Git 명령어 정리

Nerrtica

1. 저장소 생성

- 이미 있는 디렉토리에서 새 저장소 만들기

```
$ git init
```

- 이미 있는 저장소를 Clone하기

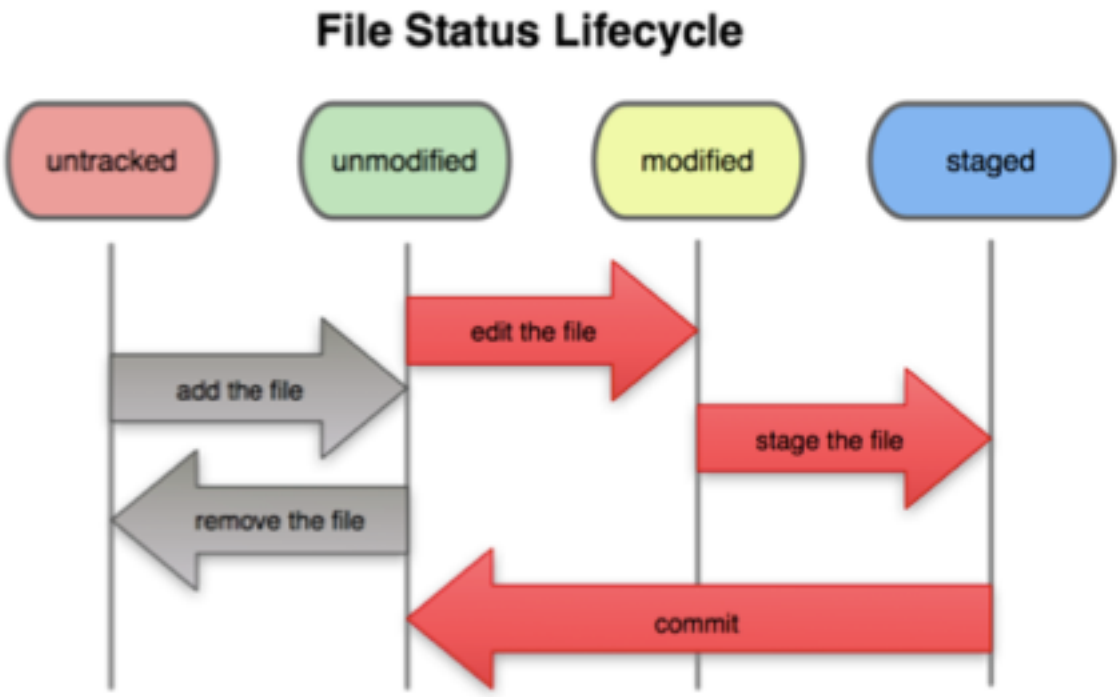
```
$ git clone [url]
```

URL은 Github, Yobi 등 사용하는 서버에서 제공한다.
HTTPS를 사용하는 경우에는 github 아이디와 비밀번호를 입력해야 한다.

2. 파일의 변경

- 파일의 상태 확인하기

```
$ git status
```



- 새 파일을 추적하기 / Modified 상태의 파일을 Stage하기

```
$ git add [file]
$ git add README
```

여러 개의 파일을 한꺼번에 add하고 싶다면 --all 옵션을 사용하거나 *을 사용한다.

- Staged와 Unstaged 상태의 변경 내용을 보기

```
$ git diff
```

수정했지만 아직 staged 상태가 아닌 파일을 비교한다.

```
$ git diff --cached
```

staging area에 있는 파일의 변경 부분을 보여준다.
--staged로도 사용할 수 있다.

- 커밋하기

```
$ git commit
```

Git 설정에 지정된 편집기가 실행되며 Commit message를 적을 수 있다.
empty message로 저장한다면 커밋되지 않는다.
-v 옵션을 추가하면 편집기에 diff 메시지도 추가된다.
-m 옵션을 사용하면 편집기를 사용하지 않고 간단하게 커밋할 수 있다.

```
$ git commit -m “Commit message”
```

- Staging Area 생략하기 (git add + git commit)

```
$ git commit -a
```

- 파일 삭제하기

```
$ git rm test.c
```

실제 파일은 지우지 않고 Git만 더 추적하지 않게 하려면 --cached 옵션을 사용한다.

```
$ git rm --cached README
```

- 파일 이름 변경하기

```
$ git mv test.c main.c
```

test.c 파일의 이름을 main.c 로 바꿔준다.

3. 히스토리 조회

```
$ git log
```

인자 없이 log 명령을 실행하면 커밋 히스토리를 시간순으로 보여준다.

```
$ git log -p
```

-p 옵션은 각 커밋의 diff 결과를 보여준다.

```
$ git log --stat
```

--stat 옵션은 어떤 파일이 수정됐는지, 얼마나 많은 파일이 변경됐는지, 또 얼마 나 많은 줄을 추가하거나 삭제했는지 보여준다.

```
$ git log --pretty=oneline
```

--pretty 옵션은 log를 출력할 때의 형식을 여러가지로 바꿔준다.
oneline, short, full, fuller 등이 있으며,

```
$ git log --pretty=format:"%h - %an, %ar : %s"
e0baf45 - Nerrtica, 10 minutes ago : delete test.c
```

format 옵션을 사용하면 나만의 형식으로 결과를 출력할 수 있다.
참고 :

```
Option Description of Output %H Commit hash

%h Abbreviated commit hash %T Tree hash
%t Abbreviated tree hash
%P Parent hashes
%p Abbreviated parent hashes
%an Author name
%ae Author e-mail
%ad Author date (format respects the -date= option) %ar Author date, relative
%cn Committer name
%ce Committer email
%cd Committer date
%cr Committer date, relative
%s Subject
```

```
$ git log -3
```

숫자를 옵션으로 넣으면 해당 갯수만큼의 log만 출력한다.

```
$ git log --graph
```

--graph 옵션은 브랜치와 머지 히스토리를 보여주는 아스키 그래프를 출력한다.

기타 옵션들

```
-p 각 커밋에 적용된 패치를 보여준다.
--stat 각 커밋에서 수정된 파일의 통계정보를 보여준다.
--shortstat '--stat' 명령의 결과 중에서 수정한 파일, 추가된 줄, 삭제된 줄만 보여준다.
--name-only 커밋 정보중에서 수정된 파일의 목록만 보여준다.
--name-status 수정된 파일의 목록을 보여줄 뿐만 아니라 파일을 추가한 것인지, 수정한 것인지, 삭제한 것 인지도 보여준다.
--abbrev-commit 40자 짜리 SHA-1 체크섬을 전부 보여주는 것이 아니라 처음 몇 자만 보여준다.
--relative-date 정확한 시간을 보여주는 것이 아니라 `2 주전`처럼 상대적인 형식으로 보여준다.
--graph 브랜치와 머지 히스토리 정보까지 아스키 그래프로 보여준다.
--pretty 지정한 형식으로 보여준다.
-(n) 최근 n 개의 커밋만 조회한다.
--since, --after 명시한 날짜 이후의 커밋만 검색한다.
--until, --before 명시한 날짜 이전의 커밋만 조회한다.
--author 입력한 저자의 커밋만 보여준다.
--committer 입력한 커미터의 커밋만 보여준다.
```

4. 작업의 취소

- 다시 커밋하기 (직전의 커밋에 합침)

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

- Modified 파일 되돌리기 (파일 변경 취소)

```
$ git checkout -- test.c
$ git checkout HEAD test.c
```

위 명령어는 commit을 하지 않은 채로 파일을 변경하기 때문에 절대로 되돌릴 수 없다.
위의 명령어는 파일을 staging area에 있는 것으로 원복한다.
아래의 명령어는 파일을 로컬 브랜치(HEAD)에 있는 것으로 원복한다.

- 파일 상태를 Unstaged로 변경하기 (add 취소)

```
$ git reset -- test.c
$ git reset HEAD test.c
```

- 커밋 취소하기

```
$ git reset HEAD^
$ git reset HEAD~3
```

HEAD^는 최종 커밋을 취소하며, 여러 개의 커밋을 취소하고 싶을 때에는 HEAD~<n>을 사용한다.
이 경우 워킹트리는 보존되므로, 워킹트리까지 모두 원복하고 싶다면 --hard 명령어를 사용한다.
(default값은 --mixed이다. --soft는 워킹트리, 깃 레포 모두 보존, mixed는 워킹트리 보존, hard는 둘 모두 보존하지 않는다.)

```
$ git revert HEAD
```

reset과는 다르게, 변경된 내용을 되돌리는 새로운 커밋을 발행한다.
역시 HEAD~<n>과 같은 방식으로 여러 개의 커밋을 돌릴 수 있다.
커밋을 이미 push해버린 경우 유용하다.

5. 나만의 명령어

```
$ git config --global alias.unstage 'reset HEAD --'
$ git unstage test.c
```

```
$ git config --global alias.last 'log -1'
$ git last
```

alias 옵션을 이용해 나만의 명령어를 만들 수 있다.

6. 브랜치

브랜치란 커밋 사이를 이동할 수 있는 어떤 포인터 같은 것이다.

- 브랜치 생성

```
$ git branch test
```

브랜치를 새로 생성하면 가장 마지막 커밋을 가리킨다.
여러 개의 브랜치 중 지금 작업 중인 브랜치가 무엇인지는 'HEAD'라는 특수한 포인터를 통해 알 수 있다.
HEAD 포인터는 지금 작업하는 로컬 브랜치를 가리킨다.
branch 명령어를 사용하면 브랜치를 새로 만들지만, HEAD 포인터를 옮기지는 않는다.

- 브랜치 이동

```
$ git checkout test
```

이제 HEAD는 test 브랜치를 가리킨다.

이후 새로 커밋을 해 보면, test 브랜치는 앞으로 이동해 새 커밋을 가리키지만, master 브랜치는 여전히 이전 커밋을 가리킨다.

```
$ git checkout master
```

master 브랜치로 HEAD를 이동시키면 Working Directory의 파일도 master 브랜치가 가리키는 커밋의 시점으로 돌아간다.
그리고 다시 커밋을 하게 된다면 이제 master와 test의 프로젝트 히스토리는 분리되어 진행하게 된다.

```
$ git checkout -b 'test2'
```

checkout에 -b 옵션을 주면 브랜치의 생성과 이동을 동시에 실행해준다.

- 브랜치 합치기

```
$ git merge test
```

test 브랜치가 master 브랜치보다 한 커밋 앞서있는 상황에서, HEAD를 master로 이동시키고 test에 merge한다면 master 브랜치는 최신 커밋(test)으로 이동하게 된다.

- 브랜치 지우기

```
$ git branch -d test
```

master와 test를 합치며 test 브랜치는 필요없어졌으므로 삭제한다.

- rebase

```
$ git rebase test
```

브랜치를 합치는 또 다른 방법이다. 커밋 히스토리가 선형적으로 나오게 된다.

7. 충돌

충돌이 났을 경우, 파일 내 충돌이 난 부분이 아래와 같은 형식으로 바뀌어 있다.

```
<<<<<<< HEAD
blabla…
=======
…blabla
>>>>>>> branch
```

위쪽의 내용은 HEAD가 가리키는 브랜치의 파일 내용, 아래쪽은 해당 branch명의 파일 내용이다.
충돌이 난 부분을 수정한 후 다시 add하고 commit하면 완료.

```
$ git mergetool
```

본인 컴퓨터에 툴이 깔려 있을 경우에 사용한다.

8. 기타

```
$ git reflog
```

커밋 기록만을 출력해주는 log 명령과는 달리, 다른 기록들도 출력해준다.
reset으로 HEAD포인터를 되돌려, 잃어버린 커밋이 생겼을 때 해당 행동을 되돌리는 등 고급 응용으로 사용할 수 있다.

출력되는 log중 돌아가고 싶은 순간의 HEAD@{<n>}을 입력해준다.

```
$ git reset HEAD@{5}
```

09a0045 HEAD@{5}: reset: moving to HEAD^ 였다면, 해당 상황으로 돌아간다.