Git

동의과학대학교 컴퓨터정보과

Orientation

https://www.youtube.com/watch?v=Bd35Ze7-dIw

시작하기

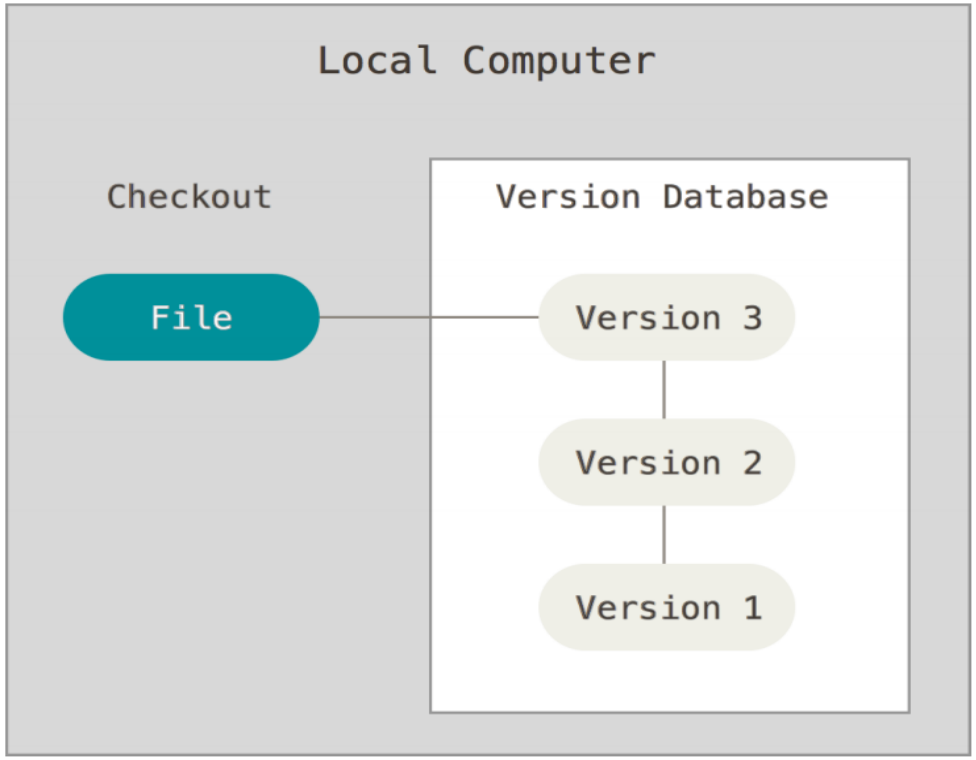
- 버전(Version)이란?
  - 의미 있는 파일의 변화 : 기능개선, 버그 수정, 커스터마이징 등
- 버전 관리(형상관리, 소스관리)란?
  - 의미 있는 파일의 변화 관리
  - 파일 변화를 시간에 따라 기록했다가 나중에 특정 시점의 버전을 다시 꺼내오는 것
- 보통 사람들이 사용하는 버전 관리
  - 파일 또는 디렉토리 전체를 별도의 디렉토리에 복사해 둔다.
    - (똑똑한 사람은 날짜, 소제목 등을 붙여서 저장하기도...)
  - 매우 간단하고 사용하기도 쉬운 방법
    - 실수로 지워버리거나, 잘못 복사하거나 하는 등의 위험에 항상 노출됨
  - 보다 체계적이고 안정적인 방법이 필요함

로컬 VCS (Local Version Control System)

- 사용하는 컴퓨터(local computer)에 간단한 데이터베이스를 만들어 파일 변경 정보를 기록 관리

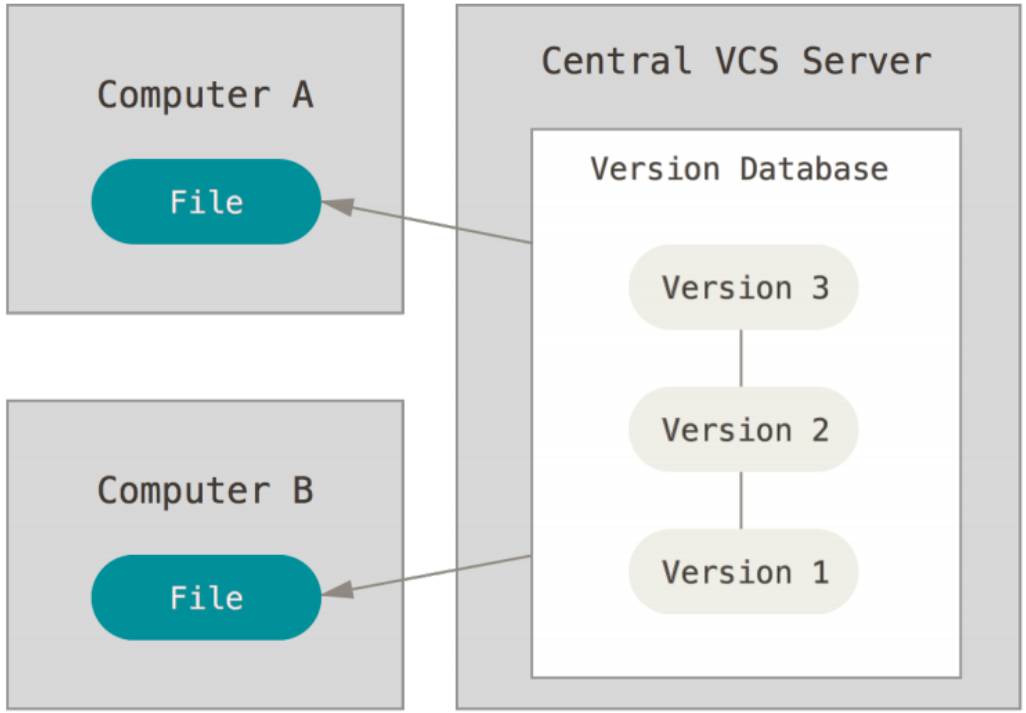
- 필요한 시점에 DB에서 특정 버전을 불러와서 사용

- 다른 개발자들과 함께 사용하기에는 문제가 많음



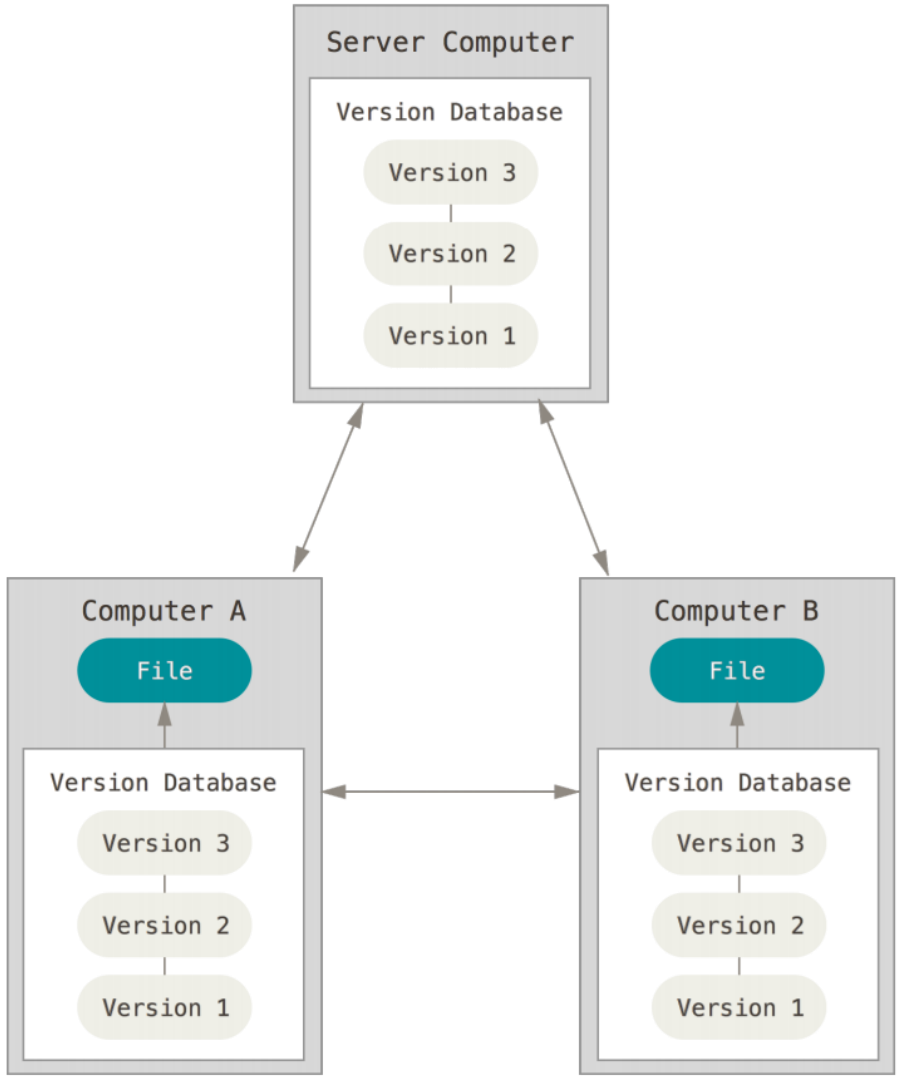
중앙집중식 VCS (Centralized VCS: CVCS)

- 파일을 관리하는 서버(CVCS Server)가 별도로 있고 클라이언트가 중앙 서버에서 파일을 받아서 사용(Checkout).
- 다른 개발자들과 함께 작업하기에 매우 편리함
  - 예) CVS, Subversion, Perforce 등
- 중앙 서버에 문제 발생시 작업 중단, 모든 자료 손실 등 치명적 한계가 있음



분산 VCS (Distributed VCS: DVCS)

- Client는 단순히 파일의 마지막 스냅샷을 복사하지 않고 그냥 저장소를 전부 복제
- 서버에 문제가 생겨도 local 또는 다른 client를 통해 완벽한 복원 가능
- 동시에 다양한 그룹과 다양한 방법 으로 협업 가능



- DVCS
  - 예) **Git**, Mecurial, Bazaar, Darcs 등

짧게 보는 Git의 역사

- Linux 커널은 굉장히 규모가 큰 오픈 소스 프로젝트
- 1991–2002 기간에는 Patch와 단순 압축 파일로만 버전 관리
- 2002년, Bitkeeper라는 상용 DVCS를 사용하여 관리 시작
  - 2005년에 이익 추구하는 Bitkeeper와 관계가 틀어지며
    Linux 개발 커뮤니티(리누스 토발즈)에서 자체 도구 개발 시작
- Git은 Bitkeeper 사용경험을 토대로 다음 목표를 설정, 개발됨
  - 빠른 속도
  - 단순한 구조
  - 비선형적 개발(수천 개의 동시다발적 브랜치)
  - Linux 커널 같은 대형 프로젝트에도 유용할 것(속도, 데이터 크기 등의 측면)

리눅스(Linux)의
단순한 버전 관리

Linux 창시자 Linus Tovalds를 중심으로
Linux 개발 커뮤니티에서 자체 툴 개발

1991

2002

2005

BitKeeper
Scalable Version Control

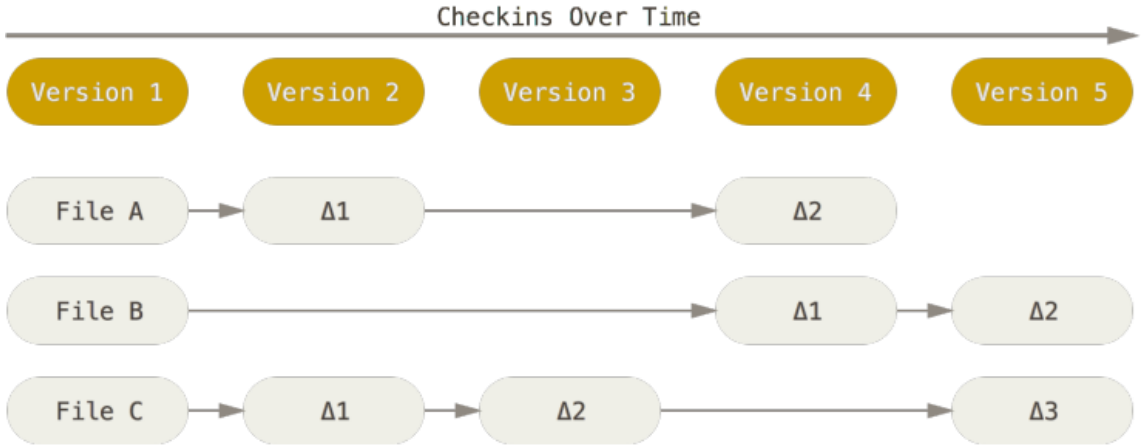
리눅스(Linux) 커널에서
DVCS인 BitKeeper 사용

- 빠른 속도
- 단순한 구조
- 비선형적인 개발
- 완벽한 분산
- Linux 커널 같은 대형 프로젝트에도 유용할 것

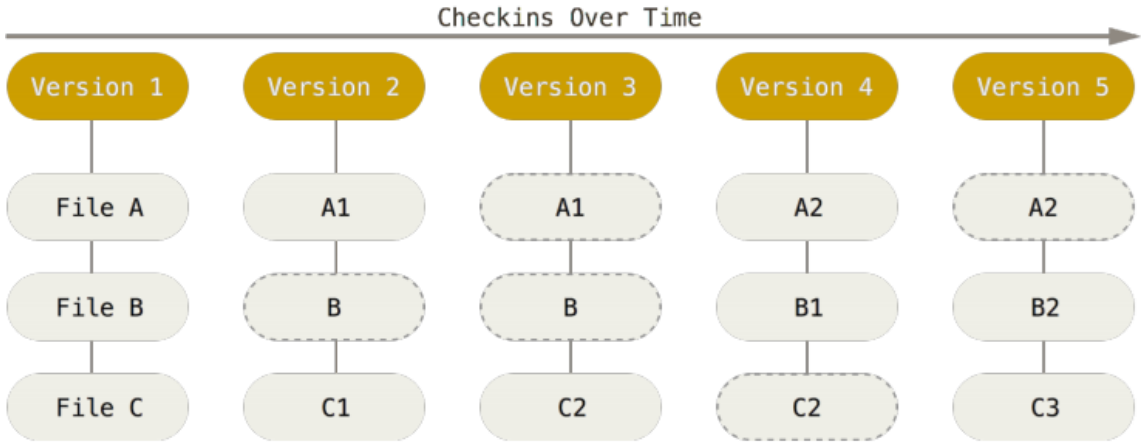
git

Git 기초 (1)

- Git의 핵심 동작 개념을 이해한다면 쉽고 효과적으로 사용 가능
- Git은 차이가 아니라 스냅샷
  - **기존 시스템**
    - 각 파일의 변화를 시간 순으로 관리하며 파일들의 집합을 관리



  - **Git**
    - 프로젝트의 상태를 저장할 때마다 파일이 존재하는 그 순간을 중요하게 다룸
    - 파일 변경이 없으면 이전 상태에 대한 링크만 저장
    - **스냅샷의 스트림**으로 취급



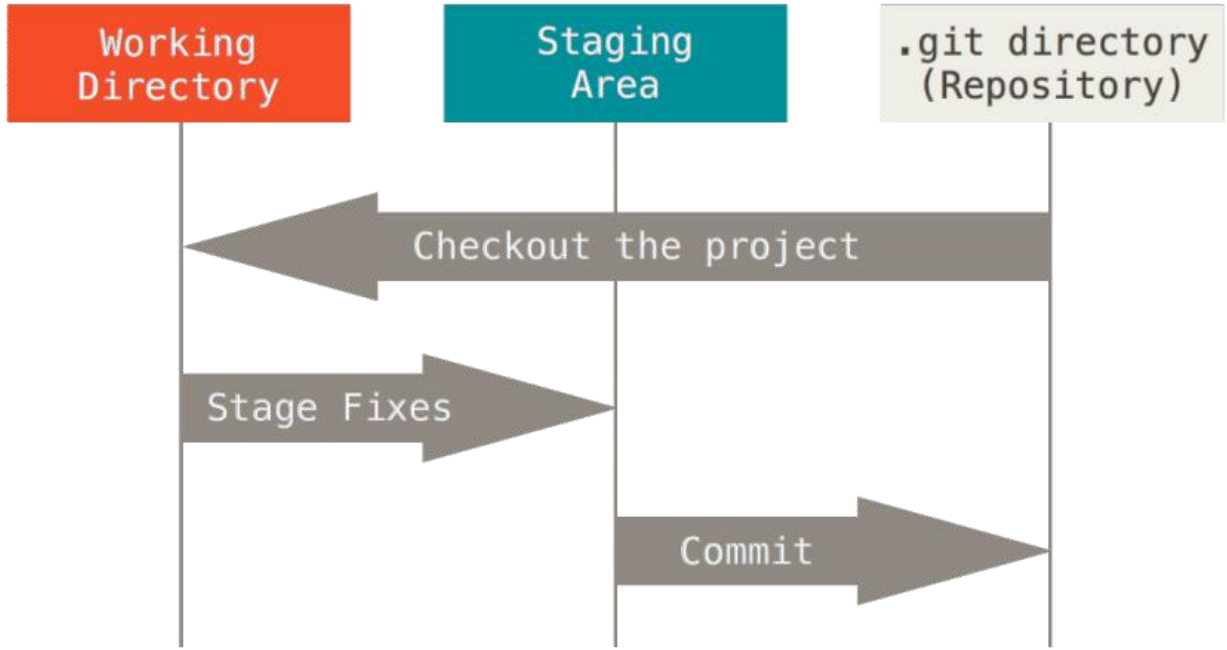
Git 기초 (2)

- **거의 모든 명령을 로컬에서 실행**
  - 거의 모든 명령이 로컬 파일과 데이터만 사용
    (오프라인, 온라인 상태에 거의 영향을 받지 않음)←다른 시스템과의 차별점
  - 프로젝트의 모든 히스토리가 로컬 디스크에 있기 때문에
    대부분의 명령들은 순식간에 실행됨
- 체크섬(checksum)은 Git의 데이터 관리의 핵심
  - Git에서 사용하는 가장 기본적인(Atomic) 데이터 단위이자 Git의 기본 철학
  - 체크섬: 40자 길이의 16진수 문자열
    (예: 24b9da6552252987aa493b52f8696cd6d3b00373)
  - 파일의 내용이나 디렉토리 구조에 대해 SHA-1 해시(hash)로 체크섬 계산
  - Git은 파일을 이름으로 저장하지 않고 해당 파일의 해시로 저장

체크섬
- 데이터의 정확성을 검사하기 위한 용도로 사용되는 합계로 오류 검출 방식의 하나이다.
- 데이터의 입력이나 전송이 제대로 되었는지를 확인하기 위해 입력 또는 전송 데이터의 맨 마지막에 앞서 보낸 모든 데이터를 다 합한 합계를 따로 보내는 것

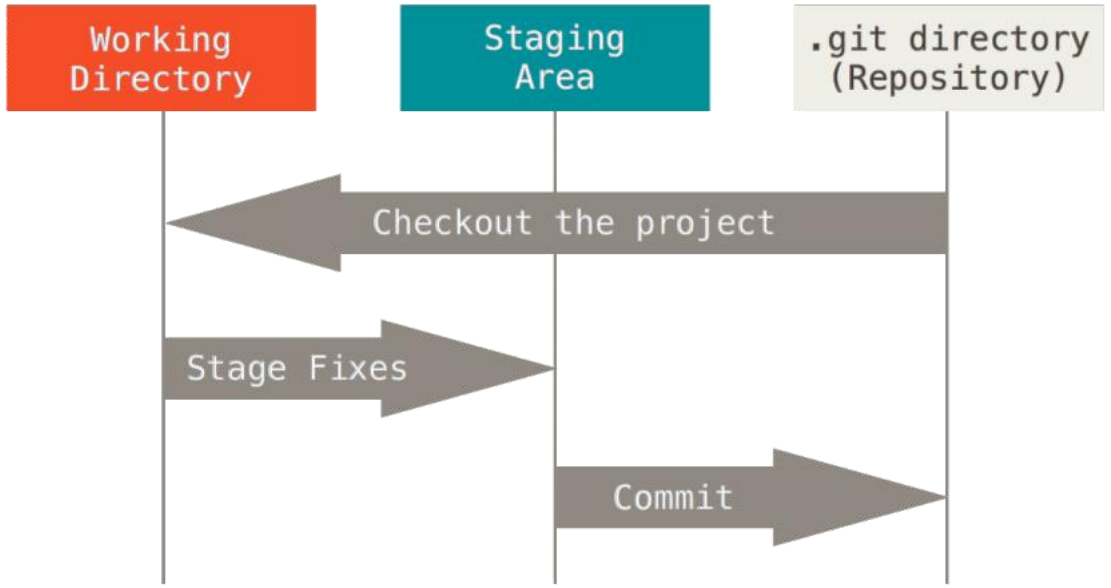
Git 파일의 3가지 상태 (매우 중요)

- Git은 파일을 **Modified**, **Staged**, **Committed**세 가지 상태로 관리
- **Modified(Working area):** 수정했으나 아직 로컬 데이터베이스에 커밋(commit)하지 않은 것
- **Staged(Staging Area):** 현재 수정한 파일을 곧 커밋 할 것이라고 표시한 상태를 의미
- **Committed(Repository):** 데이터가 로컬 데이터베이스에 안전하게 저장되었음



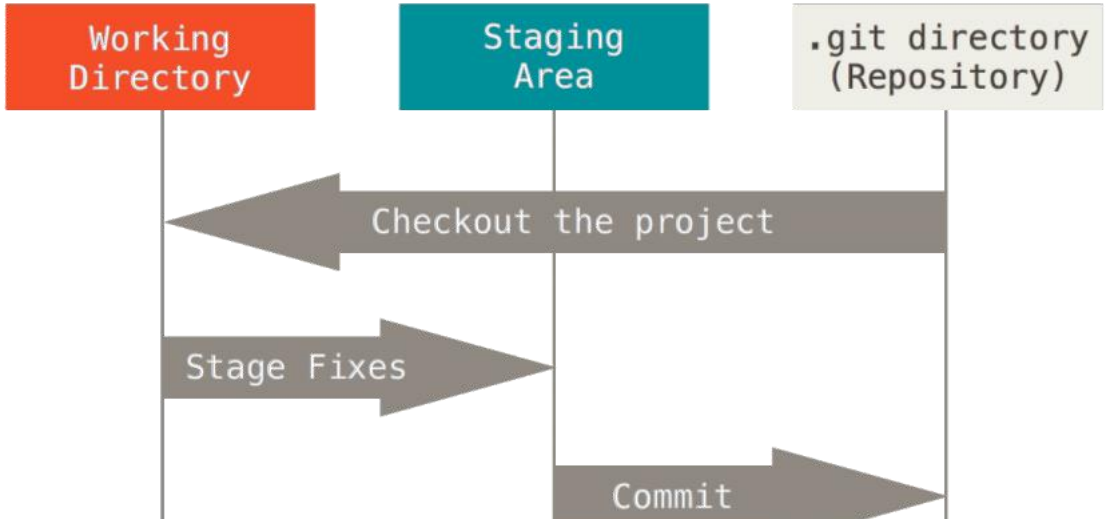
Git 디렉토리

- **Working Directory:**
  - 지금 작업하는 컴퓨터의 특정 디렉토리
  - 리모트 서버 등에서 특정 버전을 가져와(checkout) 만든 작업용 디렉토리
- **Staging Area:** (← *실제 directory가 아니고 특정 파일의 개념적 용도를 말함*)
  - Git directory에 저장된 단순 파일이며, **곧 커밋할 파일 정보**를 저장
- **Git Directory: (Git의 핵심)**
  - 보통 Working Directory의 sub directory (**.git**)로 생성됨
  - 프로젝트의 메타데이터와 객체 데이터베이스를 저장 하는 곳

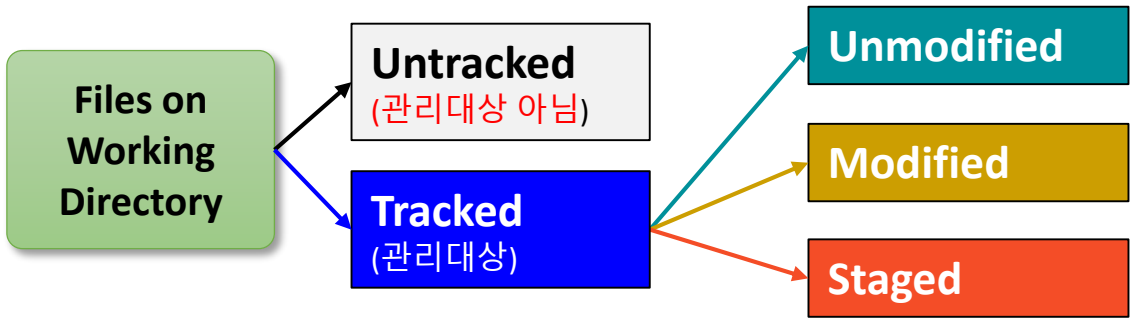


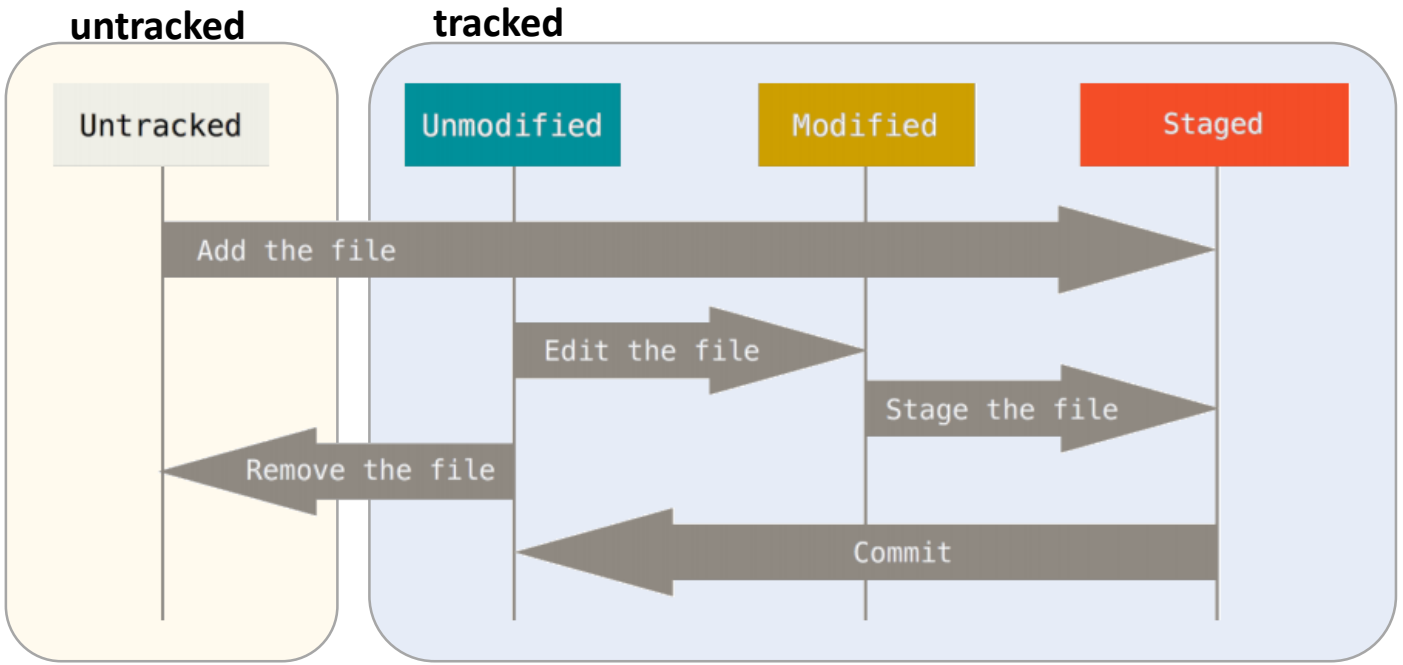
기본적인 Git 작업 순서

1. **Working Directory에서 파일을 수정(개발)**
   - 특정 디렉토리에서 Java, Javascript, C 등의 소스코드 편집, 수정하는 작업
2. **수정한 파일을 Staging Area에 stage해서 스냅샷을 생성**
   - Staging Area에 stage 된 수정파일의 상태는 **staged**.
   - 수정은 하였으나 stage 되지 않은 파일의 상태는 **modified**.
3. **커밋(commit)하면 Staging Area의 파일들이 Git Directory에 영구적 스냅샷으로 저장**
   - Git directory에 저장된 파일의 상태는 **committed**.



워킹 디렉토리 파일의 라이프 사이클





Git 설치

- **CLI vs. GUI**
  - Git의 모든 기능을 지원하는 것은 CLI 뿐
  - 대부분의 GUI는 Git의 기능을 전부 구현하지 않고 비교적 단순
- **Linux (Ubuntu 설치)**
  - **$ sudo apt-get install git**
  - 설치 후, 가능하면 최신 버전으로 업그레이드
- Windows 설치
  - http://git-scm.com/download/win (CLI 설치)
  - http://windows.github.com (CLI, GUI 모두 설치)

Git 설치 후 최초 설정

- **Git의 사용환경을 설정**
  - **한 번만 설정하면 되며, 업그레이드 시에도 유지됨.**
  - 뒤에 다시 변경할 수 있음 (git config)
- **사용자 정보 등록**
  - GUI 도구들은 처음 실행할 때 이 설정을 묻는다. (묻지 않을 경우에 설정)
  - **$ git config --global user.name "John Doe"**
  - **$ git config --global user.email "user@email.com"**
- **설정 확인**
  - **$ git config –-list**
- **도움말 보기**
  - **$ git help <명령> / git <명령> --help / man git <명령>**

Git 관련 자료

- git documentation
  - https://git-scm.com/book/ko/v2

- git reference
  - https://git-scm.com/docs

Git 기본사용법

Introduction

https://www.youtube.com/watch?v=PiQAwOxl52E

Git 저장소 만들기

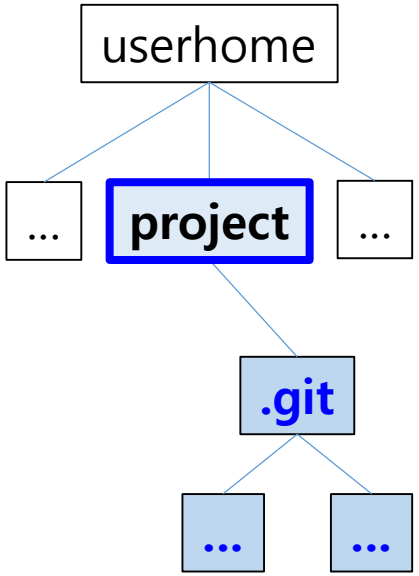
- **Git 저장소를 만드는 2가지 방법**

  1. **기존 디렉토리를** Git 저장소로 만들기 (현재 작업 중인 디렉토리 대상)

  2. **기존 저장소를 복사**(clone)하기 (서버 등 다른 컴퓨터 프로젝트 복사)

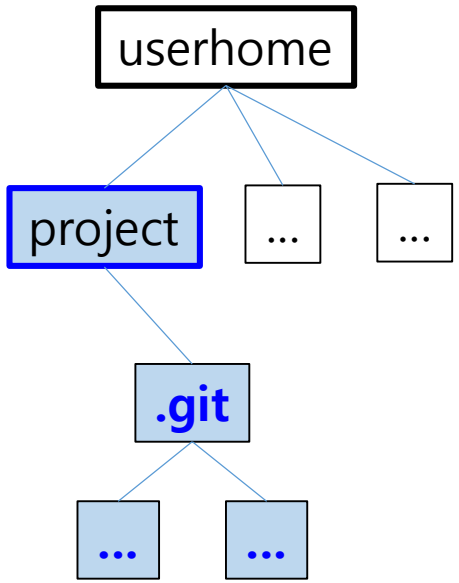
Git 저장소 만들기

- **기존 디렉토리를 Git 저장소로 만들기**
  - 프로젝트 디렉토리로 이동하여 다음 명령 실행
  - **$ git init**
    - 아직 프로젝트 디렉토리의 어떤 파일도 버전 관리가 시작되지 않았음
  - 버전 관리할 파일을 git 명령으로 추가하고 커밋해야 관리가 시작됨
    - **$ git add *.c** // 관리할 파일 추가
    - **$ git add readme.txt** // 관리할 파일 추가
    - **$ git commit -m 'first version'** // 저장



Git 저장소 만들기

- **기존 저장소를 복사하기**
  - 서버 등 다른 컴퓨터의 프로젝트를 로컬 컴퓨터로 복사
  - 기존 저장소를 복사해 올 디렉토리로 이동 후 다음 명령 실행
  - **$ git clone [url]**
    - 예) userhome 디렉토리에서 다음 명령 실행
    - **$ git clone https://github.com/gituser/project**
      - Userhome 디렉토리에 project 이름의 Git 저장소가 복사됨
  - 다른 디렉토리 이름으로도 저장할 수 있음
    - **$ git clone [url] newDirectory**



수정하고 저장소에 저장하기 (1)

- 워킹 디렉토리의 파일 상태 확인하기
  - $ git status

```
$ git status
On branch master
nothing to commit, working directory clean
```

    - 수정한 내용이 없으면 위와 같이 표시됨
    - 현재 브랜치가 master임을 표시. (* 브랜치에 관한 이야기는 이후에 다시 정리함)

수정하고 저장소에 저장하기 (2)

- 새 파일(README)을 생성하고 상태 확인하기

```
$ echo 'My Project' > README
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add"
```

  - Untracked 파일 목록에 README 표시됨
  - $ git add README
    - 위 명령으로 Tracked 상태가 되지 않는 한 절대로 저장되지 않음
  - 보통 소스코드를 컴파일 해서 생성되는 실행파일 등은 자동으로 제외됨

수정하고 저장소에 저장하기 (3)

- **파일을 새로 추적하기**
  - git add 명령으로 Git에서 파일을 새로 추적할 수 있음

```
$ git add README
```

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

    new file:   README
```

실습 – 로컬 저장소

- **Git 기본 명령어**

명령어	설명
git init	저장소의 생성(.git 폴더가 만들어짐)
git add <filename> **git add --all**	저장소에 파일 추가(staging area로 올라감)
git commit –m “message”	저장소에 변경 내용 반영
git config	git 설정(사용자 정보 설정, 설정 확인 등)
git status	저장소 상태 확인
git diff	이전 버전과 비교해 다른 점 나타냄
git log **git log --stat**	로그 상태 보기 각 파일 별 로그 상태 보기

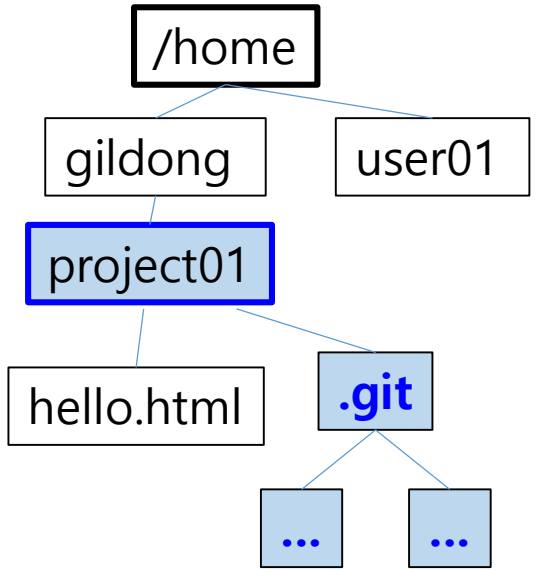
실습1 – 로컬 저장소

- 다음의 지시 사항에 맞추어 실행 하시오.

명령을 실행하고 난 후 반드시 git의 상태를 확인하세요!

1. 각자의 id로 사용자 계정과 비밀번호를 만든다(ex. gildong / 1111).
2. 만들어진 각자의 id로 우분투에 로그인한다.
3. 자신의 홈디렉토리에 working directory인 project01 디렉토리를 생성시킨다.
4. git에서 사용할 사용자 정보를 설정한다(“홍길동” / “gildong@naver.com”).
5. project01 디렉토리에 local repository를 생성한다.
6. hello.html 파일을 working directory에 작성한다.

```
<html>
        <head>
                <title>연습입니다.</title>
        </head>
        <body>
                Hello, World!!
        </body>
</html
```



7. 위의 파일을 추적가능하도록 staging area에 올린다.
8. 위의 파일을 local repository에 저장한다(message는 “first commit”).

실습1 – 로컬 저장소

9. hello.html파일을 다음과 같이 변경(modify)하고 위의 7, 8번을 반복한다
   (message는 “second commit”)

```
<html>
        <head>
                <title>연습입니다.</title>
        </head>
        <body>
                <h2>Hello, World!!</h2>
                <hr>
                참 반갑습니다.
        </body>
</html>
~
```

10. hello.html파일을 다시 한번 다음과 같이 변경(modify)하고 위의 7, 8번을 반복한다
    (message는 “third commit”)

```
<html>
        <head>
                <title>연습입니다.</title>
        </head>
        <body>
                <h2>Hello, World!!</h2>
                <hr>
                참 반갑습니다.<br>
                오늘도 즐거운 하루 되기를 바랍니다.
        </body>
</html>
```

11. git의 로그 상태를 확인한다.