




Git


👤 생성자	👤 이우주
🕒 생성 일시	@2023년 8월 11일 오전 4:17
🏷 태그	Study

Git의 공식 문서

Documentation

The entire Pro Git book written by Scott Chacon and Ben Straub is available to read online for free. Dead tree versions are available on Amazon.com.

 <https://git-scm.com/doc>

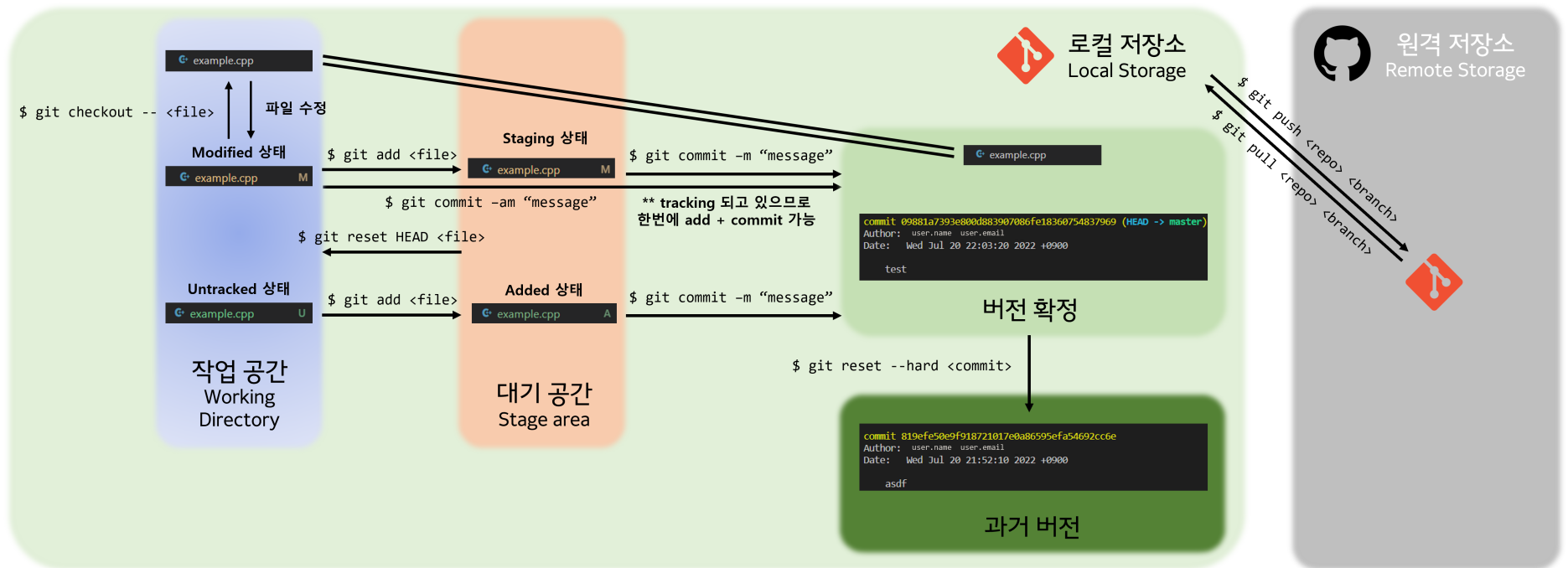

Git Basics:
Get Going with Git

 **목차**

- 기본 개념
 - 초기 설정
 - init
 - clone
 - 로컬저장소
 - add
 - commit
 - 원격저장소
 - pull
 - push
 - local에 저장된 git을 버리고 Pull로 덮어쓰고 싶을 경우
- 기본 명령어
 - Git 명령어
 - 리눅스 명령어
 - Vi 편집기
- Pull Request (PR)
- 여담

기본 개념

Git Flow



```
remote: Total 45 (delta 10), reused 45 (delta 10), pack-reused 0
Unpacking objects: 100% (45/45), 6.36 KiB | 310.00 KiB/s, done.
From https://github.com/mathpaul3/Algorithm-Problem-Solving
a357dec..7b7d29e  master    -> origin/master
hint: You have divergent branches and need to specify how to reconcile them.
hint: You can do so by running one of the following commands sometime before
hint: your next pull:
hint:
hint:   git config pull.rebase false  # merge
hint:   git config pull.rebase true   # rebase
hint:   git config pull.ff only       # fast-forward only
hint:
hint: You can replace "git config" with "git config --global" to set a default
hint: preference for all repositories. You can also pass --rebase, --no-rebase,
hint: or --ff-only on the command line to override the configured default per
hint: invocation.
fatal: Need to specify how to reconcile divergent branches.
```

초기 설정

init

```
git init
# 현재 작업중인 디렉토리에 빈 git 레포지토리를 생성한다

git config --global user.name "<유저명>"
# git에서 commit 할 때의 유저명을 설정한다 (가능하면 github와 동일한 것이 좋다)
# ex) git config --global user.name "username"

git config --global user.email "<이메일>"
# git에서 commit 할 때의 유저명을 설정한다 (가능하면 github와 동일한 것이 좋다)
# ex) git config --global user.email "user@email.com"
```

clone

```
git clone <Github주소>
# 현재 디렉토리에 repository명으로 된 폴더를 생성하고 해당 repository를 복사한다
# ex) git clone https://github.com/mathpaul3/function
```

로컬저장소

파일 flow에 관한 이미지 첨부 예정

// git에서 잘못 Edit | Staging | Commit 했을 때 되돌리는 법

\$ git checkout — <파일> : 작업트리에서 잘못 수정했을 때 되돌린다.


\$ git add <파일>

\$ git reset HEAD <파일>

: Staging 상태의 파일을 작업트리로 내린다.

[GitHub] git add, commit, push 취소/변경/덮어쓰기(reset, revert, --amend)

'git add취소하기, git commit 취소하기, git commit 변경하기, git commit 덮어쓰기, git push 취소하기'에 대해 정리한 내용입니다.

 https://velog.io/@falling_star3/GitHub-git-add-git-commit-git-push-취소변경덮어쓰기



add

```
git add <파일명.확장자>
# 파일을 Stage(임시저장공간)에 올린다
# ex) git add a.py

git add <파일명.확장자> <파일명.확장자>
# 여러개를 한 번에 add함
# ex) git add a.py b.py c.py
# ex) git add *.py b.* c.cpp
# ex) git add .

git reset HEAD <파일명.확장자>
# Staging 상태의 파일을 작업트리로 내린다
```

commit

```
git commit -m "<메시지>"
# Staging 상태의 파일들을 로컬 저장소에 올려 버전을 확정한다
# <메시지>에는 해당 버전에 대한 설명을 작성한다
# ex) git commit -m "test function has been added"

git commit -am "<메시지>"
# 모든 Modified 상태의 파일들을 한 번에 add하고, commit 한다
```

```
git reset --hard <commit의 해시값>
# hash 값의 commit 상태로 되돌린다
# commit의 hash값은 git log로 확인 가능하다
```

파일을 생성/수정하여 작업트리에 있음 [Untracked/Modified 상태]

→

\$ git add 파일명.확장자 : Stage(임시저장공간)에 올림 [Staging 상태]

→

\$ git commit -m "메시지" : Staging 상태의 파일들을 로컬저장소에 올림

원격저장소

```
git remote add origin <Github주소.git>
# local git의 origin을 Github주소의 repository로 설정한다
# ex) git remote add origin https://github.com/mathpaul3/function.git

git remote remove origin
# local git의 origin을 제거한다

git remote -v
git remote --verbose
# 원격저장소의 연결 상태를 보여준다
```

pull

```
git pull origin master
# 원격저장소의 변동 사항을 로컬저장소로 받아온다
# 처음 pull 할 경우 stream을 설정해주어야 한다
# origin의 master 브랜치(branch)를 stream으로 설정한다

git pull
# 한 번 stream이 설정되면 이후에는 위와 같이 해도 된다
```

push

```
git push -u origin master
# 처음 push 할 경우 stream을 설정해주어야 한다
# origin의 master 브랜치(branch)를 stream으로 설정한다

git push
# 한 번 stream이 설정되면 이후에는 위와 같이 해도 된다
```

local에 저장된 git을 버리고 Pull로 덮어쓰고 싶을 경우

\$ git fetch --all

\$ git reset --hard origin/master

기본 명령어

Git 명령어

\$ git —version : git 버전을 확인한다.

\$ git update : git을 업데이트한다.

\$ git status : 현재 가리키고 있는(HEAD) commit과 작업트리에 있는 파일들을 비교해 변동이 있는 파일들을 상태와 함께 보여준다.

\$ git log : commit 이력과 각 commit의 hash값, 메시지를 보여준다.

\$ git diff : 저장소의 파일과 작업트리의 수정 내용을 비교해 서로 다른 부분을 보여준다.

리눅스 명령어

리눅스 명령어가 git에서 똑같이 쓰인다.

\$ pwd : 현재 작업중인 디렉토리(Present Working Directory)의 주소를 보여준다.

\$ mkdir : 현재 위치에 디렉토리를 생성한다.

\$ cd .. : 상위 디렉토리로 이동한다. (Change Directory)

\$ cd ~ : home 디렉토리로 이동한다. (Change Directory)

\$ vim <파일명> : <파일명> 파일 생성

Vi 편집기

i 또는 a로 입력모드 (insert | add)

esc 누르면 명령모드

명령어

! 관리자권한

w write

q quit

→ !wq (관리자 권한으로, 작성한 글을 저장하고 **write**, 나감 **quit**)

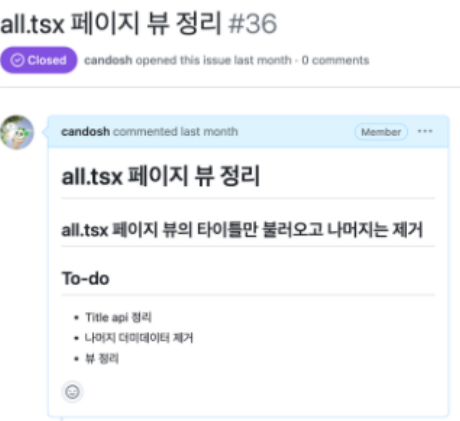
Pull Request (PR)

git merge 방식

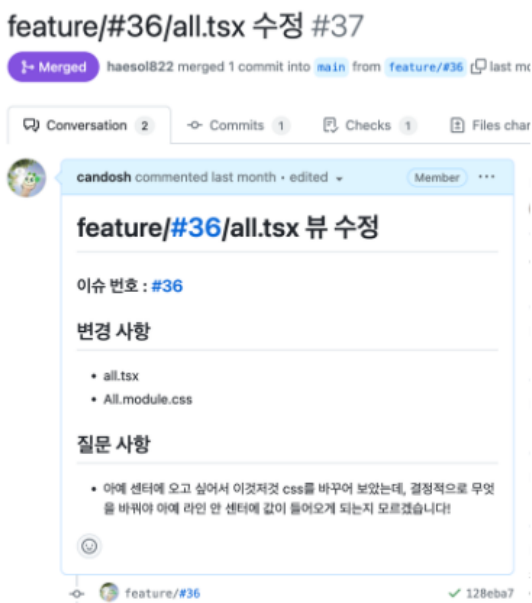
- Squash merge

간단한 머지까지의 플로우

1. 이슈 생성
 - a. 자신이 맡은 업무의 이슈를 생성합니다.
 - b. 이슈에는 어떤 일을 할것인지 적습니다.
 - c. 아래와 같이 이슈 생성시 '#~'라는 번호의 이슈번호가 생깁니다.



2. 브랜치 생성
 - a. 아래 git branch convention에 맞게 위에 생성한 이슈 번호대로 브랜치를 팝니다.
 - b. 이후 생성한 브랜치 내에서 작업을 합니다.
 - c. 작업을 한 후 아래 git commit convention에 맞는 commit && push를 합니다.
3. Pull Request
 - a. 자신의 브랜치에 푸쉬까지 마무리 했다면 이제는 PR을 작성해줍니다!
 - b. PR도 commit 메시지와 거의 동일하게 작성해주면 됩니다!



- c. 이후 merge 후 branch 삭제

git branch convention

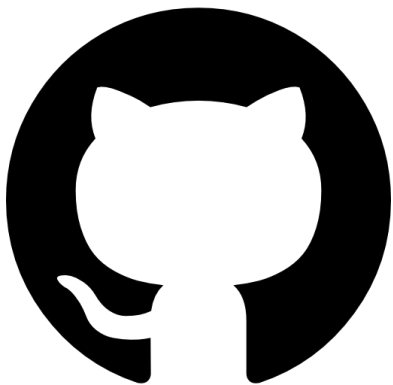
- 기본적으로 issue 번호를 따라간다.
- feature의 경우 feat/#1
- fix의 경우fix/#1

git commit convention

- 커밋convention/이슈번호/간단한커밋내용
 - ex. feat/#1/간단한 커밋 내용

Feat	기능추가	
Design	CSS 등 사용자 UI 디자인 변경	
Style	코드 포맷 변경, 세미 콜론 누락, 코드 수정이 없는 경우	
Comment	필요한 주석 추가, 변경 및 삭제	
Fix	버그 수정	
Refactor	프로덕션 코드 리팩토링, 새로운 기능이나 버그 수정없이 현재 구현을 개선한 경우	
Docs	README.md 수정	
Rename	파일 혹은 폴더명을 수정하거나 옮기는 작업만인 경우	
Remove	파일을 삭제하는 작업만 수행한 경우	
Test	테스트 코드, 리팩토링 테스트 코드 추가, Production Code(실제로 사용하는 코드) 변경 없음	
Chore	빌드 업무 수정, 패키지 매니저 수정, 패키지 관리자 구성 등 업데이트, Production Code 변경 없음	
!BREAKING	CHANGE 커다란 API 변경의 경우	
!HOTFIX	급하게 치명적인 버그를 고쳐야하는 경우	

여담



Github 아이콘의 고양이 이름은 **OctoCat**이다!
Octopus와 Cat의 합성어라고 한다

