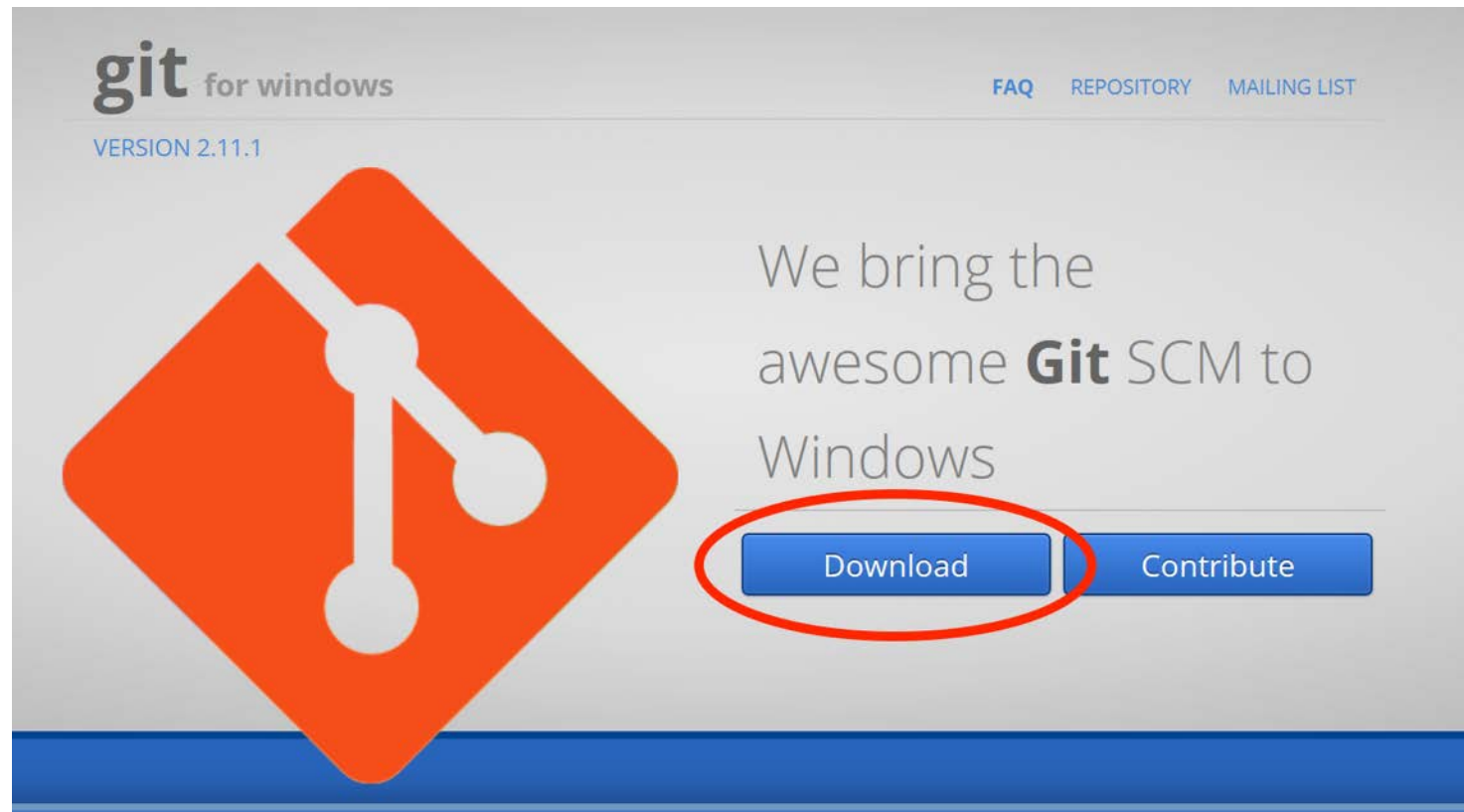


Git 설치 및 사용법 안내

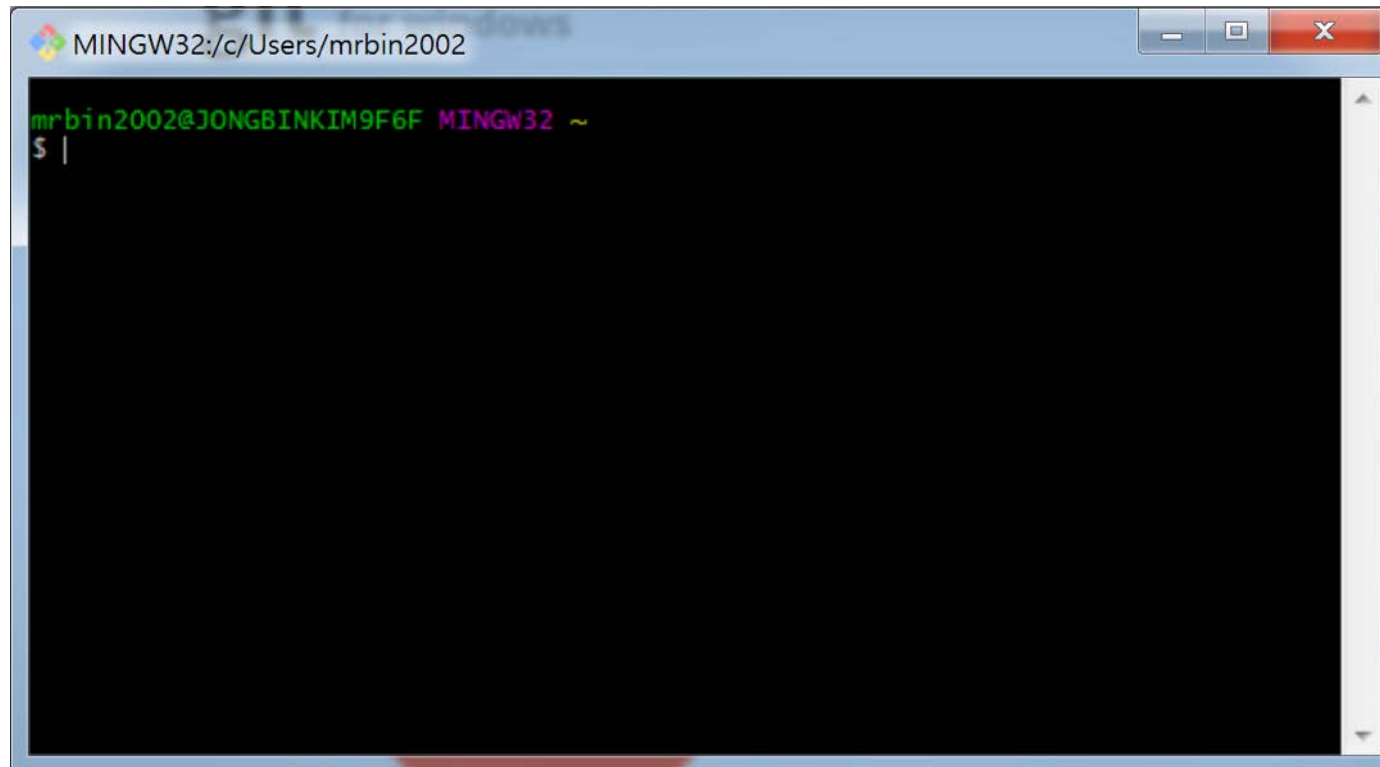
Git 설치 (Windows)

1. <https://git-for-windows.github.io/> 에 접속하여 다운로드 및 설치



Git 설치 (Windows)

2. Git Bash 실행
3. 작업할 디렉토리로 이동 (ex: \$ cd project)



Git 설치 (Linux)

Ubuntu

```
$ sudo apt-get install git
```

Fedora

```
$ sudo yum install git
```

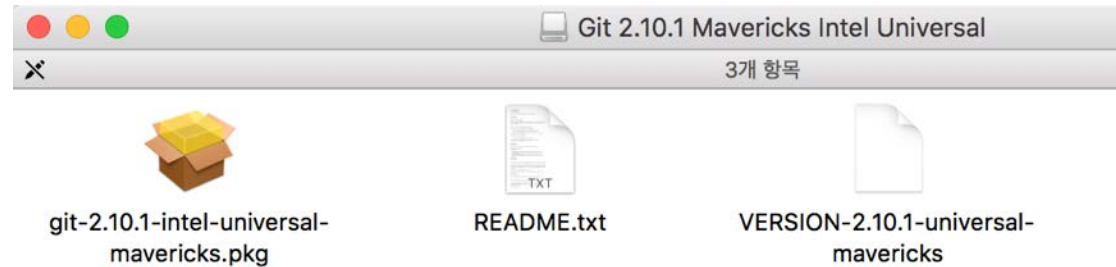


A terminal window titled 'mrbin2002 — mrbin2002@ubuntu: ~ — ssh mrbin2002@10.211.55.7 — 74x21'. The terminal shows the command 'sudo apt-get install git' being executed. The output indicates that git is already the newest version and no packages need to be upgraded. The prompt returns to 'mrbin2002@ubuntu:~\$'.

```
mrbin2002@ubuntu:~$ sudo apt-get install git
[sudo] password for mrbin2002:
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version.
0 upgraded, 0 newly installed, 0 to remove and 51 not upgraded.
mrbin2002@ubuntu:~$
```

Git 설치 (Mac OSX)

1. <https://git-scm.com/download/mac> 으로 접속 후 다운로드(git-x.x.x-xxx.dmg)
2. 다운받은 파일 실행 후 git-x.x.x-xxx.pkg 파일 실행 후 설치



Git 기본 사용법 안내

1. 설치 완료 후, Git 사용자 설정하기

```
$ git config --global user.name "2007002245"  
$ git config --global user.email "2007002245@hanyang.ac.kr"
```

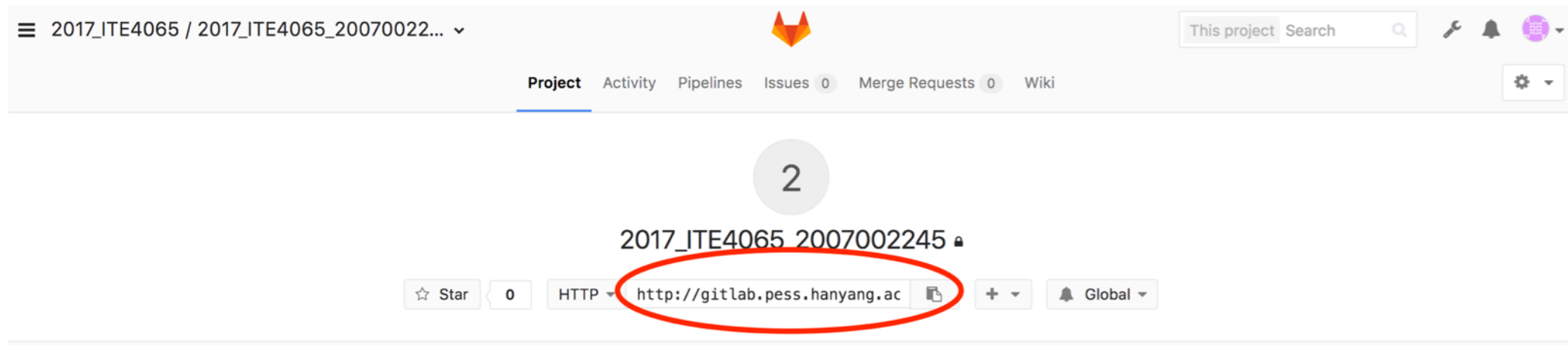
(user.name은 학번으로,
user.email은 GitLab에 등록해놓은 email로 (기본값: 학번 + @hanyang.ac.kr))

Git 기본 사용법 안내

2. 생성되어 있는 학생의 Git repository clone받기

```
$ git clone http://gitlab.pess.hanyang.ac.kr/YEAR_ITE0000/YEAR_ITE0000_20XXXXXXXXX.git
```

Git clone 주소는 GitLab webpage의 해당 프로젝트 메인화면으로 이동하여 확인 가능



Git 기본 사용법 안내

3. git clone 시 요구하는 Username은 학번으로, Password는 GitLab webpage에서 설정한 password로 입력

```
TA — mrbin2002@ubuntu: ~ — -bash — 76x21
[Jongbin:TA mrbin2002$ git clone http://gitlab.pess.hanyang.ac.kr/2017_ITE4065/2017_ITE4065_2007002245.git
Cloning into '2017_ITE4065_2007002245'...
Username for 'http://gitlab.pess.hanyang.ac.kr': 2007002245
Password for 'http://2007002245@gitlab.pess.hanyang.ac.kr':
warning: You appear to have cloned an empty repository.
[Jongbin:TA mrbin2002$ ls
2017_ITE4065_2007002245
Jongbin:TA mrbin2002$
```


Git 기본 사용법 안내

4. Clone받은 폴더로 이동 (처음에는 텅 빈 디렉토리)

```
$ cd YEAR_ITE0000_20XXXXXXXXX
```

5. 작업 파일 생성

```
$ vi test.c
```

6. 파일 작성



```
2017_ITE4065_2007002245 — mrbin2002@ubuntu: ~ — vi test.c — 76x21
1 hello world
~
~
~
~
```

Git 기본 사용법 안내

7. 현재 git 관리 상태를 확인하면 test.c가 관리되지 않는 상태로 표시된다.

```
$ git status
```

```
[Jongbin:2017_ITE4065_2007002245 mrbin2002$ git status]
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    test.c

nothing added to commit but untracked files present (use "git add" to track)
Jongbin:2017_ITE4065_2007002245 mrbin2002$
```

Git 기본 사용법 안내

8. 현재 디렉토리에 있는 모든 추가/수정된 파일들을 Stage 영역으로 이동(test.c가 git에 의해 관리됨)

```
$ git add .
```

9. Git 관리 상태를 다시 확인

```
$ git status
```

```
[Jongbin:2017_ITE4065_2007002245 mrbin2002$ git add .  
[Jongbin:2017_ITE4065_2007002245 mrbin2002$ git status  
On branch master  
  
Initial commit  
  
Changes to be committed:  
  (use "git rm --cached <file>..." to unstage)  
  
    new file:   test.c
```

Git 기본 사용법 안내

10. 추가/수정된 파일을 커밋(Local repository에 저장)

```
$ git commit -m "first commit"
```

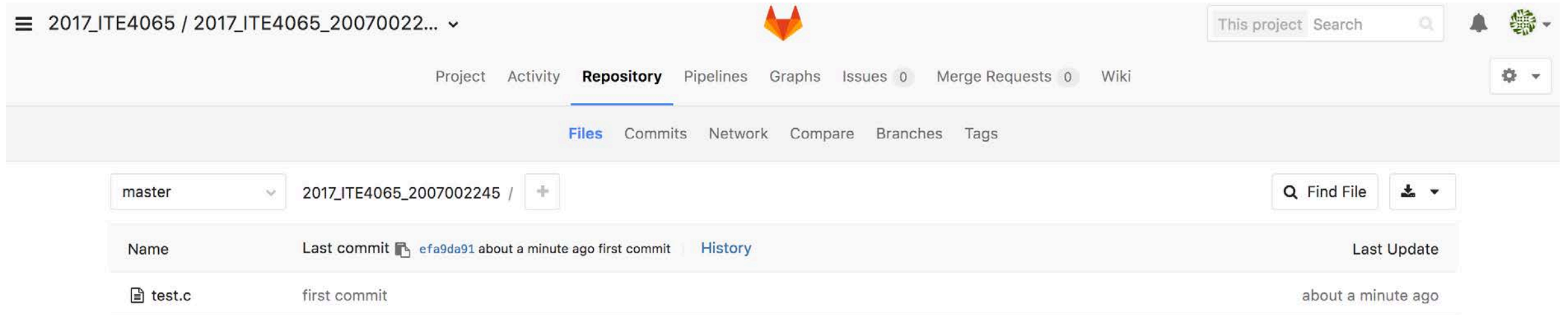
11. 커밋된 내용을 Remote repository로 전송

```
$ git push origin master
```

```
[Jongbin:2017_ITE4065_2007002245 mrbin2002$ git commit -m "first commit"
[master (root-commit) efa9da9] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 test.c
[Jongbin:2017_ITE4065_2007002245 mrbin2002$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 224 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To http://gitlab.pess.hanyang.ac.kr/2017_ITE4065/2017_ITE4065_2007002245.git
 * [new branch]      master -> master
```

Git 기본 사용법 안내

12. git push를 통해 Remote로 전송된 파일은 GitLab webpage에서 확인 가능하다



The screenshot displays the GitLab web interface for a repository named '2017_ITE4065 / 2017_ITE4065_20070022...'. The 'Repository' tab is selected, showing the 'Files' view. The current branch is 'master'. A search bar and a 'Find File' button are visible. Below the navigation bar, a table lists the files in the repository:

Name	Last commit	Last Update
test.c	first commit	about a minute ago

Version Control System

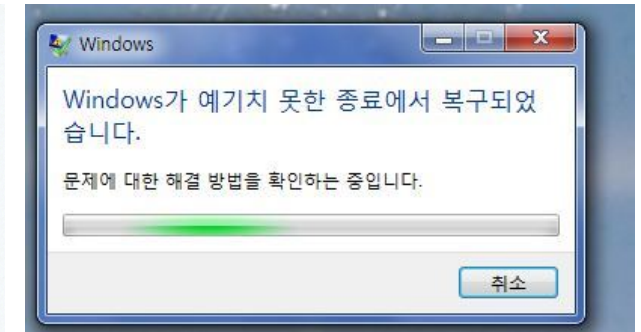
git user guide

Summary

- Version Control System
- git 소개 및 특징
- git 에서의 용어 및 명령어
- git 설치 및 설정하기

Version Control System?

- Version?
 - 프로그램의 변경이력을 의미하며 일반적으로 major.minor 로 표기



- Version Control?

151110_코드_(1).txt	2015-11-11 오후...	텍스트 문서
151110_코드_(2) - 수정.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(3) - 기존코드백업.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(4) - 수정2.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(5) - 수정33.txt	2015-11-11 오후...	텍스트 문서
151111_코드_(6) - 완성.txt	2015-11-11 오후...	텍스트 문서
151112_코드_(7) - 완성 - 수정.txt	2015-11-11 오후...	텍스트 문서

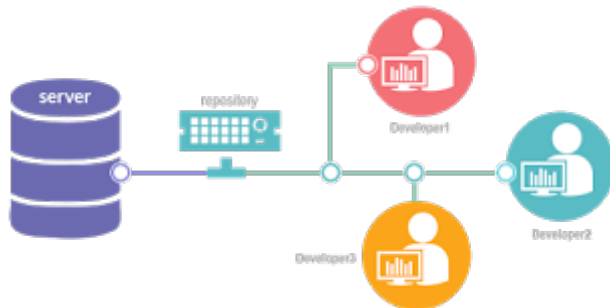
Version Control System?

- Version?
- Version Control?
 - 작업결과물에 대한 이력관리를 통한 복구



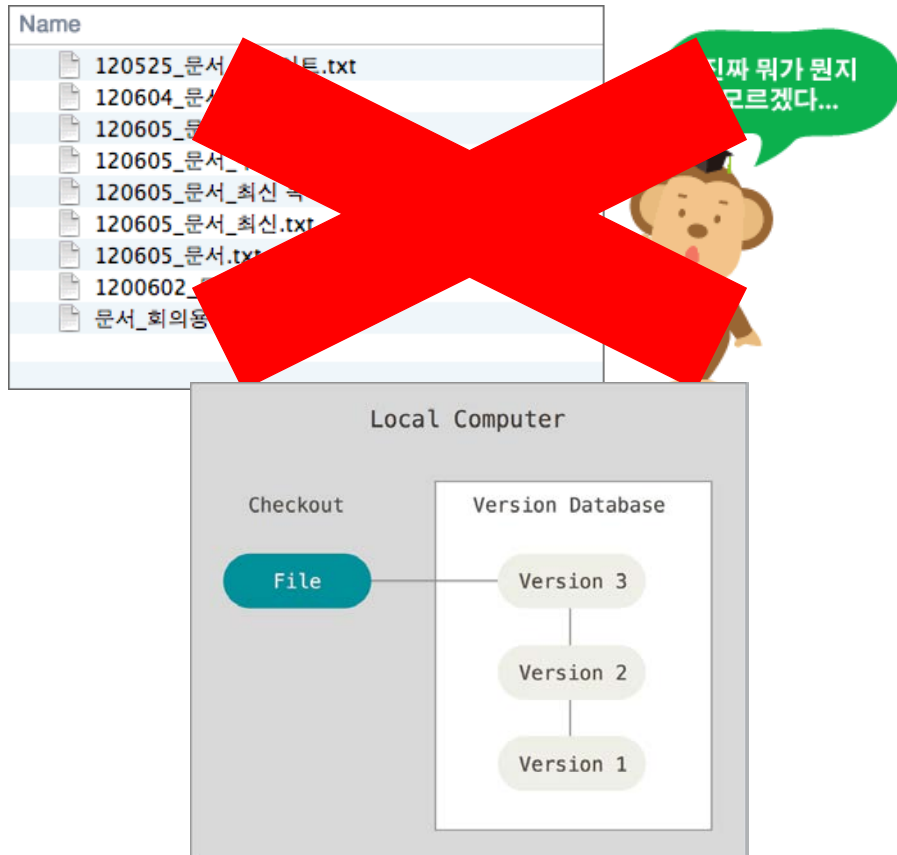
151110_코드_(1).txt	2015-11-11 오후...	텍스트 문서
151110_코드_(2) - 수정.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(3) - 기존코드백업.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(4) - 수정2.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(5) - 수정3.txt	2015-11-11 오후...	텍스트 문서
151111_코드_(6) - 완성.txt	2015-11-11 오후...	텍스트 문서
151112_코드_(7) - 완성 - 수정.txt	2015-11-11 오후...	텍스트 문서

- Something else..?

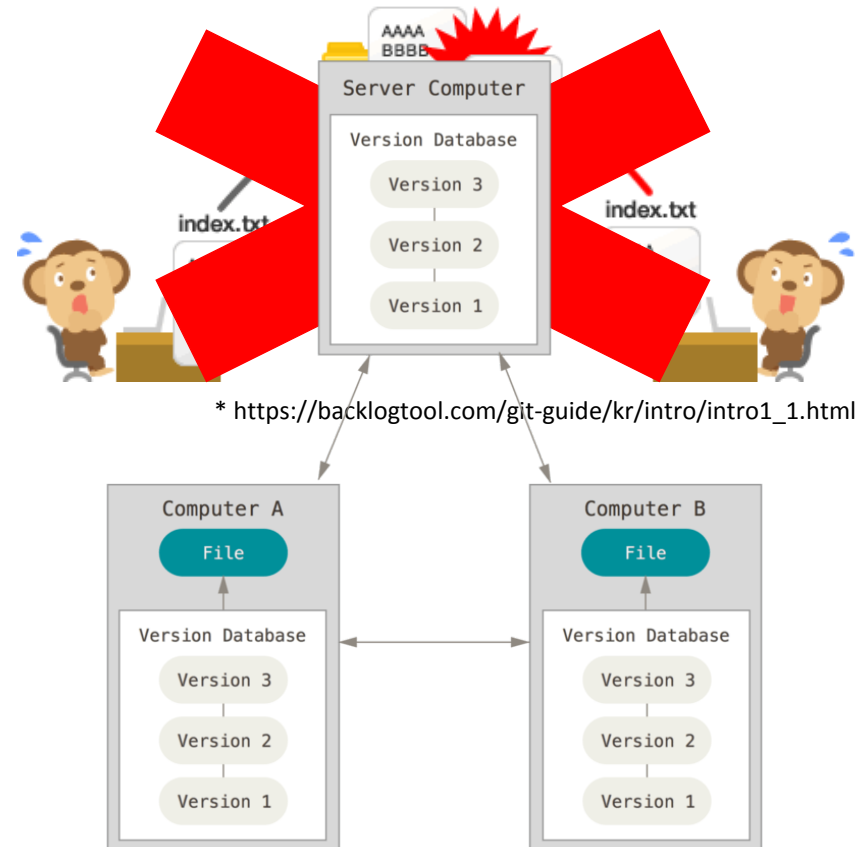


Version Control System의 좋은점

- 작업결과의 손쉬운 이력관리



- 협업중 발생하는 이력충돌 해결

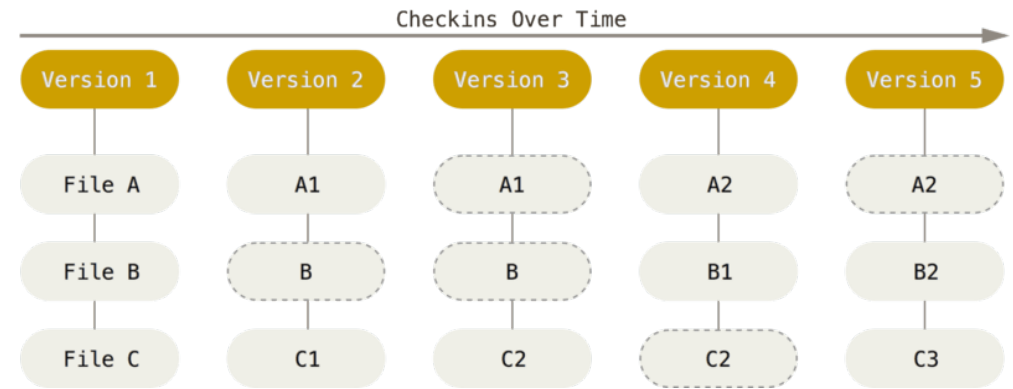
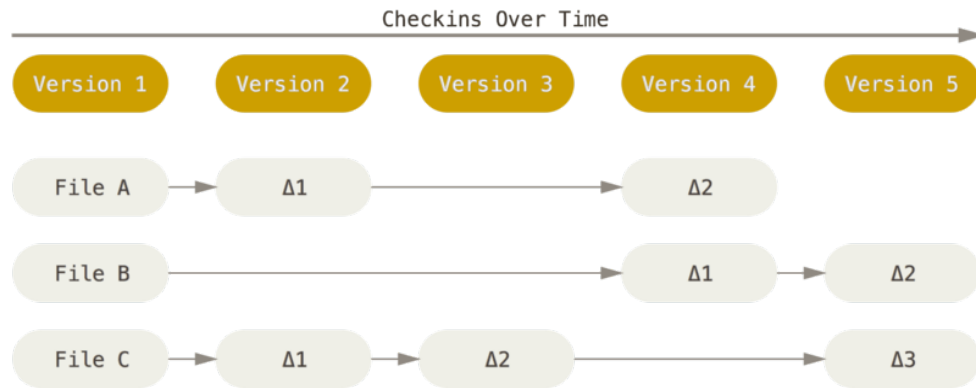


git란?

- Linux Kernel은 대표적인 대규모 오픈소스 프로젝트임
- 초창기 Linux Kernel은 BitKeeper라는 DVCS(Distributed VCS)을 사용했음
- BitKeeper를 사용하지 못하게 되자 git이라는 자체 툴을 만들어 사용하게 됨
- git의 설계목표
 - 빠른 속도
 - 단순한 구조
 - 동시다발적인 branch에 대한 충분한 지원
 - 완벽한 분산
 - 대형 프로젝트 관리도 가능함

git 특징 - 1

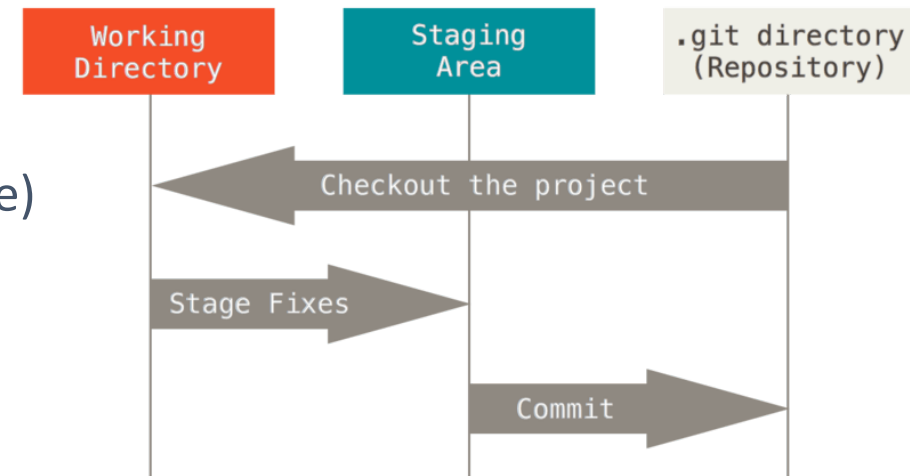
- 차이값이 아닌 Snapshot 저장



- 거의 대부분의 명령은 local에서
 - server 통신없이 데이터 변경이력 조회 및 commit 가능
- SHA-1 해싱을 통한 무결성 관리
 - 각각의 버전은 이름이 아닌 SHA-1 체크섬으로 식별
 - git을 통해서 해당 파일의 데이터에 접근

git 특징 - 2

- 차이값이 아닌 Snapshot 저장
- 거의 대부분의 명령은 local에서
- SHA-1 해싱을 통한 무결성 관리
- **작업 파일의 3가지 상태**
 1. **committed**
파일이 local (git repository)에 저장되어 있음 (up-to-date)
 2. **modified**
up-to-date 상태의 파일을 사용자가 수정한 경우
* **commit** 명령을 실행하더라도 **commit** 되지 않음
 3. **staged**
modified 상태의 파일을 commit 하기 위한 준비단계
* **commit** 명령은 **staged** 상태의 파일만을 **commit** 함



git 용어 - 1

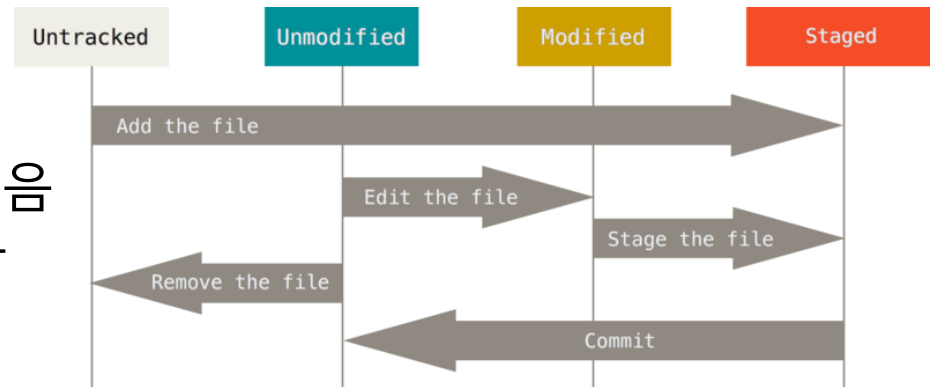
- working directory 내의 파일 상태

- untracked

- 아직 git 에 의하여 파일의 변경이 관리되고 있지 않음
 - git add 명령을 통해 version control에 포함시켜야 함

- tracked

- git 에 의하여 파일의 변경이 관리되고 있는 상태
 - git rm 명령을 통해 version control에서 제외 및 파일 삭제
(git rm --cached는 version control만 제외, 파일은 유지)



- repository

- local

- remote

- 다른 작업자와의 협업을 위하여 외부의 서버에 변경내용들을 저장
 - git clone/pull/push를 통하여 local repository로 데이터를 동기화/불러오기/저장하기

git 용어 - 2

- repository의 버전관리

- checkout

- workspace의 파일 데이터를 local repository에 저장된 특정 버전으로 변경

- pull

- workspace의 파일 데이터를 remote repository(서버)에 저장된 특정 버전으로 변경

- push

- workspace의 파일 데이터를 remote repository(서버)로 저장

- fetch

- workspace의 파일 데이터를 remote repository(서버)에서 불러옴

- commit

- 수정된 파일을 local repository에 저장

- tag

- 특정 시점의 파일에 대하여 부연설명을 태깅 (주로 버전정보를 태깅)

git 용어 - 3

- tree

: repository 내에서 commit된 snapshot은 tree 형태를 이룸

- master

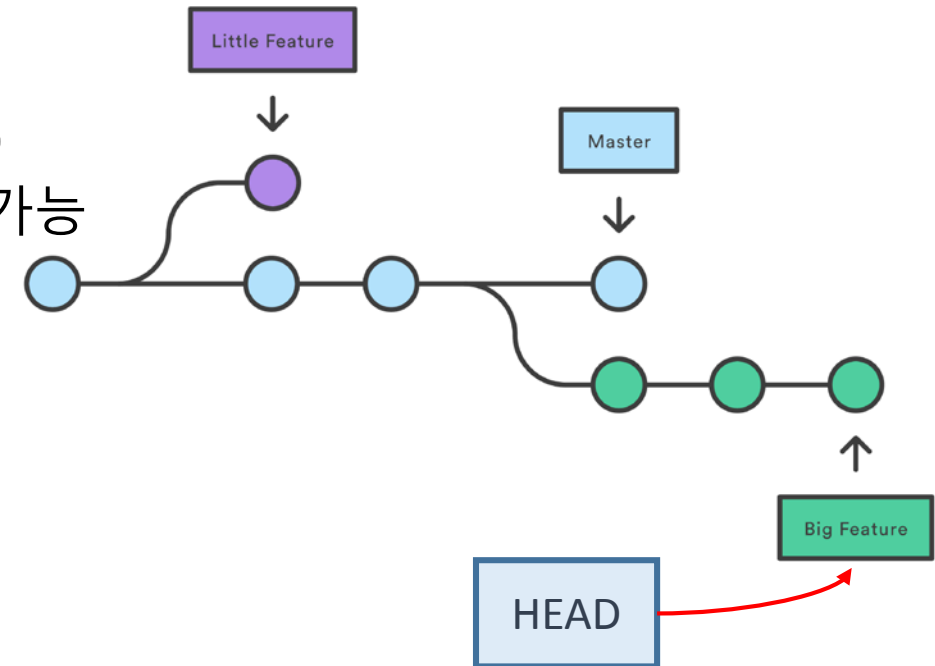
- snapshot tree에서의 main line

- Branch

- main line(master)에서 분리되어 나온 가지(branch)
- main line을 훼손하지 않으면서 독립적인 작업이 가능

- HEAD

- 현재 작업중인 branch



git 명령어

1. local repository 및 commit 관리
2. branch 관리
3. remote repository 관리

git 명령어

- repository 만들기
 1. 새 repository 만들기 (local)
`$ git init`
 2. 서버로 부터 repository 만들기 (remote)
`$ git clone [url]`
`$ git clone [url] [new-project-name]`

git 명령어

- 파일의 commit history 조회하기
 - local repository의 commit 내용들을 시간순서로 출력
`$ git log`
 - 최근 2개의 commit(-2 옵션)의 diff 결과 출력 (-p 옵션)
`$ git log -p -2`
 - commit들의 통계정보 출력(수정된 파일수, 추가/삭제된 라인수)
`$ git log --stat`
 - 특정시점 기준으로 출력
`$ git log --since=2.weeks`
`$ git log --until=2016-09-01`
 - 변경된 commit들을 텍스트로 검색하여 출력
`$ git log --S[function-name]`
- * [참고] <https://git-scm.com/book/en/v2/Git-Basics-Viewing-the-Commit-History>

git 명령어

- 파일의 수정 및 repository 관리하기 - 1

- 수정된 파일을 staging
 - tracked → staged
 - untracked → tracked + staged
 - directory인 경우 하위의 모든 파일에 적용

```
$ git add [file-name]
```

- 파일의 상태 확인

```
$ git status
```

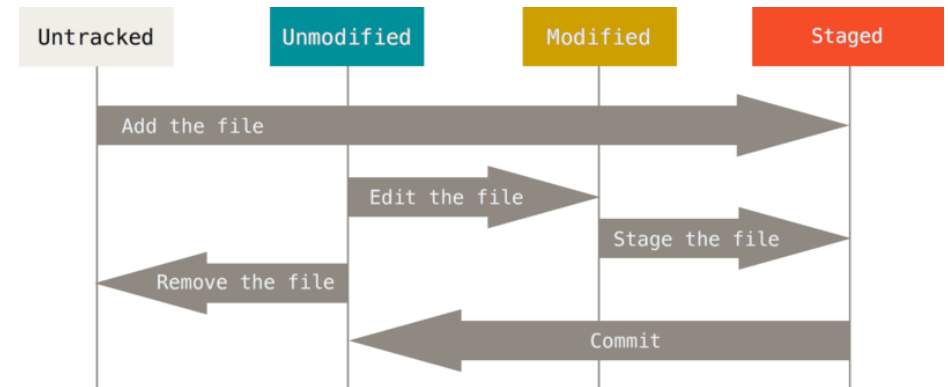
- 파일의 변경내용 비교

- unstaged vs. staged

```
$ git diff
```

- staged vs. committed

```
$ git diff --staged
```



git 명령어

- 파일의 수정 및 repository 관리하기 - 2

- 변경 사항 저장하기

- \$ git commit

- \$ git commit -m [comment]

- \$ git commit -a → staging 생략하고 바로 commit

- 파일 상태를 untracked로 변경 후 삭제

- \$ git rm [file-name]

- 파일 이름 변경하기

- \$ git mv [old-file-name] [new-file-name]

[주의!!] linux 명령어 rm / mv 과 다름.

git 명령어

- 수정사항 되돌리기

- commit 수정하기

마지막 commit에 빠진 파일이 있거나 하는 경우

```
$ git commit --amend
```

- staged → unstaged

git add * 등 실수로 staged 상태로 변경한 경우

```
$ git reset HEAD <file-name>...
```

- modified → unmodified

local repository에 commit된 상태로 덮어 씌움

(작업중의 내용이 없어지고 이 경우는 복구가 안되므로 주의할 것)

```
$ git checkout -- <file-name>
```

* [참고] <https://git-scm.com/book/en/v2/Git-Basics-Undoing-Things>

git 명령어: branch

- git branch

- branch 목록조회

- \$ git branch

- \$ git branch -v

- \$ git branch --no-merged

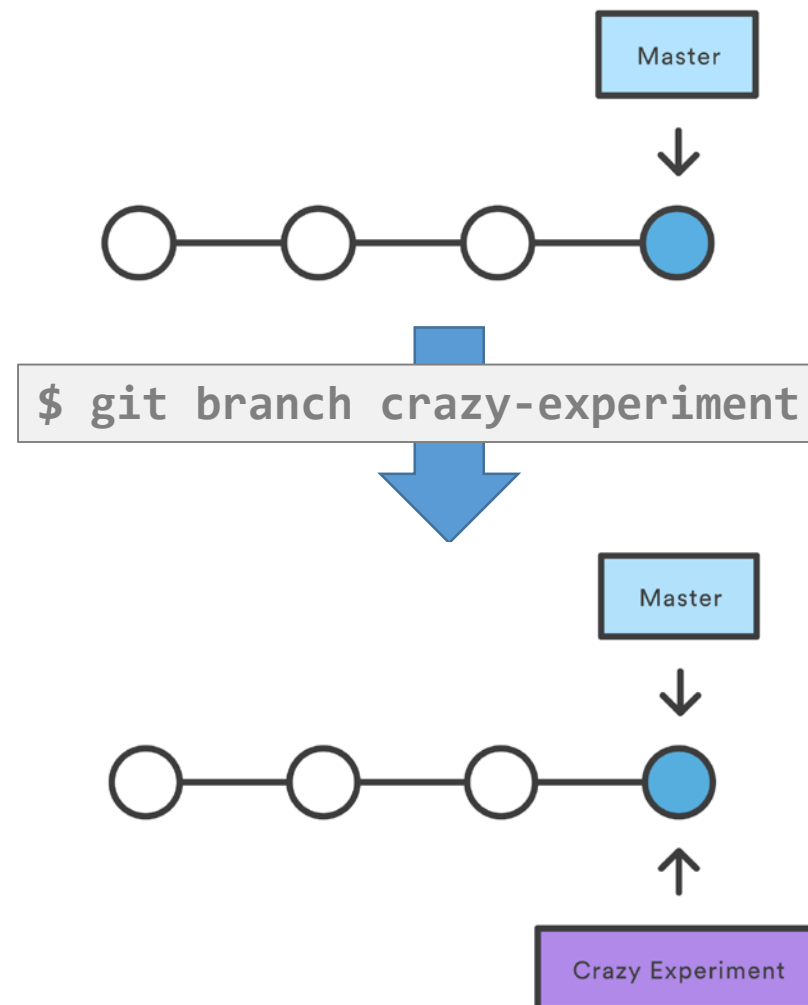
- 새로운 branch 만들기

- \$ git branch [branch-name]

- branch 삭제하기

- \$ git branch -d [branch-name]

- \$ git branch -D [branch-name]



git 명령어: branch

- git checkout

- checkout 명령을 통해 branch를 이동함

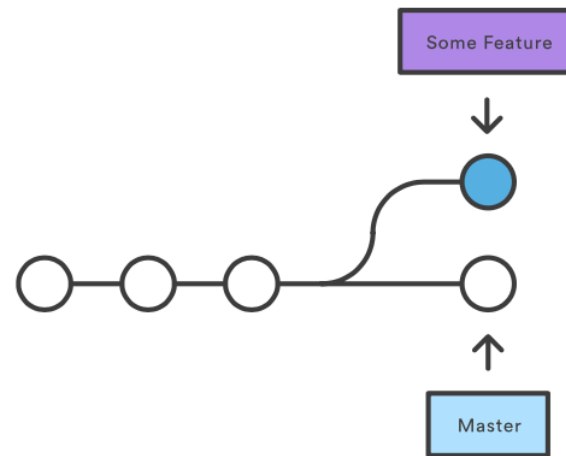
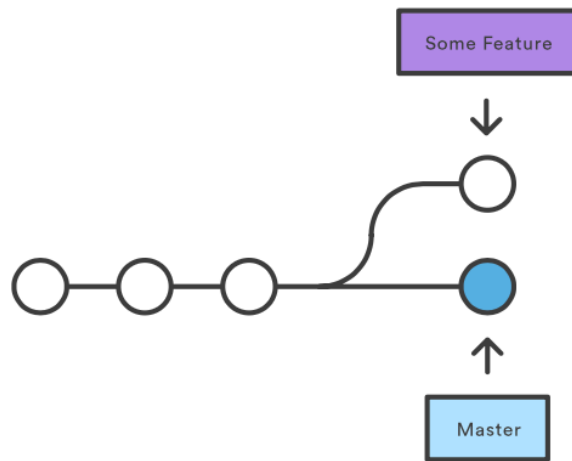
```
$ git checkout [existing-branch]
```

```
$ git checkout -b [new-branch]
```

```
$ git checkout -b [new-branch] [existing-branch]
```

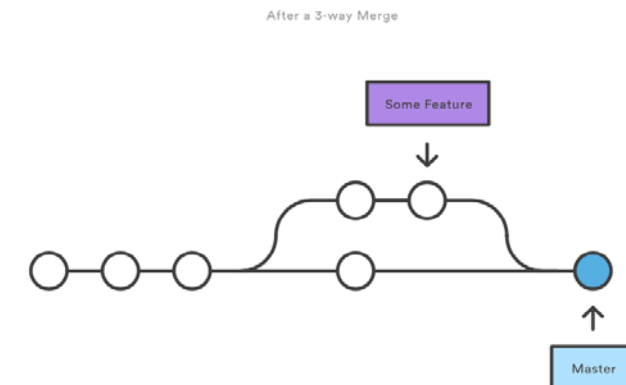
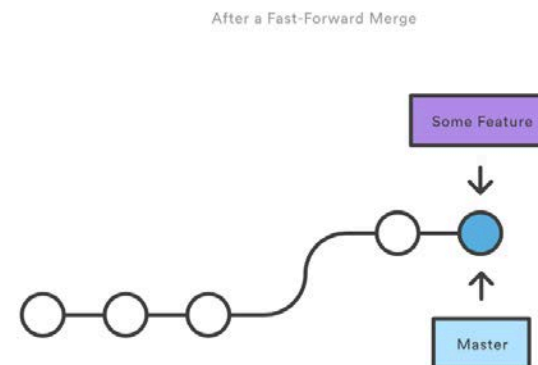
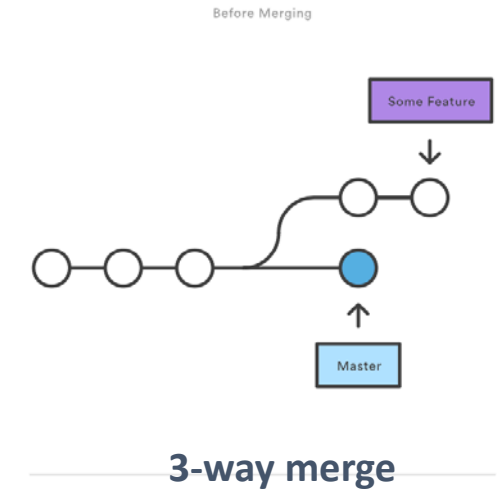
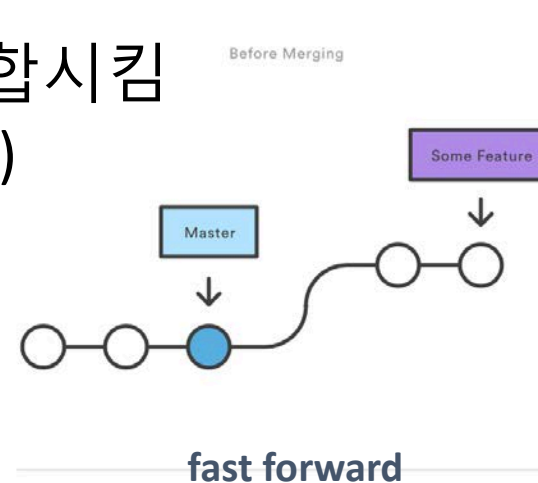
Checking Out Master

Checking Out Some Feature



git 명령어: branch

- git merge
 - 다른 branch를 현재의 branch로 병합시킴
(ex: hot-fix/testing branch → master)
 - 2가지 mode가 존재
 - fast forward
 - 3-way merge
- `$ git merge [branch-name]`



git 명령어: remote

- remote repository 관련 명령
 - remote repository 조회하기

```
$ git remote
$ git remote -v
$ git remote show [file-name]
```
 - remote repository 추가하기

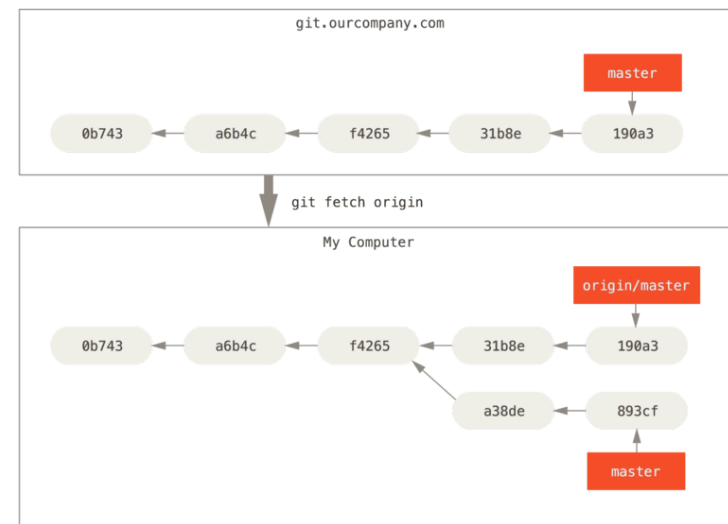
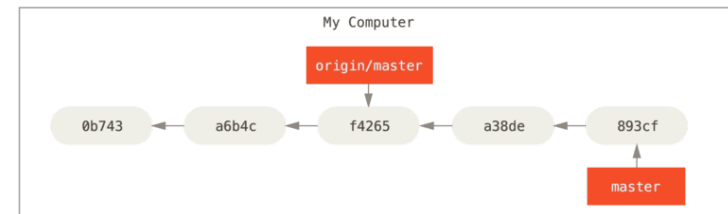
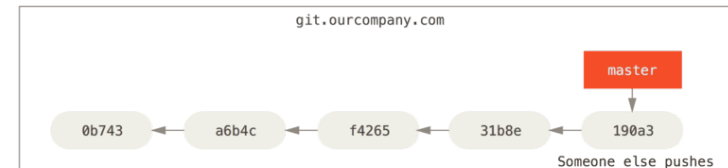
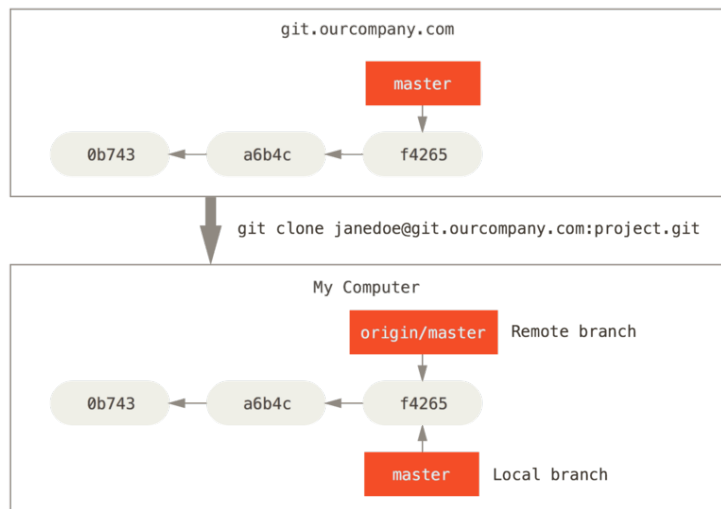
```
$ git add [remote-name] [URL]
```
 - remote repository 삭제하기

```
$ git rm [remote-name]
```
 - remote repository 이름 변경하기

```
$ git rename [old-name] [new-name]
```

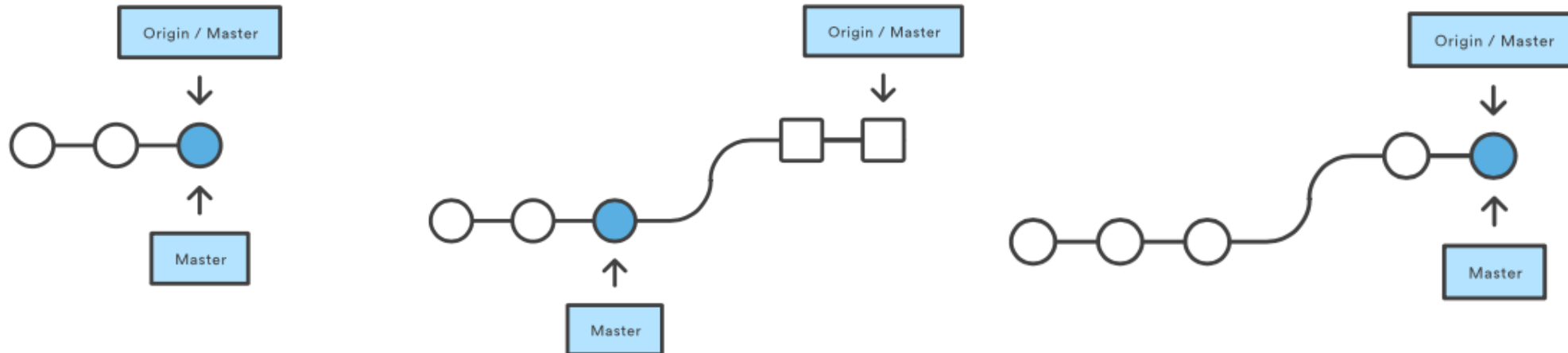
git 명령어: remote

- git fetch
 - 모든 branch fetch
`$ git fetch [remote-name]`
 - 특정 branch fetch
`$ git fetch [remote-name] [branch]`



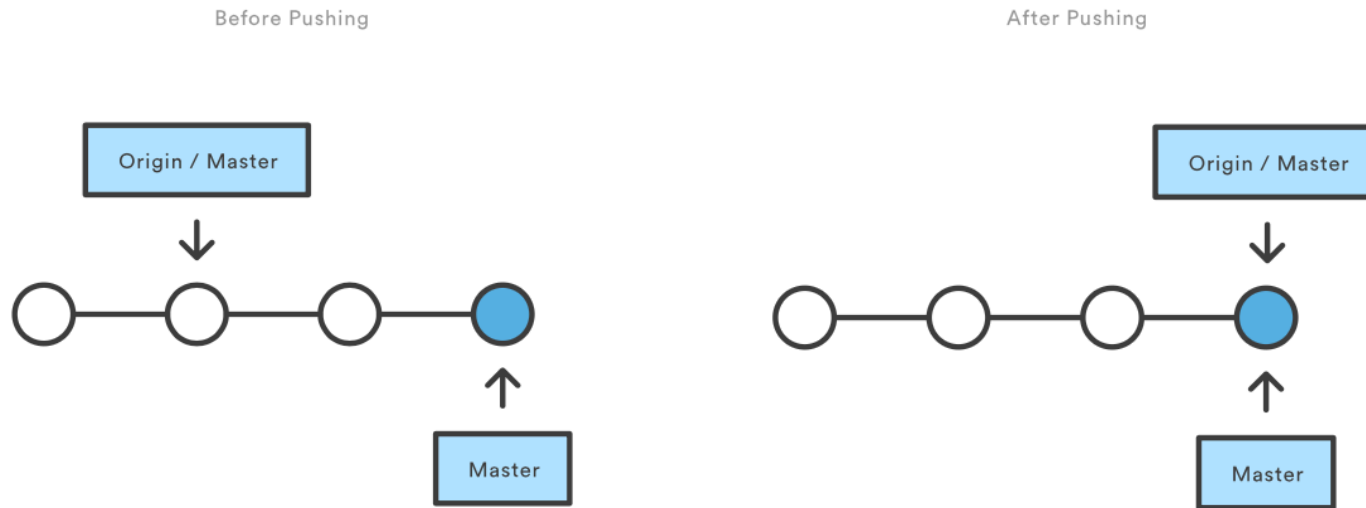
git 명령어: remote

- git pull
 - fetch 후 merge 수행
- ```
$ git pull [remote-name]
```



# git 명령어: remote

- git push
    - local에서의 변경 사항을 remote에 저장함
- ```
$ git push [remote-name]
```



git installation

- 다음의 명령으로 git 설치 (ubuntu)

```
$ sudo apt-get install git-all
```

```
wsul@infi1:~$ git --help
usage: git [--version] [--help] [-C <path>] [-c name=value]
       [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
       [-p|--paginate|--no-pager] [--no-replace-objects] [--bare]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       <command> [<args>]

The most commonly used git commands are:
add      Add file contents to the index
bisect   Find by binary search the change that introduced a bug
branch   List, create, or delete branches
checkout Checkout a branch or paths to the working tree
clone    Clone a repository into a new directory
commit   Record changes to the repository
diff     Show changes between commits, commit and working tree, etc
fetch    Download objects and refs from another repository
grep     Print lines matching a pattern
init     Create an empty Git repository or reinitialize an existing one
log      Show commit logs
merge    Join two or more development histories together
mv       Move or rename a file, a directory, or a symlink
pull     Fetch from and integrate with another repository or a local branch
push     Update remote refs along with associated objects
rebase   Forward-port local commits to the updated upstream head
reset    Reset current HEAD to the specified state
rm       Remove files from the working tree and from the index
show     Show various types of objects
status   Show the working tree status
tag      Create, list, delete or verify a tag object signed with GPG

'git help -a' and 'git help -g' lists available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
wsul@infi1:~$
```

git configuration

- git 설정파일

1. /etc/gitconfig
2. ~/.gitconfig | ~/.config/git/config
3. .git/config

- git 설정

```
$ git config -global core.editor vim
```

- git 설정 확인

```
$ git config --list
```

```
[wsul@infi1:~$ cat ~/.gitconfig
[user]
    name = Woong Sul
    email = sul.woong@gmail.com
[color]
    branch = auto
    diff = auto
    interactive = auto
    status = auto
[core]
    editor = vim
wsul@infi1:~$ █
```

```
[wsul@infi1:~$ git config --list
user.name=Woong Sul
user.email=sul.woong@gmail.com
color.branch=auto
color.diff=auto
color.interactive=auto
color.status=auto
core.editor=vim
wsul@infi1:~$ █
```

Thank You
