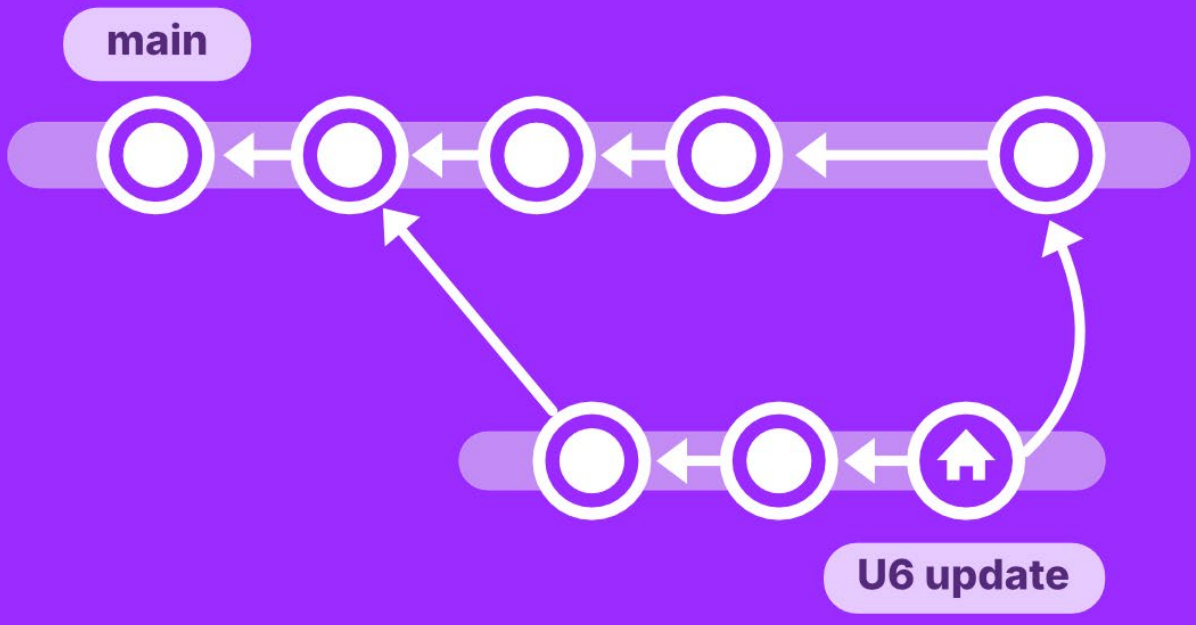
전자책

프로젝트 구성 및 버전 관리 베스트 프랙티스



Unity®

목차

들어가는 말

소프트웨어 개발은 혼자서 작업할 때와 팀에서 함께 작업할 때가 완전히 다릅니다. 모든 팀원이 액세스할 수 있도록 하려면 프로젝트를 어디에 저장해야 할까요? 여러 사람이 동시에 한 파일을 작업하면 어떤 일이 벌어질까요? 프로그래머는 소스 관리와 관련된 개념을 이해하는 경우가 많지만, 아티스트나 기술적 배경이 없는 팀원은 어떨까요? 프로그래머가 지원해 주지 않아도 아티스트나 다른 팀원이 잘못된 행동을 할지 걱정할 필요가 없도록 하려면 어떻게 해야 할까요?

소스 관리 또는 버전 관리는 특히 기술적 배경이 없는 게임 개발자에게 어려운 주제일 수 있으나, 꼭 어렵게만 생각할 필요는 없습니다. 팀이 효과적으로 버전을 관리할 수 있도록 Unity에 통합되는 다양한 툴이 있습니다.

이 가이드에서는 버전 관리의 주요 개념을 설명하고, 사용할 수 있는 여러 VCS(버전 관리 시스템)를 비교하고, Unity Asset Manager나 플레이어 참여 유도 및 분석, 게임 경제, 멀티플레이어 서비스 등 추가적인 Unity DevOps 툴에 대해 소개합니다. 팀이 원활하고 효율적으로 협업할 수 있도록 Unity 프로젝트를 설정할 때 활용할 수 있는 유용한 팁도 제공합니다. 마지막으로 팀에서 성공적으로 작업하기 위한 버전 관리 베스트 프랙티스도 살펴볼 수 있습니다.

소스 관리와 버전 관리

컴퓨팅 초창기에는 모든 소프트웨어 개발이 코드로만 이루어졌습니다. 3D 그래픽스가 발전하면서도 여전히 모든 게 코드로 작성되었습니다. 따라서 '소스 관리(source control)'라는 용어가 프로젝트의 콘텐츠를 관리하는 시스템을 나타내게 되었으며, 이러한 툴을 SCM(소스 코드 관리)이라고 불렀습니다.

현대의 소프트웨어 및 게임 개발에서는 소스 코드 이외에도 다양한 파일을 다루고 있습니다. FBX와 같은 3D 모델 형식 파일이나 텍스처, 머티리얼, 오디오 파일 등이 사용되면서 SCM이 텍스트 파일의 변경 사항만을 처리하는 것으로는 부족해졌습니다. '소스 관리'라는 용어로는 더 이상 우리에게 필요한 작업을 표현하지 못하게 되었으며, 우리가 사용하는 툴을 'VCS(버전 관리 시스템)'라는 더 적절한 용어로 부르게 되었습니다.

두 용어는 여전히 서로 바꿔 사용할 수 있습니다. 하지만 대용량 바이너리 에셋을 다루는 Unity 프로젝트에서는 버전 관리와 VCS라는 용어를 사용하는 게 더 정확하므로 앞으로 이 가이드에서는 버전 관리 및 VCS로 칭할 것입니다.

Unity와 함께 사용하기에 가장 효과적인 3가지 주요 버전 관리 시스템은 Unity Version Control(기존 Plastic SCM)[1], Git 그리고 Perforce Helix Core입니다. 이 가이드에서는 팀 단위로 Unity 프로젝트를 작업할 때 각 시스템의 장단점을 살펴봅니다.

[1] Plastic SCM은 2020년에 Unity 제품군으로 합류되었습니다. 따라서 이 툴은 Unity 에디터에 긴밀하게 통합됩니다.

기본 개념

이 섹션에서는 버전 관리의 핵심 개념들을 알아봅니다. ‘커밋(commit)’ 또는 ‘푸시(push)’에 대해 모르는 경우, 이 섹션을 읽으면 버전 관리에 관한 용어를 더 잘 이해하게 될 것입니다.

버전 관리의 원리

버전 관리를 사용하면 전체 프로젝트의 기록을 관리할 수 있습니다. 작업을 조직적으로 구성하고 팀이 효율적으로 반복 작업(iteration)을 수행할 수도 있습니다.

프로젝트 파일은 저장소 또는 '리포(repo)'라고 하는 공유 데이터베이스에 저장됩니다. 프로젝트를 정기적으로 저장소에 백업하고 문제가 발생하면 프로젝트를 이전 버전으로 되돌릴 수 있습니다.

VCS를 사용하면 여러 가지 개별적인 변경 사항을 적용하고 버전 관리를 위해 단일 그룹으로 커밋할 수도 있습니다. 이 커밋은 프로젝트 타임라인의 한 지점에 머물게 되므로, 이전 버전으로 되돌리면 해당 커밋의 모든 변경 사항이 취소되고 이전 상태로 복원됩니다. 한 커밋에 그룹화된 각 변경 사항을 검토하고 수정하거나 커밋 전체를 실행 취소할 수도 있습니다.

프로젝트의 전체 이력을 확인할 수 있으므로 어떤 변경 사항이 버그를 유발했는지 더 쉽게 파악하고, 이전에 삭제한 기능을 복원하고, 게임 또는 제품 릴리스 간 변경 사항을 간단히 문서화할 수 있습니다.

또한 버전 관리는 보통 클라우드나 분산된 서버에 저장되므로 개발 팀이 어디서든 협업할 수 있습니다. 원격 근무가 보편화되며 이러한 이점의 중요도는 더욱 높아지고 있죠.

버전 관리를 사용하는 이유

앞에서 말한 이유 외에도 버전 관리는 실험적인 변경 사항을 적용할 때 유용합니다. 프로젝트의 로컬 버전에 새 기능을 추가하고 문제가 발생했다면, 변경 사항을 되돌려 아무 문제 없이 작동하는 프로젝트 버전으로 작업하면 됩니다.

실험적인 아이디어로 반복 작업을 진행하는 중에 메인 프로젝트의 주요 문제를 해결하기 위한 지원이 필요한 경우, 버전 관리를 이용하면 나중을 위해 기존 변경 사항을 저장할 수 있습니다. 그런 다음 지원 작업을 위해 로컬 버전을 메인 브랜치로 다시 가져오면 됩니다. 문제가 해결되면 실험 작업을 복원하고 계속 진행할 수 있습니다.

대부분의 버전 관리 시스템은 사용자가 다른 팀원의 작업을 실수로 덮어쓰지 않도록 방지해 줍니다. 작업 내용을 저장소에 커밋할 때는 저장소에서 최신 업데이트를 '풀링(pulling)'해야 합니다. 그러면 다른 팀원이 같은 파일에서 작업하고 있지 않은지 확인할 수 있습니다. 버전 관리에 익숙하지 않은 사용자들은 '병합 충돌(merge conflict)'이라고 하는 상황을 두려워하는 경우가 많습니다. 하지만 이 툴을 잘 이해한다면 병합 충돌은 쉽게 해결할 수 있습니다. UVCS(Unity Version Control)는 스마트 잠금 기능을 통해 잠긴 파일이 모든 브랜치에서 해당 파일의 최신 버전인지 확인하여 병합 충돌의 위험을 줄여 줍니다.

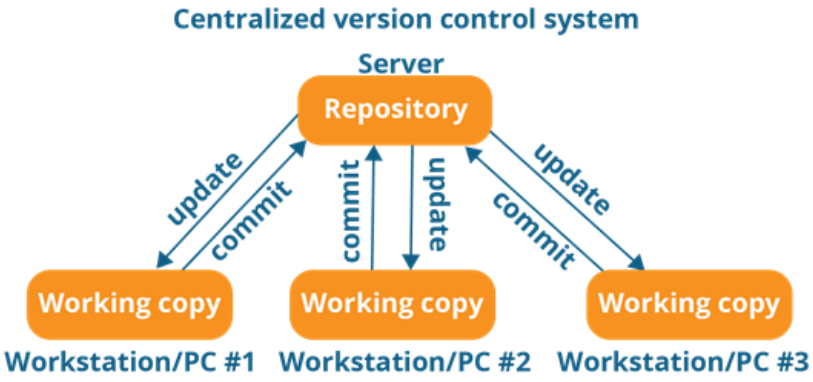
중앙 집중형 버전 관리와 분산형 버전 관리 비교

대체로 버전 관리 시스템은 중앙 집중형과 분산형이라는 두 카테고리 중 하나에 속합니다. 사용하는 버전 관리 시스템에 따라 아래에서 설명하는 용어가 적용되거나 적용되지 않을 수 있으며 아예 다른 의미로 사용되기도 합니다. 두 카테고리의 차이점을 살펴보겠습니다.

중앙 집중형

중앙 집중형 시스템과 분산형 시스템의 첫 번째 주요 차이점은 저장소의 위치입니다. 많은 기업이 자체 소프트웨어를 호스팅하는 서버를 사내에 유지하기 위해 중앙 집중형 시스템을 선택합니다. 이러한 시스템을 선택할 때는 소스 관리 보안이 중요한 요소일 때가 많습니다. 중앙 집중형 시스템도 저장소를 클라우드에서 호스팅할 수 있으므로 반드시 사내 서버를 사용할 필요는 없지만, 이러한 설정은 분산형 시스템에 비해 흔하지 않습니다.

두 접근 방식의 또 다른 주요 차이점은 사용자가 변경 사항을 저장소에 배포하는 방식의 차이입니다. 중앙 집중형 버전 관리가 더 간단한 방식으로 간주되는 경우가 많습니다. 중앙 집중형 저장소로 작업할 때는 변경 사항을 저장소에서 직접 가져오고 저장소로 직접 보냅니다. 이러한 프로세스를 저장소에서 업데이트하고 저장소에 커밋한다고 설명합니다.



이 설정의 단점은 사용자가 작업을 제출하려면 반드시 서버에 연결되어야 한다는 것입니다. 사용자는 충돌을 피하기 위해 파일을 잠그고 수정할 수 있습니다. 이를 파일을 체크아웃한다고 하며, 이렇게 하면 파일을 다시 체크인할 때까지 다른 사람은 변경 사항을 커밋할 수 없습니다.

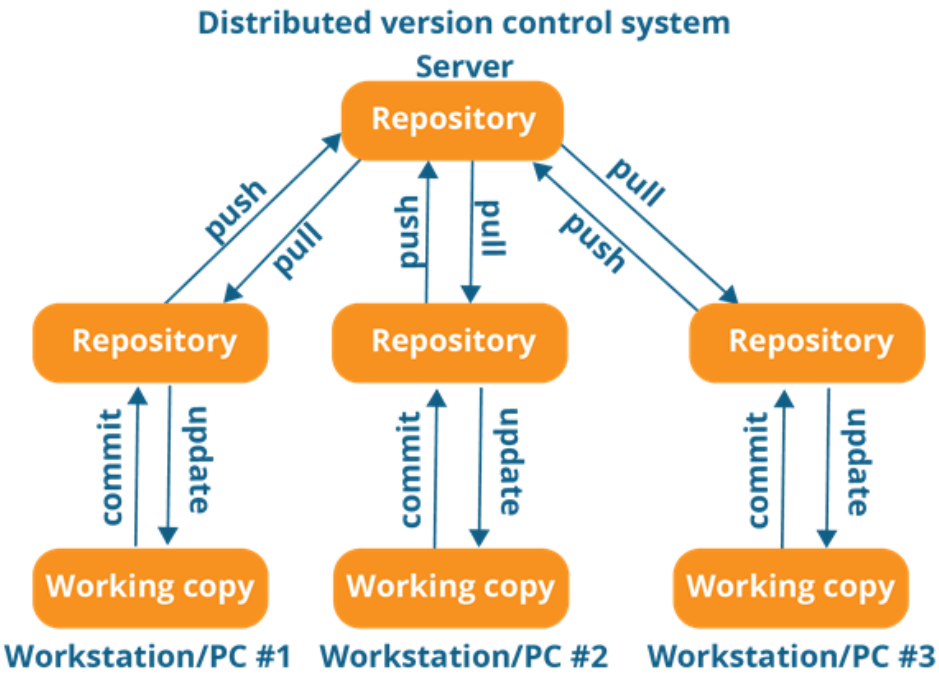
중앙 집중형 워크플로에서는 사용자가 작업 공간에 항상 최신 버전의 프로젝트 파일만 가져올 수 있으며, 서버에 프로젝트의 전체 이력이 저장됩니다.

분산형

분산형 워크플로도 GitHub 같은 한곳의 클라우드 서비스에 저장소가 위치해 있기도 하지만, 사용자는 전체 프로젝트 이력을 작업 공간에 클로닝(clone)합니다. 따라서 사용자는 중앙 서버에 연결할 필요 없이 로컬 복사본에서 작업하고 변경 사항을 빠르게 커밋할 수 있습니다. 다른 사람이 액세스할 수 있도록 변경 사항을 전송하려면 사용자가 변경 사항을 서버에 푸시하고 다른 변경 사항을 풀링해야 합니다. 하지만 중앙 집중형 시스템처럼 항상 최신 파일로 작업할 필요는 없습니다.

이렇게 작업하면 여러 체인지 세트를 모아 하나의 큰 기능을 완성한 다음에 푸시하여 팀원들과 공유할 수 있습니다. 다만 실제로는 조금씩 자주 커밋하는 방식이 권장됩니다. 이 베스트 프랙티스는 가이드 후반부에서 살펴보겠습니다.

일부 분산형 워크플로에서는 파일 잠금 기능도 지원하지만, 병합을 비교적 쉽게 처리할 수 있으므로 흔하게 사용되지 않습니다. 서버에서 최신 변경 사항을 로컬 프로젝트로 풀링하면 다른 팀원과 자신의 변경 사항을 비교하여 변경 사항을 저장소에 푸시하기 전에 충돌이 발생하지 않는지 확인할 수 있습니다.



분산형 방식은 대체로 선호되기는 하지만 몇 가지 단점이 있습니다. 첫째, 전체 프로젝트 이력을 로컬 기기에 가져오면 공간을 크게 차지합니다. 특히 바이너리 파일 유형으로 작업하는 팀에서는 이 문제가 더 심각합니다. Git에는 특정 파일의 이력을 텍스트 포인터로 전환하여 용량을 줄이는 LFS(Large File Storage)라는 시스템이 있습니다. 하지만 다른 파일은 전체 이력이 보관되므로 저장소에 오래되거나 사용되지 않는 테스트 데이터가 쌓일 수 있습니다. 작은 M2 드라이브를 사용하는 스튜디오에서는 오래된 버전이 저장소에서 많은 부분을 차지하게 되면서 드라이브 용량이 부족해질 수 있습니다.

둘째, 개발자가 중앙 서버에 연결되어 있지 않아도 되므로 긴 시간 동안 단절된 상태로 작업할 수 있습니다. 로컬 버전과 메인 저장소 버전의 차이점이 상당히 커진 상태에서 변경 사항을 병합하려 하면 예상보다 많은 작업이 필요할 수 있습니다.

일반적인 워크플로

중앙 집중형 시스템과 분산형 시스템의 워크플로를 단계별로 간단히 나타내면 다음과 같습니다.

중앙 집중형

1. 서버에 변경된 내용으로 작업 복사본을 업데이트합니다.
2. 작업 복사본을 수정합니다.
3. 변경 사항을 중앙 서버에 커밋합니다.

분산형

1. 원격 저장소의 변경 사항을 로컬 저장소로 풀링합니다.
2. 작업 복사본을 수정합니다.
3. 변경 사항을 커밋합니다.
4. 2단계 및 3단계를 원하는 만큼 반복합니다.
5. 모든 커밋을 원격 저장소에 푸시합니다.

이 가이드에서는 UVCS, Git, Perforce Helix Core라는 3가지 주요 버전 관리 시스템을 살펴봅니다. 각 VCS에 대한 내용은 '버전 관리 시스템' 섹션에서 자세히 알아볼 예정이므로 여기서는 각 VCS 및 지원하는 워크플로에 관해 간략하게 소개합니다.

— Unity Version Control 또는 UVCS(중앙 집중형 및 분산형): UVCS는 프로그래머와 아티스트를 모두 지원하는 고유의 인터페이스를 갖춘 유연한 버전 관리 시스템입니다. 대규모 저장소와 바이너리 파일 처리에 탁월하며, 파일 기반 솔루션이자 체인지 세트 기반 솔루션이므로 프로젝트 빌드 전체가 아닌 작업 중인 특정 파일만 다운로드할 수 있습니다.

— Git(분산형): 널리 사용되는 대표적인 버전 관리 시스템 중 하나입니다. Git은 유연한 무료 오픈 소스 시스템입니다. Git 플랫폼은 커맨드 라인 전용 툴입니다. 하지만 다양한 전용 GUI가 개발되었기 때문에 더 손쉽게 Git 시스템을 사용할 수 있습니다.

Git과 GitHub를 혼동하는 경우가 자주 있습니다. GitHub는 Git 저장소를 위한 호스팅 서비스이며 GitHub가 없어도 Git을 사용할 수 있습니다. 이 가이드를 읽고 있는 숙련된 개발자라면 누구나 알고 있듯이, GitHub는 일부 기능이 제한되지만 무료로 사용할 수 있어 매우 인기 있는 서비스이며 커스텀 서버를 설정할 필요가 없습니다.

— Perforce Helix Core(중앙 집중형): 엔터프라이즈급 버전 관리 시스템으로, 자체 서버에서 중앙 집중형 저장소를 호스팅하는 대형 게임 스튜디오에서 많이 사용합니다.

주요 용어

다음은 VCS의 주요 기능 및 프로세스에 관한 용어입니다.

용어	설명
저장소 (repository/repo)	프로젝트의 모든 변경 사항 및 수정 사항이 담긴 데이터베이스입니다. 사내 또는 클라우드 서버에 저장되며 프로젝트의 전체 이력이 보관됩니다.
작업 복사본 (working copy)	프로젝트의 로컬 버전입니다. 체크아웃(checkout) 또는 작업 공간(workspace)이라고도 합니다. 작업 복사본에 변경 사항을 적용하고 결과에 만족하면 저장소에 커밋합니다.
커밋(commit) /체크인(check in)	커밋은 파일 변경 사항을 인코딩합니다. 중앙 집중형 워크플로에서는 이러한 변경 사항을 서버로 전송하며, 이러한 동작을 보통 체크인이라 합니다. 분산형 워크플로에서는 커밋을 통해 체인지 세트에 변경 사항이 추가되며 나중에 서버로 푸시해야 합니다.
풀링(pulling) /업데이트(update) /체크아웃(check out)	풀링 또는 업데이트는 서버에 있는 최신 변경 사항을 가져오는 프로세스입니다. 중앙 집중형 워크플로에서는 일반적으로 체크아웃이라는 용어가 사용됩니다.
잠금(locking)	파일을 잠그면 다른 사람이 해당 파일을 수정할 수 없습니다. 말하자면 내가 이 파일을 작업하고 있으므로 다른 사용자가 변경하지 못하게 하라고 서버에 요청하는 것입니다. 분산형 워크플로에서는 일반적으로 잠금 기능을 지원하지 않습니다.
클로닝(clone)	분산형 워크플로에서 저장소를 클로닝하는 것은 처음에 프로젝트의 복사본과 전체 이력을 로컬 기기로 가져오는 것입니다.
태그(tag)	태그는 커밋에 추가할 수 있는 특별한 메모입니다. 빌드가 만들어진 시점을 표시하기 위해 태그를 사용하는 경우가 많습니다.
브랜치(branch)	브랜치는 코드라인에서 새로 생성된 복사본으로 각 브랜치에서 병렬로 작업할 수 있습니다. 브랜치를 사용하면 메인 개발 라인에 영향을 주지 않으며 격리된 상태에서 프로젝트의 일부분(예: 새로운 기능)을 작업할 수 있습니다.

병합(merge)	병합은 브랜치에서 작업을 완료한 후 메인 개발 라인으로 병합해야 하거나 두 사람이 파일을 동시에 변경했을 때 수행하는 작업입니다. 두 체인지 세트를 비교한 후 병합하여 새로운 작업 복사본을 만듭니다. 병합은 대부분 자동으로 처리됩니다.
충돌(conflict)	충돌은 병합을 자동으로 처리할 수 없을 때 발생하는 상황입니다. 주로 두 사람이 같은 코드 줄이나 바이너리 파일을 변경하면 발생합니다. 코드 충돌은 텍스트를 비교하여 어떤 변경 사항을 수락할지 선택하거나 두 변경 사항을 특정 방식으로 통합하여 해결할 수 있습니다. Unity 씬이나 프리팹 같은 바이너리 파일의 경우 충돌을 병합하기가 훨씬 더 까다롭습니다. 하지만 때로는 다른 작업자와 대화하여 어떤 변경 사항을 적용하는 것이 가장 타당한지 결정하는 것이 가장 쉬운 해결책입니다.
풀 리퀘스트 (pull request)	브랜치에서 작업을 완료한 후에는 풀 리퀘스트를 생성하는 것이 바람직합니다. 풀 리퀘스트는 브랜치에서 작업을 완료했으며 메인 개발 라인에 병합할 준비가 되었다고 팀원들에게 알리는 신호입니다. 이 시스템을 사용하면 팀 리드나 시니어 개발자가 변경 사항을 수락하여 메인 브랜치에 가져오기 전에 검토를 진행할 수 있습니다.
헤드(head)	헤드는 작업 복사본의 최신 커밋을 가리킵니다.
되돌리기(reset/revert)	사용하는 VCS 유형에 따라 reset 또는 revert 명령어를 사용하여 모든 로컬 변경 사항을 버리고 헤드 상태로 되돌릴 수 있습니다.
인덱스(index)	Git 인덱스는 작업 복사본의 모든 현재 변경 사항을 나타내는 파일입니다. 이러한 변경 사항은 다음 커밋에 추가할 변경 사항을 선택하는 스테이징 영역(staging area)이라는 곳에 저장됩니다.
Git 스태시(Git stash)	아직 변경 사항을 커밋할 준비가 되지 않았지만 다른 작업으로 전환해야 하는 경우, 스태시를 통해 임시 파일에 변경 사항을 저장해 두고 작업 복사본을 헤드 상태로 되돌릴 수 있습니다.

Unity 프로젝트 구성의 베스트 프랙티스

어떤 VCS를 선택하더라도 Unity에서 작업할 때 버전 관리 워크플로를 간소화하는 데 도움이 되는 일반적인 베스트 프랙티스가 있습니다. 먼저 효과적인 팀 협업에 도움이 되는 몇 가지 방법을 살펴보겠습니다.

프로젝트 구성

폴더 구조

바람직한 프로젝트 구성을 위한 하나의 정답은 없지만 일반적으로 권장되는 내용은 다음과 같습니다.

— 팀 차원에서 명명 규칙과 폴더 구조를 정의하고 문서화합니다. 스타일 가이드나 프로젝트 템플릿을 만들면 파일을 더 쉽게 찾고 구성할 수 있습니다. 팀에 맞는 규칙과 구조를 정하고 모두가 따르게 합니다.

— 명명 규칙에 일관성을 유지합니다. 처음 선택한 스타일 가이드나 템플릿에서 벗어나지 않아야 합니다. 명명 규칙을 수정해야 하는 경우 해당하는 모든 에셋을 한 번에 파싱하고 이름을 변경하세요. 변경해야 할 파일이 많다면 스크립트를 사용해 업데이트를 자동화하는 편이 좋습니다.

— 파일이나 폴더 이름에 공백을 사용하면 안 됩니다. 공백 대신 낙타 대문자(camelCase)를 사용하세요.

— 테스트 영역과 샌드박스 영역을 분리합니다. 정식 제작용이 아닌 씬과 실험용 폴더를 별도로 만드는 것이 좋습니다. 사용자 이름으로 하위 폴더를 만들면 작업 영역을 팀원별로 나눌 수 있습니다.

— 루트 수준에 폴더를 추가하는 일은 지양합니다. 콘텐츠 파일은 일반적으로 Assets 폴더에 저장합니다. 꼭 필요한 경우가 아니라면 프로젝트의 루트 수준에 폴더를 추가로 생성하지 않도록 하세요.

— 내부 에셋을 타사 에셋과 분리하여 저장합니다. 에셋 스토어의 에셋이나 다른 플러그인은 프로젝트 구조가 다를 확률이 높습니다. 에셋을 따로 보관하세요.

프로젝트에서 타사 에셋이나 플러그인을 수정하는 경우, 플러그인의 최신 업데이트가 필요할 때 버전 관리가 매우 유용할 것입니다. 업데이트를 임포트한 후 차이점을 확인하여 기존에 수정한 부분이 덮어쓰였는지 보고 다시 구현할 수 있습니다.

정해진 폴더 구조는 없지만 다음 몇 가지 예시처럼 Unity 프로젝트를 구성할 수 있습니다. 에셋 유형에 따른 프로젝트 분할을 기반으로 한 구조입니다. 에셋 유형 매뉴얼 페이지에서 가장 일반적으로 사용되는 에셋에 대해 자세히 알아볼 수 있습니다. 폴더 구조를 구성하는 방법의 예시로 템플릿이나 Learn 프로젝트를 사용할 수 있습니다. 반드시 이러한 폴더 이름을 사용할 필요는 없지만 좋은 시작점으로 참고할 수 있습니다.

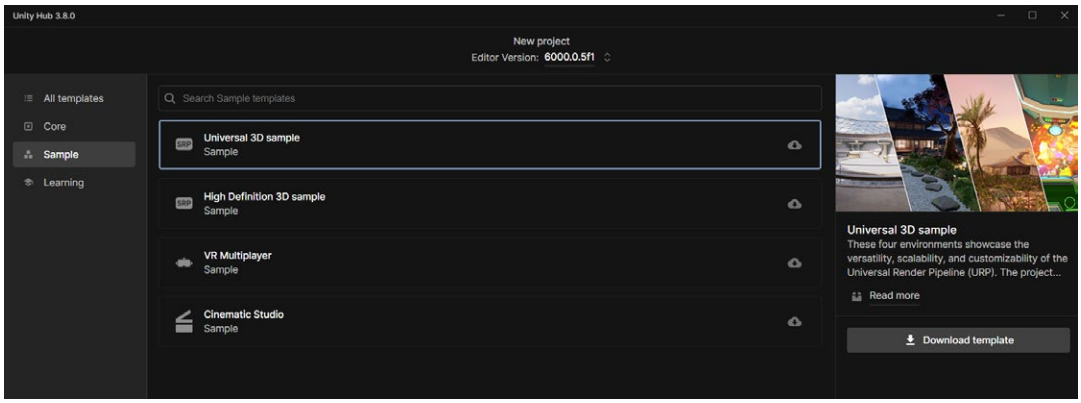
예시 1

```
Assets
+---Art
| +---Materials
| +---Models
| +---Textures
+---Audio
| +---Music
| \---Sound
+---Code
| +---Scripts  # C# 스크립트
| \---Shaders  # 셰이더 파일 및 Shader Graph
+---Docs   # 위키, 컨셉 아트, 마케팅 자료
+---Level   # Unity 게임 디자인 관련 모든 파일
| +---Prefabs
| +---Scenes
| \---UI
```

예시 2

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

Unity Hub에서 템플릿이나 스타터 프로젝트를 다운로드하면 아래 이미지와 같이 에셋 유형에 따라 프로젝트 하위 폴더가 분할되어 있는 모습을 볼 수 있습니다.

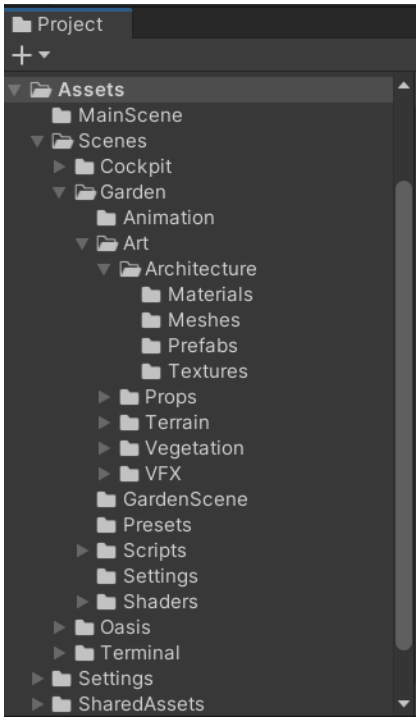


Unity Hub에서 다운로드할 수 있는 템플릿

선택한 템플릿에 따라 여러 개의 공통 에셋을 나타내는 하위 폴더가 표시됩니다. 다음은 유형별로 구성하는 한 가지 방법입니다.

Animations	Animations 폴더에는 애니메이션 모션 클립과 컨트롤러 파일을 보관합니다. 게임 내 시네마틱을 위한 타임라인 에셋이나 절차적 애니메이션을 위한 리깅(rigging) 정보도 포함될 수 있습니다.
Audio	사운드 에셋에는 오디오 클립뿐만 아니라 효과와 음악을 블렌딩하는 데 사용되는 믹서도 포함됩니다.
Editor	Unity 에디터에서 사용하기 위해 생성되었으나 타겟 빌드에는 표시되지 않는 스크립팅된 툴을 보관합니다.
Fonts	게임에서 사용하는 폰트를 보관하는 폴더입니다.
Materials	머티리얼 에셋은 표면 셰이딩 프로퍼티를 나타냅니다.

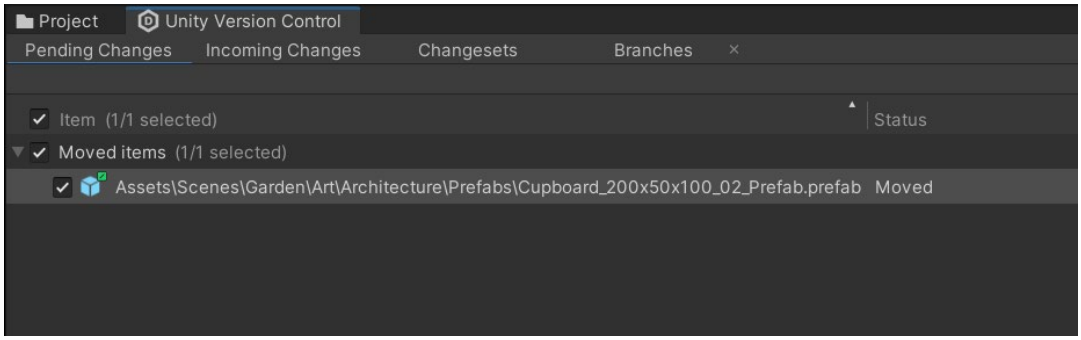
Meshes	외부 DCC(디지털 콘텐츠 제작) 애플리케이션에서 제작한 모델을 이 폴더에 보관합니다.
Particles	빌트인 파티클 시스템이나 Visual Effect Graph로 제작한 Unity 파티클 시뮬레이션을 보관합니다.
Prefabs	미리 제작된 컴포넌트가 있으며 재사용할 수 있는 게임 오브젝트를 보관합니다.
Scripts	게임플레이를 위해 사용자가 개발한 모든 코드가 이 폴더에 표시됩니다.
Scenes	Unity는 프로젝트의 소규모 기능 단위를 씬 에셋에 저장합니다. 이는 대체로 게임 레벨이나 레벨의 일부에 해당되는 경우가 많습니다.
Settings	HDRP(고해상도 렌더 파이프라인) 및 URP(유니버설 렌더 파이프라인) 같은 렌더 파이프라인의 설정을 여기에 저장합니다.
Shaders	이 프로그램은 그래픽스 파이프라인의 일부로 GPU에서 실행됩니다.
Textures	이미지 파일은 머티리얼과 표면 처리를 위한 텍스처 파일, 사용자 인터페이스를 위한 UI 오버레이 요소, 조명 정보를 저장하기 위한 라이트맵 등으로 구성될 수 있습니다.
ThirdParty	에셋 스토어 등 외부 소스에서 얻은 에셋이 있는 경우, 해당 에셋을 프로젝트의 나머지 에셋과 분리하여 여기에 보관할 수 있습니다. 외부 에셋을 따로 보관하면 타사 에셋과 스크립트를 더 쉽게 업데이트할 수 있습니다. 타사 에셋에는 변경할 수 없는 세트 구조가 있을 수 있습니다.
UI	UI 툴킷을 사용하는 경우 UXML 및 USS 파일이 여기에 저장됩니다.



여러 에셋 폴더가 포함된 URP 템플릿의 샘플 씬

프로젝트 구조를 처음부터 잘 정의하면 나중에 버전 관리 문제를 방지할 수 있습니다. 한 폴더에서 다른 폴더로 에셋을 이동할 경우, 많은 VCS에서는 파일을 옮긴 것이 아니라 파일을 삭제하고 새로운 파일을 추가한 것으로 인식할 것입니다. 그러면 원본 파일의 이력이 사라집니다.

UVCS는 Unity 내에서 파일 이동을 처리하고 이동한 모든 파일의 이력을 보존할 수 있습니다. 하지만 Unity 에디터에서 파일을 옮겨야만 에셋 파일과 함께 .meta 파일도 이동한다는 점을 기억해야 합니다.

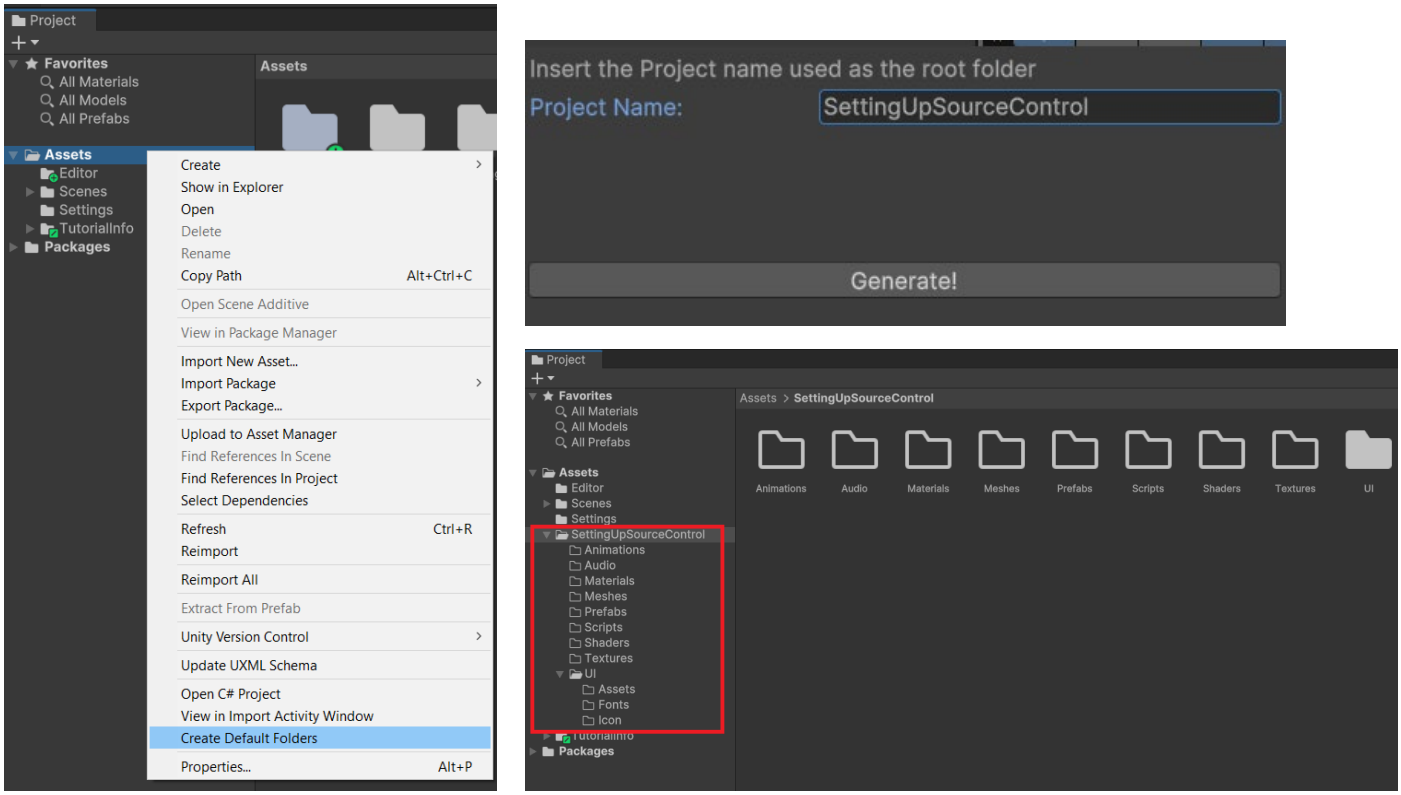


파일 이동 추적

프로젝트의 폴더 구조를 결정했으면 에디터 스크립트를 사용하여 템플릿을 재사용하고 앞으로 모든 프로젝트를 같은 폴더 구조로 만듭니다. Editor 폴더에 아래 스크립트를 저장하면 Assets 폴더 안에 'PROJECT_NAME' 변수를 이름으로 하는 루트 폴더가 스크립트에 의해 생성됩니다. 이렇게 하면 본인의 작업을 타사 패키지와 분리할 수 있습니다.

```
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(“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();
   }
  }
}
```



메뉴에서 Assets > Create Default Folders를 클릭합니다.

빈 폴더

위 이미지에서 보이는 것과 같은 빈 폴더는 버전 관리에서 문제가 될 수 있으므로 오직 필요한 폴더만 생성해야 합니다. Git과 Perforce는 기본적으로 빈 폴더를 무시합니다. 이러한 프로젝트 폴더가 설정되고 해당 폴더를 커밋하려 해도 폴더 안에 파일을 넣기 전에는 커밋할 수 없습니다.

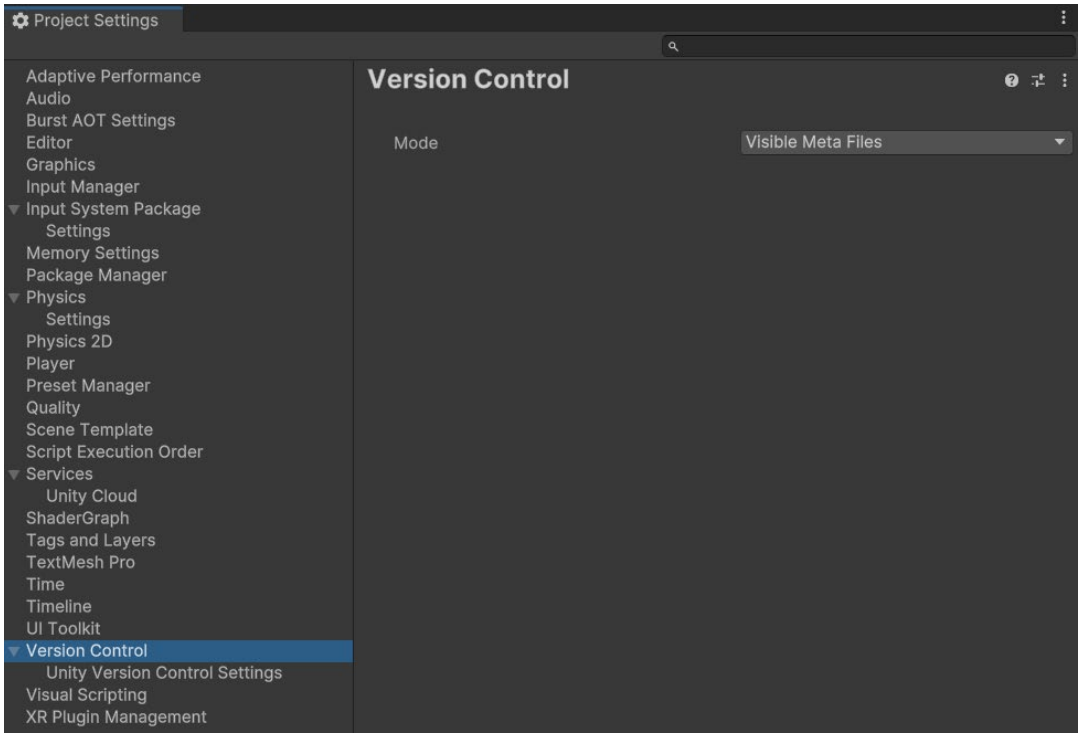
참고: 일반적인 해결책은 빈 폴더에 '.keep' 파일을 두는 것입니다. 이렇게만 해도 폴더를 저장소에 커밋할 수 있습니다.

UVCS는 빈 폴더를 처리할 수 있습니다. 디렉토리도 엔티티로 취급되며 관련된 버전 이력을 가집니다.

Unity에서 작업할 때는 이러한 점에 주의해야 합니다. Unity는 폴더를 포함한 프로젝트 내 모든 파일에 .meta 파일을 생성합니다. Git과 Perforce에서는 빈 폴더에 .meta 파일을 쉽게 커밋할 수 있지만 폴더 자체는 버전 관리에 포함되지 않습니다. 다른 사용자가 최신 변경 사항을 가져올 때는 기기에 존재하지 않는 폴더의 .meta 파일이 있으므로 Unity가 해당 .meta 파일을 삭제할 것입니다. UVCS는 빈 폴더도 버전 관리에 포함하여 이 문제를 방지합니다.

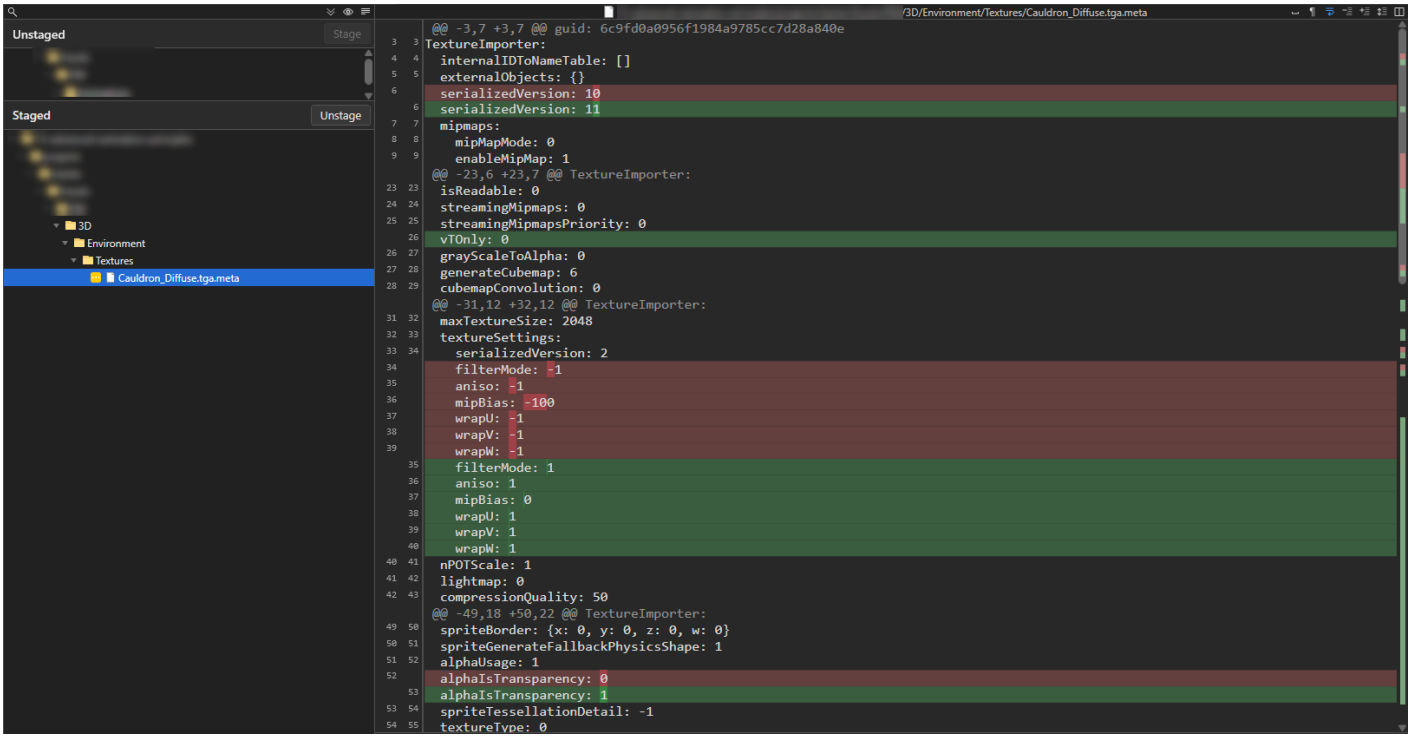
.meta 파일

Unity는 프로젝트 내 다른 모든 파일에 .meta 파일을 생성하며, 일반적으로 버전 관리에 자동 생성된 파일은 포함하지 않는 것이 권장되지만 .meta 파일의 경우 조금 다릅니다. UVCS 또는 Perforce 모드를 사용하지 않는다면 Version Control 창에 Visible Meta Files 모드가 켜져 있을 것입니다.



Git을 사용할 때는 Visible Meta Files 모드를 활성화합니다.

.meta 파일은 자동으로 생성되지만 이 파일에는 연관된 파일에 대한 많은 정보가 저장됩니다. 텍스처, 메시, 오디오 클립 등 임포트 설정이 있는 에셋이 주로 여기에 해당합니다. 이러한 파일의 임포트 설정을 변경하면 에셋 파일이 아닌 .meta 파일에 변경 사항이 작성됩니다. 따라서 .meta 파일을 저장소에 커밋해야만 모든 작업자가 같은 파일 설정을 사용해 작업할 수 있습니다.



파일의 임포트 설정을 변경했을 때 .meta 파일이 수정된 모습

명명 규칙

기준을 정해야 하는 요소는 프로젝트 폴더 구조 말고도 더 있습니다. 프로젝트 폴더의 씬 내 게임 오브젝트나 프리팹에 대한 명명 규칙을 세우면 다른 팀원의 파일을 이해하고 작업하기 쉬워집니다.

게임 오브젝트의 이름을 정할 때 반드시 따라야 하는 명명 규칙은 없지만 다음을 따르는 것이 좋습니다.

기준	예시
약어가 아니며 이해하기 쉬운 이름을 사용합니다. 몇 달이 지나도 기억할 수 있는 이름을 사용하세요. 해당 표기법을 다른 사람도 이해할 수 있는지 고려하면서 쉽게 발음하고 기억할 수 있는 이름을 선택해야 합니다. 따라서 약어를 사용하는 것은 권장되지 않습니다.	largeButton, LargeButton, leftButton 부적절한 예: lButton
낙타 대문자나 파스칼 대문자(PascalCase)를 사용합니다. 오브젝트 이름에는 공백을 사용하지 않도록 하세요. 낙타 대문자나 파스칼 대문자는 가독성을 높여 주며, 이 연구에 따르면 타이핑 정확도도 높여 줍니다.	OutOfMemoryException, dateTimeFormat 부적절한 예: Outofmemoryexception, datetimeformat

과도한 밑줄 또는 하이픈 사용을 지양합니다. 일반적으로는 밑줄과 하이픈을 사용하지 않는 편이 좋지만, 특정 상황에서는 밑줄과 하이픈을 사용해야 할 수도 있습니다. 이름 앞에 밑줄을 사용하면 알파벳순으로 정렬했을 때 가장 먼저 표시됩니다. 또는 밑줄을 사용하여 특정 오브젝트의 배리언트를 나타낼 수도 있습니다.	활성 상태: EnterButton_Active, EnterButton_ Inactive 텍스처 맵: Foliage_Diffuse, Foliage_Normalmap 디테일 수준(LOD): Building_LOD1, Building_LOD0
숫자를 마지막에 붙여 순서를 나타냅니다. 목록의 일부가 아닌 경우에는 숫자를 마지막에 붙이지 않습니다.	경로에서 노드 이름 지정: Node0, Node1, Node2 등
디자인 문서의 명명 규칙을 따릅니다.	디자인 문서에서 HighSpellTower 또는 RedDragonLair 등의 이름을 사용한다면 정확히 똑같은 이름을 사용해야 합니다.

팀 전체가 합의한 일관된 스타일을 사용해야만 더 깔끔하고 읽기 쉬우며 유연한 스케일링이 가능한 프로젝트를 만들 수 있습니다. 유니티 전자책 C# 스타일 가이드 만들기: 깔끔하고 확장 가능한 코드를 쓰는 방법에서 명명 규칙, 포맷 지정, 클래스, 메서드, 주석 등에 관한 팁과 베스트 프랙티스를 살펴볼 수 있습니다. 이 가이드는 전반적으로 Microsoft C# 스타일 기준 2를 준수하며 Unity에 관한 세부적인 기준을 제공하지만 단 하나의 정답이 있는 것은 아닙니다. Google C# 2 가이드 또한 명명 규칙, 포맷 지정, 주석 규칙에 관한 가이드라인을 정의할 때 유용하게 활용할 수 있는 리소스입니다.

워크플로 최적화

Assets 폴더에 에셋을 보관하는 방법과 위치 외에도, 특히 버전 관리를 사용하며 워크플로 속도를 높이기 위해 디자인 및 개발과 관련해 선택할 수 있는 사항이 몇 가지 있습니다.

에셋 분할

Unity에서 하나로 이루어진 대형 씬은 협업에 적합한 편이 아닙니다. 여러 개의 소규모 씬으로 레벨을 분할하여 작업하면 아티스트와 디자이너가 충돌할 위험이 최소화되며, 하나의 레벨에서 보다 원활하게 협업할 수 있습니다.

런타임 시 프로젝트에서 LoadSceneMode.Additive 파라미터 모드를 전달하는 SceneManager.LoadSceneAsync를 사용하면 씬을 부가적으로 로드할 수 있습니다.

또한 가능하다면 작업을 프리팹으로 분할하는 것이 좋습니다. 나중에 무언가 변경해야 하는 경우 씬을 수정하는 대신 프리팹을 수정하면 씬에서 작업하는 다른 팀원과 충돌을 피할 수 있습니다. 버전 관리에서 차이점을 확인할 때 프리팹 변경 사항이 더 읽기 쉬운 경우가 많습니다.

만약 씬 충돌이 발생하는 경우 Unity에서 제공하는 빌트인 YAML(사람이 읽을 수 있는 데이터 직렬화 언어) 툴이 특히 씬과 프리팹을 병합하는 데 도움이 됩니다. 자세한 내용은 Unity 기술 자료의 스마트 병합 페이지를 참조하세요.

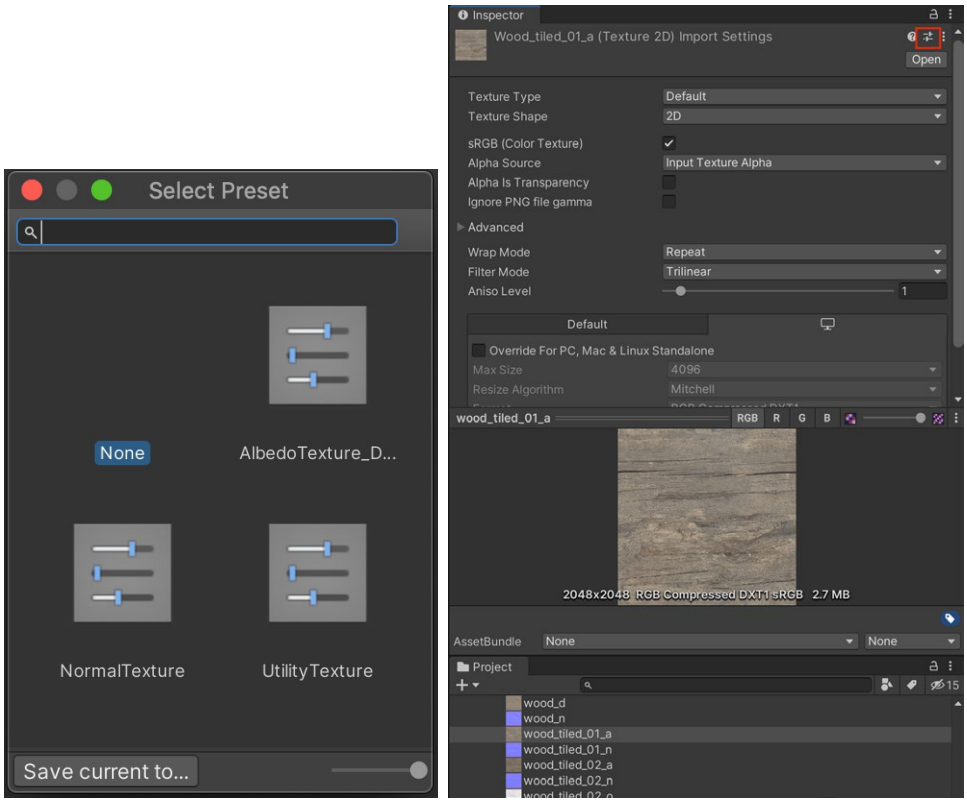
프리셋

Unity 프리셋은 사전 정의된 설정으로 프로젝트 내의 여러 컴포넌트, 에셋, 툴에 특정 설정을 저장하고 재사용하는 데 활용됩니다. 프리셋을 만들면 컴포넌트나 에셋의 설정을 복사하여 에셋으로 저장하고, 나중에 동일한 설정을 다른 항목에 적용할 수 있습니다.

프리셋을 사용하여 표준 설정을 강제하거나 새로운 에셋에 원하는 기본 설정을 적용할 수 있습니다. 프리셋을 통해 팀 전체가 일관된 표준 설정을 사용할 수 있으므로, 프로젝트 결과에 영향을 미칠 수 있는 흔한 설정 오류를 미연에 방지합니다.

프리셋 아이콘(빨간 상자 표시)

컴포넌트의 오른쪽 상단에 있는 프리셋 아이콘을 클릭합니다. **Save current to...**를 클릭하여 프리셋을 에셋으로 저장한 다음, 사용 가능한 프리셋을 클릭하여 일련의 값을 로드합니다.



2D 텍스처의 용도(알베도, 일반, 유틸리티)에 따라 다른 임포트 설정을 포함하는 프리셋 예시

프리셋을 활용하는 추가 팁은 다음과 같습니다.

— **기본값으로 게임 오브젝트 생성:** 계층 구조에 프리셋 에셋을 끌어다 놓으면 해당 컴포넌트가 프리셋 값으로 미리 채워진 새로운 게임 오브젝트를 생성할 수 있습니다.

— **특정 유형과 프리셋 연결:** Preset Manager(**Project Settings > Preset Manager**)에서 유형당 하나 이상의 프리셋을 지정하고 새로운 컴포넌트를 만들면 지정된 프리셋 값이 기본값으로 설정됩니다.

  — 고급 팁: 유형별로 여러 개의 프리셋을 만들고 필터를 이용하여 이름을 기준으로 올바른 프리셋에 연결할 수 있습니다.

— **관리자 설정 저장 및 불러오기:** Manager 창에 프리셋을 사용하여 설정을 재사용할 수 있습니다. 예를 들어 동일한 태그, 레이어, 물리 설정을 다시 적용하려는 경우 프리셋을 사용하면 다음 프로젝트 설정에 필요한 시간을 단축할 수 있습니다.

코드 기준

코딩 기준도 팀 작업의 일관성을 높이고 개발자가 프로젝트의 작업 영역을 전환하는 데 도움을 줍니다. 코딩 기준에서도 정해진 규칙은 없습니다. 팀에 가장 적합한 규칙을 결정하면 되지만 일단 결정된 규칙은 반드시 준수해야 합니다.

예를 들어 네임스페이스는 코드를 더 효과적으로 정리하는 데 도움이 될 수 있습니다. 네임스페이스를 통해 프로젝트에서 모듈을 분리하고 클래스명이 중복될 수 있는 타사 에셋과의 충돌을 피할 수 있죠.

코드에서 네임스페이스를 사용하면 네임스페이스를 기준으로 폴더 구조를 나눠 더 효율적으로 정리할 수 있습니다.

표준 헤더를 사용하는 방식도 권장됩니다. 코드 템플릿에 표준 헤더를 추가하면 클래스의 용도, 작성 날짜, 심지어 작성자까지 문서화하는 데 도움이 됩니다. 버전 관리를 사용하더라도 이 모든 정보는 프로젝트 이력이 길어지는 과정에서 손실되기 쉽습니다.

Unity에는 프로젝트에서 새 MonoBehaviour를 생성할 때마다 읽는 템플릿 스크립트가 있습니다. 새 스크립트나 셰이더가 생성될 때마다 Unity는 다음 위치에 저장된 템플릿을 사용합니다. **%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

기본 MonoBehaviour 템플릿은 다음과 같습니다. **81-C# Script-NewBehaviourScript.cs.txt**

셰이더, 기타 동작 스크립트, 어셈블리 정의를 위한 템플릿도 있습니다.

프로젝트 전용 스크립트 템플릿을 만들려면 Assets/ScriptTemplates 폴더를 생성한 다음 스크립트 템플릿을 이 폴더에 복사하고 기본값을 오버라이드하면 됩니다.

모든 프로젝트를 대상으로 하는 기본 스크립트 템플릿도 직접 수정이 가능하지만, 수정 전에 원본을 백업해야 합니다. Unity는 버전마다 템플릿 폴더가 다르므로 새 버전으로 업데이트하면 템플릿을 다시 교체해야 합니다.

81-C# Script-NewBehaviourScript.cs.txt의 원본 파일은 다음과 같습니다.

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

#ROOTNAMESPACEBEGIN#
public class #SCRIPTNAME# : MonoBehaviour
{
 // 첫 번째 프레임 업데이트 전에 Start가 호출됨
 void Start()
 {
  #NOTRIM#
 }

 // 프레임마다 Update가 한 번 호출됨
 void Update()
 {
  #NOTRIM#
 }
}
#ROOTNAMESPACEEND#
```

다음 두 가지 키워드를 기억해 두는 것이 좋습니다.

— **#SCRIPTNAME#**은 입력한 파일 이름 또는 기본 파일 이름(예: NewBehaviourScript)을 나타냅니다.

— **#NOTRIM#**은 괄호 안에 공백 줄이 포함되도록 합니다.

자체 키워드를 사용한 후에 OnWillCreateAsset 메서드를 구현하는 에디터 스크립트로 해당 키워드를 대체해도 됩니다.

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

자체 키워드를 사용한 후에 OnWillCreateAsset 메서드를 구현하는 에디터 스크립트로 해당 키워드를 대체해도 됩니다.

UI 툴킷 명명 규칙

UI 툴킷은 UXML 코드를 사용합니다. 표준 웹 기술에서 영감을 받은 UI 툴킷은 대시 표기법 또는 LISP 표기법이라고도 하는 케밥 표기법(kebab-case)을 클래스 이름에 사용하며, 이는 CSS 관련 스타일 시스템에서 사용되는 규칙과 동일합니다.

HTML 및 UXML에서는 일반적으로 클래스와 마찬가지로 ID 이름에 케밥 표기법(대시로 연결한 소문자)을 사용합니다. 따라서 my-element-id 같은 이름이 더 표준에 부합하는 이름입니다.

하지만 ID 이름은 해당 요소의 용도를 쉽게 알 수 있도록 구체적이고 충분한 설명을 포함하는 것이 좋습니다. 이를테면 submit-button, main-nav-container, logo-image가 적절한 이름입니다.

	권장	권장하지 않음
클래스	.menu-bar-blue	ButtonBlue
시각적 요소 ID	main-nav-image	

UXML의 권장 명명 규칙

케밥 표기법을 사용하는 주된 이유는 다음과 같습니다.

- CSS 선택자는 대소문자를 구분하지 않으므로 낙타 대문자(예: .buttonBlue)나 파스칼 대문자(예: .ButtonBlue)를 사용하면 혼란을 초래할 수 있습니다.
- HTML 표준에서는 속성 이름을 .buttonBlue 또는 .ButtonBlue 대신 모두 소문자로 된 .button-blue를 사용할 것을 권장합니다.
- 여러 단어가 사용될 때는 케밥 표기법이 읽고 쓰기 쉽습니다.

어떤 명명 규칙을 선택하든 중요한 것은 코드베이스 전반에서 일관성을 유지하는 것입니다. 모든 유형의 스크립트에서 명명 규칙을 일관되게 적용하면 팀원 모두 쉽게 이해하고 수정할 수 있는 깔끔한 코드를 유지할 수 있습니다.

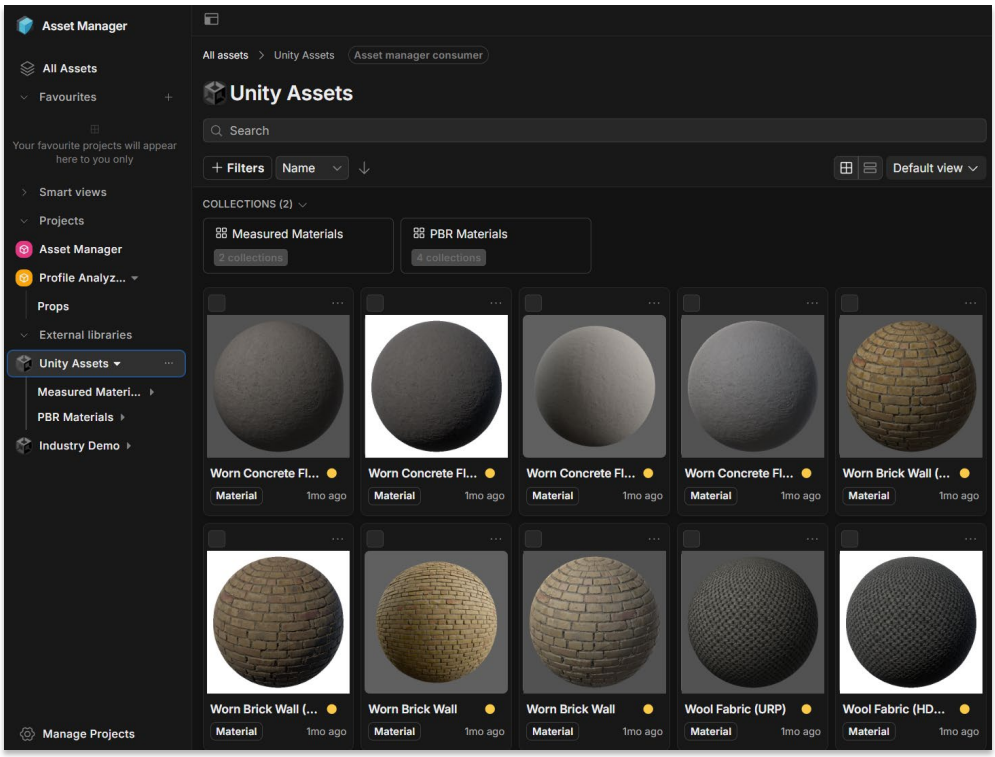
프로젝트 구성을 위한 서비스

Asset Manager

Unity Asset Manager는 유니티의 확장 가능한 클라우드 기반 DAM(디지털 에셋 관리) 솔루션으로, 조직 전반의 콘텐츠 검색 용이성, 재사용성, ROI를 높입니다. Asset Manager를 사용하면 클라우드 스토리지를 사용하는 Unity 프로젝트에서 콘텐츠를 인덱싱할 수 있으므로, 팀원 모두가 장소에 관계없이 손쉽게 에셋을 검색할 수 있습니다. 주요 기능은 다음과 같습니다.

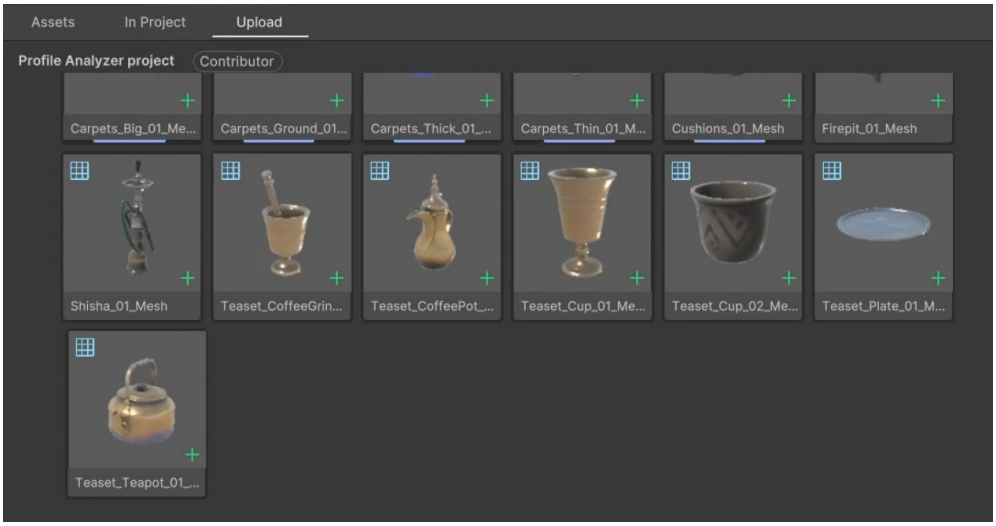
- 향상된 에셋 관리
- 직관적인 탐색 및 검색
- 제작 툴과 근본적으로 통합
- 라이프사이클 관리
- 역할 기반 권한
- 유연성 및 확장성

Unity Cloud에 로그인하면 대시보드에서 무료 머티리얼 및 텍스처와 데모 모음집을 확인할 수 있습니다. **Unity Assets** 하위 섹션을 열고 이 모음집을 프로젝트로 다운로드해 보세요.



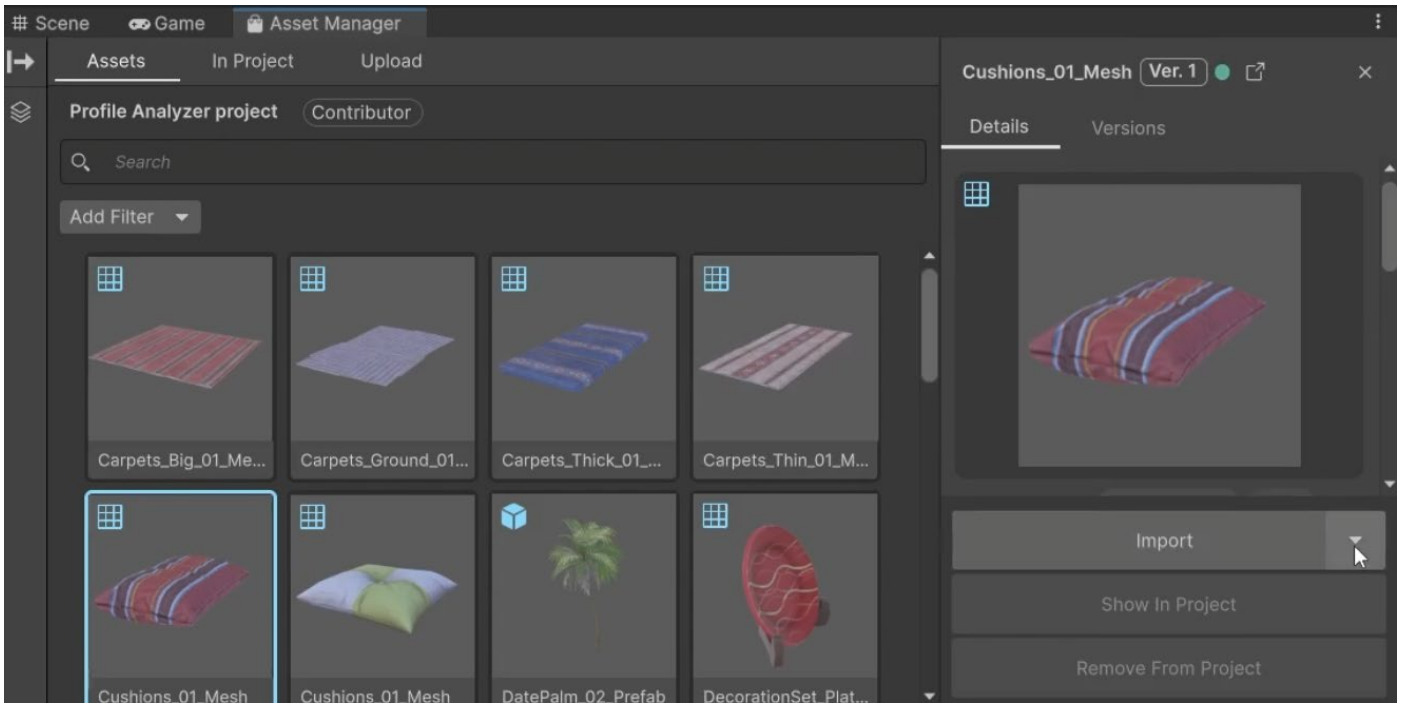
Unity Cloud의 Unity Assets 섹션에서 확인할 수 있는 무료 머티리얼 및 텍스처

Unity에서는 현재 프로젝트의 파일을 **Asset Manager** 창으로 드래그하여 클라우드에 업로드할 수 있습니다. 업로드한 파일은 Asset Manager를 사용하는 다른 프로젝트에서 사용하거나 UVCS를 통해 프로젝트에 연결된 다른 팀원이 사용할 수도 있습니다.



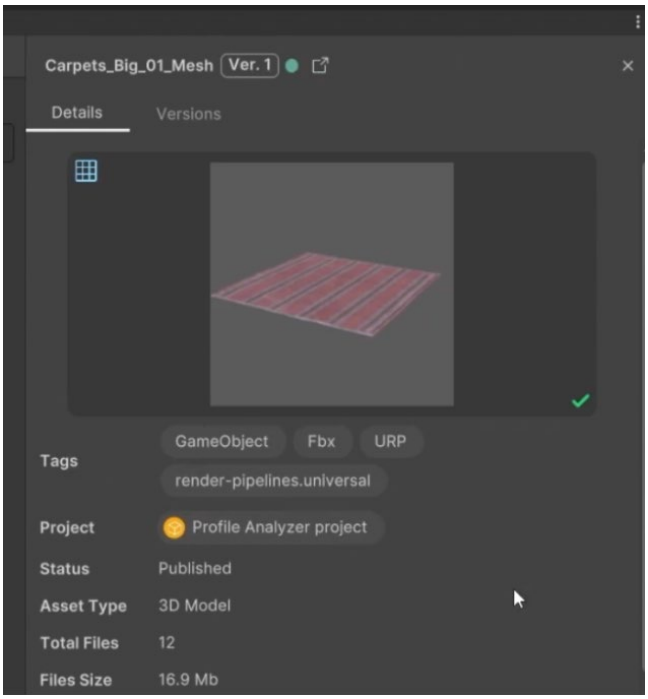
에셋을 Upload 탭에 드래그하여 클라우드에 업로드합니다.

그런 다음 Asset Manager 창에서 직접 다른 프로젝트에 파일을 임포트할 수 있습니다. 이 간단한 시스템을 통해 모든 프로젝트 에셋을 종합적으로 보관할 수 있는 것입니다.



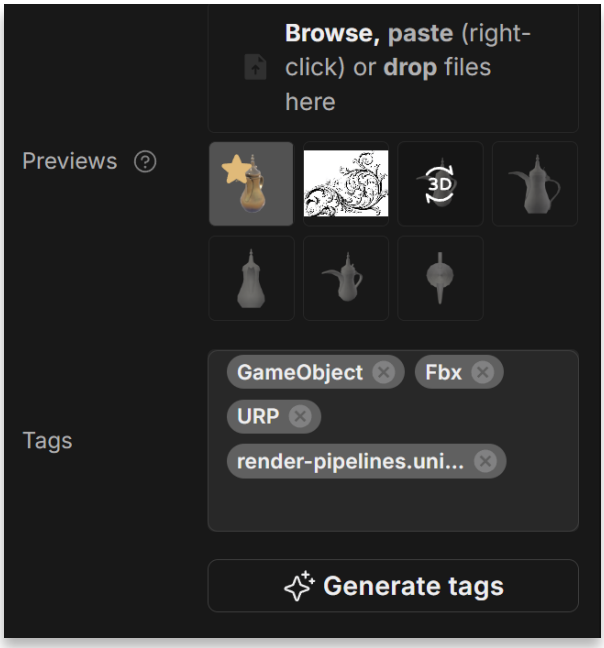
Import를 클릭하여 프로젝트에 파일을 다운로드 및 임포트합니다.

에셋이 많으면 빠르게 검색할 수 있도록 태그가 자동으로 생성됩니다. 에셋은 프로젝트에도 연결되므로 특정 프로젝트 또는 아티스트와 관련된 에셋만 제한적으로 검색할 수도 있습니다.



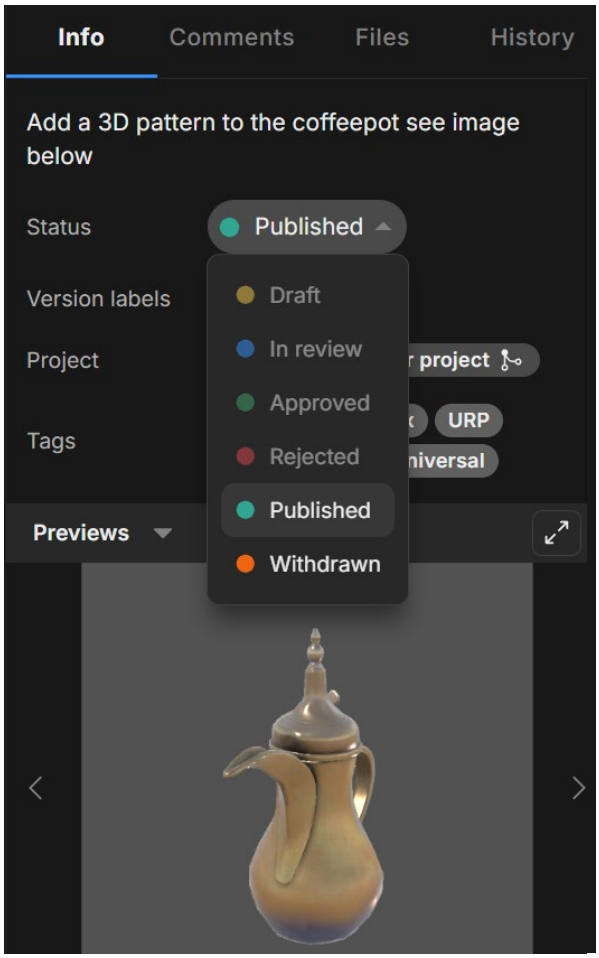
에셋을 빠르고 쉽게 검색할 수 있도록 자동 생성된 태그

Unity Cloud에서 에셋을 편집할 수 있습니다. 커스텀 썸네일, 프리뷰 이미지, 태그, 메모를 추가해 보세요.



에셋을 편집하여 미리 보기 이미지와 태그를 추가합니다.

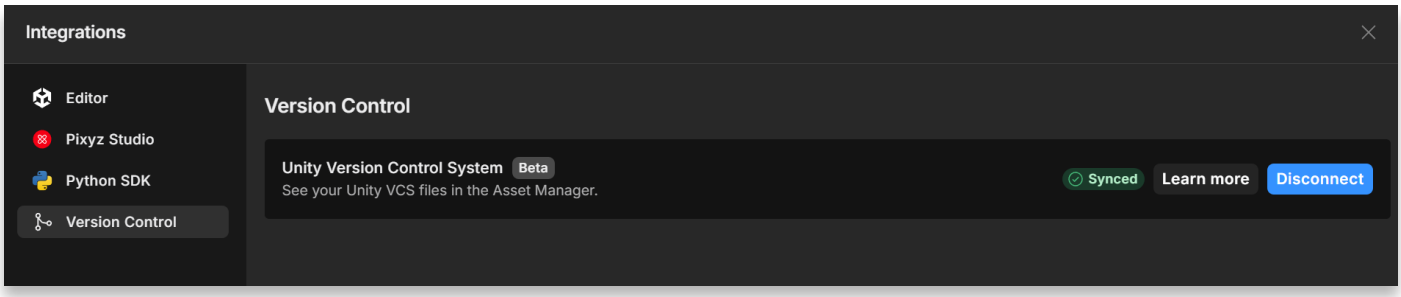
Unity Cloud에서 모델의 상태를 변경하여 에셋이 초안, 검토 중, 승인됨, 거부됨, 퍼블리시됨, 취소됨 중 어떤 상태인지 다른 사람에게 알릴 수 있습니다. 메모를 추가하면 변경 사항에 대한 자세한 내용을 남길 수 있습니다. 그러면 Unity의 Asset Manager 창에서 에셋의 상태가 반영됩니다.



에셋 상태 변경

팀원은 항상 최신 버전에 액세스할 수 있으므로 에셋을 반복 작업하며 문제가 발생하는 것을 줄일 수 있습니다.

Unity Cloud에서 연동을 활성화하면 Asset Manager를 UVCS와 함께 사용할 수 있습니다.



Asset Manager와 UVCS를 연동하여 데이터 동기화

UVCS를 거친 모든 변경 사항이 Unity Asset Manager와 자동으로 동기화되므로 팀에서 콘텐츠를 중복 제작하는 것을 예방할 수 있습니다.

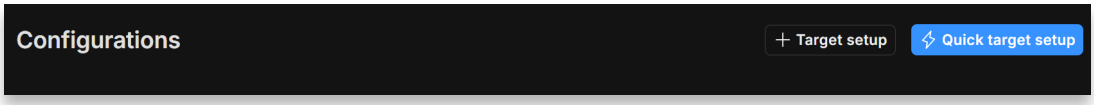
Unity Asset Manager의 장점에 대해 자세히 알아보려면 여기를 참고하세요.

Unity Learn의 Asset Manager 튜토리얼도 확인해 보세요.

Build Automation

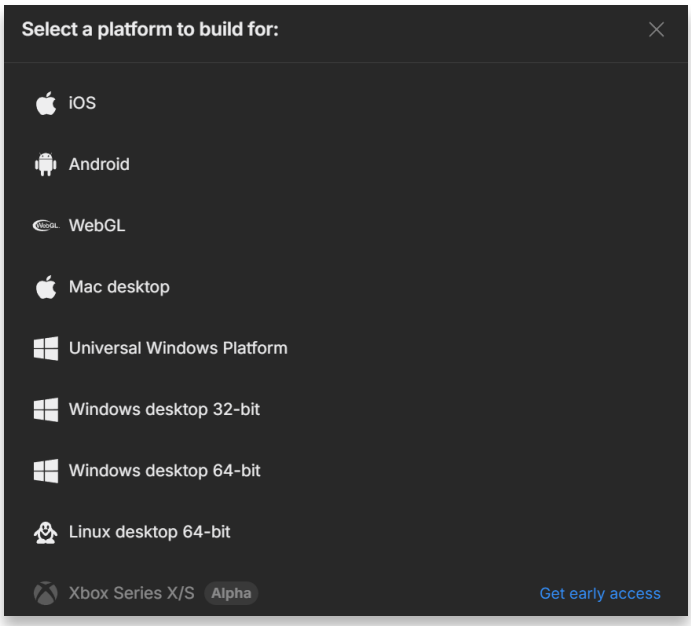
빌드 자동화는 모든 DevOps 전략에서 가장 중요한 부분입니다. 클라우드에서 빌드를 만드는 것뿐 아니라, 모든 타겟 플랫폼의 빌드를 동시에 제작할 수 있죠. 기기에서 빌드가 순차적으로 완료되는 것을 기다리는 대신 그 시간에 프로젝트 제작을 진행할 수도 있습니다.

Unity Cloud 대시보드의 **Configurations** 섹션에서 원하는 만큼 빌드 타겟을 설정할 수 있습니다.



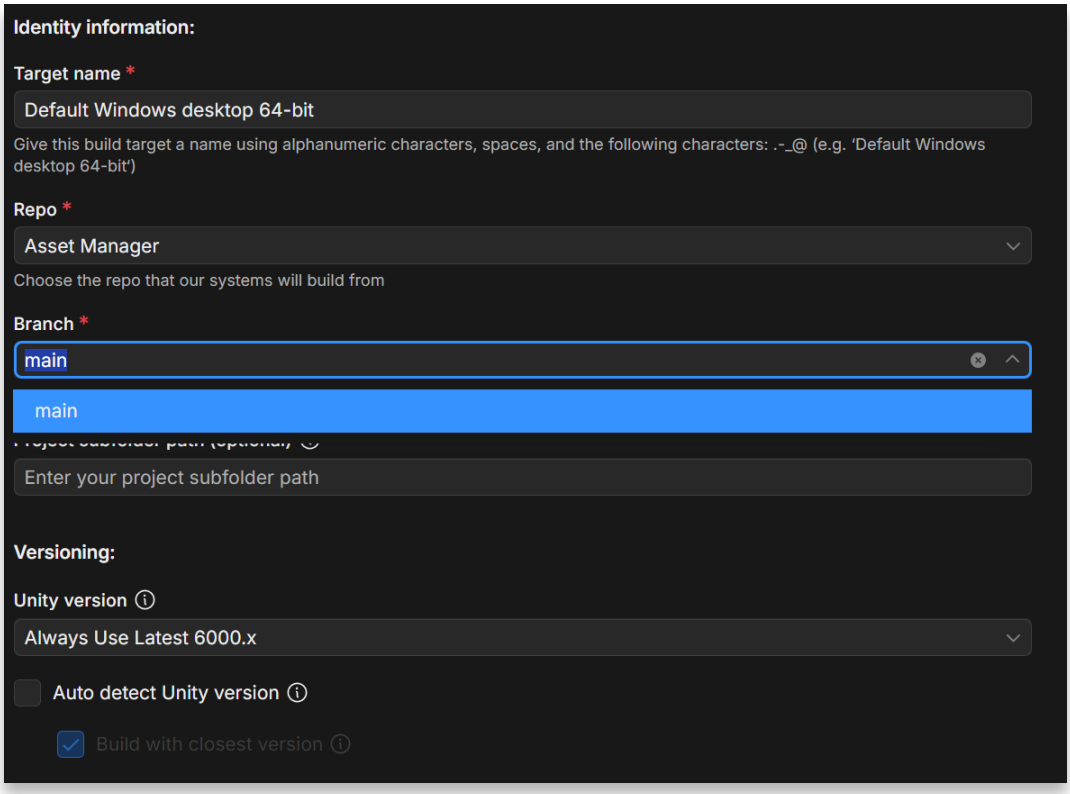
Quick target setup을 클릭하여 빌드 타겟을 생성하거나 Target setup을 클릭하여 고급 옵션을 설정합니다.

빌드할 플랫폼을 선택합니다. 플랫폼별로 빌드 타겟을 여러 개 설정할 수 있습니다.



빌드할 타겟 플랫폼

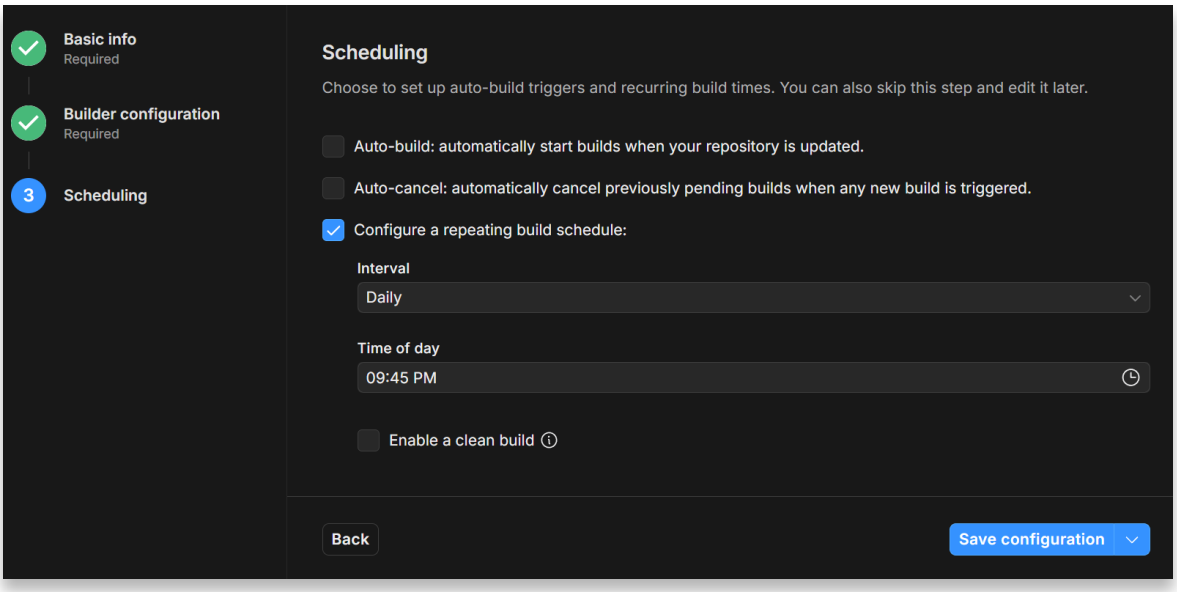
모든 브랜치에서 빌드하고 브랜치별로 여러 타겟을 설정할 수 있습니다. 특정 Unity 버전을 지정할 수도 있습니다.



빌드를 생성할 브랜치를 선택합니다.

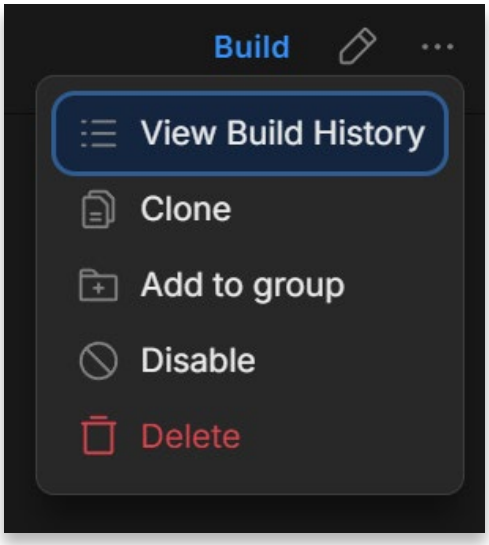
참고: 이 가이드 작성 시점을 기준으로, 빌더 운영 체제의 Windows 11에서는 매월 200분의 무료 빌드 시간을 제공합니다.

그런 다음 일정을 설정합니다. 진행 상황을 테스트하기 위해 한 번 빌드하거나 반복적인 일정을 설정할 수 있습니다. 아래 예시에서는 매일 오후 9시 45분에 빌드하여 매일 아침 최신 게임 빌드를 통해 문제나 버그가 있는지 확인할 수 있습니다. 빌드가 완료되면 빌드의 성공 또는 실패 여부를 나타내는 상태가 이메일 메시지로 전송됩니다.



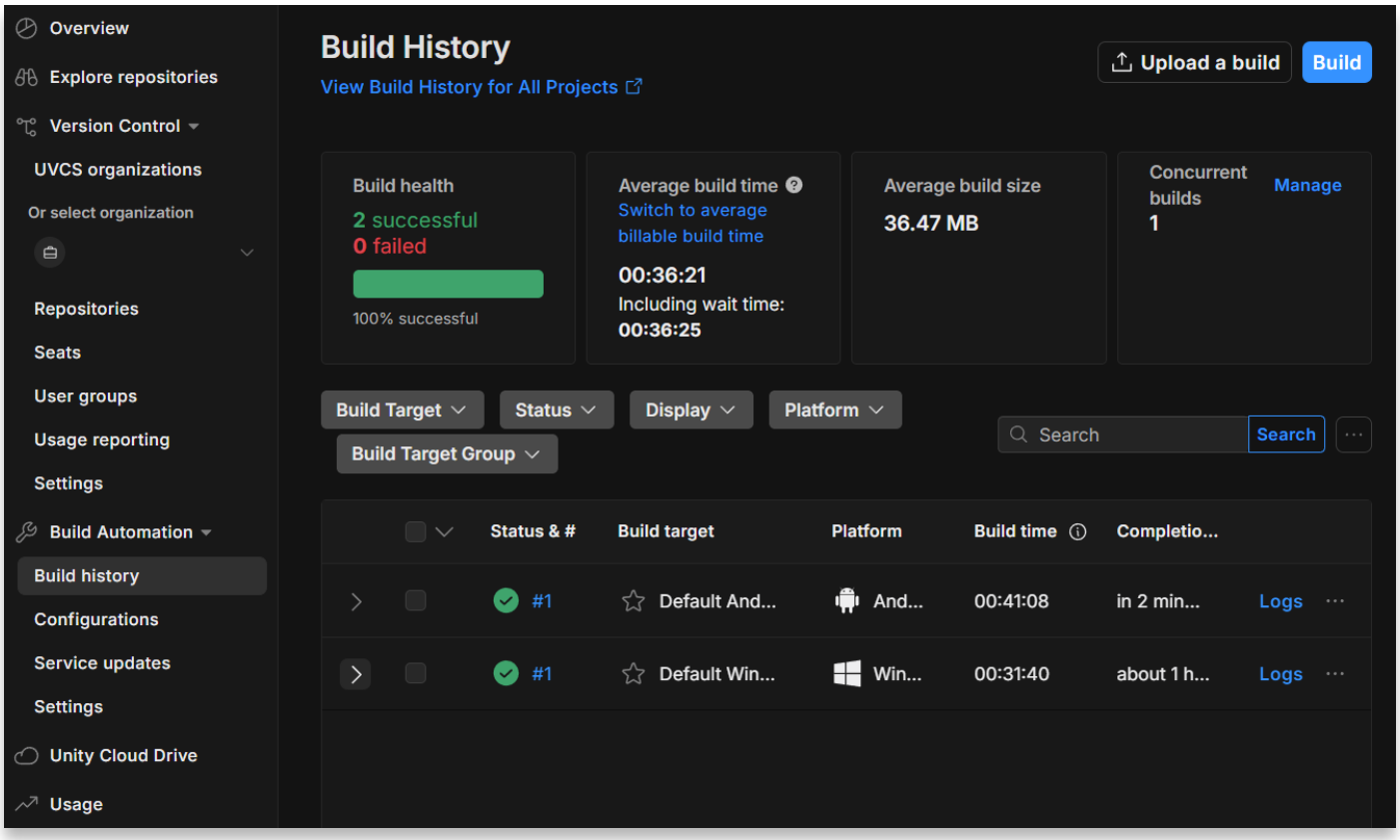
빌드 일정 설정

Configurations 섹션에서 설정 옆의 **Build**를 클릭하여 즉시 빌드할 수 있습니다. 빌드 설정을 일시 중지하거나 삭제할 수도 있습니다.



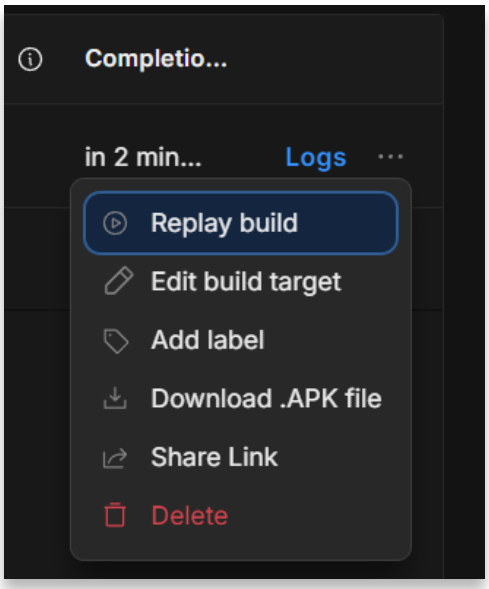
빌드 일정 비활성화 또는 삭제

Build Automation 대시보드의 **Build History** 섹션에서 빌드 결과를 확인할 수 있습니다. 실패한 빌드가 있다면 로그를 확인하여 문제를 해결할 수 있습니다.



Unity Cloud의 Build History

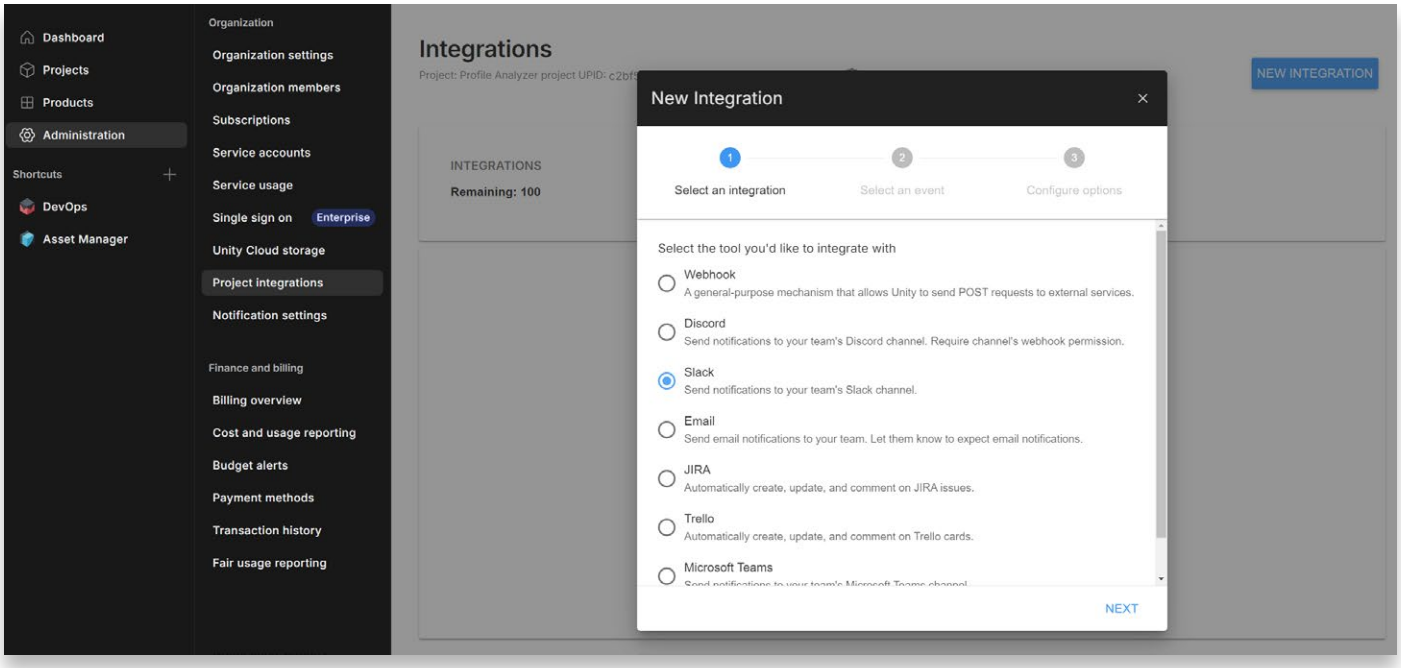
Unity Cloud에서 빌드를 다운로드, 공유, 삭제할 수 있습니다.



빌드 다운로드, 공유, 삭제

빌드 완료 시 Discord 또는 Slack 같은 다른 애플리케이션으로 메시지를 전송하려면 연동을 설정합니다.

Unity Cloud에서 **Administration** > **Project Integrations**로 이동하여 **New Integration**을 클릭합니다.



Administration > Project Integrations에서 새 연동 추가

사용할 애플리케이션을 선택합니다. 이제 설정한 애플리케이션으로 전송된 메시지를 통해 팀원들이 빌드 완료 여부를 알 수 있습니다.

Unity Build Automation에 대해 자세히 알아보세요.

버전 관리 시스템

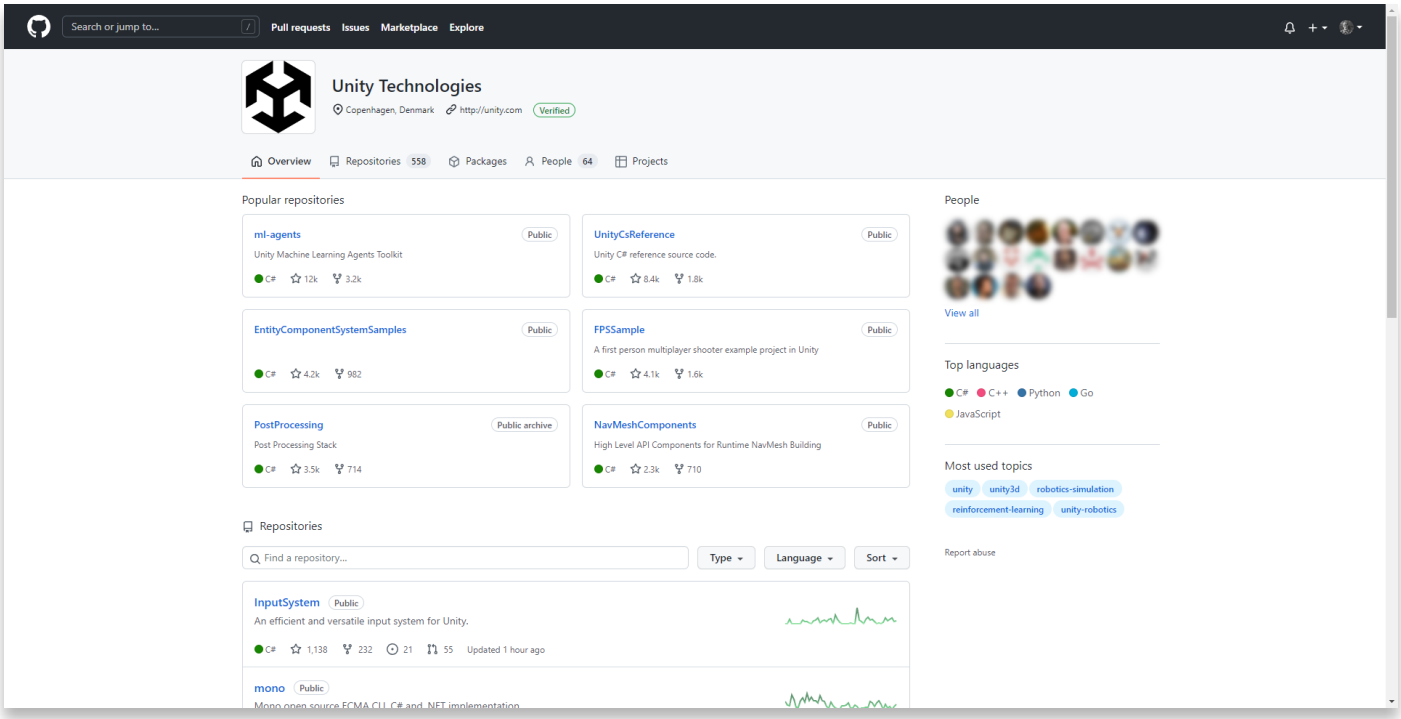
지금까지 버전 관리 시스템의 용어와 개념을 알아보고 프로젝트 구성 및 명명 규칙 베스트 프랙티스를 살펴봤으니, 이제 주요 시스템 중 일부를 살펴보겠습니다. 물론 누구에게나 완벽한 한 가지 솔루션이란 없습니다. 팀에서 사용할 VCS를 선택하려면 여러 사항을 고려해야 합니다. 그러한 결정을 내릴 때 이 가이드를 통해 많은 정보를 얻을 수 있기를 바랍니다.

Git

유연한 무료 오픈 소스 시스템인 Git은 널리 사용되는 버전 관리 시스템입니다. 하지만 분산형 시스템이므로 기술적인 배경이 없는 사용자에게 어려울 수 있습니다.

Git은 2005년에 리누스 토발즈가 Linux 커널 개발을 제어할 목적으로 개발했으며, 이후로도 계속 오픈 소스로서 잘 관리되어 왔습니다. Git 플랫폼은 커맨드 라인 전용 툴입니다. 하지만 다양한 전용 GUI가 개발되었기 때문에 더 손쉽게 Git 시스템을 사용할 수 있습니다.

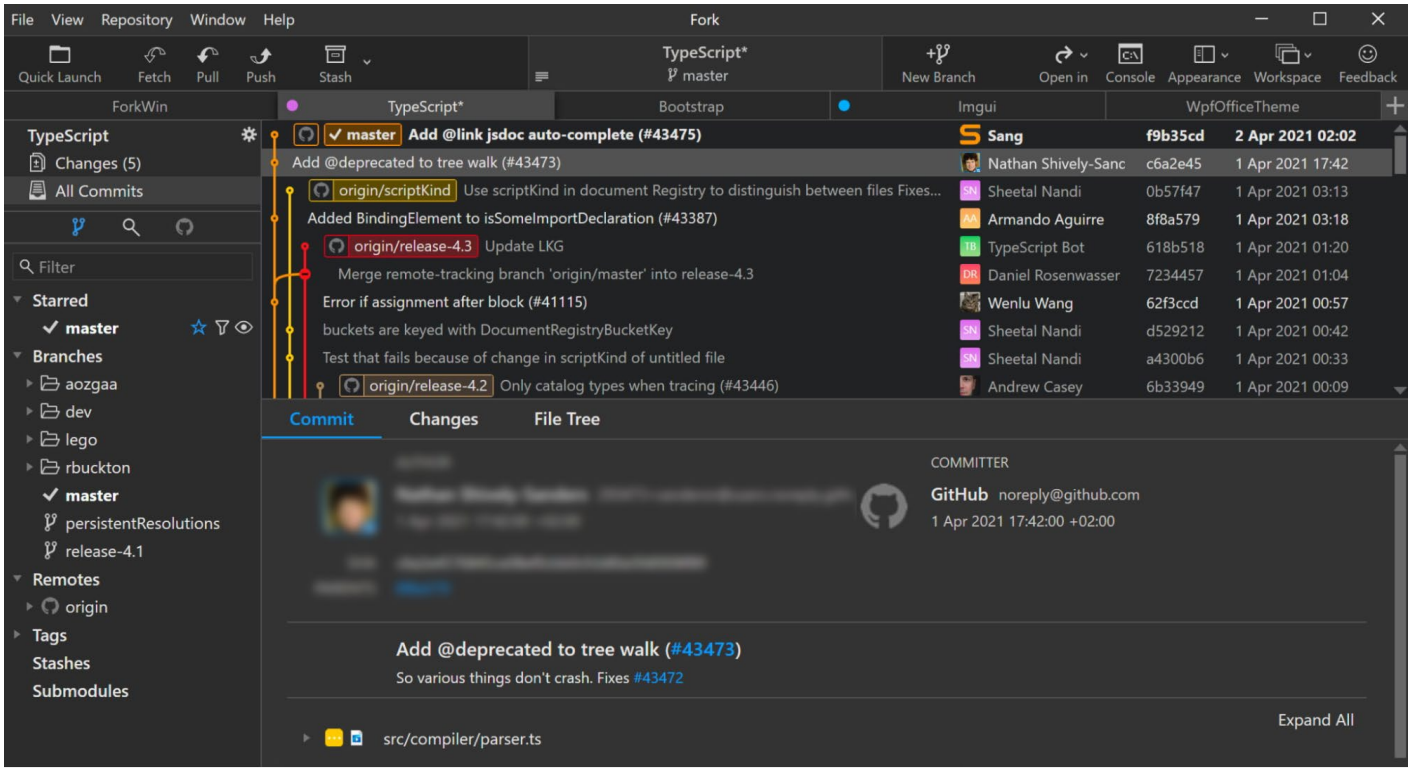
Git과 GitHub를 혼동하는 경우가 자주 있습니다. GitHub는 Git 저장소를 위한 호스팅 서비스이며 GitHub가 없어도 Git을 사용할 수 있습니다. GitHub는 일부 제한이 있지만 무료로 사용할 수 있으며, 별도로 커스텀 서버를 설정하지 않아도 되는 매우 인기 있는 서비스입니다.



GitHub

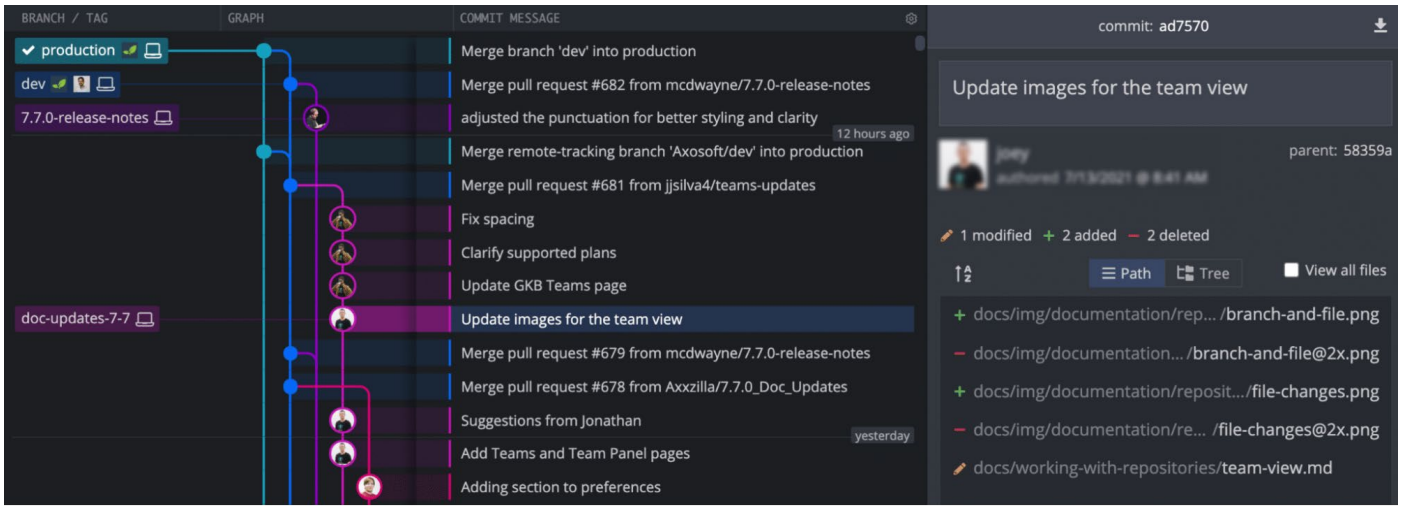
다음과 같은 Git GUI 클라이언트가 널리 사용됩니다.

Fork: 빠르고 사용자 친화적인 GUI를 제공하며 무료로 평가판을 다운로드할 수 있습니다.



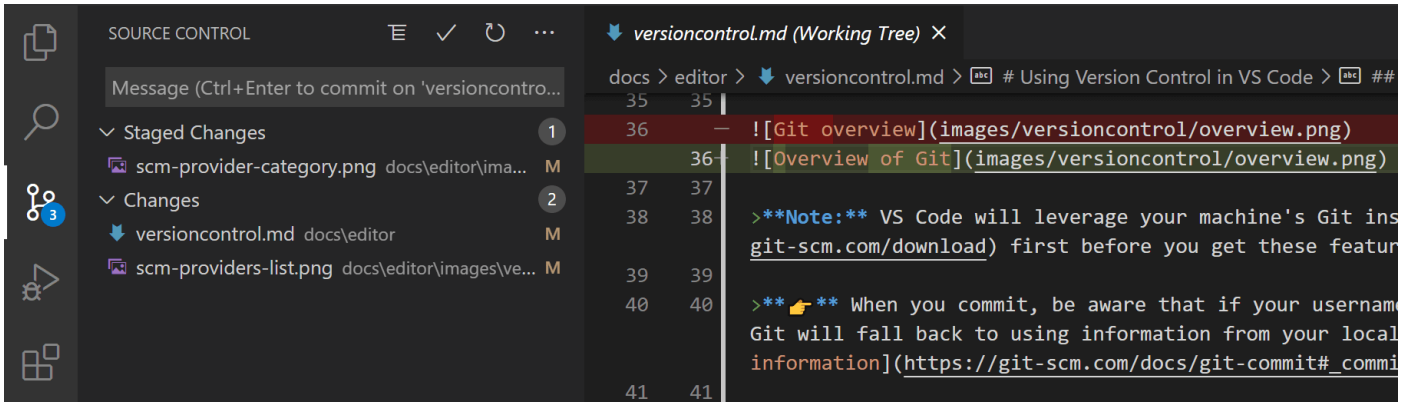
Fork

GitKraken: 직관적인 UI를 통해 더 시각적이고 이해하기 쉬운 방식으로 Git을 사용할 수 있도록 지원하며, GUI와 CLI 터미널 간에 전환할 수 있는 유연성을 갖춘 클라이언트입니다.



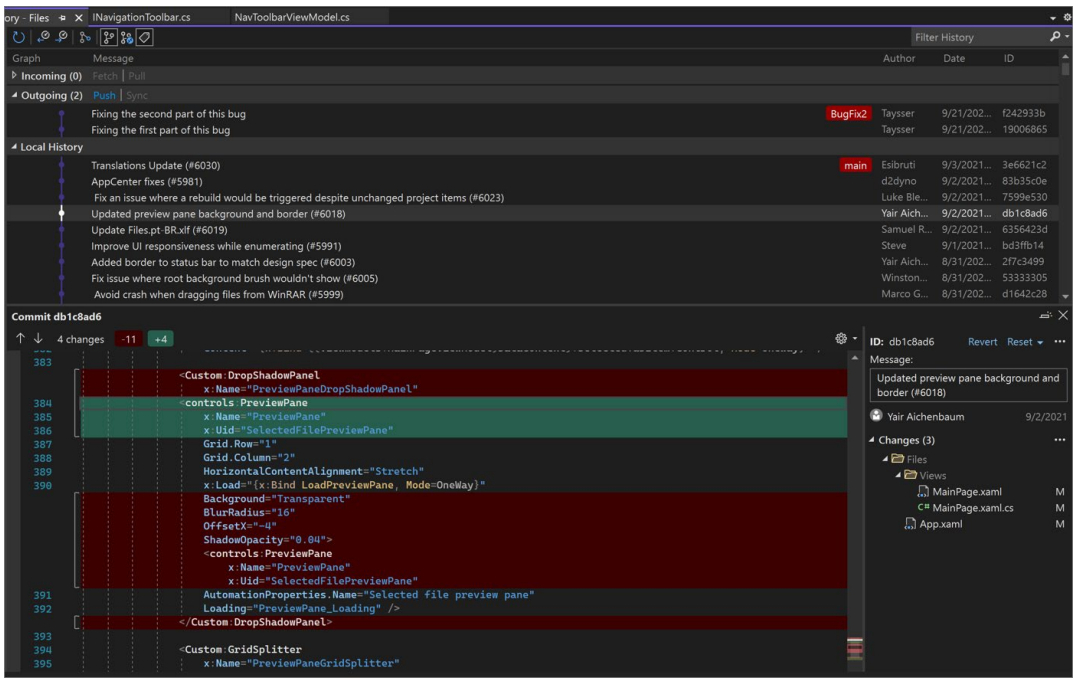
GitKraken

Microsoft Visual Studio Code: VS Code에는 소스 관리 통합 기능이 내장되어 있으며, 다양한 확장 프로그램을 사용할 수 있으므로 여러 프로그램을 별도로 사용하지 않아도 됩니다.



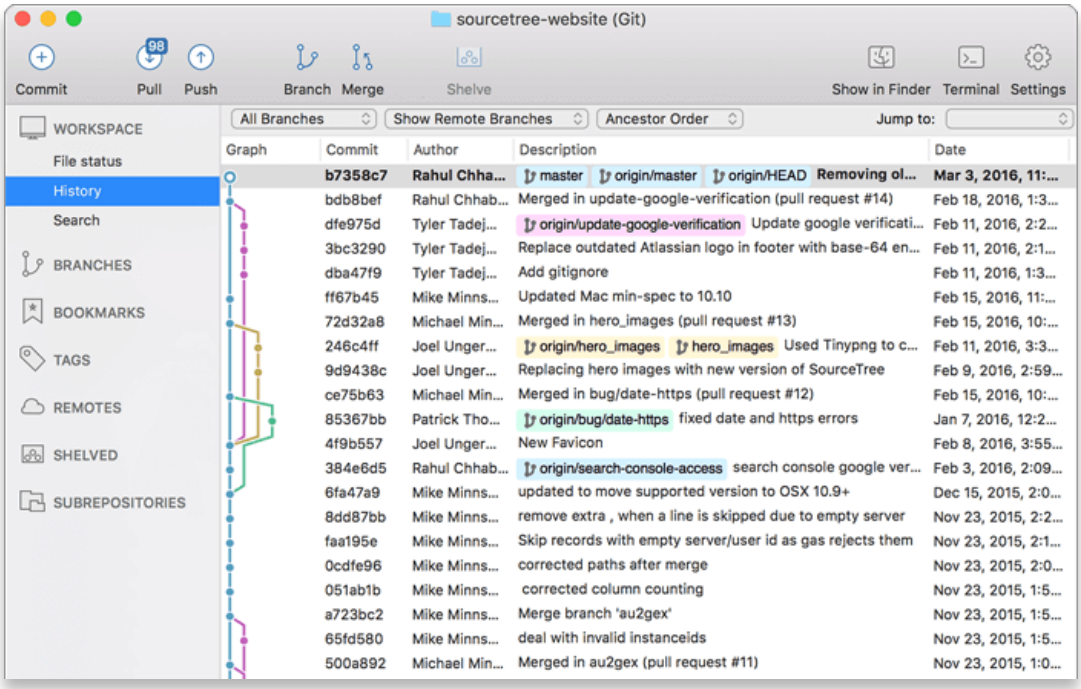
Visual Studio Code

Microsoft Visual Studio: VS Code와 마찬가지로 Visual Studio에도 Git 관리 기능이 내장되어 있으며 GitHub 확장 프로그램을 사용할 수 있습니다.



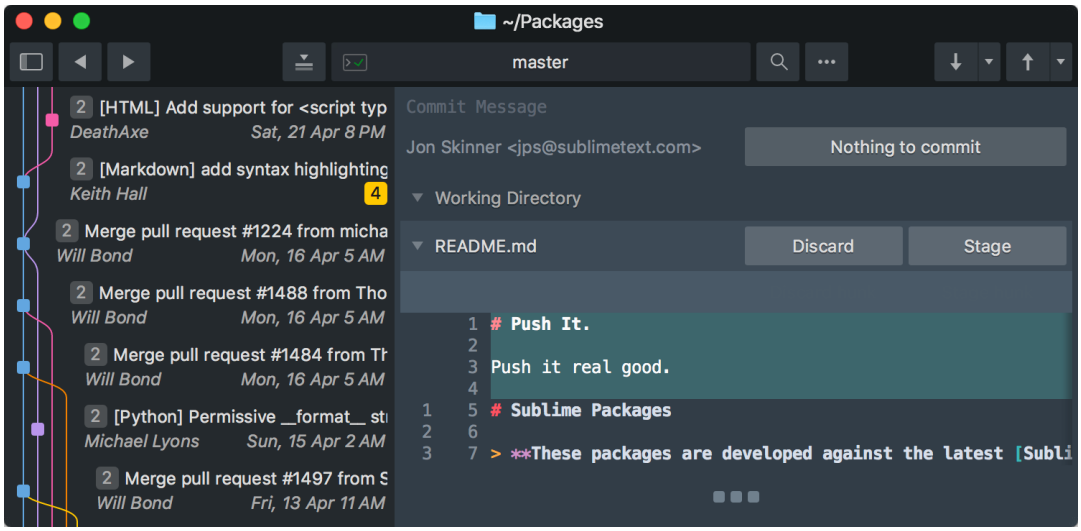
Visual Studio

SourceTree: Atlassian 제품군에 속한 SourceTree는 Windows 및 Mac용 무료 Git 클라이언트로서 Git 저장소를 간단하게 시각화하고 관리할 수 있습니다.



SourceTree

Sublime Merge: 코드를 나란히 비교하거나 구문을 강조하는 등 코드를 빠르게 검토할 수 있는 툴을 지원하는 시스템입니다. 가벼우며 성능이 뛰어난 클라이언트입니다.



Sublime Merge

Git은 일반적으로 브랜칭 및 병합 기능에서 강점을 보이지만, 대용량 바이너리 파일은 시중의 다른 솔루션만큼 효과적으로 처리하지 못합니다. Git LFS(Large File Storage)가 이러한 단점을 어느 정도 보완합니다.

Git은 분산형 클라이언트이므로 전체 저장소와 이력은 개발자의 기기에 존재합니다. 따라서 브랜치를 전환하거나 이력의 특정 시점으로 되돌리는 작업을 매우 빠르게 완료할 수 있습니다. 여러 기능 및 릴리스 브랜치로 구성된 대규모 프로젝트를 진행하는 경우 Git 워크플로를 통해 엄청난 시간을 절약할 수 있습니다.

> 유니티는 GitHub에 C# 에디터 및 엔진 코드를 공개했습니다. 일부 기능의 작동 방식을 확인하거나 프로젝트에 에디터 기능을 복제하려는 경우 유용하게 활용할 수 있습니다.

GitHub에서도 자체 Git GUI인 GitHub Desktop을 제공합니다. Unity에서 작업하는 경우 GitHub for Unity 패키지를 통해 Unity 에디터에서 직접 Git 툴을 사용할 수도 있습니다.

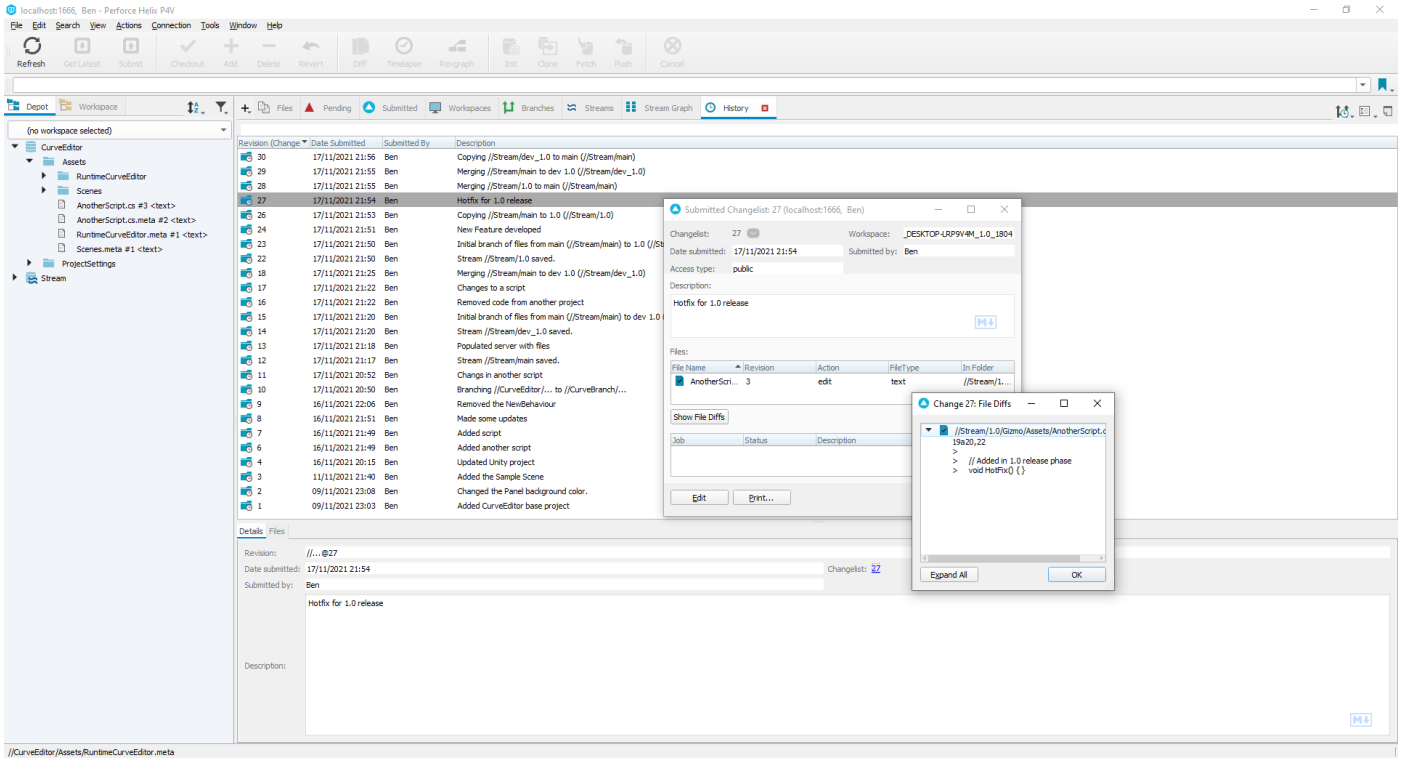
앞에서 설명한 대로 GitHub가 Git 프로젝트에 사용할 수 있는 유일한 호스팅 서비스인 것은 아닙니다. 다양한 DevOps 기능을 제공하는 Atlassian의 Bitbucket이나 GitLab을 비롯하여 사용할 수 있는 호스팅 서비스가 많습니다.

이 Unite Now 2020 세션에서 Github, GitKraken, Unity를 시작하는 방법을 알아보세요.

Perforce (Helix Core)

Helix Core는 엔터프라이즈급 버전 관리 시스템으로, 대형 게임 스튜디오에서 많이 사용합니다. Perforce는 대체로 자체 서버에 호스팅되는 중앙 집중형 저장소를 갖추고 있기 때문입니다. 저장소를 시각화하는 기능이 없으므로 기술적 배경이 없는 개발자라면 익숙해지기 어려울 수 있지만, 대형 스튜디오에는 코드베이스를 관리할 DevOps 및 릴리스 엔지니어가 있을 것입니다. 또한 엔터프라이즈 솔루션이므로 글로벌 지원 팀을 제공합니다.

소규모 팀이라도 Helix Core를 사용할 수 있으며 Amazon AWS나 Microsoft Azure 같은 솔루션을 통해 클라우드에 배포할 수 있습니다.



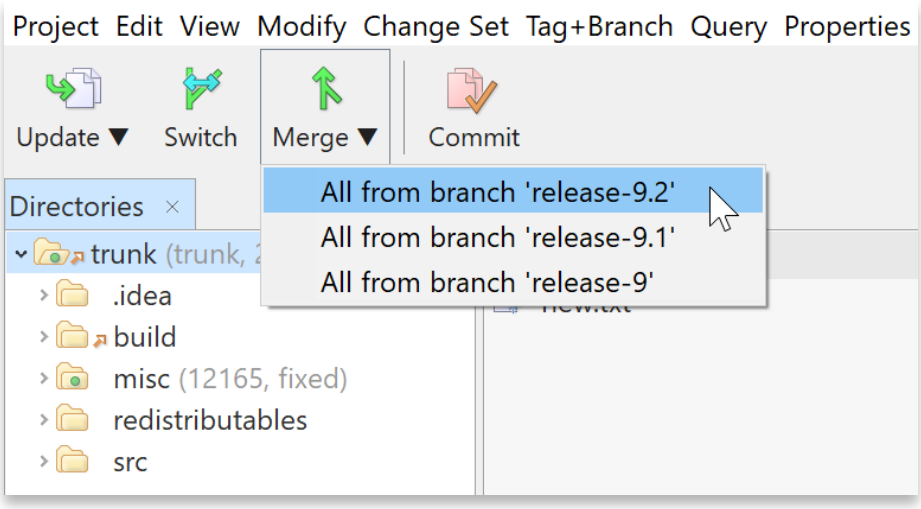
Helix Core P4V 인터페이스

Helix Core는 대용량 파일을 처리할 수 있으며 Unity 에디터 연동 기능이 내장되어 있습니다. 관련된 내용은 이후 다른 섹션에서 살펴보겠습니다.

Unity 워크플로와 Helix Core를 연동하는 방법을 자세히 알아보려면 이 Perforce 블로그 게시물을 참고하세요.

Apache Subversion

Apache Subversion(또는 SVN)은 Git과 마찬가지로 무료 오픈 소스 버전 관리 시스템입니다. Git과는 달리 대용량 바이너리 파일을 처리할 수 있는 중앙 집중형 VCS입니다. 하지만 커맨드 라인 시스템이기 때문에 조금이라도 편하게 사용하려면 타사 GUI 클라이언트를 활용해야 합니다. SmartSVN이 그중 하나입니다.



SmartSVN GUI

Git LFS가 제공되기 전에는 Unity에서 작업할 때 SVN을 많이 사용했습니다. 중앙 집중형 솔루션이라 작업하기 쉽고 언급한 대로 대용량 파일을 처리하는 데 유용했습니다. 그러나 SVN은 브랜치를 사용하고 병합하기 시작하면서 다른 툴보다 역량이 부족해지기 시작합니다. SVN에서는 병합이 쉽지 않으며, 특히 파일 간 충돌이 발생한다면 충돌의 진위 여부에 상관없이 작업이 어렵습니다. 다른 VCS에서 몇 분 만에 처리할 수 있는 병합 작업이라도 SVN에서는 몇 시간 동안 직접 처리해야 할 수 있습니다.

Unity에서 SVN을 설정하는 방법을 자세히 알아보려면 Unity 기술 자료를 참고하세요.

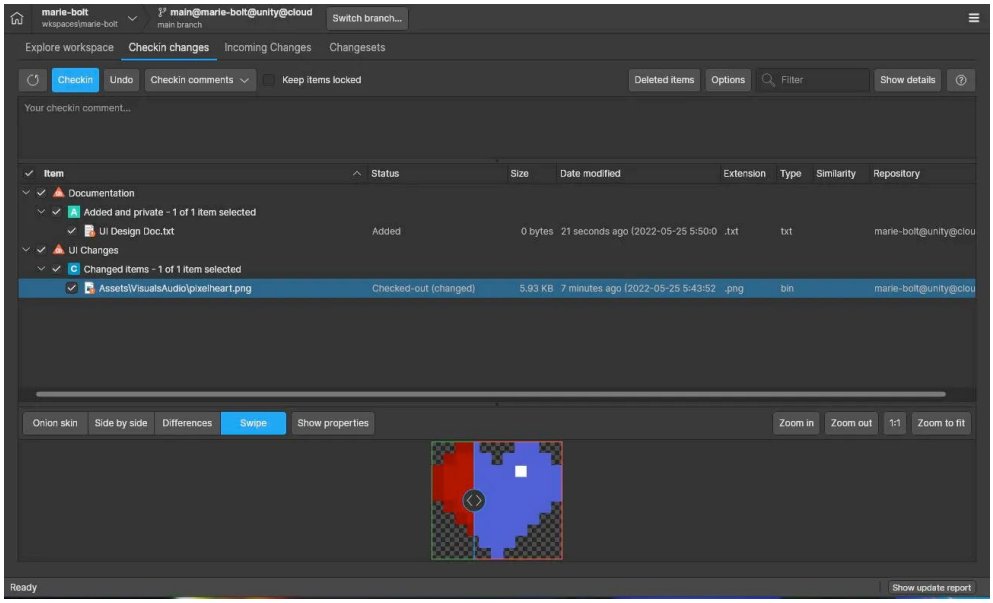
Unity Version Control

UVCS(Unity Version Control)는 고유한 인터페이스로 프로그래머와 아티스트를 모두 지원하는 유연한 버전 관리 시스템입니다. 대규모 저장소와 바이너리 파일 처리에 탁월하며, 파일 기반 솔루션이자 체인지 세트 기반 솔루션이므로 프로젝트 빌드 전체가 아닌 작업 중인 특정 파일만 다운로드할 수 있습니다.

UVCS는 3가지 방법으로 사용할 수 있습니다. UVCS 데스크톱 클라이언트를 통해 여러 애플리케이션과 저장소를 사용하거나, Unity Hub를 통해 프로젝트에 추가하거나, 웹 브라우저를 통해 Unity Cloud의 저장소에 액세스할 수 있습니다. 설정하는 방법을 자세히 알아보려면 'Unity 6에서 UVCS 시작하기' 섹션을 참고하세요.

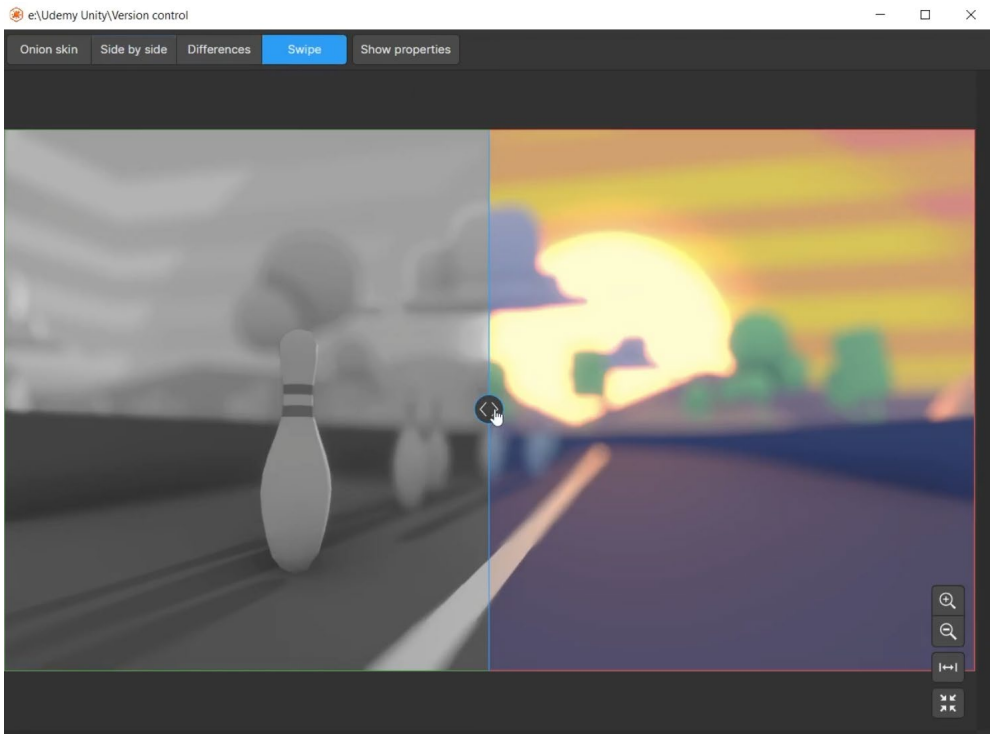
3명 이하의 사용자로 구성된 소규모 팀이라면 무료로 가입할 수 있으며, 이 가이드 작성 시점 기준으로 5GB의 클라우드 스토리지가 제공되고 Gluon을 비롯한 버전 관리 소프트웨어를 사용할 수 있습니다.

Gluon은 아티스트가 프로그래머가 아닌 아티스트 본연의 일을 하도록 지원하기 위해 만들어진 가벼운 클라이언트입니다. 작업할 파일만 선택하여 서버에서 체크아웃하고 다른 팀원이 수정하지 못하도록 잠그는 기능을 제공합니다. 작업을 완료하면 파일을 다시 체크인하면 됩니다. Gluon GUI는 기술적인 지식이 부족한 사용자보다 프로그래머에게 더 도움이 되는 복잡한 개념을 없앤 인터페이스입니다.



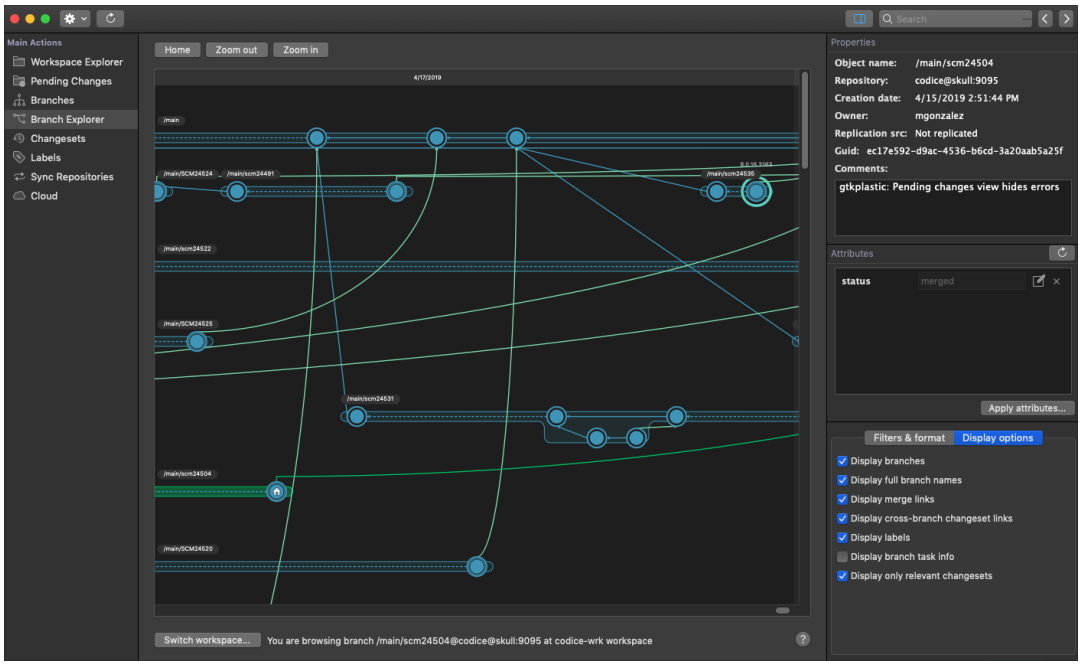
Gluon은 아티스트 전용으로 설계된 워크플로를 제공하므로 간편하게 파일과 이력을 미리 보고 변경 사항을 체크인할 수 있습니다.

아티스트는 UVCS와 Gluon 양측에서 이미지를 비교할 수 있습니다. 다른 시스템에서는 일반적으로 제공하지 않는 이미지 비교 툴을 사용하여 동일한 파일의 두 버전을 눈으로 보면서 비교할 수 있습니다.



스와이프 컨트롤을 드래그하여 두 버전을 전환하며 이미지 변화를 추적하는 데 유용한 비교 뷰어의 스와이프 모드

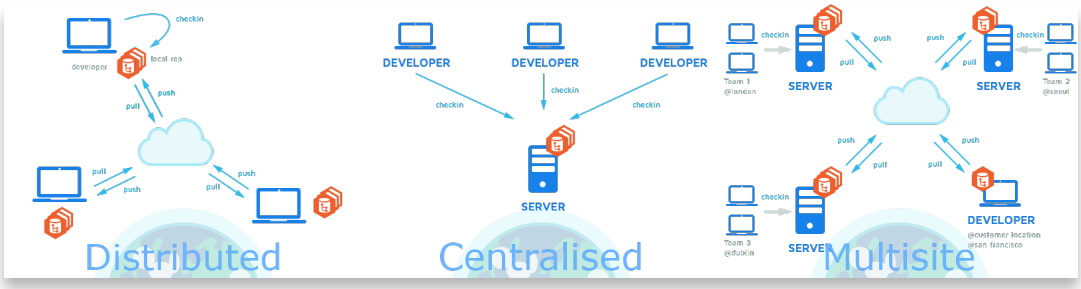
기본 UVCS GUI 클라이언트에는 프로그래밍 팀에 필요한 모든 기능이 담겨 있습니다. GUI의 Branch Explorer에서 상호 작용을 통해 시각적으로 프로젝트 내 모든 브랜치의 실제 관계를 확인할 수 있습니다. 또한 빌트인 Code Review 시스템이 있어 다른 그룹이나 팀원에게 작업 검토를 요청할 수 있습니다.



왼쪽에서 오른쪽 방향으로 진행되는 프로젝트의 병합 구조를 보여 주는 Branch Explorer

UVCS의 주요 강점 중 하나는 분산형 또는 중앙 집중형 워크플로에 맞게 설정할 수 있는 유연성을 갖췄다는 점입니다. 완전 분산형 모드에서는 개발자가 로컬 기기의 저장소에서 간편한 작업과 체크인, 브랜칭, 병합이 가능합니다. 그런 다음 준비가 되면 서버에 변경 사항을 푸시 및 풀링하여 공유하면 됩니다.

중앙 집중형 모드에서는 사용자가 서버에서 직접 변경 사항을 체크아웃 및 체크인하므로 모두가 최신 변경 사항을 가지고 작업합니다. 그러나 개발 팀이 글로벌 조직으로 성장할수록 모두가 하나의 중앙 서버와 연결하는 것이 항상 좋지는 않습니다. UVCS는 멀티사이트 시스템으로 설정할 수도 있습니다. 이 시스템에서는 각 현장에 서버를 설정하므로 로컬 서버에 체크인하여 신속한 워크플로와 넉넉한 하드 드라이브 공간을 유지할 수 있습니다. 그런 다음 분산형 서버는 중앙 서버 또는 클라우드 서버와 서로 통신합니다.



Unity에서 UVCS를 설정하는 단계를 알아보려면 Unity 6에서 UVCS 시작하기 섹션과 이 Unity Learn 빠른 시작 튜토리얼을 참고하세요.

VCS 비교

		UVCS	Git	Perforce	Subversion
유연성	중앙 집중형 푸시/풀링 없이 체크인만 지원				
	분산형 푸시/풀링 + 로컬 저장소				
바이너리	대규모 저장소				
	대용량 파일				
	병합을 방지하는 파일 잠금 기능				
GUI	저장소 시각화 (브랜칭 전문 지식 불필요)				
	사용자 친화적인 GUI				
	아티스트 친화적인 GUI 및 워크플로				
워크플로	효과적인 작업 브랜치 생성				
병합	브랜치 간 병합 탐지				
	비교 기능 및 3가지 병합 툴				
	병합을 이해하는 데 도움이 되는 툴				
	이름 변경, 파일 이동, 디렉토리, 리팩터링 병합을 잘 처리				

클라우드	클라우드에서 저장소 호스팅				
	클라우드에서 대규모 저장소를 호스팅하기 적합				
차이점 비교	파일 간 이동한 코드 확인				
	메서드 이력 표시				
	엔터프라이즈 지원				

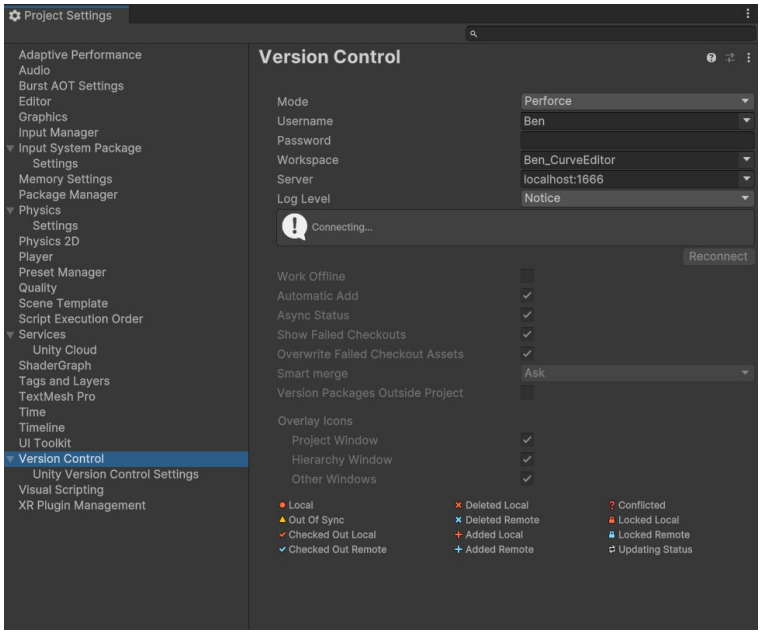
Unity에서 버전 관리 연동 설정

이 섹션에서는 Unity에서 Git, Perforce, UVCS 중에서 사용하고 싶은 시스템을 설정하는 방법을 살펴봅니다. 각 솔루션의 주요 워크플로를 이해하면 어떤 시스템이 팀에 가장 적합한지를 정보에 기반하여 결정할 수 있습니다.

에디터 프로젝트 설정

Perforce Helix Core

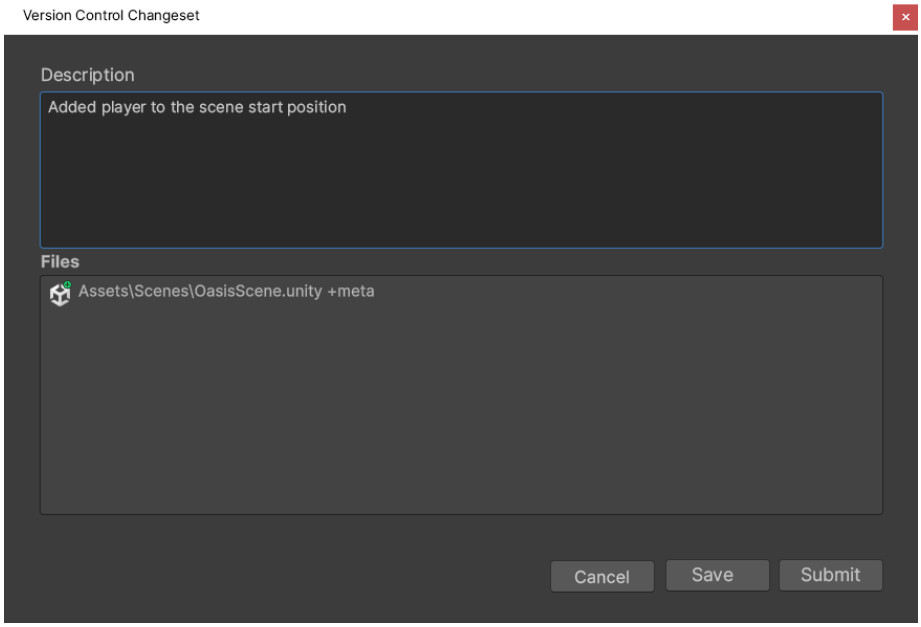
대부분의 버전 관리 시스템은 Unity 에디터에 연동할 수 있으며 Perforce Helix Core 연동 기능은 에디터에 내장되어 있습니다. **Edit** > **Project Settings** > **Version Control**에서 활성화하면 됩니다. Mode를 Perforce로 설정하고 작업 공간 및 서버 설정 정보를 입력합니다.



프로젝트에 Perforce Helix Core 설정

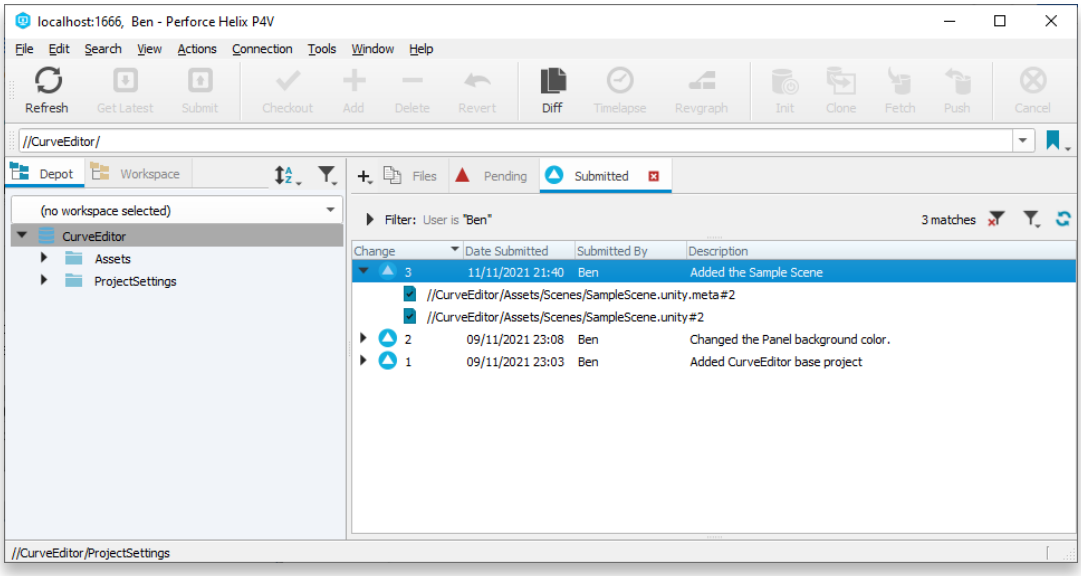
활성화하면 파일이 '버전 관리 적용 중(Under Version Control)' 상태가 되며 파일을 체크아웃할 수 있습니다.

파일을 체크아웃하면 파일을 잠그고, 잠금 해제하고, 제출하고, 되돌릴 수 있습니다. 제출을 선택하면 체인지 세트 다이얼로그가 표시되어 저장소에 제출하기 전에 커밋 메시지를 추가할 수 있습니다.



체인지 세트 다이얼로그 상자

Helix P4V 인터페이스를 사용하여 프로젝트 이력을 확인할 수 있습니다.



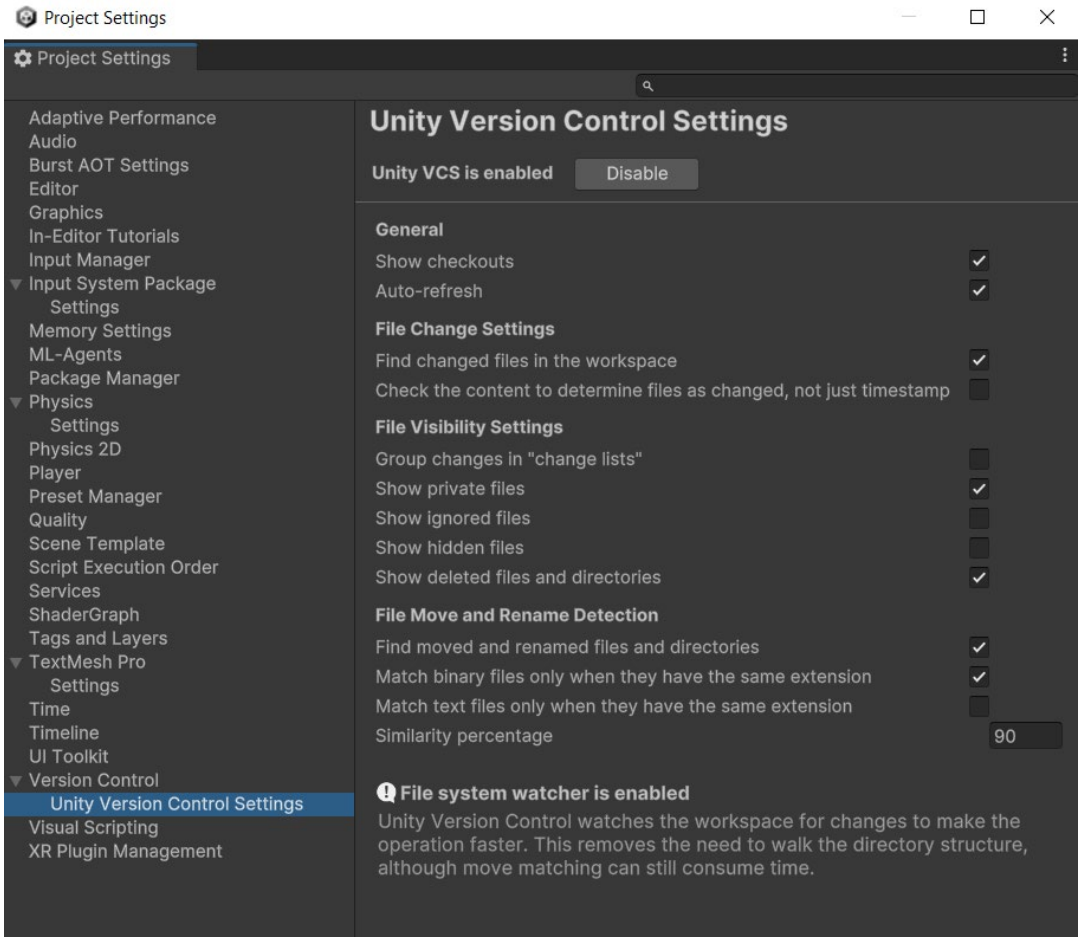
프로젝트 이력 확인

Perforce Helix Core와 Unity를 시작하는 방법을 자세히 알아보려면 Perforce 블로그 게시물을 참고하세요.

UVCS

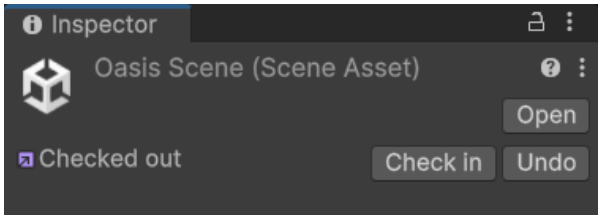
UVCS는 Unity 에디터에 연동되어 있어 손쉽게 시작하고 워크플로를 간소화할 수 있습니다. Unity Hub에서 새 Unity 프로젝트를 생성할 때 **Use Unity Version Control** 체크박스를 선택하여 즉시 Unity VCS와 연동할 수 있습니다.

에디터에서 기존 프로젝트에 UVCS를 연결하려면 **Window > Unity Version Control을 엽니다.** 프로젝트(Project) 창에서 탭이 표시됩니다.

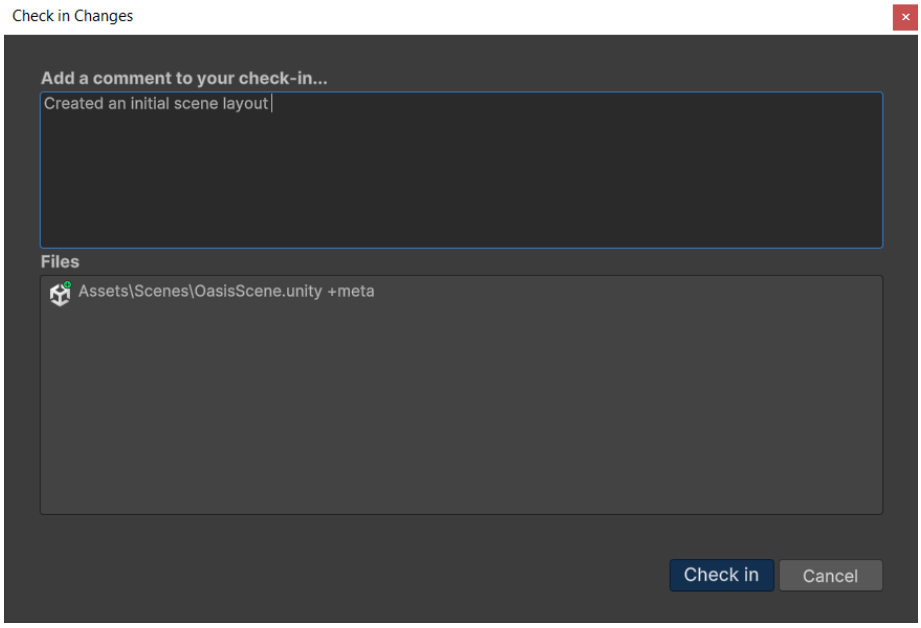


Project Settings의 UVCS

UVCS는 Unity 사용자에게 친숙하고 추가 클라이언트가 필요하지 않으므로 워크플로를 간소화합니다. 에디터에서 직접 파일을 추가하고, 체크아웃하고, 되돌리고, 체크인하고, 제출하면 됩니다.

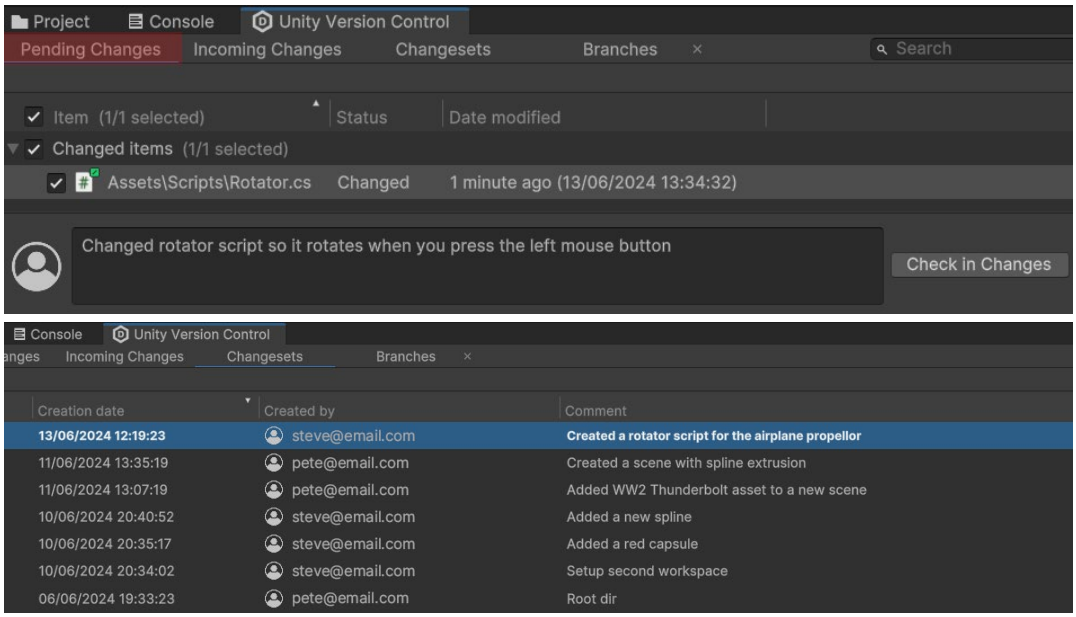


Unity 에디터의 UVCS에서 파일 작업



파일 체크인

UVCS를 사용하면 Unity 에디터의 **Window** > **Unity Version Control**에서 Changesets 탭을 사용할 수 있다는 장점도 있습니다.



Pending Changes 및 Changesets 탭

Unity에서 Version Control을 설정하는 방법을 자세히 알아보려면 기술 자료를 참고하세요.

Git 및 기타 솔루션

다른 VCS의 경우 **Edit** > **Project Settings** > **Version Control** 창을 열고 드롭다운 메뉴에서 Visible Meta Files를 선택합니다. 여기에 다른 옵션은 없지만 버전 관리 시스템이 탐지할 수 있도록 .meta 파일이 표시되어야 합니다(.meta 파일 챕터 참고).

무시할 파일

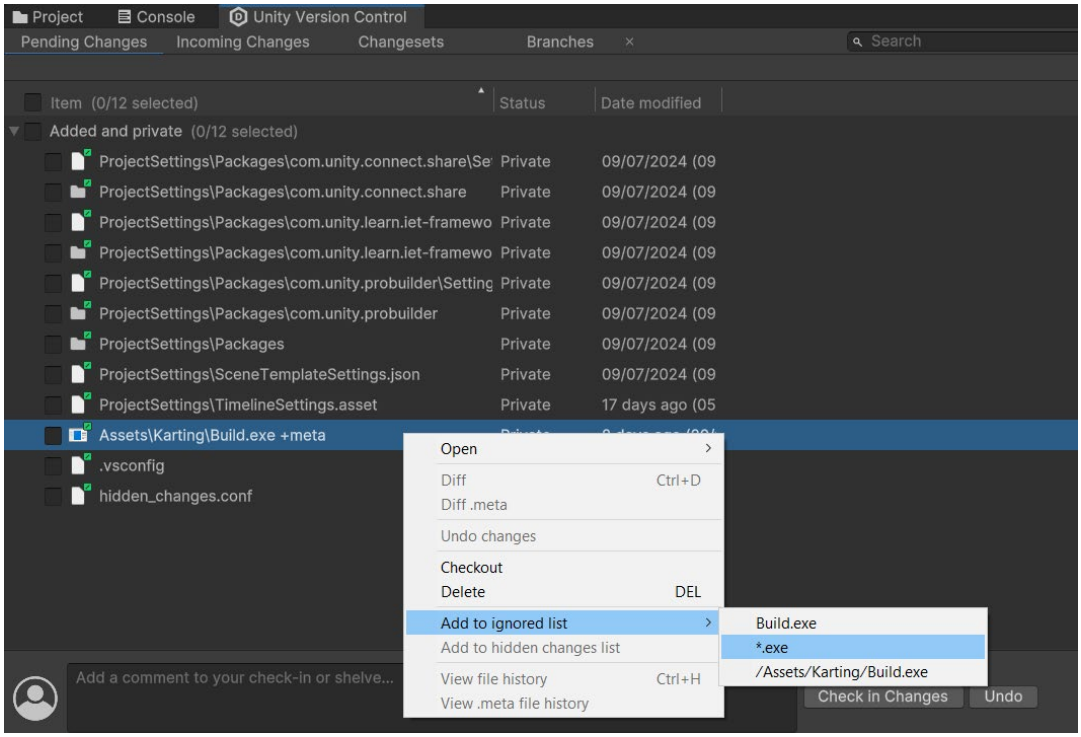
Unity 프로젝트나 다른 프로젝트를 작업할 때 생성될 수 없는 파일에만 버전 관리를 적용해야 합니다.

Unity 프로젝트의 경우 Assets 및 Project Settings 폴더의 파일만 저장소에 커밋해야 한다는 의미입니다. 나머지 폴더는 모두 Unity가 자동으로 다시 생성할 수 있습니다. Library 폴더는 절대 커밋해서는 안 됩니다. 이 폴더는 용량이 상당히 커질 수 있고 에디터를 실행할 때 이 폴더가 없으면 Unity가 다시 생성하기 때문입니다.

- Unity 에디터에서 UVCS를 설정하면 자동으로 적절한 폴더와 파일을 선택하여 버전 관리를 적용합니다. 무시할 파일의 목록이 담긴 'ignore.conf' 파일이 프로젝트 루트에 있습니다. 'ignore.conf' 파일을 설정하는 방법을 자세히 알아보려면 이 블로그 게시물을 참고하세요.
- Perforce를 사용하는 경우, 디포(depot)에 명시적으로 Assets 및 Project Settings 폴더를 추가해야 합니다.
- Git을 사용하는 경우, 무시할 파일을 나타내는 **.gitignore** 파일이 있어야 합니다. Git GUI 클라이언트에 따라 저장소를 생성할 때 템플릿을 선택할 수도 있고, 호스팅을 먼저 설정했다면 GitHub에서 템플릿을 선택할 수도 있습니다. 또는 여기에서 템플릿을 다운로드할 수 있습니다.

.exe나 .apk 같은 파일은 커밋하지 않는 것이 좋습니다. 또한 Unity 프로젝트에서 빌드한 Gradle 및 Xcode 프로젝트도 저장소에 추가하지 않아야 합니다.

Gradle 또는 Xcode 프로젝트의 자동 빌드 프로세스를 설정하는 경우 이 규칙의 예외가 될 수 있으나, 이 경우는 보통 자체 저장소에 커밋했을 것입니다.



UVCS를 사용하는 경우 Unity 에디터에서 직접 무시할 목록에 파일을 추가할 수 있습니다.

대용량 파일 작업

Unity 프로젝트는 코드 외에도 다양한 파일로 구성됩니다. 실제로 Unity 프로젝트에서는 스크립트보다 다른 에셋 파일이 훨씬 많은 경우도 많습니다. 이러한 에셋은 바이너리 파일로 저장됩니다. 텍스처, 모델, 프리팹, 오디오 클립, 타임라인 등이 그 예입니다. 이로 인해 다음과 같은 두 가지 결과가 발생합니다.

— 수정본을 서로 비교하기가 어려울 수 있습니다.

— 차이점을 나타낼 수 없으므로 저장소에 변경 사항을 푸시할 때 파일 전체를 작성하게 됩니다.

분산형 환경에서는 전체 프로젝트 이력을 사용자의 로컬 기기로 가져온다고 설명했습니다. 긴 시간 동안 많은 변경이 이루어진 대용량 파일의 이력이 있다면 기기에는 해당 파일의 많은 복사본이 저장되어 있을 것입니다. 이러한 복사본이 하드 드라이브의 많은 용량을 빠르게 차지할 수 있습니다.

이것이 전통적으로 많은 팀에서 중앙 집중형 워크플로를 선호하게 된 주된 이유 중 하나입니다. 중앙 집중형 워크플로에서는 대용량 바이너리 파일의 버전 기록이 중앙 서버에만 보관되며 개별 사용자의 기기에서는 최신 버전에만 액세스할 수 있습니다.

Perforce와 UVCS는 모두 대용량 파일을 잘 처리할 수 있는 중앙 집중형 시스템입니다. UVCS에서 분산형 파이프라인을 선택할 수도 있지만, 이러한 옵션을 선택할 때 처리할 수 있는 대용량 파일 크기가 달라진다는 점을 고려해야 합니다.

UVCS의 또 다른 기능은 가상 파일 시스템을 사용하는 Dynamic Workspace입니다. Dynamic Workspace는 온디맨드로 파일을 다운로드하므로 작업 공간에는 모든 파일이 표시되지만 실제로 모든 파일이 다운로드된 것은 아닙니다.

분산형 시스템인 Git으로는 대용량 파일을 처리하는 데 어려움이 있을 수 있습니다. 대용량 파일을 다루려면 Git LFS를 사용해야 합니다. Git LFS는 .git 폴더의 대용량 파일을 텍스트 포인터로 대체하고 실제 에셋은 GitHub 같은 서버에 보관합니다.

버전 관리 베스트 프랙티스

어떤 VCS를 사용하건 팀이 효과적으로 작업하기 위해 참고할 다양한 베스트 프랙티스가 있습니다. 모든 팀의 요구 사항은 다르므로 모든 베스트 프랙티스가 모든 팀에 적합하지는 않을 수 있습니다.

대형 스튜디오의 실제 프로젝트 최적화를 지원하는 유니티 엔터프라이즈 지원 팀에서 제공하는 팁은 다음과 같습니다.

소량으로 자주 커밋하기

워크플로에 가장 간단하게 적용할 수 있는 내용임에도 일부 개발자는 가장 어려워하는 팁입니다. 다른 프로젝트 관리 툴을 사용하고 있다면 이미 작업을 작고 관리가 용이한 단위로 세분화했을 가능성이 높습니다. 커밋도 같은 방식으로 처리해야 합니다.

한 줄의 코드로 여러 버그를 마법처럼 수정할 수 있지 않은 이상, 하나의 커밋은 하나의 작업 또는 티켓만 담당해야 합니다. 작업 중인 기능의 규모가 크다면 더 작은 작업으로 나누고 해당 작업별로 커밋하세요. 기능 브랜치에 관해서는 나중에 살펴보겠습니다.

문제가 발생하면 해당 변경 사항을 훨씬 더 쉽게 발견하고 긍정적인 변경 사항에 영향을 주지 않으면서 부정적인 변경 사항을 되돌릴 수 있다는 것이 소량 커밋의 가장 큰 장점입니다.

깔끔한 커밋 메시지 작성

커밋 메시지에서 프로젝트의 이력을 확인할 수 있습니다. 게임에 고득점 표를 추가한 다음 '메뉴에 고득점 표 추가'라는 커밋 메시지를 남기면 '메뉴 업데이트'라는 메시지를 남겼을 때보다 해당 변경 사항을 찾기 쉬울 것입니다.

JIRA나 GitLab 같은 작업 티켓팅 시스템으로 작업하는 경우 커밋에 티켓 번호를 추가하면 더 좋습니다. 많은 시스템에서 스마트 커밋 기능을 사용할 수 있으며, 이 기능을 사용하면 커밋 메시지를 통해 티켓을 참조하고 상태를 변경할 수 있습니다.

예를 들어 'JRA-123 #close #comment task completed'라고 커밋하면 JRA-123라는 JIRA 티켓을 종료하고 티켓에 'task completed'라는 메시지를 남깁니다.

이 워크플로를 설정하는 방법을 자세히 알아보려면 JIRA 기술 자료 또는 GitLab의 Pivotal Tracker 서비스를 참고하세요.

무분별한 커밋 피하기

모든 변경 사항을 커밋하는 Git 커맨드 'commit -a' 또는 비슷한 커맨드는 프로젝트의 첫 커밋에만 사용해야 합니다. 보통 이 시점에는 프로젝트에 README.md 파일만 있습니다.

커밋에는 저장소에 커밋하려는 변경 사항과 관련된 파일만 포함해야 합니다. Unity 프로젝트 작업에서는 특히 주의해야 합니다. 한 변경 사항으로 인해 변경할 의도가 없었던 씬, 프리팹, 스프라이트 아틀라스 등 여러 파일이 변경되었다고 표시될 수 있기 때문입니다.

다른 사람이 작업 중인 씬의 변경 사항을 실수로 커밋하면 그 사람이 자신의 변경 사항을 커밋하려 할 때 여러분의 변경 사항을 먼저 병합해야 하는 골치 아픈 일이 발생할 수 있습니다.

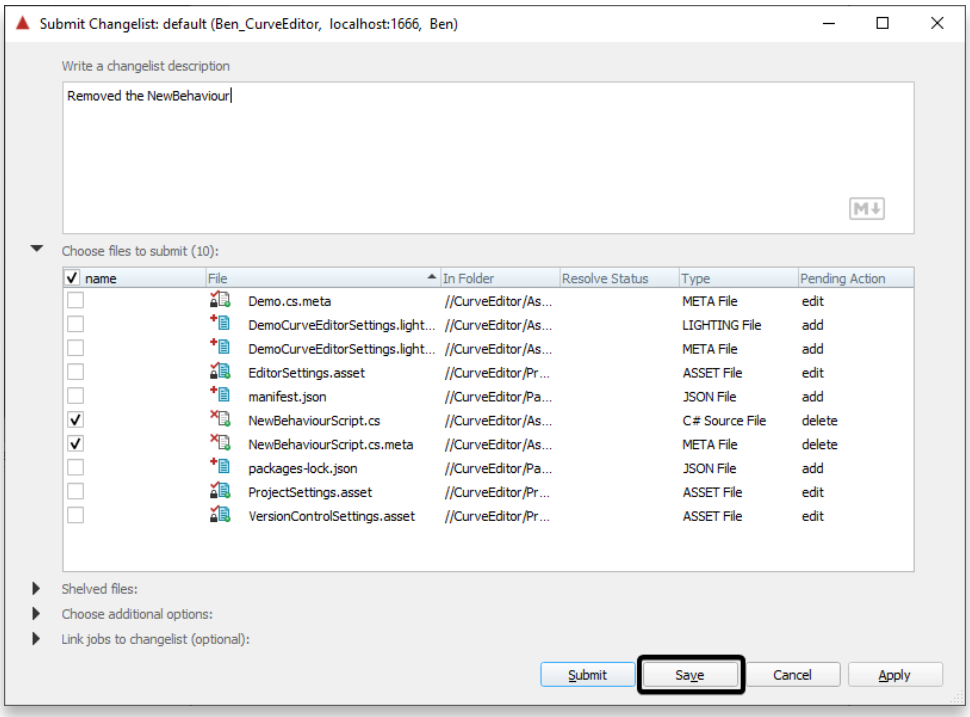
버전 관리를 처음 접하는 사람이 저지르는 흔한 실수 중 하나입니다. 프로젝트에서 자신이 변경한 사항만 커밋해야 한다는 점을 이해하는 것이 중요합니다. 자세히 알아보려면 워크플로 속도를 향상하는 방법에 관한 이 블로그 게시물을 참고하세요.

최신 변경 사항 가져오기

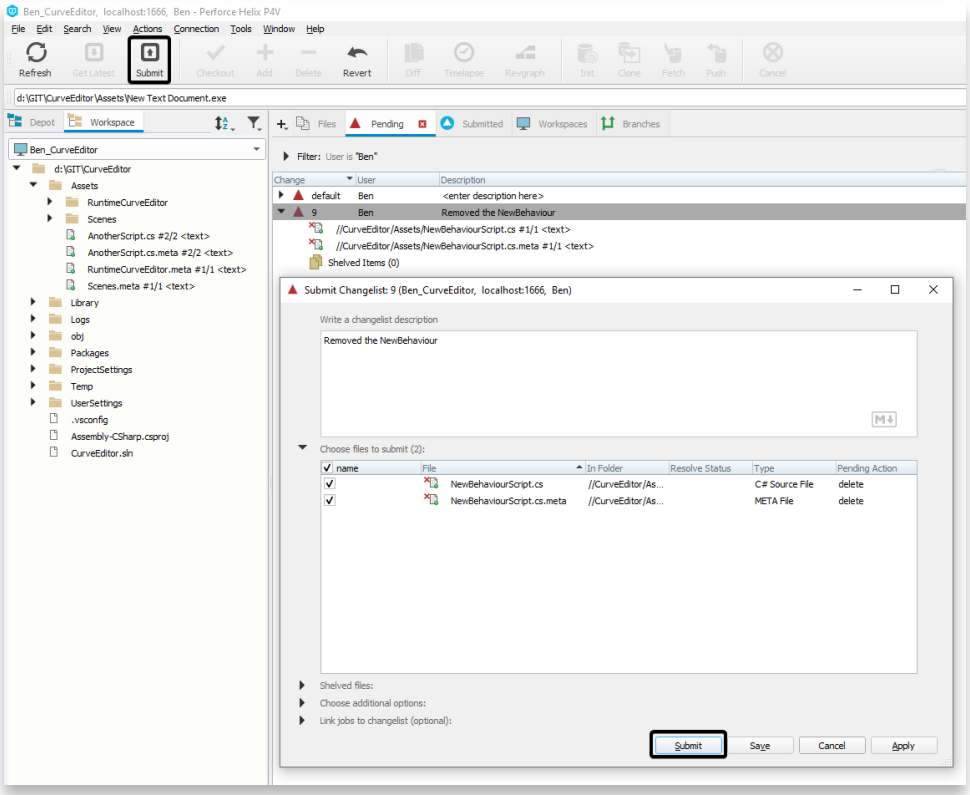
가능한 한 자주 저장소에서 작업 복사본으로 최신 변경 사항을 풀링해야 합니다. 고립된 상태로 작업하는 것은 병합 충돌의 가능성만 늘어나게 되므로 바람직하지 않습니다. 시스템별 일반적인 일일 워크플로는 다음과 같습니다.

Git	Perforce

— Git에서 풀링
— 원하는 만큼 다음 과정을 반복:
  — 작업 복사본에서 파일 수정
  — 변경 사항을 Git으로 커밋
  — Git에서 최신 변경 사항 풀링
— 커밋의 체인지 세트가 만족스러운 경우:
  — Git에서 한 번 더 풀링
  — Git 저장소로 커밋 푸시
	— 최신 변경 사항 가져오기
	— 작업할 파일 체크아웃
	— 파일 수정
	— 변경 사항 제출



P4V에서 새 체인지 세트에 변경 사항 저장



P4V에서 체인지 세트 제출

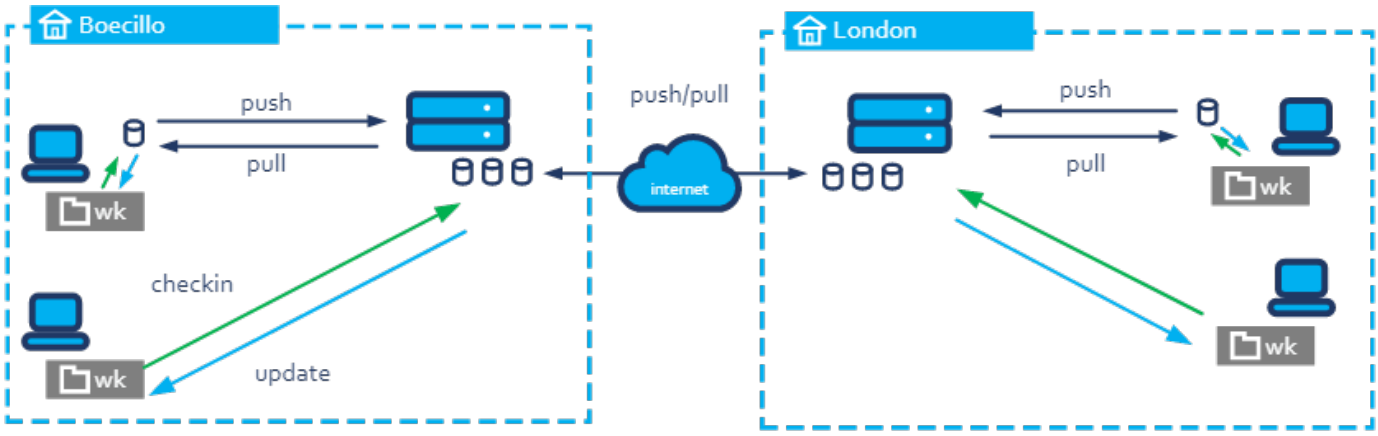
UVCS는 중앙 집중형, 분산형, 멀티사이트 설정으로 작업할 수 있고 각자 조금씩 다른 워크플로를 가집니다.

UVCS(중앙 집중형)	UVCS(분산형)	UVCS(멀티사이트)
— 저장소 동기화 — 표시되는 파일 풀링 — 작업할 파일 체크아웃 — 파일 수정 — 변경 사항 체크인 — 저장소 동기화 — 표시되는 파일 푸시	— 서버에서 변경 사항 풀링 — 로컬 복사본에 변경 사항 체크인 — 새로운 변경 사항 풀링 — 서버에 변경 사항 다시 푸시	— 설정에 따라 두 방식 혼합

멀티사이트 설정은 사용자별 요구 사항에 따라 조정할 수 있으며, 각 사용자가 중앙 집중형 또는 분산형 워크플로로 작업할 수 있습니다.

다음은 두 팀이 협업하는 예시입니다.

- 각 팀에 사내 서버가 있습니다.
- 두 팀의 팀원 모두 로컬이나 분산된 서버에 체크인하며 가까운 사내 서버의 빠른 속도를 활용합니다.
- 서버는 상호 간에 푸시 및 풀링을 통해 전체 또는 부분적으로 동기화를 유지합니다.



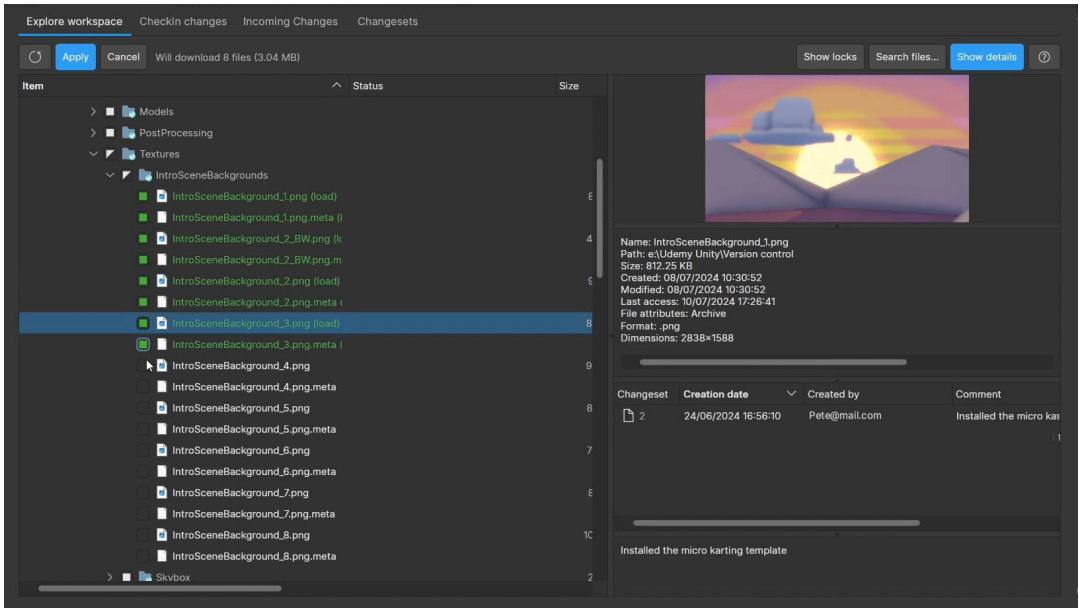
멀티사이트 UVCS 구성

툴셋 이해하기

어떤 VCS를 선택하건 팀이 편안하게 사용할 수 있고 시각화 클라이언트를 포함하여 어떤 툴을 사용할 수 있는지 이해해야 합니다.

Git을 사용한다면 모두 같은 GUI 클라이언트를 사용할 필요가 없습니다. 하지만 커밋 > 풀링 > 푸시 워크플로에 모두 익숙하고 필요한 파일만 커밋하는 방법을 알고 있는지 확인해야 합니다.

UVCS를 사용한다면 아티스트가 워크플로를 간소화할 수 있는 Gluon 에 익숙해지도록 도움을 줘야 합니다. Gluon을 사용하면 작업할 파일만 선택해 다운로드할 수 있으므로 프로젝트 전체를 다운로드하고 관리할 필요가 없습니다. 다른 팀원이 작업할 수 없도록 파일을 잠글 수 있고, 작업을 완료하면 파일을 저장소에 제출하고 다시 잠금을 해제할 수 있습니다.

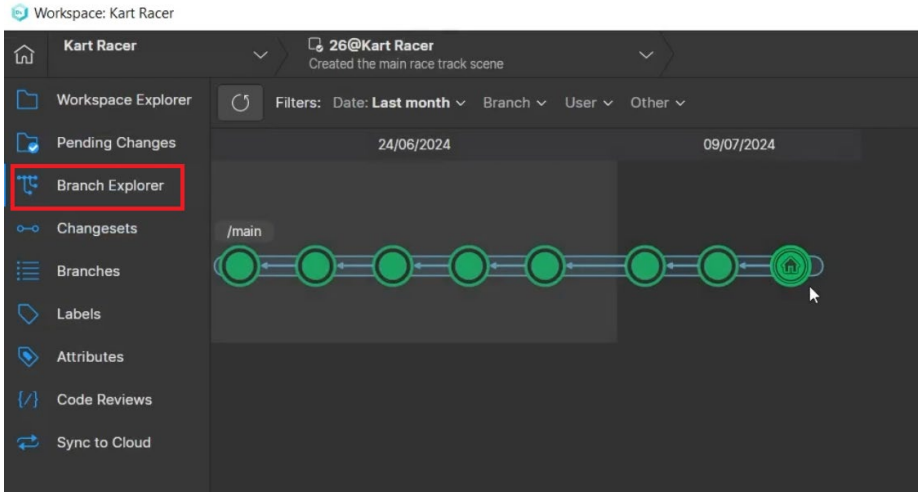


UVCS의 Gluon

Perforce Helix Core를 사용하면 빌트인 Unity 에디터 툴로 에디터에서 직접 버전 관리를 처리할 수 있습니다. 아티스트를 포함해 씬, 프리팹 등 일반적인 Unity 에셋 파일을 다루는 팀원 모두에게 매우 유용한 툴입니다. 에디터에서 수정할 에셋을 체크아웃하고 변경한 다음, 다시 체크인하는 과정을 Unity 내에서 전부 수행할 수 있습니다.

기능 브랜치와 Git Flow

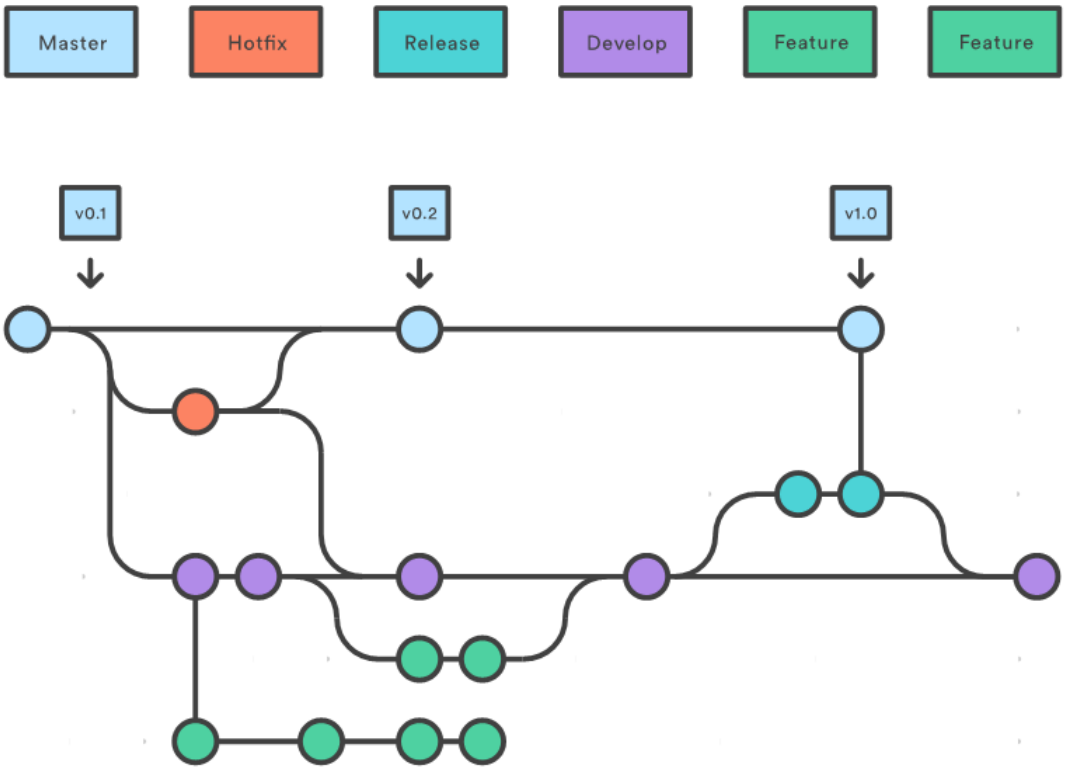
여러 릴리스 주기를 거치는 장기 프로젝트를 작업하는 경우, 기능 브랜치를 활용하면 워크플로에 큰 도움이 됩니다. 보통은 트렁크(trunk), 마스터(master), 메인(main)이라고 하는 저장소 내의 하나의 브랜치에서 작업하는 경우가 많습니다. 이렇게 하면 전체 프로젝트가 동일한 타임라인에서 진행됩니다.



UVCS의 메인 브랜치에서 개발 진행

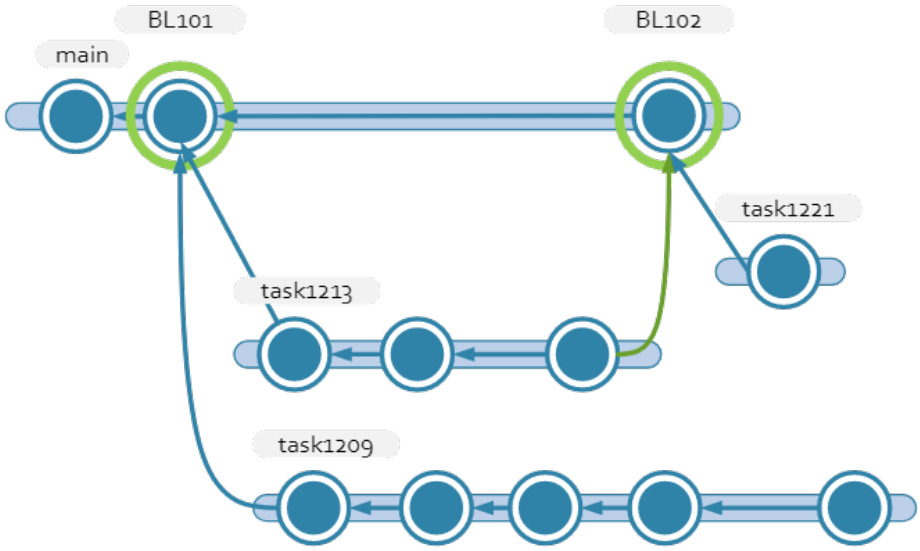
하지만 작업을 여러 브랜치로 분할하여 진행하면 팀 단위 작업에서 더 효과적일 수 있습니다.

Git에는 기능, 버그 수정, 릴리스와 같은 여러 브랜치로 나눠 작업을 진행하는 Git Flow라는 워크플로가 있습니다. 개발자는 별도의 브랜치에서 새 기능 작업을 시작한 다음, 작업을 완료하면 메인 브랜치에 병합합니다. 그동안 다른 팀원이 이전 릴리스에 핫픽스를 적용하고, 버그를 수정하고, 새로운 버전을 안전하게 출시했을 수 있습니다. 이때 아직 개발 중인 어떤 기능도 이 과정에 영향을 받지 않았을 것입니다.



릴리스를 쉽게 관리할 수 있는 Git Flow 워크플로

UVCS는 작업 브랜치 기능도 제공합니다. 이 패턴에서는 추적할 모든 작업에 새 브랜치를 생성합니다. Git Flow는 기능 브랜치로 완전하고 규모가 큰 기능을 개발하지만 UVCS의 작업 브랜치는 단기적으로 사용하는 것이 목적입니다. 하나의 작업을 구현하는 데 커밋이 여러 번 필요하다면 더 작은 작업으로 나눌 가능성이 큽니다.

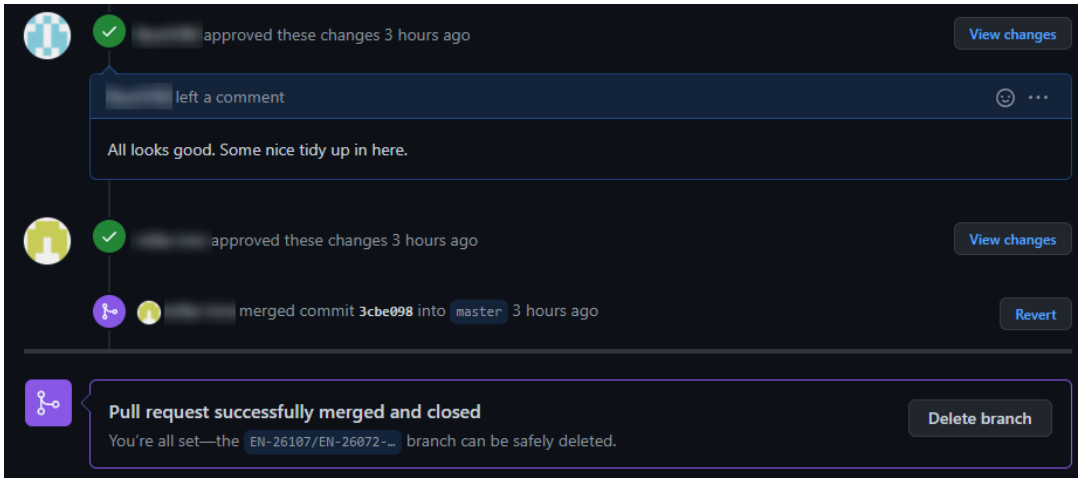


작업 패턴별 UVCS 브랜치

Perforce Helix Core에서는 스트림 시스템을 사용하여 이런 스타일의 워크플로를 구현합니다. 작업할 디포를 생성할 때 스트림 디포 유형으로 설정해야 합니다. 그런 다음 Stream Graph 뷰에서 새 스트림을 생성할 수 있습니다. 메인라인 스트림을 제외한 모든 스트림에는 부모 스트림이 있으므로 변경 사항을 부모 스트림으로 복사할 수 있습니다.

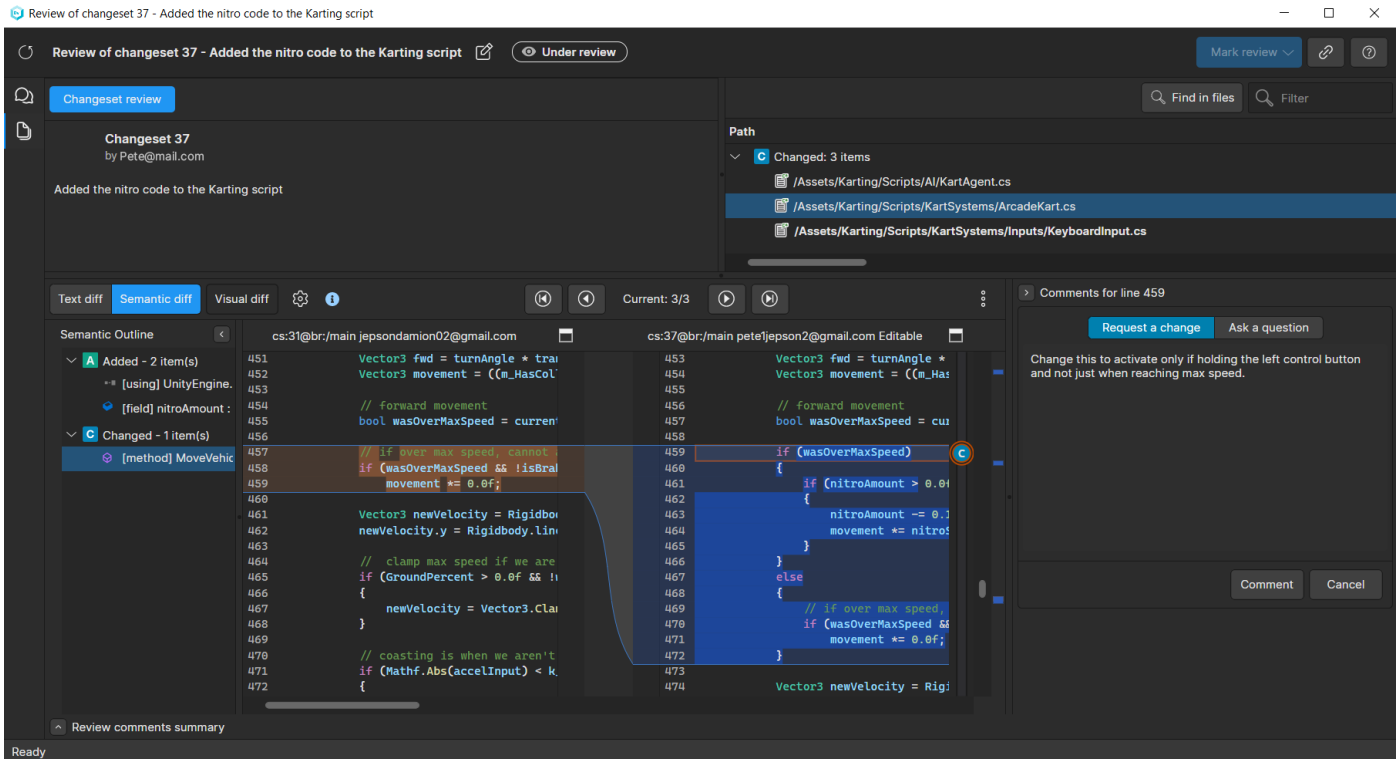
풀 리퀘스트

기능 브랜치에서 작업을 완료한 다음에는 저장소의 메인 스트림에 변경 사항을 적용하기 위해 풀 리퀘스트를 사용하는 것이 바람직합니다. 풀 리퀘스트는 기능 또는 작업을 완료한 개발자가 생성하며, 일반적으로는 시니어 개발자 또는 DevOps가 변경 사항을 메인라인에 적용하기 전에 검토해야 합니다.



GitHub의 종료된 풀 리퀘스트

UVCS와 Perforce 모두 메인라인으로 브랜치를 병합하는 작업을 관리하는 자동화 툴을 제공합니다. UVCS는 저장소의 검토 및 확인된 브랜치를 자동으로 병합하는 Mergebot의 도움으로 이 작업을 수행합니다. Perforce는 코드 검토를 관리하는 추가 플랫폼 Helix Swarm을 제공하며, 자동 테스트를 설정할 수도 있습니다.



GUI에 포함된 UVCS 코드 검토 기능

Unity 6에서 UVCS 시작하기

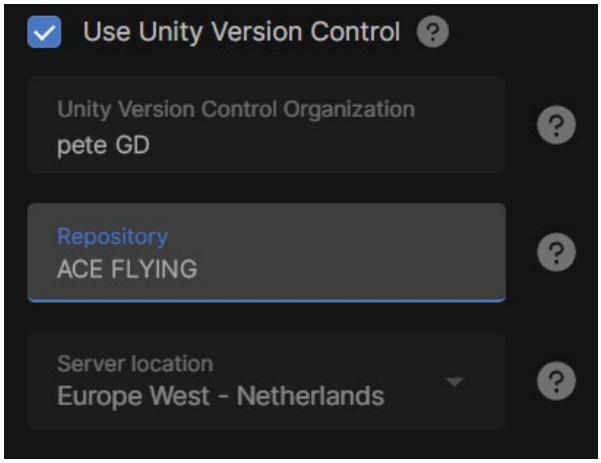
Unity DevOps 제품은 UVCS와 Build Automation으로 구성되어 있습니다. UVCS는 최대 3명의 팀원(시트)이 무료로 사용할 수 있으며 매월 최대 5GB의 데이터가 제공됩니다. 그 이후에는 월간 액티브 사용자 수와 총 클라우드 스토리지에 따라 가격이 달라집니다. UVCS는 모든 UVCS 서비스에서 허용된 사용량의 50%, 75%, 90%에 도달할 때마다 UVCS 조직 소유자에게 경고 이메일을 전송하므로 항상 사용량을 간편하게 파악할 수 있습니다.

자세히 알아보려면 Unity Cloud 플랜 가격 페이지를 확인하거나 Unity Cloud 대시보드에 로그인하고 DevOps 대시보드의 'About' 페이지를 참고하세요.

UVCS는 3가지 방법으로 사용할 수 있습니다. UVCS 데스크톱 클라이언트를 통해 여러 애플리케이션과 저장소를 사용하거나, Unity Hub를 통해 프로젝트에 추가하거나, 웹 브라우저를 통해 Unity Cloud의 저장소에 액세스할 수 있습니다.

Unity 프로젝트에서 UVCS 사용하기

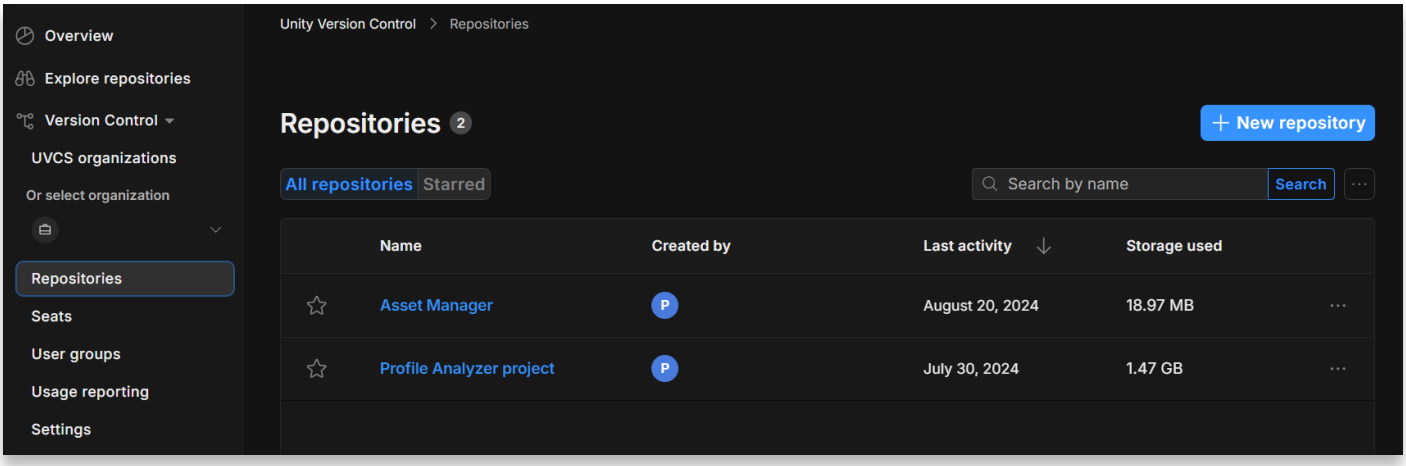
Unity Hub에서 프로젝트를 생성하면서 **Version Control** 체크박스를 선택하면 Unity가 자동으로 저장소를 설정합니다.



Unity Hub에서 Version Control 설정

이렇게 하면 프로젝트의 데이터와 파일이 로컬 기기와 클라우드에 모두 저장되어 협업과 백업을 동시에 진행할 수 있습니다. 저장소 이름은 프로젝트 이름과 동일하게 설정되며, 가장 가까운 서버를 선택해 빠르게 연결할 수 있습니다.

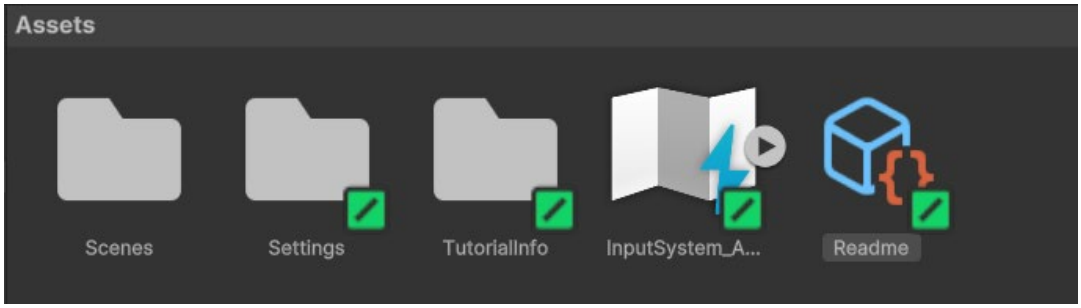
Assets, Packages, Project Settings 폴더는 처음 사용 시 서버에 백업됩니다. Unity Cloud에 로그인하고 저장소에서 **File Explorer**를 선택하여 파일에 액세스할 수 있습니다. 클라우드 저장소에 저장된 모든 파일이 표시됩니다.



저장소 내 파일과 폴더의 구조를 표시하는 Unity Cloud 대시보드의 File Explorer

UVCS는 컴퓨터에 자동으로 생성되고 많은 용량을 차지하는 Library, Logs, User Settings 폴더를 무시합니다.

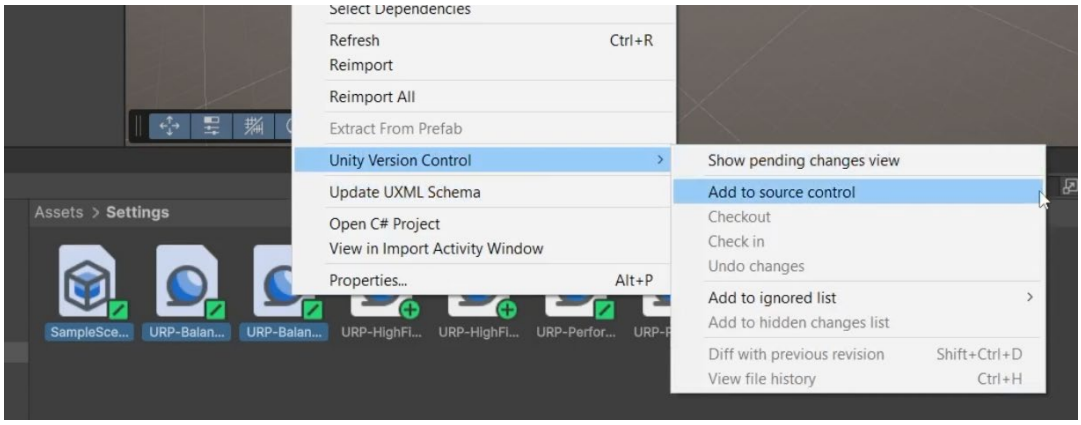
UVCS에서 무시하는 폴더와 파일을 프로젝트 창에서 확인해 보면 해당 아이콘 밑에 초록색 상자와 슬래시 아이콘이 표시됩니다.



클라우드에 백업되지 않도록 무시하는 파일은 밑에 초록색 상자와 슬래시 아이콘이 표시됩니다.

프로젝트에서 URP 또는 HDRP를 사용한다면 Version Control 창의 **Pending Changes** 섹션에 Settings 폴더를 포함해야 합니다. Settings 폴더에 초록색 상자와 슬래시 아이콘이 표시된다면 해당 폴더를 선택하고 오른쪽 클릭한 다음, **Version Control > Add to source control**을 선택합니다.

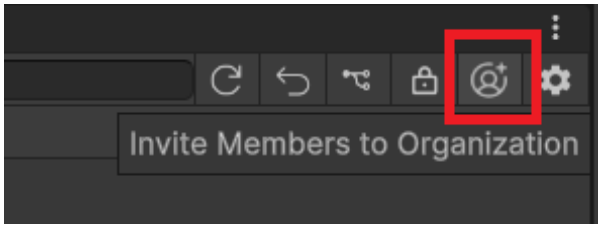
슬래시 아이콘이 초록색 원과 더하기 아이콘으로 바뀌고 해당 파일은 보류 중인 변경 사항 목록에 추가되어 클라우드에 업로드될 준비를 마칠 것입니다.



Pending Changes 뷰에 Settings 파일 추가

다른 팀원 초대

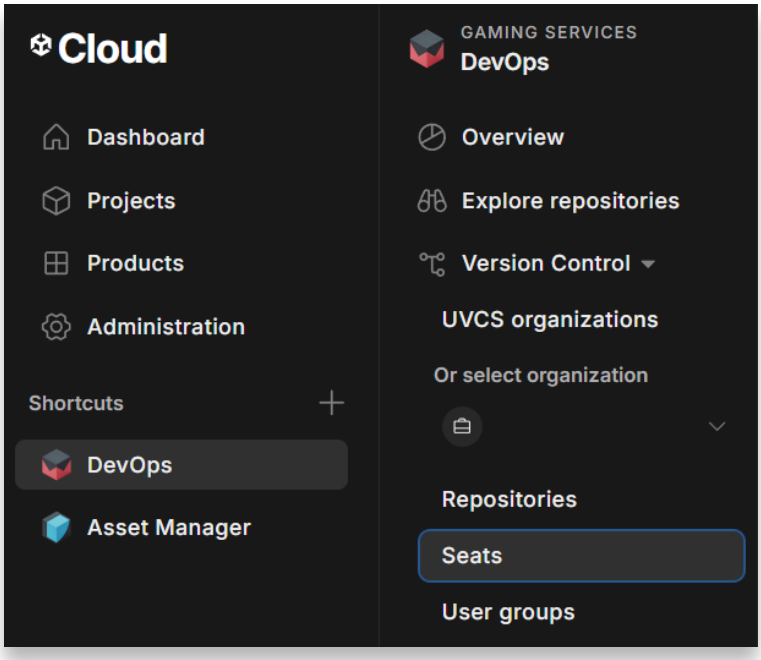
Version Control 창 우측에 팀원을 초대할 수 있는 버튼이 있습니다.



Version Control 창의 **Invite Members to Organization** 버튼

버튼을 클릭하면 Unity Cloud 로그인 페이지로 이동됩니다. Unity ID 및 비밀번호로 로그인합니다.

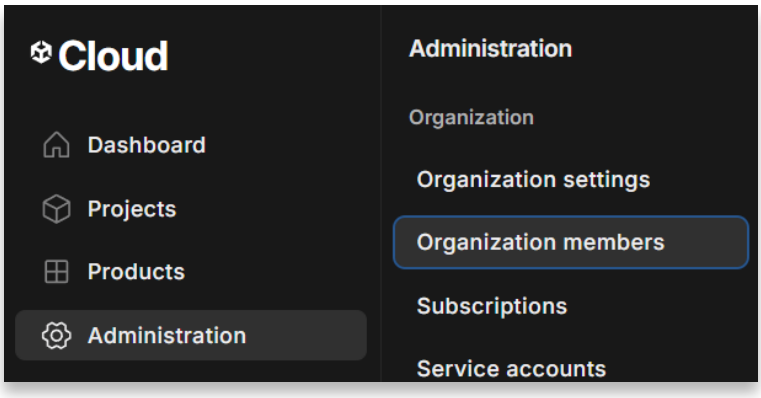
왼쪽의 Shortcuts 섹션에서 DevOps가 열리고 Seats 섹션이 열리면 현재 프로젝트에 시트를 추가할 수 있습니다. 이 가이드 작성 시점을 기준으로 무료 티어에서는 최대 3개의 시트까지 추가할 수 있습니다. Cloud Pro로 업그레이드하면 시트를 추가로 구매할 수 있습니다. 자세히 알아보려면 Unity Cloud 플랜 가격 페이지를 확인하거나 Unity Cloud 대시보드에 로그인하고 DevOps 대시보드의 'About' 페이지를 참고하세요.



Unity Cloud의 DevOps 섹션에서 시트 추가

프로젝트 시트에 새 팀원을 할당합니다. 그리고 조직 구성원으로 초대도 해야 합니다. 그러면 해당 팀원은 사용량 통계를 비롯한 클라우드 대시보드의 모든 데이터에 액세스할 수 있습니다.

Administration > Organization members로 이동하고 **invite organization members** 버튼을 클릭하여 조직에 구성원을 추가합니다.



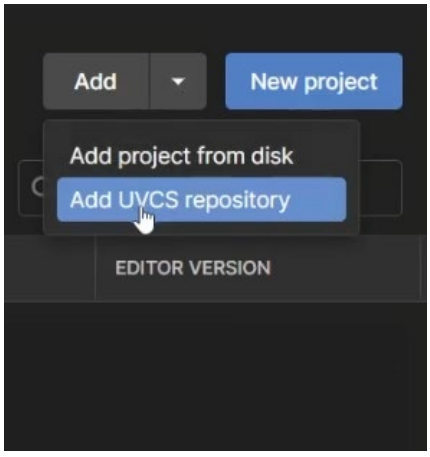
Administration 섹션에서 조직 구성원을 추가합니다.

각 구성원에게 적절한 역할을 할당합니다. 게스트, 사용자, 관리자 중에 선택할 수 있습니다. 사용자와 관리자 구성원은 모든 프로젝트를 확인할 수 있으며, 관리자는 프로젝트 구성원 섹션에서 역할을 할당하여 프로젝트 데이터에 대한 적절한 수준의 액세스 권한을 부여할 수 있습니다.

역할을 할당받은 구성원은 프로젝트에 액세스하고 역할에 따라 기여할 수 있습니다. 관리자 상태의 사용자는 클라우드에서 제공되는 모든 옵션을 이용할 수 있습니다.

초대된 구성원에게는 이메일이 전송됩니다.

초대받은 구성원은 Unity Hub를 열고 **Add** 버튼 옆의 드롭다운을 클릭한 뒤 **Add UVCS repository**를 선택합니다.



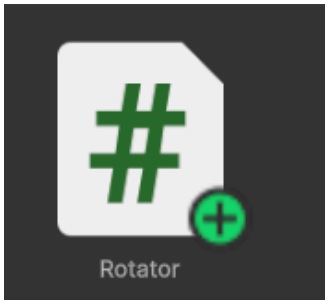
Add UVCS repository를 클릭하여 Unity Cloud에서 프로젝트를 엽니다.

참고: 모든 팀원이 같은 버전의 Unity를 사용해야 합니다. 그러지 않으면 프로젝트 버전을 전환할 때 문제가 발생할 수 있습니다.

이렇게 하면 클라우드 저장소에서 사용자의 컴퓨터로 프로젝트가 다운로드됩니다.

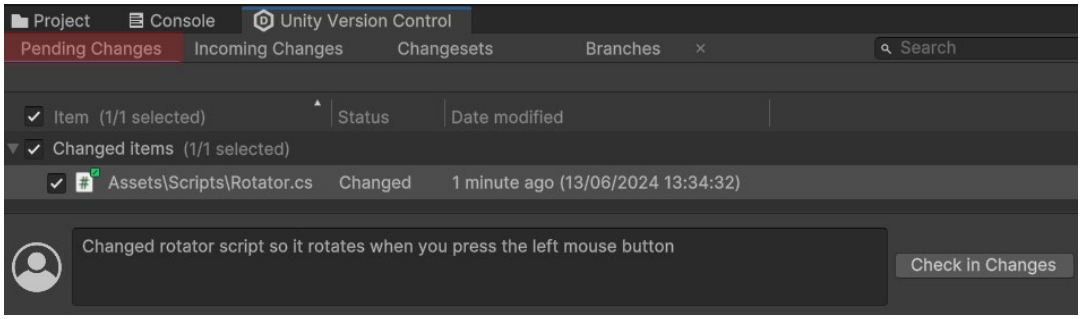
변경 사항 체크인

프로젝트를 수정하면 변경된 파일에 초록색 **+** 아이콘이 표시됩니다. 이 아이콘은 서버에 체크인하여 업데이트할 파일을 나타냅니다.



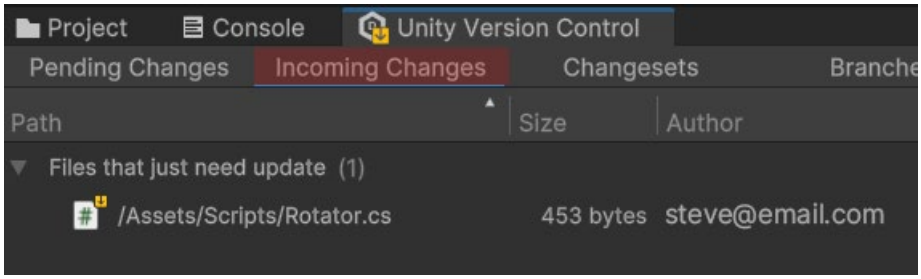
초록색 원과 더하기 아이콘은 변경되어 체크인해야 할 파일을 나타냅니다.

쉽게 롤백할 수 있도록 프로젝트를 자주 업데이트하세요. 업데이트된 파일은 Version Control 창의 **Pending Changes** 섹션에 표시됩니다. 변경 사항을 짧고 간결하게 설명하는 메시지를 추가하고 **Check in Changes** 버튼을 클릭합니다.



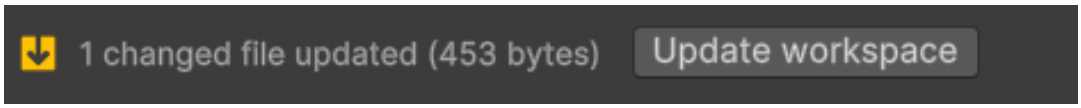
Pending Changes 섹션에서 프로젝트의 변경 사항을 체크인합니다.

동일한 브랜치에서 작업하는 다른 사용자는 Version Control 창의 **Incoming Changes** 탭에서 노란색 다운로드 아이콘이 표시될 것입니다. 이 아이콘은 다른 사용자가 변경한 사항을 나타냅니다.

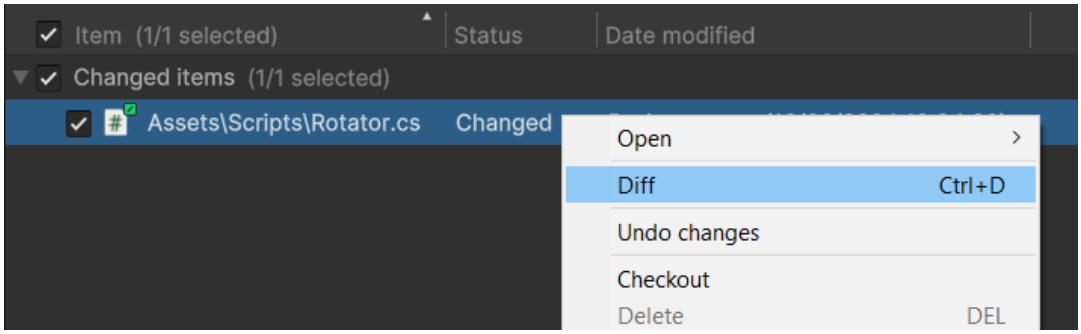


다른 사용자는 내가 체크인한 변경 사항을 확인하고 작업 공간을 업데이트하여 변경 사항을 반영할 수 있습니다.

다른 사용자는 **Update workspace** 버튼을 클릭하여 작업 공간을 업데이트하고 변경 사항을 반영할 수 있습니다.

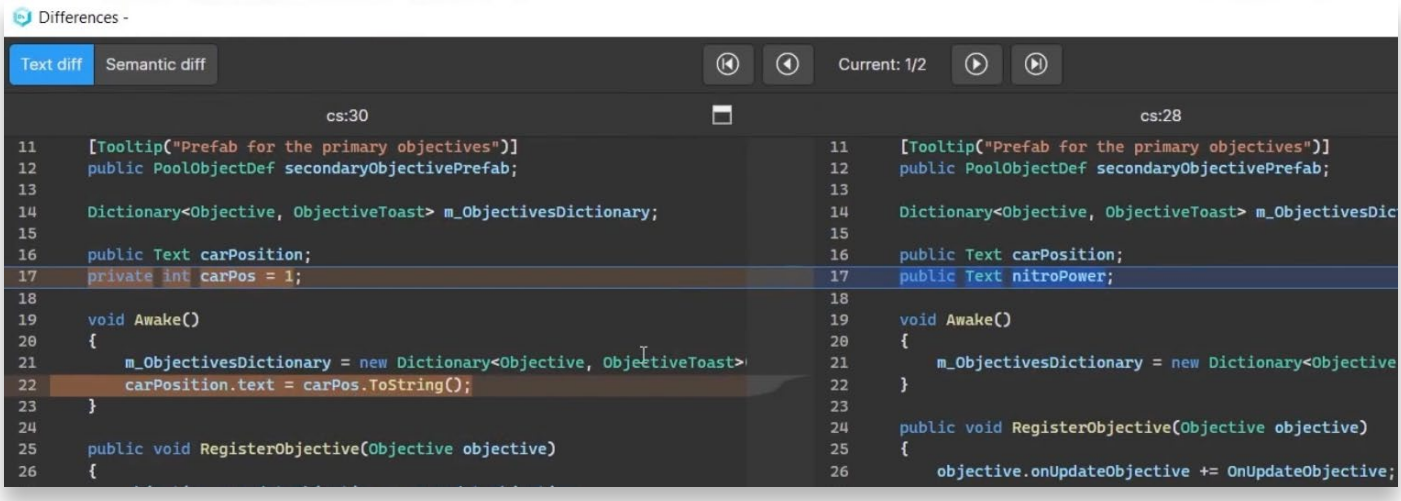


기존에 존재했던 파일이 수정된 경우, 해당 파일을 오른쪽 클릭하고 **Diff**를 선택하여 수정 전후의 차이점을 확인할 수 있습니다.



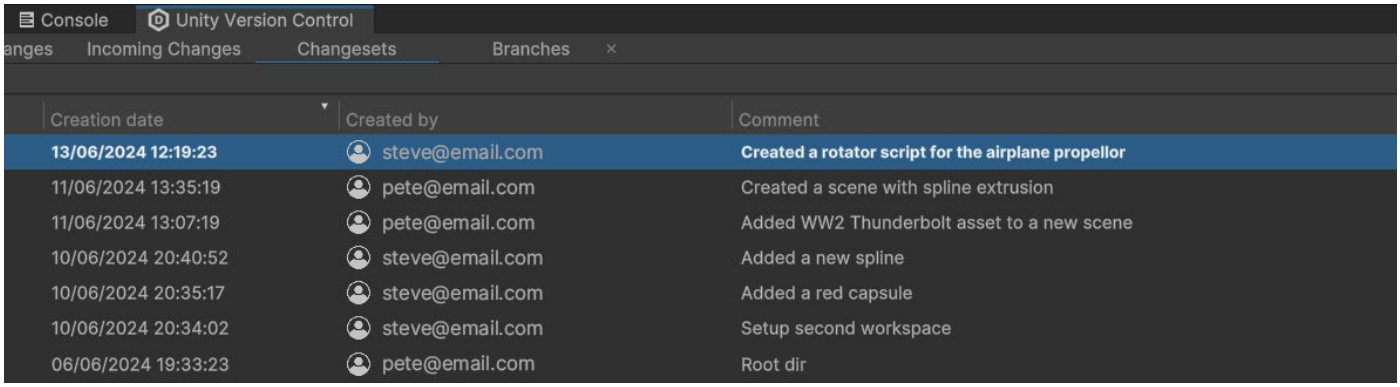
변경된 파일과 원본 파일의 차이점 확인

Diff 뷰어의 왼쪽에 표시되는 원본 버전과 오른쪽에 표시되는 변경 사항을 비교할 수 있습니다. 이 기능은 C# 스크립트에 유용합니다. Unity의 코드 인식 병합 기술인 Semantic Merge는 이동한 코드를 추적하므로 해당하는 변경 사항에만 집중할 수 있습니다.



Diff 뷰어는 원본 파일을 왼쪽에 주황색으로, 변경된 파일을 오른쪽에 파란색으로 표시합니다.

Changesets 섹션에서는 모든 체인지 세트 목록을 확인할 수 있으며, 수정한 작업자와 시점도 표시됩니다. 체인지 세트 목록에서 변경 사항에 관한 설명과 업데이트된 파일도 확인할 수 있습니다.

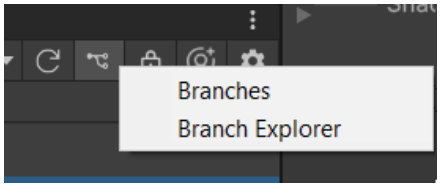


프로젝트에 적용된 모든 체인지 세트가 표시됩니다. 원하는 체인지 세트로 롤백할 수 있습니다.

UVCS 데스크톱 클라이언트

Unity 에디터의 Version Control 창은 클라우드에서 데이터를 푸시하고 풀링하기 위해 필요한 기본적인 기능을 제공합니다. 데스크톱 클라이언트는 추가 기능을 제공합니다.

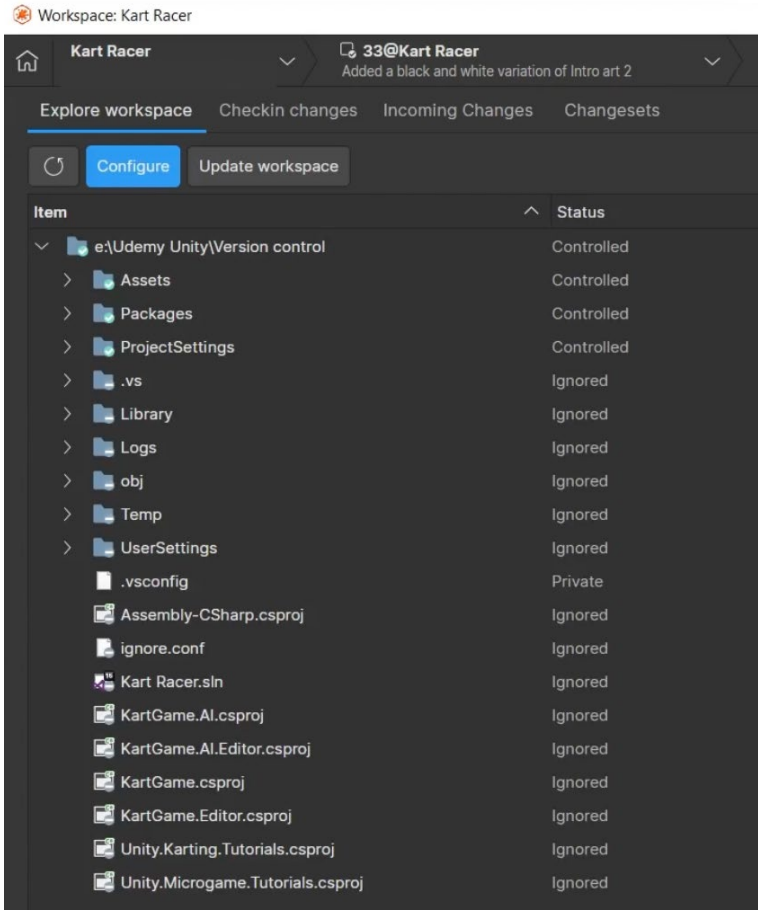
데스크톱 클라이언트를 실행하려면 확장 모드에서 작업하면서 **Branch Explorer**를 클릭합니다. 아직 클라이언트를 설치하지 않았다면 설치하라는 메시지가 표시됩니다.



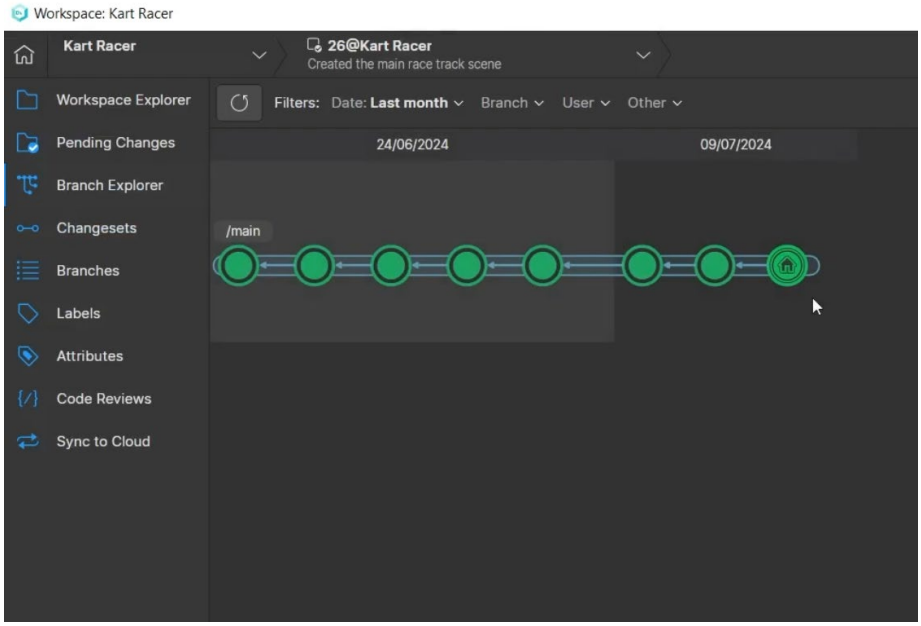
Branch Explorer 버튼을 클릭하여 UVCS 데스크톱 앱 실행

버전 관리 앱인 Gluon과 Unity DevOps Version Control(UVCS 데스크톱 앱)

Gluon은 초보자 친화적인 소프트웨어이며 UVCS는 모든 기능을 갖춘 소프트웨어입니다. Branch Explorer를 클릭하면 UVCS 소프트웨어가 열립니다. Gluon이나 UVCS로 전환하려면 컴퓨터에 설치된 소프트웨어에서 열면 됩니다. Unity는 사용 중인 버전을 자동으로 탐지하여 해당 시스템을 사용하도록 전환합니다.



Gluon은 아티스트 친화적인 버전 관리 앱입니다.



UVCS는 완전한 기능을 갖춘 소프트웨어입니다.

Gluon 사용하기

앞서 UVCS를 소개하는 섹션에서 설명한 대로 Gluon은 아티스트를 염두에 두고 설계된 가벼운 클라이언트입니다. 작업할 파일만 선택하여 서버에서 체크아웃하고 다른 팀원이 수정하지 못하도록 잠그는 기능을 제공합니다. 작업을 완료하면 파일을 다시 체크인하면 됩니다. Gluon GUI는 기술적인 지식이 부족한 사용자보다 프로그래머에게 더 도움이 되는 복잡한 개념을 없앤 인터페이스입니다.

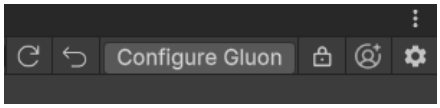
설치된 Unity DevOps Version Control 애플리케이션에서 Gluon을 엽니다.

설치된 Unity DevOps 소프트웨어에서 Gluon 데스크톱 앱을 찾을 수 있습니다.

전체 모드에서 Gluon이 실행되는 부분 모드로 전환하라는 메시지가 표시됩니다.

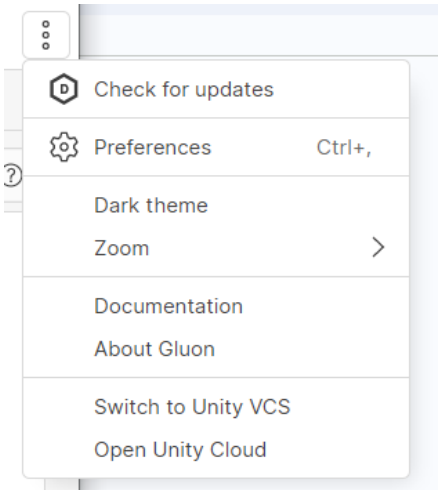
이 경고 메시지에서 **Run an update**를 클릭하여 부분 모드로 전환하면 Gluon으로 작업할 수 있습니다.

Unity에서는 **Configure Gluon** 버튼이 **브랜치** 아이콘을 대체합니다. 버튼을 클릭하면 항상 Gluon 앱이 열립니다.



이제 Gluon을 사용하는 팀원이 Unity의 Version Control 창에서 직접 Gluon 앱을 열 수 있습니다.

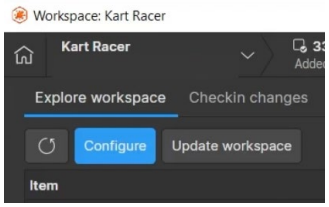
Gluon 앱 우측 상단의 옵션 버튼을 클릭하여 어두운 테마를 활성화할 수 있습니다.



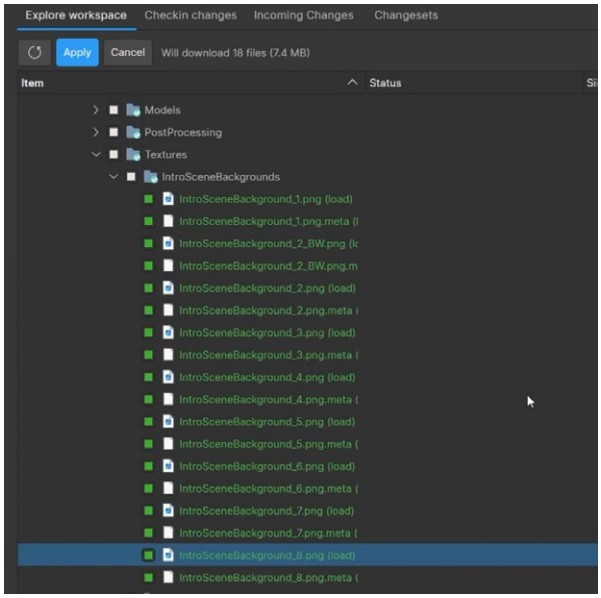
Gluon에서 우측 상단의 옵션 드롭다운을 사용하여 어두운 테마를 활성화할 수 있습니다.

Gluon은 메인 브랜치만 사용하므로 시각적인 아티스트 친화적 작업 공간을 제공합니다. Gluon의 **Explore workspace** 기능을 사용하면 필요한 파일을 손쉽게 업로드하고 다운로드할 수 있습니다.

Explore workspace 섹션에서 **Configure** 버튼을 클릭하여 저장소의 파일을 수정합니다.



저장소의 파일을 수정하려면 **Configure** 버튼을 클릭합니다.

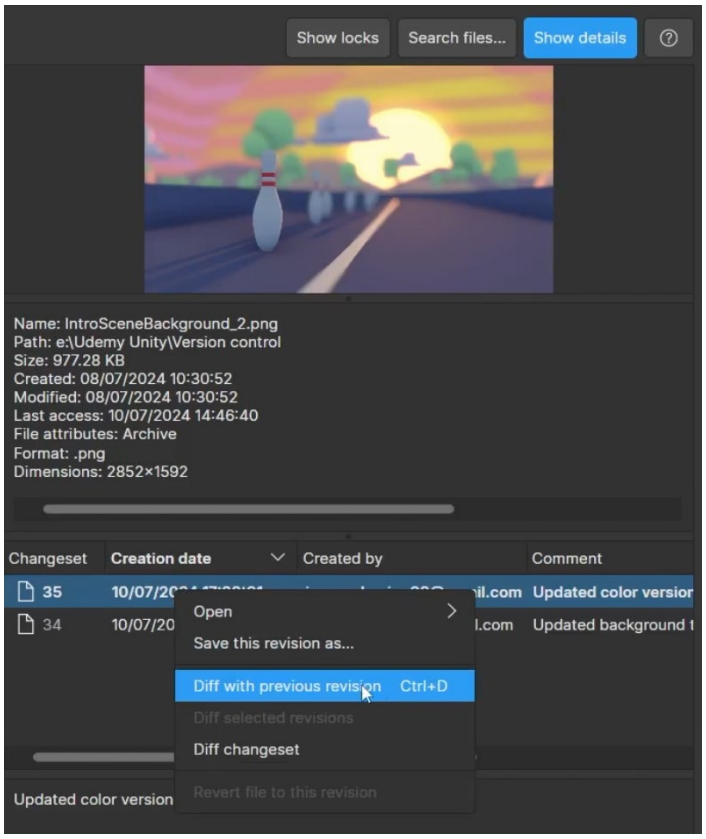


Gluon에서 다운로드할 파일을 선택하고 Apply를 클릭합니다.

파일 옆의 체크박스를 클릭하여 파일을 다운로드하거나 Explore workspace 뷰에서 제거합니다. Unity 에디터로 다운로드될 파일은 초록색으로 변하고 Unity에서 제거할 파일은 빨간색으로 변합니다. 완료되면 **Apply**를 클릭한 다음 **Update workspace**를 클릭하여 Unity에서 파일을 사용할 수 있도록 업데이트합니다.

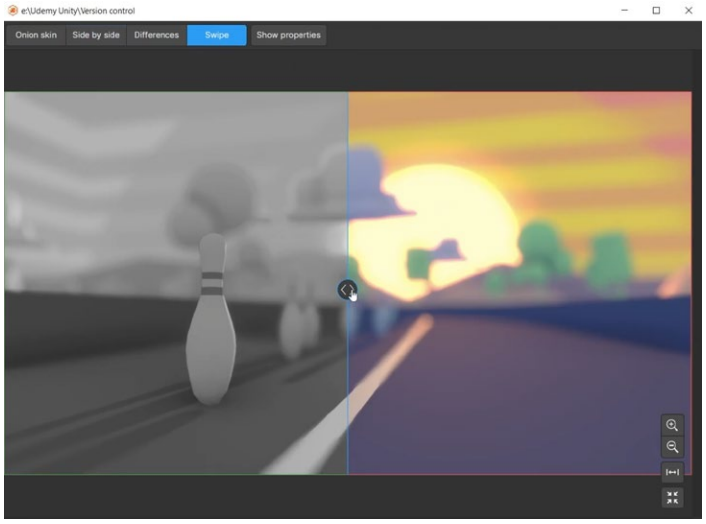
Gluon에서 파일을 삭제하더라도 현재 체인지 세트 및 향후 생성할 체인지 세트에만 반영됩니다. 삭제한 파일을 이전 체인지 세트에서 참조하므로 해당 파일은 계속 클라우드에 보관됩니다.

Gluon 앱 오른쪽의 옵션 섹션에서 여러 차례 수정된 텍스처의 수정본을 오른쪽 클릭하고 **Diff with previous revision**을 선택하여 무엇이 변경되었는지 확인할 수 있습니다.



Diff with previous revision을 선택하여 수정본이 어떻게 변경되었는지 확인합니다.

이를 통해 이전 수정본과 현재 버전의 차이를 확인할 수 있습니다. 이미지를 따라 스와이프 슬라이더를 드래그하여 수정 전후를 비교할 수 있는 스와이프 모드를 비롯한 몇 가지 모드가 제공됩니다.



Diff 뷰어의 스와이프 모드

UVCS 데스크톱 앱

UVCS 데스크톱 앱은 완전한 기능을 갖춘 버전 관리 소프트웨어로, 브랜치 기능을 사용하려는 팀원에게 적합합니다. 설치된 Unity DevOps Version Control 애플리케이션에서 열 수 있습니다.

이전에 Gluon을 사용했다면 부분 모드에서 UVCS가 실행되는 전체 모드로 전환하라는 메시지가 표시됩니다.

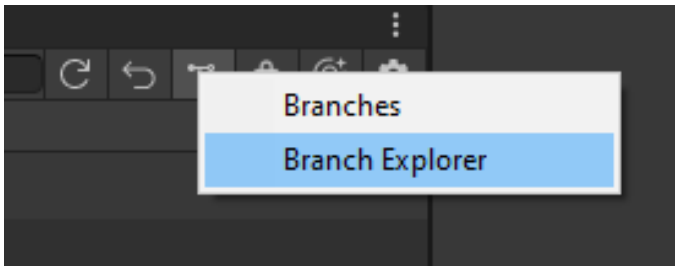
UVCS 데스크톱 버전 관리 소프트웨어를 실행하기 위해 부분 모드에서 전체 모드로 전환합니다.

브랜치

브랜치를 활용하면 서로 다른 팀원이 같은 프로젝트에서 독립적으로 작업할 수 있습니다. 예를 들면 한 팀원이 별도의 브랜치에서 새 기능을 제작하는 동안 다른 팀원은 메인 브랜치에서 작업이 가능합니다. 새 기능은 업적 기능을 위한 코드를 작성하거나 다양한 유형의 픽업과 관련된 메카닉을 추가하는 경우를 예로 들 수 있습니다.

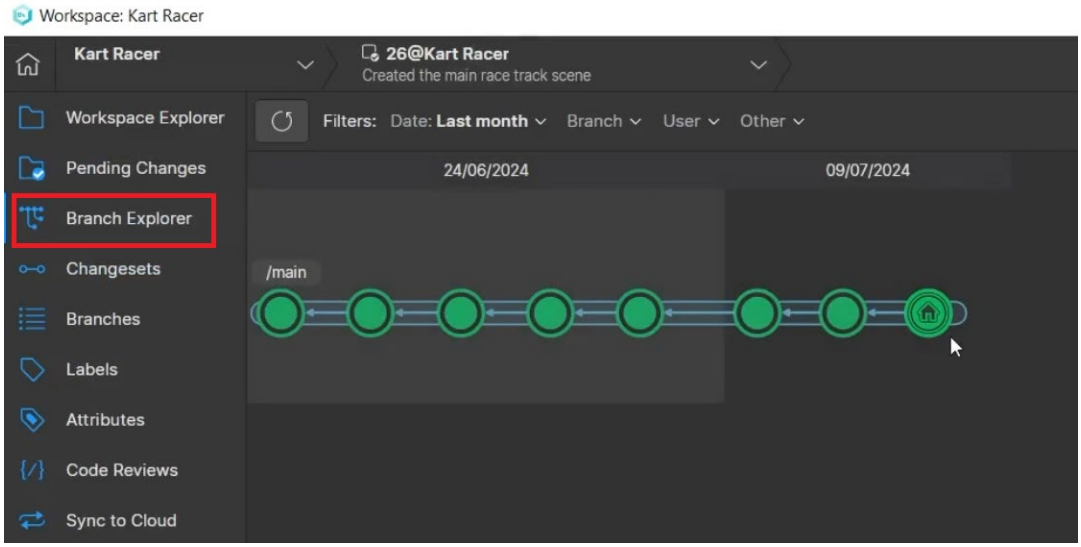
작업을 완료하면 작업한 브랜치를 메인 브랜치에 병합합니다. 그런 다음 프로젝트를 작업하는 모든 팀원이 작업 공간을 업데이트하여 완료된 작업을 반영할 수 있습니다.

Unity 에디터의 Version Control 창에서 Branch Explorer를 선택하면 UVCS 앱이 열리고 Branches 섹션으로 이동됩니다.

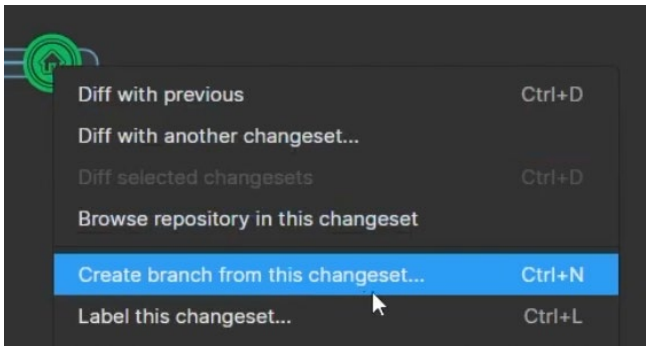


UVCS 데스크톱 앱의 Branch Explorer 섹션을 열 수 있습니다.

각 원은 체인지 세트를 나타냅니다. 현재 체인지 세트에는 집 아이콘이 표시됩니다.



체인지 세트를 오른쪽 클릭하고 **Create branch from this changeset를 선택합니다.**

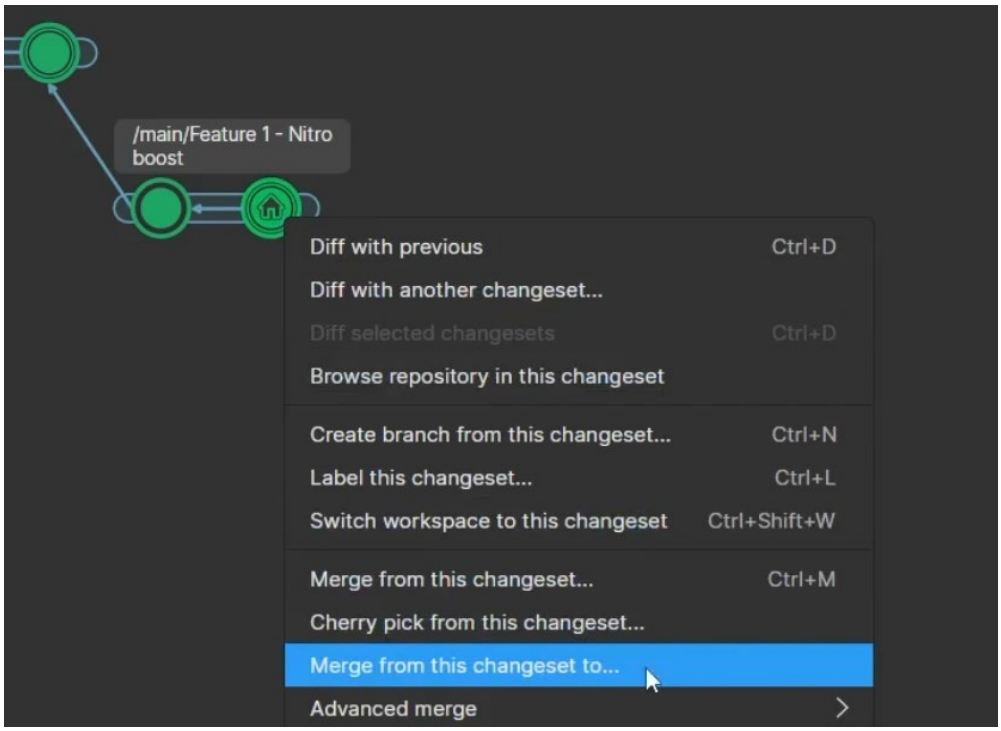


Create branch from this changeset 옵션

이제 별도의 브랜치에서 작업하게 됩니다. 메인 브랜치나 다른 기능 브랜치에서 팀원이 변경한 사항은 이 브랜치에 반영되지 않습니다. 따라서 내가 변경한 사항과 다른 팀원의 변경 사항이 충돌하는 것을 피할 수 있습니다. 해당 변경 사항은 병합한 다음 업데이트할 수 있습니다.

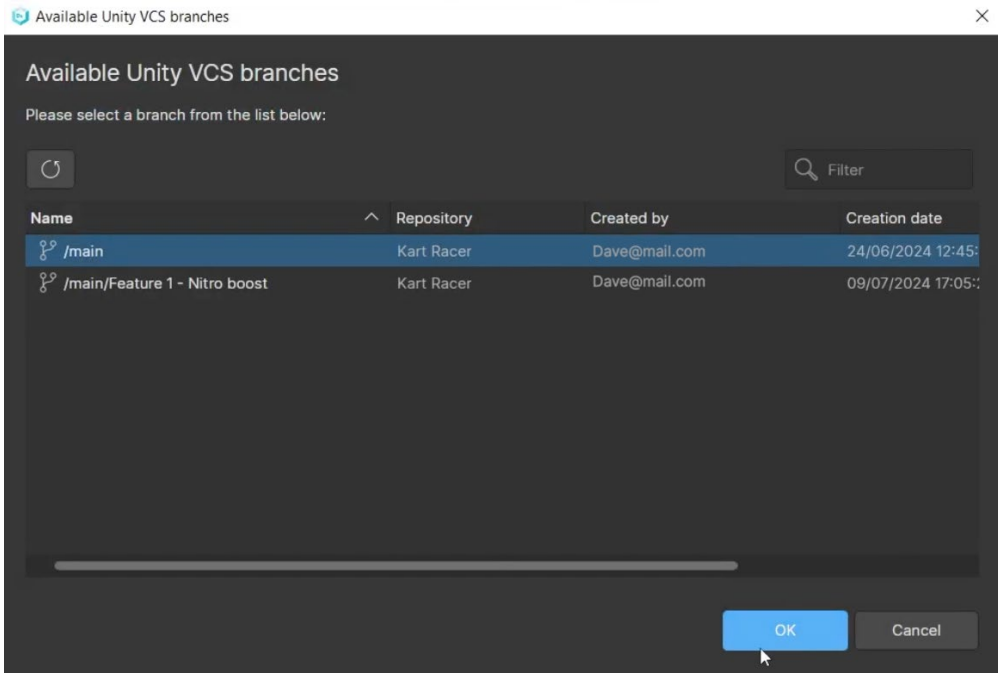
기능 브랜치도 메인 브랜치처럼 원으로 표시된 여러 체인지 세트를 가질 수 있습니다.

브랜치에서 작업을 완료했으면 오른쪽 클릭하고 **Merge from this changeset to…**를 선택하여 메인 브랜치에 병합할 수 있습니다.



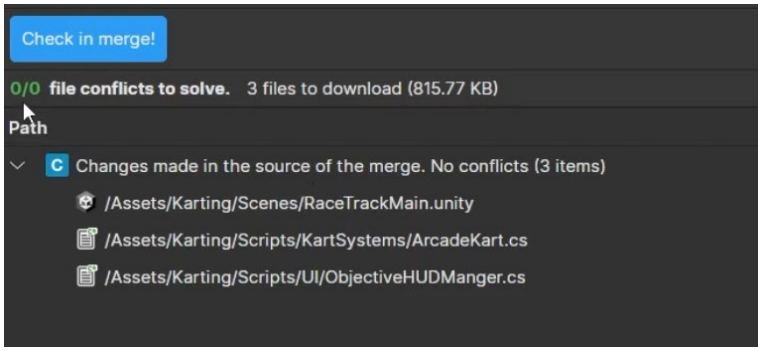
브랜치를 메인 브랜치 등 다른 브랜치에 병합합니다.

병합할 수 있는 브랜치가 표시됩니다. 병합할 브랜치를 선택합니다.



병합할 수 있는 브랜치

병합 섹션이 열리고 충돌이 있는지 표시됩니다. 충돌이 없다면 **Check in merge** 버튼을 클릭합니다.

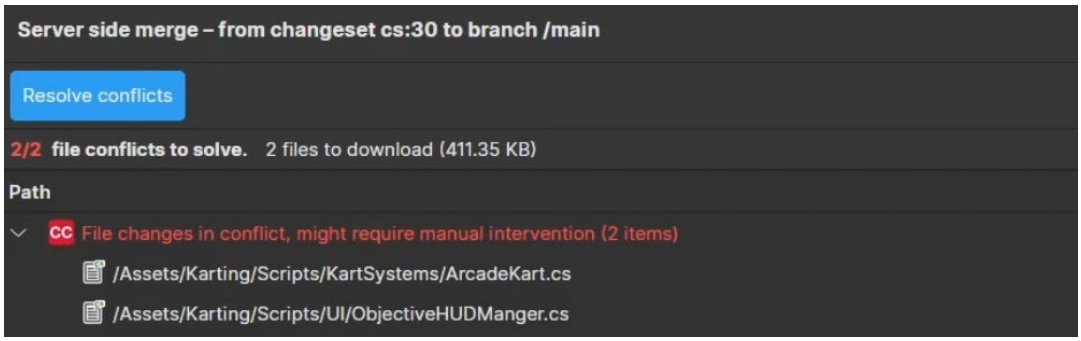


UVCS가 해당 브랜치의 모든 파일이 메인 브랜치와 충돌하지 않는지 확인합니다.

메인 브랜치에서 작업하던 팀원이 같은 파일을 변경했다면 병합할 때 충돌이 발생할 수 있습니다.

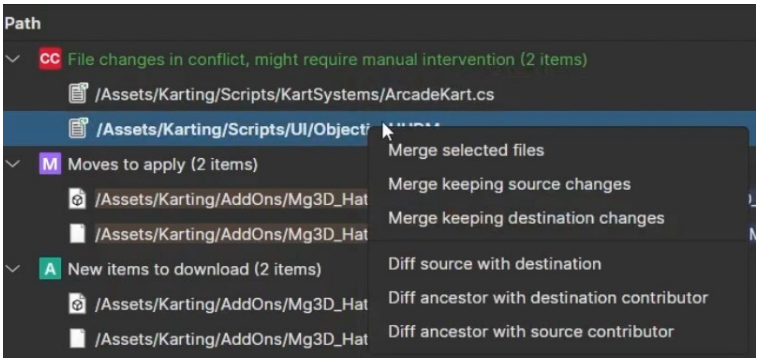
충돌 처리

서로 다른 브랜치에서 여러 사람이 같은 스크립트나 씬 파일을 변경할 수 있습니다. 이러한 파일을 병합하려고 하면 병합 충돌이 발생합니다.



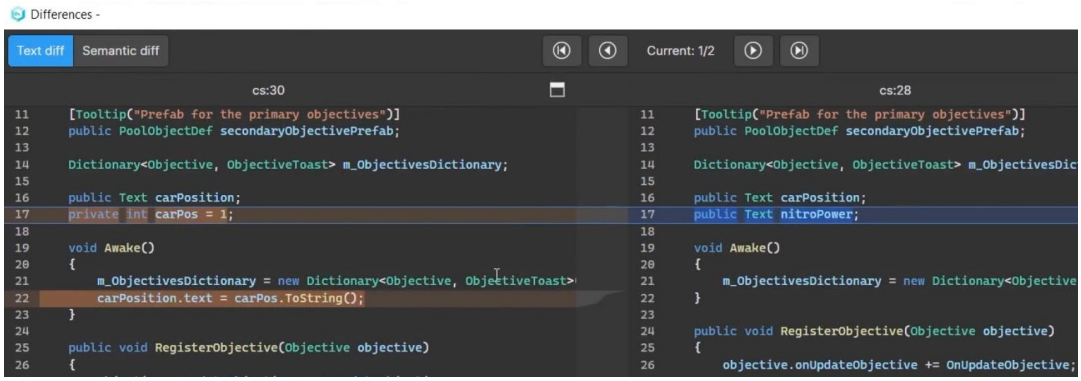
2개의 충돌이 발생하여 충돌한 파일이 표시되었습니다.

충돌한 파일을 오른쪽 클릭하여 차이점을 확인하고 변경 사항을 유지할지 삭제할지 결정합니다.



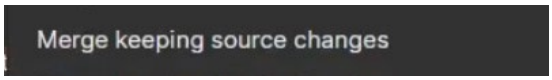
병합 충돌을 처리할 옵션이 표시되는 오른쪽 클릭 메뉴

스크립트의 경우 **Diff source with destination**을 선택할 수 있습니다. 선택하면 Diff 뷰어가 열려 병합하려는 브랜치의 파일과 다른 점을 확인할 수 있습니다.

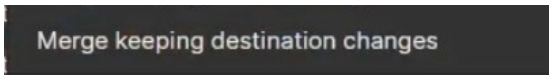


내 파일과 병합하려는 브랜치 파일의 코드를 비교하는 Diff 뷰어

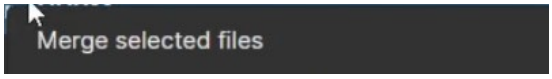
비교를 통해 다른 브랜치의 파일을 내 파일로 덮어쓸지 결정할 수 있습니다. 덮어쓰려면 **Merge keeping source changes**를 선택합니다.



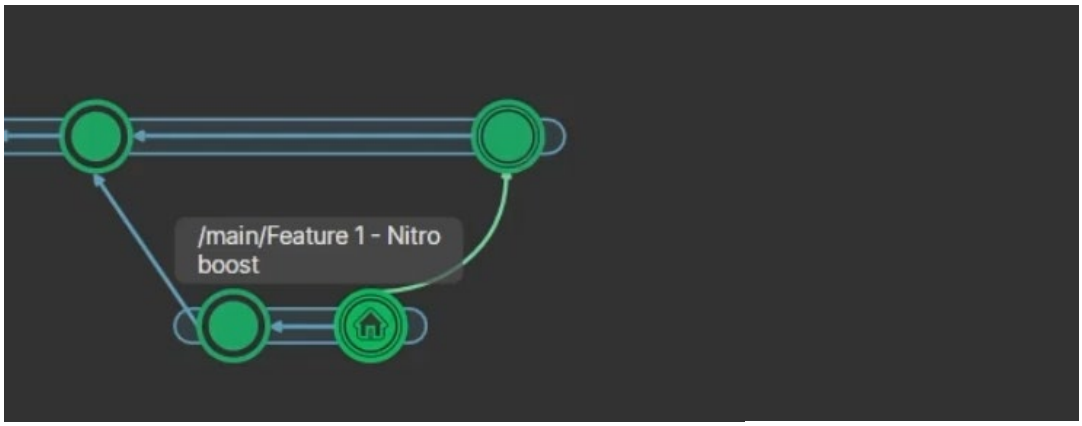
아니면 내 변경 사항을 버리고 다른 브랜치의 원본 파일을 유지할 수도 있습니다. 유지하려면 **Merge keeping destination changes**를 선택합니다.



Merge selected files를 선택하면 UVCS는 두 체인지 세트를 모두 유지하려고 시도하며 두 파일을 병합합니다. 하지만 병합 후에 스크립트를 확인하여 올바르게 병합되었는지 확인하는 것이 좋습니다.

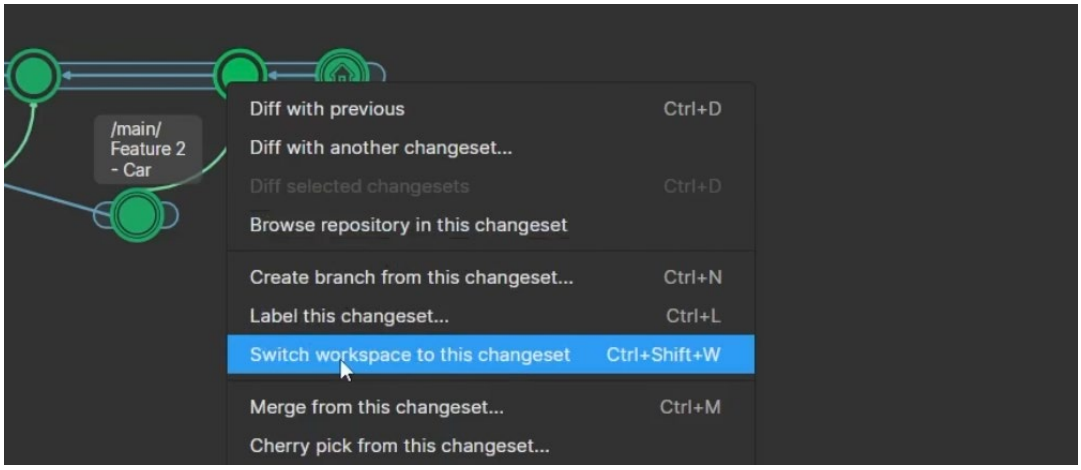


Branch Explorer에 다른 브랜치에서 병합된 체인지 세트가 표시됩니다.



메인에 병합된 브랜치

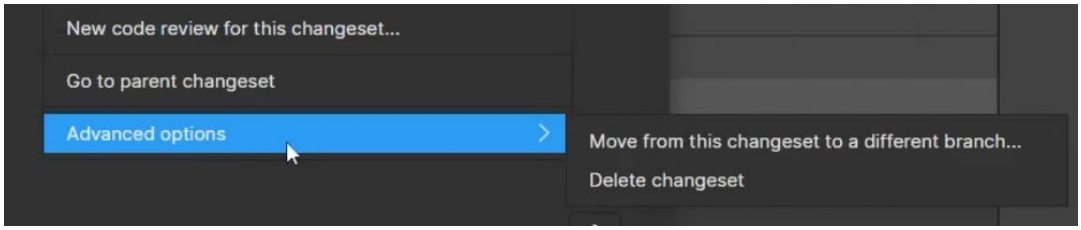
이전 체인지 세트 중 하나를 오른쪽 클릭하고 **Switch workspace to this changeset**를 선택하여 해당 시점으로 되돌릴 수도 있습니다. 게임을 방해하거나 문제를 일으키는 아이템이나 스크립트를 추가한 경우 이런 작업이 필요할 수 있을 것입니다.



원하는 체인지 세트를 오른쪽 클릭하여 이전으로 되돌릴 수 있습니다.

심각한 오류나 버그를 초래한 체인지 세트를 삭제할 수도 있습니다. 이때 해당 체인지 세트에서 이어지는 체인지 세트, 레이블, 하위 브랜치가 없어야 합니다.

오류가 발생하는 체인지 세트를 오른쪽 클릭하고 **Advanced options > Delete changeset**를 클릭합니다.

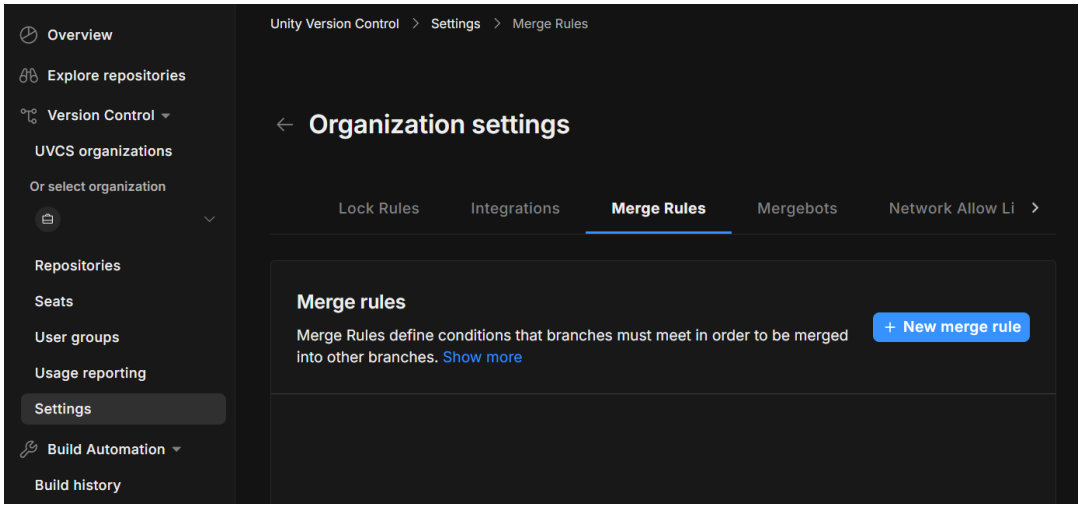


오른쪽 클릭 메뉴에서 체인지 세트를 삭제합니다.

병합 규칙

병합 충돌이 팀의 워크플로를 방해하지 않도록 병합 규칙을 만드는 것도 좋은 아이디어입니다. 브랜치를 메인 브랜치로 병합하는 것을 허용하기 전에 팀 관리자가 해당 브랜치를 승인하는 대형 팀에서 사용할 수 있는 방법입니다. 병합 규칙을 적용하면 메인 워크플로에 오류가 생기는 것을 예방할 수 있습니다.

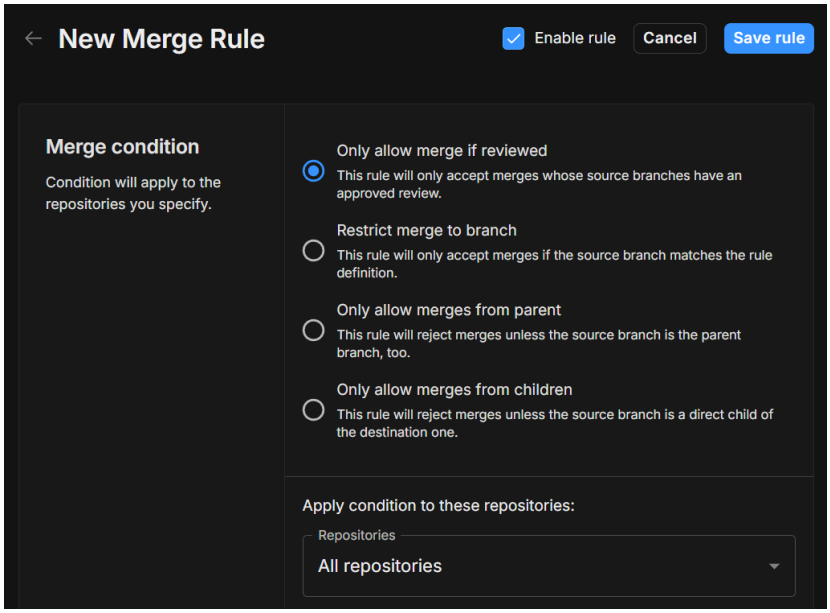
Unity Cloud 대시보드에 로그인하고 **Settings** > **Merge rules**로 이동합니다.



Unity Cloud 설정에서 새 병합 규칙 생성

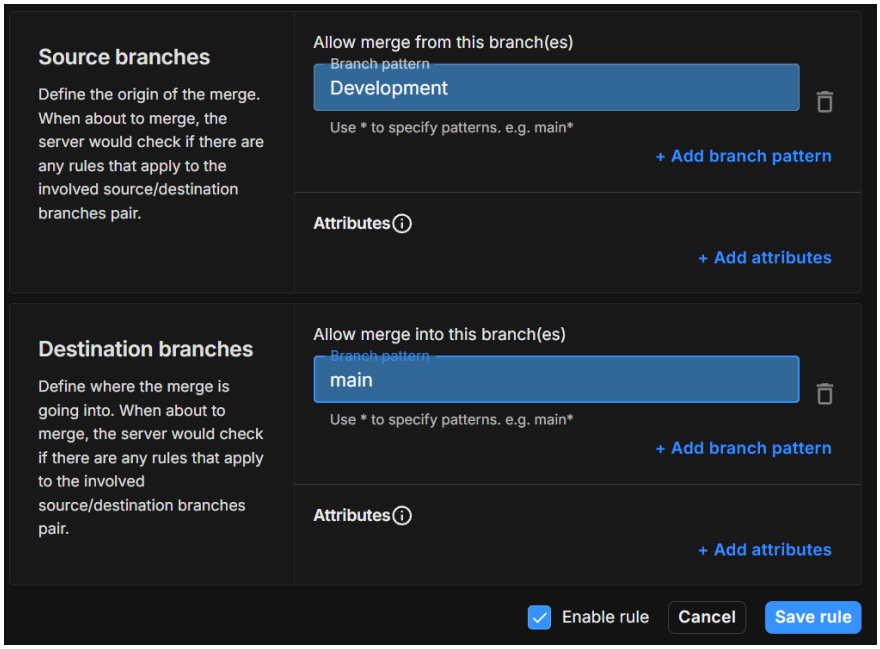
New merge rule 버튼을 클릭하고 네 가지 옵션 중 하나를 선택합니다. **Only allow merge if reviewed**로 설정하면 다른 팀원 또는 팀 관리자가 검토하고 승인해야만 병합을 진행할 수 있습니다.

작업하는 저장소가 여러 개 있다면 모든 저장소에 적용하거나 드롭다운 목록에서 특정 저장소만 지정할 수 있습니다.



병합하기 전에 다른 팀원이 브랜치를 검토해야 하는 **Only allow merge if reviewed** 옵션

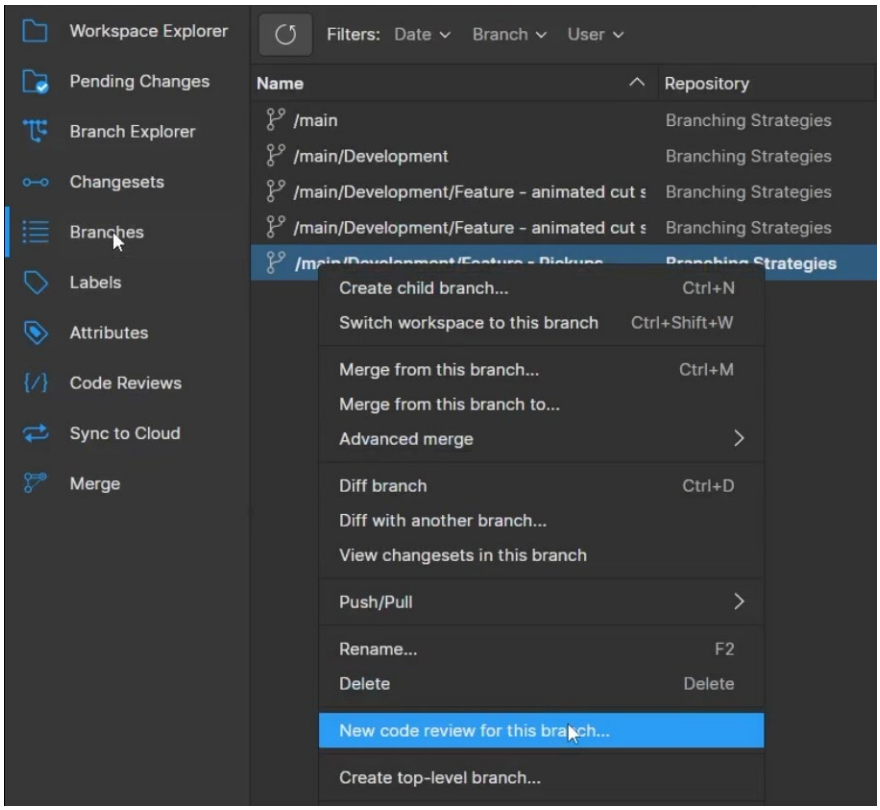
병합 규칙을 적용할 브랜치를 지정합니다. 작업한 브랜치인 소스 브랜치와 병합하려는 브랜치인 목표 브랜치를 설정해야 합니다. **Save rule** 버튼을 클릭하면 규칙이 활성화됩니다. Unity Cloud에서 언제든지 규칙을 비활성화하거나 삭제할 수 있습니다.



병합 규칙을 적용할 브랜치를 추가합니다.

브랜치를 병합하려는 팀원은 브랜치 검토를 요청해야 합니다.

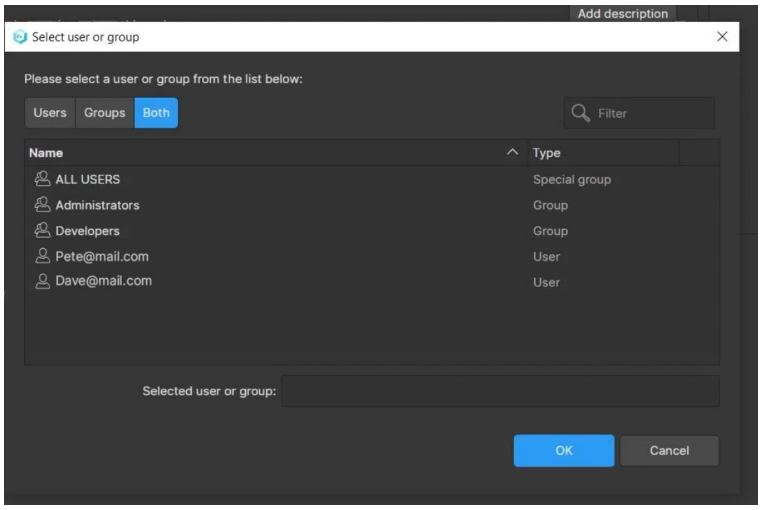
Branches 섹션에서 병합하려는 브랜치를 오른쪽 클릭합니다. **New code review for this branch를 선택합니다.**



병합하려는 브랜치의 코드 검토를 요청합니다.

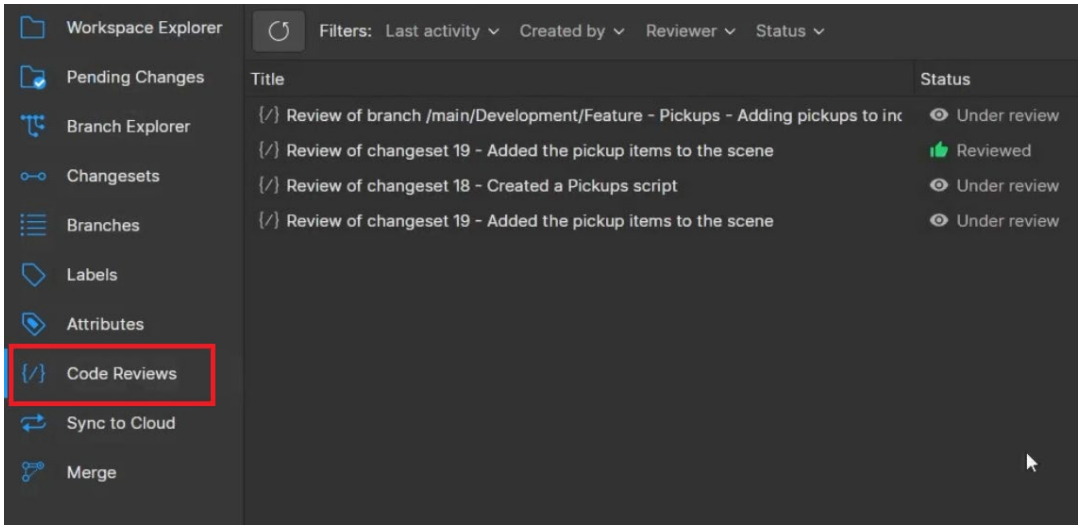
여기에는 브랜치의 전체 기간 동안 변경된 모든 파일이 포함됩니다.

conversations를 클릭하여 메시지를 남기고 작업을 검토할 팀원을 초대합니다. 초대한 모든 팀원이 브랜치를 검토해야 병합할 수 있으므로 관련된 팀원만 초대해야 합니다. 초대한 팀원은 코드를 검토해 달라는 이메일 메시지를 받습니다.



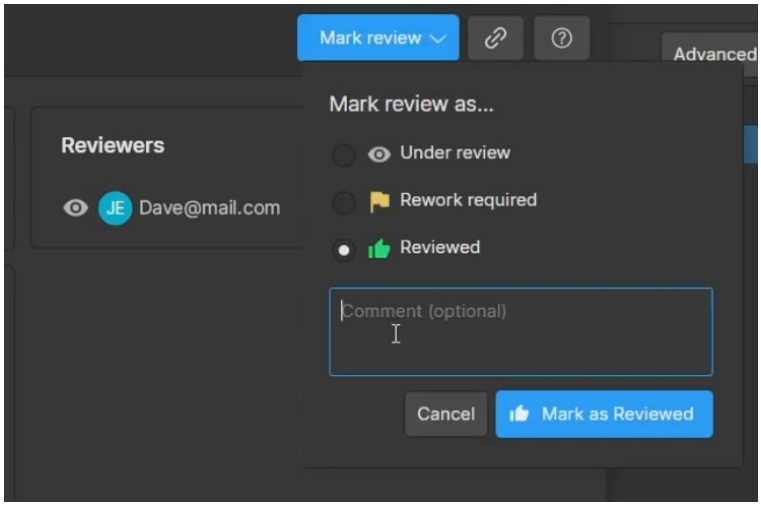
브랜치를 검토할 개인 또는 그룹 구성원을 선택합니다.

Code Reviews에서 검토 진행 상황을 확인할 수 있습니다. 항목을 더블 클릭하여 자신에게 남겨진 메시지를 확인할 수 있습니다.



요청한 검토의 진행 상황을 보여 주는 Code Reviews

초대된 팀원은 브랜치를 **Rework required** 또는 **Reviewed**로 표시하고 메시지를 남길 수 있습니다.

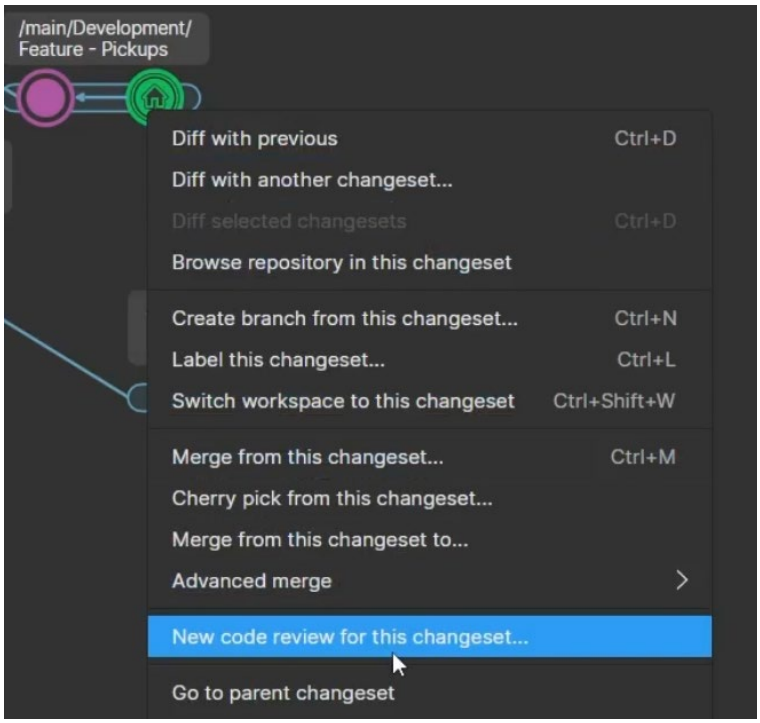


검토자는 **Mark Review** 버튼을 클릭하여 Rework required 또는 Reviewed로 표시할 수 있습니다.

브랜치 검토가 완료되면 병합할 수 있습니다. 이를 통해 프로젝트 안전성이 향상됩니다.

동일한 절차로 특정 체인지 세트의 코드를 검토할 수도 있습니다. 버그나 문제가 있는지 팀원에게 코드 검토를 요청하거나 코드의 개선 방법에 대한 조언을 받고 싶을 때 유용합니다.

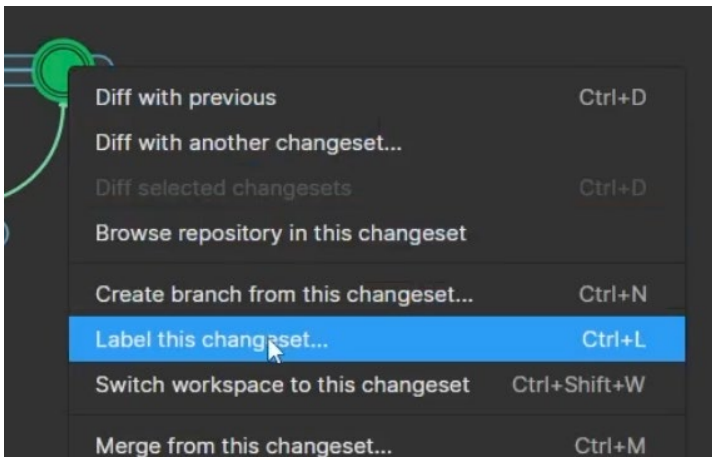
검토를 요청하려면 체인지 세트를 오른쪽 클릭하고 **New code review for this changeset**를 선택합니다. 그런 다음 파일을 검토하고 제안 사항을 작성할 팀원을 초대할 수 있습니다.



체인지 세트의 코드 검토 요청

체인지 세트에 레이블을 추가할 수도 있습니다. 버전 1.0 등의 승인된 버전이나 브랜치를 분기하는 데 사용할 수 있는 안정적인 체인지 세트에 해당하는 경우 유용할 기능입니다.

체인지 세트를 오른쪽 클릭하고 **Label this changeset**를 선택합니다. 레이블을 달면 체인지 세트를 삭제할 수 없습니다. 필요하다면 Labels 섹션에서 레이블을 제거할 수 있습니다.



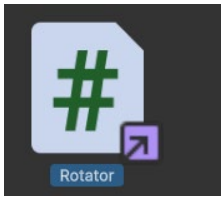
오른쪽 클릭 메뉴에서 체인지 세트에 레이블 추가

레이블은 다른 사용자에게 체인지 세트에 관한 중요한 정보를 제공합니다.

파일 잠금

작업할 파일을 잠그면 큰 병합 충돌을 피할 수 있습니다. 클라우드에서 파일이 잠기면 그동안 다른 팀원은 파일을 읽기 전용으로만 액세스하고 변경하지는 못합니다. 파일을 체크인하면 잠금이 해제되어 다른 팀원이 파일을 변경할 수 있습니다.

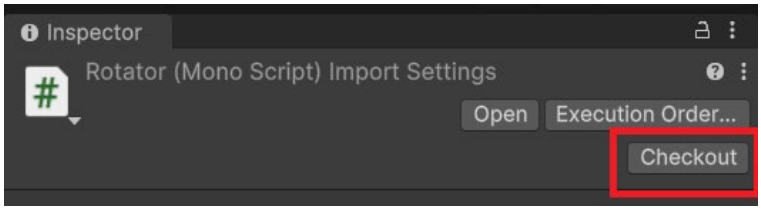
잠금은 자동으로 적용되지 않습니다. 기본적으로 Unity에 진행 상황을 저장할 때 작업 중인 모든 파일에 체크아웃 아이콘이 표시됩니다.



내 Unity와 다른 팀원의 Unity에서 파일 아이콘에 보라색 체크아웃 아이콘이 표시되어 누군가가 파일을 작업 중이라는 것을 알립니다.

모든 팀원의 컴퓨터에서 보라색 상자로 된 체크아웃 아이콘이 파일에 표시됩니다.

작업을 시작하기 전에 수동으로 파일을 체크아웃할 수 있습니다. 인스펙터에서 **Checkout** 버튼을 클릭합니다. 이 파일을 작업 중이라는 것을 다른 팀원들이 알 수 있지만, 파일이 잠겨 있지 않으므로 다른 팀원이 같은 파일을 변경할 수 있습니다. 이 기능은 파일을 잠그는 것만큼 효과적이지 않습니다.

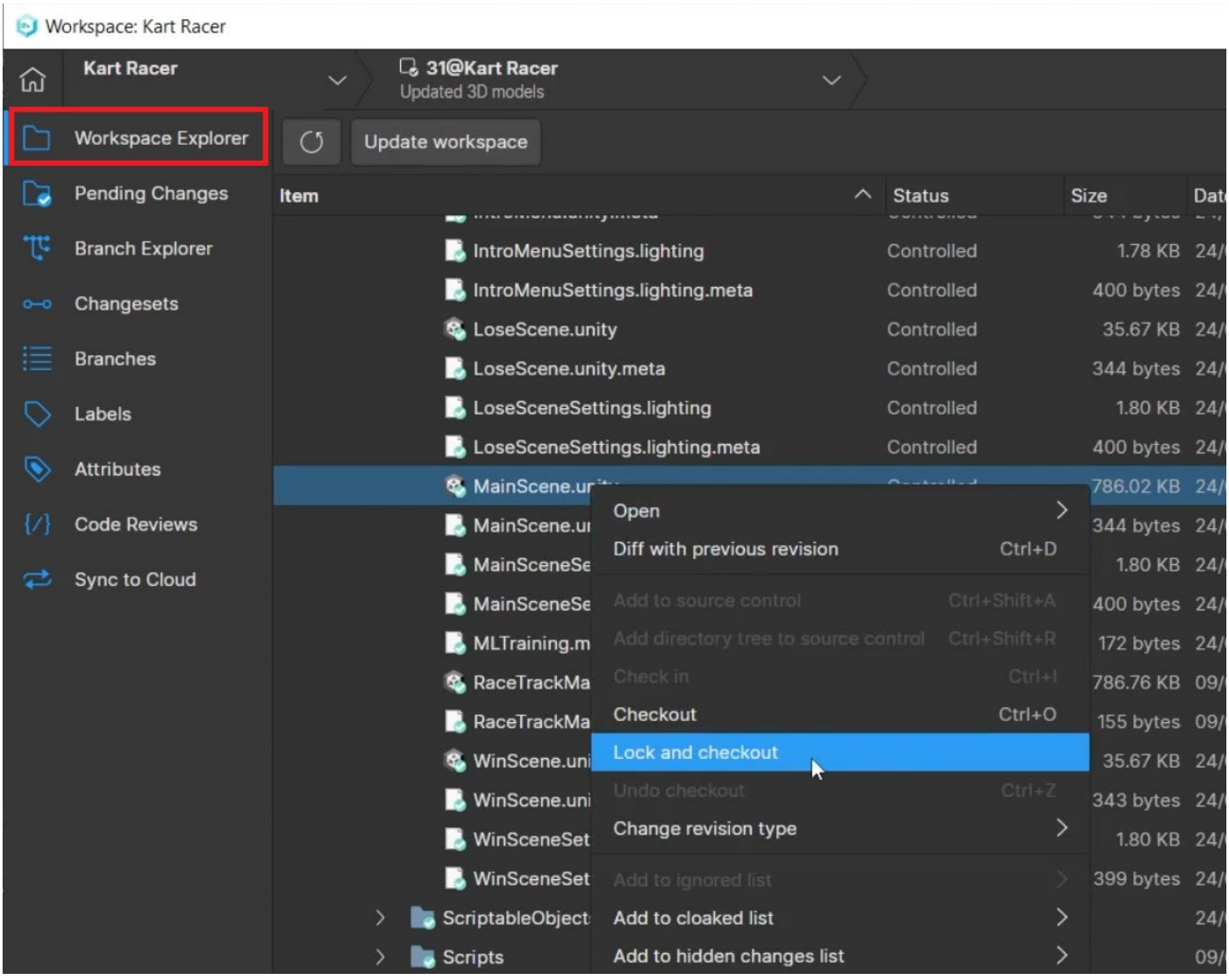


파일을 변경할 때 체크아웃하면 해당 파일을 작업 중이라는 것을 다른 팀원이 알 수 있습니다.

UVCS는 스마트 잠금 기능을 사용합니다. 이렇게 하면 이 파일을 변경하려는 다른 팀원이 파일의 최신 수정본을 사용할 수밖에 없습니다. 해당 파일의 오래된 버전을 사용하는 브랜치에서 작업할 수도 있기 때문에 이러한 조치는 중요합니다. 스마트 잠금 기능을 활용하면 메인 브랜치를 포함한 모든 브랜치를 확인하여 이 파일의 최신 수정본을 찾아 오래된 버전이 아닌 최신 버전에서 작업할 수 있습니다.

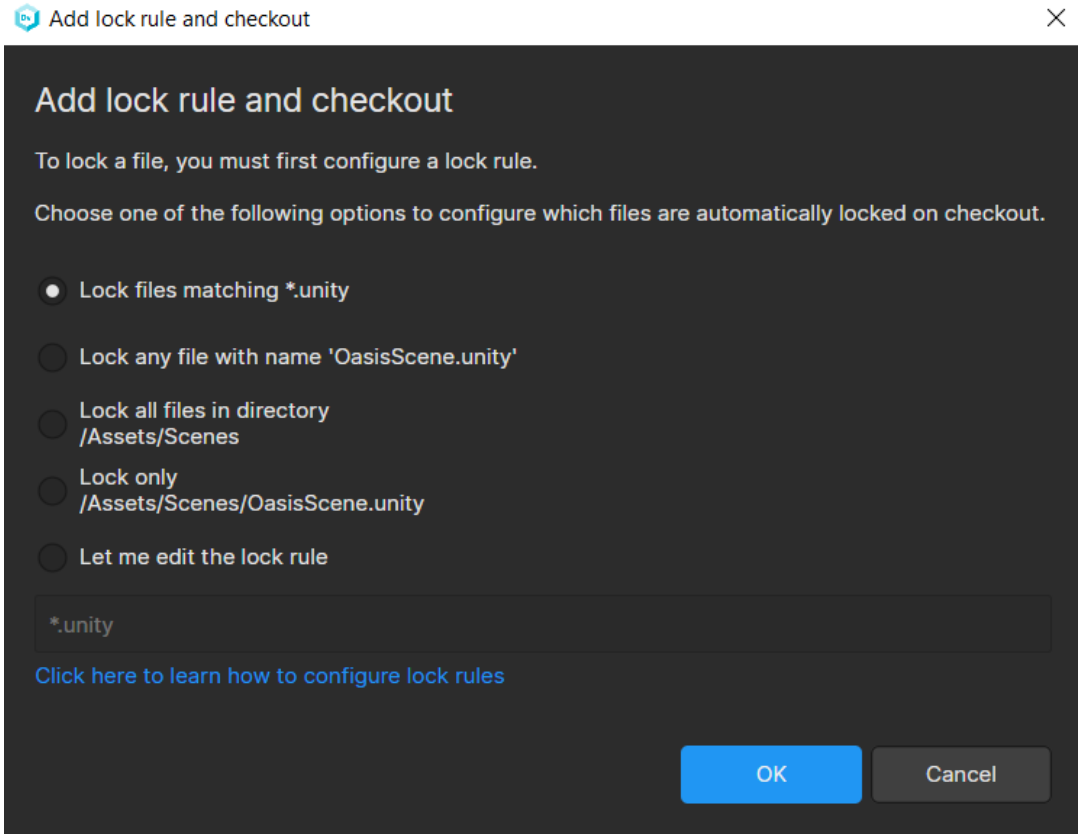
파일을 잠그려면 Branch Explorer로 이동합니다.

Workspace Explorer 뷰에서 잠그려는 파일을 오른쪽 클릭하고 **Lock and checkout**을 선택합니다. 다른 팀원이 해당 파일을 체크아웃하지 않았을 때만 사용할 수 있습니다.



UVCS 데스크톱 앱의 Workspace Explorer에서 파일을 잠급니다.

이 파일 유형에 대한 잠금 규칙을 정의할 수 있습니다. 이 파일만 잠그려면 아래 목록에서 네 번째 옵션인 **Lock only**를 선택합니다.



이 파일 또는 파일 유형의 잠금 규칙을 정의합니다.

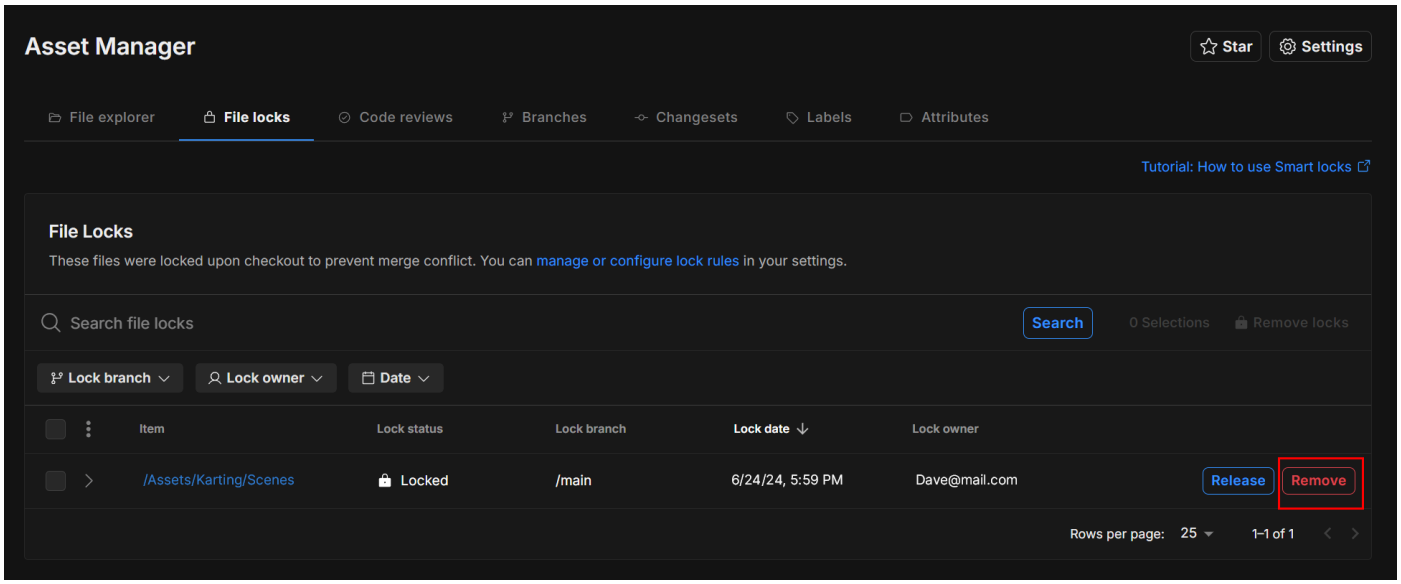
이제 모든 팀원의 Unity에서 파일에 보라색 잠금 아이콘이 표시되어 파일이 수정을 위해 잠겼다는 것을 팀원들이 확인할 수 있습니다.

다른 팀원들에게 파일이 잠겼다는 것을 나타내는 보라색 잠금 아이콘

팀원들이 파일을 읽기 전용으로 볼 수는 있지만 파일이 다시 체크인되기 전에는 해당 파일을 변경하지 못합니다.

변경을 완료한 후 파일을 체크인하면 잠금이 해제됩니다.

관리자 상태의 팀원은 클라우드에 로그인하고 **Repositories** > **File locks**로 이동하여 수동으로 잠금을 해제할 수 있습니다.

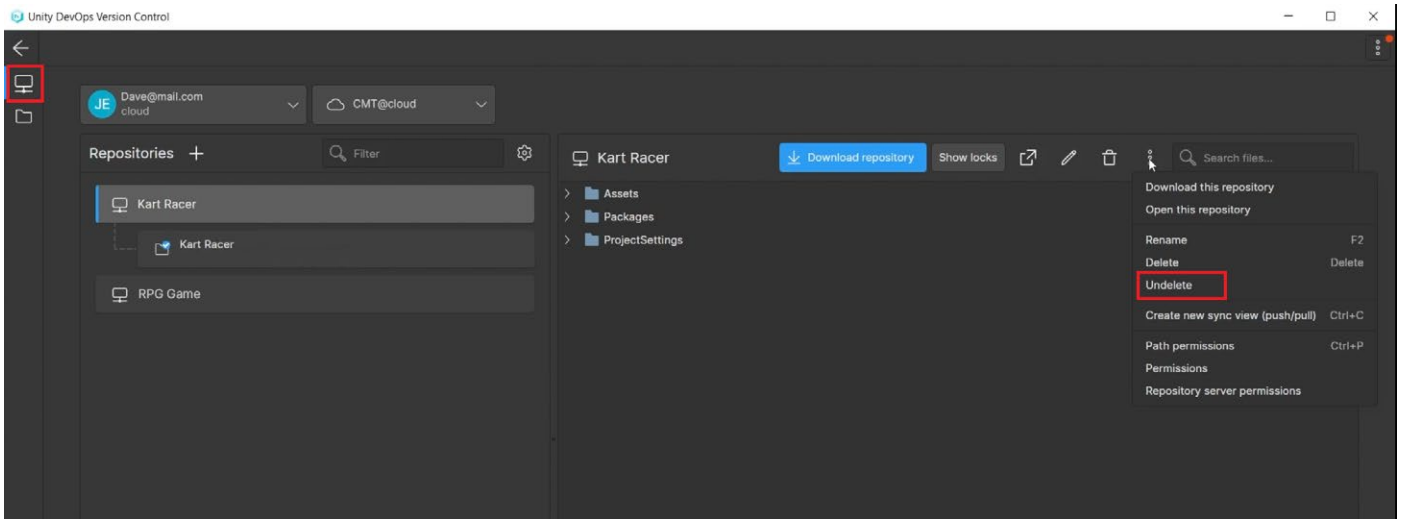


Unity Cloud에서 파일 잠금 해제

저장소 모니터링 또는 삭제

데스크톱 앱에서 저장소를 모니터링할 수 있습니다. 좌측 상단의 **홈** 아이콘을 클릭하고 상단 아이콘을 선택하면 현재 저장소를 확인할 수 있습니다.

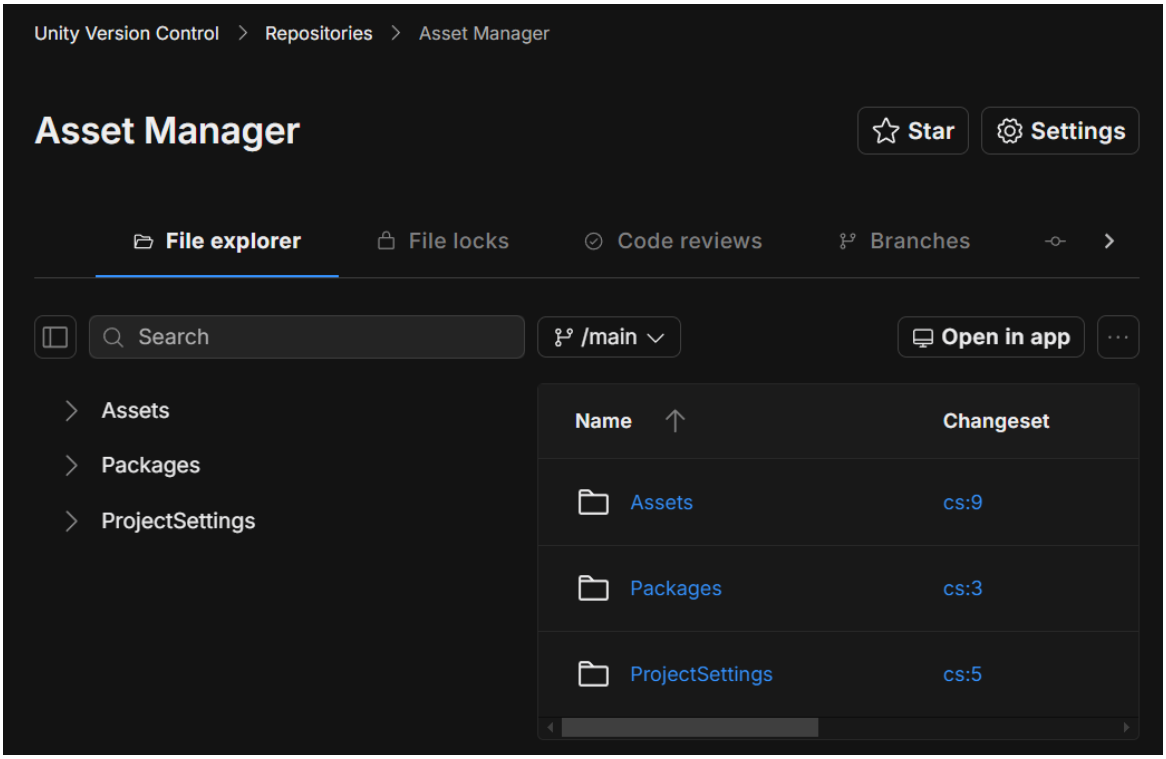
여기에서 전체 저장소를 컴퓨터에 다운로드하거나, 데스크톱 앱에서 열거나, 이름을 변경하거나, 삭제할 수 있습니다. 변심하는 경우에 대비하여 저장소는 삭제 후 10일 동안 클라우드에 유지됩니다. 삭제한 저장소를 되돌리려면 **Undelete**를 선택합니다.



저장소 다운로드, 삭제, 삭제 취소

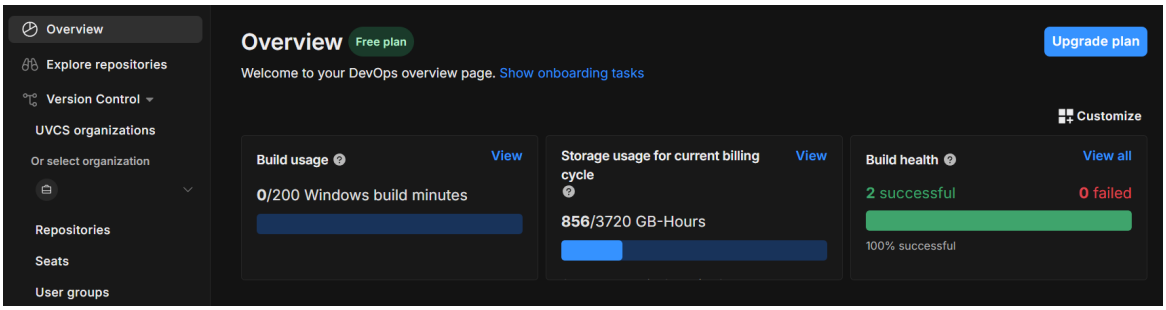
Unity Cloud의 DevOps 섹션에서 모든 저장소를 확인할 수 있습니다. 컴퓨터에 저장된 프로젝트 용량보다 훨씬 적은 실제 스토리지 사용량이 표시됩니다.

여기에서 저장소를 추가하거나 삭제할 수도 있습니다. DevOps 섹션에서는 사용량 보고서나 설정 등 DevOps 계정과 관련된 다양한 정보를 제공합니다.



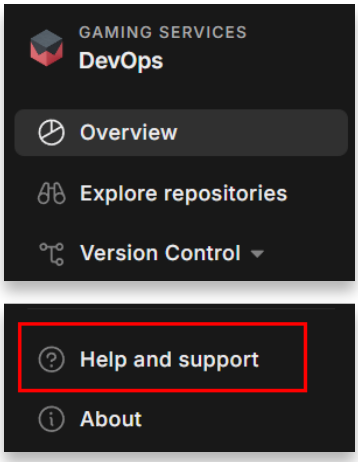
Unity Cloud DevOps 섹션에 로그인하여 저장소의 스토리지를 확인합니다.

클라우드의 **Overview** 섹션에서 현재 데이터 사용량을 확인할 수 있습니다. 현재 월간 GB 시간이 표시됩니다. 자세히 알아보려면 Unity Cloud 플랜 가격 페이지를 확인하거나 Unity Cloud 대시보드에 로그인하고 DevOps 대시보드의 'About' 페이지를 참고하세요.



월간 스토리지 사용량을 보여 주는 Unity Cloud의 Overview 섹션

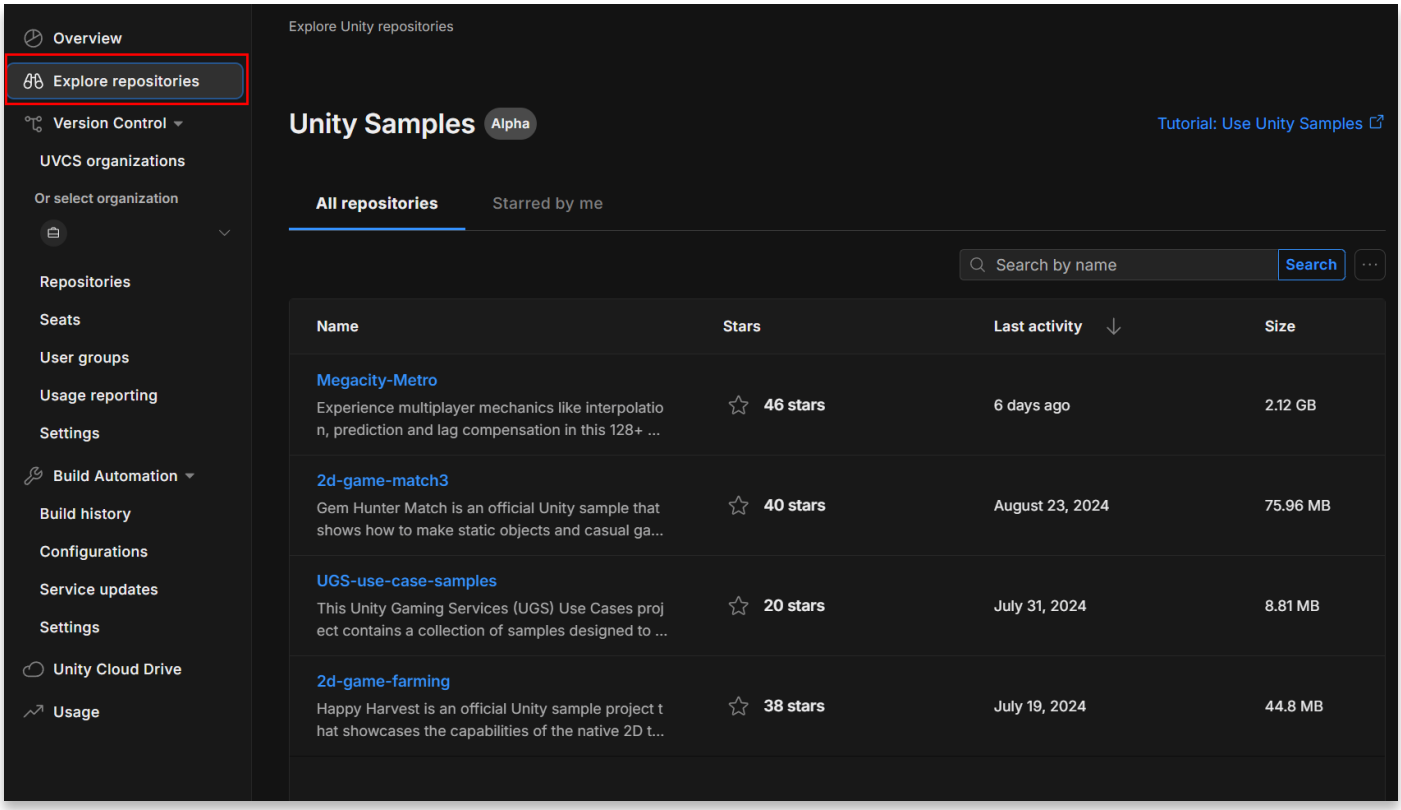
Unity 지원



Unity Cloud 대시보드의 DevOps 섹션에서 **Help and support**를 클릭합니다.

클라우드 대시보드에서 **Help and support** 버튼을 클릭하면 FAQ, 포럼, 블로그 게시물을 확인하거나 문제를 해결하기 위한 지원 티켓을 생성할 수 있습니다.

또한 **DevOps** 섹션의 **Explore repositories**에서 샘플 저장소 프로젝트를 확인할 수 있습니다. 이러한 공개 저장소는 컴퓨터에 다운로드하여 템플릿 프로젝트로 사용할 수 있습니다.



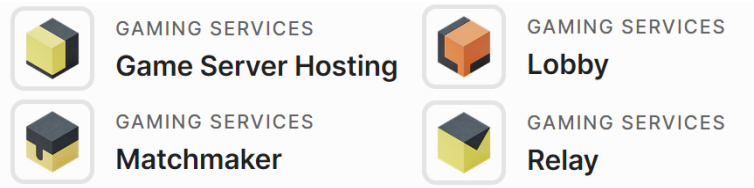
다운로드할 수 있는 공개 샘플 저장소

라이브 게임의 기반 구축

유니티는 DevOps 외에도 여러 플랫폼에서 대규모로 플레이어를 확보하고 연결하며 백엔드 인프라를 관리할 수 있는 UGS(Unity Gaming Services)를 통해 라이브 게임을 총괄하는 완전한 생태계를 제공합니다. 아래에서 몇 가지 서비스를 간략하게 소개해 드리겠습니다.

Unity Gaming Services

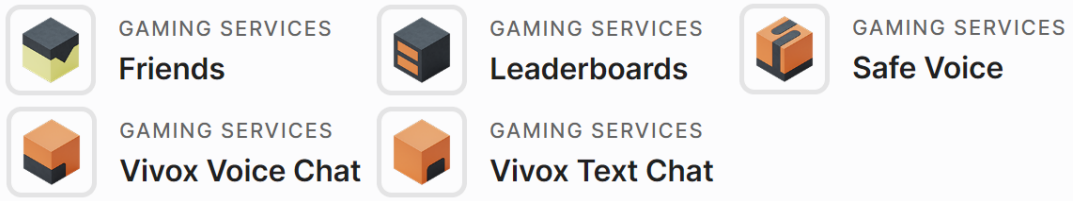
멀티플레이어



Unity는 두 가지 네트워킹 스택인 Netcode for GameObjects 및 Netcode for Entities와 더불어 Game Server Hosting 및 Vivox 음성 채팅 같은 Unity Gaming Services의 멀티플레이어 툴을 제공합니다. 필요에 따라 Unity 생태계에서 콘텐츠를 제작하거나 선호하는 툴과 서비스를 혼합해 사용할 수 있습니다.

Unity Multiplayer에 대해 자세히 알아보기

커뮤니티



2인 개발 프로젝트부터 수백만 명의 동시 사용자를 지원하는 게임까지 규모와 엔진에 구애받지 않는 음성 및 텍스트 채팅을 비롯한 커뮤니티 게임 서비스 제품군을 제공합니다. 친구나 리더보드 같은 소셜 기능을 게임에 추가하여 플레이어 참여와 리텐션을 높여 보세요.

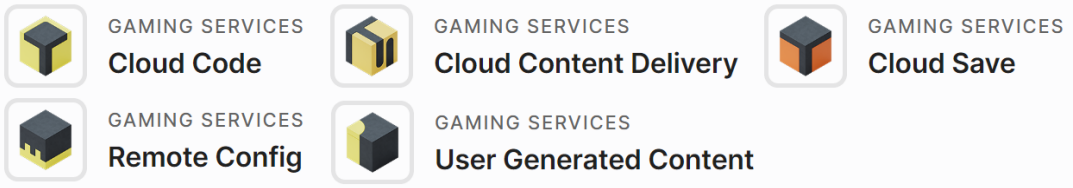
Unity 게임 커뮤니티 솔루션에 대해 자세히 알아보기

계정

Unity Authentication 및 Unity Player Accounts는 게임과 타겟 플랫폼에 적합한 혼합 기능을 제공하므로 플레이어의 ID를 유연하게 관리할 수 있습니다.

Unity Authentication 및 Player Accounts에 대해 자세히 알아보기

콘텐츠 관리

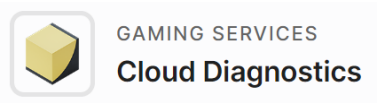


Cloud Save를 사용하면 플레이어가 게임 데이터를 클라우드에 저장할 수 있어 더 나은 플레이어 경험을 효율적으로 제공할 수 있습니다. Cloud Content Delivery를 사용하면 클라우드의 강력한 에셋 관리 및 콘텐츠 제공 기능으로 게임 업데이트를 빌드 및 릴리스할 수 있으며, 초반 50GB의 대역폭을 매달 무료로 사용할 수 있습니다.

Remote Config 서비스를 사용하면 Game Overrides를 통해 특정 플레이어 세그먼트를 타게팅하는 A/B 테스트를 효율적으로 수행할 수 있습니다. 코드를 수정하거나 앱을 업데이트하기 전에 모든 플레이어나 일부 플레이어를 대상으로 게임 난이도, 광고 빈도, 일반적인 속성 등을 변경해 볼 수 있는 것입니다.

Unity 콘텐츠 관리에 대해 자세히 알아보기

크래시 리포트



Cloud Diagnostics 서비스를 사용하면 안정성을 향상하고 플레이어 이탈률을 낮출 수 있습니다. 실시간 크래시 데이터를 제공하므로 팀이 신속하게 버그를 고치고 플레이어 이탈을 방지할 수 있기 때문입니다. Unity는 무료로 시작할 수 있고 프로젝트가 성장하면서 확장할 수 있는 크래시 및 오류 보고 솔루션을 제공합니다.

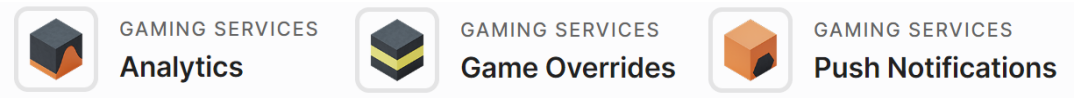
Unity Cloud Diagnostics에 대해 자세히 알아보기

게임 경제

Game Economy 서비스는 커스텀 인게임 경제를 구축하고 플레이어의 원활한 구매, 재화 전환, 인벤토리 관리 등을 지원할 수 있는 기능을 제공합니다. 게임에 완전한 기능을 갖춘 경제 시스템을 통합한 후에는 간소화된 대시보드를 통해 효율적으로 관리할 수 있습니다.

Unity의 Economy 서비스에 대해 자세히 알아보기

참여 유도 및 분석

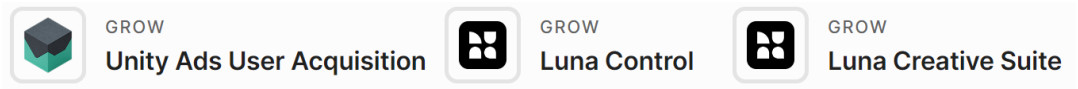


게임을 이해하고 플레이어의 참여를 유도하세요. 이 서비스를 통해 게임 경험을 효율적으로 개선하고, 플레이어를 유지하며, 게임을 성장시킬 수 있습니다.

Unity Analytics에 대해 자세히 알아보기

Unity Grow

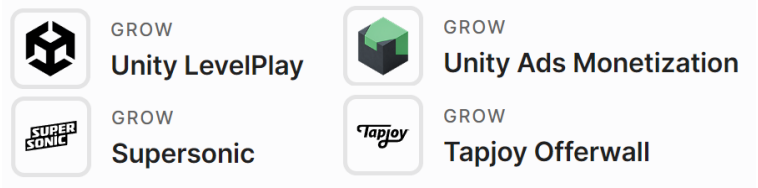
사용자 확보



Unity의 사용자 확보 서비스 제품군을 사용하면 게임에 적합한 모바일 사용자를 확보할 수 있습니다. 전 세계에 걸친 Unity Ads의 광범위한 규모를 바탕으로 ROAS(광고수익률), 리텐션, 확장에 맞게 캠페인 목표를 최적화할 수 있습니다.

Unity 사용자 확보에 대해 자세히 알아보기

수익화



게임을 효과적으로 수익화하기 위한 업계의 최첨단 기술과 강력한 툴셋으로 여러분의 앱이 지닌 매출 잠재력을 최대한 끌어내세요.

Unity 수익화에 대해 자세히 알아보기

맺는말

이 가이드를 통해 여러분의 팀이 Unity에서 버전 관리를 활용하는 데 도움이 되기를 바랍니다. 단독 프로젝트를 진행하는 경우라 해도 프로젝트 구성 및 버전 관리 시스템 사용에 관한 원칙은 매우 유용하게 활용할 수 있습니다.

무엇보다도 팀이 명확하게 소통하고 툴, 프로세스, 구조, 코드 스타일에 대해 동의하는 것이 가장 중요합니다. 프로젝트를 어떻게 구성하고, 어떤 버전 관리 시스템을 사용하고, 해당 시스템으로 어떤 워크플로를 채택할지 가이드라인을 만들고 모두가 동의해야 하죠. 그런 다음 JIRA, GitLab, 빌드 툴, 자동 테스트 툴 등 기타 툴을 통합하기 시작하면 프로젝트와 워크플로를 구성했던 작업이 큰 효과를 발휘할 것입니다.

마지막으로 이 전자책에서 설명한 다양한 버전 관리 시스템에 관한 풍부한 정보와 더불어, Unity 프로젝트를 효과적으로 설정하기 위한 추가 팁을 다음 리소스에서 살펴보세요.

추가 리소스

- 버전 관리 시스템 선택 시 고려해야 할 8가지 요소
- Unite Now 2020, 버전 관리 소개(영문)
- 크리스 벨랑거와 바겟 싱의 Git 입문서(영문)
- Unity Plastic SCM을 사용한 게임 버전 관리(영문)
- Plastic SCM 제품 기술 자료(영문)
- Plastic SCM의 Mergebot(영문)
- 버전 관리를 사용하는 Unity 오픈 프로젝트(영문)
- KO_OP가 Plastic SCM을 사용하여 제작 기간을 단축한 방법
- 출시 타임라인을 방해하는 숨겨진 제작 비용

Perforce 설정

- Helix Core와 게임 엔진을 설정하는 방법(영문)
- Helix Core 기술 자료(영문)

unity.com/kr