Unity のドキュメントをご覧ください。

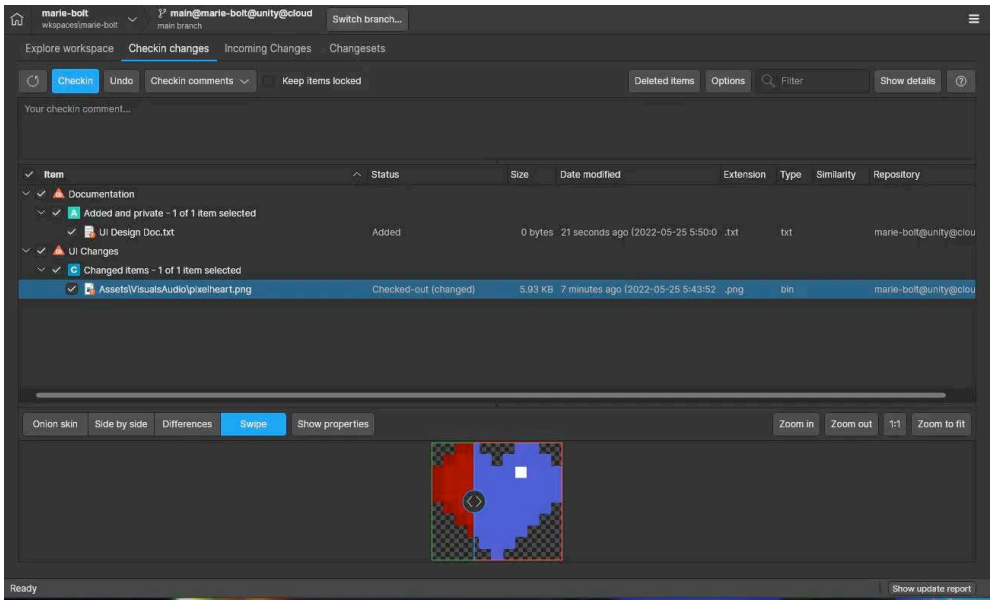
Unity Version Control

Unity Version Control（UVCS）は、プログラマーやアーティストをサポートするユニークなインターフェースを備えた柔軟なバージョン管理システムです。大規模なリポジトリやバイナリファイルの扱いに優れており、ファイルベースと変更セットベースのソリューションとして、プロジェクト全体ではなく、作業中の特定のファイルのみをダウンロードできます。

UVCS にアクセスする方法は 3 つあります。UVCS デスクトップクライアントを使用して複数のアプリケーションやリポジトリにアクセスする、Unity Hub を通じてプロジェクトに追加する、またはウェブブラウザから Unity Cloud 上のリポジトリにアクセスする方法です。詳しい設定方法については、「Unity 6 で UVCS を始める」のセクションをご覧ください。

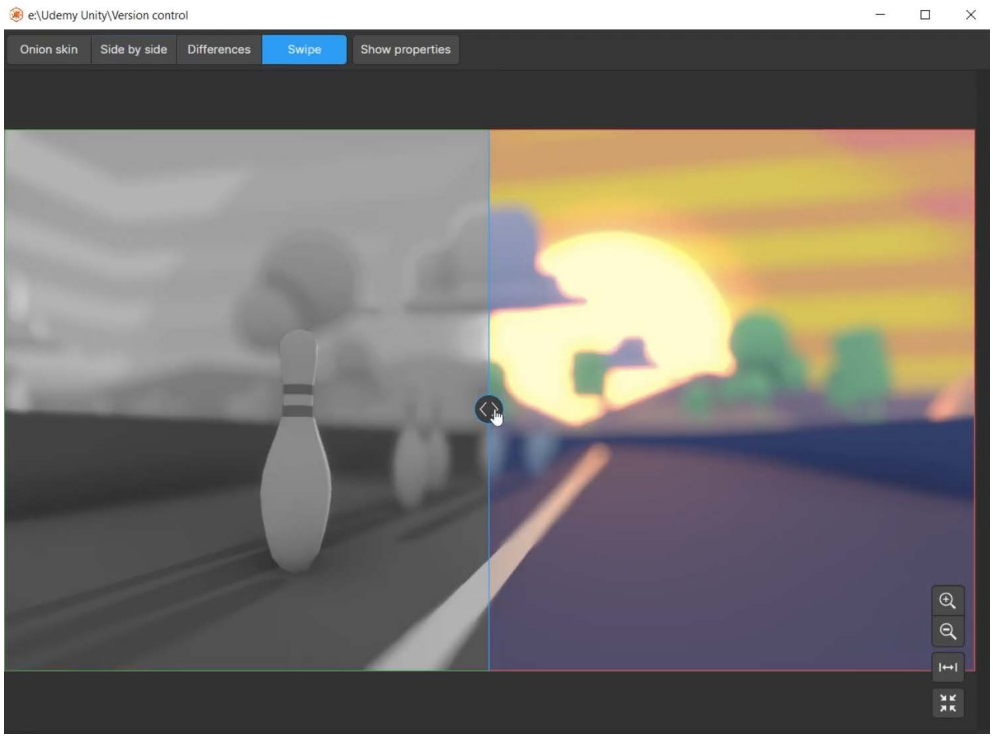
小規模なチーム（最大 3 人まで）は無料で登録でき、本稿執筆時点では最大 5GB のクラウドストレージと、Gluon を含むバージョン管理ソフトウェアへのアクセスが提供されています。

Gluon は、技術的な部分に煩わされずに、アーティストらしく作業できるよう設計されたスリムなクライアントです。作業予定のファイルだけを選択してサーバーからチェックアウトし、他のユーザーが編集できないようロックすることができます。作業が完了したら、再びファイルをチェックインします。Gluon GUI では、プログラマーには適していても、技術的スキルが少ないユーザーには馴染みにくい複雑なコンセプトが排除されています。



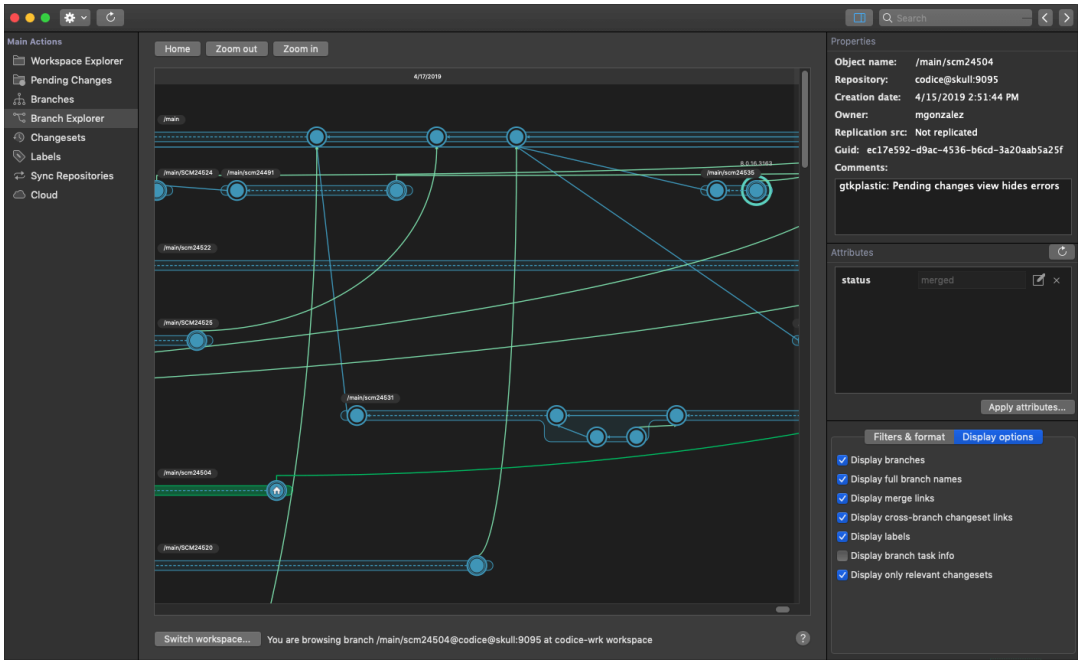
Gluon では、アーティスト向けに設計されたワークフローを提供しており、ファイルや履歴のプレビュー、変更のチェックインを簡単に行うことができます。

UVCS と Gluon の両方には、アーティスト向けに画像の差分を確認する機能が備わっています。画像差分ツールを使えば、同じファイルの 2 つのバージョンを視覚的に比較できます。このような機能は、他のシステムでは提供されていないことがほとんどです。



Swipe モードでの差分ビューアースワイプコントロールをドラッグしてバージョンを切り替えることができます。画像の変更を追跡するのに役立つ機能です。

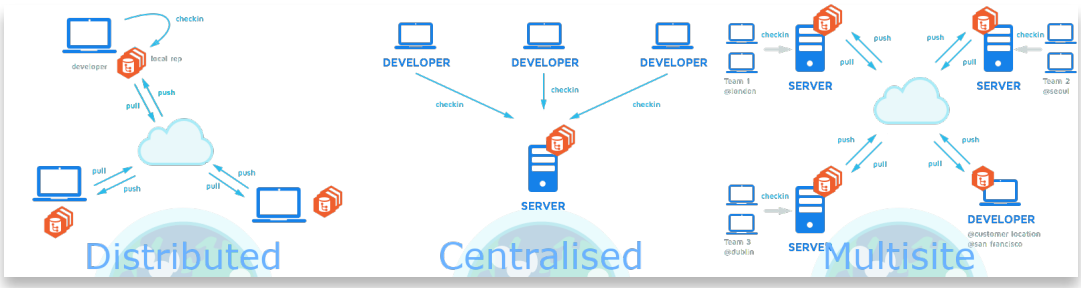
標準の UVCS GUI クライアントには、アーティストが求めるすべての機能に加え、プログラミングチーム向けの機能も備わっています。GUI には、プロジェクト内のすべてのブランチ間の真の関係を視覚的に示すインタラクティブな Branch Explorer が搭載されています。また、他のグループやチームメンバーに自分の作業のレビューをリクエストできる Code Review システムも組み込まれています。



ブランチエクスプローラーは、プロジェクトのマージ構造を視覚的に表示します。これは、左から右へ横方向に展開されます。

UVCS の大きな強みの 1 つは、分散型でも集中型でも、ワークフローに応じて柔軟に構成できる点です。完全な分散型では、開発者はローカルマシン上のリポジトリで作業し、簡単にチェックイン、ブランチ作成、マージを行うことができます。準備が整ったら、変更をサーバーにプッシュまたはプルして共有します。

集中型では、ユーザーが変更を直接サーバーにチェックアウトおよびチェックインするため、全員が最新の状態で作業できます。しかし、開発チームがグローバルな組織へと成長すると、全員が 1 つの中央サーバーを通じて作業する方法が、必ずしも効果的とは言えなくなります。UVCS は、マルチサイトシステムで動作するように構成することも可能です。このシステムでは各サイトにサーバーが設置されているため、チームはローカルサーバーにチェックインすることで、高速なワークフローを保ちつつハードドライブへの負荷を軽減できます。その後、分散サーバーは中央サーバーまたはクラウドサーバーと連携します。



Unity 6 で UVCS を設定する手順については、「Unity 6 で UVCS を始める」セクションと、この Unity Learn のクイックスタートチュートリアルをご参照ください。

VCS の比較

		UVCS	Git	Perforce	Subversion
柔軟性	集中型開発 チェックインのみ、プッシュ / プルは不可				
	分散型開発 プッシュ / プル + ローカルリポジトリ				
バイナリ	大規模リポジトリ				
	大規模ファイル				
	ファイルロックでマージを回避				
GUI	リポジトリの可視化 （ブランチについての専門知識がなくても使いやすい）				
	ユーザーフレンドリーな GUI				
	アーティストフレンドリーな GUI とワークフロー				
ワークフロー	効果的なタスクブランチの作成				
マージ	ブランチ間のマージを検知				
	差分と 3 方向のマージツールを提供				
	マージ理解を助けるツール				
	名前変更、ファイルやディレクトリの移動、リファクタリングに対応したマージ				

クラウド	クラウド上でのリポジトリホスティングが可能				
	大規模リポジトリのクラウドホスティングにも対応				
差分	ファイル間で移動したコードの差分確認が可能				
	メソッドの履歴を表示				
	Enterprise サポート				

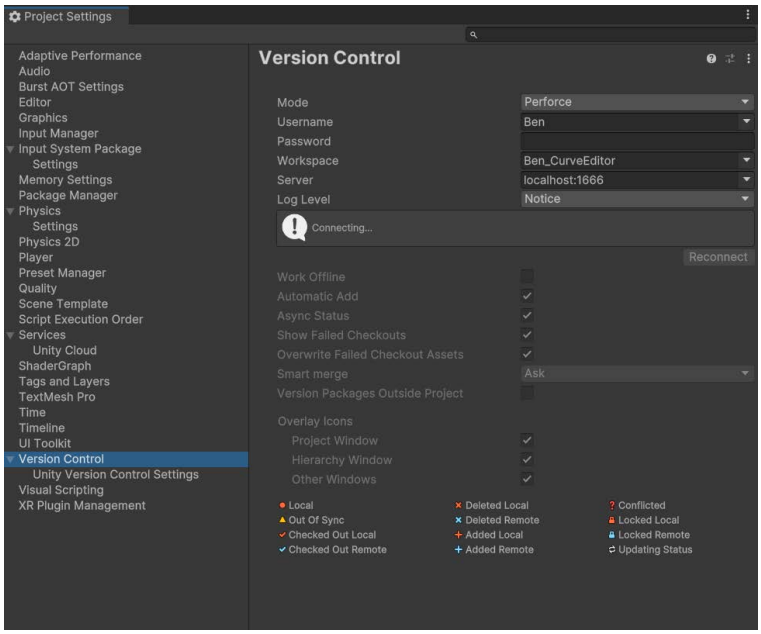
Unity をバージョン管理システムに連携する

このセクションでは、Git、Perforce、または UVCS を Unity と連携させる手順を紹介します。各ソリューションの主要なワークフローを理解することは、チームに最適なシステムを選ぶ判断材料になります。

エディターのプロジェクト設定

Perforce Helix Core

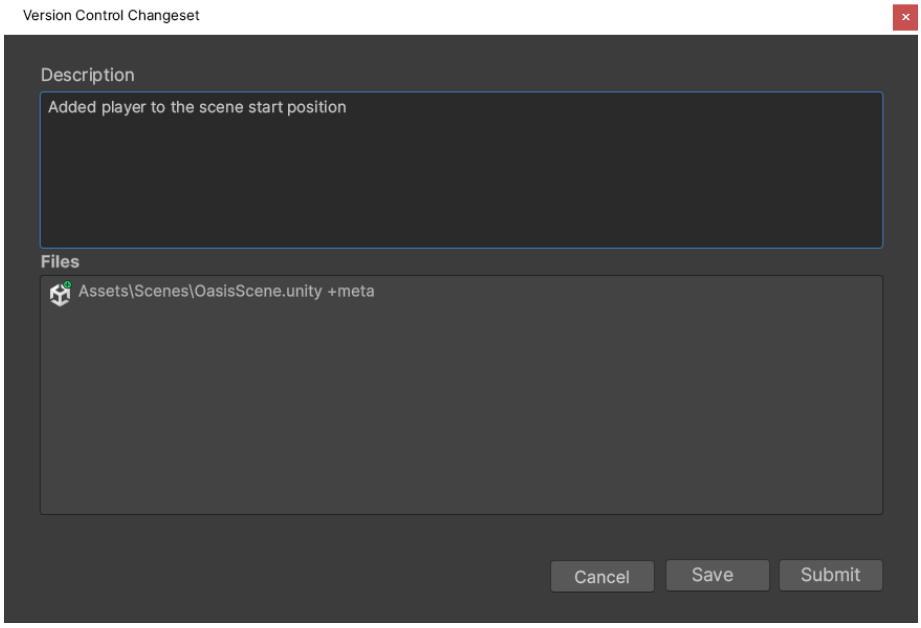
ほとんどのバージョン管理システムは Unity エディターと統合可能で、Perforce Helix Core はエディターにあらかじめ組み込まれています。**「Edit」** > **「Project Settings」** > **「Version Control」** から有効化するだけで利用できるようになります。Mode を Perforce に設定し、ワークスペースとサーバー設定の情報を入力してください。



プロジェクトに Perforce Helix Core を設定する

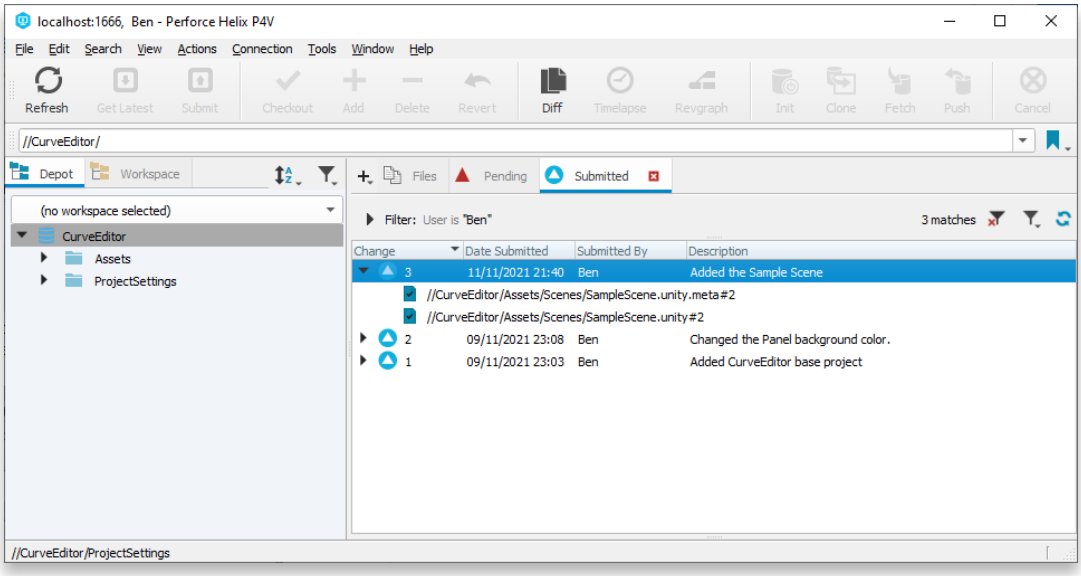
これが有効になると、ファイルは「バージョン管理下にある」と見なされ、ファイルのチェックアウトオプションが表示されます。

ファイルをチェックアウトすると、そのファイルをロック、アンロック、サブミット、またはリバートすることができます。サブミットを選択すると、リポジトリに送信する前にコミットメッセージを追加するための変更セットダイアログが表示されます。



変更セットダイアログボックス

Helix P4V インターフェースから、プロジェクト履歴を確認できます。



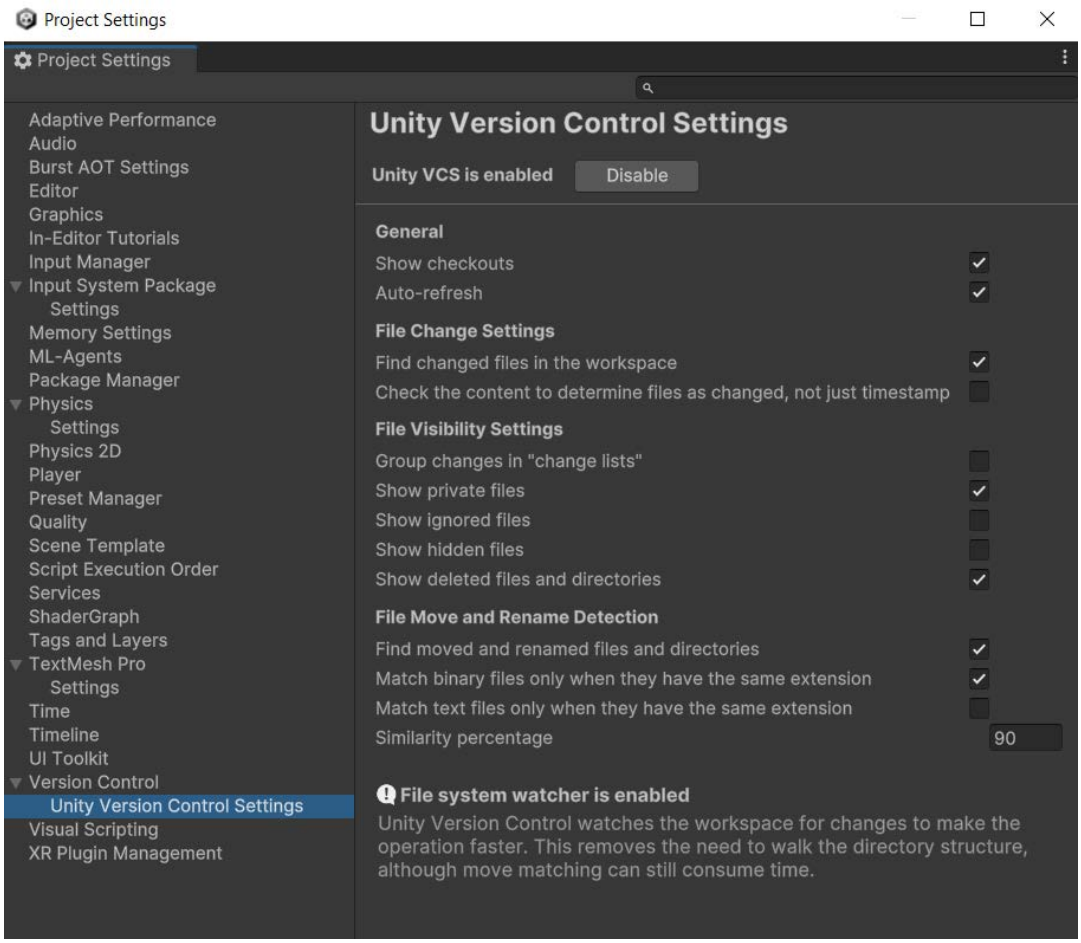
プロジェクト履歴を確認。

Perforce Helix Core と Unity の連携方法について、詳しくは Perforce のブログをご覧ください。

UVCS

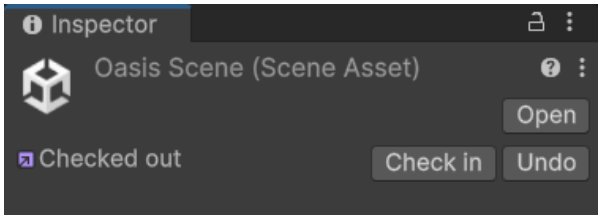
UVCS は Unity エディターに統合されているため、手軽に使い始めて、ワークフローを効率化できます。Hub で新しい Unity プロジェクトを作成する際に「**Use Unity Version Control**」チェックボックスを選択するだけで、すぐに Unity VCS を統合できます。

既存のプロジェクトで UVCS をエディターに接続するには、**「Window」>「Unity Version Control」**を開いてください。Project ウィンドウにタブが表示されます。

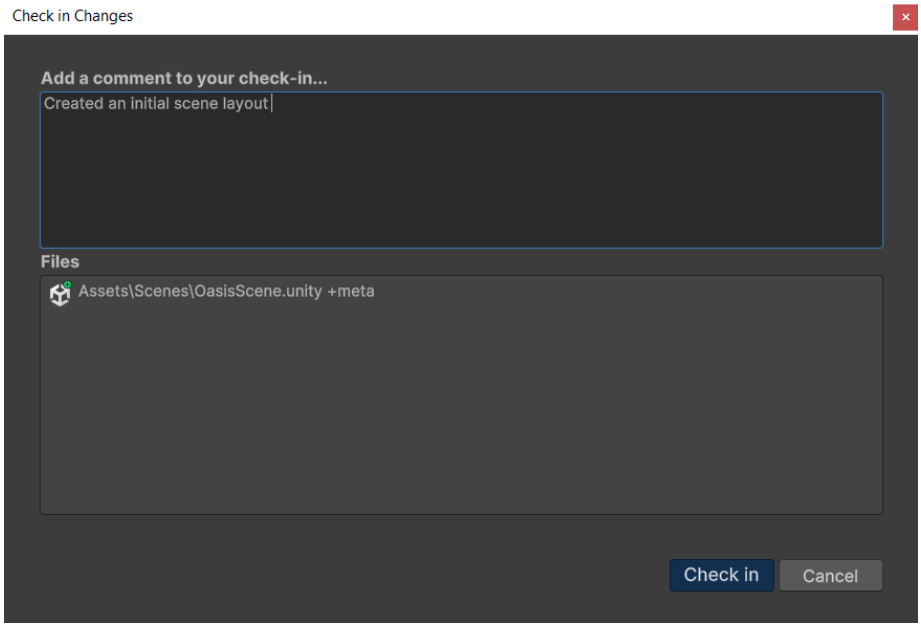


Project Settings で UVCS が表示されている

UVCS は Unity ユーザーにとって馴染みやすく、追加のクライアントが不要になることで、ワークフローをより効率的に進められます。ファイルはエディターから直接追加、チェックアウト、リバート、チェックイン、またはサブミットできます。

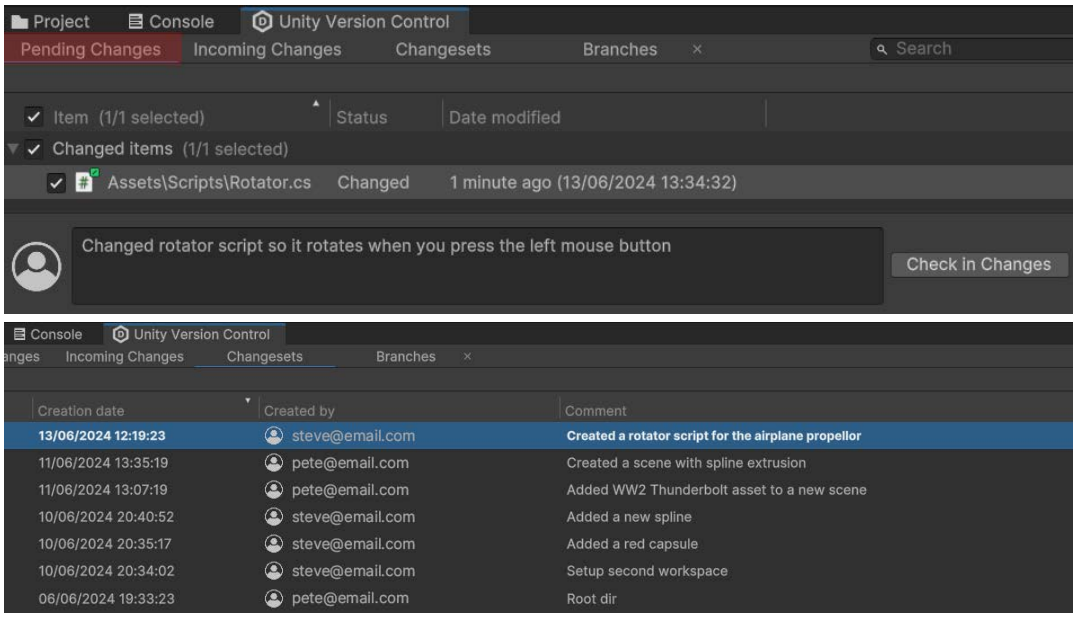


UVCS を有効にした Unity エディターでのファイル操作



ファイルのチェックイン

また、UVCS には、Unity エディター内で**「Window」**>**「Unity Version Control」**から「Changesets」タブにアクセスできるという利点があります。



「Pending changes」と「Changesets」タブ

Unity での Version Control の設定方法について、詳しくはドキュメントをご覧ください。

Git やその他のソリューション

その他の VCS を使用する場合は、「**Edit**」>「**Project Settings**」>「**Version Control**」ウィンドウを開き、ドロップダウンメニューから「Visible Meta Files」を選択してください。ここでは他のオプションはありませんが、バージョン管理システムがメタファイルを検出できるようにするには、メタファイルを表示設定にする必要があります（メタファイルについては前章を参照してください）。

無視するファイル

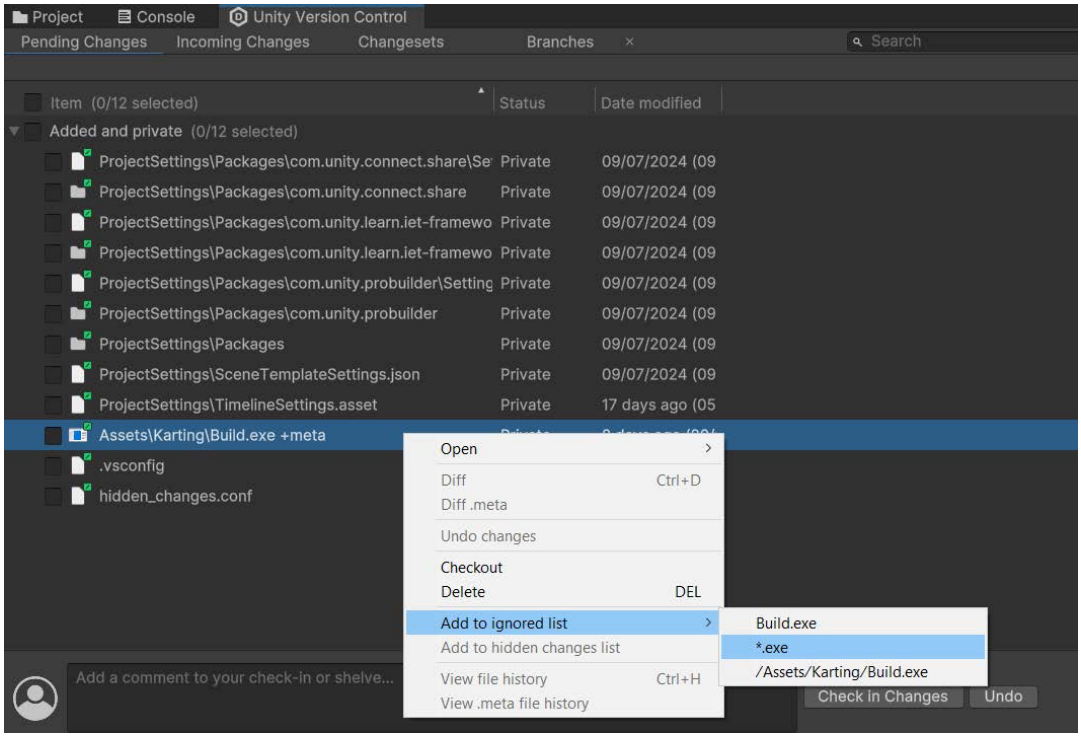
Unity プロジェクトでもその他のプロジェクトでも、バージョン管理下に置くべきなのは、生成できないファイルだけです。

Unity プロジェクトでは、Assets フォルダーと Project Settings フォルダー内のファイルのみをリポジトリにコミットする必要があります。Unity は他のすべてのフォルダーを自動的に再作成できます。Library フォルダーをコミットすることは絶対に避けてください。このフォルダーは非常に大きくなる可能性があり、存在しない場合は Unity がエディター起動時に自動で再生成します。

- Unity エディターから設定すると、UVCS は適切なフォルダーとファイルを自動的にバージョン管理下に置きます。無視するファイルを記述したリストが、プロジェクトのルートにある「ignore.conf」ファイルに保存されます。「ignore.conf」ファイルの設定方法について、詳しくはこのブログ記事をご覧ください。
- Perforce では、明示的に Assets と Project Settings フォルダーをディポに追加する必要があります。
- Git では、管理下に含めないファイルを記述した **.gitignore** ファイルが必要です。使用する Git GUI クライアントによっては、リポジトリ作成時にテンプレートを選択できます。また、ホスティングを先に設定しておけば、GitHub 上で作成することも可能です。または、こちらからテンプレートをダウンロードすることもできます。

.exe や .apk ファイルなどのコミットも避けるべきです。さらに、Unity プロジェクトからビルドされた Gradle や Xcode プロジェクトもリポジトリに追加すべきではありません。

このルールには例外として、Gradle や Xcode プロジェクトの自動ビルドプロセスを設定するケースがありますが、その場合、通常は専用のリポジトリにコミットされることになります。



UVCS を使用している場合、Unity エディターから直接 ignore リストにファイルを追加できます。

大きなファイルの管理

Unity プロジェクトの構成ファイルは、コードだけではありません。実際、Unity プロジェクトでは、スクリプトよりも他のアセットファイルの方が圧倒的に多い場合がほとんどです。これらのアセット（テクスチャ、モデル、プレハブ、オーディオクリップ、タイムライン等）は、バイナリファイルとして保存されます。これにより次の 2 つの問題が発生します：

— リビジョン間での比較が難しい。

— 差分を記述できないため、リポジトリに変更をプッシュする際にファイル全体が再記録される。

分散型環境では、プロジェクトの全履歴がユーザーのローカルマシンに保存されます。長期間にわたり多くの変更が加えられた大きなファイル履歴がある場合、そのファイルの複数のコピーが自身のマシンに保存されることになります。これによって、ハードドライブの容量があっという間に消費されてしまう可能性があります。

この理由から、これまでのチーム作業では集中型ワークフローが好まれる傾向がありました。この方法では、大量のバイナリファイルのバージョン履歴が中央サーバーにのみ保存され、個々のユーザーのマシンには最新バージョンのみが同期されます。

Perforce と UVCS はどちらも大きなファイルを効率的に管理できる集中型システムです。UVCS では分散型パイプラインで作業するオプションも提供されていますが、これらのオプションを選択する際には、ファイルサイズが巨大化するというトレードオフを考慮する必要があります。

UVCS のもう 1 つの機能は、仮想ファイルシステムを利用した Dynamic Workspace です。これは、Dynamic Workspace が必要に応じてファイルをダウンロードする仕組みであることを意味します。つまり、ワークスペース内にはすべてのファイルが表示されますが、実際にはすべてがダウンロードされているわけではありません。

分散型である Git では、大きなファイルの扱いが難しい場合があります。大きなファイルを扱う場合は、必ず Git LFS も含めてください。Git LFS は、.git フォルダー内の大きなファイルをテキストポインタに置き換え、実際のアセットを GitHub などのサーバーに保存します。

バージョン管理のベストプラクティス

使用する VCS に関係なく、多くのベストプラクティスがチーム作業の効率化に役立ちます。チームごとに異なるニーズがあるため、すべてのプラクティスがすべてのチームに合うわけではありません。

これらのヒントは、世界有数のスタジオのプロジェクト最適化を支援している Unity Enterprise Support Team から提供されたものです。

コミットは小さく頻繁に

これはワークフローに最も簡単に取り入れられる変更ですが、意外と苦労する開発者も少なくありません。他のプロジェクト管理ツールで作業する際、既に作業を細かく分けて管理しやすいタスクにしていることが多いでしょう。コミットもこれと同じです。

単一のコミットは、1 つのタスクやチケットにのみ関連付けるべきです。ただし、1 行のコードで複数のバグを魔法のように修正できる場合は例外です。大きな機能に取り組む場合は、それを小さなタスクに分割し、それぞれのタスクごとにコミットを作成してください。機能ブランチについては、後ほど詳しく説明します。

小さいコミット粒度の最大のメリットは、問題が発生した際に変更箇所を簡単に特定でき、他の問題のない変更に影響を与えることなく問題のある変更だけを元に戻すことができる点です。

コミットメッセージは簡潔に

コミットメッセージからは、プロジェクトの変遷がわかります。ゲームにハイスコアテーブルを追加した履歴を見つけようと思った場合、コミットメッセージが「メニューを更新しました!」ではなく「メニューにハイスコアテーブルを追加しました」と書かれている方がずっと簡単です。

JIRA や GitLab のようなタスクチケット管理システムを使用している場合は、コミットにチケット番号を含めるとさらに便利です。多くのシステムはスマートコミットに対応しており、設定次第でコミットメッセージからチケットを参照したり、そのステータスを変更したりできます。

例えば、「JRA-123 #close #comment タスク完了」というコミットは、JIRA チケット JRA-123 をクローズし、チケットに「タスク完了」というコメントを残します。

このワークフローの設定方法について詳しくは、JIRA のドキュメントや GitLab の Pivotal Tracker サービスを参照してください。

無計画なコミットを避ける

「commit -a」（すべての変更をコミットする git コマンド）やその類似コマンドを使用すべき場面は、プロジェクトの初回コミット時だけです。通常、これはプロジェクト内のファイルが README.md だけの時です。

コミットには、リポジトリに反映させる変更に関連するファイルだけを含めるべきです。Unity プロジェクトで作業する際は特に注意が必要で、シーン、プレハブ、スプライトアトラスなど、意図していないファイルが変更済みとしてマークされる場合があります。

他の人が作業中のシーンに誤って変更をコミットしてしまうと、その人が自分の変更をコミットしようとした際に、まずあなたの変更をマージする必要があり、厄介な問題を引き起こす可能性があります。

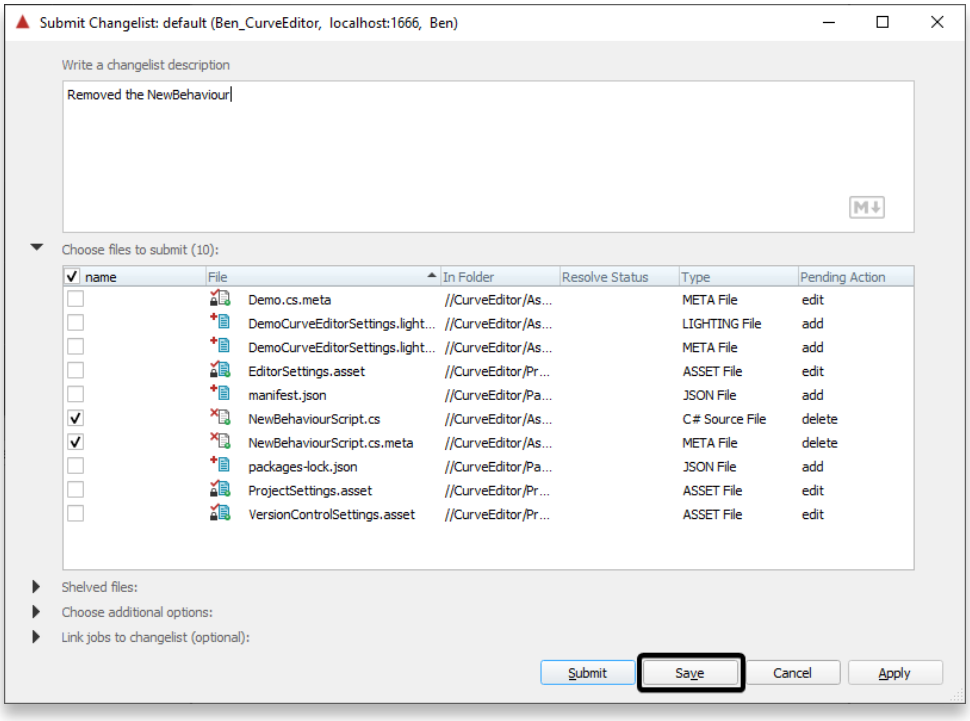
これはバージョン管理に不慣れな人がよく犯す代表的なミスの 1 つです。プロジェクトでは、*自分*が加えた変更だけをコミットするべきだと理解することが重要です。ワークフローを高速化する方法について詳しくは、このブログ記事を参照してください。

最新の変更を取得

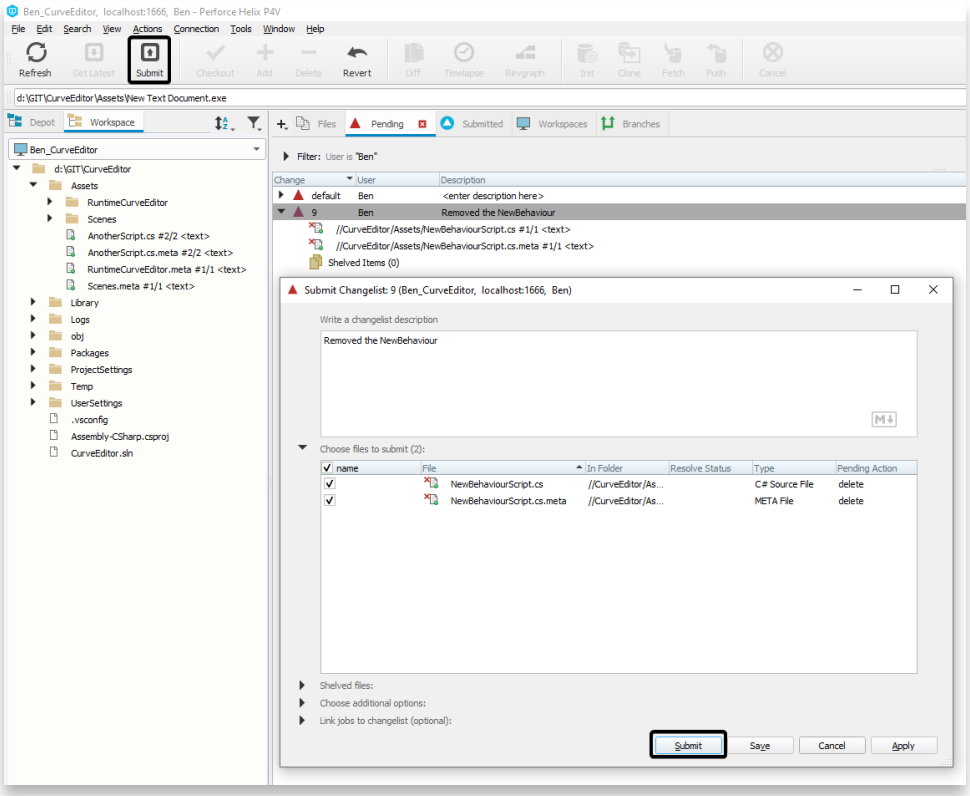
必要に応じて頻繁に、リポジトリから最新の変更をワーキングコピーにプルしましょう。孤立して作業するのは良くありません。マージコンフリクトのリスクを高めることになるからです。各システムを使用した場合の典型的な日々のワークフローは、以下のようなものです。

Git	Perforce

— git pull
— その後は、自由に：
  — ワーキングコピーで編集する。
  — 変更を git commit する。
  — 最新の変更を git pull する。
— コミットの変更セットが適切な形になったら：
  — もう一度 git pull する。
  — git push してコミットをリポジトリに送信する。
	— 最新の変更を取得
	— 作業ファイルをチェックアウト
	— 編集
	— 変更を送信



P4V で新しい変更セットに変更を保存



P4V で変更セットを送信

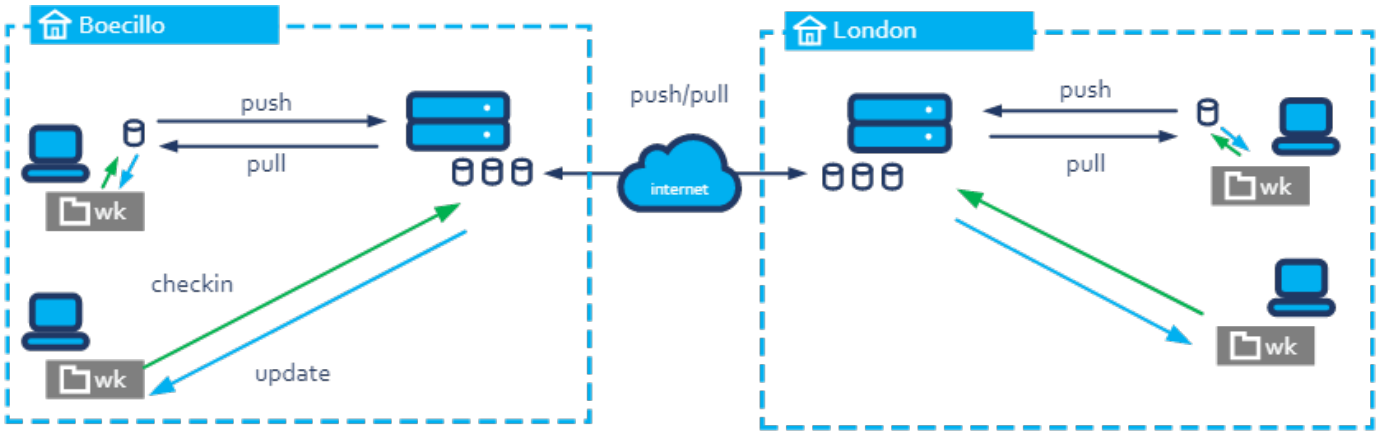
UVCS のワークフローは少し異なり、集中型、分散型、マルチサイト構成のいずれでも作業できます。

UVCS（集中型）	UVCS（分散型）	UVCS（マルチサイト）
— リポジトリを同期 — 表示ブランチをプル — 作業ファイルをチェックアウト — 編集 — 変更をチェックイン — リポジトリを同期 — 表示ブランチをプッシュ	— サーバーから変更をプル — 変更をローカルコピーにチェックイン — 新しい変更をプル — 変更をサーバーにプッシュ	— セットアップに応じて、これら 2 つのハイブリッド

マルチサイト構成はカスタムニーズに合わせて調整でき、各ユーザーは集中型または分散型のワークフローで作業できます。

次の 2 つのチームの例を考えてみましょう。

- 各チームにはオンサイトサーバーがあります。
- 両サイトのチームメンバーはローカルまたは分散型の手法でチェックインを行い、近くのオンサイトサーバーによる高速処理の利点を活用します。
- サーバー間でプッシュやプルを行い、完全または部分的に同期を維持します。



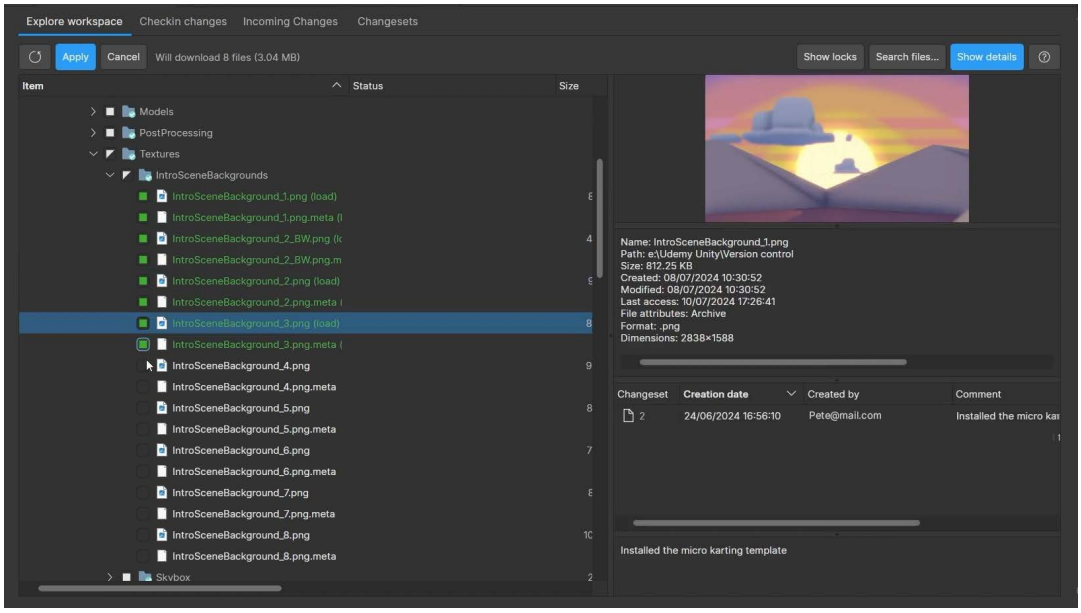
マルチサイト UVCS 構成

ツールセットを理解する

どの VCS を選ぶにせよ、チームがそのツールを使いこなし、ビジュアルクライアントを含む利用可能なツールを十分に把握していることを確認してください。

Git を使用する場合、全員が同じ GUI クライアントを使用する必要はありません。ただし、全員が commit > pull > push のワークフローに慣れ、必要なファイルだけをコミットする方法を理解している旨を確認してください。

UVCS を使用する場合は、アーティストがワークフローを簡素化できるよう、Gluon に慣れてもらいましょう。Gluon を使えば、作業したいファイルだけを選んでダウンロードできるため、プロジェクト全体をダウンロードして管理する必要がなくなります。ファイルをロックして他の人が作業できないようにし、完了したらファイルをリポジトリに送信して再度アンロックできます。

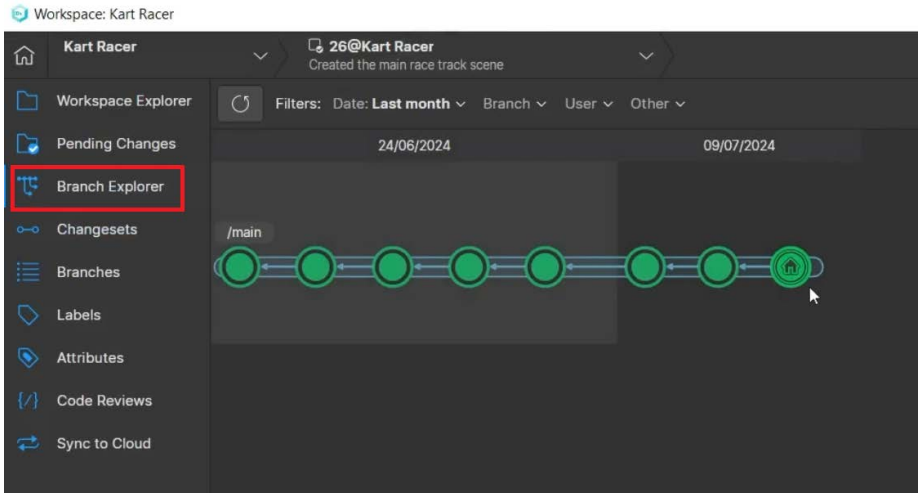


UVCS の Gluon

Perforce Helix Core を使用する場合は、Unity エディター内蔵のツールを使用して、エディターから直接バージョン管理を行います。これは、アーティストの作業だけでなく、シーンやプレハブなどの Unity アセットファイルを扱う一般的な作業にも非常に便利です。エディターでアセットをチェックアウトして修正を加え、そのまま Unity を離れることなくチェックインできます。

機能ブランチと Git Flow

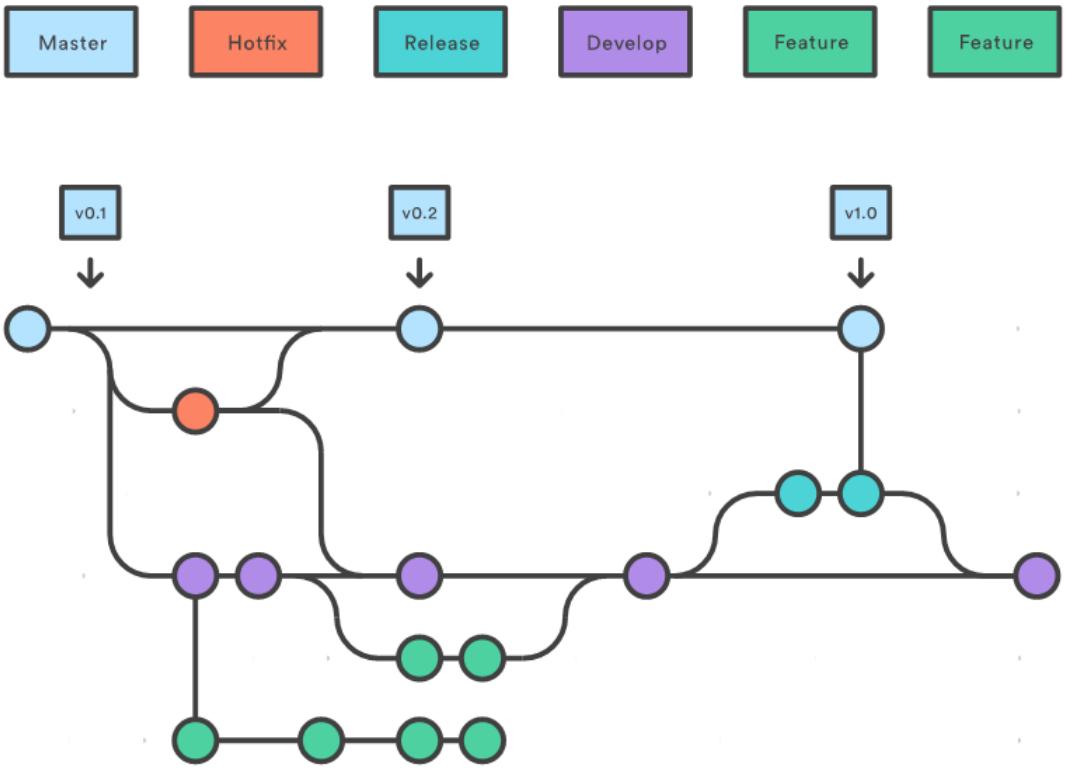
複数のリリースサイクルを持つ長期的なプロジェクトでは、作業している場合、機能ブランチを活用することでワークフローが大幅に改善されます。多くの場合、チームは trunk、master、または main と呼ばれるリポジトリの同じブランチで作業します。この場合、プロジェクト全体が同じタイムラインで進行します。



UVCS でのメインブランチに沿った開発

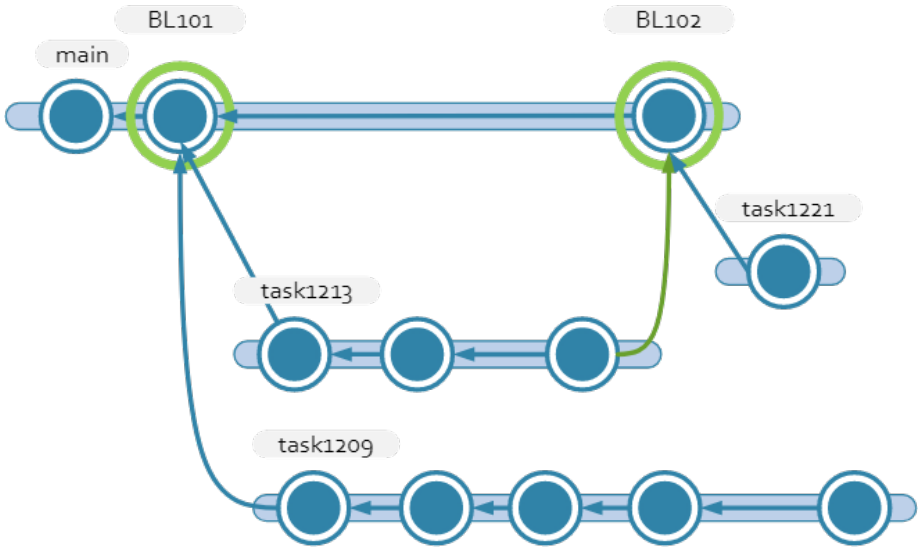
しかし、作業をいくつかのブランチに分割することで、チームとしての効率が高まる場合があります。

Git には、機能開発、バグ修正、リリースごとに異なるブランチを活用することを目的とした、Git Flow という特定のワークフローがあります。開発者は新しい機能を隔離されたブランチで作業し、作業が完了した後にメインブランチへマージします。一方で、他のメンバーが以前のリリースに対してホットフィックスを適用し、バグを修正して新しいバージョンを安全にリリースすることも可能です。この際、まだ開発中の機能は含まれません。



Git Flow ワークフローによって、リリース管理が容易になる。

UVCS にはタスクブランチも備わっています。このパターンでは、追跡するタスクごとに新しいブランチを作成します。Git Flow では、完全で時には大規模な機能を開発するために機能ブランチを使用しますが、UVCS のタスクブランチは短期間での利用が想定されています。もしタスクの実装に複数のコミットが必要な場合は、そのタスクをより小さなタスクに分割できる可能性が高いです。

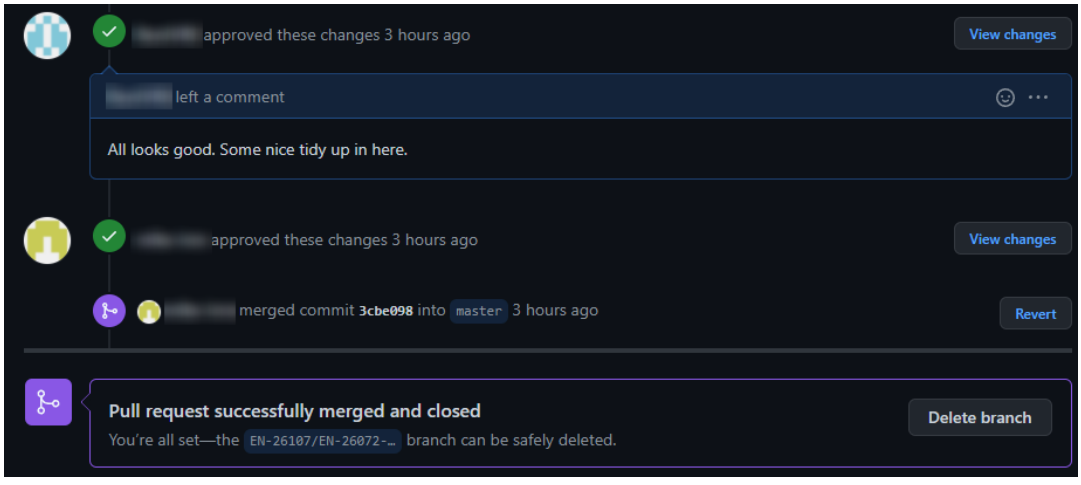


UVCS のタスク別ブランチパターン

Perforce Helix Core は、このスタイルのワークフローをサポートするために、Streams と呼ばれるシステムを使用します。作業用のディポを作成する際は、それをストリームディポタイプとして設定する必要があります。設定後、Stream Graph ビューを使用して新しいストリームを作成できます。メインラインストリーム以外のすべてのストリームには、親ストリームが必要です。これにより、変更内容を上流のストリームにコピーできます。

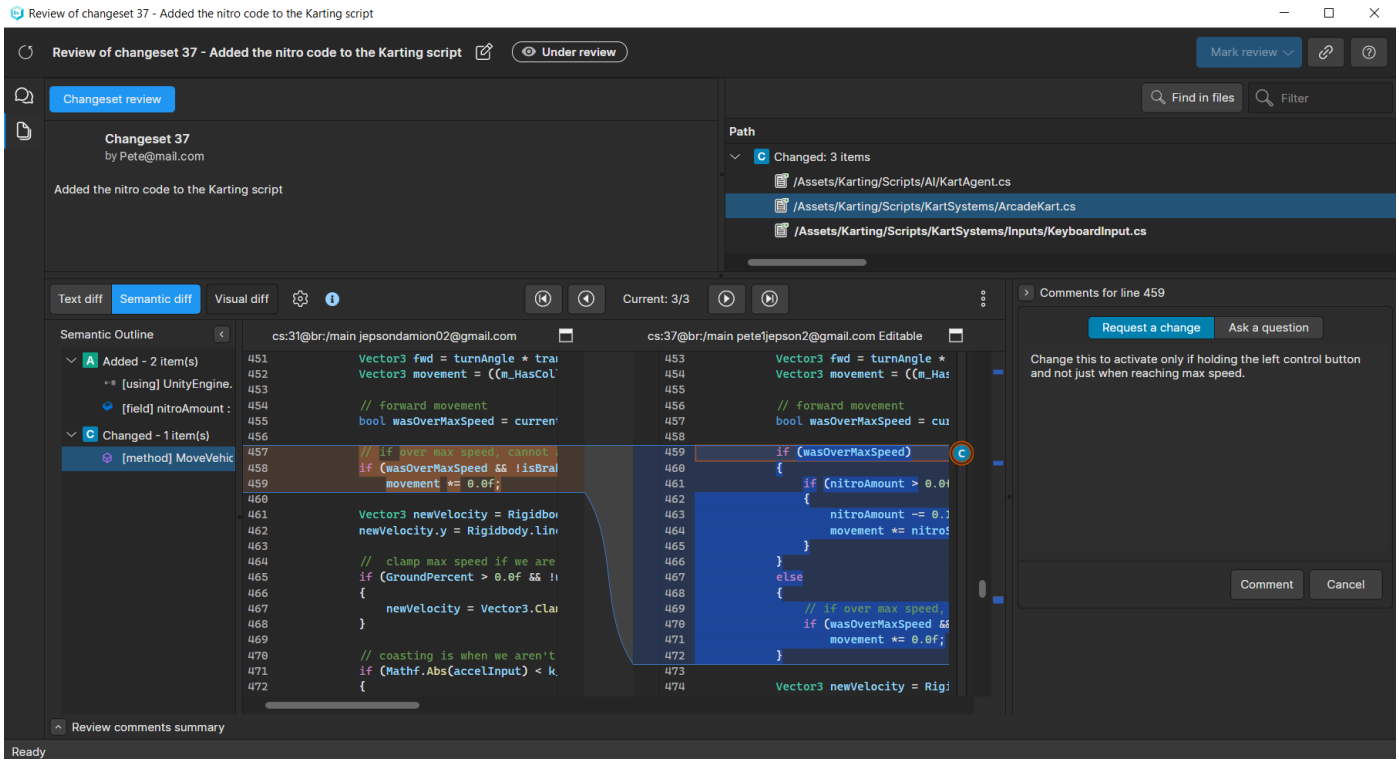
プルリクエスト

機能ブランチでの作業を完了したら、プルリクエストを使用してリポジトリのメインストリームに変更を戻すのが望ましいプラクティスです。プルリクエストは機能やタスクを担当した開発者が作成し、通常は上級開発者や DevOps がメインラインに統合する前に変更をレビューする役割を担います。



GitHub 上のクローズされたプルリクエスト

UVCS と Perforce には、ブランチをメインラインにマージする自動化ツールが備わっています。UVCS は Mergebot を利用して、レビューと検証を通過したリポジトリのブランチを自動的にマージします。Perforce には Helix Swarm という追加プラットフォームがあり、コードレビューを管理するだけでなく、レビューに自動テストを組み込むこともできます。



UVCS のコードレビュー機能は GUI に統合されています。

Unity 6 で UVCS を始める

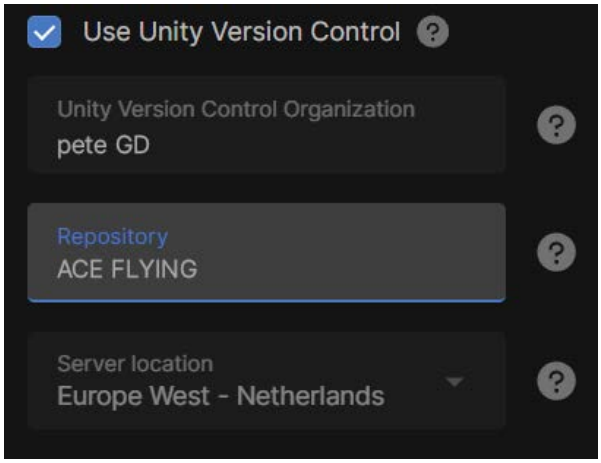
Unity DevOps 製品 は UVCS と Build Automation で構成されています。UVCS は、最大 3 名のチームメンバー（シート）と月間 5GB のデータまで無料で利用できます。それ以降の料金は、月間アクティブユーザー数とクラウドストレージの総使用量に基づいて計算されます。UVCS では、UVCS サービスの使用許容量が 50%、75%、90% に達した際に、UVCS 組織のオーナーに警告メールを送信します。これにより、使用状況を簡単に把握できるようになります。

詳細については、Unity Cloud の価格プランページを参照するか、Unity Cloud ダッシュボードにサインインして、DevOps ダッシュボードの「About」ページをご確認ください。

UVCS にアクセスする方法は 3 つあります。UVCS デスクトップクライアントを使用して複数のアプリケーションやリポジトリにアクセスする方法、Unity Hub を通じてプロジェクトに追加する方法、またはウェブブラウザから Unity Cloud 上のリポジトリにアクセスする方法です。

Unity プロジェクトで UVCS を使用する

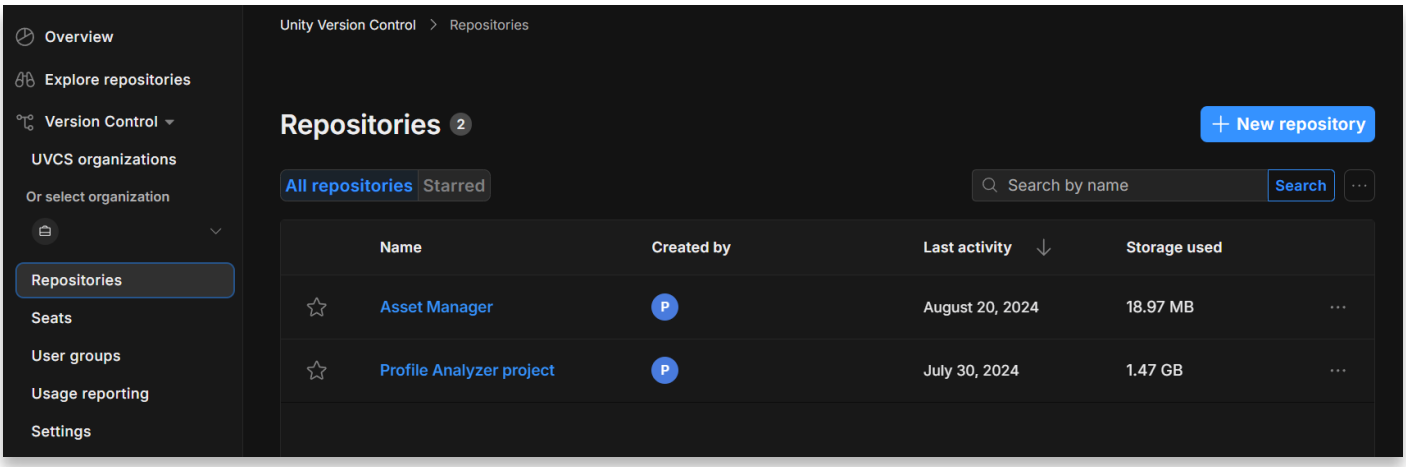
Unity Hub でプロジェクトを作成し、「**Version Control**」のチェックボックスを選択すると、Unity は自動的にリポジトリ（リポ）を設定します。



Unity Hub でバージョン管理を設定する

これは、プロジェクトのデータとファイルがローカルのマシンとクラウドの両方に保存され、コラボレーションを支援するとともに、安全なバックアップとして機能することを意味します。プロジェクトの名前はリポジトリの名前となり、高速な接続を確保するために最も近いサーバーを選択できます。

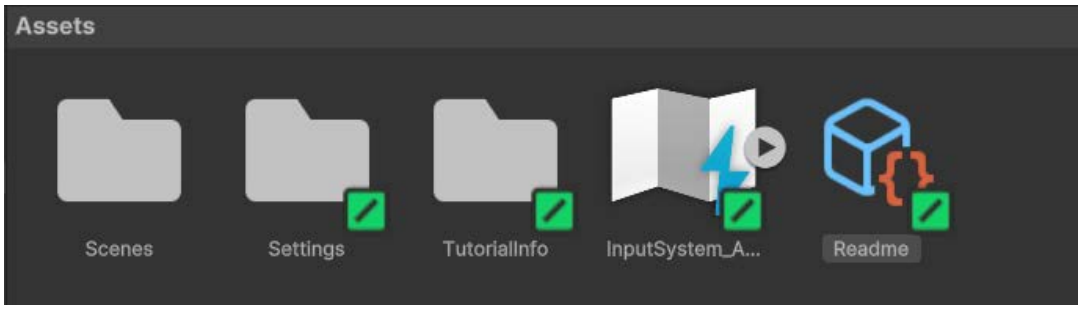
初回使用時に、Assets、Packages、Project Settings フォルダーがサーバーにバックアップされます。Unity Cloud にログインし、リポジトリ内で「**File Explorer**」を選択すると、これらのファイルにアクセスできます。これにより、クラウドリポジトリ内のすべてのファイルが表示されます。



Unity Cloud ダッシュボードの File Explorer では、リポジトリ内のファイルおよびフォルダー構造が表示されます。

UVCS は、Library、Logs、User Settings フォルダーを無視します。これらのフォルダーはコンピュータ上で自動生成され、大量のスペースを消費するためです。

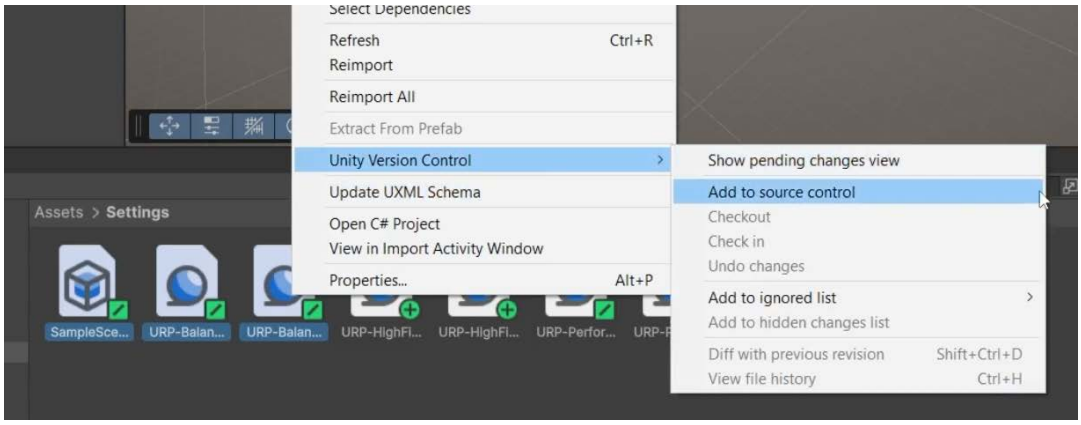
UVCS によって無視される他のフォルダーには、Project ウィンドウのフォルダーまたはファイルの下に、スラッシュアイコン付きの緑色のボックスが表示されます。



クラウドにバックアップされない無視されたファイルは、スラッシュアイコン付きの緑色のボックスで示されます。

プロジェクトが URP または HDRP を使用している場合は、Version Control ウィンドウの「**Pending Changes**」セクションに Settings フォルダーを必ず含めてください。Settings フォルダーにスラッシュ付きの緑色のボックスが表示されている場合、それを選択して右クリックします。**「Version Control」>「Add to source control」**を選択します。

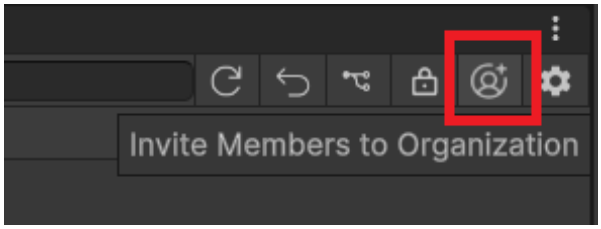
スラッシュは緑色の円にプラスマークが表示されたアイコンに変わり、これらのファイルが保留中の変更リストに追加され、クラウドへのアップロード準備が整います。



設定ファイルを保留中の変更ビューに含める

他のチームメンバーを招待する

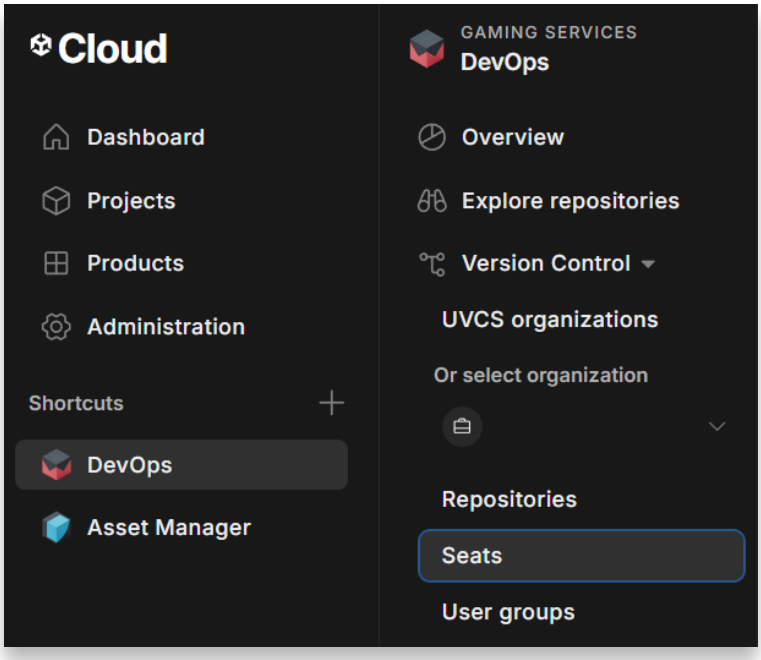
Version Control ウィンドウの右側には、チームメンバーを招待するためのボタンがあります。



Version Control ウィンドウの「**Invite Members to Organization（組織にメンバーを招待）**」オプション

これをクリックすると、Unity Cloud のログインページに移動します。Unity ID とパスワードを使用してログインしてください。

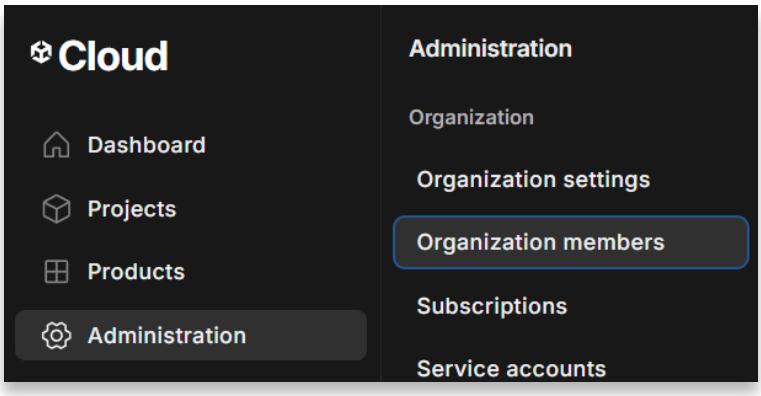
左側のショートカットから DevOps が開き、シートセクションも表示され、現在のプロジェクトに追加のシートを割り当てることができます。執筆時点では、無料プランでは最大 3 つのシートまでしか追加できません。追加のシートを購入するには、Cloud Pro へのアップグレードが必要です。詳細については、Unity Cloud の価格プランページ を参照するか、Unity Cloud ダッシュボードにサインインして、DevOps ダッシュボードの「About」ページをご確認ください。



追加のシートは Unity Cloud の「DevOps」セクションから割り当て可能。

新しいメンバーにプロジェクト内のシートを割り当てましょう。その際、組織メンバーとしても招待する必要があります。これにより、クラウドダッシュボード上の使用統計を含む、すべてのデータにフルアクセスできるようになります。

「Administration」>「Organization members」に移動し、「**invite organization members**」ボタンをクリックして組織にメンバーを追加します。



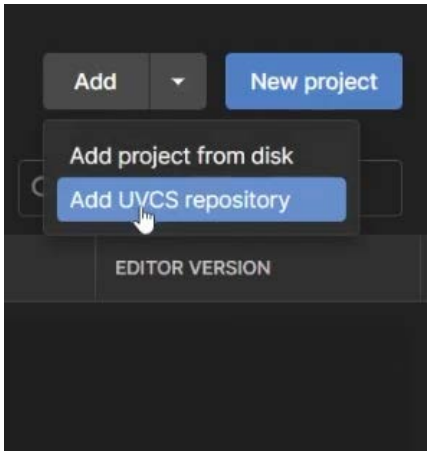
Administration オプションから組織メンバーを追加。

各メンバーに適切なロールを割り当てましょう。ゲスト、ユーザー、管理者から選択します。ユーザーおよび管理者メンバーは、すべてのプロジェクトを閲覧できます。プロジェクトメンバーセクションでロールを割り当て、プロジェクトデータへの適切なアクセス権限を付与できます。

これにより、メンバーはプロジェクトにアクセスし、割り当てられたロールに応じて貢献できるようになります。管理者権限を持つユーザーは、クラウド上で利用可能なすべてのオプションにフルアクセスできます。

招待した他のメンバーにはメールが送信されます。

招待されたメンバーは Unity Hub を開き、「Add」ボタンの横にあるドロップダウンをクリックして、「**Add UVCS repository**」を選択します。



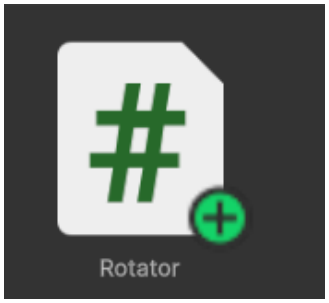
「**Add UVCS repository**」をクリックして、Unity Cloud でプロジェクトを開く。

注：すべてのチームメンバーが同じ Unity バージョンを使用していることが重要です。使用バージョンが統一されていないと、バージョン間でプロジェクトを変換する際に問題が発生する可能性があります。

これで、クラウドリポジトリのプロジェクトがユーザーのコンピューターにダウンロードされます。

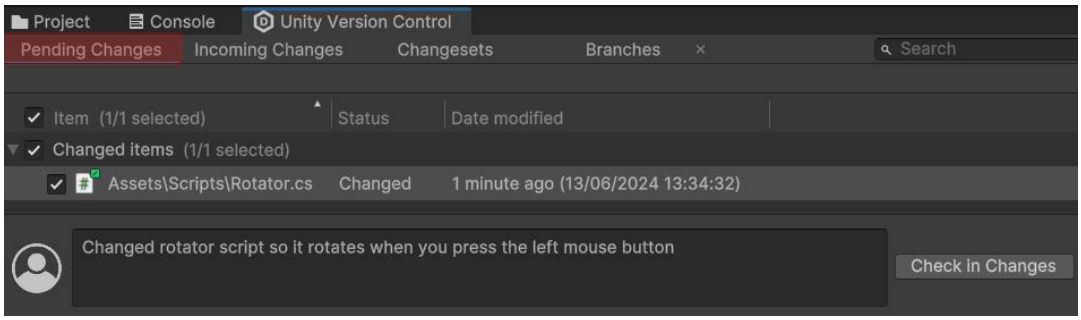
変更をチェックイン

プロジェクトに変更を加えると、変更したファイルに緑の「**+**」アイコンが表示されます。これは、どのファイルをサーバーにチェックインして更新すべきかを示しています。



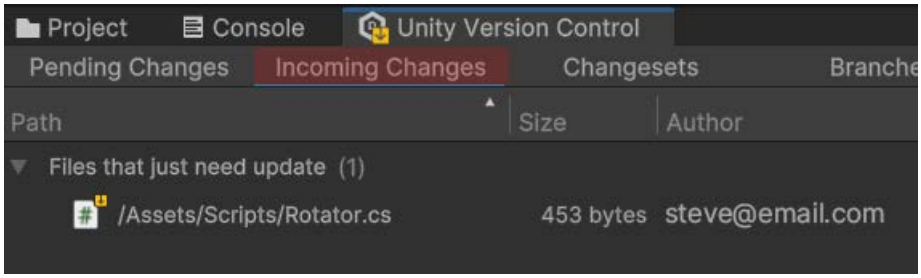
プラスマークが書かれた緑の円アイコンは、ファイルに変更があり、チェックインが必要であることを示す。

プロジェクトは頻繁に更新し、簡単にロールバックできる状態にしておきましょう。更新されたファイルは、Version Control ウィンドウの「**Pending Changes**」セクションに表示されます。行った変更について短く簡潔な説明を追加し、「**Check in Changes**」ボタンをクリックしてください。



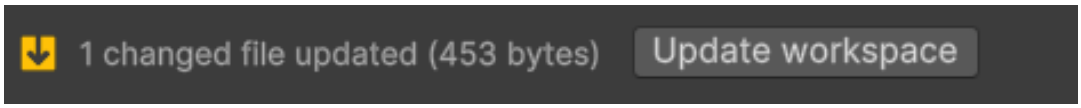
「Pending Changes」セクションでは、プロジェクトに加えた変更をチェックインできます。

同じブランチで作業している他のユーザーには、Version Control ウィンドウの「**Incoming Changes**」タブに黄色いダウンロードアイコンが表示されます。これは、他のユーザーによる変更を表示します。

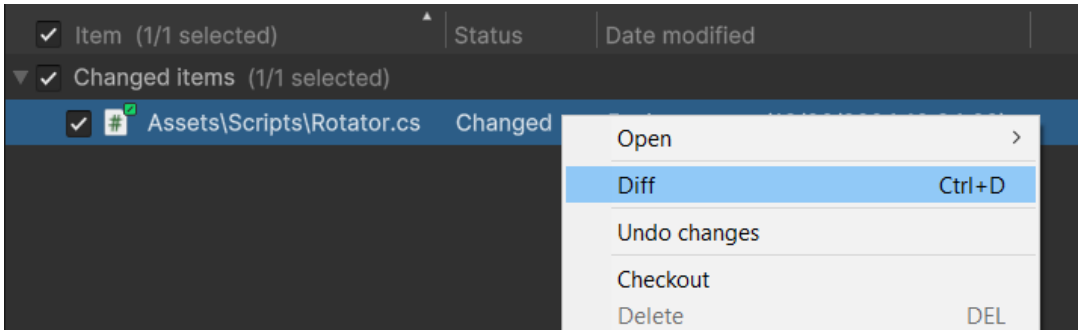


他のユーザーは、あなたがチェックインした変更を確認し、自分のワークスペースを更新して変更を統合できる。

「**Update** workspace」ボタンをクリックすることで、ワークスペースを更新し、変更を取り込むことができます。

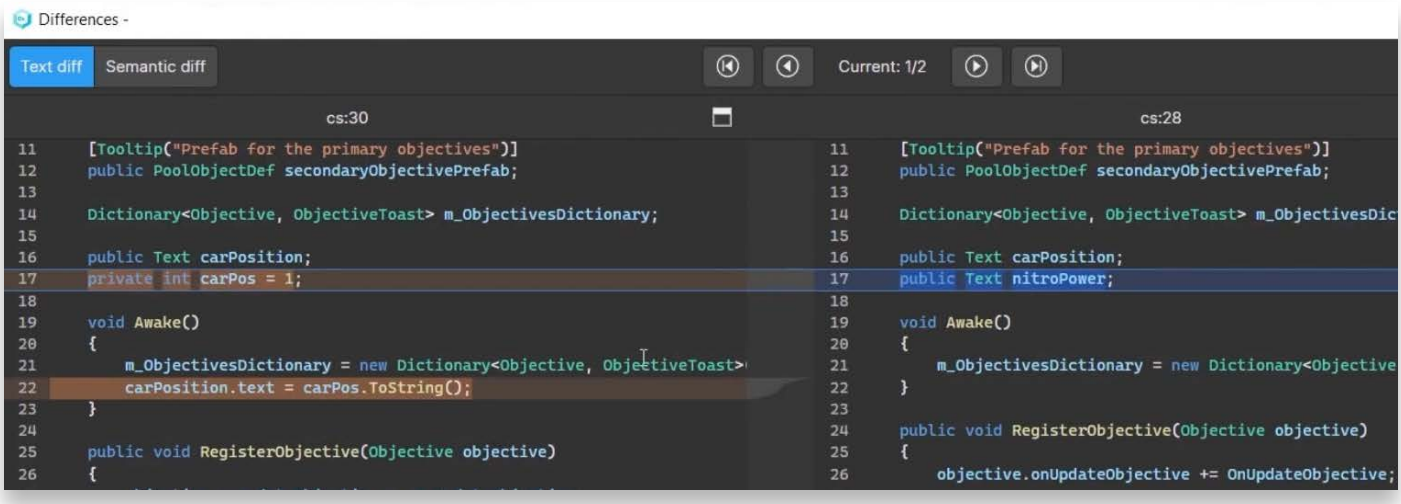


また、既存のファイルが編集された場合は、変更されたファイルを右クリックして、「**Diff**」を選択することで、変更前と後の差分を確認できます。



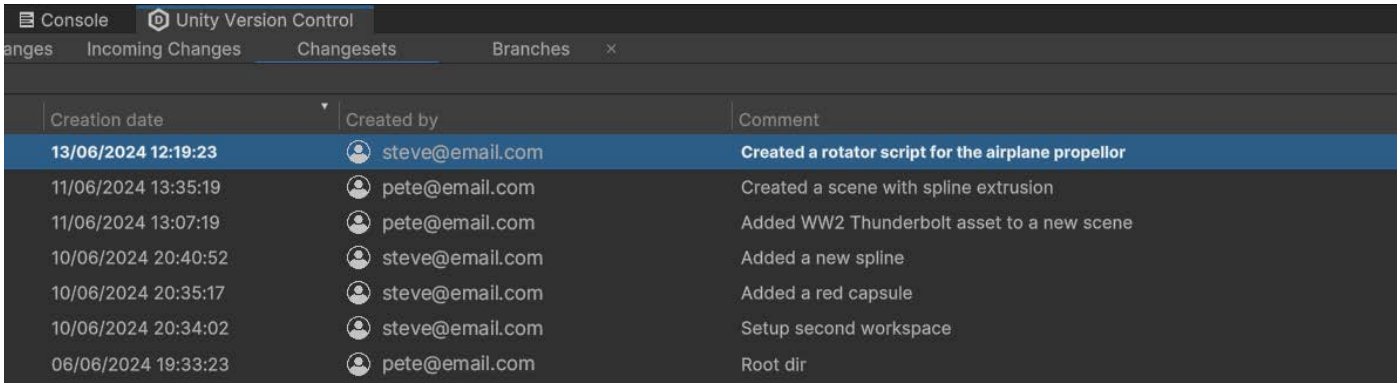
変更されたファイルと元のファイルの差分を確認。

Diff ビューアーでは、左側に元のバージョン、右側に変更後のバージョンが表示されます。この機能は特に C# スクリプトで便利です。Unity のコードを意識したマージ技術「Semantic Merge」は、コードの移動を追跡し、関連する変更にのみ集中できるようにします。



Diff ビューアーでは、上部にオレンジ色で元のファイルが、下部に青色で変更後のファイルが表示されます。

すべての変更セットは「**Changesets**」セクションでリスト形式で表示され、誰がどの日時に変更したかを確認できます。変更リストには、変更内容の説明や更新されたファイル情報が含まれています。

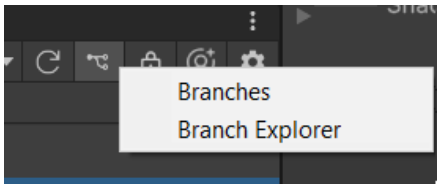


プロジェクトのすべての変更セットが表示されており、任意の変更セットにロールバック可能

UVCS デスクトップクライアント

Unity エディターの Version Control ウィンドウは、クラウドへのデータのプッシュやクラウドからのプルに必要な標準機能を提供します。デスクトップクライアントでは、さらに多くの機能が利用可能です。

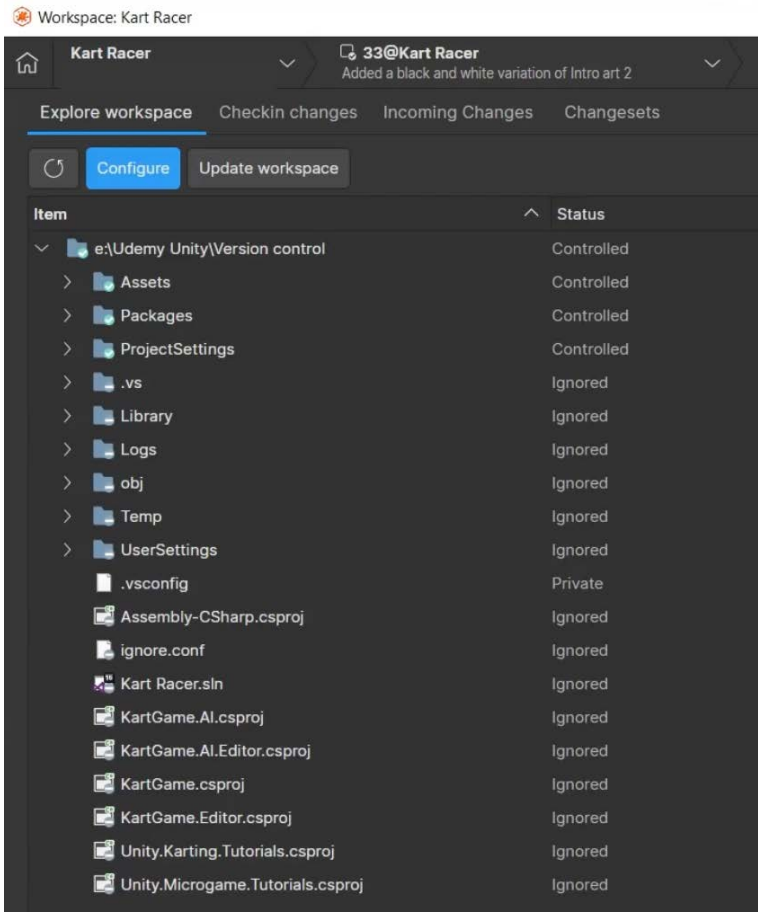
デスクトップクライアントを起動するには、拡張モードで「**Branch Explorer**」をクリックしてください。まだクライアントのインストールが済んでいない場合、インストールを完了するよう促すプロンプトが表示されます。



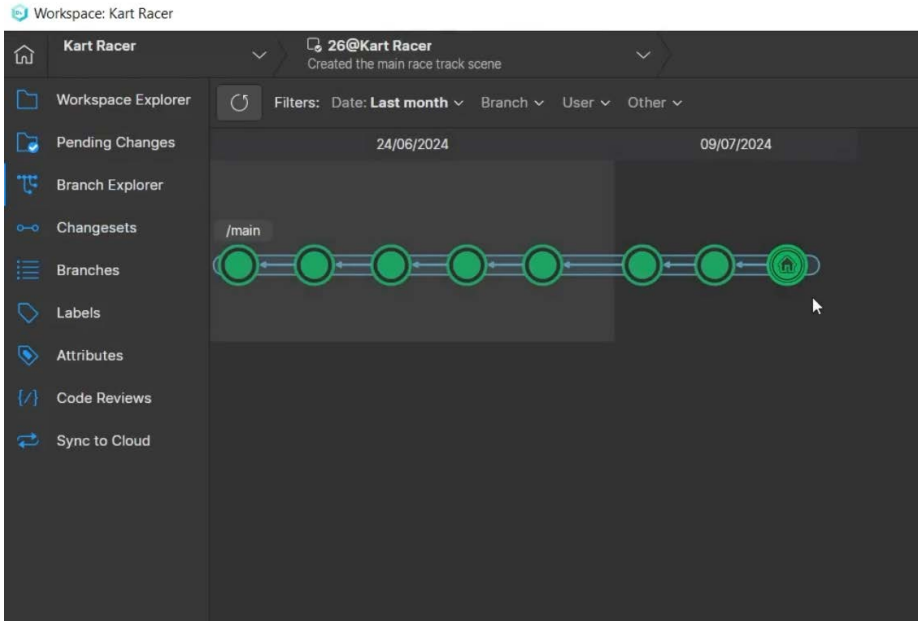
「Branch Explorer」ボタンをクリックして、UVCS デスクトップアプリを起動。

Gluon と Unity DevOps Version Control（後者が UVCS デスクトップアプリ）を含むバージョン管理アプリ

Gluon は初心者向けのソフトウェアで、UVCS はフル機能のオプションです。デフォルトでは、「Branch Explorer」をクリックすると、UVCS ソフトウェアが開きます。コンピューターにインストールされたソフトウェアから Gluon と UVCS を開くことで、これらを切り替えることができます。Unity は使用しているバージョンを自動で検出し、そのシステムに切り替えます。



Gluon はアーティストフレンドリーなバージョン管理アプリ。



UVCS はフル機能のソフトウェア。

Gluon を使用する

UVCS の導入セクションで説明したように、Gluon はアーティスト向けに設計された軽量クライアントです。作業予定のファイルだけを選択してサーバーからチェックアウトし、他のユーザーが編集できないようロックすることができます。作業が完了したら、再びファイルをチェックインします。Gluon GUI では、プログラマーには適していても技術的スキルが少ないユーザーには馴染みにくい、複雑なコンセプトが排除されています。

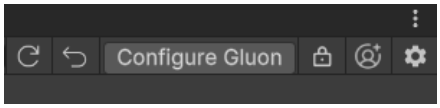
インストールされた Unity DevOps Version Control アプリから Gluon を開きます。

Gluon デスクトップアプリは、インストール済みの Unity DevOps ソフトウェアからアクセスできます。

フルモードから、Gluon が動作するパーシャルモードへの切り替えを促すプロンプトが表示されます。

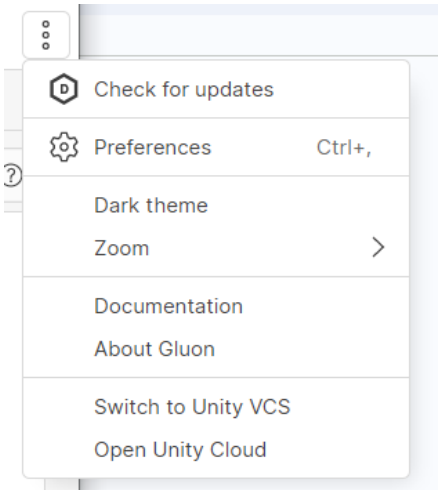
この警告の「**Run an update**」をクリックすることで、Gluon 作業時にパーシャルモードに切り替えることができます。

Unity 内では、**branches** アイコンが「**Configure Gluon**」ボタンに置き換わります。このボタンをクリックすると、都度 Gluon アプリが開きます。



Gluon を使用しているチームメンバーは、Unity の Version Control ウィンドウから直接 Gluon アプリを開けるようになります。

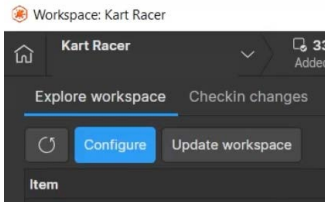
Gluon アプリの右上にあるオプションボタンをクリックすると、ダークテーマを有効にできます。



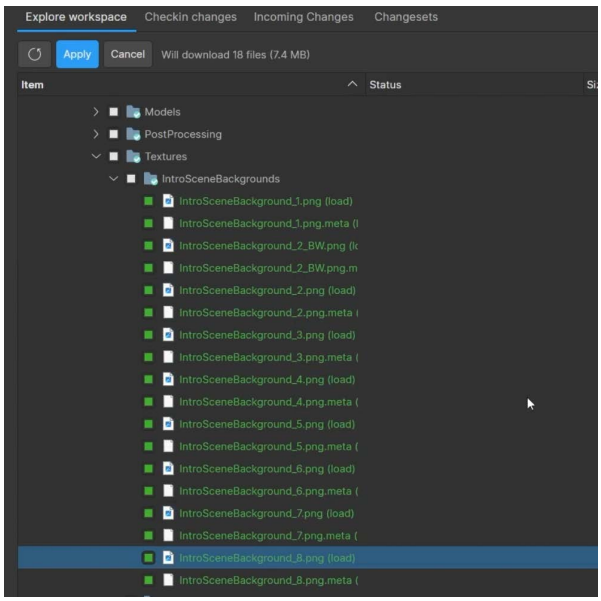
Gluon では、右上のドロップダウンオプションを使用してダークテーマを有効にできます。

Gluon はメインブランチのみを使用しアーティストフレンドリーなワークスペースを提供します。Gluon を使用すると、**Explore workspace** 機能に簡単にアクセスして、必要に応じてファイルをアップロードまたはダウンロードできます。

「Explore workspace」セクションで「**Configure**」ボタンをクリックすると、リポジトリ上のファイルを変更できます。



「**Configure**」ボタンをクリックして、リポジトリ上のファイルを変更する。

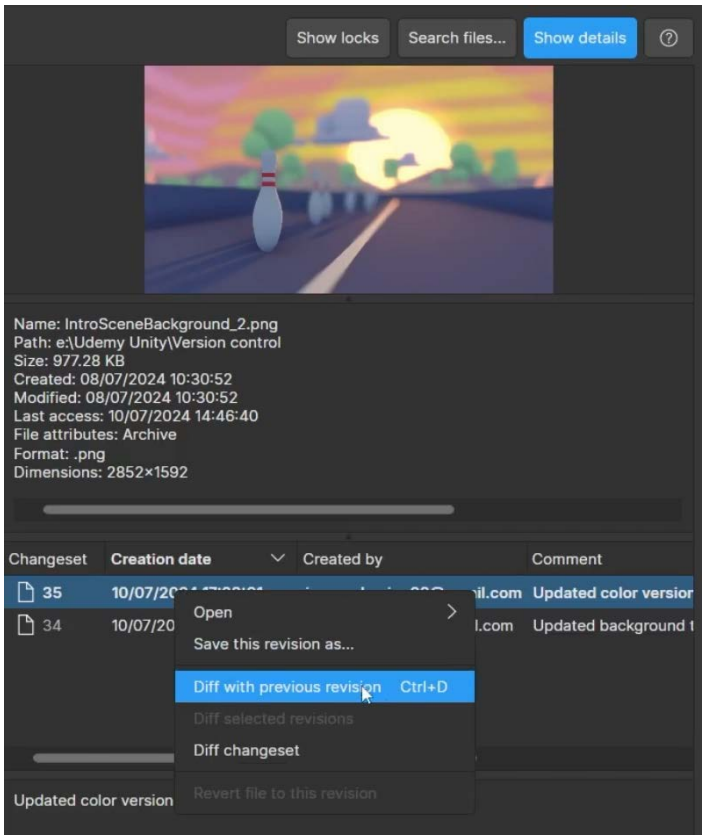


ダウンロードするダイルを選択し、Gluon で「Apply」をクリックする。

ファイルの隣にあるチェックボックスをクリックして、Explore workspace ビューに表示するファイルをダウンロードまたは除外できます。Unity Editor にダウンロードされるファイルは緑色に変わり、Unity から外すファイルは赤色に変わります。作業が完了したら「**Apply**」をクリックし、その後「**Update workspace**」をクリックして、自分の Unity でファイルが表示されるようにします。

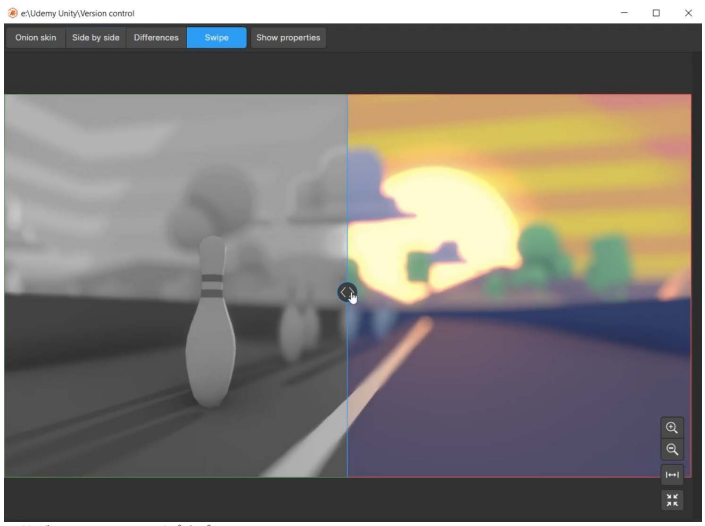
Gluon 内でファイルを削除した際に影響を受けるのは、現在の変更セットと今後作成する変更セットのみです。削除されたファイルは、以前の変更セットで参照されるため、クラウド上には残ります。

Gluon アプリの右側にあるオプションセクションでリビジョンを右クリックし、「**Diff with previous revision**」を選択することで、複数回変更されたテクスチャの変更を視覚的に確認できます。



「**Diff with previous revision**」を選択して、以前のリビジョンを視覚的に確認。

これにより、以前のリビジョンと現在のリビジョンの差分が表示されます。複数の表示方法があり、例えばスワイプオプションを使用すると、スワイプスライダーを画像上でドラッグして変更前と変更後を比較できます。



Diff ビューアーのスワイプオプション

UVCS デスクトップアプリ

UVCS デスクトップアプリは、完全版のバージョン管理ソフトウェアで、ブランチ機能を利用したいチームメンバーに最適です。これは、インストール済みの Unity DevOps Version Control アプリケーションから開くことができます。

Gluon をすでに使用したことがある場合、パーシャルモードから、UVCS が動作するフルモードに切り替えるプロンプトが表示されます。

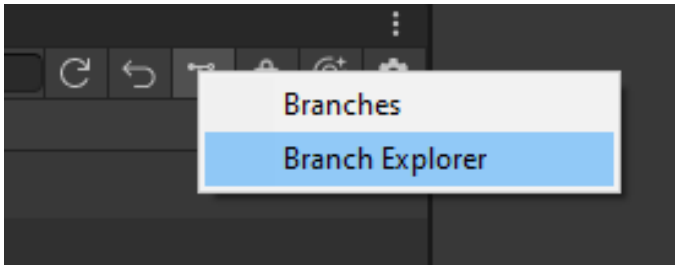
UVCS デスクトップ版バージョン管理ソフトウェアを使用するために、パーシャルモードからフルモードに切り替える。

ブランチ

ブランチは、チームメンバーが互いに独立して同じプロジェクト上で作業することを可能にします。例えば、チームメンバーの 1 人が別のブランチで新機能を開発し、他のメンバーがメインブランチで作業を進めることが可能です。機能の例として、実績用コードの作成や、異なる種類のピックアップメカニクスの実装などがあります。

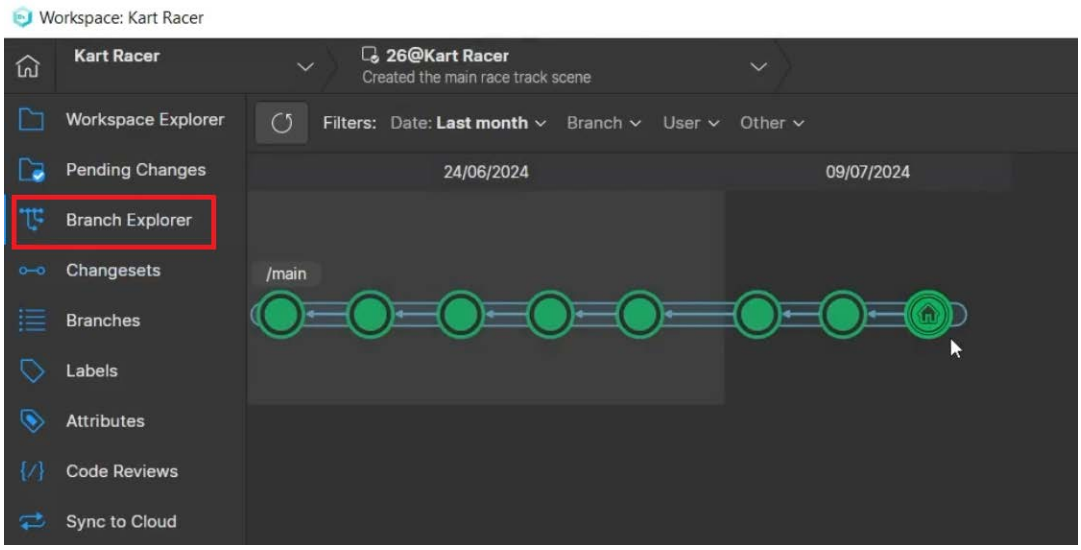
タスクが完了したら、作業ブランチをメインブランチにマージすることができます。その後、同じプロジェクトで作業するすべてのチームメンバーがワークスペースを更新し、変更を取り込めるようになります。

Unity エディターの Version Control ウィンドウ内にある「Branch Explorer」を選択すると、UVCS アプリが開き、「Branches」セクションに移動します。

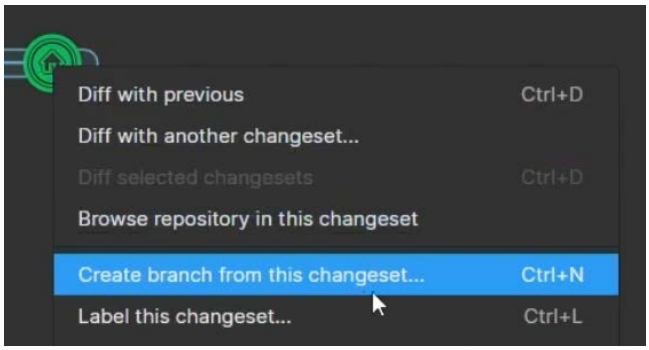


UVCS デスクトップアプリから「Branch Explorer」セクションを開ける。

円は各変更セットを表しています。現在の変更セットには家のアイコンが表示されます。



変更セットの 1 つを右クリックし、「**Create branch from this changeset**」を選択します。

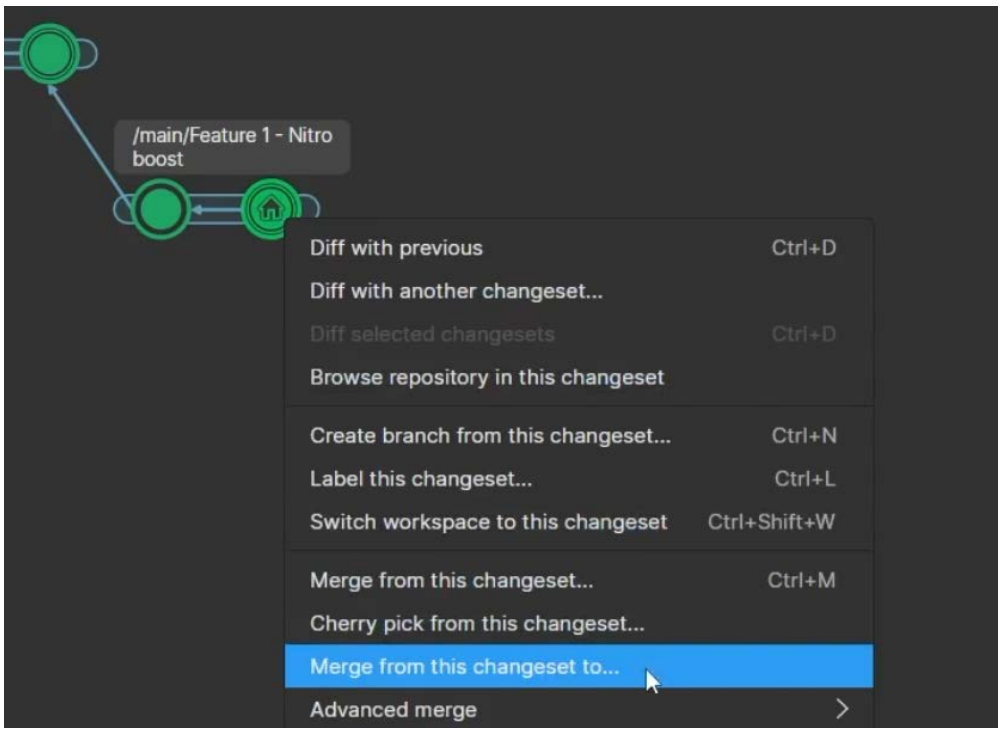


変更セットからブランチを作成するオプション

これで、新たなブランチ上で作業を行うことができます。他のメンバーがメインブランチ上や別の機能ブランチ上で行った変更は、このブランチには送られません。これにより、他の人の変更が現在の作業と競合するリスクを減らすことができます。他の人の変更は、マージ後に統合することができます。

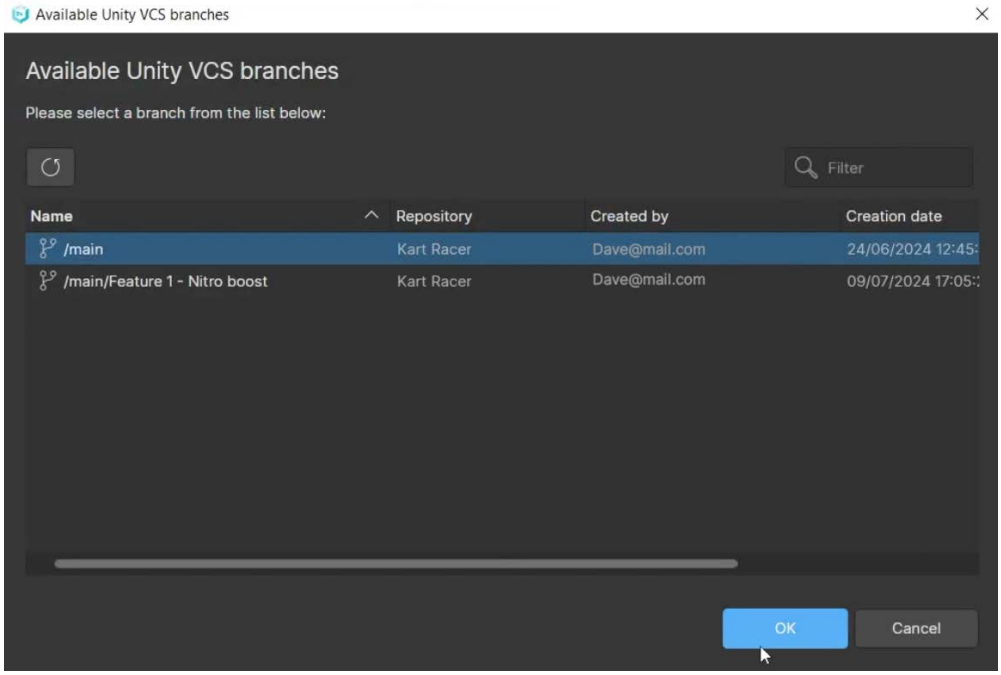
ブランチには、メインブランチと同様に、円で示される複数の変更セットが含まれることがあります。

ブランチ上での作業が完了したら、右クリックして「**Merge from this changeset to…**」を選択することで、作業ブランチをメインブランチにマージできます。



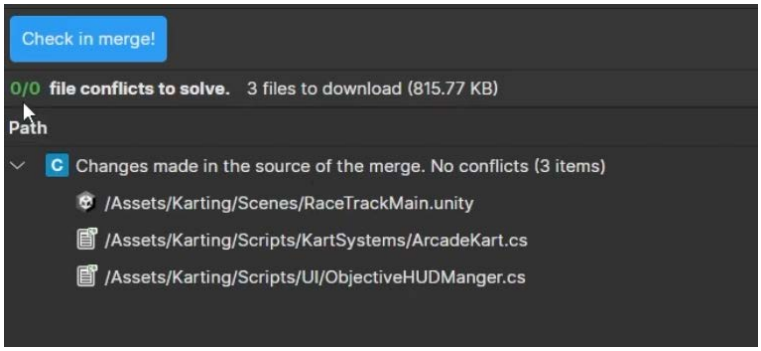
作業ブランチを他のブランチ（メインブランチなど）にマージ。

マージ可能なブランチが表示されます。マージ先のブランチを選択してください。



マージ可能なブランチ

これにより、マージセクションが開き、競合が発生しているかどうかを確認できます。競合がなければ、「**Check in merge**」ボタンをクリックしてください。

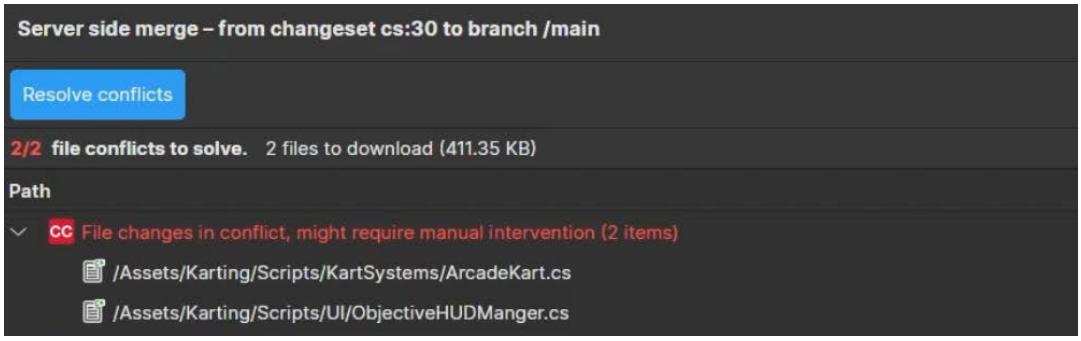


UVSCS はブランチ上のすべてのファイルを確認し、メインブランチとの競合がないかを調べます。

メインブランチのメンバーが同じファイルに変更を加えていた場合、マージ時に競合が発生する可能性があります。

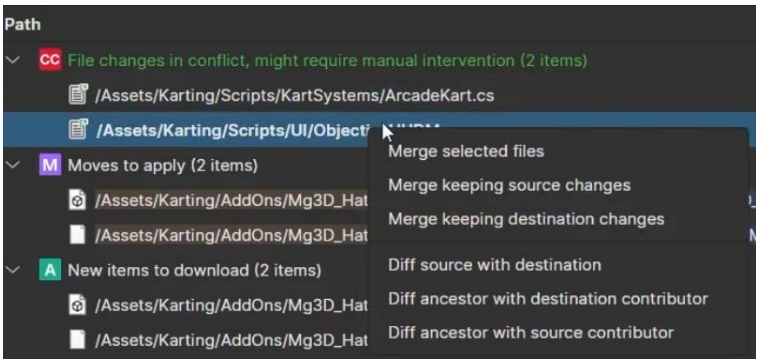
競合の処理

スクリプトやシーンファイルが、異なるブランチ上で作業している複数のメンバーによって変更されることがあります。それらの変更をマージしようとすると、競合が発生する可能性があります。



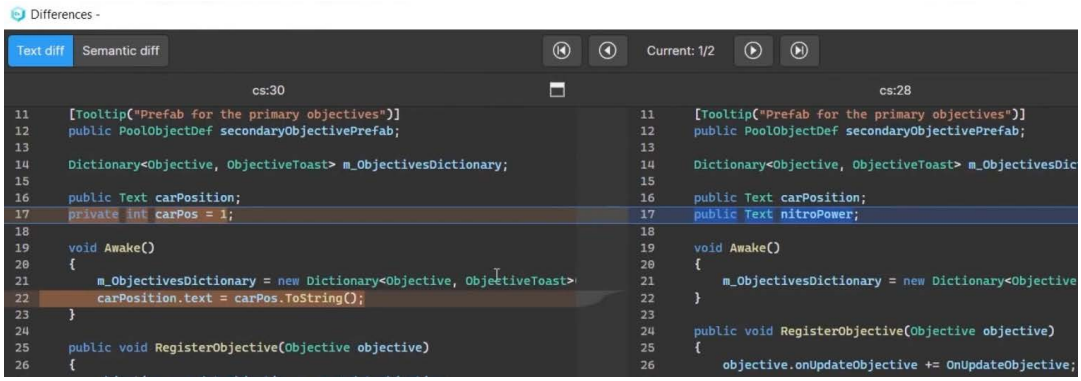
2 件の競合が検出され、競合を含むファイルが表示されている

競合があるファイルを右クリックして差分を表示し、変更を保持するか破棄するかを選択してください。



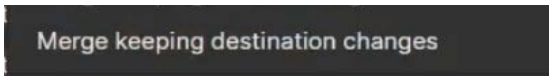
右クリックメニューに、マージ競合を処理するためのオプションが表示されている

スクリプトの場合、「**Diff source with destination**」を選択することができます。これにより、Diff ビューアーが開き、マージ先のファイルとの違いを確認できます。

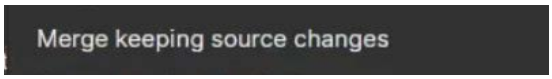


Diff ビューアーでは、自分のファイルとマージ先のブランチのファイルとのコードの差分が表示される。

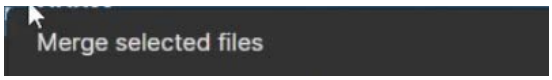
これを使用して、他のブランチのファイルを自分の作業ファイルで上書きするかどうかを判断します。上書きするには、「**Merge keeping destination changes**」を選択します。



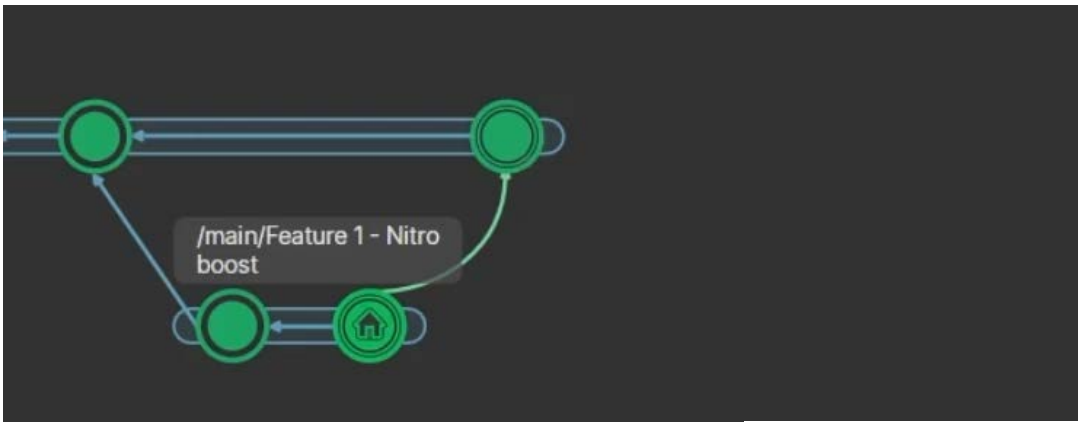
また、自分の変更を破棄して、他のブランチのファイルを保持することもできます。自分の変更を破棄するには、「**Merge keeping source changes**」を選択します。



最後に、「**Merge selected files**」を選択すると、UVCS は両方の変更を保持しつつマージを試みます。ただし、マージ後にスクリプトが正しく統合されているか確認することをお勧めします。

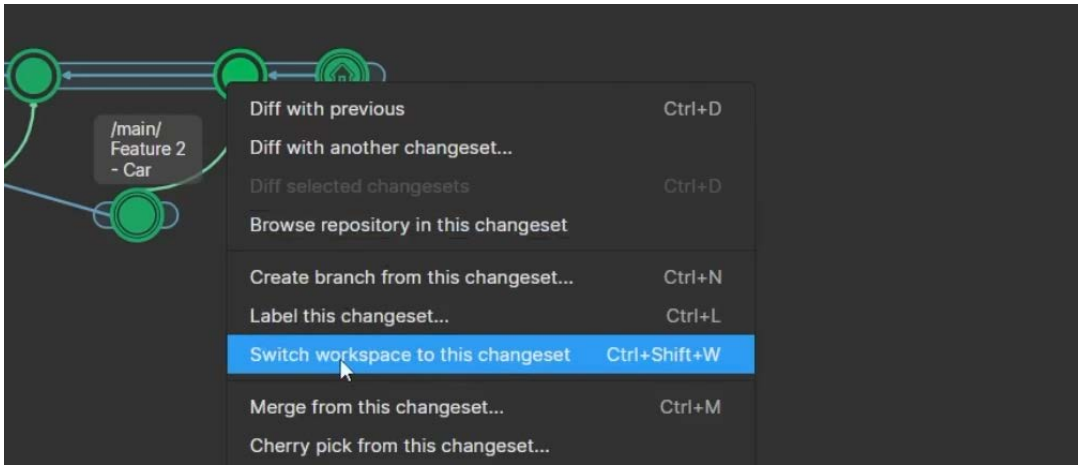


Branch Explorer には、どの変更セットが他のブランチからマージを継承したかが表示されるようになります。



メインブランチにマージされたブランチ

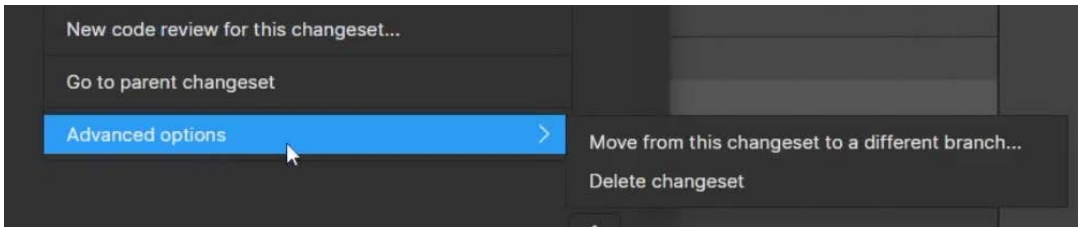
また、以前の変更セットを右クリックして「**Switch workspace to this changeset**」を選択すると、その時点の状態に戻すことができます。これは、追加したアイテムやスクリプトによってゲームに問題が発生した場合に必要になることがあります。



以前の変更セットを右クリックすることで、その時点の状態に戻すことができる。

重大なエラーやバグを引き起こした変更セットは削除できます。その変更セットから派生した変更セットやラベル、サブブランチがあってはいけません。

エラーが発生している変更セットを右クリックし、「**Advanced options**」>「**Delete changeset**」を選択してください。

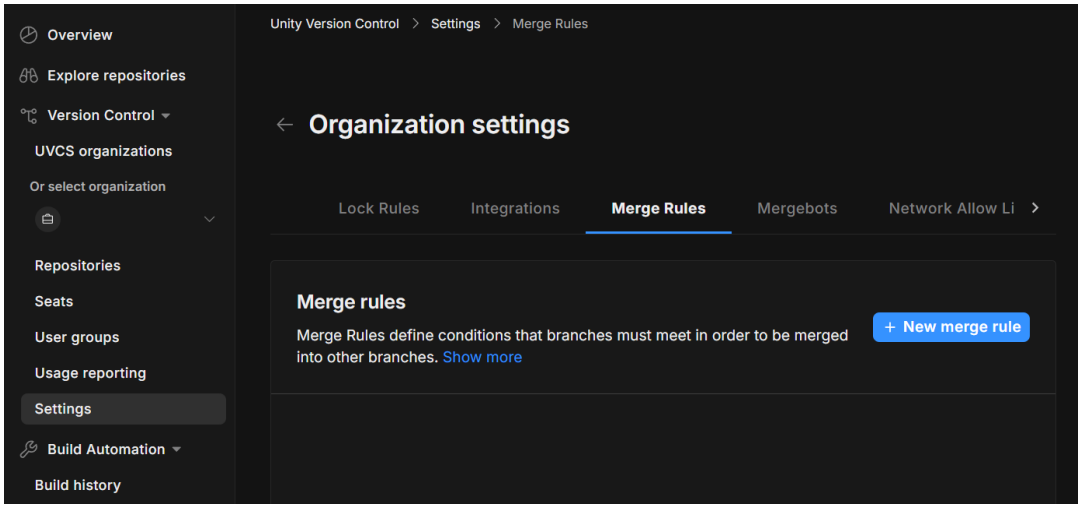


右クリックメニューから変更セットを削除する。

ルールをマージする

マージ競合がチームの作業を妨げないよう、マージルールを設定することをお勧めします。この方法は、大規模なチームで、メインブランチにマージする前にチームマネージャーがブランチを承認する必要がある場合に有効です。これにより、メインワークフローにエラーが持ち込まれるのを防げます。

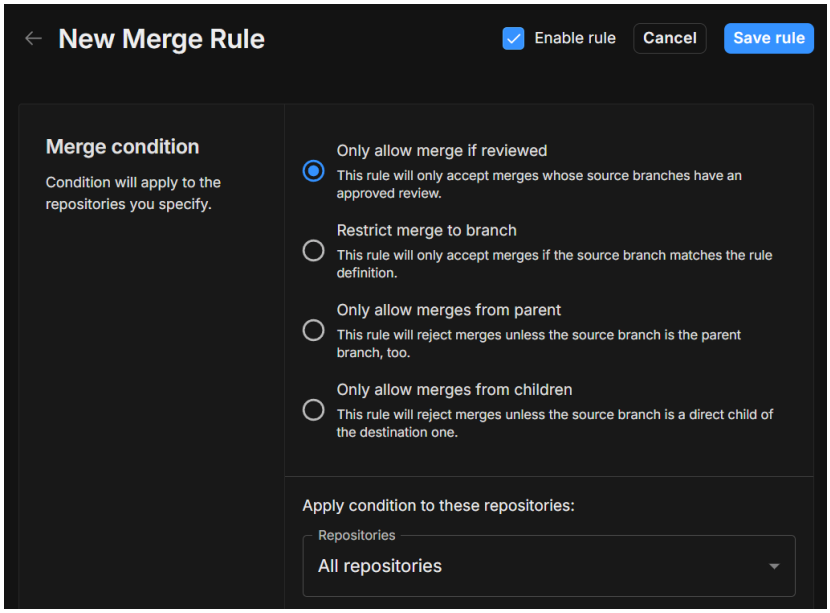
Unity Cloud ダッシュボードにログインし、**Settings** から **Merge rules** を開きます。



Unity Cloud 設定で新しいマージルールを作成する。

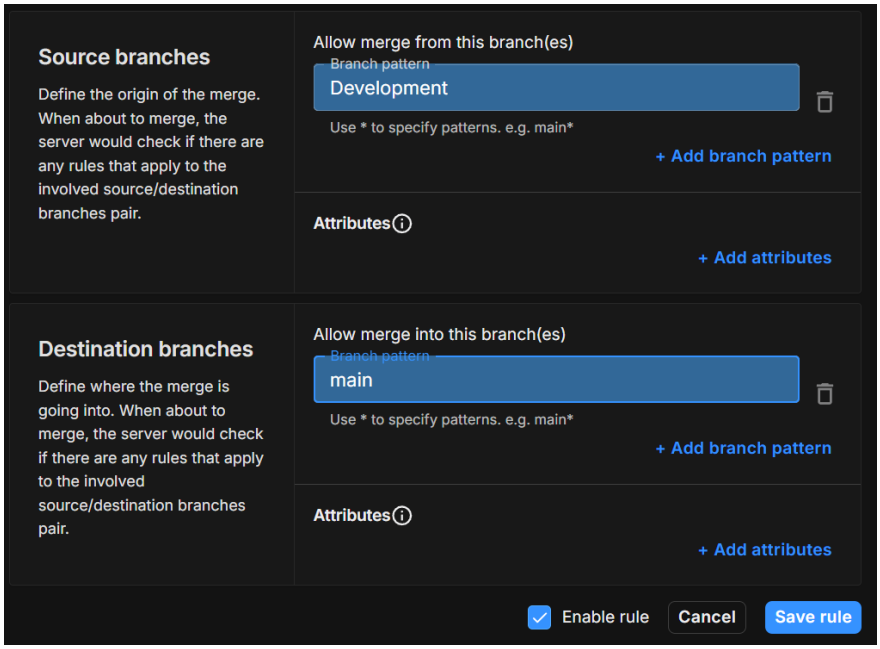
「**New merge rule**」ボタンをクリックし、4 つのオプションから 1 つを選択します。「**Only allow merge if reviewed**」は、他のチームメンバーやチームマネージャーがレビューを行い承認しない限りマージできないように設定できます。

この設定は、複数のリポジトリで作業している場合、すべてのリポジトリに適用することも、ドロップダウンリストから特定のリポジトリを指定することもできます。



「**Only allow merge if reviewed**」オプションを選択した場合、他のチームメンバーのレビューを受けないとマージできなくなる。

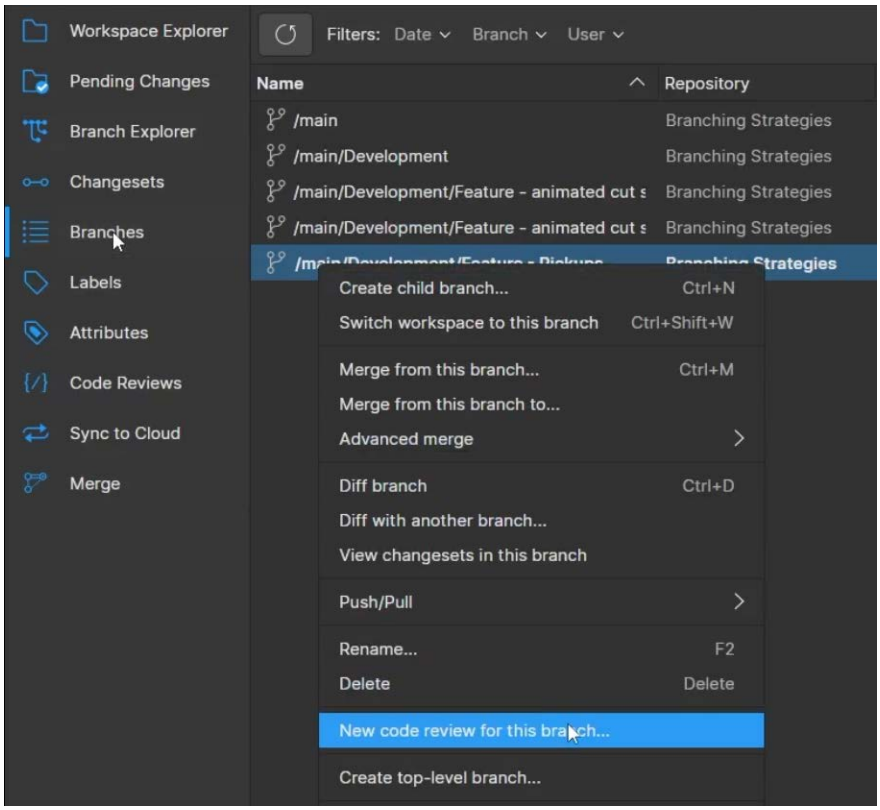
マージルールを適用したいブランチを指定します。作業中のブランチを指すソースブランチと、マージ先となるデスティネーションブランチの指定が必要です。「**Save rule**」ボタンをクリックすると、ルールが有効になります。ルールは、Unity Cloud からいつでも無効化または削除することができます。



マージルールを適用するブランチを選択する。

ブランチをマージするチームメンバーは、ブランチレビューをリクエストする必要があります。

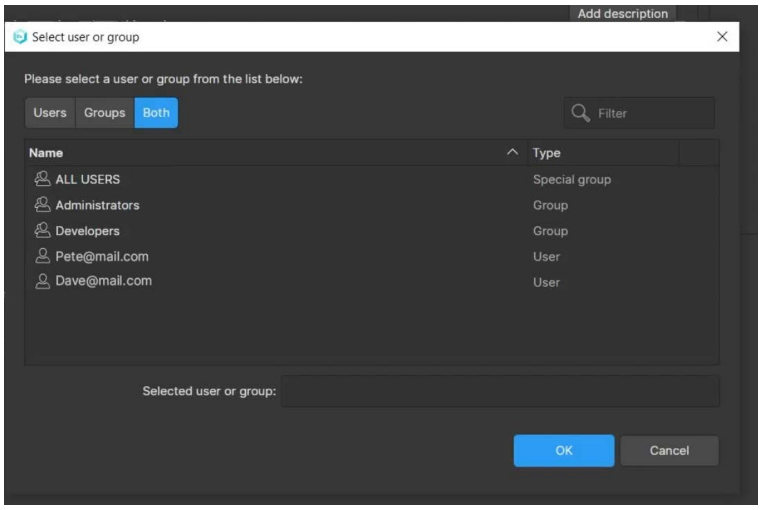
「Branches」セクションに移動し、マージしたいブランチを右クリックします。「**New code review for this branch**」オプションを選択します。



マージするブランチのコードレビューをリクエストします。

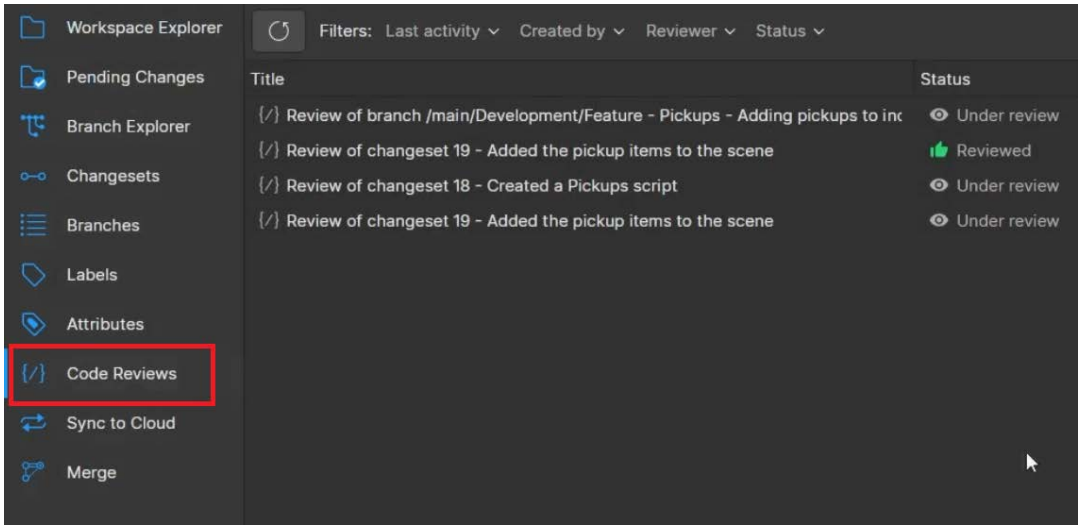
これには、ブランチ全体で変更されたすべてのファイルが含まれます。

「**conversations**」をクリックしてメッセージを残し、他のチームメンバーにレビューを依頼しましょう。招待されたすべてのチームメンバーがレビューを完了しない限り、マージは実行できません。そのため、関係のあるメンバーだけを含めるようにしてください。招待されたメンバーにはコードレビューの依頼メールが送信されます。



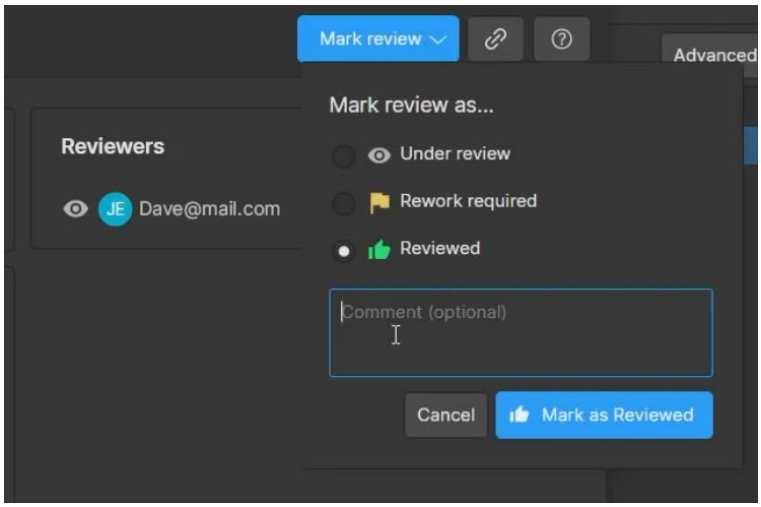
ブランチのレビューを依頼するメンバーまたはグループを選択してください。

レビューの進捗を確認するには、「**Code Reviews**」に移動します。ダブルクリックして開き、自分宛てのメッセージが残されているか確認します。



コードレビューには、リクエストされたレビューの進捗状況が表示される。

招待されたチームメンバーは、ブランチを「**Rework required**（再作業要）」または「**Reviewed**（レビュー完了）」とマークし、メッセージを残せます。

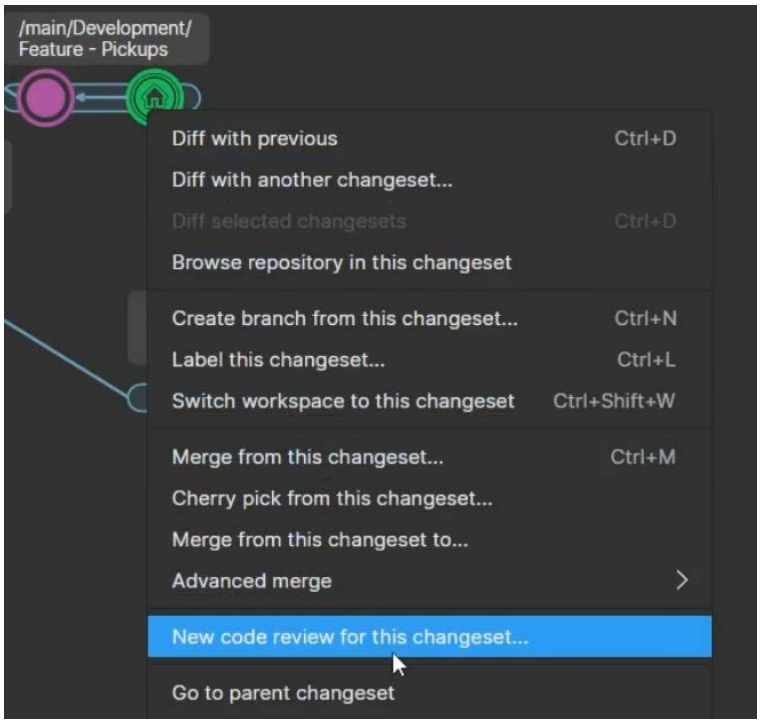


レビューアーは「**Mark Review**」ボタンをクリックして、「Rework required」または「Reviewed」とマークできる。

ブランチのレビューが完了すると、マージすることができます。これで、プロジェクトの安全性が向上します。

特定の変更セットに対しても、同じコードレビューのプロセスを適用できます。これは、他のチームメンバーに、コードのバグや問題を確認してもらいたい場合や、コードの改善案を提案してもらいたい場合に役立ちます。

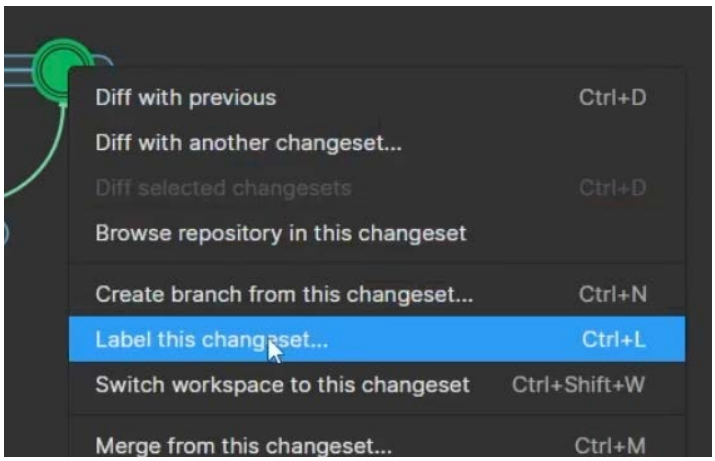
これを行うには、変更セットを右クリックして、「**New code review for this changeset**」を選択します。その後、チームメンバーを招待して、ファイルを確認してもらい、改善案を提案してもらうことができます。



変更セットに対するコードレビューをリクエストする

変更セットにラベルを追加することもできます。例えば、ブランチの分岐に使用できる、承認済みバージョン（例：Version 1.0）の変更セットや安定した変更セットの場合に便利です。

変更セットを右クリックし、「**Label this changeset**」を選択します。ラベルは、変更セットが削除されるのを防ぎます。必要に応じて、ラベルセクションからラベルを削除することができます。



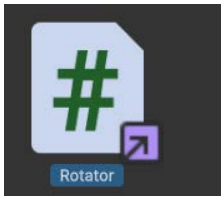
右クリックメニューから変更セットにラベルを付ける。

ラベルは、この変更セットに関する重要な情報を他のユーザーに伝えることができる。

ファイルのロック

作業中のファイルをロックすることで、大規模なマージ競合を防止できます。ロックしたファイルはクラウド上でロックされ、その間、他のチームメンバーは編集できなくなります（ただし、読み取り専用でアクセスすることは可能です）。ファイルがチェックインされるとロックが解除され、他のチームメンバーが編集できるようになります。

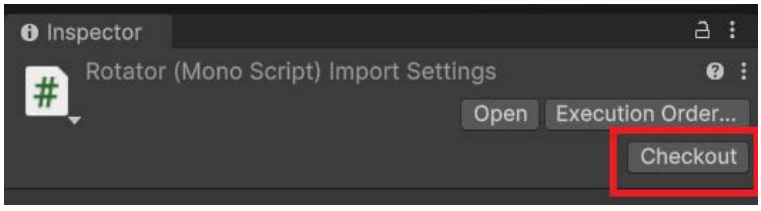
ロックは自動では適用されません。デフォルトでは、Unity で作業内容を保存すると、編集中のファイルにチェックアウトアイコンが表示されます。



紫色のチェックアウトアイコンが Unity のファイルアイコンや、他のメンバーのバージョン上でも表示され、誰かがそのファイルで作業していることを知らせます。

紫色のボックス内のチェックアウトアイコンが、チームメンバー全員のコンピュータで表示されます。

ファイルは、作業開始前に手動でチェックアウトできます。Inspector 内の「**Checkout**」ボタンをクリックしてください。他のチームメンバーは、あなたがそのファイルを作業中であることが分かります。ただし、これはファイルロックではないため、他のメンバーもそのファイルに変更を加えることができます。これは、ファイルロックほど効果的ではありません。

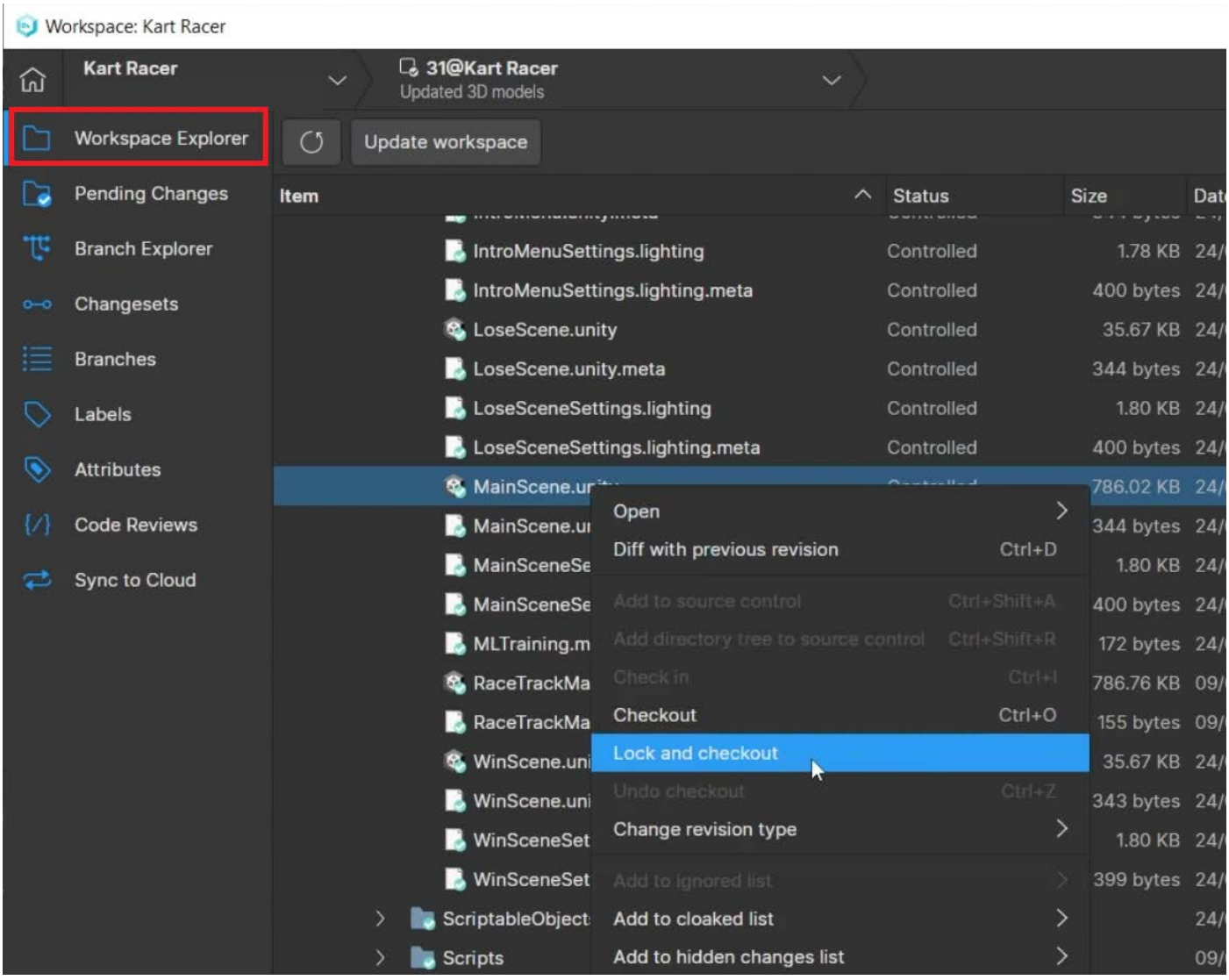


ファイルを編集している場合は、チェックアウトを行い、作業中であることを他のチームメンバーに通知してください。

UVCS はスマートロックを使用します。これにより、同じファイルに変更を加えようとする他のチームメンバーは、必ず最新のリビジョンを使用することが求められます。古いバージョンを使用しているブランチで作業している場合もあるため、これは重要です。スマートロックは、メインブランチを含むすべてのブランチを調べてファイルの最新リビジョンを確認し、古いバージョンではなく最新のバージョンを基に作業するよう保証します。

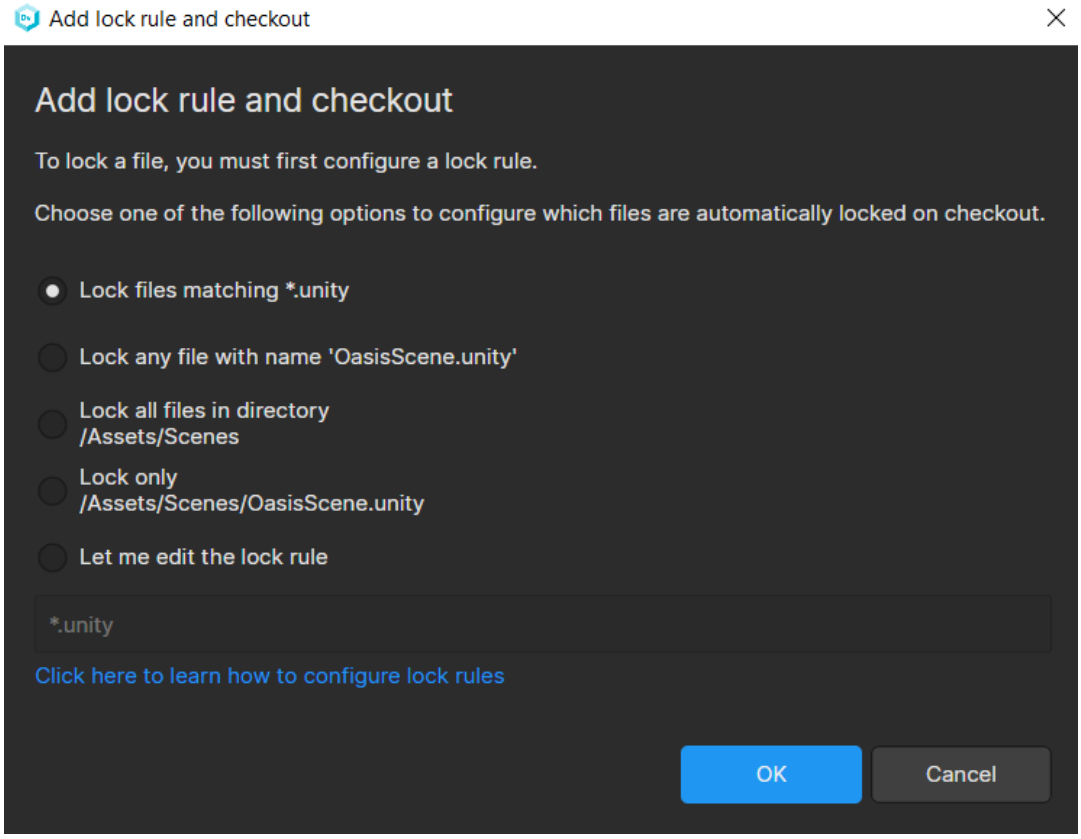
ファイルをロックするには、Branch Explorer に移動してください。

Workspace Explorer ビューで、ロックしたいファイルを見つけ、右クリックして「**Lock and checkout**」を選択します。この操作は、他のチームメンバーがそのファイルをチェックアウトしていない場合にのみ有効です。



UVCS デスクトップアプリの Workspace Explorer でファイルをロックする。

この種類のファイルに対してロックルールを定義できます。この特定のファイルのみをロックする場合は、下記リストの 4 番目のオプションである「**Lock only**」を選択してください。



このファイルまたはファイルタイプに対してロックルールを定義する。

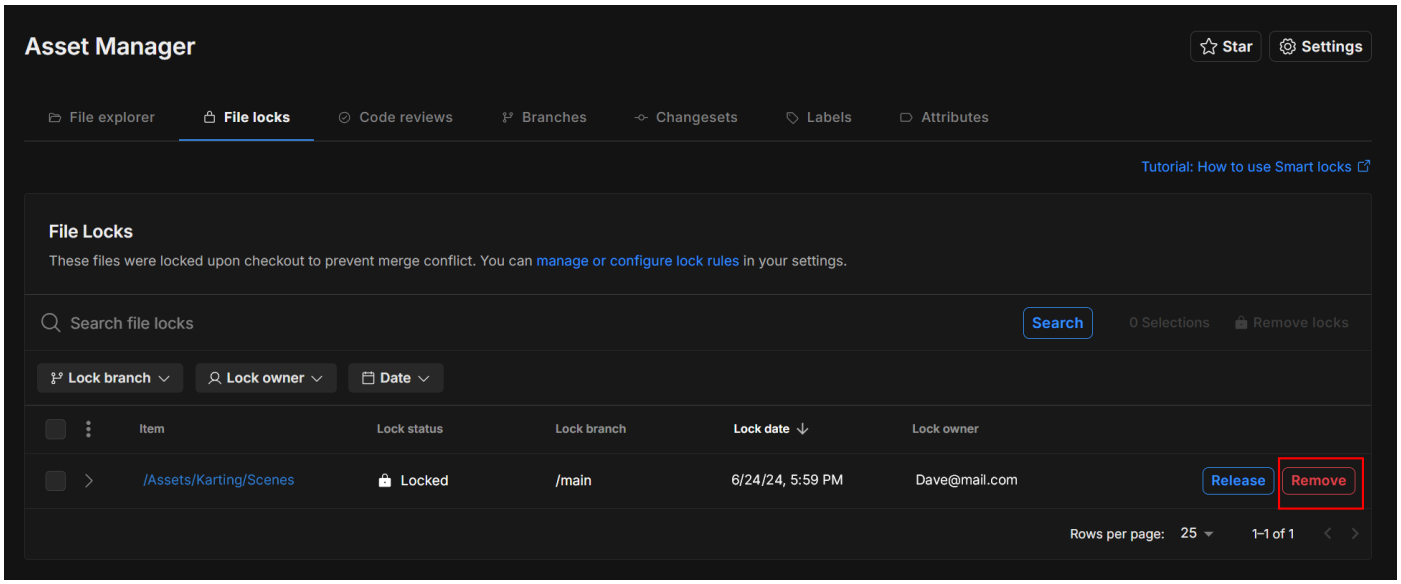
ファイルは、チームメンバーが使用する Unity 内で紫色のロックアイコンとして表示され、編集のためにファイルがロックされていることを他のメンバーに知らせます。

紫色のロックアインコンは、ファイルがロックされていることをチームメンバーに知らせる。

チームメンバーは、そのファイルを読み取り専用として閲覧できますが、あなたがチェックインするまで変更を加えることはできません。

編集が完了したら、ファイルをチェックインしてロックを解除します。

管理者権限を持つチームメンバーは、クラウドにログインして「**Repositories**」を開き、「**File locks**」に移動してロックを手動で解除できます。

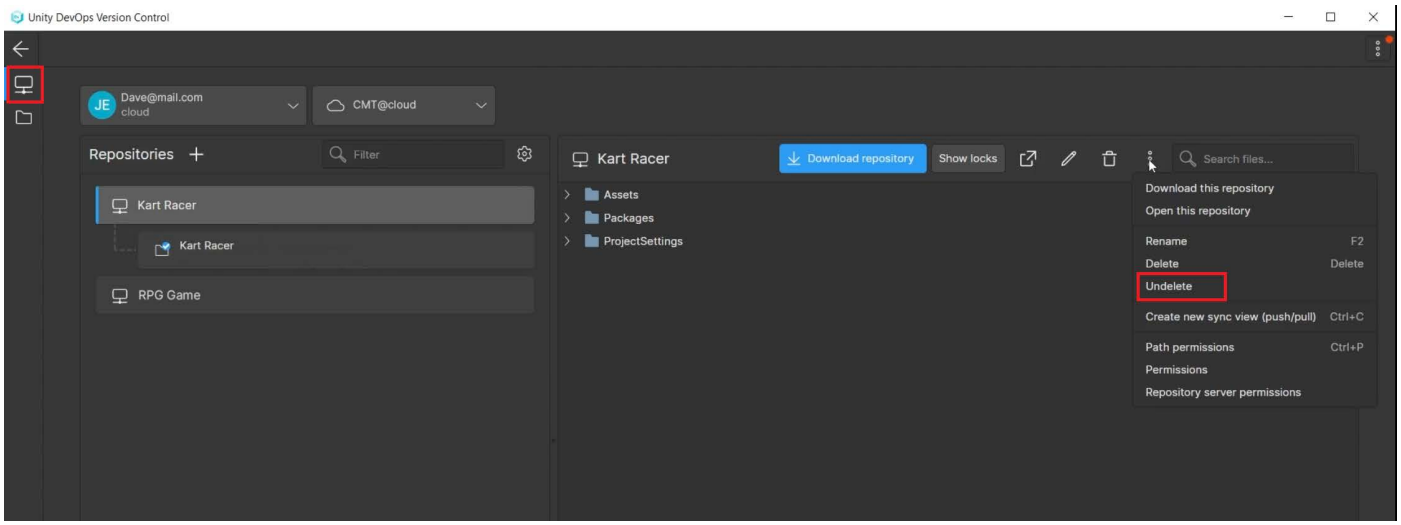


クラウド上でのファイルロックの解除

リポジトリを監視または削除する

リポジトリを監視するには、デスクトップアプリを使用します。右上の **Home** アイコンをクリックし、上部のアイコンを選択すると、現在のリポジトリを閲覧できます。

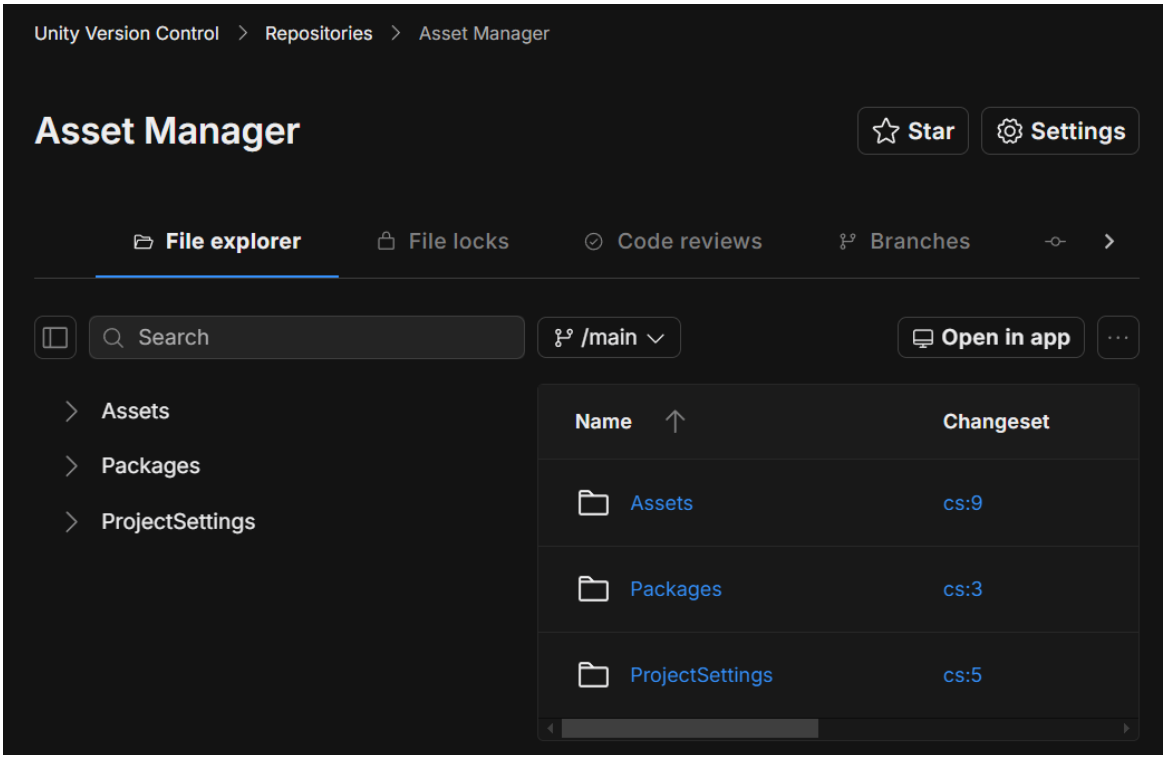
ここからリポジトリ全体をコンピュータにダウンロードしたり、デスクトップアプリで開いたり、名前を変更したり、削除したりできます。削除されたリポジトリは、再度必要になった場合に備えて、クラウド上に 10 日間保存されます。削除したリポジトリを復元するには、「**Undelete**」を選択してください。



リポジトリのダウンロード、削除、復元

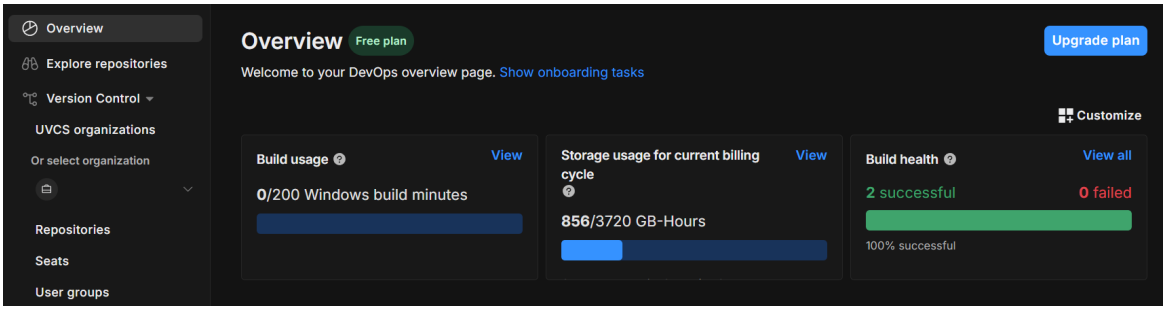
「DevOps」セクションでは、クラウド上のすべてのリポジトリを閲覧できます。ここでは、使用しているストレージ容量（通常、ローカルのプロジェクトよりも小さい）を確認できます。

また、リポジトリの追加や削除も行えます。「DevOps」セクションでは、使用状況レポートや設定など、DevOps アカウントに関する詳細情報が提供されます。



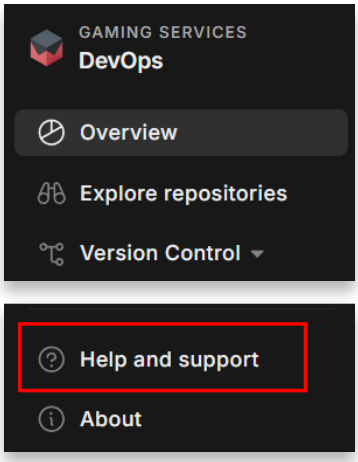
Unity Cloud の「DevOps」セクションにログインし、リポジトリのストレージ情報を閲覧する。

クラウドの「**Overview**」セクションでは、現在のデータ使用状況を確認できます。ここには月ごとの GB 時間が表示されます。詳しくは、Unity Cloud 料金プランページを参照するか、Unity Cloud ダッシュボードにサインインして DevOps ダッシュボードの「About」ページをご確認ください。



クラウドの「Overview」セクションでは、月当たりのストレージ使用状況を確認できる。

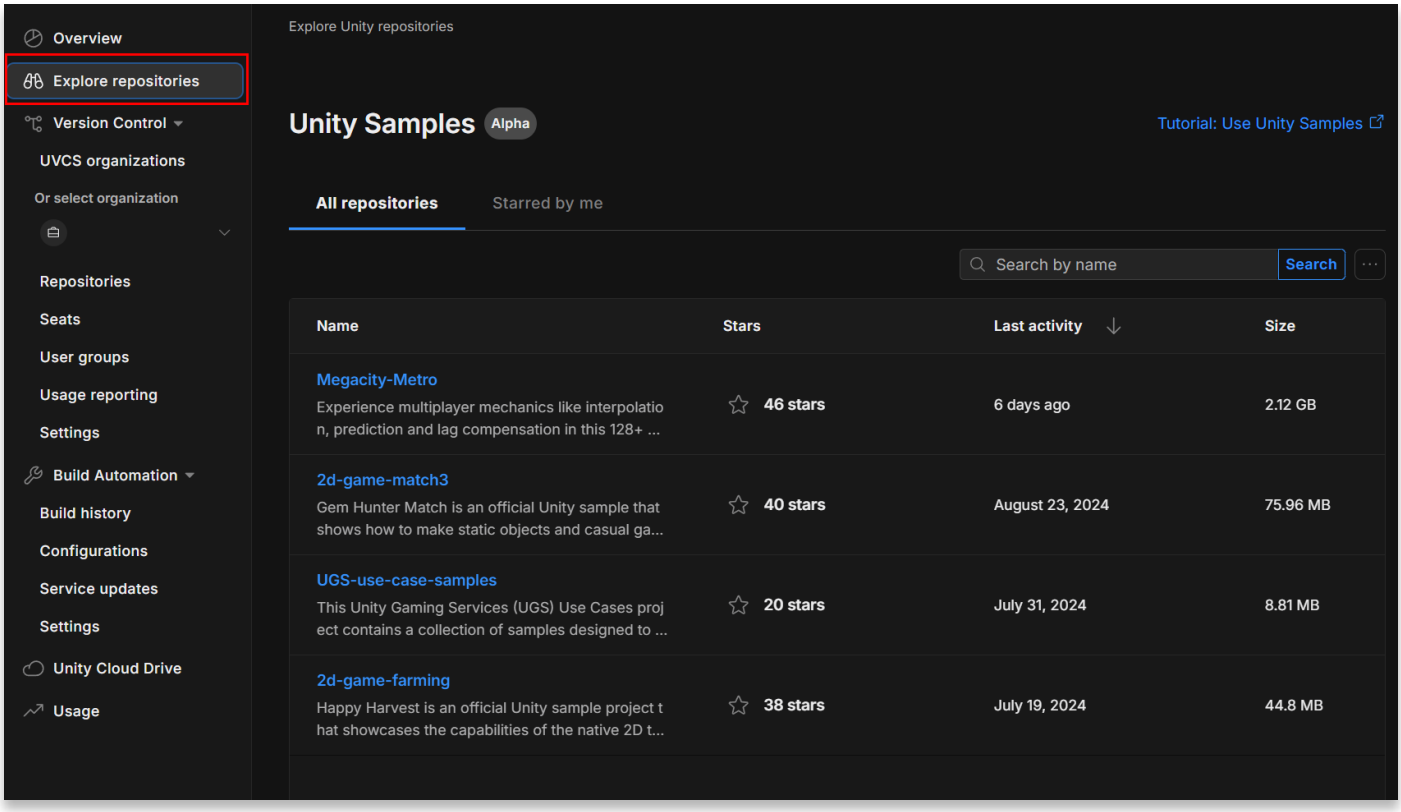
Unity サポート



Unity Cloud ダッシュボードの「DevOps」セクションで、「**Help and support**」をクリックする。

クラウドダッシュボードで、「**Help and support**」ボタンをクリックすると、FAQ 、フォーラム、ブログ記事を確認したり、サポートチケットを作成して問題解決の支援を受けたりすることができます。

また、「**Explore repositories**」にある **DevOps** カテゴリーから、サンプルリポジトリプロジェクトを取得できます。これらの公開リポジトリは、作業用テンプレートとしてコンピュータにダウンロード可能です。



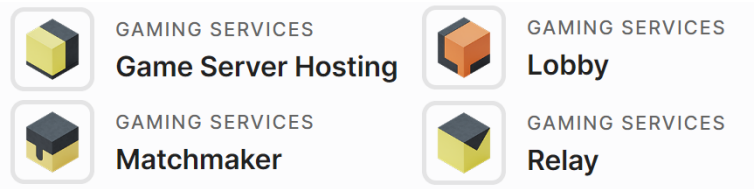
ダウンロード可能な公開サンプルリポジトリの例

ライブゲームの基盤を構築する

Unity は、DevOps に加えて、ライブゲームを管理するための包括的なエコシステムを提供しています。Unity Gaming Services（UGS）を活用すれば、複数のプラットフォームでプレイヤーを効率的に獲得および接続し、バックエンドインフラを管理できます。以下は UGS の一部のサービス概要です。

Unity Gaming Services

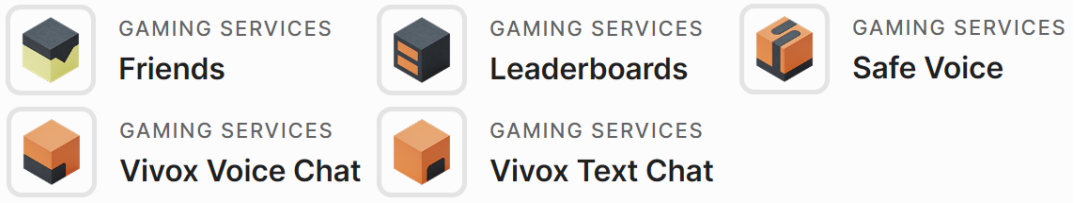
マルチプレイヤー



Unity は 2 つのネットワーキング スタックである Netcode for GameObjects と Netcode for Entities、そして Unity Gaming Services のマルチプレイヤーツールである Game Server Hosting や Vivox Voice Chat を提供しています。Unity のエコシステム内で構築することも、ニーズに合わせて任意のツールやサービスを組み合わせることもできます。

Unity マルチプレイヤーについて詳しくはこちら

コミュニティ



2 人用プロジェクトから、数百万人の同時接続ユーザー向けまで、あらゆる規模のゲームに対応しており、エンジンに依存しない音声 / テキストチャットを含む、多様なコミュニティゲーミングサービスが利用可能できます。フレンド機能やリーダーボードなどのソーシャル要素をゲームに追加して、プレイヤーのエンゲージメントやリテンションを向上させましょう。

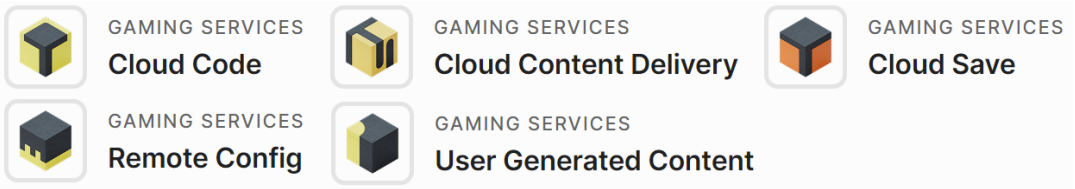
Unity ゲームコミュニティソリューションについて詳しくはこちら

アカウント

Unity Authentication と Unity Player Accounts を使用すれば、プレイヤー情報を柔軟に管理できます。ゲームやプラットフォームに最適な機能を組み合わせて利用できます。

Unity Authentication と Player Accounts について詳しくはこちら。

コンテンツ管理

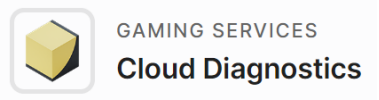


Cloud Save を使用すると、ゲームデータをクラウドに保存し、より快適なプレイヤー体験を提供できます。Cloud Content Delivery を使用すると、強力なアセット管理とコンテンツ配信機能を活用して、クラウド経由でゲームアップデートを構築およびリリースできます。また、毎月最初の 50 GB の帯域幅は無料で利用できます。

Remote Config サービスでは、Game Overrides を使用して、特定のプレイヤーセグメントを対象にした A/B テストを効率的に実施できます。コードの変更やアプリの更新を行うことなく、すべてのプレイヤーまたは特定のプレイヤーグループに対して、ゲームの難易度、広告の頻度、一般的な属性を変更できます。

Unity のコンテンツ管理について詳しくはこちら

クラッシュレポート



Cloud Diagnostics サービスは、ゲームの安定性を向上させ、プレイヤーの離脱率を低減するのに役立ちます。このサービスでは、リアルタイムのクラッシュデータが提供されるため、チームはバグを迅速に修正し、プレイヤーの離脱を防ぐことができます。Unity は、無料で始められ、プロジェクトの成長に合わせて拡張可能なクラッシュおよびエラー報告ソリューションを提供しています。

Unity Cloud Diagnostics について詳しくはこちら

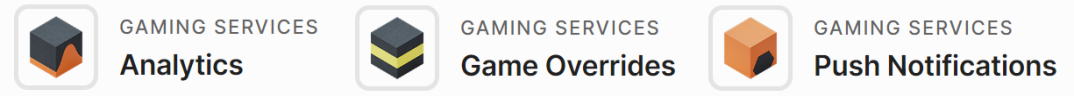
Game Economy

Game Economy サービスでは、カスタマイズ可能なゲーム内経済を構築するための機能が提供されます。これにより、プレイヤーにシームレスな購入体験、通貨変換、在庫管理などを提供できます。完全なゲーム内経済システムを実装し終えてからは、簡潔に設計されたダッシュボード上で効率的に管理できます。

Unity の Economy サービスについて詳しくはこちら

エンゲージメントと分析

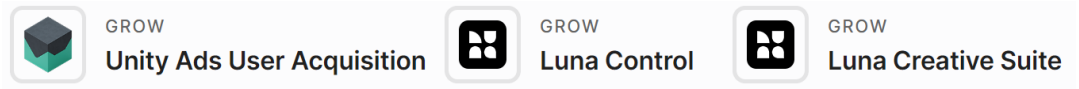
ゲームを理解し、プレイヤーとのエンゲージメントを高めましょう。これらのサービスを活用することで、ゲーム体験を効率的に向上させ、プレイヤーを維持し、ゲームを成長させることができます。



Unity Analytics について詳しくはこちら

Unity Grow

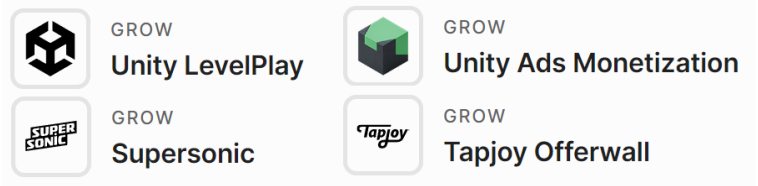
ユーザー獲得



Unity が提供するユーザー獲得サービスのスイートを使用すれば、ゲームに適したモバイルユーザーを獲得できます。世界中に広がる Unity Ads のネットワークを活用して、広告費用対効果（ROAS）、リテンション、またはスケールに合わせてキャンペーン目標を最適化できます。

Unity のユーザー獲得について詳しくはこちら

Monetization



市場で最先端のテクノロジーと強力なツールセットを活用して、アプリの収益可能性を最大限に引き出し、効果的にゲームを収益化しましょう。

Unity monetization について詳しくはこちら

まとめ

このガイドが、チームの一員として Unity でバージョン管理に取り組む際の不安を軽減する助けになれば幸いです。ソロプロジェクトに取り組む場合でも、プロジェクトの整理やバージョン管理の原則は非常に役立ちます。

最も重要なのは、チーム内での明確なコミュニケーションと、使用ツール、プロセス、構成、コードスタイルに関する共通認識を持つことです。チームとして、プロジェクトの構成方法や使用するバージョン管理システム、そのシステム内でのワークフローの進め方などについてのガイドラインを設定し、それに同意する必要があります。その後、JIRA、GitLab、ビルドツール、自動テストなどのツールを統合する段階になると、これまで行ったプロジェクト構成やワークフローに関する取り組みが本領を発揮します。

最後に、以下のリソースをご確認ください。本書で取り上げたさまざまなバージョン管理システムに関する情報や、Unity プロジェクトを成功させるための追加ヒントが含まれています。

追加リソース

- バージョン管理システムを選択する際に考慮すべき 8 つの要素
- バージョン管理の概要（Unite Now 2020）
- Git Apprentice 著：Chris Belanger、Bhagat Singh
- Unity の Plastic SCM を使ったゲームのバージョン管理
- Plastic SCM プロダクトドキュメント
- Plastic SCM の Mergebot
- バージョン管理を使った Unity 公開プロジェクト
- KO_OP が Plastic SCM を使用してプロダクションを加速させた方法
- リリースタイムラインの隠れた障壁となる生産性コスト

Perforce の設定

- Helix Core と ゲームエンジンの設定方法
- Helix Core ドキュメント


Unity®