

4

Git (2) VCS (Version Control System)

이력 수정

- 최종 완료한 commit 내역 수정
 - Commit 후 Staging Area에 추가한 파일을 다시 포함하여 commit
 - Commit 메시지 수정

```
walab-HGU:~/hello:> git commit -m 'commit message'  
walab-HGU:~/hello:> git add new_file  
walab-HGU:~/hello:> git commit --amend
```

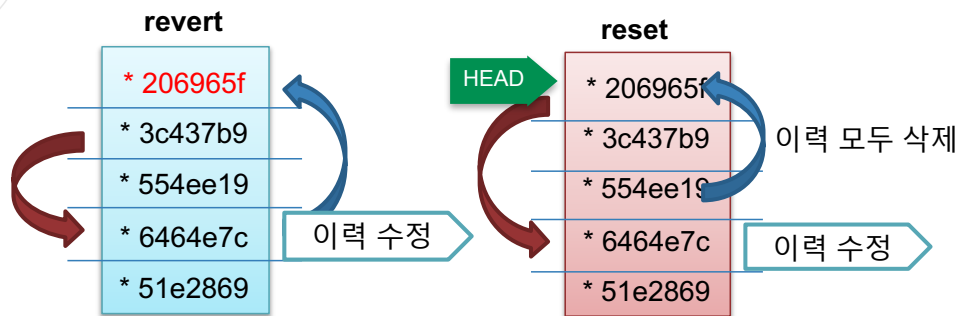
- WD에서 수정한 파일(modified)을 수정되기 전 상태(unmodified)로 변경

```
walab-HGU:~/hello:> git checkout <file1>
```

- git reset, git revert

revert vs reset

- 특정 커밋으로 되돌아 갈 수 있음. 취소 효과



reset

- Commit 취소
- 특정 커밋으로 돌아감으로 취소 작업
- 최근 커밋부터 특정 커밋 이후의 버전들을 히스토리에서 삭제

```
$ git reset [option] commit_checksum
```

- git add 취소 (Staging Area => WD 로 이동)

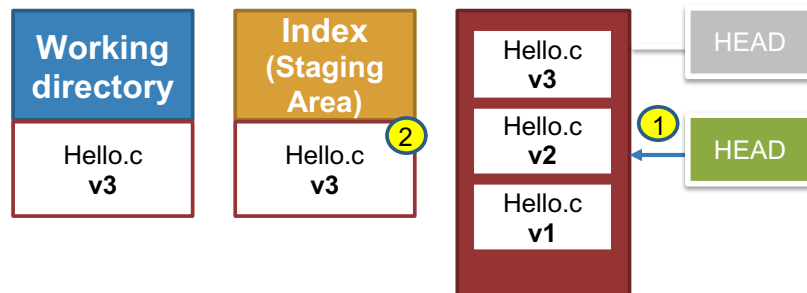
```
$ git reset HEAD <file명>
```

- Option
 - --soft
 - --mixed
 - --hard

git reset --soft

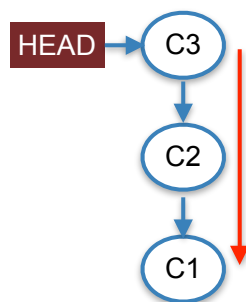
- HEAD를 특정 커밋으로 이동
- WD파일 보존, 해당 파일은 staged로 이동
- Commit 하면 원래 상태로 복원 가능

```
$ git reset --soft HEAD~
```



5

git reset --soft



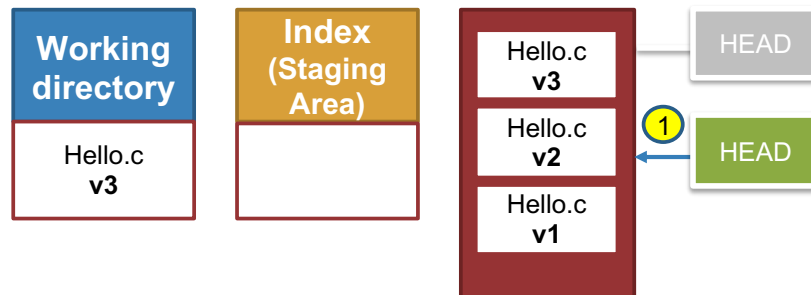
```
walab-HGU:~/lab7> touch test.txt
walab-HGU:~/lab7> git add test.txt
walab-HGU:~/lab7> git commit -m "C1"
walab-HGU:~/lab7> vim test.txt
walab-HGU:~/lab7> git commit -am "C2"
walab-HGU:~/lab7> vim test.txt
walab-HGU:~/lab7> git commit -am "C3"
walab-HGU:~/lab7> git log --oneline
dcf2782 (HEAD -> master) C3
edc8683 C2
a977e1c C1
walab-HGU:~/lab7> git reset --soft a977e1c
walab-HGU:~/lab7> git log --oneline
a977e1c (HEAD -> master) C1
walab-HGU:~/lab7> git status
```

6

git reset --mixed

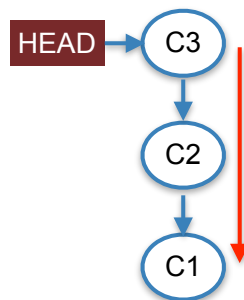
- Default 옵션
- WD파일 보존, 해당 파일 unstaged, HEAD 이동

```
$ git reset --mixed HEAD~  
$ git reset HEAD~
```



7

git reset --mixed



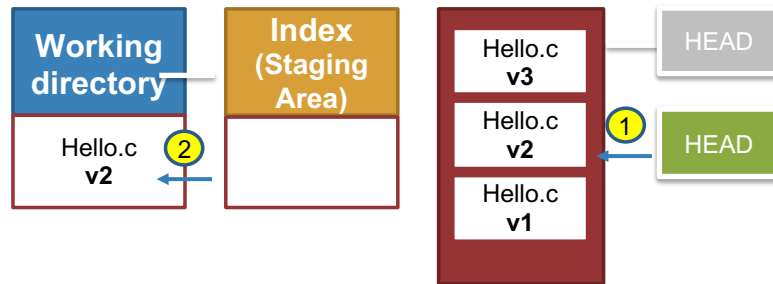
```
walab-HGU:~/lab7> git commit -m "C2"  
walab-HGU:~/lab7> vim test.txt  
walab-HGU:~/lab7> git commit -am "C3"  
walab-HGU:~/lab7> git log --oneline  
408214c (HEAD -> master) C3  
6e793c0 C2  
a977e1c C1  
walab-HGU:~/lab7> git reset --mixed a977e1c  
walab-HGU:~/lab7> git status  
walab-HGU:~/lab7> git log --oneline  
a977e1c (HEAD -> master) C1
```

8

git reset --hard

- WD파일 변경사항 삭제, 해당 파일 unstaged, HEAD 이동

```
$ git reset --hard HEAD~
```

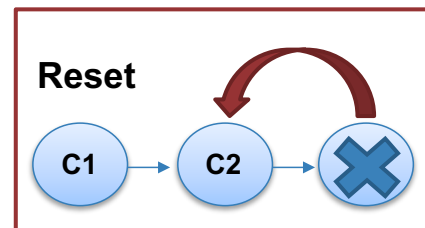
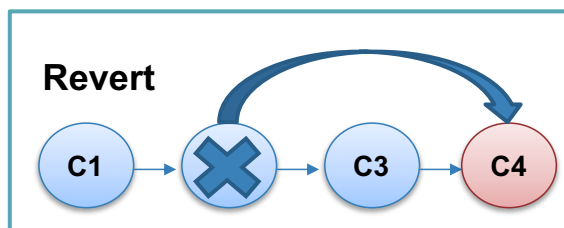


9

git revert

- commit된 스냅샷을 취소하는 명령
- 커밋 이력에서 취소하기 원하는 커밋에 의해 변경된 내용을 취소하기 위한 방법을 찾고 그 결과를 새로운 커밋으로 추가함
- 현재 커밋 이력을 삭제하지 않음

```
$ git revert HEAD~
```



10

git rm

- Untracked files

```
$ rm sample.txt
```

- Tracked files

- git 저장소 + 로컬디렉터리(WD) 모두 삭제

```
$ git rm sample.txt
```

- git 저장소 삭제, 로컬디렉터리(WD) 삭제하지 않음

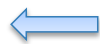
```
$ git rm --cached sample.txt
```

11

git clone

- 원격저장소에서 로컬 저장소로 복제

Local Repo



Remote Repo

```
$ git clone <remote repository URL>  
$ git clone <remote repository URL> <new folder>  
$ git clone -b <branchname> <remote repository URL>
```

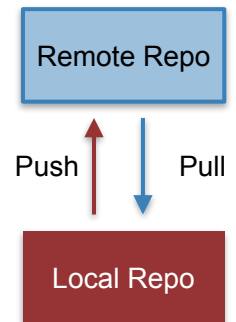
```
walab-HGU:~:> git clone -b gh-pages https://github.com/ahfarmer/calculator.git  
walab-HGU:~/calculator:> git branch -a  
walab-HGU:~:> git clone https://github.com/ahfarmer/calculator.git  
walab-HGU:~/calculator:> git branch -a
```

12

Remote repo

- 로컬저장소에 원격 저장소를 연결하여 소스를 push 하거나 pull 할 수 있음
- 로컬저장소에 원격저장소 연결/삭제/ 정보보기

```
$ git remote
$ git remote -v
$ git remote -h
$ git remote add origin <remote repository URL>
$ git remote remove origin
$ git remote add calculator <remote repository URL>
```



13

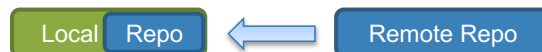
Remote repo

- Push



```
$ git push origin master
```

- Pull



```
$ git pull origin master
```

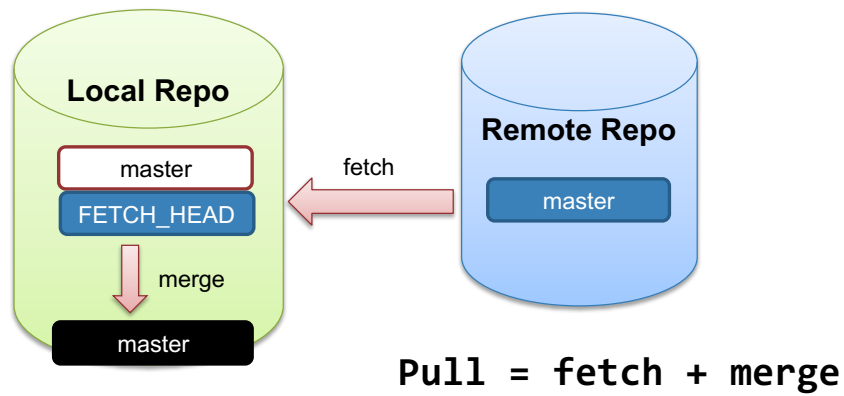
- fetch



```
$ git fetch
```

14

Pull vs fetch

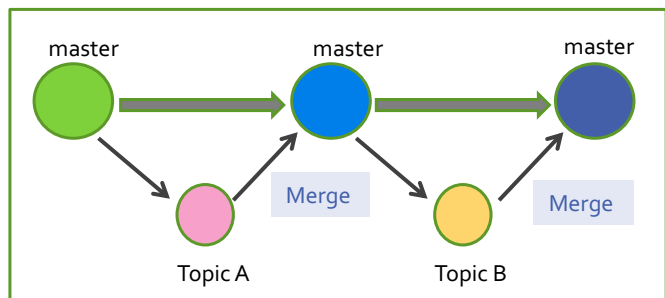


15

Branch

- 기본 브랜치 : master branch
- 새로운 작업이 발생할 때 브랜치를 생성하여 작업
- 브랜치에서 작업이 완료되면 변경내용 및 이력을 master branch로 병합(merge)

```
$ git branch <new branch>  
$ git branch -h  
$ git branch -a  
$ git branch  
$ git branch -d <branchname>  
$ git branch -D <branchname>
```



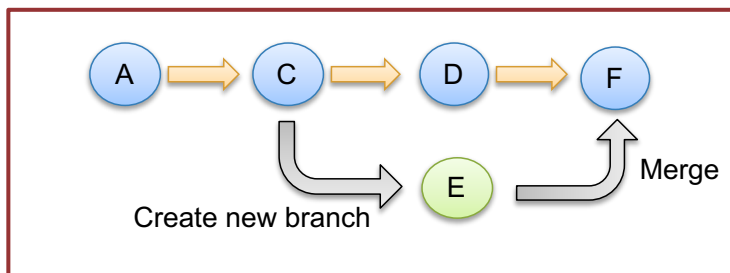
16

Checkout

- 다른 브랜치 전환할 때 사용

```
$ git checkout <branchname>  
$ git checkout -b <new branch>
```

```
walab-HGU:~:> git branch  
master  
test  
* test2
```

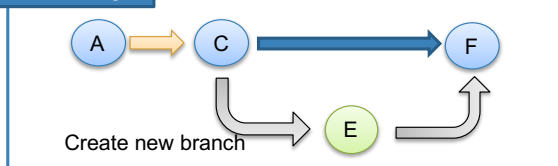


17

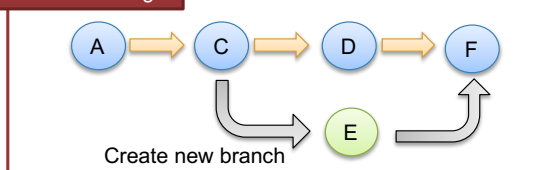
git merge

- 브랜치와 브랜치(master)에 병합하는 작업

Fast-forward merge



No fast-forward merge



```
$ mkdir git1  
$ cd git1  
$ git init  
$ touch hello.c  
$ git add hello.c  
$ git commit -m "A"  
$ vim hello.c  
$ git commit -am "C"  
$ git checkout -b new  
$ vim hello.c  
$ git commit -am "E"  
$ git checkout master  
$ git merge new
```

18

5

Git (3) VCS (Version Control System)

git stash

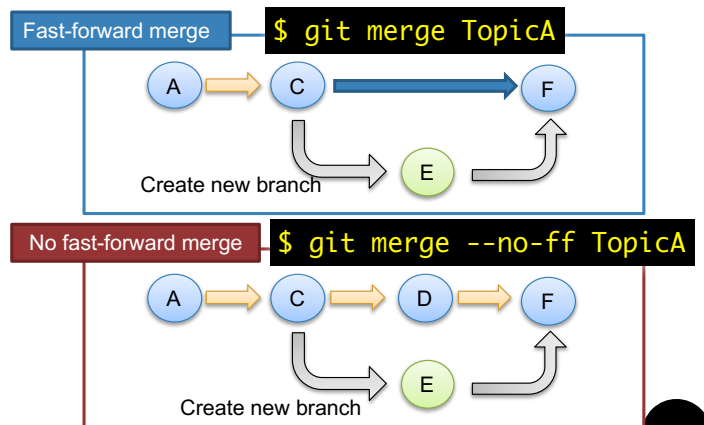
- 현재 작업 내용을 임시 저장하여 보관하는 장소
- WD(Working Directory)에서 수정한 파일만 저장
 - Modified면서 Tracked 상태인 파일
 - Staging Area에 있는 파일

```
$ git stash
$ git stash save
$ git stash list
$ git stash apply [stash이름]
$ git stash pop // apply + drop
$ git stash drop
$ git stash clear
```

```
$ git commit -m "create fruit.txt"
$ vim fruit.txt
$ git stash
$ git stash list
$ cat fruit.txt
$ git stash pop
$ cat fruit.txt
```

git merge

- 브랜치와 브랜치(master)에 병합하는 작업
- Fast forward(--ff) **default**
 - Merge commit 생성 X
- No Fast Forward(--no-ff)
 - Merge commit 생성 O
- --squash
 - Merge 후 파일 상태로 변경
 - Merge commit 생성 X
 - 별도 commit을 해야 함



21

git merge --ff

```
$ mkdir git2
$ cd git2
$ git init
$ touch fruit.txt
$ git add fruit.txt
$ git commit -m "C1"
$ vim fruit.txt
$ git commit -am "C2"
$ git checkout -b TopicA
$ vim fruit.txt
$ git commit -am "C3"
$ git checkout -
$ git merge TopicA
$ git log --oneline --graph
```

```
walab-HGU:~/lab7/git2:> git log --oneline --graph
* dbb65d1 (HEAD -> master, TopicA) C3
* 825c2a1 C2
* 0b2b6d3 C1
```

22

git merge --no-ff

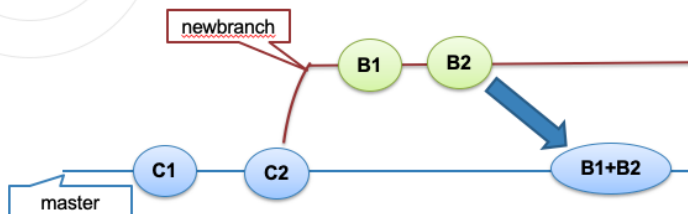
```
$ vim fruit.txt
$ git commit -am "C4"
$ git checkout -b TopicB
$ git branch
$ vim fruit.txt
$ git commit -am "C5"
$ git checkout -
$ git merge --no-ff TopicB
$ git log --oneline --graph
```

```
walab-HGU:~/lab7/git2:> git log --oneline --graph
* c8997d2 (HEAD -> master) Merge branch 'TopicB'
| \
| * 69c0440 (TopicB) C5
| /
* 7f9daf0 C4
* dbb65d1 (TopicA) C3
* 825c2a1 C2
* 0b2b6d3 C1
```

23

git merge --squash

- 다른 브랜치에 있는 여러 커밋을 하나로 병합
- Merge commit이 만들어지지 않음
- 별도로 Commit을 해야 함

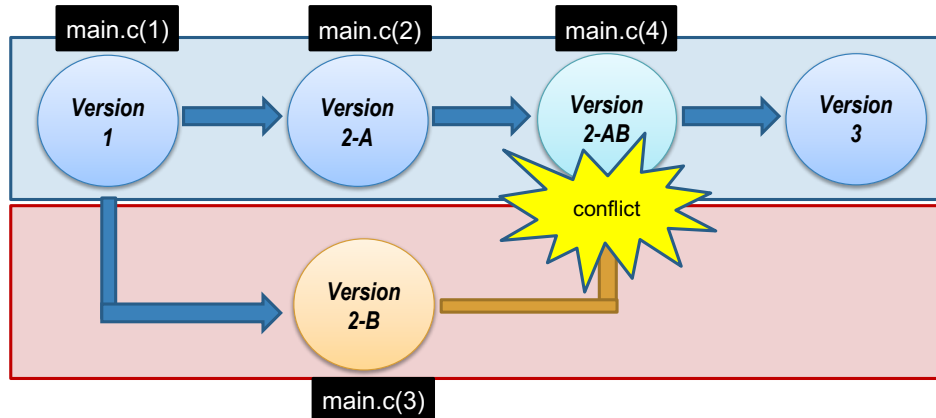


```
$ mkdir git4
$ cd git4
$ ls
$ git init
$ touch fruit.txt
$ git add fruit.txt
$ git commit -m "C1"
$ vim fruit.txt
$ git commit -am "C2"
$ git log --oneline
$ git checkout -b TopicA
$ vi fruit.txt
$ git commit -am "B1"
$ git log --oneline
$ vi fruit.txt
$ git commit -am "B2"
$ git checkout -
$ git merge --squash TopicA
$ git log --oneline
$ git commit -am "B1+B2"
$ cat fruit.txt
```

24

merge conflict

- 두 브랜치에서 같은 파일을 수정하여 각각 커밋한 후, 병합하는 경우 충돌 발생



25

merge conflict

- 충돌 메시지 발생

```
walab-HGU:~/lab7/git3:> git merge TopicA
Auto-merging main.c
CONFLICT (content): Merge conflict in main.c
Automatic merge failed; fix conflicts and then commit the result.
```

- 충돌 발생한 파일

```
<<<<<<< HEAD
V1 : This is some content for merging
V2-A: content to append
=====
V2 : totally different content to merge later
>>>>>> new_branch
```

26

merge conflict

- 충돌 해결방법
 - 파일의 내용을 적절히 수정
 - git add
 - git commit

```
#include <stdio.h>
<<<<<<< HEAD
int minus(int n1, int n2){
    return n1-n2;
=====

int plus(int n1, int n2){
    return n1+n2;
>>>>>>> TopicA
}

int main(){
    int num1, num2;
    printf("두 수를 입력:");
    scanf("%d %d", &num1, &num2);
    return 0;
}
```

27

merge conflict

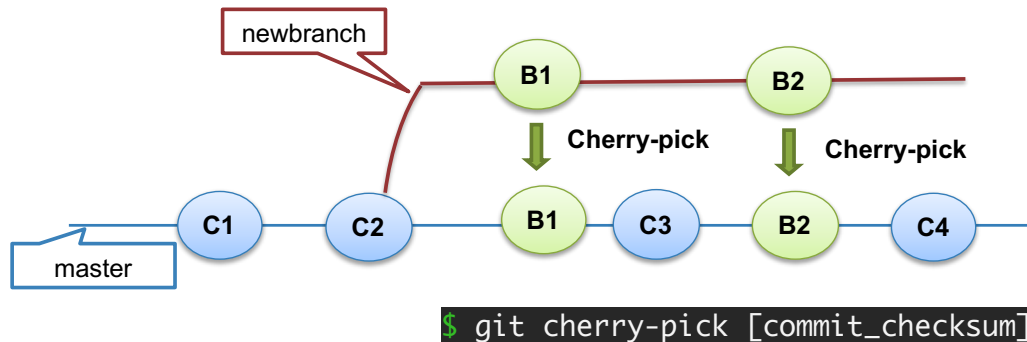
Master branch	TopicA
main.c 파일 생성	
두 수 입력 추가	
	브랜치 생성
뺄셈함수 구현	덧셈함수구현
TopicA 병합	
충돌	
충돌해결	

```
$ mkdir git3
$ cd git3
$ git init
$ touch main.c
$ git add main.c
$ git commit -m "create main.c"
$ vim main.c
$ git commit -am "add 두수입력"
$ git log --oneline
$ git checkout -b TopicA
$ vi main.c
$ git commit -am "add 덧셈함수"
$ git log --oneline
$ git checkout -
$ vi main.c
$ git commit -am "add minus func."
$ git log --oneline
$ git merge TopicA
$ git status
$ vim main.c
$ git commit -am "add 커밋해결"
$ git log --oneline --graph
```

28

Cherry-pick

- 다른 브랜치의 특정 커밋을 선택하여 현재 브랜치에 적용하는 명령어
- Git merge를 통한 충돌을 피할 수 있음



29

Reflog

- 모든 참조 기록을 확인

```
walab-HGU:~/lab7/git4:> git reflog
0890f98 (HEAD -> master) HEAD@{11}: commit: B1+B2
e7b1a91 HEAD@{12}: checkout: moving from TopicA to master
6015707 HEAD@{13}: commit: B2
9efdfb9 HEAD@{14}: commit: B1
e7b1a91 HEAD@{15}: checkout: moving from master to TopicA
e7b1a91 HEAD@{16}: commit: C2
e5da715 HEAD@{17}: commit (initial): C1
```

- 원하는 커밋으로 이동(취소)

```
walab-HGU:~/lab7/git4:> git reflog
walab-HGU:~/lab7/git4:> git reset --hard [commit id]
```

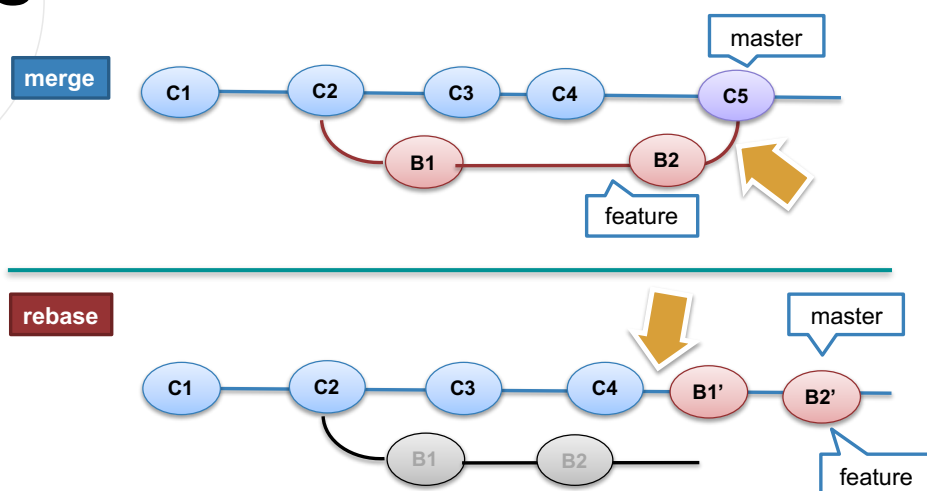
30

Rebase

- merge와는 다른 형태의 병합
- 새 브랜치의 Base를 master의 마지막 commit으로 rebase
 - [새브랜치]로 checkout
 - [master]로 rebase
 - [master]로 checkout
 - [master]브랜치에서 [새브랜치]를 병합 (fast-forward merge)

31

Merge vs Rebase



32

Rebase -i

- commit 히스토리를 편집할 때 사용

```
walab-HGU:~/lab7/git4:> git rebase -i [수정커밋의 이전커밋]
```

- 수정커밋 ~ 현재커밋(HEAD) 범위에 있는 모든 commit 리스트 출력
- 명령어

```
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
```

33

Rebase -i

- Commit history 변경하기

```
walab-HGU:~/lab7/git5:> git log --oneline
1d7b0d6 (HEAD -> master) C4
a3e7438 C3
353b47c C2
ac22061 C1
```



```
walab-HGU:~/lab7/git5:> git log --oneline
35d02e1 (HEAD -> master) C3
dfaec5e C2(Reword)
ac22061 C1
```

34

참고 :

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



누구나 쉽게 이해할 수 있는 Git 입문

버전 관리를 완벽하게 이용해보자

입문 편

발전 편

찾아보기

트윗

좋아요 392개

목차

누구나 쉽게 이해할 수 있는 Git 입문

버전 관리를 완벽하게 이용해보자!

누구나 쉽게 이해할 수 있는 Git 에 입문하신 것을 환영합니다. 지금부터 Git을 사용한 버전 관리 기능을 함께 공부해 보자구요!!! 총 3가지의 코스가 준비되어 있습니다. Git 초보자 분들은 '입문편'부터 시작해주세요. Git을 사용한 적이 있으신 분은 '발전편'을 추천 합니다. '어? 뭐였지...?' 싶을 때는 '찾아보기'를 확인하세요.



6

Github

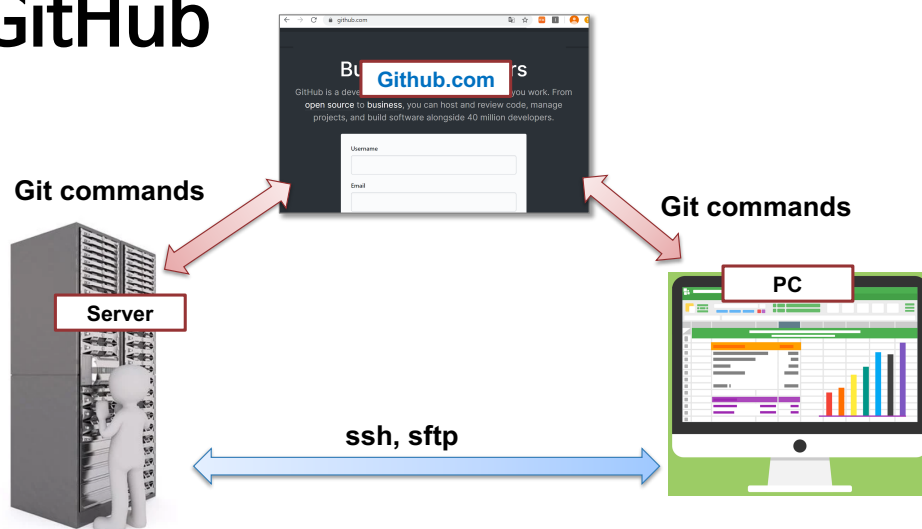
Git/Github review

- Git/Github의 커밋 사용자 이름과 이메일 설정
- Git 설치하기 (Windows / Mac / Linux)
- github에서 새 저장소 생성하는 방법 (Remote Repo)
- Github 원격 저장소와 로컬 저장소 연동 방법
- 로컬 저장소 초기화
- 원격 저장소 복제하는 방법
- 로컬 저장소 삭제하는 방법 (복구불가)
- 원격 저장소 삭제하는 방법 (복구불가)



37

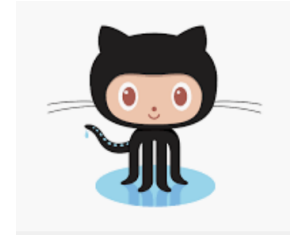
Git & GitHub



Github



- 분산 버전 관리 툴인 Git 사용 프로젝트를 지원하는 웹 호스팅 서비스
- 현재 가장 인기 있는 소스 코드 호스팅 서비스 및 소셜 코딩 플랫폼
- Public Repo / Private Repo 모두 무료
- 다양한 오픈 소스 프로젝트 소스 제공
- 협업 기능 지원
- 2018년에 Microsoft 가 인수
- Github외에도 GitLab, BitBucket 등이 있음



39

Github 특징



- 원격 저장소 제공
- Organization을 생성하여 팀단위로 저장소를 만들 수 있음
- Github page 제공 - 정적웹사이트 호스팅 서비스 제공
 - html, css, javascript, markdown로 개발, 다수 무료 theme 제공
 - <https://pages.github.com/>
- Gist : Code snippet 공유서비스 (<https://gist.github.com/>)
- Github Desktop : Window, MacOS에서 사용하는 Github전용 프로그램
- Github mobile (Android, iOS 제공)
- Gitstar Ranking (<https://gitstar-ranking.com/>)

40

Remote Repo.



Overview

Repositories 52

Projects



Public

Anyone on the internet can see this repository. You choose who can commit.

Private

You choose who can see and commit to this repository.

Create a new repository

A repository contains all project files, including the revision history. [Import a repository.](#)

Owner *

jerry10004

Repository name *

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ Add a README file

This is where you can write a long description for your project. [Learn more.](#)

☐ Add .gitignore

Choose which files not to track from a list of templates. [Learn more.](#)

☐ Choose a license

A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

41

Push : local -> remote repo.



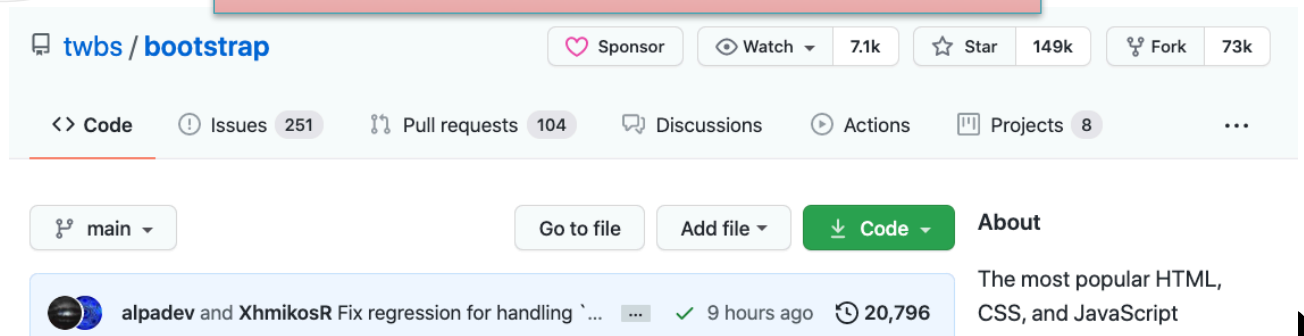
- Remote Repo가 Empty repository인 경우
 - Local Repo에서 버전 기록
 - Local repo에서 remote repo로 바로 전송가능
 - **git push origin master**
- Remote Repo에 한 개 이상 파일과 버전 기록이 있는 경우
 - Local Repo에서 버전 기록
 - Local repo에서 remote repo로 바로 전송 불가능
 - **git push origin master => 오류발생**
 - **git pull origin main**
 - **git push origin main**

42

Fork / Pull Request



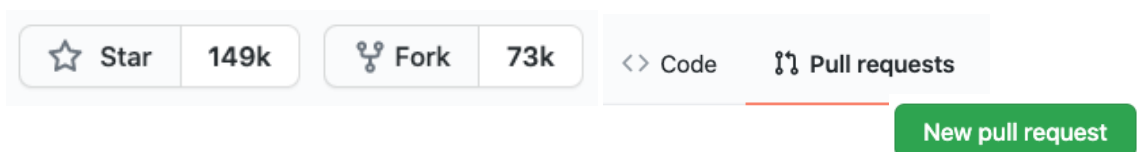
“회사 기술블로그 Repo 포크해서 후기 작성하시고,
저한테 PR 보내주세요”



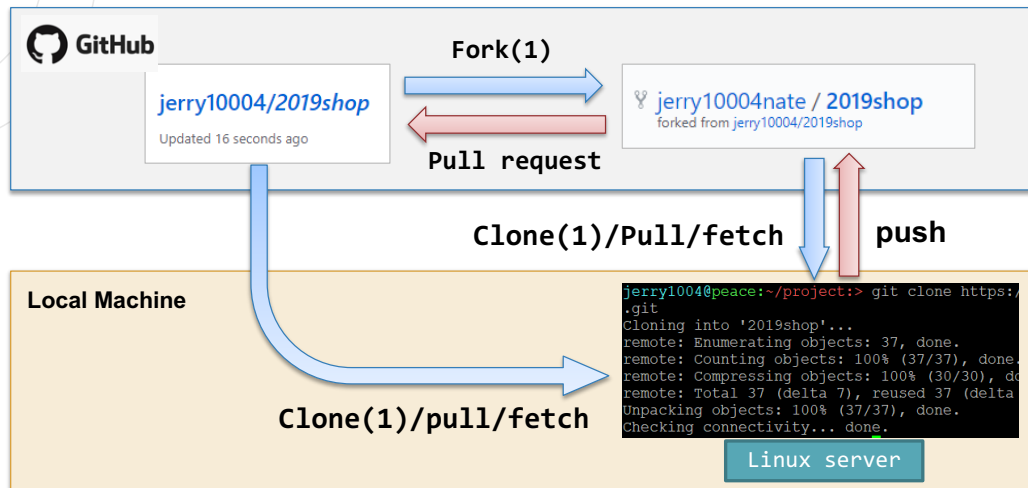
Fork / Pull Request



- Fork
 - 하나의 소프트웨어 프로젝트를 복사하여 새로운 소프트웨어로 개발
 - 레드햇 리눅스 포크 -> CentOS 개발
 - MySQL 포크 -> MariaDB 개발
- PR(Pull Request)
 - 현재 진행중인 Remote repository에 기여하기 위하여 원본 Remote repository에 변경내용(파일 및 커밋이력)의 반영을 요청하는 과정



Fork / Pull Request



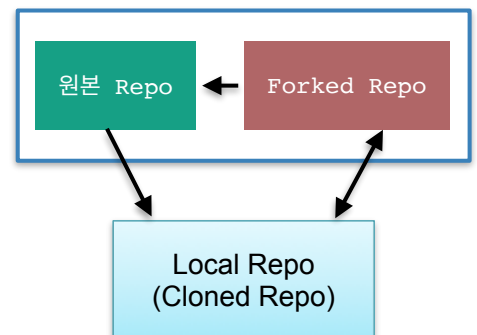
45

새 프로젝트 개설하기

- 새 프로젝트 개설을 위해 Repo 생성
 - README.md 파일 포함
- contributor로부터 온 PR 확인 및 처리
- 원본 Repo를 local Repo에 동기화하여 작업
 - Repo 충돌 발생시 해결




Github

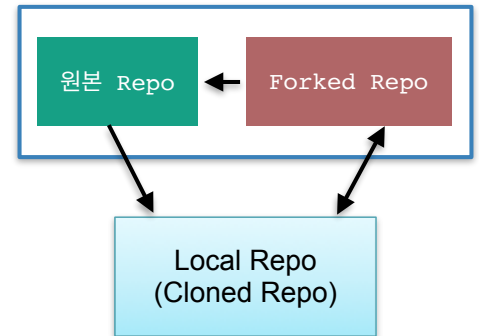
Github



46

프로젝트에 참여하기(Contributor)

- repository 를 선택 및 fork 
- Forked Repo를 clone : local
 - `git clone https://PAT@github repo주소`
- Remote 추가 : 원본 Repo
- Branch 생성 : local
- 새 파일 생성, add, commit
- 새로운 브랜치 main 브랜치에 merge
- 원본 Repo에서 pull 진행
- Forked Repo에 push 
- PR생성(Forked Repo => 원본 Repo) 










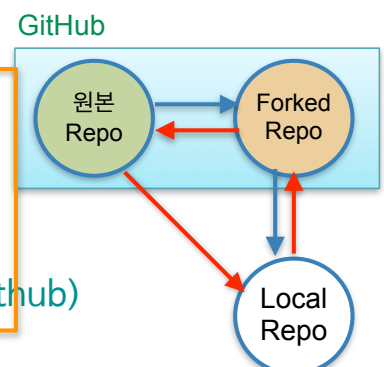
PAT : Personal Access Token

47

Contributor 활동



- 참여하기 원하는 GitHub Repository 선택(안내문이 있다면 확인)
- Remote Repository Fork ()
- Forked Repo를 clone ()
- 파일 수정 -> `git add` > `git commit` ()
- Local Repo에 원본 Remote Repo 연결 ()
- 원본 Remote Repo에서 pull ()
- Forked Remote Repo에 push ()
- Forked Remote Repo에서 원본 Remote Repo로 PR()

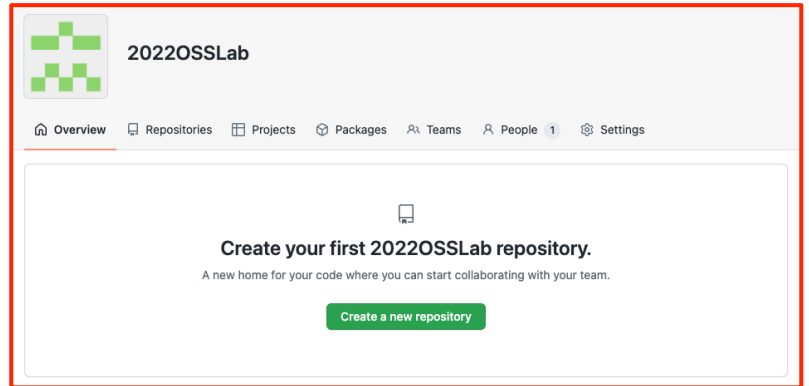


48

Organization



- 팀으로 혹은 조직으로 프로젝트 진행할 때 사용
- github에서 New organization 으로 생성 (free/team/enterprise)
- Organization 설정
 - Account name
 - Contact email
 - Members



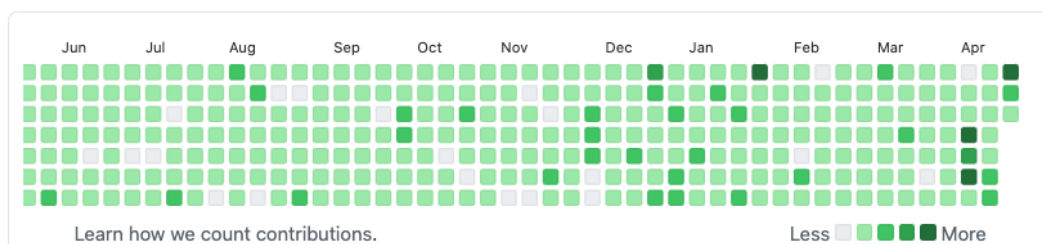
잔디심기



- github에 commit, issue, PR 등 활동 내역이 존재할 때 잔디가 심어짐
- 개발자는 1일 1잔디 심기 목표로(1 commit /day)

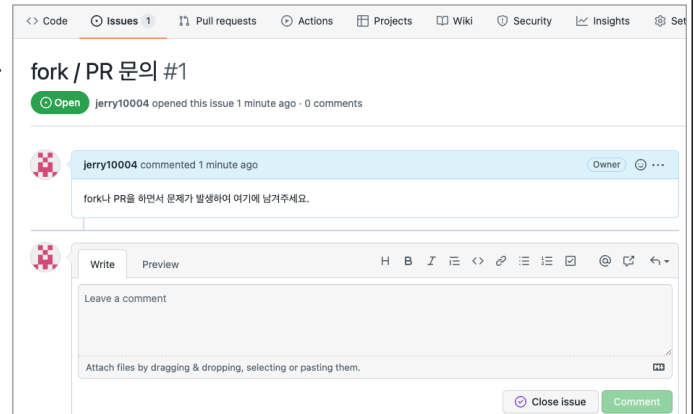
Gitstar Ranking

3,507 contributions in the last year



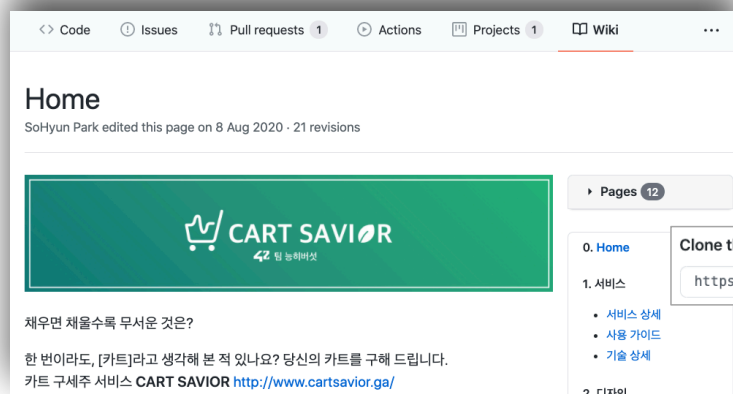
ISSUES

- 프로젝트를 진행하면서 문제점에 대한 의논, 건의 등의 이슈가 필요할 때 사용
- 모든 개발자는 이슈를 확인하며 현재 상황을 확인할 수 있으며 의견 조율 가능
- 새로운 이슈가 발생할 때 생성(Open)하며, 해당 이슈 해결시 Close



WIKI

- 다수가 협업을 통해 내용과 구조를 수정할 수 있는 웹사이트
- 오픈소스 프로젝트에 대한 내용, 구조, 매뉴얼 등으로 구성함
- Markdown으로 제작



Markdown

- 텍스트 기반 마크업 언어 (*.md)
- Html, ppt 등 다양한 형태로 변환 가능
- 매우 간단한 구조의 문법 사용하고 관리가 쉬움
- 제목, 강조, 목록, 이미지, 링크, 코드, 표, 인용문, html 문법 사용 가능
- git을 사용한 버전관리
- 다양한 에디터가 있음

typora



StackEdit

In-browser Markdown editor

Visual Studio Code

```
# 제목 1단계
## 제목 2단계
### 제목 3단계
#### 제목 4단계
##### 제목 5단계
##### 제목 6단계
```

제목 1단계

제목 2단계

제목 3단계

제목 4단계

제목 5단계

제목 6단계

53

Markdown

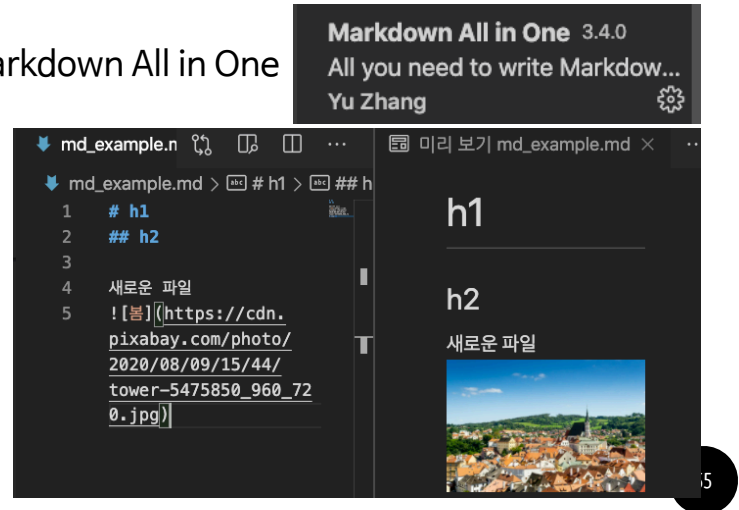
- Headers
- Lists
- Font styles
- Blockquote
- Inline code
- Code block
- Link
- Horizontal line
- Image
- Table
- Emoji <https://kr.piliapp.com/twitter-symbols/>
 - Ctrl + cmd + space (Mac)
 - 윈도우키 + . (Windows)

Markdown 실습

54

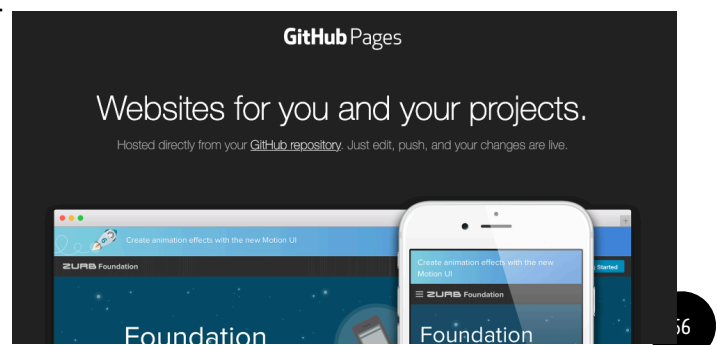
Vscode + GitHub + markdown

- Git 설치
- Markdown Extension 설치 : Markdown All in One
- 미리보기를 통해 확인
- 본인 Remote Repository 선택
- git clone(F1)
- 새 파일, add, commit
- Push
- GitHub 확인



Github page

- 정적 웹사이트 호스팅 서비스 제공
- Github계정에 Repo를 생성하여 웹 콘텐츠를 관리함
- Markdown을 사용하여 웹페이지 구성
- 다양한 Jekyll theme 사용할 수 있음



Github 블로그 제작

- github에 새로운 repo 생성
 - Repo 이름 : github_yourname.github.io
 - Add a README file 체크
- 블로그 생성 확인 : Settings > Pages >

✓ Your site is published at <https://jerry10004.github.io/>

- 웹브라우저에서 URL 확인

< → ↻ jerry10004.github.io

jerry10004.github.io

- index.html 페이지 추가

57

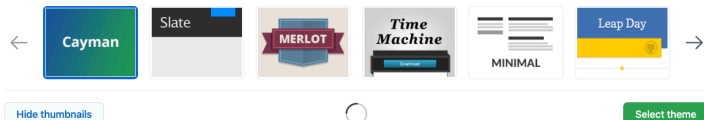
Github 블로그 제작 - theme

- Github에서 theme 선택 : Settings > Pages

Theme Chooser

Select a theme to publish your site with a Jekyll theme. [Learn more.](#)

Choose a theme



Cayman theme

Cayman is a clean, responsive theme for GitHub Pages.

[View on GitHub](#)

[Download .zip](#)

[Download .tar.gz](#)

58

Github 블로그 제작 : jekyll theme

- Jekyll
 - 간단한 정적 웹사이트를 생성기
 - <http://jekyllthemes.org/>
 - Ruby & Jekyll 설치
 - 테마선택 및 clone
 - 웹페이지 작성
 - 웹서버(Github repo)에 push

