Git 사용하기

temp	
# 챕터번호	4

01. Git 사용하기

01-01. 설정하기

01-01-01. 사용자 인증

Git을 처음 다운로드를 하게되면 Git을 사용하는 사용자 설정을 해주어야 하는데 사용자 설정의 경우 글로벌 설정과 프로젝트 마다 설정하는 방식으로 나뉘게 된다. 이때 인증 방식은 SSH(시크릿 키) 방식과 설정 및 로그인 방식이 있으며 SSH 설정은 앞에서 다루어 보았다. 이번 챕터는 전역 설정과 프로젝트별 설정을 구분하여 설정한다.

A. Global 설정

"git --global config"는 Git의 전역 설정을 변경하기 위한 명령어로 이 명령어를 이용하면 Git의 전역 설정을 변경할 수 있게 되며 이는 해당 컴퓨터를 이용하는 모든 사용자에게 git의 설정 정보가 공유된다.

a. Git을 설치한뒤 Command 창에서 git을 입력을 해본다.
이때 git의 관련된 내용이 나오면 정상 설치가 된 것이며 알 수 없는 명령어 문구가 출력되면 재설치를 해주어야 한다.
git의 명령어 인식이 된다면 다음과 같은 명령어를 입력한다.

```
$ git --clobal config user.name "" // 사용자 식별을 위한 이름을 설정한다.
$ git --clobla config user.email "" // 사용자가 사용하는 git의 Email을 입력한다.
```

b. 설정된 정보를 확인하기 위해서는 명령어를 사용한다.

```
$ git config user.name // 설정된 사용자의 이름을 확인한다.
$ git config user.email // 설정된 사용자의 이메일을 확인한다.
```

B. Directory 설정

회사에서 사용하는 계정과 개인적으로 사용하는 계정으로 2가지가 구분되는 경우 매번 설정을 변경하는 것은 번거로운 작업이 될 것이다. 그렇기 때문에 다음과 같은 방식으로 우리가 Clone(복제)한 Project별로 사용자 설정을 관리하는 것이 가능하다.

a. 프로젝트를 내려받은 디렉토리로 이동한다.

```
$ git config user.name "" // 해당 디렉토리의 사용자 이름을 입력한다.
$ git config user.email "" // 해당 디렉토리의 사용자 이메일을 입력한다.
```

b. 설정된 정보를 확인 하기 위해서는 다음 명령어를 사용한다.

```
$ git config user.name // 설정된 사용자의 이름을 확인한다.
$ git config user.email // 설정된 사용자의 이메일을 확인한다.
```

C. 설정 제거하기

어떠한 일로 인해 컴퓨터를 변경해야 하는 경우 사용자의 정보를 삭제하는 것이 필요한 순간이 오는데 사용자 정보를 삭제하지 않는 경우 해당 아이디를 통해 우리의 원격 저장소의 수정 및 제거가 가능하기 때문에 주의를 해야 하는 부분이다.

a. Global 설정 제거하기
Command 창에서 아래의 명령어를 입력한다.

```
$ git config --global --unset user.name // 사용자의 이름을 제거한다.
$ git config --global --unset user.email // 사용자의 이메일을 제거한다.
```

b. Directory 설정 제거하기
Command 창에서 삭제하고자 하는 Directory로 이동한다.

```
$ git config --unset user.name //사용자의 이름을 제거한다.
$ git config --unset user.emil // 사용자의 이메일을 제거한다.
```

01-02-01. oh-my-zsh 설정하기

처음 Git을 사용하면 가장 불편하고 어려운 부분으로 현재 나의 화면이 Git에서 관리하는 부분인지 아닌지 식별하는 것이며 이후 Branch로 별도의 분기를 나누어 관리를 하게 되면서 Branch의 대한 설정이 어려워진다.
이외로 현재 로컬 작업의 변경 사항등의 다양한 문제를 확인 할 수 있도록 Command 창을 꾸밀수 있다.

이는 Mac과 Windos 설정이 둘다 가능하다.
해당 내용은 아래의 정리가 잘 되어 있는 링크를 첨부한다.

https://velog.io/@saemsol/Oh-My-Zsh

01-02. 명령어

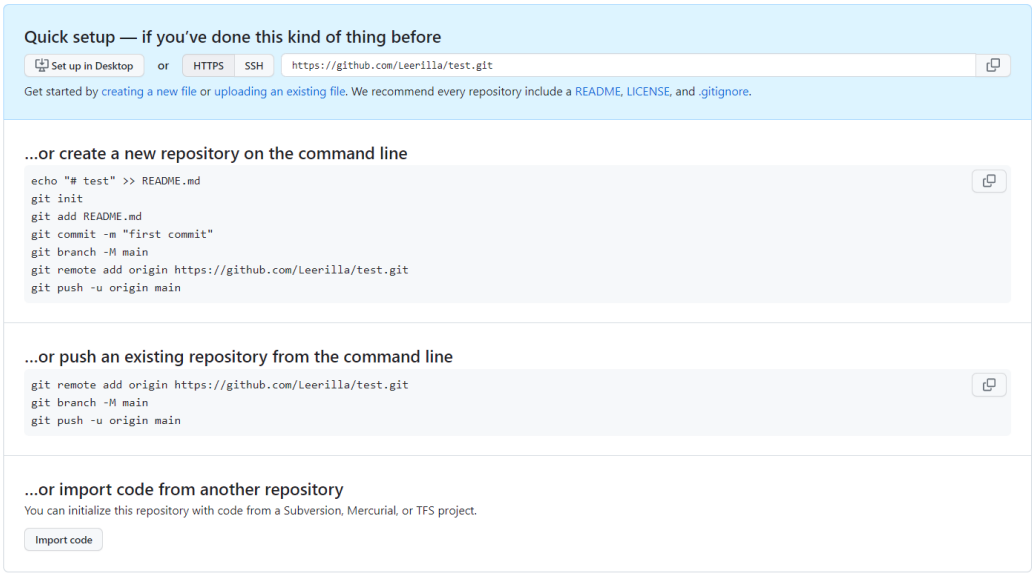
01-02-01. Git 기초 문법

Github에서 우리가 만든 프로젝트를 관리하기 위해서는 Git을 통해 로컬 환경에 있는 코드를 관리해야 하는데 기초 문법 파트에서는 Github에서 파일을 관리할 수 있도록 하는 기초 명령어를 배운다.

A. 레파지토리 연결하기

우리가 Github에서 Repository를 생성할 때 아무런 내용이 없으면 비어있는 Repository가 생성이 되는데 여기서 우리가 로컬에서 작업한 내용을 추가하고자 하는 경우 로컬에서 Git이 관리할 수 있도록 remote 설정을 통해 연결을 해준 다음 push를 해주어야 한다.

a. 원격 레파지토리 생성하기
레파지토리를 생성할 때 아무런 파일을 만들지 않으면 아래와 같은 형식으로 생성된다.



b. 로컬 디렉토리와 원격 브랜치 연결하기

연결하고자 하는 디렉토리에 Git을 생성해 준다.

```
$git init // 로컬 디렉토리를 git이 관리하도록 한다.
```

c. 추가하고자 하는 파일/폴더을 추가한다.
스테이징 영역에 파일/폴더을 추가하는 것으로 자세한 내용은 이후에 다룬다.

```
$git add [파일/폴더 이름] // 로컬 git에서 스테이징 영역으로 파일을 추가한다.
```

d. 추가한 파일의 설명을 추가한다.
자세한 내용은 이후에 다루지만 간단하게 추가한 파일/폴더의 목적을 설명한다.
최초 파일이라는 것을 알리기 위해 “frist commit”으로 작성한다.

```
$git commit -m "message" // 추가한 파일의 설명을 message 부분에 작성한다.
```

e. 로컬에 브랜치를 main으로 설정해준다.
브랜치 관련된 내용도 이후 자세히 설명을 한다.

```
$git branch -m main // 현재 브랜치를 메인으로 설정하는 것이다.
```

f. 로컬 브랜치와 원격 브랜치를 연결한다.
git clone을 하게 되면 자동으로 설정이 되지만 로컬에서 생성된 git의 경우 따로 설정을 해주어야 한다.
여기서 remote는 인터넷이나 네트워크 어딘가에 있는 저장소를 의미하며 아래의 명령어는 해당 저장소에 연결을 하는 것이다.

```
$git remote url.git // github 레파지토리 연결
```

g. 원격저장소(Github)에 저장된 코드를 업로드 한다.

```
$git push -u origin main // main 브랜치에 로컬에서 관리된 파일을 업로드 한다.
```

B. Git 기초 문법 알아보기

a. git clone

"git clone"은 Git 저장소를 로컬 컴퓨터로 복제하는 명령어로 즉, 원격 저장소에 있는 Git 프로젝트를 다운로드하여 로컬 컴퓨터에서 작업할 수 있도록 해주는 명령어이다.

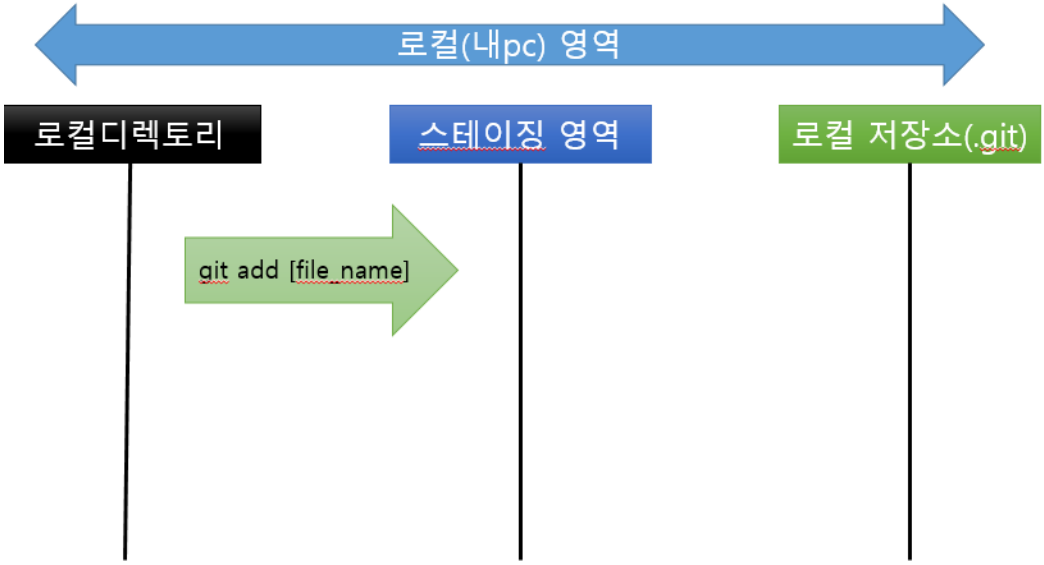
```
// 원격 저장소의 파일을 뒤에 설정한 로컬 디렉토리에 저장한다.
$git clone [원격 저장소 주소] [로컬 디렉토리]
// 기본 설정은 현재 루트로 저장이 되며 커맨드 라인이 존재하는 공간에 저장된다.
$git clone [원격 저장소 주소]
```

b. git add [파일/폴더 이름]
로컬 영역에 관리하는 파일 혹은 폴더를 스테이징 영역에 추가하는 명령어이다.

- 스테이징

  버전 제어 시스템의 중요한 구성 요소로 리포지토리의 변경 사항을 모니터링 하며
  변경 사항의 내용을 저장하고 관리하는 역할을 담당한다.

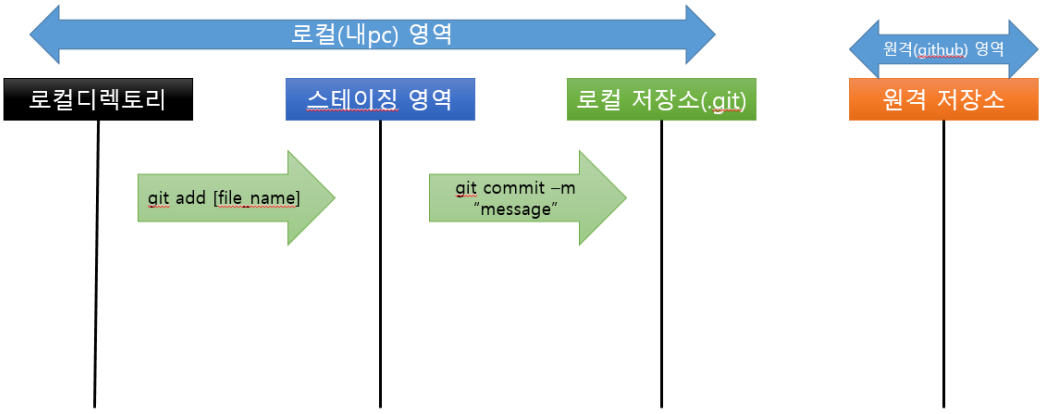
```
$git add [파일/폴더 이름] // 파일 및 폴더를 스테이징 영역에 추가한다.
$git add . // 변경된 모든 정보를 추가한다. (주의해서 사용해야 한다.)
```



c. git commit -m “message”

스테이징에 저장된 수정 내용을 추가하고 관리하는 역할을 담당한다.
commit은 현재 스테이징에 추가된 파일의 묶음 단위로 생성이 되며 commit 이후 새로운 변경사항을 저장하게 되면 해당 파일은 새로운 commit으로 관리를 해주어야 한다.

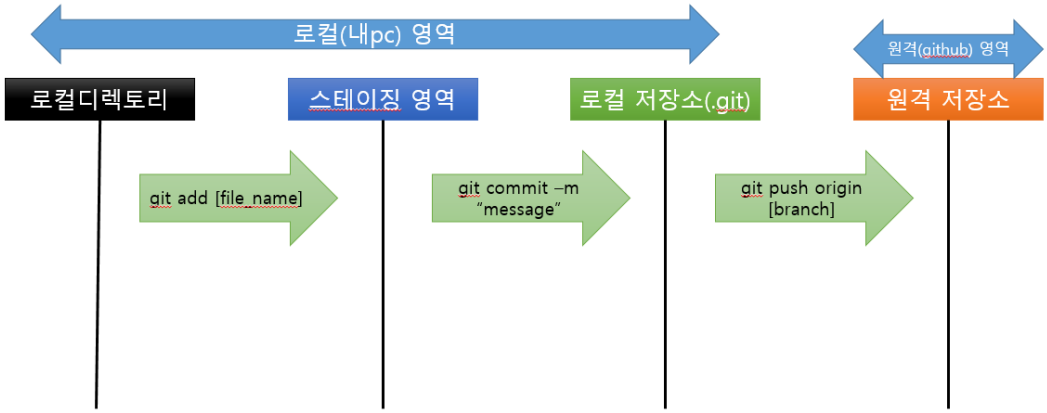
```
$git commit -m "message" // message 부분에 변경된 내용을 작성한다.
```



d. git push origin [branch]

"git push origin [branch]" 원격 저장소에 로컬에서 관리하는 파일을 등록하는 명령어로
원격 저장소에 파일을 등록하게 되면 로컬에서 관리하는 파일이 업로드가 되며 원격 저장소와 동기화가 된다.

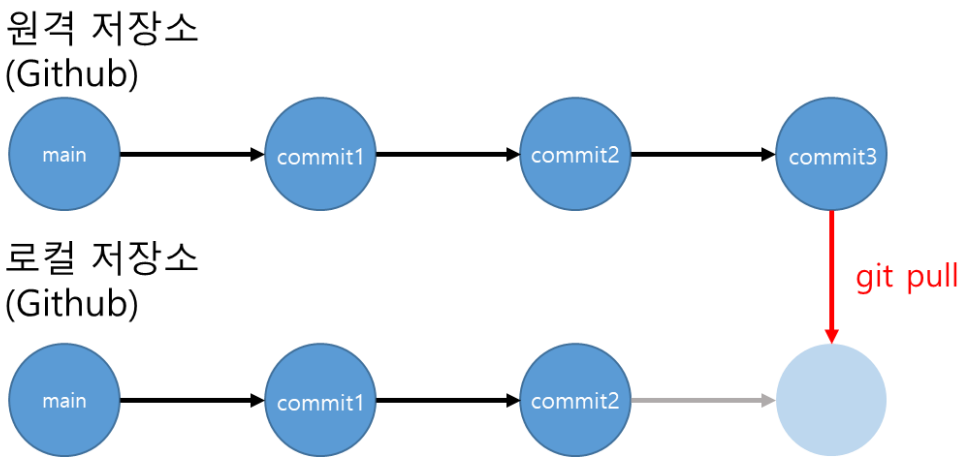
```
$git push origin [branch] // 업로드 하고자 하는 branch의 이름을 입력한다.
$git push -u origin [branch] // 설정된 내용을 기본값으로 저장하여 생략할 수 있도록함
$git push //origin [branch] 생략하고 push가능

$git config --global push.default current
// git branch를 만들게 되면 push 명령어 수행시 해당 branch로 push를 해야 하는데
// 이는 매번 브랜치 이동시 변경해야 하기 때문에 번거롭다.
// 위 명령어는 이러한 불편함을 개선하고자 기본 push를 현재 브랜치 이름으로 push한다.
// 주의 사항은 원격과 로컬 브랜치의 이름이 동일해야 한다.

$git push -f origin [branch] // 원격 저장소의 코드 변경 이력을 로컬로 덮어써준다.
//최대한 사용을 하지 않는게 좋다.

$git push origin [branch] --force
// 원격 브랜치에 푸시가 안되는 문제가 종종 발생된다.
// 이는 브랜치의 분기가 맞지 않거나 혹은 충돌로 인한 문제가 대부분이지만
// 이러한 방식으로 해결이 안되는 경우 강제 푸시를 이용할 수 있다.
```



e. git pull origin [branch]

원격 저장소에 있는 내용을 내려받는 명령어로 원격 브랜치와 로컬저장소의 분기를 맞추는 작업을 해준다.
분기를 맞추는 이유는 git은 공용으로 코드를 관리하는 저장소로 참여한 모두의 코드 상태가 다르기 때문에 분기를 맞춰서 원격 저장소는 항상 최신의 상태를 유지해야 하기 때문이다.

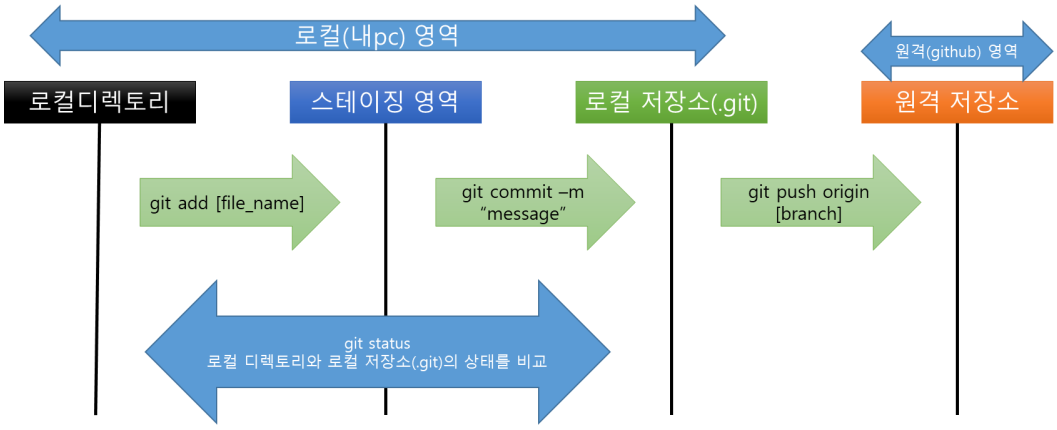
```
$git pull origin [branch] // 해당 브랜치의 변경된 내용을 내려받는다.
```



f. git status

최근 연결된 분기(pull)와 현재 로컬의 상태를 비교하여 변경된 내용을 확인하고자 할 때 사용하는 명령어로 변경된 이력을 추적할 수 있다.

```
$git status
```



01-02-01. git 중급 문법

Git의 중급 문법 파트는 우리가 사용하는 Git의 특수한 기능을 수행하는 문법으로 이로 인해 history 및 시스템에 영향을 줄 수 있는 명령어를 배울 것이다.

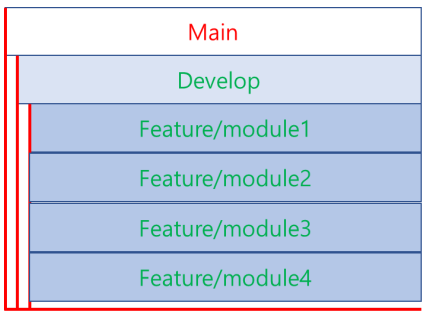
A. branch 관련 명령어

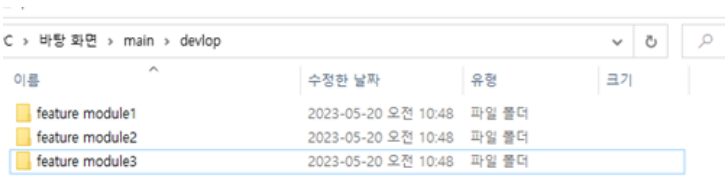
branch는 독립적으로 어떠한 작업을 진행하기 위한 개념으로 필요에 의해 만들어지는 각각의 브랜치는 **다른 브랜치의 영향을 받지 않기** 때문에 여러 작업을 동시에 진행할 수 있게 된다.

여러 명의 개발자와 협업을 하게 되면 git을 사용하면서 많은 문제를 경험한다.
이중 가장 대표적인 문제는 충돌 문제와 branch의 history가 꼬여버리는 문제이다.
이러한 것을 해결하기 위해서 branch를 사용하여 이를 통해 충돌의 문제를 최소화 할 수 있게 된다.
혼자서 개발하는 것이 아닌 이상 git branch는 필수로 볼 수 있기 때문에 해당 파트는 중요하다.

git branch는 아래와 같은 모양으로 형성이 되며 이는 git flow 전략의 일부분을 참고하였다.
git branch 전략은 다양하게 있으며 무조건 해당 전략을 따르는 것이 아닌 해당 모델을 참조하여 내부의 상황에 맞춰서 사용을 한다.

아래의 브랜치는 main 브랜치에서 develop 브랜치를 형성하고 이후 feature 브랜치를 만든 것이다.





이해를 높이기 위해 윈도우로 생성

a. git branch [option]

git branch 이후 option을 입력하여 브랜치 선택, 보기 등의 기능이 가능하다.

merge(병합) : 각각 다른 branch에 있는 내용을 합치는 것을 의미한다.

$git branch -i : 로컬에 있는 branch의 정보를 보여준다. [-i 생략가능]

$git branch -v : 로컬 브랜치의 정보를 마지막 commit 내역과 함께 보여준다.

$git branch -r : 리모트 저장소의 branch 정보를 보여준다. (리모트 = 원격)

$git branch -a : 로컬 / 리모트 저장소의 모든 정보를 보여준다.

$git branch [브랜치명] : 브랜치를 생성한다.

$git branch [-merged | -no-merged] : 옵션으로 merged는 merge된 branch를 보여줌 -no-merged는 아직 merge 되지 않은 브랜치만 보여준다.

$git branch -d [branch 이름] : 해당 브랜치를 삭제한다.

$git branch -m [변경할 branch 이름] [변경될 branch 이름] : 브랜치의 이름을 변경한다.

b. git checkout [option] [브랜치명]

git의 브랜치를 변경할 때 사용용하는 문법이다.
현재 git checkout 문법은 **git 2.23 버전부터 switch로 변경**해서 사용되고 있는데 이는 git checkout 하나의 명령어가 가진 기능이 너무 많아서 분리를 하고자 함이다.

아래의 내용은 branch의 관련된 checkout 내용만 작성되었다.

💡 $git checkout [브랜치명] : 해당 브랜치로 이동한다.

💡 $git checkout -t [origin/브랜치명] : 리모트의 브랜치를 가져온다.

💡 $git checkout -b [branch Name] [origin/branch] : 원격 브랜치를 branch name으로 변경해서 가져옴

c. git switch [옵션] [브랜치명]

💡 $git switch [브랜치명] : 브랜치를 이동한다.

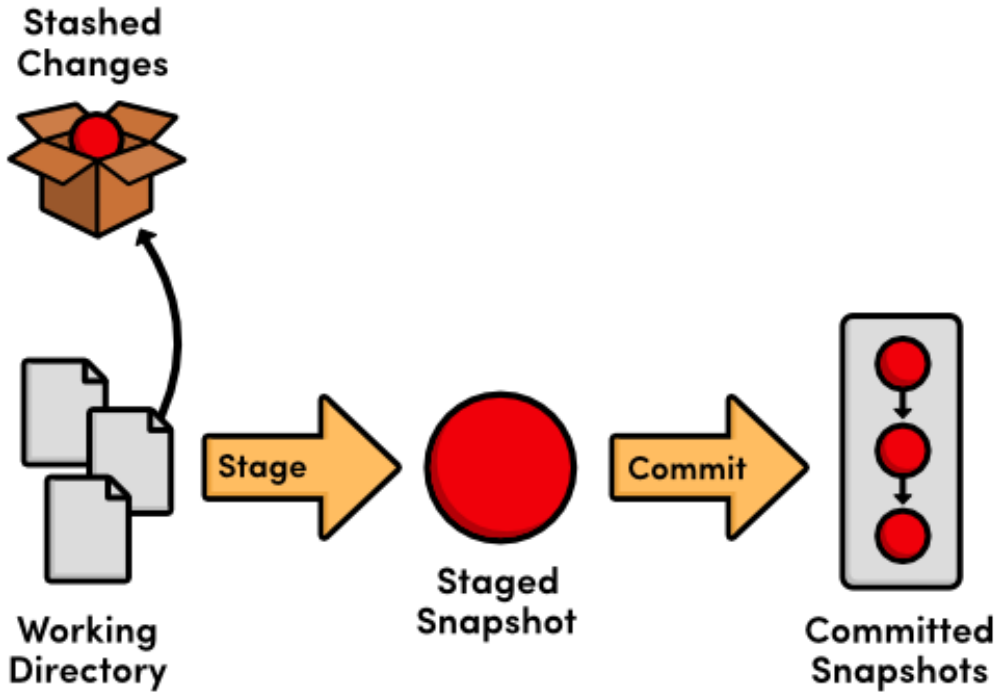
💡 $git switch -c [브랜치명] : 브랜치를 생성 후 이동한다.

💡 $git switch -c [브랜치명] <commit id> : 브랜치를 생성 후 commit id를 기점으로 시작한다. | 시작한다는 것은 해당 커밋의 내용을 복제해서 사용한다는 뜻

B. git stash

아직 마무리하지 않은 작업을 스택에 잠시 저장할 수 있도록 하는 명령어로 이를 통해 완료하지 않은 일을 commit 하지 않고 나중에 다시 꺼내와서 마무리할 수 있다.

여러 명의 개발자와 git을 사용하면 코드의 분기가 다르기 때문에 push전 원격 레파지토리의 내용을 내려받고 이후 내용을 push해야 한다. 그러나 내려받기를 하려고 하면 충돌의 문제로 내려받기가 되지 않는 경우가 있는데 이러한 경우 우리가 작업해 놓은 것을 임시 공간에 저장하고 이후 내려받아 분기를 맞춘 뒤 이후 합쳐서 병합을 한 내용을 업로드할 수 있다.



a. git stash

> 💡 $git add .
> $git stash : staging area 영역에 저장되어 있는 파일을 임시 저장공간에 보관한다.

b. git statsh save

> 💡 $ git statsh save : staging area 영역에 저장되어 있는 파일을 임시 저장공간에 보관한다.

c. git stash pop

> 💡 $git stash pop : 임시 저장소에 저장된 파일을 working diectory로 꺼내온다.
> 여기서 임시 저장소에서 꺼내오게 되면 이후 내용은 제거된다.

d. git stash apply

> 💡 $git stash apply : 안전하게 코드를 꺼오는 것으로 저장된 변경 사항을 삭제하지 않고 꺼내올 수 있다.

e. git stash list

> 💡 $git stash list : 임시 저장공간에 있는 stash 목록을 확인할 수 있다.

f. git stash apply stash@[number]

> 💡 $git stash apply stash@[number] : stash list 입력하면 맨 앞에 stash@{number}가 있다 이는 가장 최근에 변경한 내용을 순서대로 쌓는데 여기서 number를 입력하여 순서대로 꺼내올 수 있다.

g. git stash drop stash@{number}

> 💡 $git stash drop stash@{number} : 임시 공간에 있는 stash를 삭제한다.

C. git fetch

git pull은 원격 브랜치의 내용과 로컬 브랜치의 내용을 내려받아 병합을 하지만 git fetch는 git의 최신 변경 사항만 확인하여 분기를 맞추고 push를 할 수 있도록 하는 것으로 병합을 하지 않는다

a. git fetch —all

모든 브랜치의 변경 사항을 체크한다.

```
$git fetch --all
```

b. git fetch -f

확인 가능한 브랜치 내역이 나온다.

```
$ git fech -r
```

git을 사용하면 의도하지 않은 문제들로 인해 문제가 발생될 수 있다. 이러한 경우 원격 브랜치의 내용을 로컬의 내용으로 덮어 씌우거나 로컬을 원격 브랜치의 내용으로 덮어 씌우는 방식이 존재한다.

D. git restore

워킹 트리의 파일을 복원해 주는 역할을 한다.
해당 명령어는 git **2.23 버전으로 업데이트 되면서 추가 되었다.**
git checkout에서 작업이 가능하지만 모호성으로 인해 해당 문법이 추가되었다.

a. git restore

$git restore [파일이름] //파일을 수정한 뒤 복원하려고 하는 경우 사용한다.

$git checkout — [파일이름] // 위와 동일한 작업을 하며 모호성으로 인해 위 문법이 나왔다.

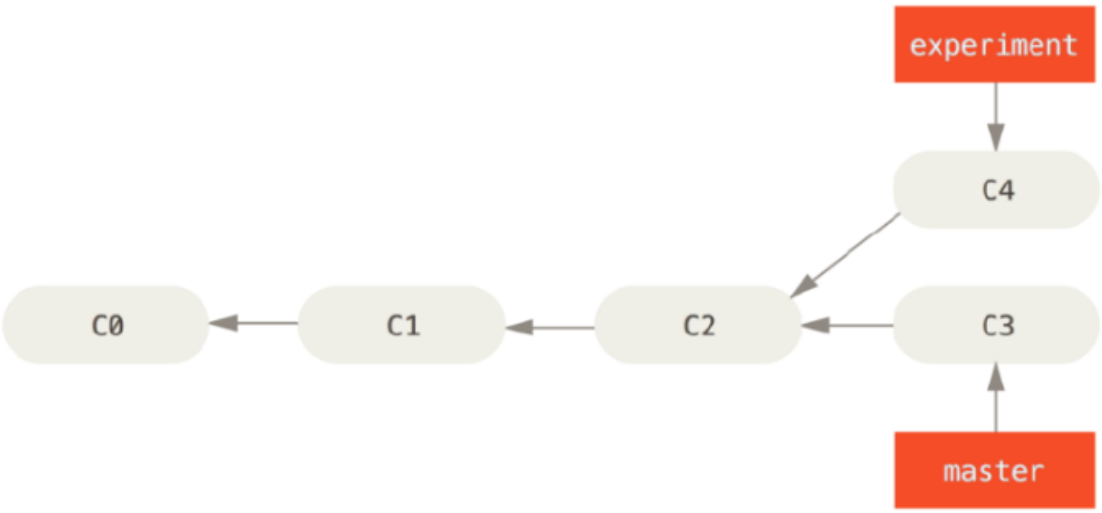
$git resore — staged [파일이름] // stage 영역에 파일을 넣은 경우 다시 되돌릴 때 사용한다.

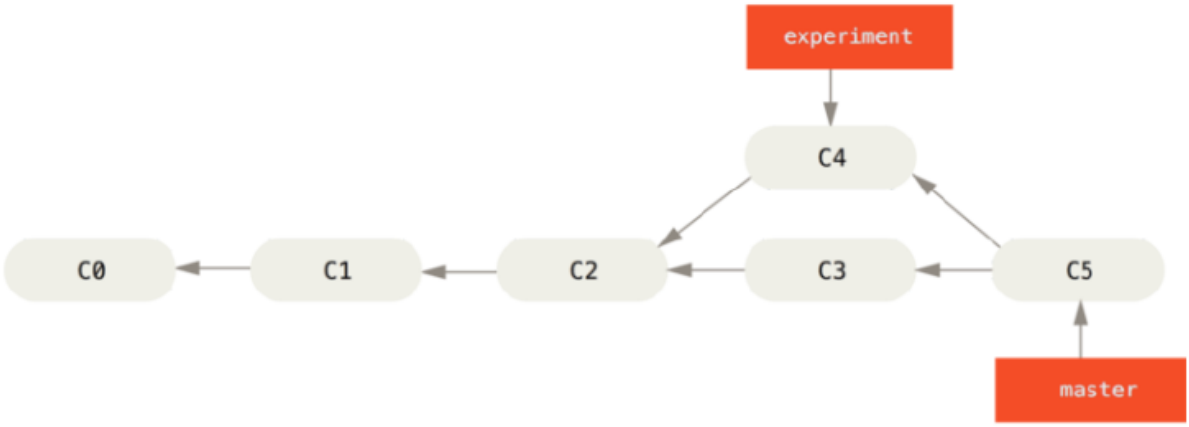
$git reset HEAD [파일이름] // 위와 동일하다.

$git cherry-pick [커밋id] // 체리처럼 콕 집는다는 의미로 해당 commit으로 분기를 이동한다.

$git merge [branch] // 현재 브랜치에 입력된 브랜치를 병합한다.

- Merge로 통합하기

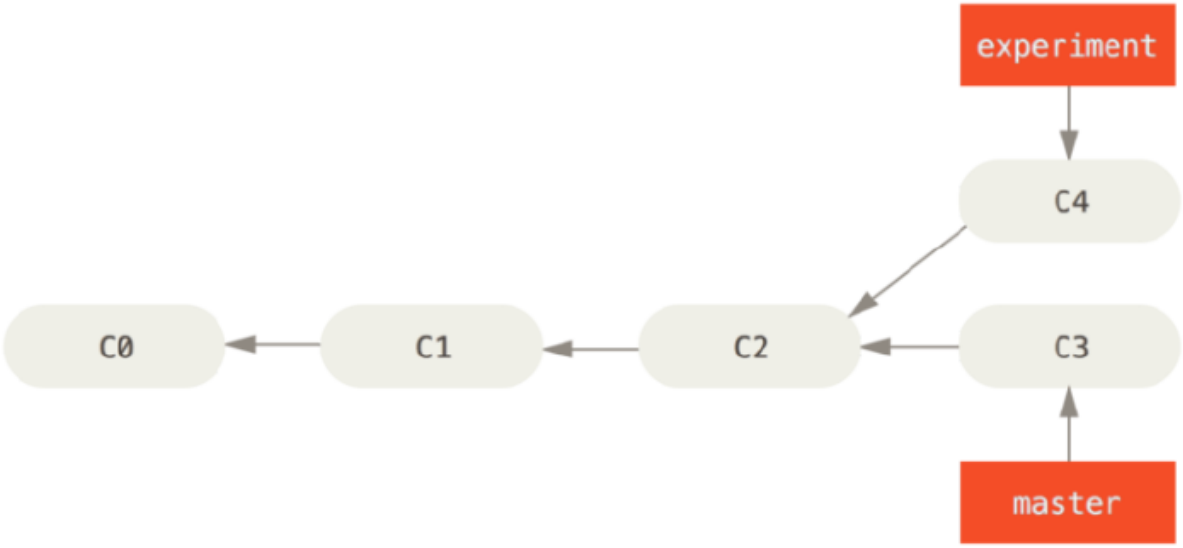


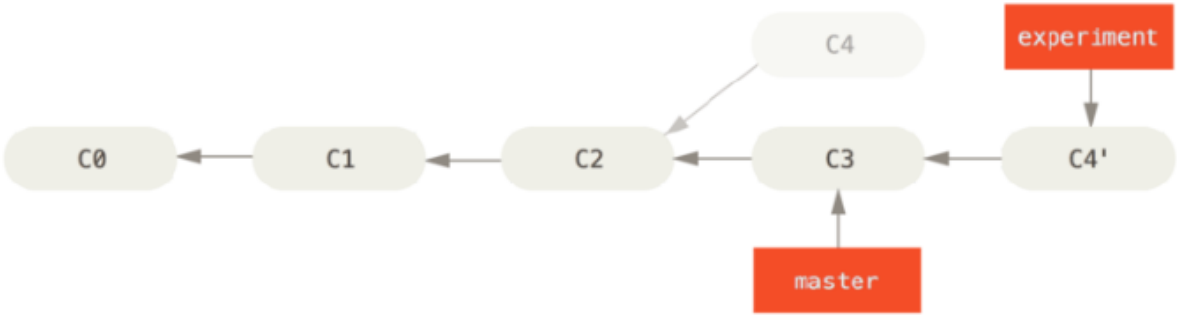


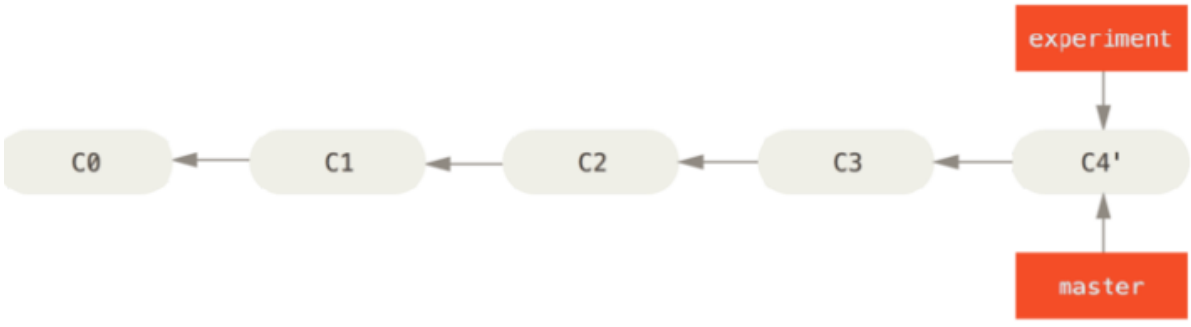
$git rebase [branch] //커밋의 히스토리를 깔끔하게 정리하는 역할을 수행하는 명령어이다.

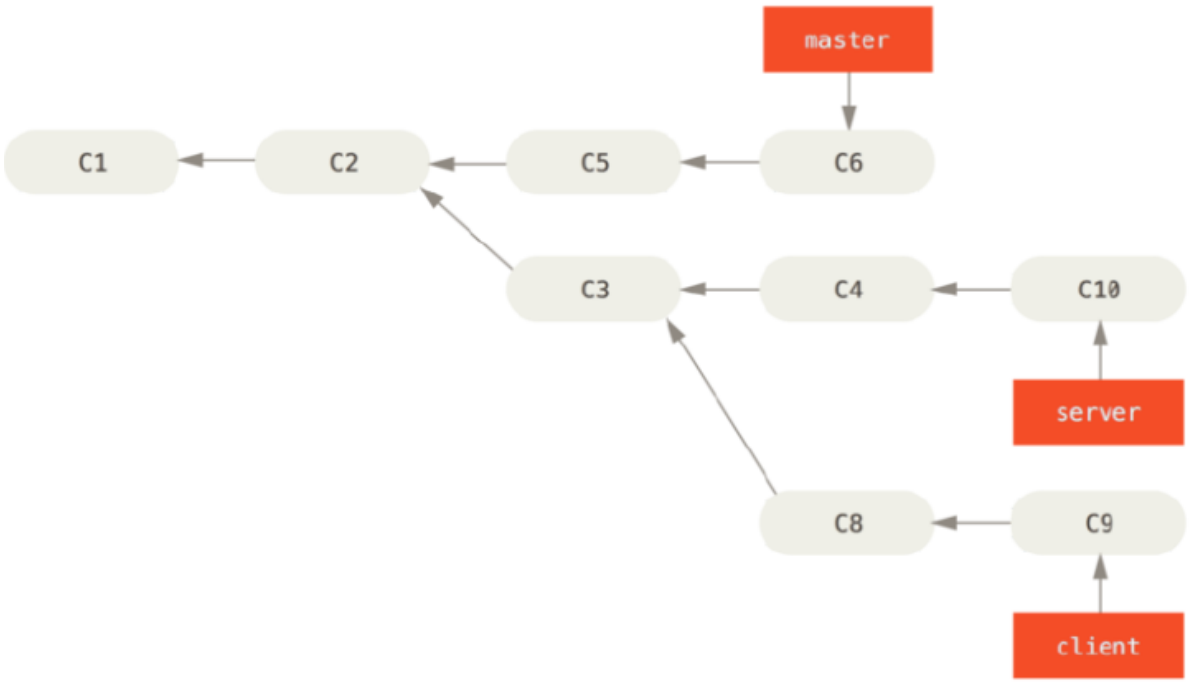
rebase의 경우 현재 브렌치를 다른 브랜치에 원래 하나였던 것 처럼 설정하여 베이스로 잡아 해당 브랜치를 합치는 과정을 수행한다.

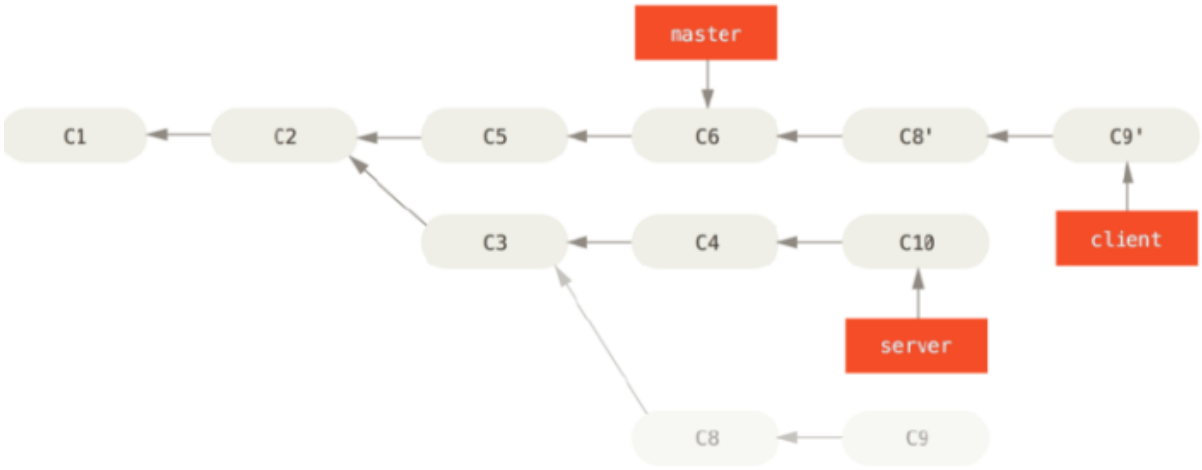
merge와 차이는 머지는 해당 해당 브랜치에 다른 브랜치에 합치는 방식이지만 rebase는 현재 브랜치를 합치려고 하는 브랜치 뒤에 분기를 만들어 원래 하나의 브랜치인 것 처럼 합치고 commit을 하나로 합치게 된다.

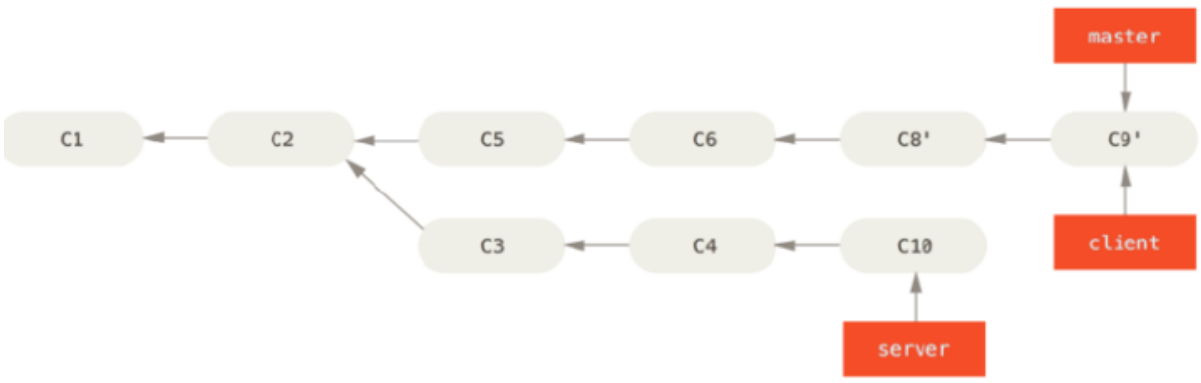

experiment
C4
C0
C1
C2
C3
master


C4
experiment
C0
C1
C2
C3
C4'
master


experiment
C0
C1
C2
C3
C4'
master


master
C1
C2
C5
C6
C3
C4
C10
server
C8
C9
client


master
C1
C2
C5
C6
C8'
C9'
client
C3
C4
C10
server
C8
C9


master
C1
C2
C5
C6
C8'
C9'
client
C3
C4
C10
server


master
C1
C2
C5
C6
C8'
C9'
C3'
C4'
C10'
C3
C4
C10
client
server


master
C1
C2
C5
C6
C8'
C9'
C3'
C4'
C10'