



# Git 개요

- 소스코드 버전 관리
- Git 기초
- Git 활용

## [교재]

The entire Pro Git book

(Scott Chacon and Ben Straub, 2014, Apress)

<https://git-scm.com/book/en/v2>

## [교재-2]

누구나 쉽게 이해할 수 있는 Git 입문

<https://backlog.com/git-tutorial/kr/>



# 소스코드 버전 관리

## ❑ 버전 관리(Version Control, Revision Control, Source Control)

- 동일한 정보에 대한 여러 버전을 관리하는 것 (Wikipedia)
- 시간적으로 변경 사항과 그 변경 사항을 작성한 작업자를 추적
- 팀 단위로 개발 중인 소스 코드나 설계도 등의 디지털 문서를 관리
- 변경 사항들에 숫자나 문자로 이뤄진 '버전'을 부여해서 구분
- Version Control System(VCS)

## ❑ 소프트웨어 버전 관리(Software Version Management)

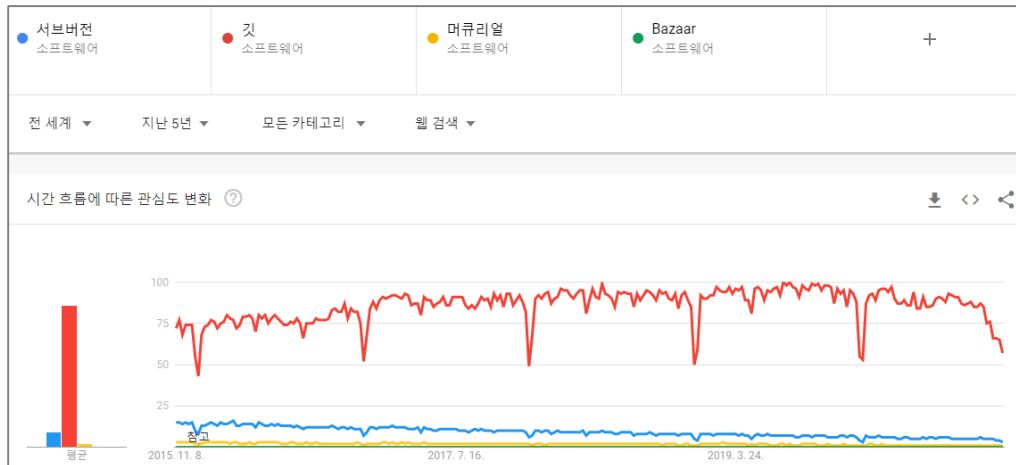
- 코드 수정될 때마다 다른 파일로 저장하거나 수동 백업하는 것은 번거로움
- 소스 코드만을 관리하는 것을 주로 '버전 관리'라고 정의
- 특정 시점 및 변경 추적, 버전 관리, 협업을 위한 코드 공유, 접근 통제 등



# 소스코드 버전 관리

## □ 소프트웨어 버전 관리 시스템(Software Version Management System)

- 소스 코드가 변경된 부분을 모두 기억해주는 소프트웨어 시스템
- 종류: SVN, Git, Mercurial, Bazaar 등
- 가장 많이 사용하는 SVM: Git, SVN



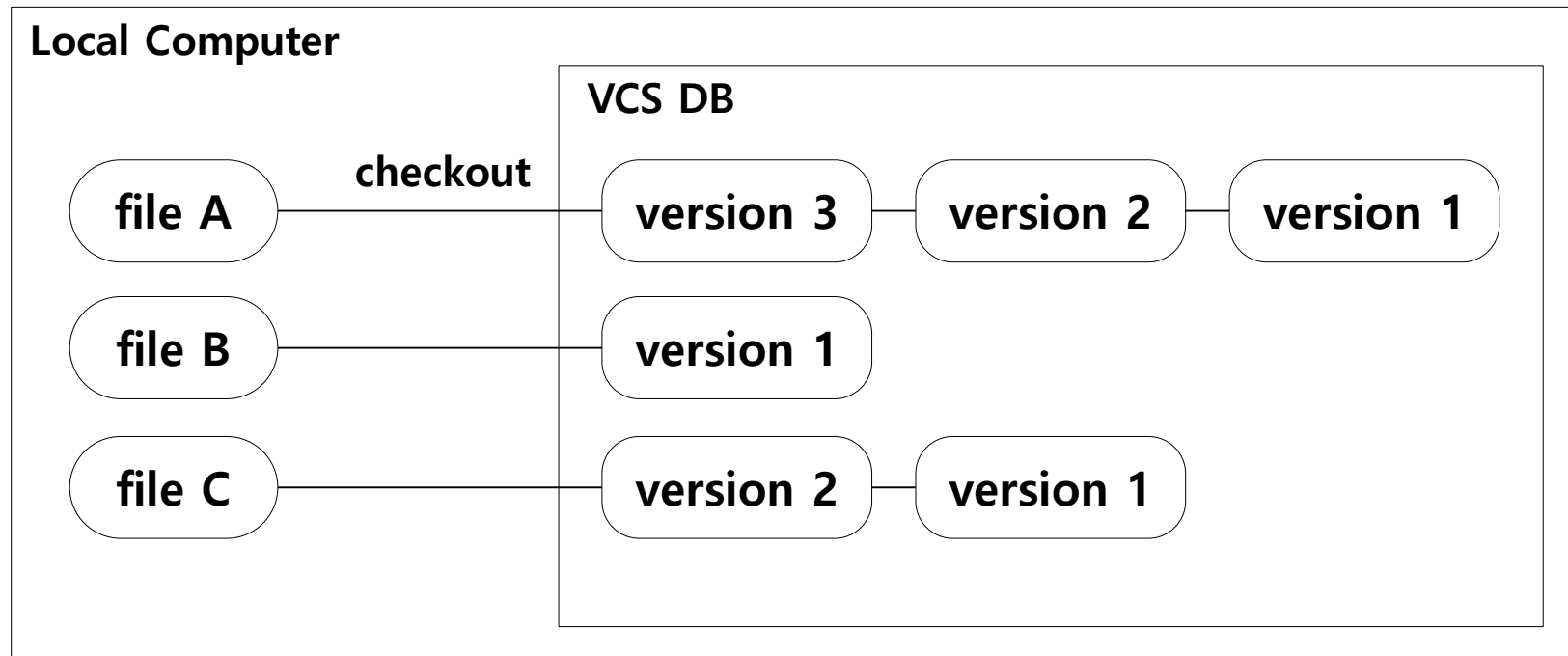


# 소스코드 버전 관리

## □ 버전 관리 시스템(VCS) 형태(1)

### ▪ Local VCS

- 오직 로컬에서만 변경된 부분(patch-set)에 대한 버전 관리 (협업 불가)
- 디렉터리를 이용한 파일 시스템이나 DB를 활용



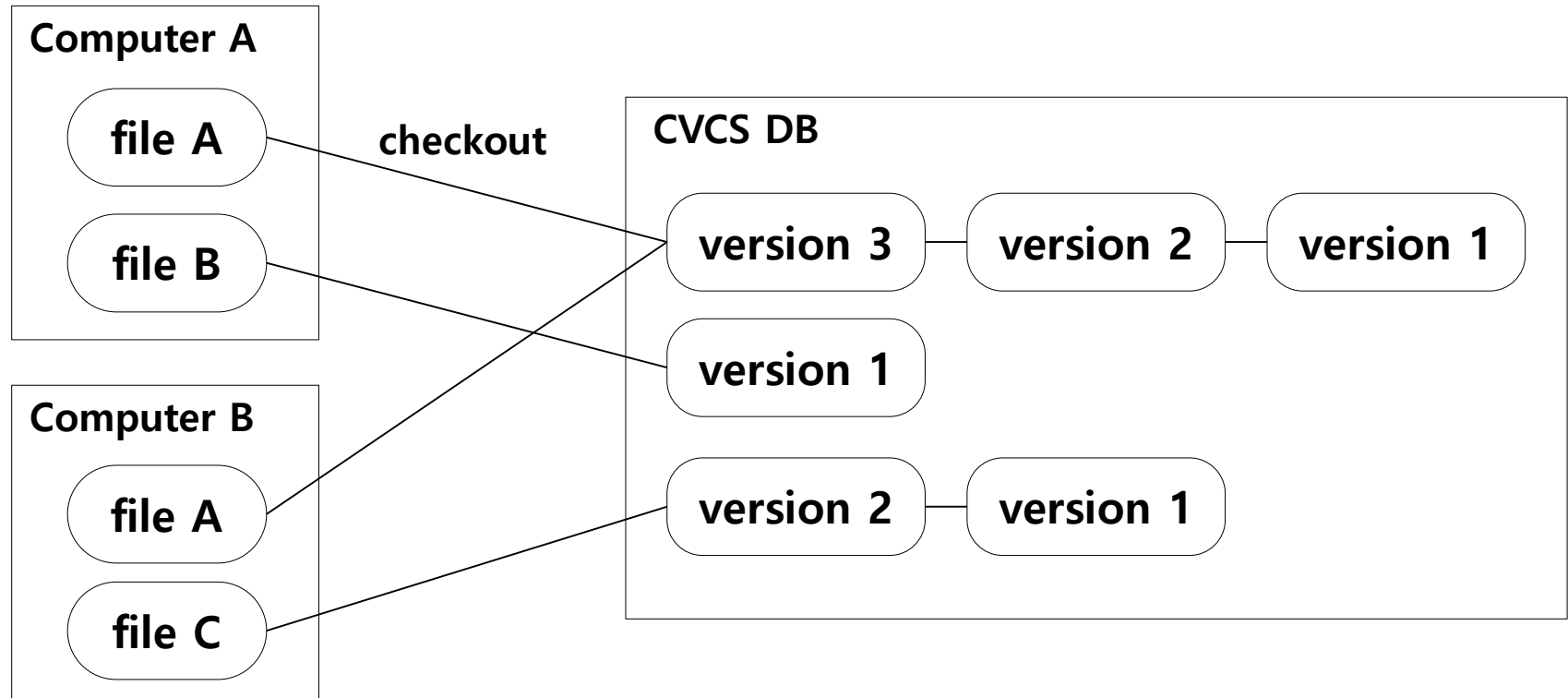


# 소스코드 버전 관리

## □ 버전 관리 시스템(VCS) 형태(2)

### ▪ Centralized VCS (중앙 집중형, CVCS)

- 중앙 서버에 저장소를 두고 클라이언트 요청에 의한 patch-set 관리
- 중앙 서버에 문제가 발생하면 협업 불가 (ex. SVN)



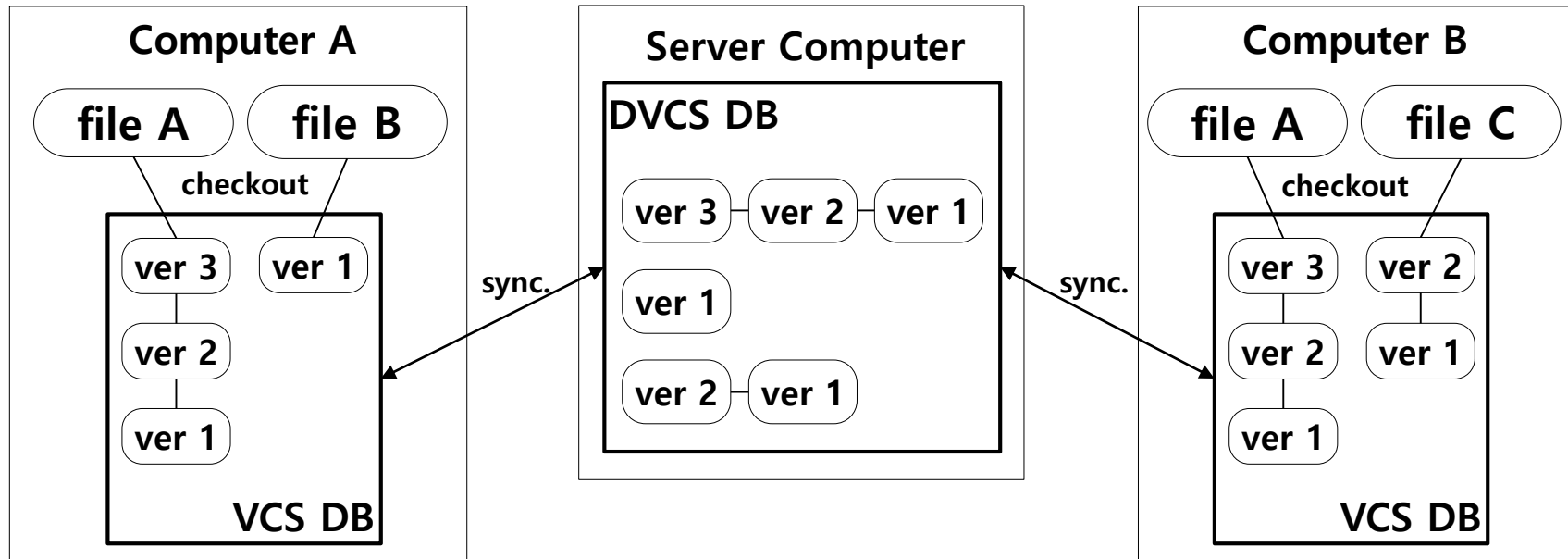


# 소스코드 버전 관리

## □ 버전 관리 시스템(VCS) 형태(3)

### ▪ Distributed VCS (분산형, DVCS)

- 클라이언트와 서버에 저장소를 두고 동기화를 통한 patch-set 관리
- 원격 저장소 접근 못하면 오프라인에서 작업 가능 (ex. Git)

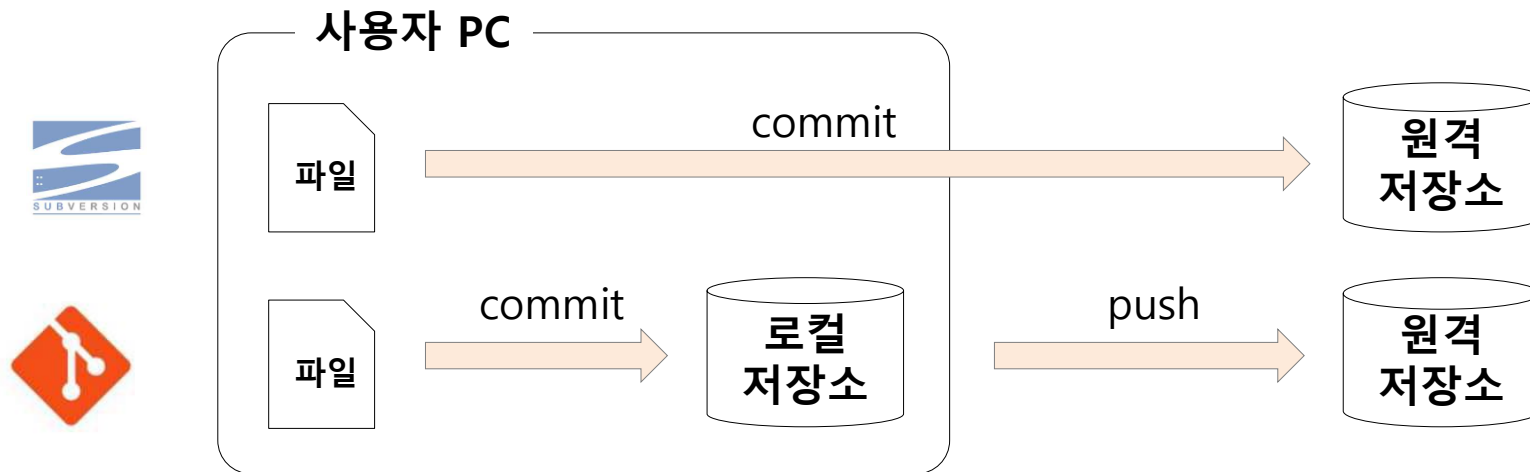




# 소스코드 버전 관리

## □ SVN(Subversion) vs. Git

SVN	Git
중앙 집중 모델	분산 개발 모델
소규모 시스템 개발에 적합	대규모 시스템 개발에 적합
간단한 기능과 간편한 조작	높은 오픈 소스 활용과 다양한 기능
브랜치 병합 작업 부담이 큼	커밋할 때 메시지 필수
저장소 서버의 지속적 관리 필요	사설 저장소 이용에 대한 비용 발생
안정된 네트워크 환경 필요	사용법에 대한 학습 필요



# Git 기초

## □ Git(깃)

- 분산형 버전 관리 시스템(DVCS) 소프트웨어
- 2002년부터 리눅스 커널 배포를 위해 무료로 사용하던 BitKeeper가 2005년 유료화 요청으로 관계가 틀어지면서 리눅스 토발즈가 직접 git을 만들어 지금까지 발전 시킴

### ▪ Git 개발 목표

- 빠른 속도
- 단순한 구조
- 비선형적 개발 (동시성)
- 완벽한 분산
- Linux 커널 같은 대형 프로젝트에도 유용할 것 (속도/크기 측면)

#### 짧게 보는 Git의 역사

우리네 삶의 삼라만상처럼 Git 또한 창조적 파괴와 활활 타오르는 갈등 속에서 시작됐다.

Linux 커널은 굉장히 규모가 큰 오픈소스 프로젝트다. Linux 커널의 삶 대부분은(1991~2002) Patch와 단순 압축 파일로만 관리했다. 2002년에 드디어 Linux 커널은 BitKeeper라고 불리는 상용 DVCS를 사용하기 시작했다.

2005년에 커뮤니티가 만드는 Linux 커널과 이익을 추구하는 회사가 개발한 BitKeeper의 관계는 틀어졌다. BitKeeper의 무료 사용이 재고된 것이다. 이 사건은 Linux 개발 커뮤니티(특히 Linux 창시자 Linus Torvalds)가 자체 도구를 만드는 계기가 됐다. Git은 BitKeeper를 사용하면서 배운 교훈을 기초로 아래와 같은 목표를 세웠다.

- 빠른 속도
- 단순한 구조
- 비선형적인 개발(수천 개의 동시 다발적인 브랜치)
- 완벽한 분산
- Linux 커널 같은 대형 프로젝트에도 유용할 것(속도나 데이터 크기 면에서)

Git은 2005년 탄생하고 나서 아직도 초기 목표를 그대로 유지하고 있다. 그러면서도 사용하기 쉽게 진화하고 성숙했다. Git은 미친 듯이 빨라서 대형 프로젝트에 사용하기도 좋다. Git은 동시다발적인 브랜치에도 끄떡없는 슈퍼 울트라 브랜칭 시스템이다([Git 브랜치](#) 참고).

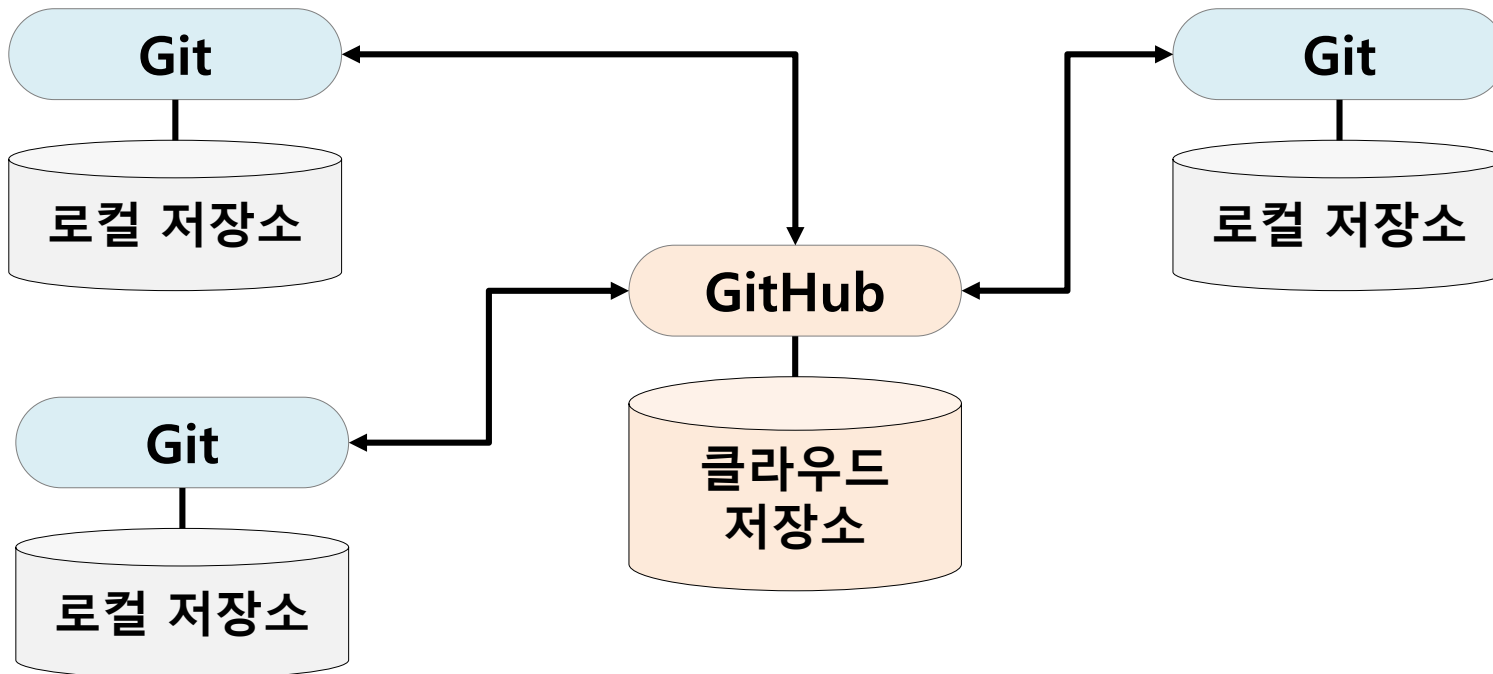




# Git 기초

## □ GitHub(깃허브)

- 클라우드 방식으로 관리되는 버전 관리 시스템(VCS)
- 자체 구축이 아닌 빌려쓰는 클라우드 개념
- 오픈 소스는 일정 부분 무료로 저장 가능, 아닐 경우 유료 사용



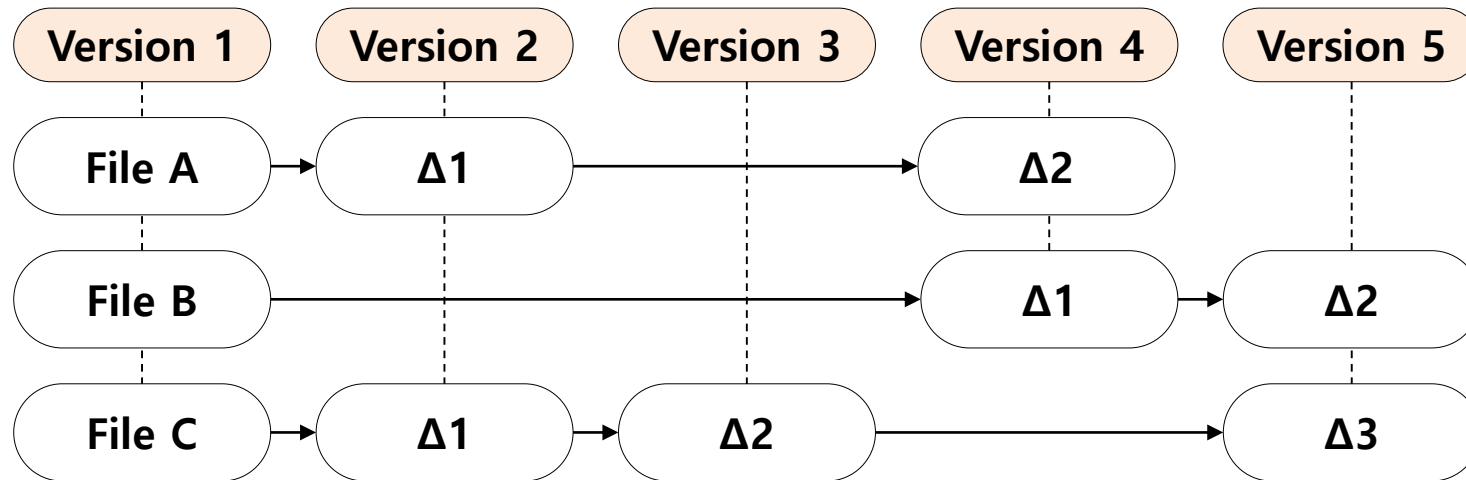
# Git 기초

## □ Git 다운로드

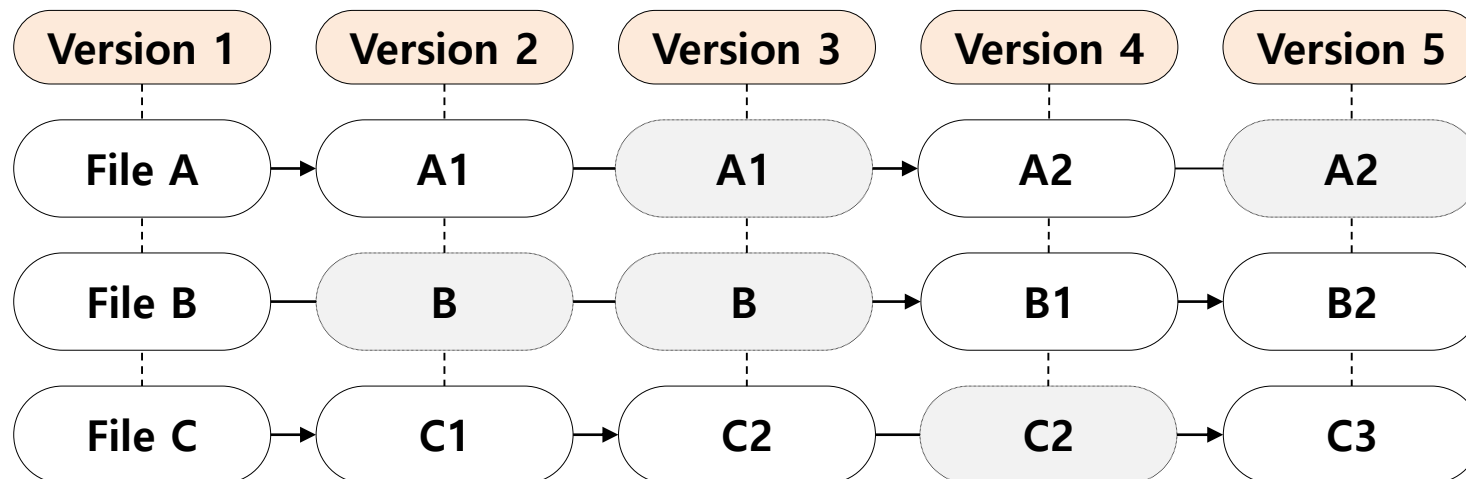
- 저장소 : <https://git.kernel.org/pub/scm/git/git.git/>  
(안정화 버전: 2.28.0)
- Linux 버전 다운로드: <https://git-scm.com/download/linux>
- 윈도우 버전 다운로드: <https://gitforwindows.org>  
<https://git-scm.com/download/win>
- Git for Windows SDK: <https://github.com/git-for-windows/build-extra/releases/tag/git-sdk-1.0.7>

# Git 기초

- 기존 VCS의 데이터 관리: 각 파일의 변화를 시간순으로 관리(그 차이를 저장)



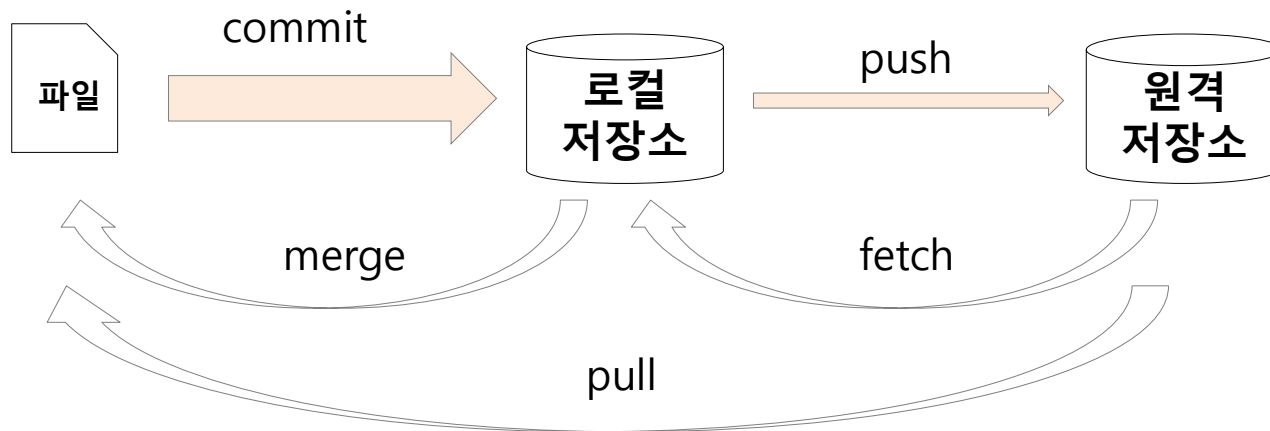
- Git의 데이터 관리 : 파일 시스템의 **스냅샷**으로 취급



# Git 기초

## □ 거의 모든 명령을 로컬에서 실행

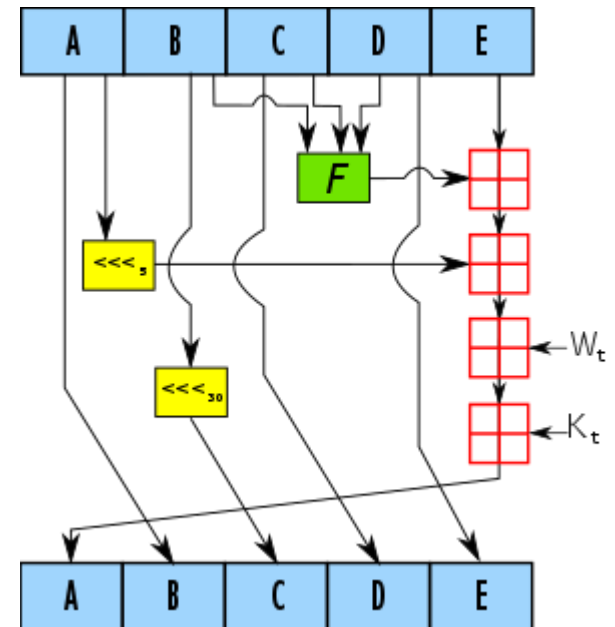
- 거의 모든 명령이 로컬 파일과 로컬 데이터만 사용
- 프로젝트의 모든 히스토리가 로컬 디스크에 존재  
=> 히스토리 조회 및 명령 실행 속도 빠름
- 오프라인 상태에서도 문제 없이 작업하고 커밋 가능  
=> 필요하다면 원격 저장소 활용 가능



# Git 기초

## □ Git의 데이터 무결성

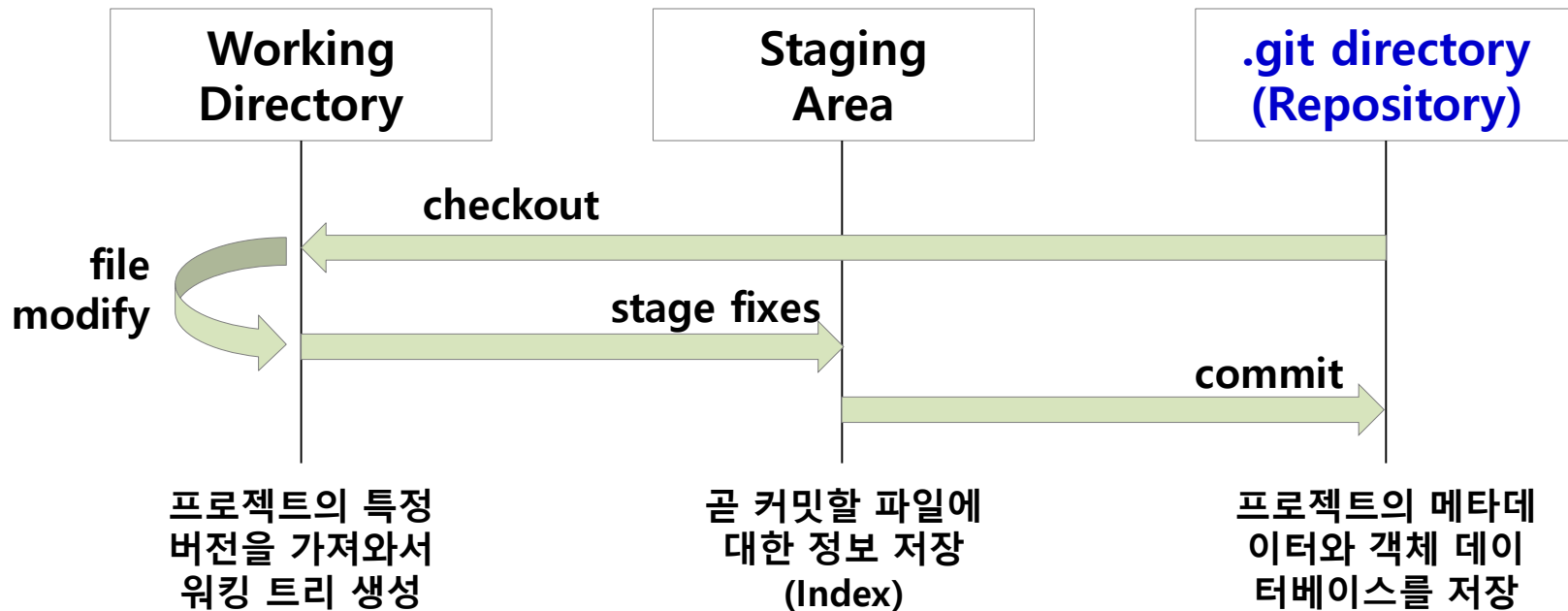
- 저장될 데이터에 대한 SHA-1 해시값을 사용하여 체크섬 생성
- 체크섬은 40자 길이의 16진수 문자열  
ex> 24b9da6552252987aa493b52f8696cd6d3b00373
- Git은 파일을 이름으로 저장하지 않고 해당 파일의 해시로 저장
- 각 명령에서는 체크섬 40자에서 6자리만 사용해도 식별 가능



# Git 기초

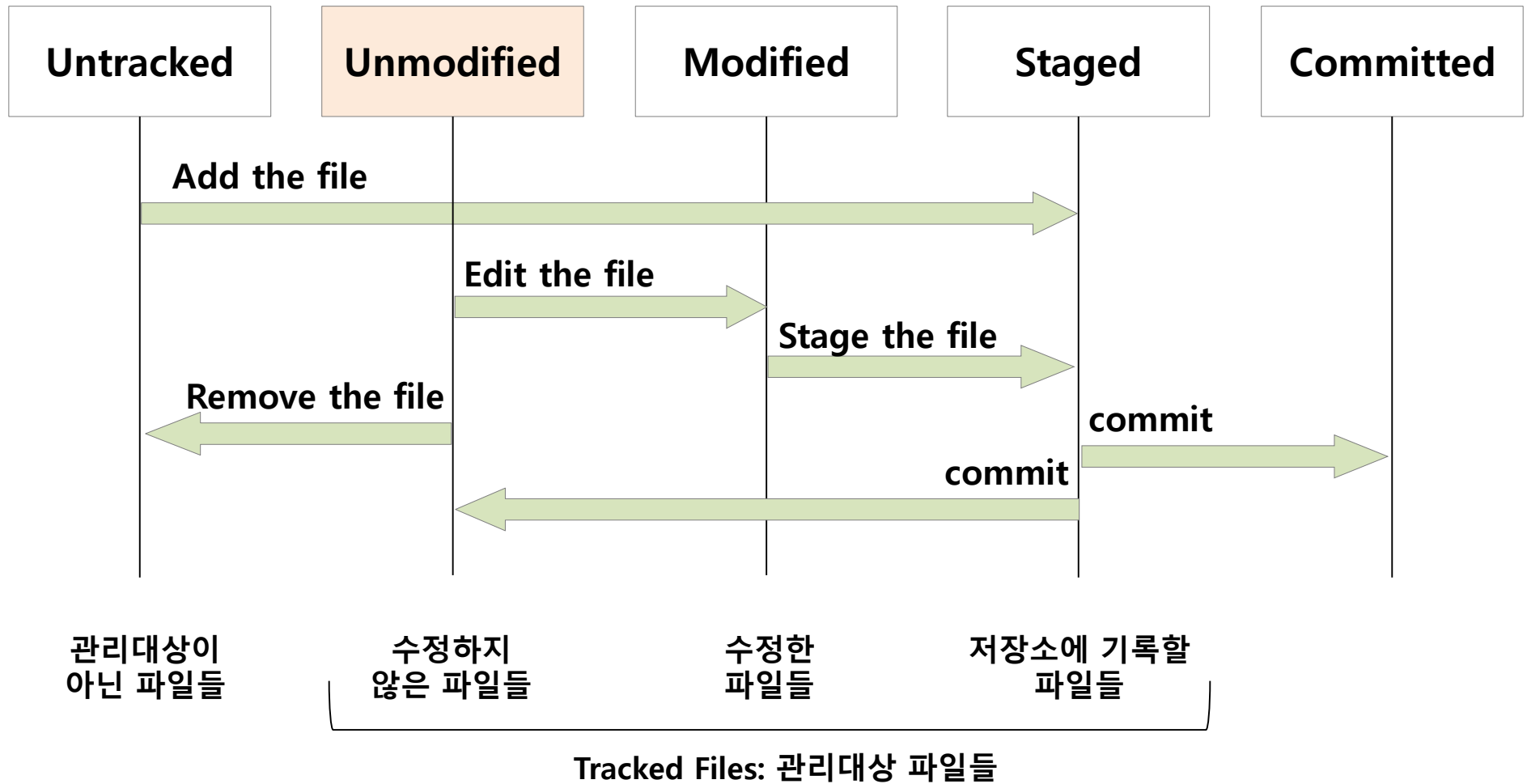
## □ 파일 관리를 위한 세 가지 상태

- **Committed**: 데이터가 로컬 데이터베이스에 안전하게 저장된 상태
- **Modified**: 수정한 파일 중에서 아직 커밋하지 않은 상태
- **Staged**: 현재 수정한 파일을 곧 커밋할 것이라고 표시한 상태
- **snapshot 생성 단계**: 원본파일 → Modified → Staged → Committed



# Git 기초

## □ 파일의 상태 변화 (Git File Lifecycle)



# Git 기초

## □ Git 용어 정리(1)

- **storage(저장소):** 변경 이력 관리 대상 파일이나 디렉토리를 저장하는 장소
  - Local Storage : 관리 대상을 사용자 컴퓨터에 저장
  - Remote Storage : 관리 대상을 서버나 네트워크에 저장
- **branch:** 병렬로 진행되는 여러 버전 관리를 위한 분기점  
동일 저장소에서 각 지점별 독립 버전 관리 가능
- **checkout:** 현재 작업중인 워킹 트리의 일부 혹은 전체를 업데이트하는 것  
(clone과 동일)
- **commit:** 파일을 추가하거나 변경된 내용을 storage에 저장하는 작업
- **push:** 파일을 추가하거나 변경된 내용을 remote storage에 업로드
- **fetch:** remote storage에서 branch의 head ref를 가져오는 것
- **merge:** 현재 작업 중인 branch에 다른 branch를 합치는 것



# Git 기초

## □ Git 용어 정리(2)

- **fast-forward**: master branch의 내용을 다른 branch의 최신 내용으로 업데이트 하는 작업
- **pull**: branch의 내용을 fetch한 후 merge하는 것
- **head**: branch의 제일 앞에 있는 commit을 가리키는 레퍼런스
- **HEAD**: 현재 branch를 지칭
- **index**: 변경 내역이 포함된 파일들의 모음 ('stage area'로 변경되고 있음)
- **rebase**: 현재 branch를 다른 branch의 head로 이동시키는 것

# Git 활용

## □ Git 설치

- CLI : Git의 모든 기능을 지원 (Terminal, cmd, Powershell 등 사용)
- GUI : 편리성을 고려한 CLI 일부 기능만 제공 (gitk, Sourcetree 등)
- Linux (Ubutu)

```
linuxer@linuxer-PC:~$ sudo apt-get install git-all
```

- Mac (OSX) : <http://git-scm.com/download/mac>
- Windows
  - Git for Windows : <http://git-scm.com/download/win>
  - GitHub for Windows: <http://windows.github.com>

# Git 활용

## □ Git 최초 설정(1)

- /etc/gitconfig : 시스템의 모든 사용자와 모든 저장소에 적용되는 설정

```
linuxer@linuxer-PC:~$ git config --system
```

- ~/.gitconfig, ~/.config/git/config : 특정 사용자에게 적용되는 설정

```
linuxer@linuxer-PC:~$ git config --global
```

- .git/config : 특정 저장소에만 적용되는 설정

- 사용자 정보 설정

```
linuxer@linuxer-PC:~$ git config --global user.name "<영문이름>"
```

```
linuxer@linuxer-PC:~$ git config --global user.email "<개인 이메일>"
```

- 편집기 설정

```
linuxer@linuxer-PC:~$ git config --global core.editor vi
```

# Git 활용

## □ Git 최초 설정(2)

### ▪ 설정 상태 확인

```
linuxer@linuxer-PC:~$ git config --list  
linuxer@linuxer-PC:~$ git config user.name  
linuxer@linuxer-PC:~$ git config user.email  
linuxer@linuxer-PC:~$ git config core.editor
```

### ▪ 도움말 보기

```
linuxer@linuxer-PC:~$ git help <명령>  
linuxer@linuxer-PC:~$ git <명령> --help  
linuxer@linuxer-PC:~$ man git-<명령>  
linuxer@linuxer-PC:~$ git help config
```

# Git 활용

## □ Git 저장소 만들기(1)

- 기존 디렉터리를 Git 저장소로 만들기  
: 프로젝트용 디렉터리를 Git으로 관리하고 싶을 때 활용

```
linuxer@linuxer-PC:~$ cd gcc_test
linuxer@linuxer-PC:~/gcc_test$ git init
linuxer@linuxer-PC:~/gcc_test$ ls -aF
./ ../ .git/ hello/ long/ long2/ long3/ main/ reverse/ test/
linuxer@linuxer-PC:~/gcc_test$ ls -F .git
HEAD branches/ config description hooks/ info/ objects/ refs/
```

# Git 활용

## □ Git 저장소 만들기(2)

- 저장소에 파일 추가 및 커밋 : **add** , **commit**

```
linuxer@linuxer-PC:~/gcc_test$ git add *.c
linuxer@linuxer-PC:~/gcc_test$ git commit -m 'initial project version'
[master (최상위-커밋) 4767b4c] initial project version
13 files changed, 207 insertions(+)
create mode 100644 hello/hello.c
create mode 100755 long/long.c
create mode 100755 long2/copy.c
create mode 100755 long2/main.c
...
create mode 100644 main/second.c
create mode 100755 reverse/reverse.c
create mode 100755 test/test1.c
create mode 100755 test/test2.c
create mode 100755 test/test3.c
linuxer@linuxer-PC:~/gcc_test$
```

# Git 활용

## □ Git 저장소 만들기(3)

- 기존 저장소를 복제 하기: **clone**

: 다른 프로젝트의 소스 코드와 history를 모두 받아옴

```
linuxer@linuxer-PC:~$ git clone http://github.com/libgit2/libgit2
```

또는

```
linuxer@linuxer-PC:~$ git clone http://github.com/libgit2/libgit2 mylibgit
```

```
linuxer@linuxer-PC:~$ ls libgit2
```

AUTHORS	SECURITY.md	cmake	fuzzers	script
CMakeLists.txt	api.docurium	deps	git.git-authors	src
COPYING	azure-pipelines	docs	include	tests
README.md	azure-pipelines.yml	examples	package.json	

```
linuxer@linuxer-PC:~$ ls -F libgit2/.git
```

HEAD	config	hooks/	info/	objects/	refs/
branches/	description	index	logs/	packed-refs	

# Git 활용

## □ 수정하고 저장소에 저장하기(1)

### ▪ Clone한 파일의 상태 확인: **status**

```
linuxer@linuxer-PC:~$ git status
fatal: (현재 폴더 또는 상위 폴더 중 일부가) 깃 저장소가 아닙니다: .git
```

```
linuxer@linuxer-PC:~$ cd libgit2/
linuxer@linuxer-PC:~/libgit2$ git status
현재 브랜치 master
브랜치가 'origin/master'에 맞게 업데이트된 상태입니다.
```

```
커밋할 사항 없음, 작업 폴더 깨끗함
linuxer@linuxer-PC:~/libgit2$
```



# Git 활용

## □ 수정하고 저장소에 저장하기(2)

### ▪ 새로운 파일 생성 및 상태 확인

```
linuxer@linuxer-PC:~/libgit2$ echo 'My Project' > README
```

```
linuxer@linuxer-PC:~/libgit2$ git status
```

현재 브랜치 master

브랜치가 'origin/master'에 맞게 업데이트된 상태입니다.

추적하지 않는 파일:

(커밋할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)

**README**

커밋할 사항을 추가하지 않았지만 추적하지 않는 파일이 있습니다 (추적하려면 "git add"를 사용하십시오)

```
linuxer@linuxer-PC:~/libgit2$
```

# Git 활용

## □ 수정하고 저장소에 저장하기(3)

### ▪ 파일을 추적(tracked) 하기

```
linuxer@linuxer-PC:~/libgit2$ git add README
linuxer@linuxer-PC:~/libgit2$ git status
현재 브랜치 master
브랜치가 'origin/master'에 맞게 업데이트된 상태입니다.
```

커밋할 변경 사항:

(use "git restore --staged <file>..." to unstage)

새 파일: README

```
linuxer@linuxer-PC:~/libgit2$
```

## □ 수정하고 저장소에 저장하기(4)

### ▪ Modified 상태의 파일을 Stage 상태로 변경하기(1)

```
linuxer@linuxer-PC:~/libgit2$ vi AUTHORS
... 저자 목록 끝에 본인의 영문 이름을 기록후 저장(:wq)
```

```
linuxer@linuxer-PC:~/libgit2$ git status
현재 브랜치 master
브랜치가 'origin/master'에 맞게 업데이트된 상태입니다.
```

커밋할 변경 사항:

(use "git restore --staged <file>..." to unstage)

새 파일: **README**

커밋하도록 정하지 않은 변경 사항:

(무엇을 커밋할지 바꾸려면 "git add <파일>..."을 사용하십시오)

(use "git restore <file>..." to discard changes in working directory)

수정함: **AUTHORS**

```
linuxer@linuxer-PC:~/libgit2$
```

# Git 활용

## □ 수정하고 저장소에 저장하기(4)

### ▪ Modified 상태의 파일을 Stage 상태로 변경하기(2)

```
linuxer@linuxer-PC:~/libgit2$ git add AUTHORS
```

```
linuxer@linuxer-PC:~/libgit2$ git status
```

현재 브랜치 master

브랜치가 'origin/master'에 맞게 업데이트된 상태입니다.

커밋할 변경 사항:

(use "git restore --staged <file>..." to unstage)

수정함: AUTHORS

새 파일: README

```
linuxer@linuxer-PC:~/libgit2$
```

## □ 수정하고 저장소에 저장하기(4)

### ▪ Modified 상태의 파일을 Stage 상태로 변경하기(3)

```
linuxer@linuxer-PC:~/libgit2$ vi AUTHORS
... 저자 목록 끝에 친구의 영문 이름을 기록후 저장(:wq)
```

```
linuxer@linuxer-PC:~/libgit2$ git status
현재 브랜치 master
브랜치가 'origin/master'에 맞게 업데이트된 상태입니다.
```

커밋할 변경 사항:

(use "git restore --staged <file>..." to unstage)

수정함:       AUTHORS  
새 파일:       README

커밋하도록 정하지 않은 변경 사항:

(무엇을 커밋할지 바꾸려면 "git add <파일>..."을 사용하십시오)

(use "git restore <file>..." to discard changes in working directory)

수정함:       AUTHORS

# Git 활용

## □ 수정하고 저장소에 저장하기(4)

### ▪ Modified 상태의 파일을 Stage 상태로 변경하기(4)

```
linuxer@linuxer-PC:~/libgit2$ git add AUTHORS
```

```
linuxer@linuxer-PC:~/libgit2$ git status
```

현재 브랜치 master

브랜치가 'origin/master'에 맞게 업데이트된 상태입니다.

커밋할 변경 사항:

(use "git restore --staged <file>..." to unstage)

수정함:       AUTHORS

새 파일:       README

```
linuxer@linuxer-PC:~/libgit2$ git status -s
```

M AUTHORS

A README

# Git 활용

## □ 관리 대상에서 제외 시키기

- 로그 파일이나 빌드하면서 임시로 생성된 파일들은 관리 대상에서 제외 시킬 필요 있음
- **.gitignore** 파일에 제외할 파일 패턴을 기술

```
linuxer@linuxer-PC:~/libgit2$ cat .gitignore
build/                                => build 디렉터리 모두 제외
.DS_Store
*.oa                                  => 확장자가 .o 또는 .a 파일 제외
*~                                    => ~로 끝나는 모든 파일 제외
*.swp
tags
CMakeSettings.json
.vs
!lib.a                                => 확장자 .a 중에 lib.a 파일은 포함
linuxer@linuxer-PC:~/libgit2$
```

# Git 활용

## □ 변경된 내용을 상세하게 확인하기: **diff**

- 워킹 디렉터리에 있는 것과 Staging Area에 있는 것을 비교해서 표시함

```
linuxer@linuxer-PC:~/libgit2$ vi AUTHORS
... 마지막 행에 임의의 영문 이름 추가 및 저장(:wq)
```

```
linuxer@linuxer-PC:~/libgit2$ git diff
diff --git a/AUTHORS b/AUTHORS
index e2aecb921..93b5359fd 100644
--- a/AUTHORS
+++ b/AUTHORS
@@ -78,3 +78,4 @@ Trent Mick
  Vicent Marti
  JongYeol Lee
  test user
+old user
```

<storage에 커밋한 것과 staging area에 있는 것을 비교하려면>

```
linuxer@linuxer-PC:~/libgit2$ git diff --staged
```

....



## □ 변경사항 커밋하기: **commit**

linuxer@linuxer-PC:~/libgit2\$ git **commit**



# 변경 사항에 대한 커밋 메시지를 입력하십시오. '#' 문자로 시작하는  
# 줄은 무시되고, 메시지를 입력하지 않으면 커밋이 중지됩니다.

#

(생략...)

#

# 커밋하도록 정하지 않은 변경 사항:

#     수정함:       AUTHORS

:q!

linuxer@linuxer-PC:~/libgit2\$ git **commit -m** "commit test..."

[**master aaf3e18fb**] commit test...

3 files changed, 4 insertions(+)

create mode 100644 README

- branch: **master**
- checksum: **aaf3e18fb**

# Git 활용

## □ Staging Area 생략하기

- Tracked 상태에 있는 파일들을 Stage Area에 보관하고(**git add**), 다시 Local Repository에 보관하기(**git commit**) 위해 두 번의 명령을 수행하는 것을 한 번으로 해결 가능

```
linuxer@linuxer-PC:~/libgit2$ git commit -a -m 'new version'
```

- **-a** 옵션을 사용하면 모든 파일이 자동으로 커밋하는데 포함 됨  
=> 커밋될 필요 없는 파일도 함께 커밋되기 때문에 주의 요망

## □ 파일 삭제하기(1)

- Stage Area에 있는 Tracked 파일을 삭제: **rm**
  - stage area에서 삭제되면 실제 디렉터리에서도 삭제됨 (주의 요망)
  - 강제로 삭제하려면 **-f**, stage area에서만 삭제하려면 **--cached** 사용

```
linuxer@linuxer-PC:~/gcc_test$ git status
```

현재 브랜치 master

커밋할 변경 사항:

(use "git restore --staged <file>..." to unstage)

새 파일: rmtest.txt

```
linuxer@linuxer-PC:~/gcc_test$ git rm rmtest.txt
```

error: 다음 파일이 인덱스에 스테이징한 변경 사항이 있습니다:

rmtest.txt

(파일을 유지하려면 **--cached** 옵션, 강제로 제거하려면 **-f** 옵션을 사용하십시오)

```
linuxer@linuxer-PC:~/gcc_test$ git status
```

현재 브랜치 master

커밋할 변경 사항:

(use "git restore --staged <file>..." to unstage)

새 파일: rmtest.txt

```
linuxer@linuxer-PC:~/gcc_test$ git rm --cached rmtest.txt
```

## □ 파일 삭제하기(2)

### ▪ file-glob 패턴 활용

- 유닉스 계열에서 와일드카드 문자로 여러 파일들 지정할 때 사용되는 패턴
- stage area에 있는 파일을 지정하기 위한 패턴 사용 가능

패턴	설명	사용예
*	/를 제외한 모든 문자열과 매칭(문자열 길이 0이상)	*.c
**	/를 포함한 모든 문자열과 매칭(문자열 길이 0이상)	/**/*.c
?	/를 제외한 하나의 문자와 매칭(빈 문자 미포함)	data_?.log
[abc]	[ ] 안에 있는 각 문자들과 매칭	ex[123].c
[^abc]	[ ] 안에 있는 각 문자들을 제외한 문자들과 매칭	ex[^123].c
[a-z]	[ ] 안의 첫 문자와 마지막 문자 범위에 있는 패턴 매칭	ex[1-3].c
{a,b,c}	{ } 안에 있는 콤마로 구분된 문자열들과 매칭	*.{c, cpp}

```
linuxer@linuxer-PC:~/gcc_test$ git rm log/W*.log => .log로 끝나는 모든 파일  
linuxer@linuxer-PC:~/gcc_test$ git rm W*~      => ~로 끝나는 모든 파일
```

# Git 활용

## □ 파일 이름 변경하기

- Git은 파일 이름을 관리하는 기능은 없지만, 파일 이름 변경 기능은 있음
- 파일 이름 변경은 단축 명령으로 가능: **mv**

```
linuxer@linuxer-PC:~/gcc_test$ cp ../libgit2/README .  
linuxer@linuxer-PC:~/gcc_test$ git status  
추적하지 않는 파일:  
    README
```

```
linuxer@linuxer-PC:~/gcc_test$ mv README README.md  
linuxer@linuxer-PC:~/gcc_test$ git status  
추적하지 않는 파일:  
    README.md
```

```
linuxer@linuxer-PC:~/gcc_test$ git add README.md  
linuxer@linuxer-PC:~/gcc_test$ git status  
커밋할 변경 사항:  
    새 파일:    README.md
```

```
linuxer@linuxer-PC:~/gcc_test$ git mv README.md README  
linuxer@linuxer-PC:~/gcc_test$ git status  
커밋할 변경 사항:  
    새 파일:    README
```

```
$ mv README.md README  
$ git rm README.md  
$ git add README
```

# Git 활용

## □ 커밋 히스토리 조회하기(1)

- 저장소의 커밋 히스토리를 시간순으로 출력: **log**

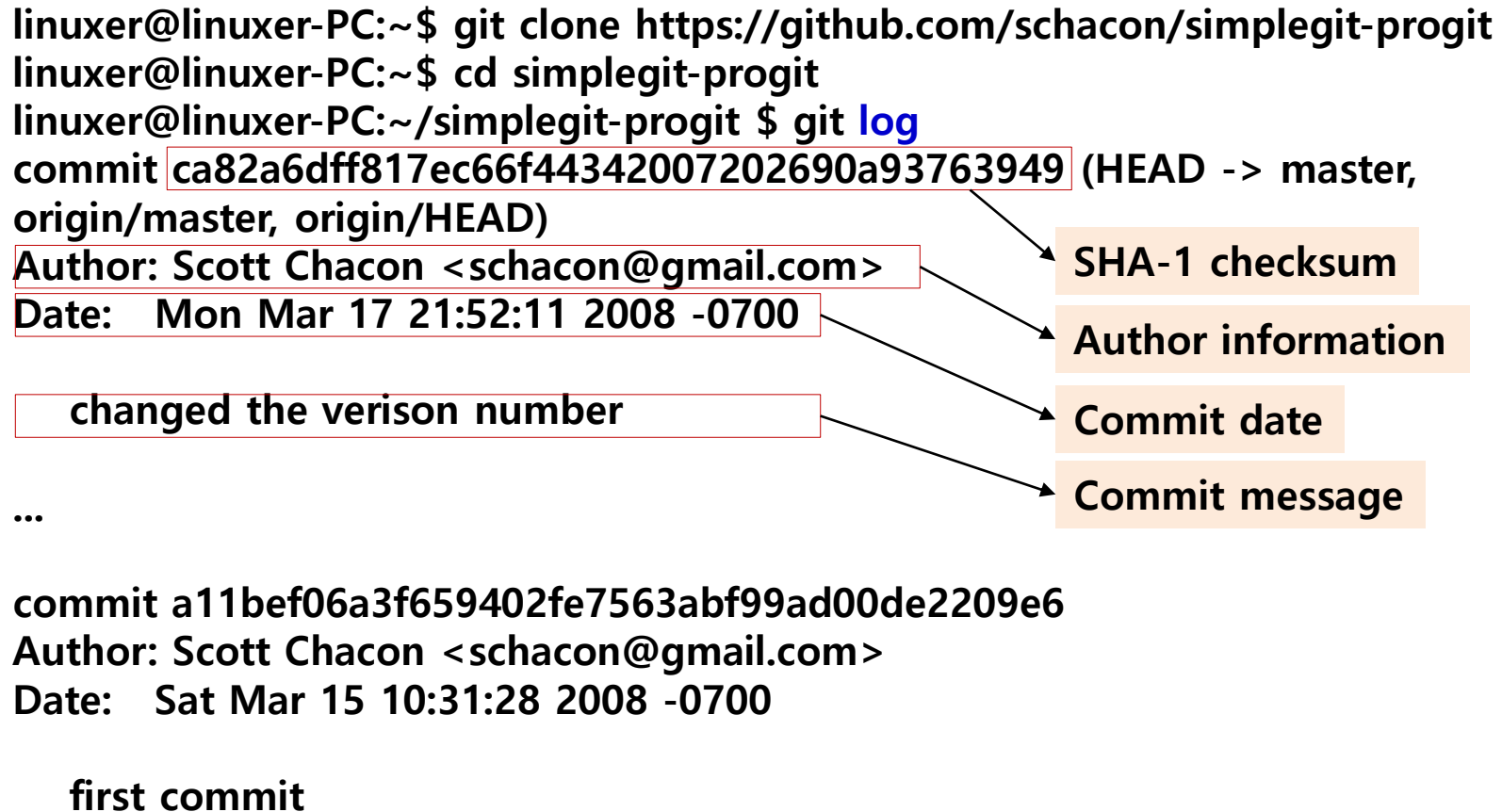
```
linuxer@linuxer-PC:~$ git clone https://github.com/schacon/simplegit-progit
linuxer@linuxer-PC:~$ cd simplegit-progit
linuxer@linuxer-PC:~/simplegit-progit $ git log
commit ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master,
origin/master, origin/HEAD)
Author: Scott Chacon <schacon@gmail.com>
Date: Mon Mar 17 21:52:11 2008 -0700

    changed the verison number

...

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gmail.com>
Date: Sat Mar 15 10:31:28 2008 -0700

    first commit
```



## □ 커밋 히스토리 조회하기(2)

- 각 커밋의 diff 결과 조회: **-p** 옵션
- 최근 몇 개의 결과만 조회: **-<개수>** 옵션
- 히스토리의 통계 조회: **--stat** 옵션

```
linuxer@linuxer-PC:~/simplegit-progit $ git log -p -2
```

```
...
diff --git a/Rakefile b/Rakefile
--- a/Rakefile
+++ b/Rakefile
```

```
...
- s.version = "0.1.0"
+ s.version = "0.1.1"
```

```
...
linuxer@linuxer-PC:~/simplegit-progit $ git log --stat
```

```
...
 README      | 6 ++++++
 Rakefile     | 23 ++++++++++++++++++++++
 lib/simplegit.rb | 25 ++++++++++++++++++++++
 3 files changed, 54 insertions(+)
```

## □ 커밋 히스토리 조회하기(3)

- 히스토리 조회 내용 선택: **--pretty** 옵션
  - =oneline : 각 커밋을 한 라인으로 출력
  - =short, =full, =fuller : 표시할 항목들을 다양하게 선택

```
linuxer@linuxer-PC:~/simplegit-progit$ git log --pretty=oneline
ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master,
origin/HEAD) changed the verison number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 removed unnecessary test code
a11bef06a3f659402fe7563abf99ad00de2209e6 first commit
```

```
linuxer@linuxer-PC:~/simplegit-progit$ git log --pretty=short
linuxer@linuxer-PC:~/simplegit-progit$ git log --pretty=fuller
```



## □ 커밋 히스토리 조회하기(4)

- 히스토리 조회 결과 커스트마이징: **=format** 옵션
  - 출력 포맷을 지정해서 다른 프로그램으로 파싱할 대 유용함

```
linuxer@linuxer-PC:~/simplegit-progit$ git log --pretty=format:"h - %an, %ar : %s"
h - Scott Chacon, 13년 전 : changed the verison number
h - Scott Chacon, 13년 전 : removed unnecessary test code
h - Scott Chacon, 13년 전 : first commit
```

옵션	설명
%H	커밋 해시
%h	짧은 길이 커밋 해시
%T	트리 해시
%t	짧은 길이 트리 해시
%P	부모 해시
%p	짧은 길이 부모 해시
%an	저자 이름
%ae	저자 메일

옵션	설명
%ad	저자 시간
%ar	저자 상대적 시간
%cn	커미터 이름
%ce	커미터 이메일
%cd	커미터 시간
%cr	커미터 상대적 시간
%s	요약

## □ 커밋 히스토리 조회하기(5)

- 그래프로 히스토리 조회 : **--graph** 옵션
  - 브랜치와 머지 히스토리를 ASCII 그래프로 출력

```
linuxer@linuxer-PC:~/libgit2$ git log --pretty=format:"h %s" --graph
* aaf3e18fb commit test...
* 2a5167900 Merge pull request #5659 from libgit2/ethomson/name_is_valid
|\
| * 8b0c7d7cd changelog: include new reference validity functions
| * 29715d408 refs: introduce git_reference_name_is_valid
| * d70979cf8 refs: error checking in internal name validation
|/
* 52294c413 Merge pull request #5685 from staticfloat/sf/mbedtls_inc_fix
|\
| * 1822b0827 Include `${MBEDTLS_INCLUDE_DIR}` when compiling `crypt_mbedtls.c`
|/
* b106834d7 Merge pull request #5668 from libgit2/ethomson/tlsdata
|\
| * 246bc3ccf threadstate: rename tlsdata when building w/o threads
|/
...
* c15648cbd Initial draft of libgit2
```

## □ 커밋 히스토리 조회하기(5)

### ▪ 조회 범위 제한 (조건 검색)

옵션	설명
-n	최근 n개의 커밋만 조회
--since, --after	명시한 날짜 이후의 커밋만 조회
--until, --before	명시한 날짜 이전의 커밋만 조회
--author	입력한 저자의 커밋만 조회
--committer	입력한 커미터의 커밋만 조회
--grep	커밋 메시지 안의 텍스트 검색
-S[검색어]	커밋 변경 내용 안의 텍스트 검색

```
linuxer@linuxer-PC:~/libgit2$ git log --pretty="h - %s" --since="2008-10-01" --before="2008-11-01" --no-merges
5610e3b - Fix testcase failure when extended attributes are in use
acd3b9e - Enhance hold_lock_file_for_{update,append}() API
...
b0ad11e - pull: allow "git pull origin $something:$current_branch" into an unborn branch
```