



# CHAPTER 01

# 깃과 버전 관리

# 1장 깃과 버전 관리

---

1.1 버전 관리

1.2 버전 관리 시스템

1.3 깃

1.4 깃의 동작 한눈에 보기

1.5 정리

# 1.1 버전 관리





# 1. 버전 관리

## ➤ 버전 관리

- 프로그래밍은 컴퓨터 언어로 글을 작성하는 창작 활동이라고 할 수 있음
- 프로그래밍 개발 과정은 수많은 코드를 변경하고 테스트하는 것
- 지속적으로 변경되는 과정 속에서 코드는 잠시 불안정한 수정 상태와 안정된 상태를 반복함
- 개발자는 안정된 상태의 코드와 불안정한 상태의 코드를 인지하고, 항상 안정된 상태를 유지하도록 노력해야 함



# 1. 버전 관리

## ➤ 버전이란?

### ● 버전:

- 이전과 약간씩 다른 변화들을 구분하는 표시
- 버전을 표시하는 데 숫자를 많이 사용하지만 꼭 숫자만 사용해야 하는 것은 아님
- 2016에디션, 2017에디션, 윈도우 XP, macOS X 엘카피텐처럼 연도나 다른 기호를 사용하기도 함

### ● 서브버전:

- 버전과 버전 사이에 변화된 것
- 1.0 버전과 2.0 버전 사이에는 1.01, 1.02, 1.03처럼 수많은 서브버전이 있음

### ● 버전의 숫자나 기호 역시 일련의 규칙들이 있음

- 버전을 부여하려면 소스 코드를 구별 할 수 있는 의미 있는 변화가 있어야 함
  - 개발 도중 임시로 작업한 것을 버전이라고 말하지는 않음



# 1. 버전 관리

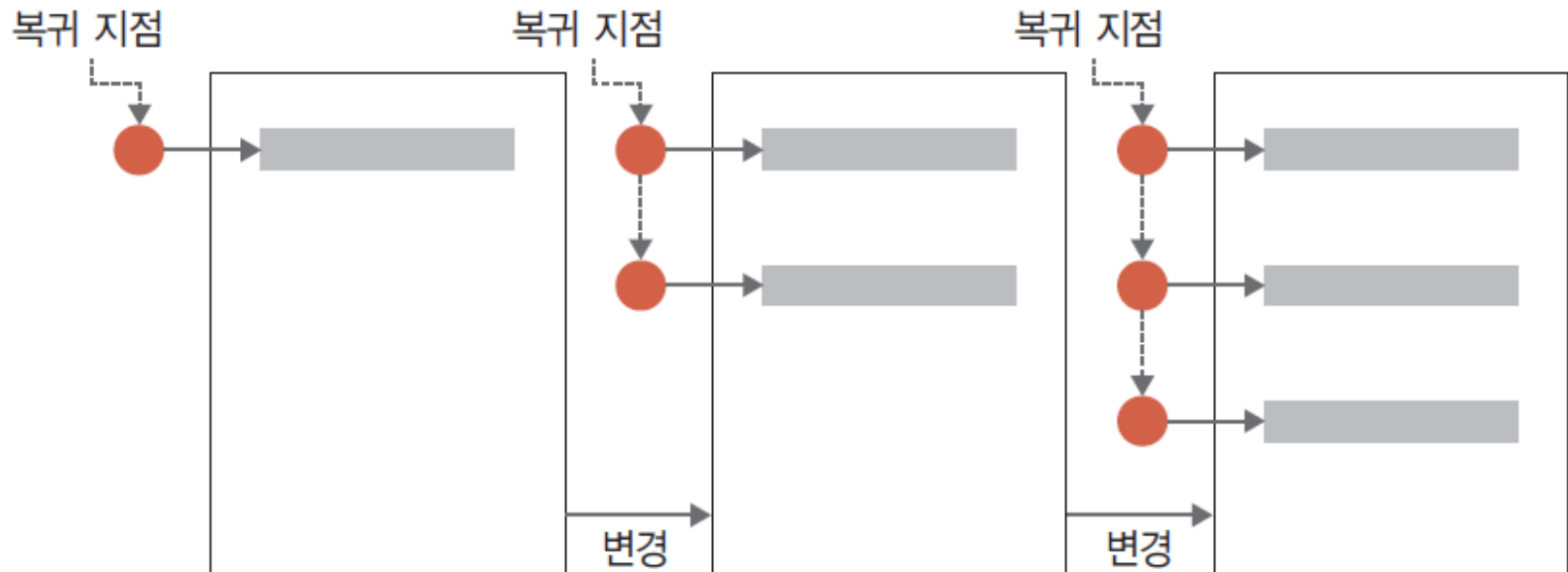
## ➤ 버전 관리는 왜 필요할까?

- 개발 도중에는 많은 기능이 추가되고, 수많은 코드가 변경됨
- 변경되는 동안 코드들은 잠시 불안정한 상태가 되고,  
이후 정상적인 테스트와 동작을 확인하고 나면 다시 안정된 상태의 코드가 됨
- 숙련된 개발자라면 작업 과정에서 코드들을 안정되게 유지해야 한다는 것을 잘 알고 있음
- 개발 또는 테스트하는 과정에서 불안정한 코드가 있다면 계속 이어서 작업하기 불안할 것임
  - 때에 따라 더 이상 작업하기 어려울 수도 있음
- 이때는 이전 상태로 돌아가 다시 시작할 수 있는 코드의 복귀(포인트) 지점이 필요함



# 1. 버전 관리

## ▼ 그림 1-1 코드 복귀 지점





# 1. 버전 관리

## ➤ 버전 관리는 왜 필요할까?

- 코드 복귀 지점은 반드시 안정된 코드 상태를 기준으로 설정해야 함
- 복귀 지점을 기록해 두면 좀 더 자유롭게 안정적으로 개발할 수 있음



## 1.2 버전 관리 시스템

---



## 2. 버전 관리 시스템

### ➤ 버전 관리 시스템

- 우리는 컴퓨터를 사용하면서 이미 다양한 방법으로 버전을 기록하고 있음
- 작업 상태를 저장하는 것 또한 변화의 기록임
- 다른 이름으로 저장하는 방식은  
한 문서를 여러 파일로 계속 저장하는 방식이라,  
시간 차이를 두고 저장하다 보면 이름 규칙을 잊곤 함



## 2. 버전 관리 시스템

### ➤ 버전 관리 시스템

- 다음과 같이 통일성 없이 파일 이름을 만들어 저장했다면 버전을 알아보기 힘들

### ▼ 그림 1-2 새 파일로 계속 저장

 hello.txt	2019-06-02 오후...	텍스트 문서
 hello_1.txt	2019-06-02 오후...	텍스트 문서
 hello_2 재수정.txt	2019-06-02 오후...	텍스트 문서
 hello_2.txt	2019-06-02 오후...	텍스트 문서
 hello_3.txt	2019-06-02 오후...	텍스트 문서
 hello4.txt	2019-06-02 오후...	텍스트 문서



## 2. 버전 관리 시스템

### ➤ 버전 관리 소프트웨어

- 버전 관리 시스템(VCS, Version Control System):
  - 코드와 콘텐츠의 변화를 관리하고 추적하는 소프트웨어
- 최초의 버전 관리 시스템은 유닉스 환경에서 사용 가능한 SCCS(Source Code Control System)임
  - SCCS는 1970년대 마크 로치킨드(Marc J. Rochkind)가 개발했으며, 이후 다양한 VCS 프로그램이 등장함
- 저장소(repository)(리포지터리):
  - VCS에서 버전 파일들을 관리하고 저장하는 공간



## 2. 버전 관리 시스템

### ➤ 버전 관리 소프트웨어

#### ● 집중형

- 집중형 시스템은 말 그대로 모든 소스 코드가 한곳에 집중되어 있는 형태임
- 하나의 메인 중앙 서버에서 개발 구성원의 모든 소스 코드를 통합적으로 관리함
- 클라이언트-서버 모델이라고도 함

#### ● 장점: 저장소 하나를 중심으로 관리하기 때문에 시스템을 운영하기 수월함

#### ● 단점: 중앙 저장 공간인 서버에 문제가 생기면, 소스 코드가 있는 메인 저장소에 모든 개발자가 접근할 수 없는 심각한 상황이 발생할 수 있음

- 동시에 여러 개발자가 접근하면 충돌이 발생하기에 코드 수정을 안정적으로 할 수 있게 잠금 모델을 적용함
- 개발자들은 파일을 변경하려면 순서대로 대기하고 있어야 함



## 2. 버전 관리 시스템

### ➤ 버전 관리 소프트웨어

- 대표적인 집중형 관리 시스템은 다음과 같음
  - SCCS: 1970년대 최초의 버전 관리
  - RCS: 1980년대 정방향/역방향 개념 도입
  - CVS: 1986년
  - 서브버전: 2000년



## 2. 버전 관리 시스템

### ➤ 버전 관리 소프트웨어

#### ● 분산형

- 분산형 버전 관리 시스템(DVCS, Distributed Version Control System)은 집중형 시스템과 달리 저장소가 여러 개 있음
- 여러 저장소에 각 버전별 소스를 개별 보관함
- 분산 저장소는 P2P(Peer-to-Peer) 방식으로 공유하며, 각 개발자에게 공유 가능한 저장소 사본을 제공함
- 서버는 각 저장소 자료를 동기화하고 중개하는 역할만 수행함
- 메인 서버에 문제가 생기더라도 지속적으로 개발할 수 있음
- 그 대신 익숙해지는데 시간이 걸리는 단점이 있음



## 2. 버전 관리 시스템

### ➤ 버전 관리 소프트웨어

- 대표적으로 분산형 관리 시스템은 다음과 같음
  - 깃(Git): 현재 가장 많이 사용하는 분산형 VCS임
    - 오픈 소스라서 무료로 사용이 가능함
  - 머큐리얼(Mercurial): 파이썬 언어로 개발했으며, 무료임
    - 자세한 정보는 <https://www.mercurial-scm.org/>에서 확인 가능함
  - 비트키퍼(BitKeeper): 1998년에 출시했으며, 상용(유료) 제품임
    - 자세한 정보는 <http://www.bitkeeper.org/>에서 확인 가능함



## 1.3 깃





## 3. 깃

### ➤ 깃

- 깃은 2005년에 리눅스 개발자인 리누스 베네딕트 토르발스(Linus Benedict Torvalds)가 개발함
- 깃의 모든 소스는 깃허브2에 공개되어 있으며, 깃허브에서 git으로 검색하면 됨
  - <https://github.com/git/git>

# 3. 깃



## ▼ 그림 1-3 깃 소스

GitHub - git/git: Git Source Code Mirror

Why GitHub? Enterprise Explore Marketplace Pricing Search Sign in Sign up

git / git Watch 2.2k Star 31.1k Fork 18k

Code Pull requests 18 Security Insights

Git Source Code Mirror - This is a publish-only repository and all pull requests are ignored. Please follow Documentation/SubmittingPatches procedure for any of your improvements.

c shell

58,053 commits 5 branches 0 packages 742 releases 1,343 contributors View license

Branch: master New pull request Find file Clone or download

gitster Sync with maint Latest commit 2323784 6 hours ago

.github	point pull requesters to GitGitGadget	10 months ago
Documentation	Revert "Merge branch 'ra/rebase-i-more-options'"	4 days ago
block-sha1	sha1: provide another level of indirection for the SHA-1 functions	4 years ago
builtin	Revert "Merge branch 'ra/rebase-i-more-options'"	4 days ago
ci	Merge branch 'sg/osx-force-gcc-9'	last month
compat	Sync with maint	6 hours ago
contrib	Sync with maint	6 hours ago
ewah	ewok_rlw.h: add missing 'inline' to function definition	last year
git-gui	Merge https://github.com/prati0100/git-gui	8 days ago



## 3. 깃

### ➤ 깃

- 깃은 다음과 같이 몇 가지 특징으로 구분할 수 있음
- 대표적인 분산형 버전 관리 시스템임
- 원격 저장소(remote repository)와 별개로 개발자 각각의 로컬 컴퓨터에 완벽한(원격 저장소의 내용과 동일한) 복제본 소스 코드를 저장할 수 있음
- 완벽한 복제본이 있으면 매번 중앙 저장소를 조회하지 않아도 개발을 진행할 수 있음
- 네트워크나 인터넷이 연결되어 있지 않은 상태에서도 로컬 컴퓨터의 소스 코드만으로 버전을 관리할 수 있음
- 작업 후 나중에 인터넷에 연결되었을 때 동기화만 하면 됨
- 원격 저장소로 많은 개발자의 저장소와 연결하거나 동기화 작업을 할 수 있음
- 또 직접 만든 새로운 소스 코드를 배포하거나 내려받은 소스 코드를 수정한 후 다시 병합(merge) 할 수도 있음



## 3. 깃

### ➤ 백업 기능

- 분산형 깃은 자신의 로컬 컴퓨터에서 독립적으로 소스의 버전 관리를 할 수 있음
- 독립적이란것은 로컬 컴퓨터에서 자체적으로 버전을 기록하고 관리할 수 있는 시스템을 의미함
- 컴퓨터에 문제가 생긴다면 지금까지 개발한 모든 소스를 잃을 수 있음
- 이에 대비하여 별도의 외부 저장 장치에 데이터를 백업하곤 함
- 외부 저장 장치에 백업하는 코드는 단순히 파일을 복사하는 것으로 기존 소스 코드와는 동기화하지 않은 별개의 파일임
- 깃을 사용하면 코드를 원격 저장소에 저장할 수 있음
- 로컬 컴퓨터의 저장소를 동기화하여 원격 저장소에 백업함
- 사무실, 집 등 여러 공간에서 원격 저장소에 저장된 내용을 다시 내려받아 프로젝트 개발을 이어서 할 수 있음



## 3. 깃

### ➤ 협업 개발

- 깃은 다수의 개발자와 코드를 공유하고 협업할 때 매우 유용함
- 코드 공유
  - 예전에는 팀 내에서 코드를 공유하려고 외부 저장 장치를 이용하곤 함
  - 이는 매우 번거로운 작업임
  - 깃을 사용하면 네트워크를 통해 코드를 좀 더 쉽게 공유할 수 있음
  - 심지어 인터넷이 연결되지 않은 상태에서도 코드 이력을 관리하고, 다른 개발자와 공유하여 협업할 수 있음



## 3. 깃

### ➤ 협업 개발

#### ● 책임과 기록

- 깃은 변경된 모든 이력을 저장함
- 누가 언제 어떤 파일을 수정했는지 기록하기 때문에 코드를 좀 더 책임감 있게 작성하고 유지할 수 있음
- 깃은 커밋(commit)을 거쳐 모든 코드의 수정 이력을 기록함
- 깃의 커밋은 신중하게 작업해야 함
- 커밋으로 저장된 원본 객체는 수정할 수 없음



## 3. 깃

### ➤ 협업 개발

#### ● 원격 공유

- 분산된 여러 저장소 간에 정보를 주고받으려면 중앙 서버가 필요함
- 깃에서는 원격 저장소가 중앙 서버 역할을 함
- 자신의 코드 저장소를 원격 서버에 푸시(push)하여 저장(동기화)함
- 또 다른 개발자의 소스를 원격 서버에서 풀(pull) 또는 페치(fetch)하여 언제든지 내려받을 수 있음
- 깃을 사용하면 개발 구성원 간에 소스 코드를 쉽게 주고받을 수 있음
- 원격 저장소로 전송된 코드는 여러 개발자와 소스 코드를 공유함
- 협업하여 코드를 개발할 때 공유 기능은 매우 중요함





## 3. 깃

### ➤ 협업 개발

#### ● 병합

- 깃은 하나의 소스 코드를 여러 가지 브랜치로 분기하여 독립된 기능을 구현할 수 있음
- 독립적으로 구현된 소스를 주고받으며, 필요하다면 각 브랜치를 하나로 병합하기도 함
- 독립적으로 구현된 코드들을 수작업으로 병합하는 것은 어려움
- 깃에서는 다양한 병합 알고리즘을 제공하기 때문에 이를 이용하여 소스 코드의 충돌을 최소화하고 최종 코드를 쉽게 유지할 수 있음



## 3. 깃

### ➤ 협업 개발

#### ● 공개

- 원격 저장소를 사용하여 개발 중인 코드를 외부로 공개할 수 있음
- 코드를 공개함으로써 내부 개발자가 만든 기능 한계를 극복하고 외부 개발자와 협업할 수 있음
- 공개된 프로젝트는 많은 개발자 간의 협력으로 프로젝트를 빠르게 성장 및 발전시킬 수 있음
- 외부 개발자는 원격 저장소를 포크(fork)하여 소스 코드의 버그를 수정하거나 기능을 개선할 수 있음
- 수정한 소스 코드를 풀 리퀘스트(pull request)하여 기존 코드에 병합할 수도 있음

## 1.4 깃의 동작 한눈에 보기

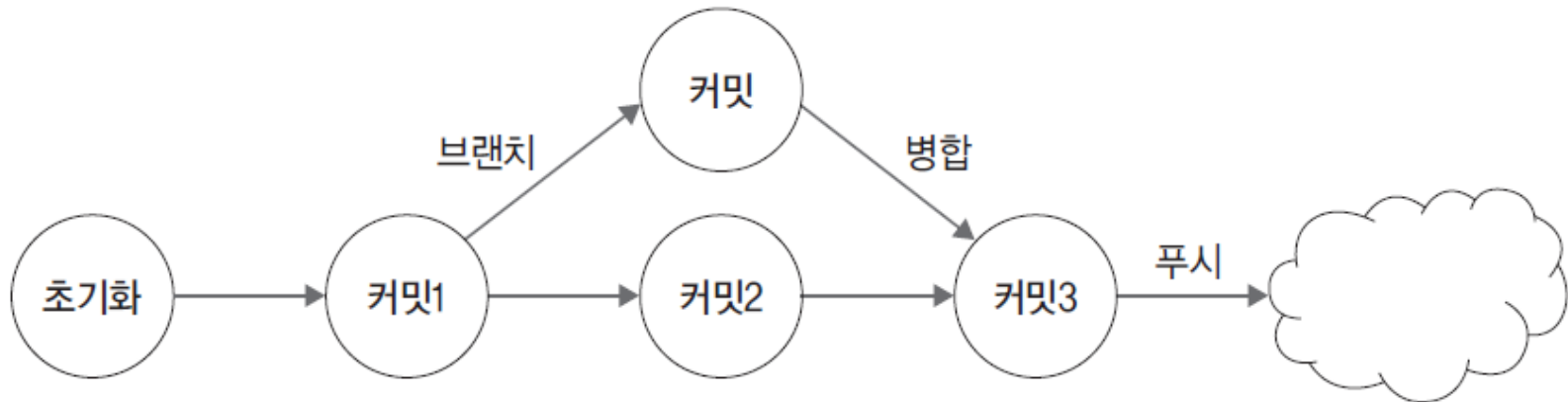




## 4. 깃의 동작 한눈에 보기

### ➤ 깃의 동작 한눈에 보기

#### ▼ 그림 1-4 깃의 다섯 단계



- 초기화 : 폴더를 깃 저장소로 변경
- 커밋 : 변경된 코드의 이력을 기록
- 브랜치 : 분리 격리된 코드 이력을 기록
- 병합 : 기존 이력과 분리된 이력을 통합
- 푸시 : 로컬 저장소의 이력을 서버로 전송 및 공유

## 1.5 정리





## 5. 정리

### ➤ 정리

- 깃은 다양한 알고리즘과 기술로 무장하고 있음
- 많은 프로젝트를 깃으로 관리하면서 안정성과 가치를 인정받고 있음
- 깃을 잘 활용하려면 깃의 개념을 이해하고 많이 사용해 보아야 함