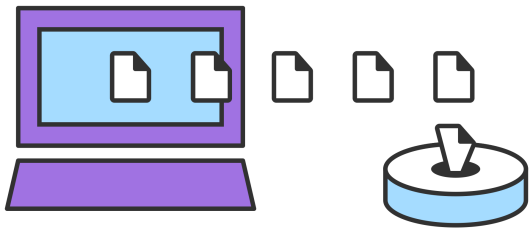
git commit



git commit 명령은 프로젝트의 현재 단계별 변경 사항에 대한 스냅 샷을 캡처합니다. 커밋 된 스냅 샷은 프로젝트의 "안전한"버전으로 생각할 수 있습니다. Git은 명시 적으로 요청하지 않으면 변경하지 않습니다. git commit을 실행하기 전에 git add 명령을 사용하여 커밋에 저장 될 프로젝트의 변경 사항을 승격 또는 '수행'합니다. 이 두 명령 git commit과 git add는 가장 자주 사용되는 두 가지 명령입니다.

Git commit 과 SVN commit 차이점

동일한 이름을 공유하는 동안 git commit은 svn commit과 같은 것이 아닙니다. 이 공유 용어는 svn 배경을 가진 Git 신규 사용자에게 혼란의 포인트가 될 수 있으며 그 차이를 강조하는 것이 중요합니다. git commit 대 svn commit을 비교하는 것은 중앙 집중식 애플리케이션 모델 (svn)과 분산 애플리케이션 모델 (Git)을 비교하는 것입니다. SVN에서 커밋은 변경 사항을 로컬 SVN 클라이언트에서 원격 중앙 집중식 공유 SVN 저장소로 푸시합니다. Git에서는 저장소가 배포되고 스냅 샷은 로컬 저장소에 커밋되며 다른 Git 저장소와는 전혀 상호 작용이 필요하지 않습니다. 자식 커밋은 나중에 임의의 원격 리포지토리에 푸시 될 수 있습니다.

How it works

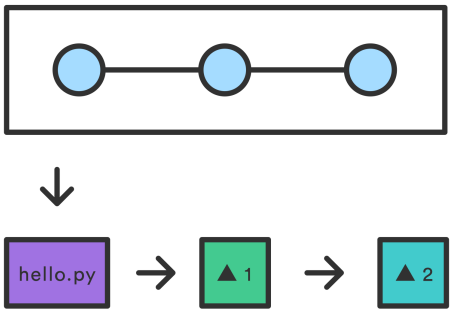
높은 수준에서 Git은 타임 라인 관리 유틸리티로 생각할 수 있습니다. 커밋은 Git 프로젝트 타임 라인의 핵심 빌딩 블록 단위입니다. 커밋은 Git 프로젝트의 타임 라인을 따라 스냅 샷 또는 마일스톤으로 생각할 수 있습니다. 커밋은 git commit 명령으로 생성되어 해당 시점의 프로젝트 상태를 캡처합니다. 스냅 샷은 항상 로컬 저장소에 커밋됩니다. 이는 SVN과 근본적으로 다르며 작업 카피는 중앙 저장소에 커밋됩니다. 대조적으로, 힘내라는 준비가 끝날 때까지 중앙 저장소와 상호 작용하도록 강요하지 않습니다. 준비 영역은 작업 디렉토리와 프로젝트 기록 사이의 버퍼이므로 각 개발자의 로컬 저장소는 기여와 중앙 저장소 사이의 버퍼입니다.

Git 사용자를위한 기본 개발 모델을 변경합니다. Git 개발자는 변경 사항을 중앙 repo에 직접 커밋하는 대신 로컬 레포에 커밋을 축적 할 수 있습니다. 이것은 SVN 스타일의 협업에 비해 많은 이점을 가지고 있습니다. 즉, 기능을 원자 단위로 분할하고, 관련 커밋을 그룹화하고, 중앙 저장소에 공개하기 전에 로컬 내역을 정리하는 것을 더 쉽게 만듭니다. 또한 개발자는 격리 된 환경에서 작업 할 수 있으므로 다른 사용자와 편리하게 병합 할 때까지 통합이 지연됩니다. 격리 및 지연된 통합은 개별적으로 유익하지만, 자주 및 작은 단위로 통합하는 것이 팀의 이익입니다. 힘내 팀 공동 작업에 대한 우수 사례에 대한 자세한 내용은 팀에서 어떻게 힘내 워크 플로우를 구성하는지 읽어보십시오.

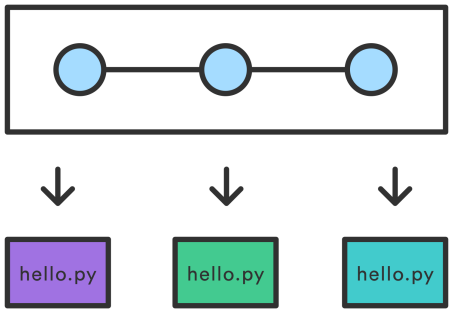
Snapshots, not differences

SVN과 Git 사이의 실질적인 차이점을 제외하고는 기본 구현은 완전히 다른 디자인 철학을 따릅니다. SVN이 파일의 차이를 추적하는 반면 Git의 버전 제어 모델은 스냅 샷을 기반으로합니다. 예를 들어, SVN 커밋은 저장소에 추가 된 원본 파일과 비교되는 diff로 구성됩니다. Git은 모든 커밋에서 각 파일의 전체 내용을 기록한다.

Recording File Diffs (SVN)



Recording Snapshots (Git)



이는 특정 버전의 파일을 diff에서 "어셈블"할 필요가 없기 때문에 많은 Git 작업을 SVN보다 훨씬 빠르게 만듭니다. 각 파일의 전체 버전은 Git의 내부 데이터베이스에서 즉시 사용할 수 있습니다.

Git의 스냅 샷 모델은 버전 제어 모델의 거의 모든 측면에 광범위한 영향을 미치며 분기 및 병합 도구에서부터 협업 작업 흐름에 이르는 모든 것에 영향을 미칩니다.

공통 옵션

```
git commit
```

준비된 스냅 샷을 커밋합니다. 그러면 커밋 메시지를 묻는 텍스트 편집기가 시작됩니다. 메시지를 입력 한 후 파일을 저장하고 편집기를 닫아 실제 커밋을 만듭니다.

```
git commit -a
```

작업 디렉토리의 모든 변경 사항에 대한 스냅 샷을 커밋합니다. 이것은 오직 추적 된 파일 (히스토리의 어느 시점에서 git add로 추가 된 것들)에 대한 수정 만 포함합니다.

```
git commit -m "commit message"
```

전달 된 커밋 메시지로 즉시 커밋을 만드는 바로 가기 명령입니다. 기본적으로 git commit은 로컬로 구성된 텍스트 편집기를 열고 커밋 메시지를 입력하라는 메시지를 표시합니다. -m 옵션을 전달하면 인라인 메시지를 선호하는 텍스트 편집기 프롬프트가 표시되지 않습니다.

```
git commit -am "commit message"
```

-a 및 -m 옵션을 결합한 파워 유저 바로 가기 명령입니다. 이 조합은 즉시 모든 단계적 변경 사항의 커밋을 생성하고 인라인 커밋 메시지를받습니다.

```
git commit --amend
```

이 옵션은 커밋 명령에 다른 수준의 기능을 추가합니다. 이 옵션을 전달하면 마지막 커밋이 수정됩니다. 새 커밋을 만드는 대신 이전 커밋에 단계별 변경 사항이 추가됩니다. 이 명령은 시스템에 구성된 텍스트 편집기를 열고 이전에 지정한 커밋 메시지를 변경하라는 메시지를 표시합니다.

Examples

커밋으로 변경 저장하기

다음 예제는 현재 브랜치에서 hello.py라는 파일의 일부 내용을 편집 한 것으로 가정하고 프로젝트 히스토리에 커밋 할 준비가되어 있다고 가정합니다. 먼저 git add로 파일을 준비한 다음 준비된 스냅 샷을 커밋 할 수 있습니다.

```
git add hello.py
```

이 명령은 hello.py를 Git 스테이징 영역에 추가합니다. git status 명령을 사용하여이 작업의 결과를 검사 할 수 있습니다.

```
git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
  new file: hello.py
```

녹색 출력 새로운 파일 : hello.py는 hello.py가 다음 커밋과 함께 저장 될 것임을 나타냅니다. 커밋에서 다음을 실행하여 만듭니다.

```
git commit
```

커밋 된 로그 메시지를 요구하는 텍스트 편집기 (git config를 통해 사용자 정의 가능)가 열립니다.

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
# (use "git reset HEAD ..." to unstage)
#
#modified: hello.py
```

Git은 특정 형식 제약 조건을 따르는 커밋 메시지를 요구하지 않지만 표준 형식은 첫 줄에 전체 커밋을 50 자 미만으로 요약하고 빈 줄을 남기고 변경된 내용에 대한 자세한 설명을 제공합니다. 예 :

```
Change the message displayed by hello.py

- Update the sayHello() function to output the user's name
- Change the sayGoodbye() function to a friendlier message
```

전자 메일과 마찬가지로 커밋 메시지의 첫 번째 줄을 제목 줄로 사용하는 것이 일반적입니다. 나머지 로그 메시지는 본문으로 간주되어 커밋 변경 집합의 세부 정보를 전달하는 데 사용됩니다. 많은 개발자들이 커밋 메시지에 현재 시제를 사용하기를 좋아한다는 점에 유의하십시오. 이것은 리포지토리에서의 작업과 같이 더 많이 읽을 수있게 해줌으로써 많은 역사 작성 작업을 더욱 직관적으로 만듭니다.

마지막 커밋 수정하기

위의 hello.py 예제를 계속 진행하십시오. hello.py에 대한 추가 업데이트를하고 다음을 실행합니다.

```
git add hello.py
git commit --amend
```

그러면 다시 구성된 텍스트 편집기가 열립니다. 그러나 이번에는 이전에 입력 한 커밋 메시지로 미리 채워질 것입니다. 이것은 우리가 새로운 커밋을 생성하는 것이 아니라 마지막 커밋을 편집하고 있음을 나타냅니다.