

버전 관리 툴

□ 버전 관리 툴이란?

□ CVS

- CVS란?
- CVS를 왜 사용하는가?
- CVS 사용하기

□ Git

- Git ?
 - Git Repository 생성
 - Git 설치하기
 - Git 사용하기
-

□ 버전 관리(revision control)

- 소스 코드 관리라고도 한다.
- 동일한 정보에 대한 여러 버전을 관리하는 것을 말한다.
- 소프트웨어 개발에서 팀 단위로 개발 중인 소스 관리에 사용
- 문서의 변경 사항들에 숫자나 문자로 이뤄진 “버전”을 부여하여 구분한다.
- “버전” 통해서 시간적으로 변경 사항과 그 변경 사항을 작성한 작업자를 추적할 수 있다.

버전 관리를 왜 사용하는가?

□ 공동 프로젝트 관리

- 버전 관리를 사용하면 대규모의 프로젝트에서 각 모듈 개발자가 자신의 버전을 유지하면서 전체 프로젝트에 참여할 수 있도록 할 수 있음

□ 프로젝트 백업

- 버전관리는 중간에 서버가 있어서, 프로젝트 데이터의 저장소 역할을 하여 실수로 자신의 프로젝트가 날아간다고 해도 염려할 필요가 없음

□ 데이터 동기화

- 한 명이 여러 공간에서 작업을 하는 경우, 한 곳에서 작업 후 서버에 등록하고, 다른 곳에서 다시 작업을 시작할 때 서버에 등록된 최신의 데이터를 받아와서 작업을 함으로써 데이터 동기화가 가능

버전의 의미

□ 버전의 의미

1. 어떤 소프트웨어가 몇 번 개정되었는지를 나타내는 번호. 기능이 보완되거나 추가될 때 버전을 올려 나감
2. 한 소프트웨어를 서로 다른 시스템 환경에서 사용할 수 있도록 각각 제작된 프로그램을 이르는 말

ex) MS-DOS 6.2

→ 버전의 수치는 개정의 정도를 의미한다. 맨 처음 소프트웨어가 공식적으로 발표되면 보통 1.0이란 버전이 붙게 되고, 그 기능이 개선되면서 점점 버전 숫자가 커지게 되는데, 소수점 위의 숫자는 큰 개정을 의미하고, 소수점 아래의 숫자는 약간의 개정을 의미한다.

버전의 의미

버전 1.1 main.c

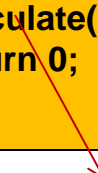
```
main.c

#include "diary.h"
Int main()
{
    memo();
    calendar();
    return 0;
}
```

버전 1.2 main.c

```
main.c

#include "diary.h"
Int main()
{
    memo();
    calculate();
    return 0;
}
```

 **calendar();** 삭제되거나
calculate();라는 새로운
소스가 추가됨.

- ◆ 위와 같이 처음 1.1버전의 **main.c**의 파일이 수정되어 1.2버전의 **main.c**파일이 되어 버전업된다.

□ 저장소(Repository)

- 파일의 현재 버전과 변경 이력 정보를 저장하는 저장소

□ 가져오기(Import)

- 버전 관리가 되고 있지 않은 로컬 디렉토리의 파일을 처음으로 저장소에 복사한다.

□ 체크 아웃(Check Out)

- 저장소(Repository)에서 파일을 가져온다.

□ 체크 인(Check In, Commit)

- 체크아웃(Check Out)한 파일의 수정이 끝난 경우 저장소에 새로운 버전으로 갱신 하는 일이다.
- 이때 이전에 갱신된 것이 있는 경우 충돌(Conflict)을 알려주며 diff 도구를 이용해 수정하고 commit하는 과정을 거치게 된다.

□ 버전 관리 툴이란?

□ CVS

- CVS란?
- CVS 설치하기
- CVS 사용하기

□ Git

- Git ?
 - Git Repository 생성
 - Git 설치하기
 - Git 사용하기
-

CVS란?

- Concurrent Version System 의 줄임말
- 공동 버전 시스템
- "공동으로 진행하는 프로젝트의 버전 관리 시스템"



□ CVS 서버 설치하기

- `sudo apt-get install cvs`
- `sudo apt-get install xinetd`
- `sudo apt-get install apache2-utils`

□ CVS 유저 추가

```
#sudo adduser cvs
```

- 계정을 생성 한 후, **cvs** 계정 홈폴더에 프로젝트 파일들을 보관하는 저장소(**Repository**)를 생성

```
#sudo mkdir /home/cvs/repository
```

저장소 디렉토리 생성 및 초기화

```
jyn@jyn-sslslab: ~  
File Edit View Search Terminal Help  
jyn@jyn-sslslab:~$ sudo adduser cvs  
[sudo] password for jyn:  
Adding user `cvs' ...  
Adding new group `cvs' (1001) ...  
Adding new user `cvs' (1001) with group `cvs' ...  
Creating home directory `/home/cvs' ...  
Copying files from `/etc/skel' ...  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully  
Changing the user information for cvs  
Enter the new value, or press ENTER for the default  
    Full Name []: cvs  
    Room Number []:  
    Work Phone []:  
    Home Phone []:  
    Other []:  
Is the information correct? [Y/n] y  
jyn@jyn-sslslab:~$ sudo mkdir /home/cvs/repository  
jyn@jyn-sslslab:~$
```

CVS 유저 생성 및 권한 설정

- 저장소 디렉토리에 **CVS**에서 사용하는 제어파일들이 들어있는 디렉토리를 생성하기 위해 초기화
 - -d 옵션 : cvs 저장소의 위치

```
#sudo cvs -d /home/cvs/repository init
```

- 초기화한 후, 시스템에 **CVS**유저를 등록하고 저장소 디렉토리에 대한 **CVS**유저의 접근 권한을 설정

```
#sudo chown -R cvs /home/cvs/repository  
#sudo chgrp -R cvs /home/cvs/repository
```

- -R 옵션 : 경로와 그 하위까지 모두 바꿈

- **CVS** 서버에 인증을 거쳐 프로젝트를 받아올 때 인증에 유효한 사용자를 생성
- **CVS**사용자는 시스템 사용자와 같이 실제 시스템 사용 권한을 가지고 있는 대상은 아니며 단지 **CVS**서버만이 인지하는 사용자임
- **CVS** 사용자를 등록하는 방법은 저장소 디렉토리의 **CVSROOT** 디렉토리 내에 **passwd**파일을 생성하면 됨

- 사용자 등록을 하기 위해 **passwd** 파일 생성

```
#cd /home/cvs/repository/CVSROOT  
#sudo htpasswd -c passwd cvs  
암호 입력
```

- **passwd** 파일 수정

```
#sudo vi passwd
```



```
cvs:JzA129Z7vWqpg
```

 →

```
cvs:JzA129Z7vWqpg:cvs
```

첫 번째는 사용자 ID, 두 번째 JzA129Z7vWqpg는 암호화된 사용자의 암호
세 번째에 추가하는 cvs 는 서버에 파일을 기록할 때 파일의 권한을 가지는 계정

□ 기본 명령어 형식

```
#cvs [cvs 옵션] 명령 [명령 옵션과 인자]
```

□ 모든 CVS 명령은 저장소의 위치를 알아야 수행됨

- 환경변수 CVSROOT 의 값으로 저장소의 위치 지정
- 또는 명령어 마다 -d 옵션 사용

CVS 명령의 종류

전체 명령	동의어	명령 실행
login		cvس 서버에 로그인
logout		.cvspass 파일에서 저장소 제거
import		프로젝트 파일 등록
checkout	co, get	프로젝트 파일 가져오기
commit	ci	프로젝트 파일 수정 후 cvs 서버에 반영
update	up	CVS 서버의 최신 버전을 작업 디렉토리에 반영
add	new	파일 또는 디렉토리 추가
remove	m, delete	파일 삭제

CVS 명령의 종류(계속)

전체 명령	동의어	명령 실행
diff		파일 버전에 따른 차이점 비교
log	rlog	파일 로그 보기
annotate	ann	행별 정보 출력(작성 날짜, 작성자 등)
status		파일 상태 보기
history		각종 히스토리 보기
tag		프로젝트 파일에 태깅하기
rtag		저장소 디렉토리에 태깅하기
release		모듈 release

□ 버전 관리 툴이란?

□ CVS

- CVS란?
- CVS 설치하기
- CVS 사용하기

□ Git

- Git ?
 - Git Repository 생성
 - Git 설치하기
 - Git 사용하기
-

□ Git

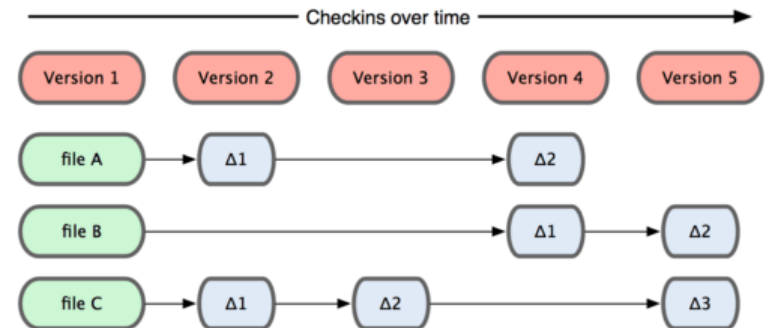
- Git은 프로그램등의 소스 코드 관리를 위한 분산 버전 관리 시스템
- 특징
 - 델타가 아닌 스냅샷으로 저장
 - 분산 버전 관리 시스템
 - 빠른 속도로 대형 프로젝트에 사용하기 좋음
 - 해시값을 이용한 무결성
 - Committed, Modified, Staged
 - 브랜치 사용

Git이란?

□ 델타와 스냅샷

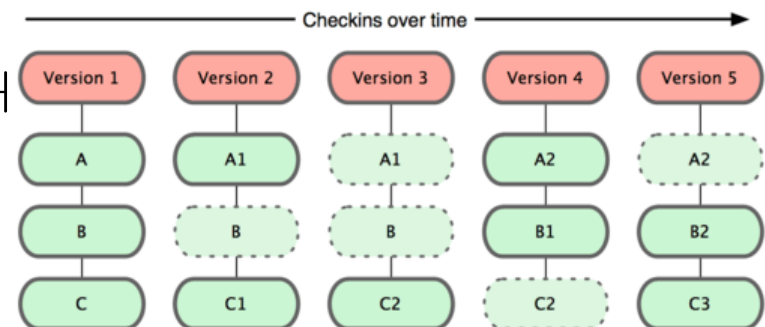
■ 델타

- 파일의 변화를 저장한다.
- ex) version 5를 내려 받을 때,
모든 변경내역을 내려 받는다.



■ 스냅샷

- 순간의 스냅샷으로 저장한다.
- ex) version 5를 내려 받을 때
A2,B2,C3를 내려 받는다.

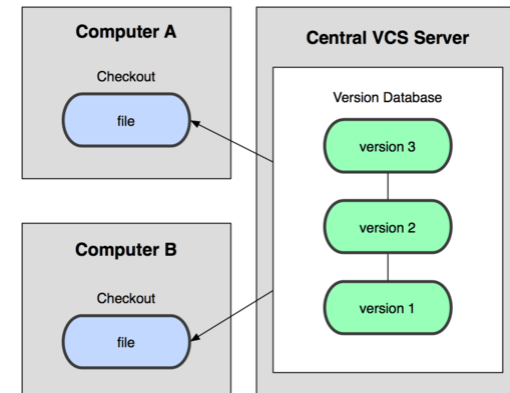


Git이란?

□ 버전 관리

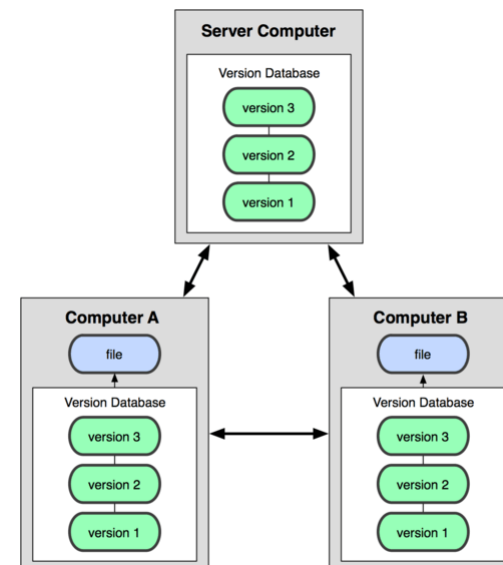
■ 중앙집중식 버전 관리 시스템

- 하나의 중앙서버와 다수의 클라이언트가 존재
- 서버가 다운될 경우 버전관리가 불가능



■ 분산 버전 관리 시스템

- 클라이언트가 마지막 스냅샷을 가져오는 대신
저장소를 통째로 복제
언제든지 서버 복구 가능



☐ 파일 관리

■ Committed

- ☐ 데이터를 로컬 데이터베이스에 저장

■ Modified

- ☐ 수정한 파일을 아직 로컬 데이터 베이스에 **commit** 하지 않은 상태

■ Staged

- ☐ 현재 수정 파일을 곧 **commit** 할 것 이라고 표시한 상태
-

Git이란?

□ Git 디렉토리

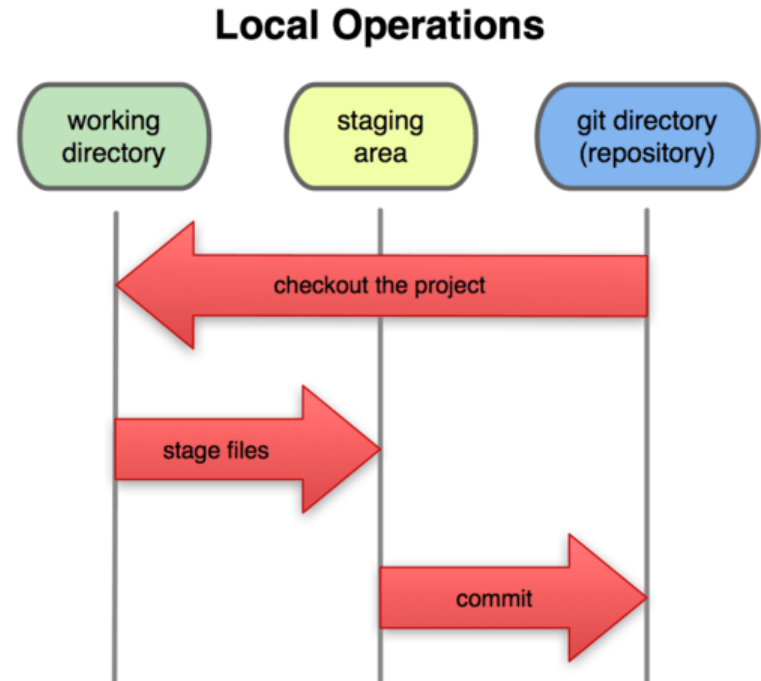
- 메타데이터와 객체 데이터베이스를 저장하는 곳
- 다른 저장소를 Clone 할 때 git 디렉토리가 생성

□ Working 디렉토리

- 프로젝트의 특정 버전을 가져온 데이터(Checkout)
- Git 디렉토리에서 압축된 데이터베이스를 가져와 구성한다

□ Staging Area

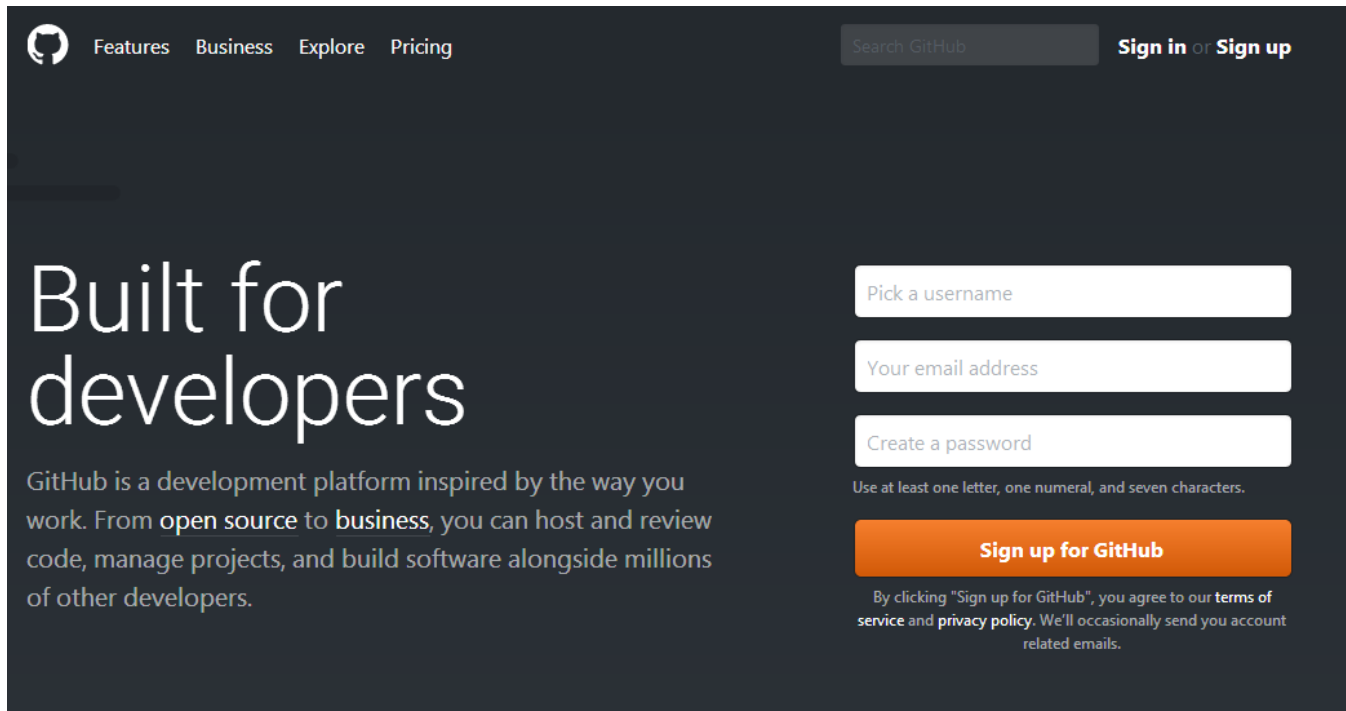
- Git 디렉토리에 있고, 곧 commit 할 파일에 대한 정보를 저장한다.



Git Repository 생성

□ Git 저장소 생성하기

- Git-Hub 아이디 만들기 (<https://github.com/>)



The screenshot shows the GitHub sign-up page. At the top, there is a navigation bar with the GitHub logo, links for Features, Business, Explore, and Pricing, a search bar labeled 'Search GitHub', and links for 'Sign in' or 'Sign up'. The main content area has a dark background. On the left, it says 'Built for developers' in large white text, followed by a paragraph: 'GitHub is a development platform inspired by the way you work. From open source to business, you can host and review code, manage projects, and build software alongside millions of other developers.' On the right, there are three white input fields: 'Pick a username', 'Your email address', and 'Create a password'. Below the password field, there is a small text requirement: 'Use at least one letter, one numeral, and seven characters.' At the bottom right, there is an orange button labeled 'Sign up for GitHub'. Below the button, there is a small disclaimer: 'By clicking "Sign up for GitHub", you agree to our terms of service and privacy policy. We'll occasionally send you account related emails.'

Git Repository 생성

- **Git** 저장소 생성하기
 - 리눅스에 Git 설치

```
#sudo apt-get install git
```

- 아이디 및 Email 설정

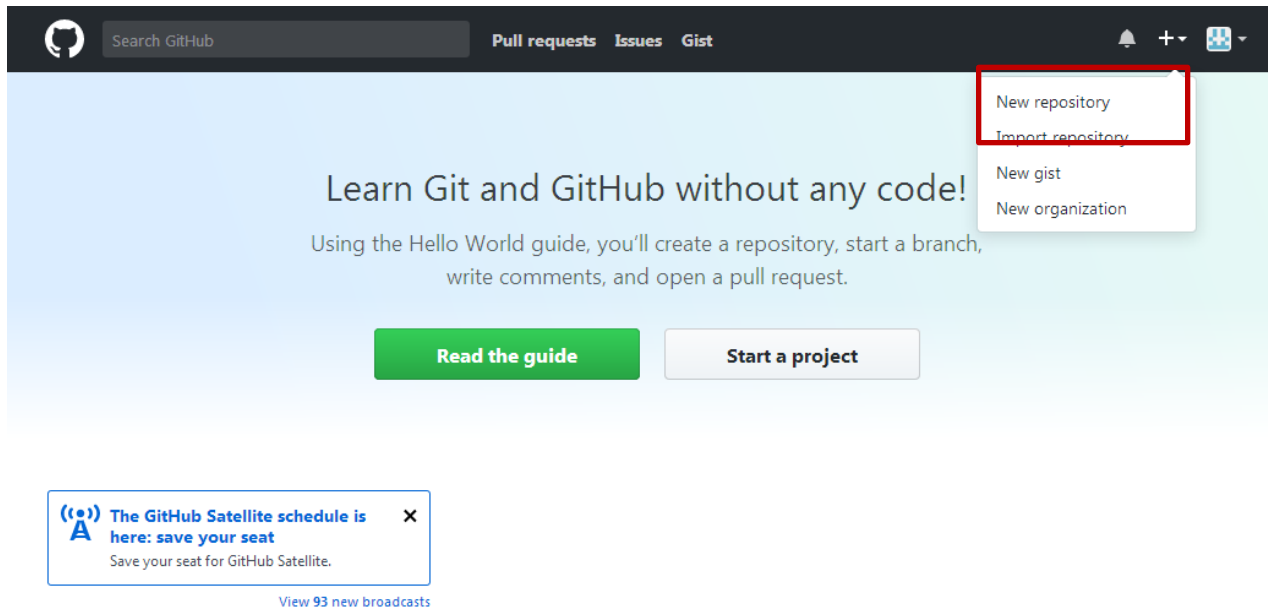
```
#git config --global user.name 아이디  
#git config --global user.email Email
```

Git Repository 생성

□ Git 저장소 생성하기

■ Repo / Repository 생성

□ 오른쪽 위 '+' 클릭 후 New repository 클릭

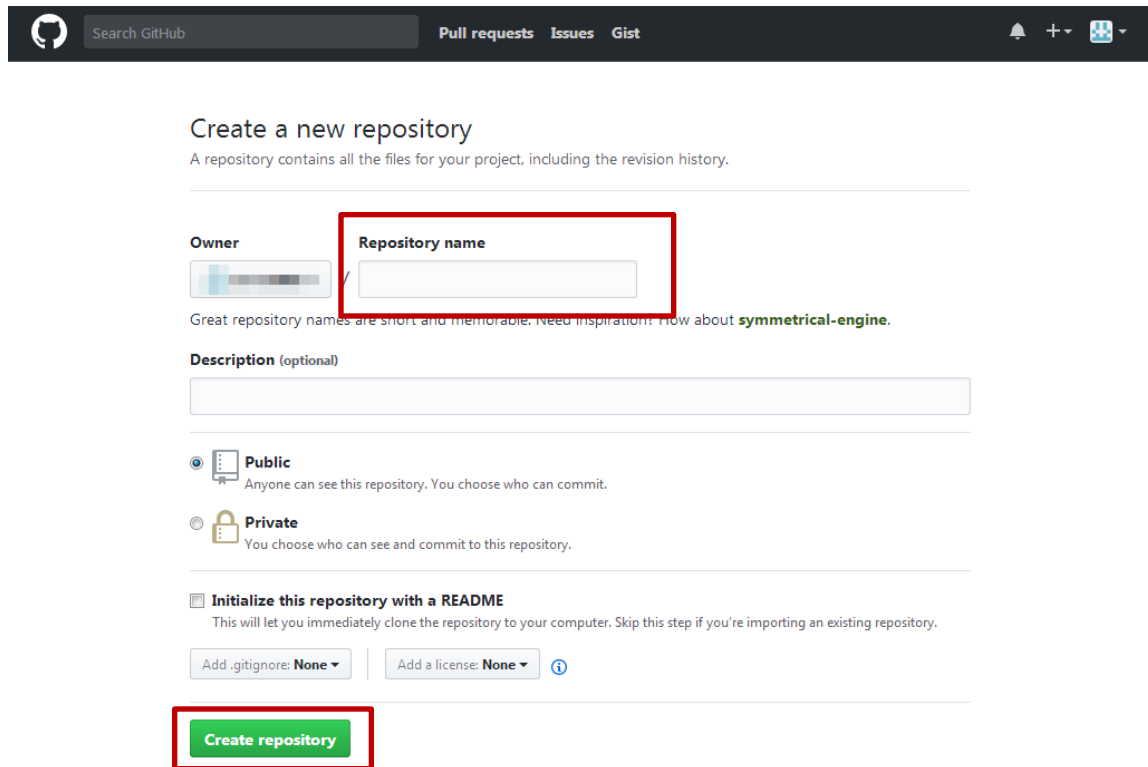


Git Repository 생성

□ Git 저장소 생성하기

■ Repo / Repository 생성

□ Repository Name 설정 후 Create Repository 클릭



Search GitHub Pull requests Issues Gist

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

Repository name

Great repository names are short and memorable. Need inspiration? How about **symmetrical-engine**.

Description (optional)

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

Git Repository 생성

- **Git** 저장소 생성하기
 - 로컬 컴퓨터에 디렉토리 생성
 - 리눅스에 생성한 Repository 의 이름과 같은 디렉토리 생성

#mkdir 디렉토리

Git Repository 생성

□ Git 저장소 생성하기

■ Git 초기화

- 생성한 로컬 디렉토리로 이동 후 git init 을 입력

```
dlcksw@dlcksw-Virtual-Machine:~/Repository$ git init
Initialized empty Git repository in /home/dlcksw/Repository/.git/
dlcksw@dlcksw-Virtual-Machine:~/Repository$ ll
total 12
drwxrwxr-x  3 dlcksw dlcksw 4096 4월 26 17:15 ./
drwxr-xr-x 20 dlcksw dlcksw 4096 4월 26 17:13 ../
drwxrwxr-x  7 dlcksw dlcksw 4096 4월 26 17:15 .git/
```

Git 서버에 프로젝트 파일 등록

- 아래의 파일들을 **git**과 연결된 디렉토리에 생성

diary.h

```
#include<stdio.h>
int memo();
int calendar();
```

memo.c

```
#include "diary.h"
int memo()
{
    printf("function memo.\n");
    return 0;
}
```

main.c

```
#include "diary.h"
Int main()
{
    memo();
    calendar();
    return 0;
}
```

calendar.c

```
#include "diary.h"
Int calendar()
{
    printf("function calendar.\n");
    return 0;
}
```

Git 서버에 프로젝트 파일 등록

□ Git

■ 명령어

□ git status

- Git 파일 상태 확인하기
- Git에 등록된 파일들만 git이 관리한다. 따라서 파일을 생성하거나 수정을 했을 경우 git에 알려줘야 git이 업데이트를 해준다.
- (붉은색으로 된 것은 git이 신경을 쓰지 않는 파일들)

```
dlcksw@dlcksw-Virtual-Machine:~/Repository$ vi diary.h
dlcksw@dlcksw-Virtual-Machine:~/Repository$ vi memo.c
dlcksw@dlcksw-Virtual-Machine:~/Repository$ vi main.c
dlcksw@dlcksw-Virtual-Machine:~/Repository$ vi calendar.c
dlcksw@dlcksw-Virtual-Machine:~/Repository$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    calendar.c
    diary.h
    main.c
    memo.c

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

Git 서버에 프로젝트 파일 등록

□ Git

■ 명령어

□ git add [파일명]

- Git add 명령어를 통해 파일을 git에 알려준다.
- (예제는 *.c를통해 c파일만올렸으므로 diary.h도 나중에 포함시켜줄것)

```
dlcksw@dlcksw-Virtual-Machine:~/Repository$ git add *.c
dlcksw@dlcksw-Virtual-Machine:~/Repository$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   calendar.c
        new file:   main.c
        new file:   memo.c

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

        diary.h
```


□ Git

■ 명령어

□ git commit

- Staging area에 저장된 데이터를 Git 디렉토리에 기록한다.
- 옵션 `-m` 과 함께 `commit` 할 때 메시지를 입력하여 무엇을 수정하였는지 적는다.

```
dlcksw0@dlcksw0-Virtual-Machine:~/Repository$ git commit -m "Init"
[master a0e97e8] Init
3 files changed, 23 insertions(+)
create mode 100644 calendar.c
create mode 100644 main.c
create mode 100644 memo.c
```

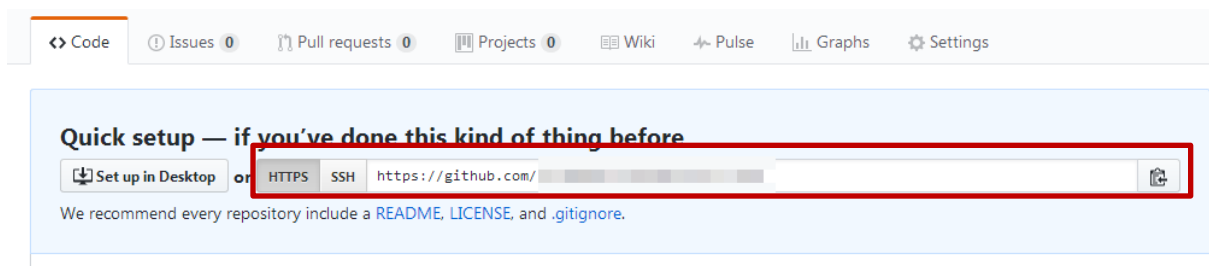
- 주로 수정을했을경우 다시 git에 add하고 commit을 해줘야하는 과정이 번거롭기때문에 **git commit -a -m “[메시지]”** 명령어를 통해 번거롭지않게 한번에 commit 및 add를 실행한다.
- 아직까지는 **Local**에서 이루어지는 작업들이다. **Commit**이 이루어진 파일에 대해서는 git을 통해 복구가 가능하다.

□ Git

■ 명령어

□ git remote add [리모트저장소이름] [URL]

- 현재 로컬 디렉토리와 온라인의 Git-repository를 연결



- 자신의 Repository저장소 URL을 복사
- 연결하고자 하는 디렉토리로 이동 한 뒤
git remote add [리모트저장소] [URL주소] 를 입력
- 처음 리모트저장소의 이름은 origin으로 설정한다.
→ git remote add origin [URL]
- Remote 저장소(네트워크상의 Git Repository)를 연결하여 다른사람과 협업이 가능하다.

Git 원격저장소 등록

□ Git

■ 명령어

□ git remote -v

- 현재 git의 연결상태를 보여주는 명령어.

```
dlcksw@dlcksw-Virtual-Machine:~/Repository$ git remote -v  
origin http://github.com/dlcksw0112/test.git (fetch)  
origin http://github.com/dlcksw0112/test.git (push)
```

Git 원격저장소에 자료 업로드

□ Git

■ 자료 업로드 명령어

□ git push [리모트저장소이름] [브랜치이름]

- 로컬 저장소의 데이터를 원격 저장소로 push (upload)한다.
- Master라는 브랜치는 최초로 생성되는 main branch이다.
- (branch에대한 설명은 후에 계속)

```
dlckswo@dlckswo-Virtual-Machine:~/Repository$ git push origin master
Username for 'https://github.com': dlckswo0112
Password for 'https://dlckswo0112@github.com':
Counting objects: 8, done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (8/8), 713 bytes | 0 bytes/s, done.
Total 8 (delta 0), reused 0 (delta 0)
To http://github.com/dlckswo0112/test.git
 * [new branch]      master -> master
```

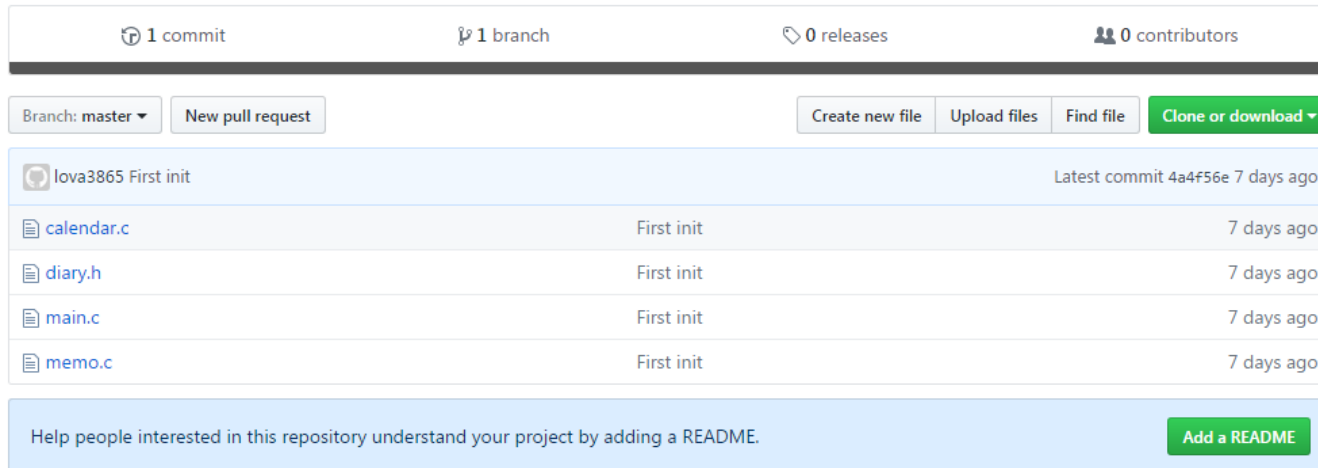
Git 원격저장소에 자료 업로드

□ Git

■ 자료 업로드 명령어

□ git push [리모트저장소이름] [브랜치이름]

- 로컬 저장소의 데이터를 원격 저장소로 push (upload)한다.
- Master라는 브랜치는 최초로 생성되는 main branch이다.
- (branch에대한 설명은 후에 계속)



1 commit 1 branch 0 releases 0 contributors

Branch: master New pull request Create new file Upload files Find file Clone or download

lova3865 First init Latest commit 4a4f56e 7 days ago

calendar.c	First init	7 days ago
diary.h	First init	7 days ago
main.c	First init	7 days ago
memo.c	First init	7 days ago

Help people interested in this repository understand your project by adding a README. Add a README

Git 원격저장소에 자료 업로드

□ Git

■ 자료 업로드 명령어

□ git push origin master

- 로컬 저장소를 원격 저장소로 push한다.
- 단, 파일을 수정하거나 새로 만들었을 경우 **git에 add** 하는작업을 해주어야한다.

```
lova@lova-virtual-machine: ~/testRepository
1 #include "diary.h"
2 //this is Calendar
3 int calendar()
4 {
5     printf("function calendar.\n");
6     return 0;
7 }
8
```

```
lova@lova-virtual-machine:~/testRepository$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   calendar.c

no changes added to commit (use "git add" and/or "git commit -a")
```

```
lova@lova-virtual-machine:~/testRepository$ git commit -a -m "modified calendar.c"
[master fc9985b] modified calendar.c
1 file changed, 1 insertion(+)
```

- **git commit -a -m** “답을 메시지” 를 넣으면 수정한 파일들이 자동으로 **commit** 된다. 이후 **git push** 동작을 수행하면 서버에 **upload**된다.
- **Git push origin master** 는 master branch에 push한다는 것.

Git 저장소에서 파일 가져오기

□ Git

■ 저장소에서 파일 가져오기

□ `git fetch [저장소] [브랜치]`

□ `git pull [저장소] [브랜치]`

- [저장소]에 존재하는 브랜치들을 로컬에 등록 및 다운로드한다.
- 저장소에 등록된 브랜치들 중 받고싶은 브랜치를 지정하면 해당 파일을 현재 폴더로 가져온다.

```
lova@lova-virtual-machine:~/Repository$ git fetch origin
remote: Counting objects: 15, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 15 (delta 2), reused 14 (delta 1), pack-reused 0
Unpacking objects: 100% (15/15), done.
From https://github.com/Lova3865/testRepository
 * [new branch]      firstBranch -> origin/firstBranch
 * [new branch]      master      -> origin/master
```

■ fetch와 pull의 차이점

- `fetch` : 네트워크 저장소의 소스를 로컬 저장소로 가져오지만 Merge는 하지않는다.
- `pull` : 네트워크 저장소의 소스를 로컬 저장소로 가져오고 현재 작업중인 소스들의 Merge 작업까지 수행한다.

□ Git

■ 명령어

□ git checkout

- 현재 디렉토리내의 파일과 로컬 저장소의 파일을 비교하여 파일의 변경/삭제/가 이루어졌는지 보여준다.
- M : Modified (변경됨)
- D : Deleted (삭제됨).

```
lova@lova-virtual-machine:~/Repository$ git checkout
M    calendar.c
lova@lova-virtual-machine:~/Repository$ rm *
lova@lova-virtual-machine:~/Repository$ git checkout
D    calendar.c
D    diary.h
D    main.c
D    memo.c
```

□ git checkout [파일명]

- 해당 파일을 로컬 저장소에 저장된 상태로 되돌린다.

□ Git

■ 명령어

□ git checkout [파일명]

- 해당 파일을 로컬 저장소에 저장된 상태로 되돌린다.

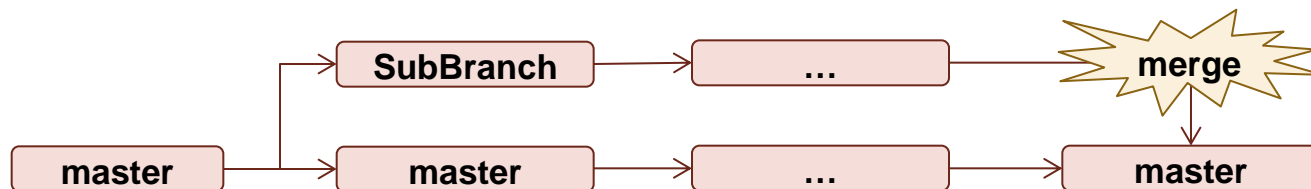
```
lova@lova-virtual-machine:~/Repository$ ll
total 12
drwxrwxr-x  3 lova lova 4096 4월 28 17:43 ./
drwxr-xr-x 27 lova lova 4096 4월 28 17:42 ../
drwxrwxr-x  8 lova lova 4096 4월 28 17:43 .git/
lova@lova-virtual-machine:~/Repository$ git checkout *
lova@lova-virtual-machine:~/Repository$ ll
total 28
drwxrwxr-x  3 lova lova 4096 4월 28 17:44 ./
drwxr-xr-x 27 lova lova 4096 4월 28 17:42 ../
-rw-rw-r--  1 lova lova   83 4월 28 17:44 calendar.c
-rw-rw-r--  1 lova lova   47 4월 28 17:44 diary.h
drwxrwxr-x  8 lova lova 4096 4월 28 17:44 .git/
-rw-rw-r--  1 lova lova   69 4월 28 17:44 main.c
-rw-rw-r--  1 lova lova   76 4월 28 17:44 memo.c
```

- → git checkout * 을 통해 checkout로 표시된 모든 파일들을 되돌림

□ Git Branch (브랜치)

■ Branch

- 작업을 분기로 저장할 수 있는것
 - 현재 프로젝트가 진행되는 상황에서 추가적인 기능을 만들 때 이 기능은 메인 기능에 영향을 주면 안된다.
 - 따라서 메인작업과 추가적인 작업을 동시에 진행하여야 할때 두 작업이 개별적으로 진행을 한다.
 - 이렇게 동일한 소스코드를 복사하여 독립적으로 작업할 수 있게 하는것이 **Branch** 라고한다.
-
- 후에 **Branch**로 작업한 작업물들은 **merge**과정을 통해 통합이 가능하다.



Git Branch

□ Git Branch (브랜치)

■ Branch 만들기

□ git branch [branchname]

- Git branch [name] 을 통해 원하는 이름으로 브랜치를 생성할수있다.

```
lova@lova-virtual-machine:~/Repository$ git branch secondBranch
lova@lova-virtual-machine:~/Repository$ git branch
* master
  secondBranch
lova@lova-virtual-machine:~/Repository$
```

- Git branch 만 입력하면 현재 내가사용하고있는 브랜치 (* 표시) 와 생성된 브랜치들의 목록을 보여준다.

□ Git Branch (브랜치)

■ Branch 전환

□ git checkout [branch이름]

```
lova@lova-virtual-machine:~/Repository$ git checkout secondBranch
Switched to branch 'secondBranch'
lova@lova-virtual-machine:~/Repository$ git branch
  master
* secondBranch
```

- git checkout 명령어 뒤 **branch이름**을 넣으면 원하는 브랜치로 **branch**가 변경된다. 이후 작업은 해당 **branch**로 이루어진다.
- git checkout -b [branch이름] 을 이용하면 **branch**의 생성과 함께 이동도 동시에 수행한다.

Git Branch

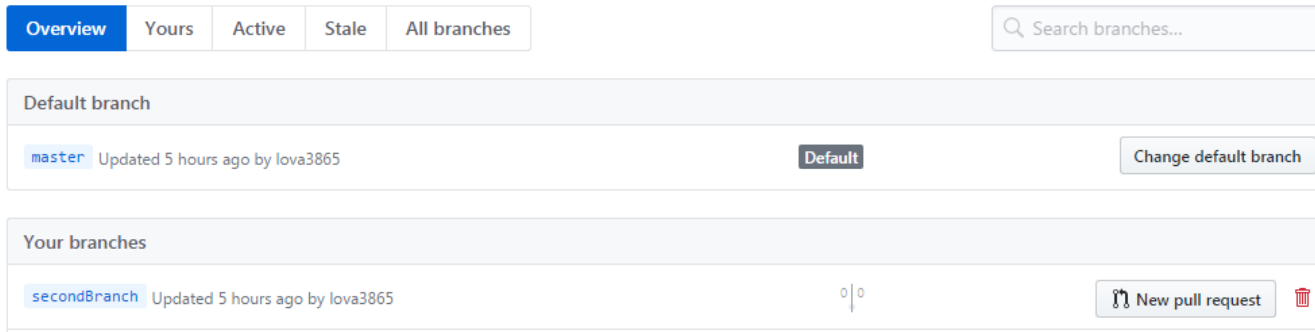
□ Git Branch (브랜치)

■ Branch

- git pull 명령어를 통해 branch를 업데이트해본다.

```
lova@lova-virtual-machine:~/Repository$ git push origin secondBranch
Username for 'https://github.com': Lova3865
Password for 'https://Lova3865@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/Lova3865/testRepository.git
 * [new branch]      secondBranch -> secondBranch
```

- Git-hub 사이트를 보면 branch가 생성된 것을 볼 수 있다.



The screenshot shows the GitHub interface for a repository. At the top, there are tabs for 'Overview' (selected), 'Yours', 'Active', 'Stale', and 'All branches'. A search bar labeled 'Search branches...' is on the right. Below the tabs, the 'Default branch' section shows 'master' as the default branch, updated 5 hours ago by lova3865. A 'Change default branch' button is next to it. The 'Your branches' section shows 'secondBranch' as a new branch, also updated 5 hours ago by lova3865. A 'New pull request' button and a trash icon are visible next to 'secondBranch'.

□ Git Branch (브랜치)

■ Branch 병합

□ git merge [commit name]

- 작업중인 branch를 원하는 branch와 함께 병합시킨다.
- 예제) 먼저 master branch 로 변경한다.
- git checkout master
- 그다음 test.txt를 생성 “Hi, I’m master branch” 를 작성한다.
- 그후 commit 및 업로드를 실행한다.

```
lova@lova-virtual-machine:~/Repository$ cat test.txt
lova@lova-virtual-machine:~/Repository$ git add test.txt
lova@lova-virtual-machine:~/Repository$ git commit -a -m "add test.txt"
[master 2a4acbe] add test.txt
1 file changed, 1 insertion(+)
create mode 100644 test.txt
lova@lova-virtual-machine:~/Repository$ git push origin master
Username for 'https://github.com': Lova3865
Password for 'https://Lova3865@github.com':
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 288 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lova3865/testRepository.git
062d758..2a4acbe master -> master
```

Git Branch

- 이전에 생성한 secondBranch로 변경한다.
(git checkout secondBranch)

```
lova@lova-virtual-machine:~/Repository$ git checkout secondBranch
Switched to branch 'secondBranch'
lova@lova-virtual-machine:~/Repository$ ll
total 28
drwxrwxr-x  3 lova lova 4096 4월 28 18:13 ./
drwxr-xr-x 27 lova lova 4096 4월 28 18:07 ../
-rw-rw-r--  1 lova lova   83 4월 28 17:44 calendar.c
-rw-rw-r--  1 lova lova   47 4월 28 17:44 diary.h
drwxrwxr-x  8 lova lova 4096 4월 28 18:13 .git/
-rw-rw-r--  1 lova lova   69 4월 28 17:44 main.c
-rw-rw-r--  1 lova lova   76 4월 28 17:44 memo.c
```

- 현재 secondBranch는 업데이트를 하지 않아 test.txt가 존재하지 않는다.
따라서 git pull 명령어를 통해 master로부터 업데이트한다.

```
lova@lova-virtual-machine:~/Repository$ git pull origin master
From https://github.com/Lova3865/testRepository
* branch          master       -> FETCH_HEAD
Updating 062d758..2a4acbe
Fast-forward
 test.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
lova@lova-virtual-machine:~/Repository$ ll
total 32
drwxrwxr-x  3 lova lova 4096 4월 28 18:13 ./
drwxr-xr-x 27 lova lova 4096 4월 28 18:07 ../
-rw-rw-r--  1 lova lova   83 4월 28 17:44 calendar.c
-rw-rw-r--  1 lova lova   47 4월 28 17:44 diary.h
drwxrwxr-x  8 lova lova 4096 4월 28 18:13 .git/
-rw-rw-r--  1 lova lova   69 4월 28 17:44 main.c
-rw-rw-r--  1 lova lova   76 4월 28 17:44 memo.c
-rw-rw-r--  1 lova lova   21 4월 28 18:13 test.txt
```

Git Branch

- vi 를 이용하여 test.txt 파일에 내용을 추가한다.

```
1 Hi i'm master branch
2
3 Hi i'm secondBranch
```

- 작성한 test.txt 를 네트워크 저장소에 업로드한다.

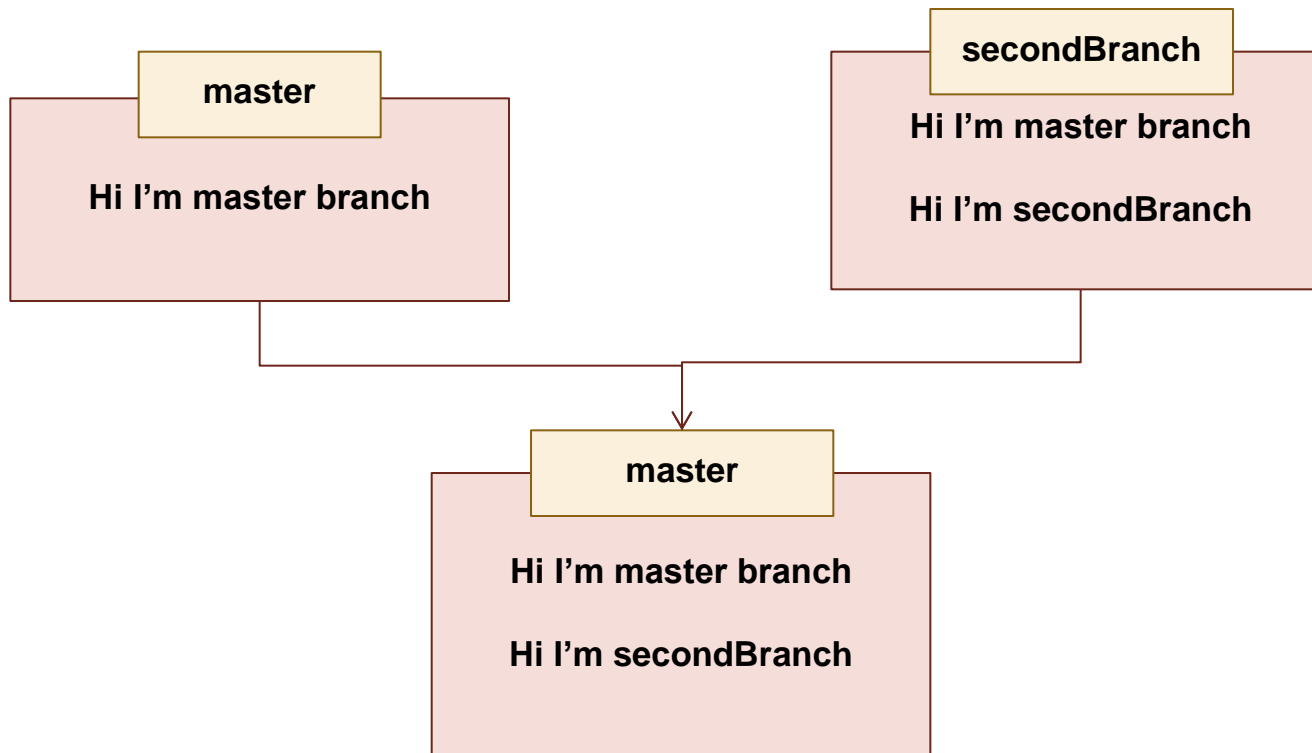
```
lova@lova-virtual-machine:~/Repository$ cat test.txt
Hi i'm master branch

Hi i'm secondBranch
lova@lova-virtual-machine:~/Repository$ git add test.txt
lova@lova-virtual-machine:~/Repository$ git commit -a -m "modified test.txt in seconBranch"
[secondBranch 857861f] modified test.txt in seconBranch
1 file changed, 2 insertions(+)
lova@lova-virtual-machine:~/Repository$ git push origin secondBranch
Username for 'https://github.com': Lova3865
Password for 'https://Lova3865@github.com':
Counting objects: 3, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 299 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/Lova3865/testRepository.git
062d758..857861f secondBranch -> secondBranch
```


Git Branch

- Master branch 로 이동한 후 secondBranch를 merge한다.

```
lova@lova-virtual-machine:~/Repository$ git checkout master
Switched to branch 'master'
lova@lova-virtual-machine:~/Repository$ git merge secondBranch
Updating 2a4acbe..857861f
Fast-forward
 test.txt | 2 ++
1 file changed, 2 insertions(+)
```

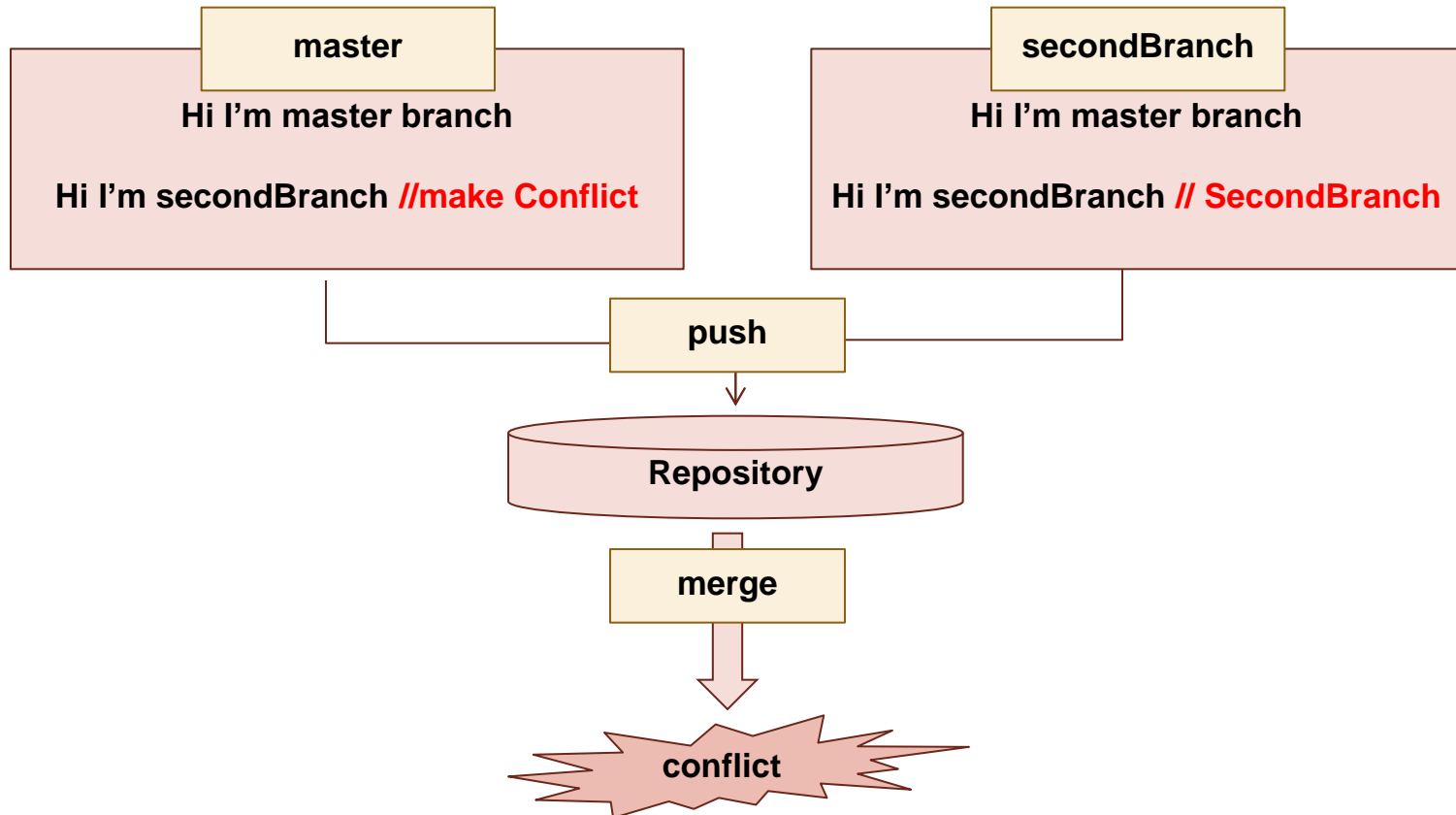


- 만약 두 사람이 같은 파일을 수정하고 업로드 후 **merge**를 했는데 수정한 부분이 다르다면 **git**이 알아서 관리해줌
- 만약 같은 파일의 같은 부분을 수정할 경우 소스 충돌이 발생함. 이 경우 **git**는 어떤 부분에서 소스충돌이 발생했는지에 대해서 개발자에게 알리고 개발자가 결정하게 함
- 충돌이 발생한 소스를 수정한 후 다시 **git commit** 명령으로 반영할 경우 충돌 상황은 종료

Git Branch

□ Conflict (충돌)

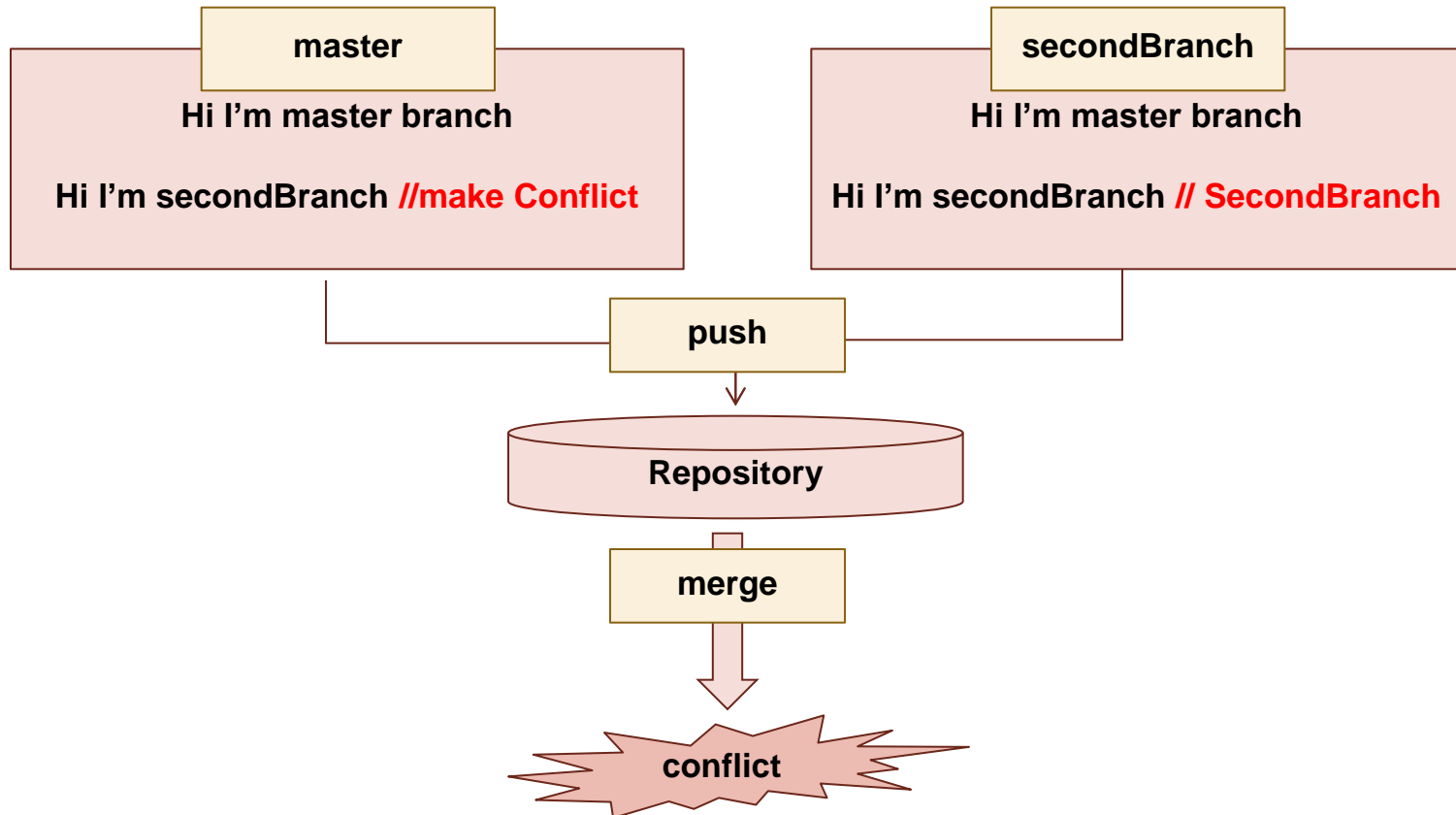
- 만약 같은소스코드의 같은 부분을 수정하게된다면 merge 단계에서 충돌이 발생하게 된다.



Git Branch

□ Conflict (충돌)

- 만약 같은소스코드의 같은 부분을 수정하게된다면 merge 단계에서 충돌이 발생하게 된다.



□ Conflict (충돌)

- 만약 같은소스코드의 같은 부분을 수정하게된다면 **merge** 단계에서 충돌이 발생하게 된다.
- Master branch
 - vi test.txt 를 통해 수정
 - git commit -a -m "Modify test.txt in master branch" 으로 commit
 - git push origin master 를 통해 저장소에저장
- git checkout secondBranch 로 secondBranch로 branch 변경
- secondBranch Branch
 - git pull origin secondBranch 로 저장소의 secondBranch 파일 다운로드
 - vi test.txt 를 통해 수정
 - Git commit -a -m "Modify test.txt in secondBranch" commit
 - Git push origin secondBranch 를 통해 저장소에 저장

Git Branch

□ Conflict (충돌)

- Git checkout master 로 master branch 로 변경 후 secondBranch와 merge

- git merge secondBranch

```
lova@jeff-700-212kr:~/Repository$ git checkout master
Switched to branch 'master'
lova@jeff-700-212kr:~/Repository$ git merge secondBranch
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- 동일한부분에 대한 수정이 이루어졌기 때문에 merge Fail.
- vi 를 통해 test.txt를 열어보면

```
1 Hi i'm master branch
2
3 <<<<<<< HEAD
4 Hi i'm secondBranch // make Conflict
5 =====
6 Hi i'm secondBranch // SecondBranch Modify
7
8 >>>>>>> secondBranch
```

- 위와 같이 <<<<HEAD (master 부분) 의 수정사항과
- =====
- >>>>secondBranch 에서의 수정사항을 보여줌.

□ Conflict (충돌)

- 이러한 경우 직접 파일을 열어 수동으로 수정한 뒤 master branch에 업데이트해주면 된다.

```
1 Hi i'm master branch
2
3
4 Hi i'm secondBranch // merge modify
5 █
```

- 수정후 add한 뒤 commit을 해주면 된다.

□ Git Branch (브랜치)

■ Branch 삭제

□ `git branch -d [branch name]`

- 더 이상 `branch`를 사용하지 않을 때 삭제한다.
- 삭제 시 `-d` 옵션을 통해 삭제할 수 있다.

□ Tag 만들기

■ Git tag [version number] [identifier]

- 자신이 만든 소스의 버전을 등록할 때 사용
- identifier는 git log 명령어를 통해 알 수 있다.

```
lova@jeff-700-212kr:~/Repository$ git log
commit 73016258d9589fb845ec186f1dfd64c75983feab
Merge: 87aa397 c8e911b
Author: lova <coffee90506@gmail.com>
Date: Sat Apr 29 17:13:41 2017 +0900
```

- Git tag 1.0.0 [identifier] 를 통해 1.0.0 이라는 태그를 부여.

```
lova@jeff-700-212kr:~/Repository$ git tag 1.0.0 73016258d9589fb845ec186f1dfd64c75983feab
lova@jeff-700-212kr:~/Repository$ git tag
1.0.0
```

- 태그를 통해 보다 편리하게 이전시점으로 되돌릴 수 있다.

□ 유용한 **Git** 명령어

■ Git 의 내장 gui → gitk

- Sudo apt-get install gitk 를 통해 gitk를 설치하면 GUI형태의 git을 사용할 수 있다.

■ 콘솔에서의 git 출력문을 색을 입혀서 출력

- git config color.ui true

■ Git log 명령어에서 log의 수를 1개만 출력

- git config format.pretty oneline

■ 파일을 추가할때 대화형으로 추가

- git add -i