

# Version Control System

---

git / gitlab user guide

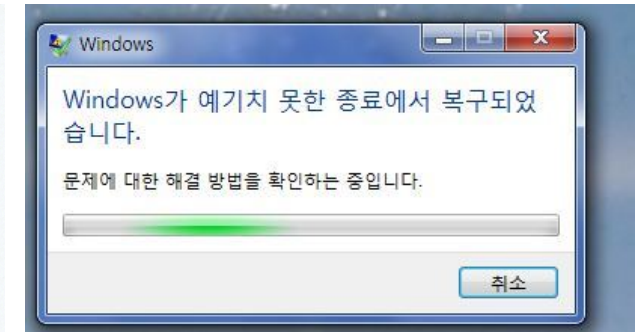
# Summary

---

- Version Control System
- git 소개 및 특징
- git 에서의 용어 및 명령어
- git 설치 및 설정하기
- gitlab 소개
- gitlab 설정 및 프로젝트 생성 실습

# 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

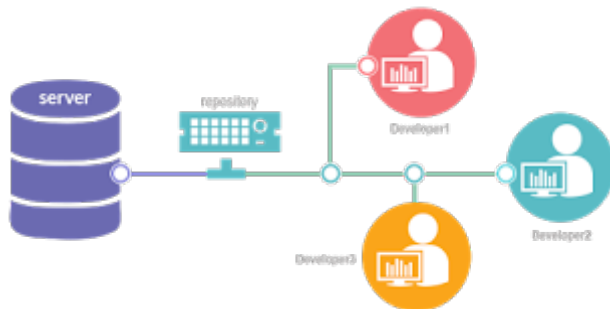
# Version Control System?

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



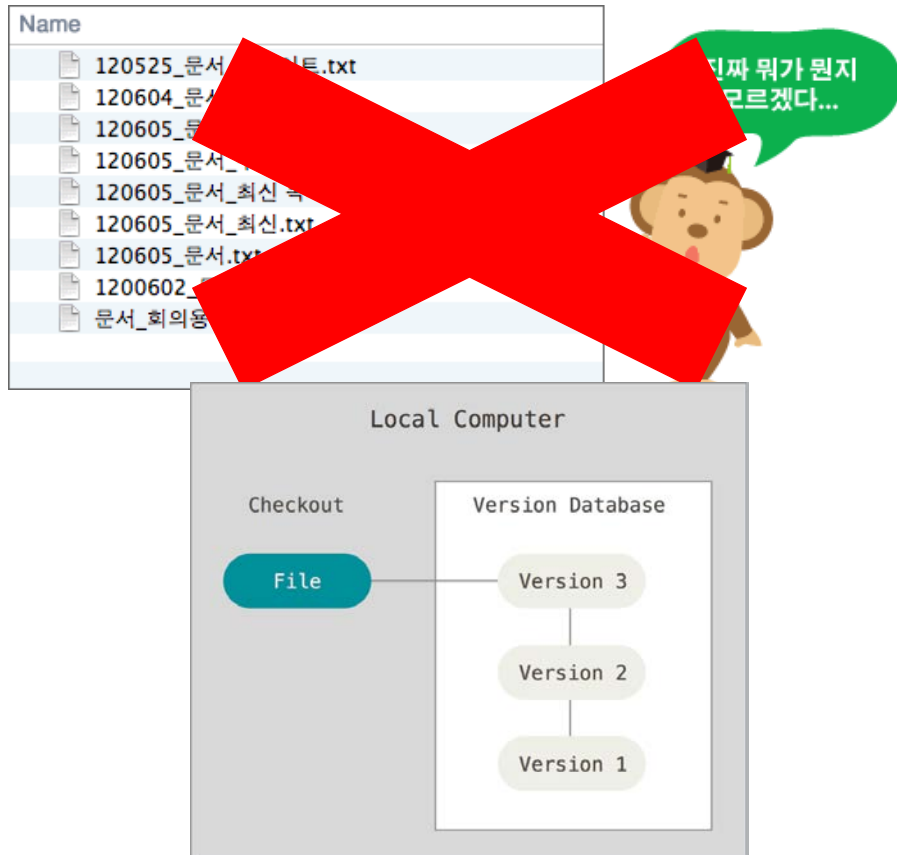
151110_코드_(1).txt	2015-11-11 오후...	텍스트 문서
151110_코드_(2) - 수정.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(3) - 기존코드백업.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(4) - 수정2.txt	2015-11-11 오후...	텍스트 문서
151110_코드_(5) - 수정3.txt	2015-11-11 오후...	텍스트 문서
151111_코드_(6) - 완성.txt	2015-11-11 오후...	텍스트 문서
151112_코드_(7) - 완성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

- Something else..?

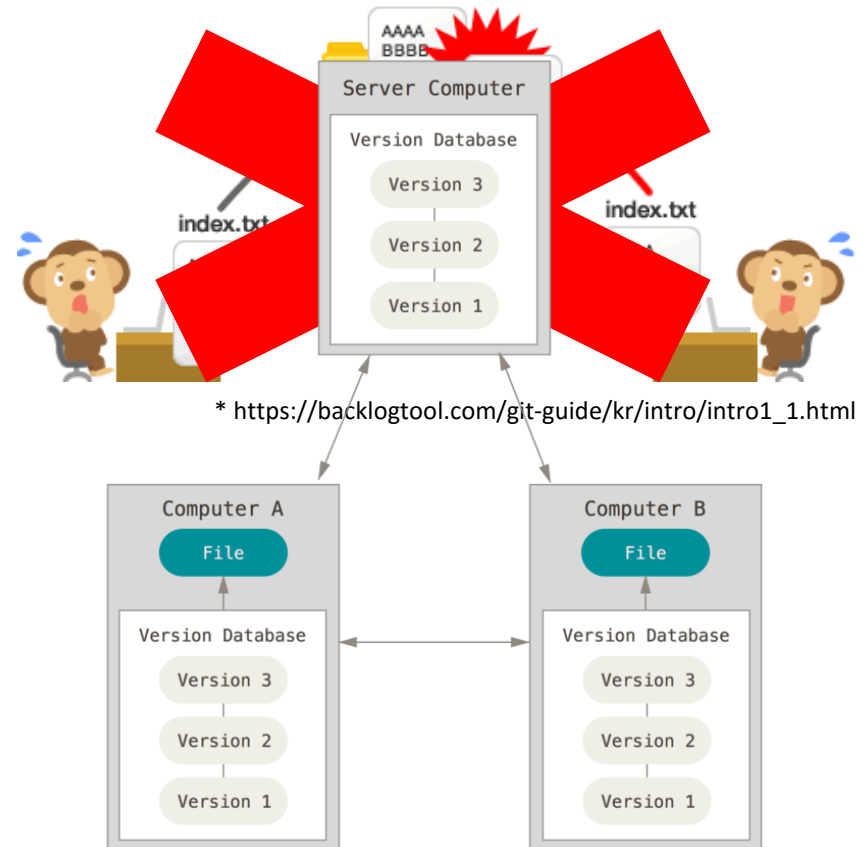


# Version Control System의 좋은점

- 작업결과의 손쉬운 이력관리



- 협업중 발생하는 이력충돌 해결

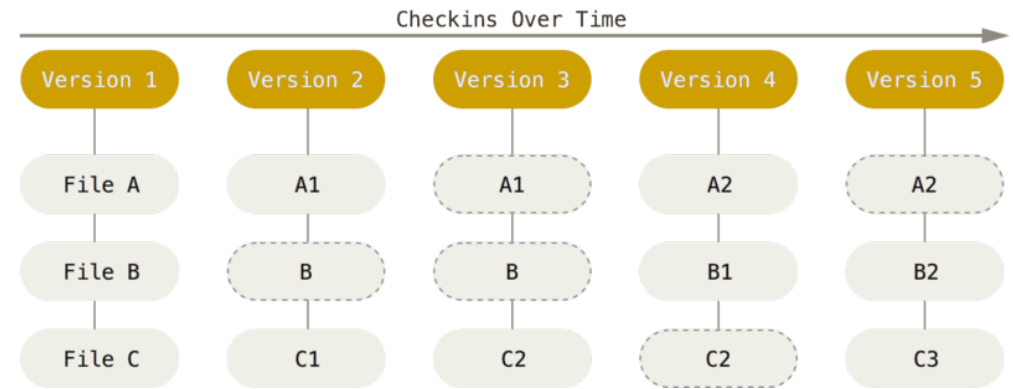
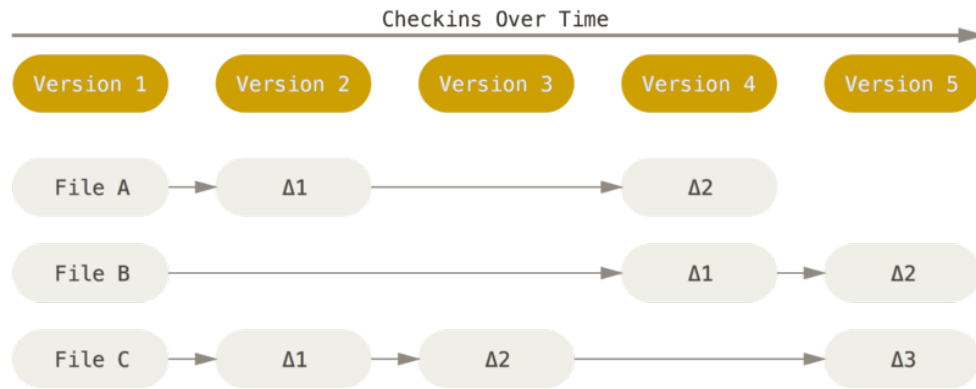


# git란?

- 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

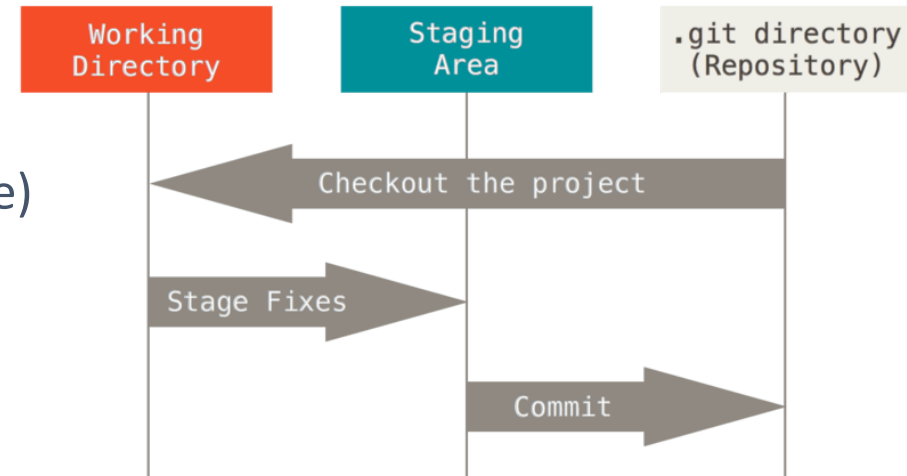
- 차이값이 아닌 Snapshot 저장



- 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

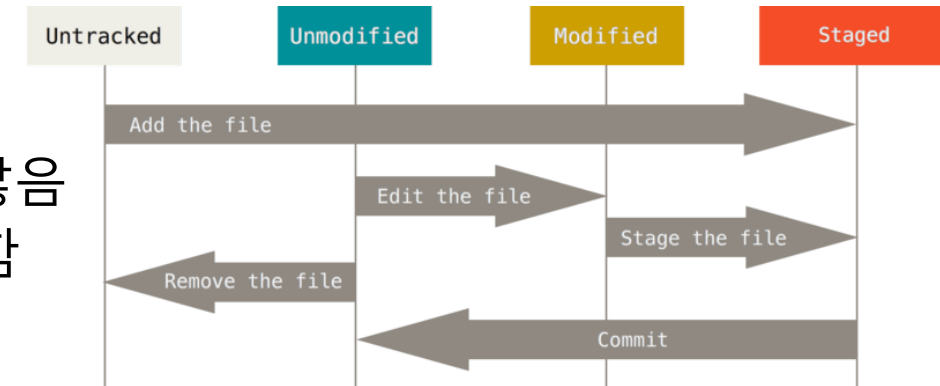
- working directory 내의 파일 상태

- 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

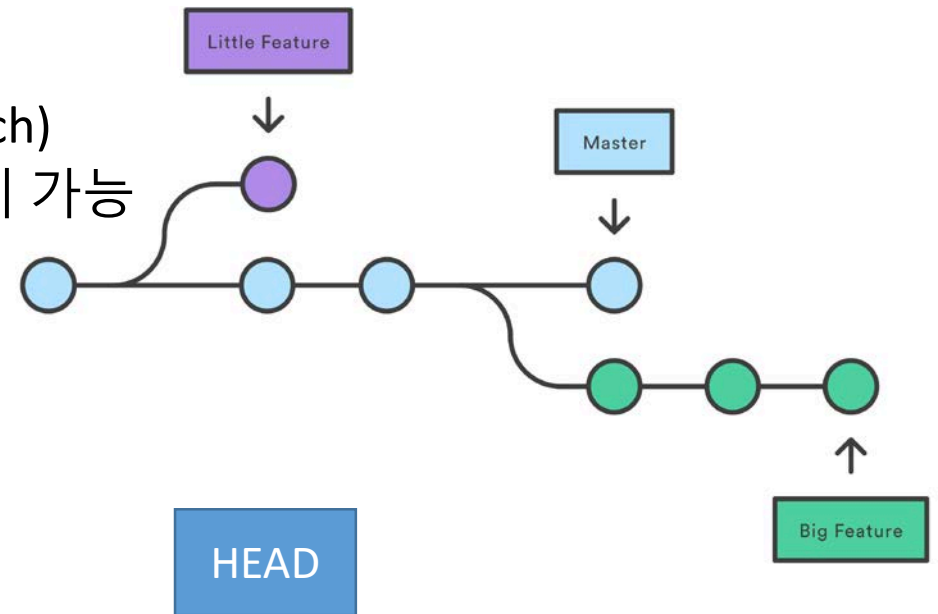
- snapshot tree에서의 main line

- Branch

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

- HEAD

- 현재 작업중인 branch



# git 명령어

- 저장소(repository) 만들기
  1. 새 저장소 만들기 (local)

```
$ git init
```
  2. 서버로 부터 저장소 만들기 (remote)

```
$ git clone [url]
```

```
$ git clone [url] [new-project-name]
```

# git 명령어

- 파일의 commit history 조회하기

- 저장소의 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 라인 수)

```
$ 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 저장소에 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 저장소에 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]

# 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
```

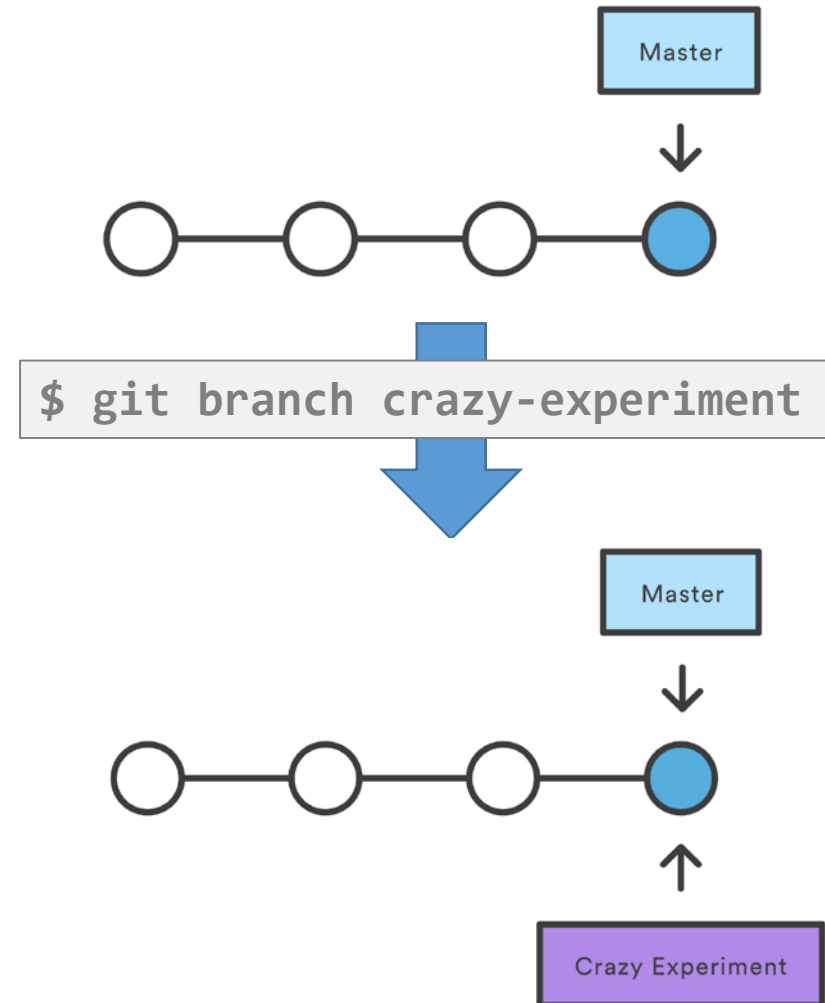
- 새로운 branch 만들기

- ```
$ 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

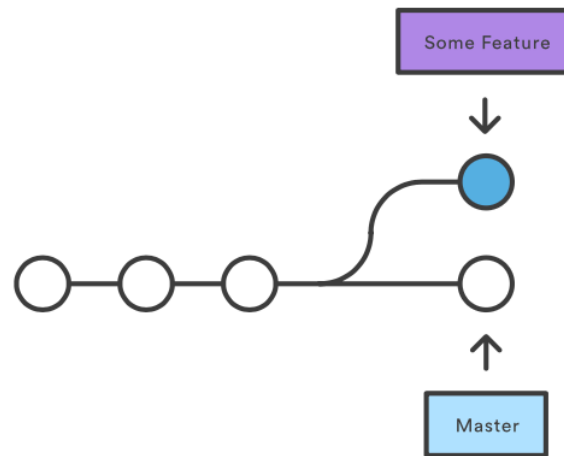
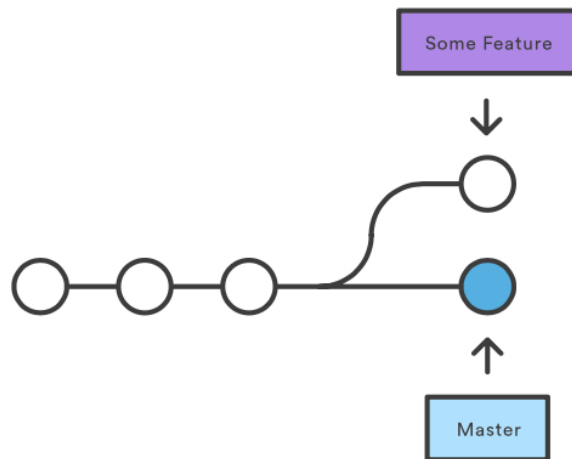
```
$ git checkout [existing-branch]
```

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

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

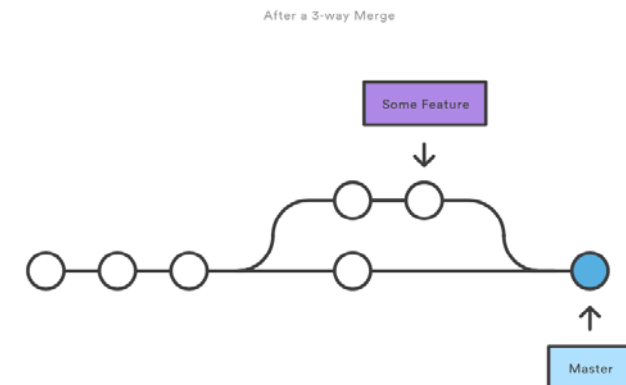
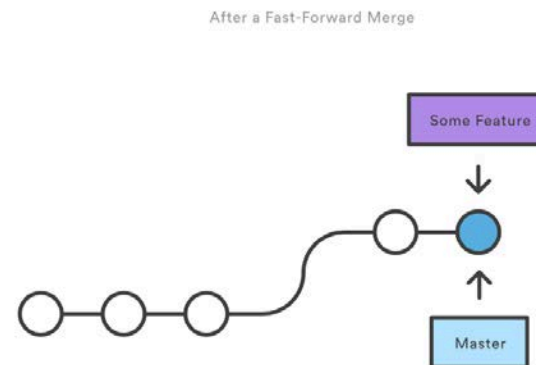
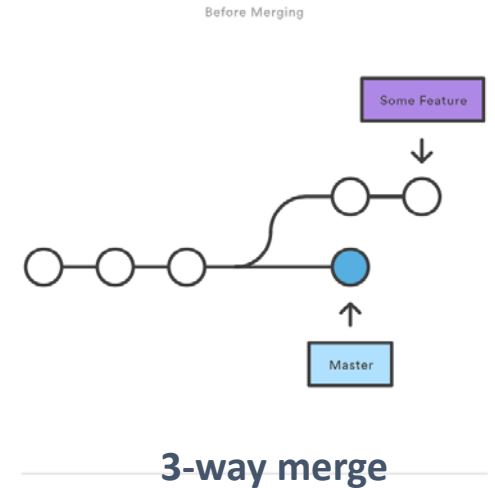
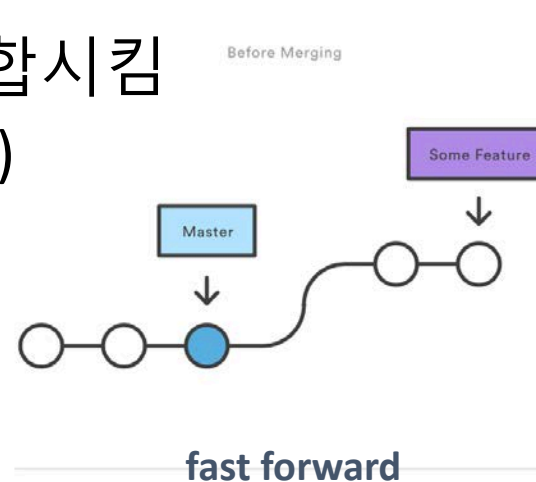
Checking Out Master

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

- 원격 저장소(remote repository) 관련 명령

- 원격 저장소 조회하기

- \$ git remote

- \$ git remote -v

- \$ git remote show [file-name]

- 원격 저장소 추가하기

- \$ git add [remote-name] [URL]

- 원격 저장소 삭제하기

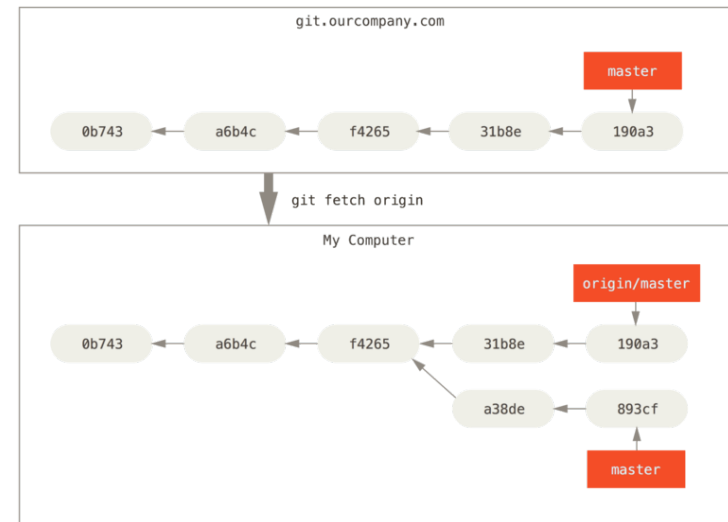
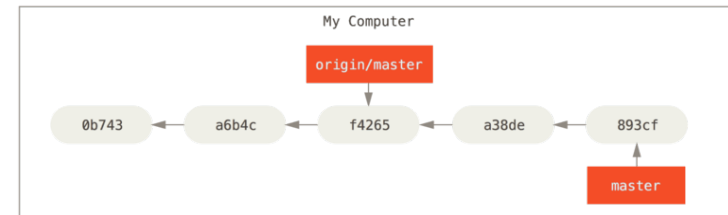
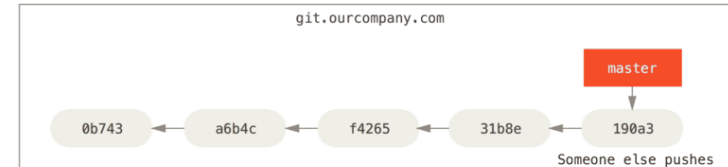
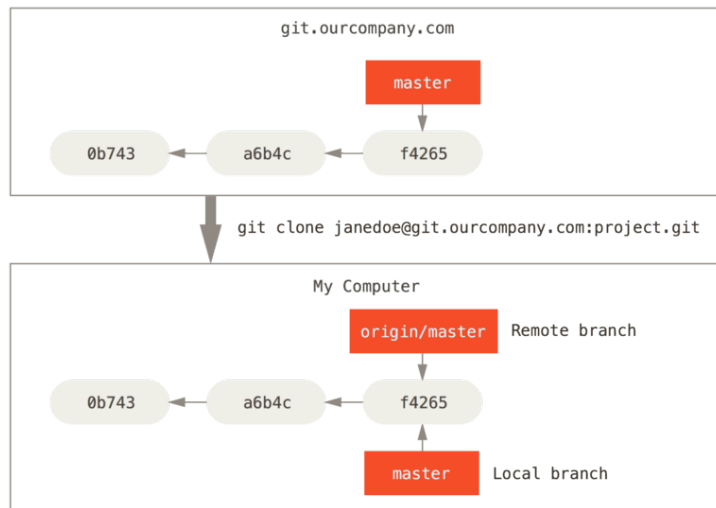
- \$ git rm [remote-name]

- 원격 저장소 이름 변경하기

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

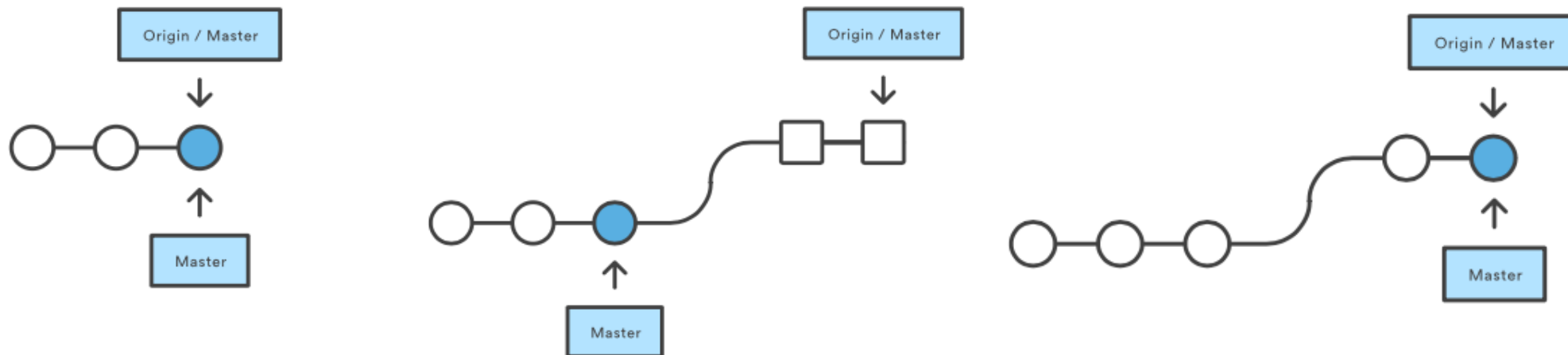
# git 명령어: remote

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



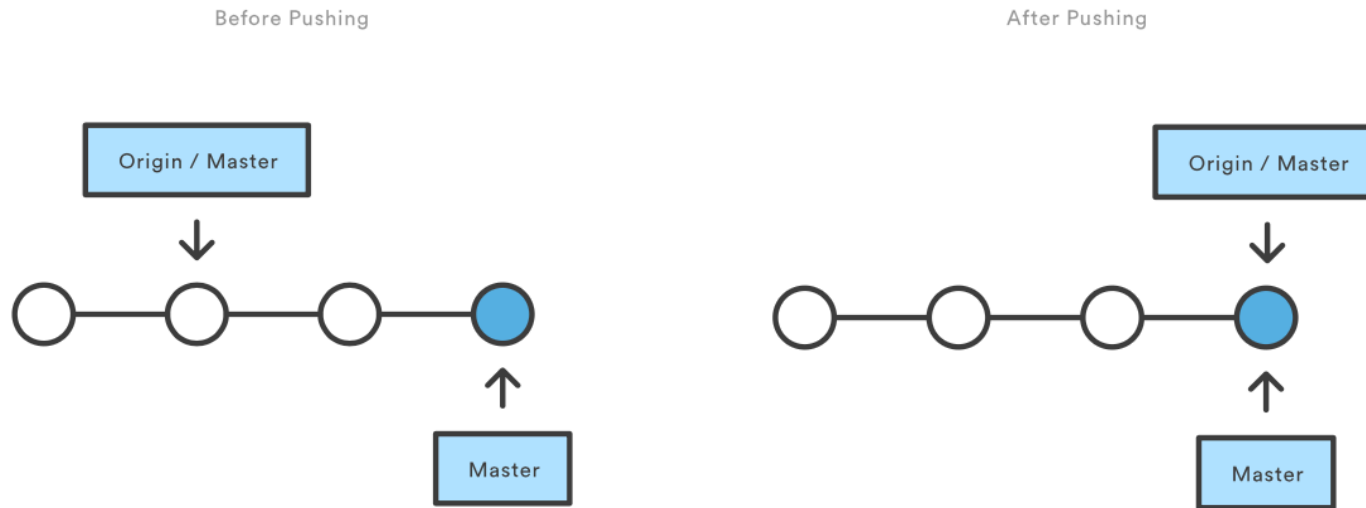
# git 명령어: remote

- 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
```

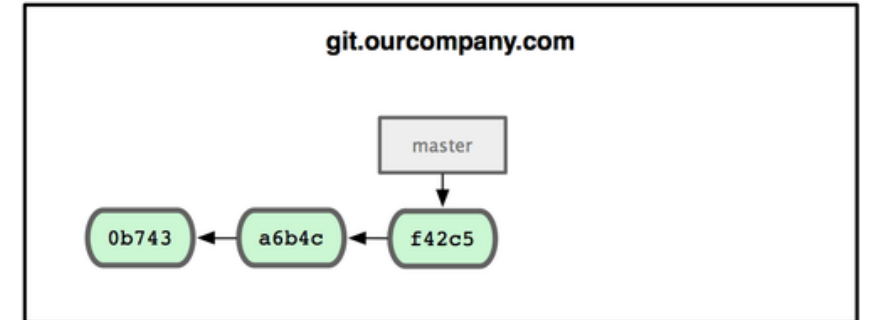
```
[wsul@infi1:~$ cat ~/.gitconfig
[user]
    name = Woong Sul
    email = sul.woong@gmail.com
[color]
    branch = auto
    diff = auto
    interactive = auto
    status = auto
[core]
    editor = vim
wsul@infi1:~$ █
```

```
[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:~$ █
```

# gitlab tutorial

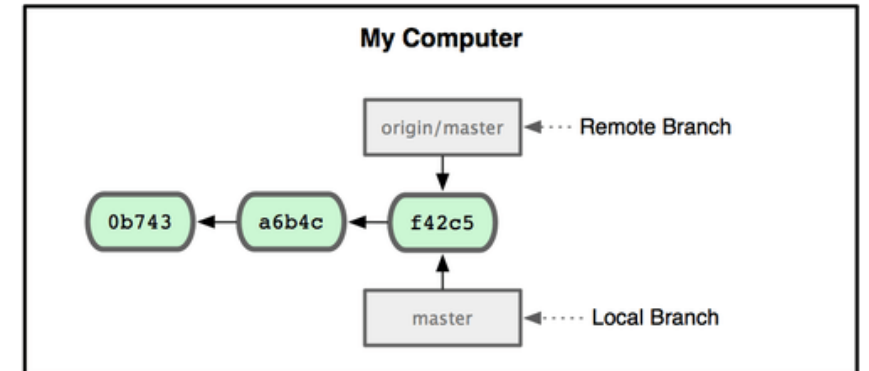
- gitlab이란?
  - Remote Repository 서비스
  - 웹 인터페이스를 통한 관리 기능 제공
  - 바로가기 ➔ [link](#)

## \* remote repository (서버)



## \* local (PC)

git clone schacon@git.ourcompany.com:project.git



# gitlab tutorial

## 1. 계정설정

- 조교에게 신청한 계정확인
- 원격 저장소(remote repository)와 통신을 위한 보안키 등록

## 2. 프로젝트 생성

- 과제제출 및 작성/변경 이력관리를 위한 프로젝트 생성
- Naming Rule → {연도}\_{과목명}\_{학번}  
ex. 2016\_ITE1015\_201220789
- 프로젝트 내에서 과제별로 directory 구분할 것  
ex. ./2016\_ITE1015\_201220789/hw1

# gitlab tutorial: 계정설정

GitLab Community Edition

Open source software to collaborate on code

Manage git repositories with fine grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

**\* 조교에게 신청한 계정정보로 로그인**

Existing user? Sign in

Username or Email

Password

☐ Remember me [Forgot your password?](#)

Sign in

New user? Create an account

Name

Username

Email

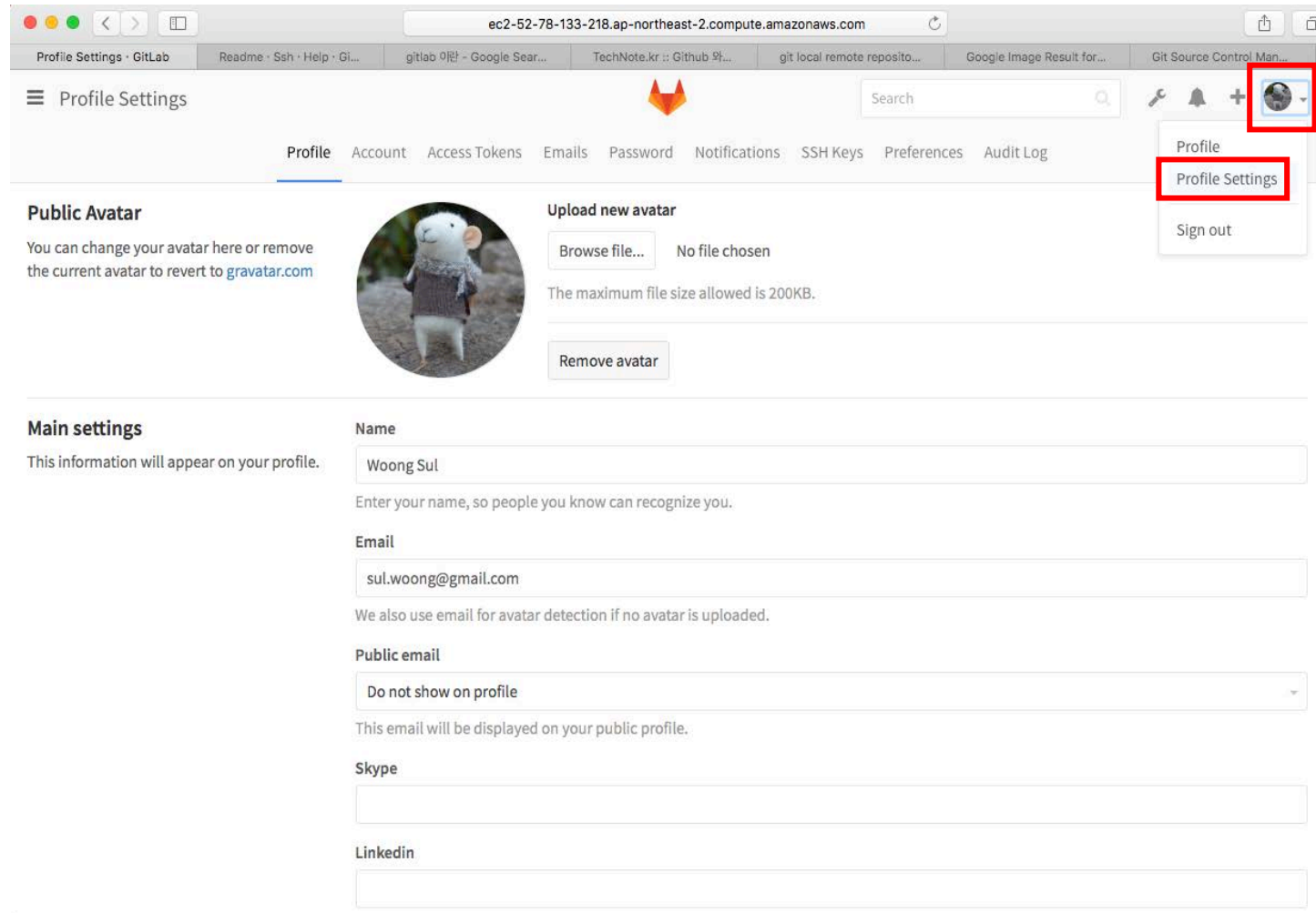
Password - minimum length 8 characters

Sign up

Didn't receive a confirmation email? [Request a new one.](#)

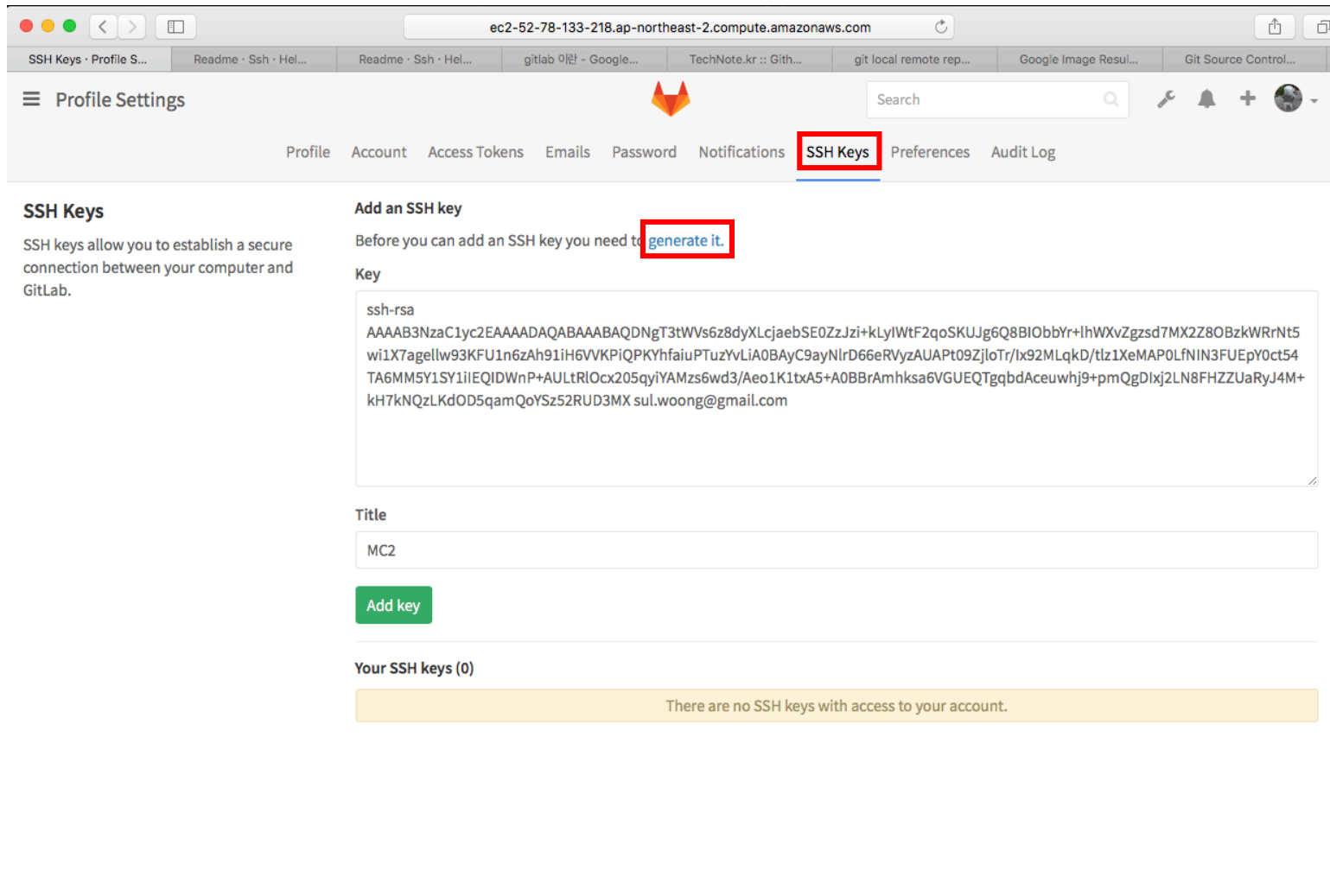
[Explore](#) [Help](#) [About GitLab](#)

# gitlab tutorial: 계정설정 - 1



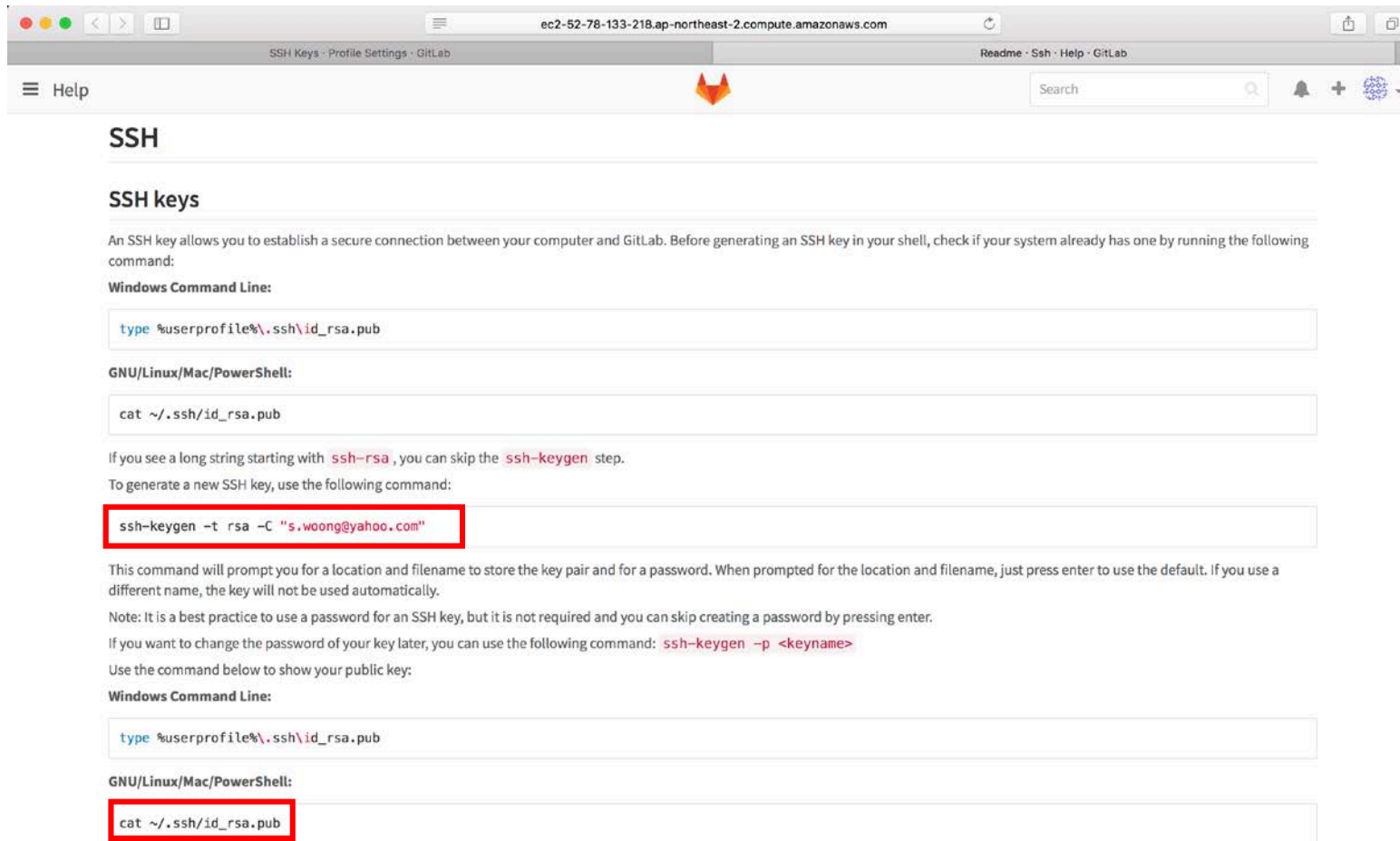
1. Profile
2. Profile setting

# gitlab tutorial: 계정설정 - 2



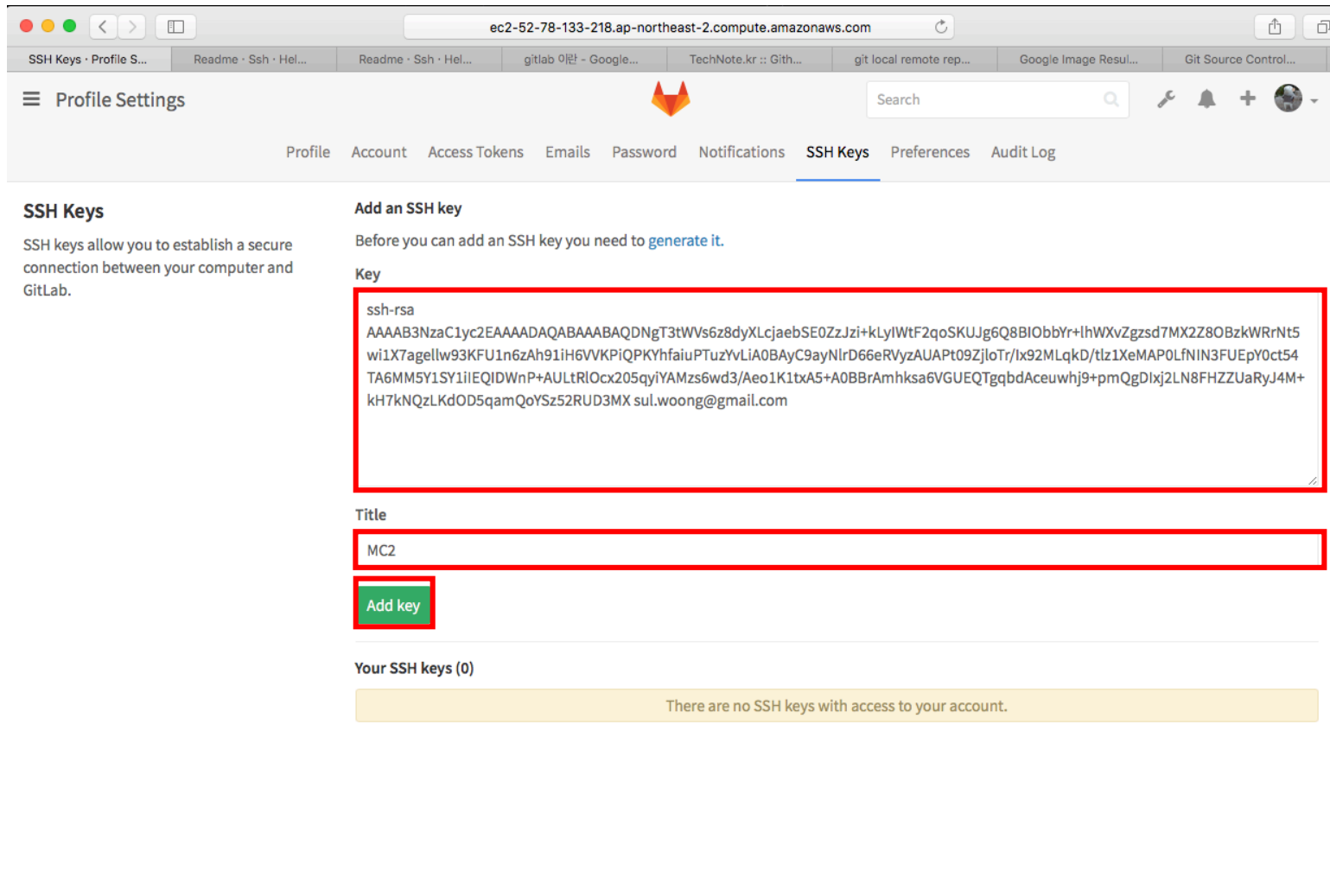
1. Profile
2. Profile setting
3. SSH keys
4. generate it (link)

# gitlab tutorial: 계정설정 - 2



1. Profile
2. Profile setting
3. SSH keys
4. generate it (link)
5. ssh key 생성
6. 생성된 key 복사

# gitlab tutorial: 계정설정 - 2



SSH Keys · Profile S...

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

**Add an SSH key**

Before you can add an SSH key you need to [generate it](#).

Key

```
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDNgt3tWVs6z8dyXLcjaebSE0Zz.Jzi+kLyIWtF2qoSKUJg6Q8BIObbYr+lhWxvZgzsd7MX2Z8OBzkWRrNt5
wi1X7agellw93KFU1n6zAh91iH6VVKPiQPKYhfaiuPTuzYvLiA0BAyC9ayNlrD66eRVyzAUAPt09ZjloTr/lx92MLqkD/tlz1XeMAP0LfNIN3FUEpY0ct54
TA6MM5Y1SY1iIEQIDWnP+AULtRlOcx205qyiYAMzs6wd3/Aeo1K1txA5+A0BBRAmhksa6VGUEQTgqbdAceuwHj9+pmQgDlxj2LN8FHZZUaRyJ4M+
kH7kNQzLKdOD5qamQoYSz52RUD3MX sul.woong@gmail.com
```

Title

MC2

Add key

Your SSH keys (0)

There are no SSH keys with access to your account.

1. Profile
2. Profile setting
3. SSH keys
4. generate it (link)
5. ssh key 생성
6. 생성된 key 복사
7. 생성된 key 붙이기
8. Title 입력
9. Add key



# gitlab tutorial: 계정설정 -4

ec2-52-78-133-218.ap-northeast-2.compute.amazonaws.com

SSH Keys · Profile S... Readme · Ssh · Hel... Readme · Ssh · Hel... gitlab 이란 - Google... TechNote.kr :: Gith... git local remote rep... Google Image Resul... Git Source Control...

Profile Settings

Profile Account Access Tokens Emails Password Notifications **SSH Keys** Preferences Audit Log

**SSH Keys**

SSH keys allow you to establish a secure connection between your computer and GitLab.

**Add an SSH key**

Before you can add an SSH key you need to [generate it](#).

**Key**

Don't paste the private part of the SSH key. Paste the public part, which is usually contained in the file '~/.ssh/id\_rsa.pub' and begins with 'ssh-rsa'.

**Title**

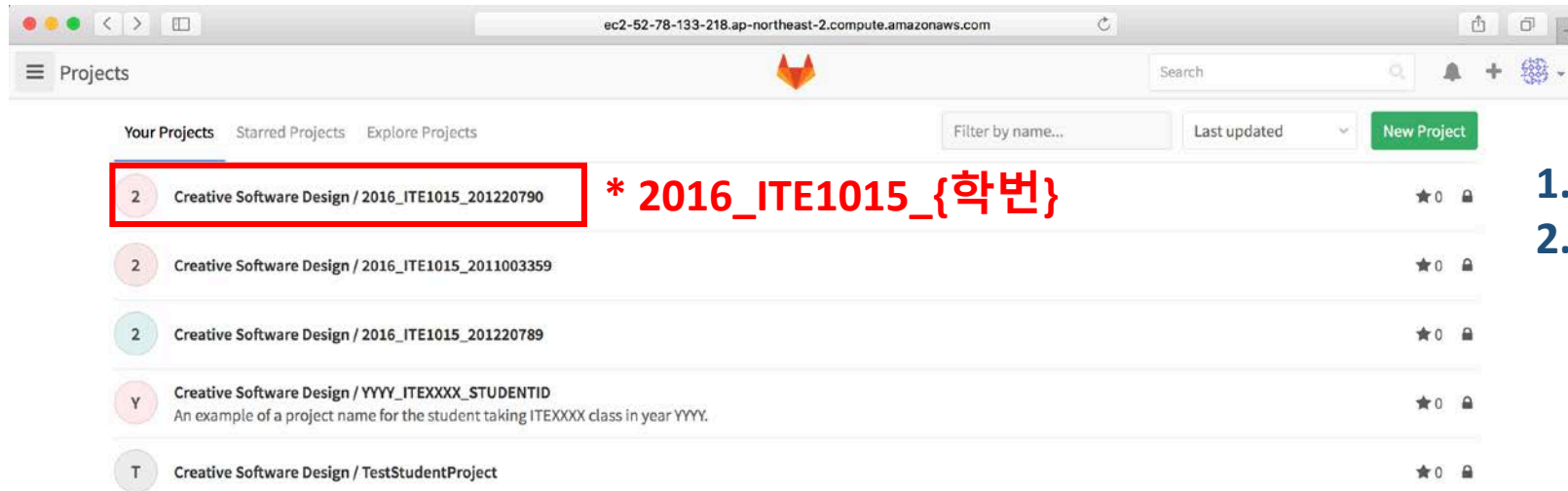
Add key

**Your SSH keys (1)**

MC2  
05:e4:6a:27:57:3a:3b:a1:9b:a5:d4:f1:ac:61:2c:82 created 3 minutes ago

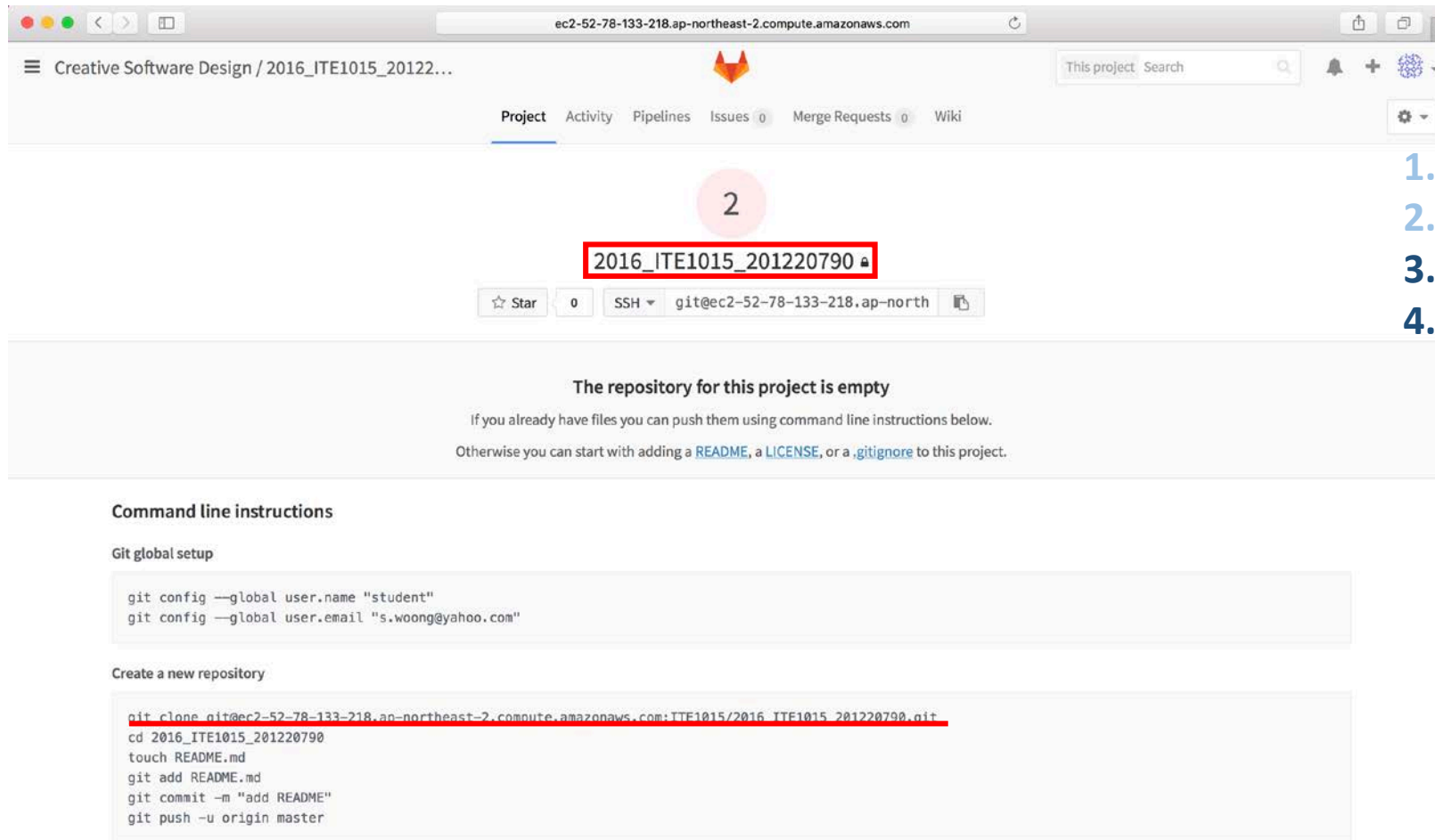
1. Profile
2. Profile setting
3. SSH keys
4. generate it (link)
5. ssh key 생성
6. 생성된 key 복사
7. 생성된 key 붙이기
8. Title 입력
9. Add key
10. key 등록 확인

# gitlab tutorial: 프로젝트 만들기 - 1



1. Project name 확인
2. Link click

# gitlab tutorial: 프로젝트 만들기 - 2



ec2-52-78-133-218.ap-northeast-2.compute.amazonaws.com

Creative Software Design / 2016\_ITE1015\_20122...

Project Activity Pipelines Issues 0 Merge Requests 0 Wiki

2

2016\_ITE1015\_201220790

Star 0 SSH git@ec2-52-78-133-218.ap-north

The repository for this project is empty

If you already have files you can push them using command line instructions below.

Otherwise you can start with adding a [README](#), a [LICENSE](#), or a [.gitignore](#) to this project.

Command line instructions

Git global setup

```
git config --global user.name "student"
git config --global user.email "s.woong@yahoo.com"
```

Create a new repository

```
git clone git@ec2-52-78-133-218.ap-northeast-2.compute.amazonaws.com:ITE1015/2016_ITE1015_201220790.git
cd 2016_ITE1015_201220790
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

1. Project name 확인
2. Link click
3. 내 프로젝트 확인
4. Instructions을 따름

# gitlab tutorial: 프로젝트 확인

```
[wsul@mc2:~/Projects]$ git clone git@ec2-52-78-133-218.ap-northeast-2.compute.amazonaws.com:ITE1015/2016_ITE1015_201220789.git
Cloning into '2016_ITE1015_201220789'...
Enter passphrase for key '/home/wsul/.ssh/id_rsa':
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
wsul@mc2:~/Projects$ ll
total 12
drwxrwxr-x 3 wsul wsul 4096 Sep  7 13:35 ./
drwxr-xr-x 6 wsul wsul 4096 Sep  6 21:48 ../
drwxrwxr-x 3 wsul wsul 4096 Sep  7 13:35 2016_ITE1015_201220789/
wsul@mc2:~/Projects$
```

1. 복사한 ssh URL로 clone

2. 프로젝트 확인

# Thank You

---