

A silver laptop sits on a light-colored wooden desk. A semi-transparent white rectangular box is centered over the laptop screen, featuring a solid orange horizontal bar at its top. Inside the box, the text 'Chapter3-1. Git' is displayed in a bold, black, sans-serif font. The background shows a minimalist desk setup with a small potted plant on the right and some geometric objects on the left.

## Chapter3-1. Git

A photograph of a silver laptop on a light-colored wooden desk. A semi-transparent white rectangular overlay is positioned in front of the laptop screen. A solid orange horizontal line is located just above the top edge of this overlay. The text 'Part 0. Git이란?' is centered on the overlay in a bold, black, sans-serif font. In the background, to the left of the laptop, are three white geometric blocks of varying heights. To the right, there is a small potted plant with green leaves in a white square pot, and a small, thin, gold-colored geometric object.

## Part 0. Git이란?

# Version Control System

---

## | 분산형 버전관리 시스템 Git



파일의 변경사항을 저장하고, 기록하고  
원하는 시점의 버전을 다시 꺼내올 수 있는 시스템을  
버전/형상관리 시스템이라고 합니다.

Git은 대표적인 분산형 버전관리 시스템입니다.



# Version Control System

버전 관리 시스템을 왜 써야하나요?

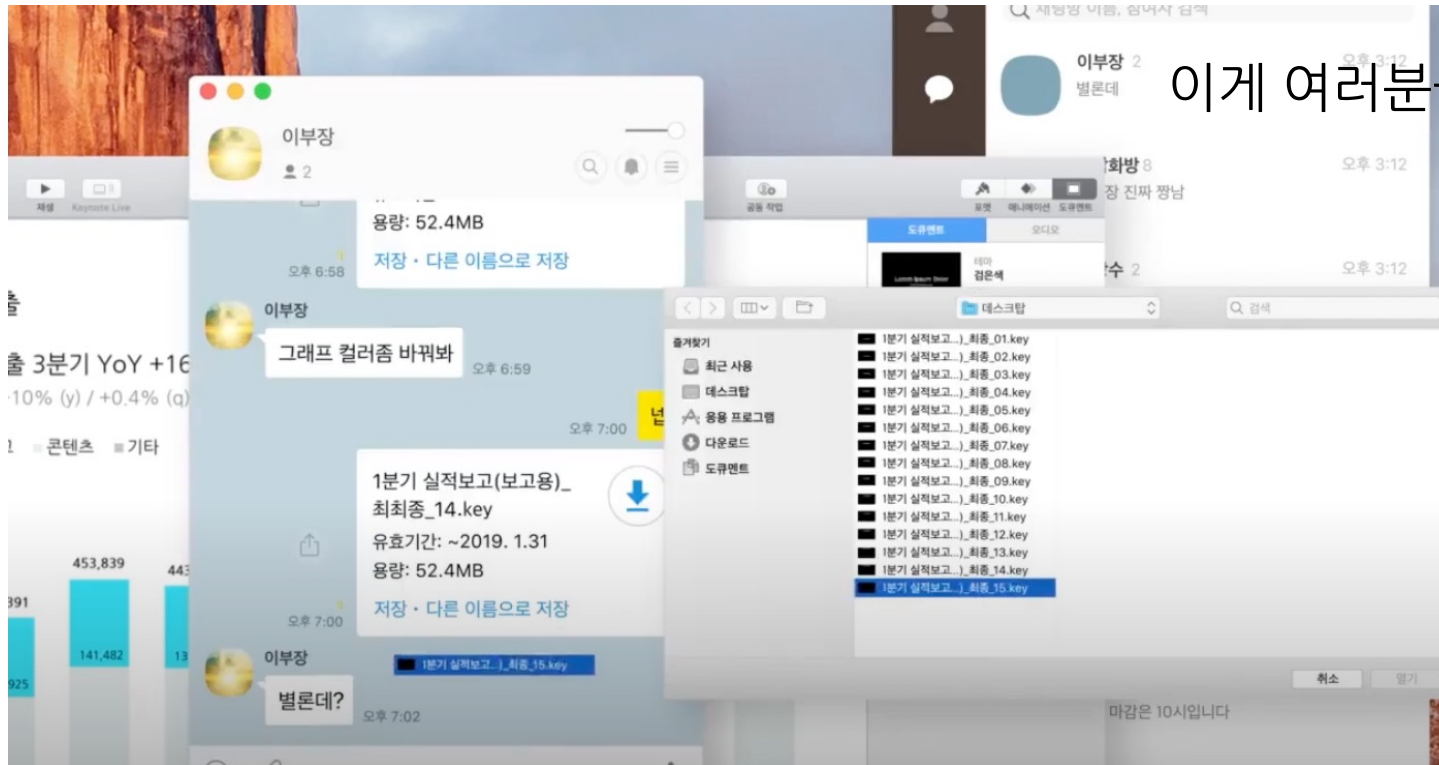


보고서 작성할때 이런 경험 없으신가요?

어떤 파일이 최종 파일이고, 언제 어떤 내용을 수정했는지 모르겠는 그런 ~~같은 상황...?

# Version Control System

버전 관리 시스템을 왜 써야하나요?



이게 여러분의 현실이 되지는 않겠죠?

개발한 소스코드를 설마,,, 카카오톡으로 공유하는건 아니겠죠?

# Version Control System

버전 관리 시스템을 왜 써야하나요?



서로 같은 프로젝트를 여러 명에서 개발하다보면  
당연히 충돌이 발생하겠죠...?

# Git

| 개발자들의 개비스콘 Git!



누가 언제 뭘 건드렸는지, 변경/수정사항을 기록하고  
되돌리기 기능을 지원하고,  
용량을 간소화하여 수정내역을 저장합니다.

무한 ctrl+Z는 이제 그만~

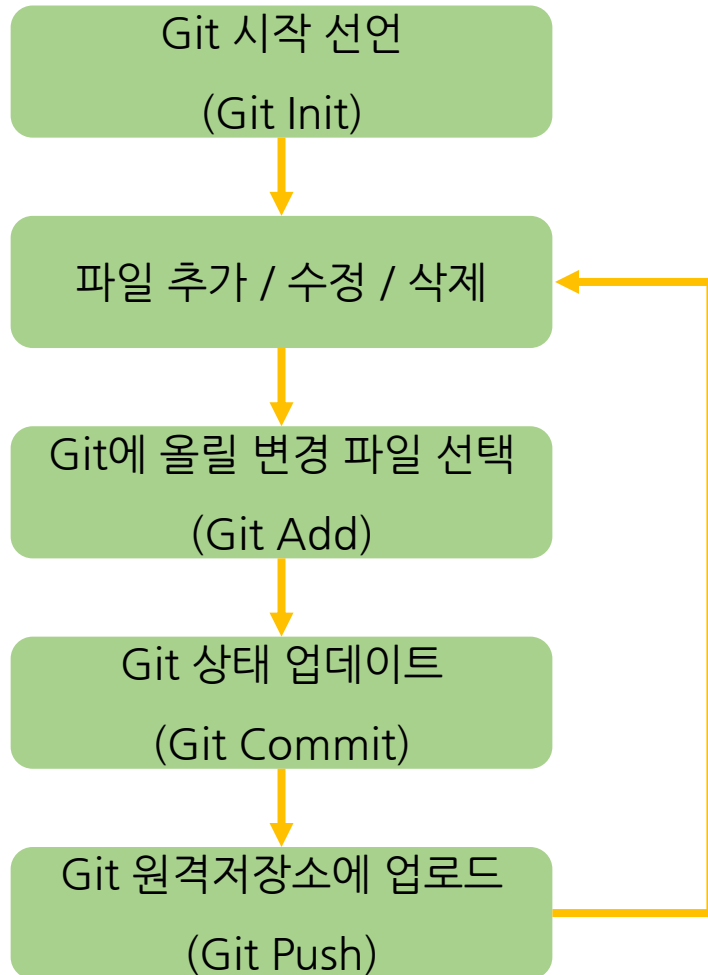
A silver laptop is open on a light-colored wooden desk. The background is a plain, light-colored wall. To the left of the laptop, there are some geometric wooden blocks. To the right, there is a small potted plant in a white square pot and a small geometric wooden structure. A semi-transparent white rectangular box is centered over the laptop screen, containing the title text. A solid orange horizontal line is positioned above the white box.

# Part 1. Git & Github Workflow



# 간단한 Git의 Flow

## | 아주 간단한 Git workflow



Git의 아주 간단한 기본 Workflow는 다음과 같습니다.

Git Init → Git Add → Git Commit → Git Push

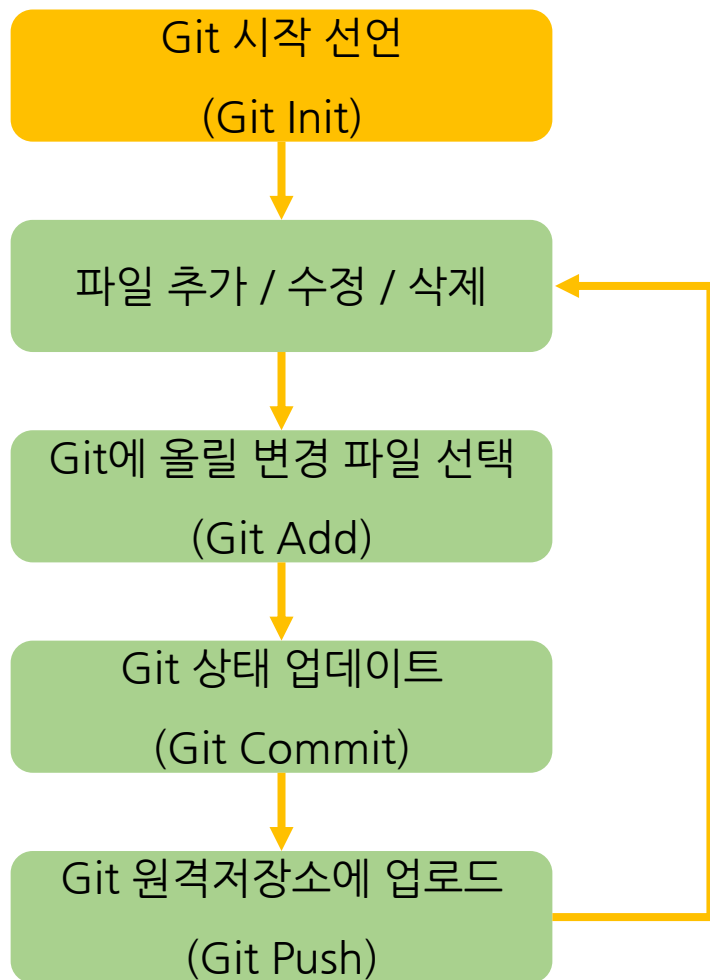
의 4단계에서 변형이 기본 동작이며

여기에서 branch, merge 등의 동작이 추가됩니다.

처음에는 이 4가지 동작만 잘 수행할 줄 아셔도  
좋습니다!

# 간단한 Git의 Flow

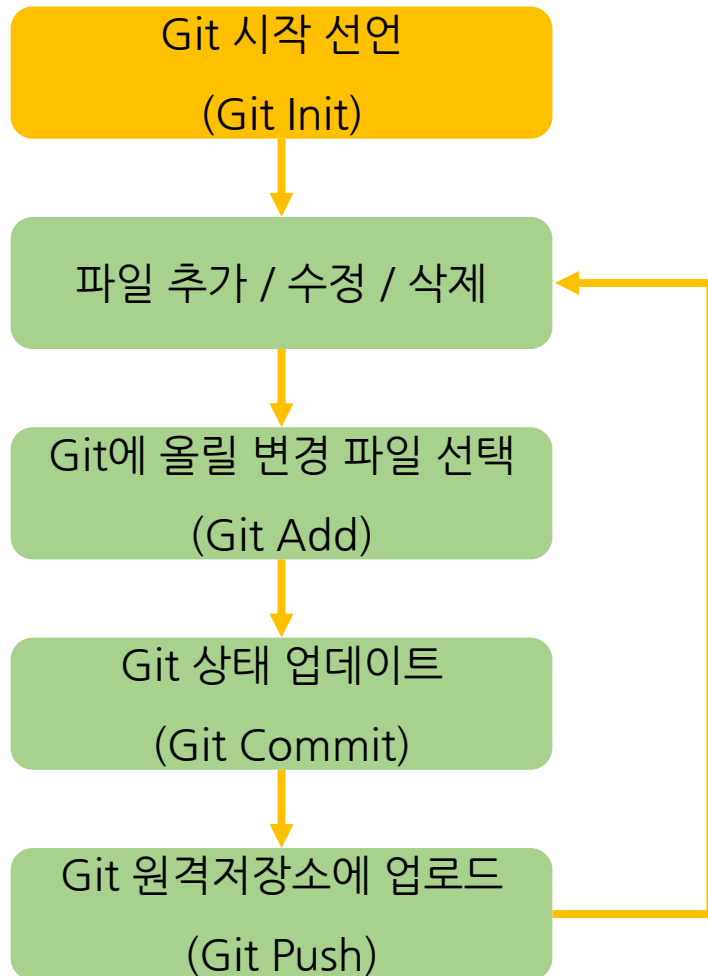
## | 아주 간단한 Git workflow



Git Init은 내 개발 디렉토리를  
Git으로 관리하겠다는 것을 선언하고  
원격저장소와 연결하는 작업을 수행하는 절차입니다.

# 간단한 Git의 Flow

## | 아주 간단한 Git workflow

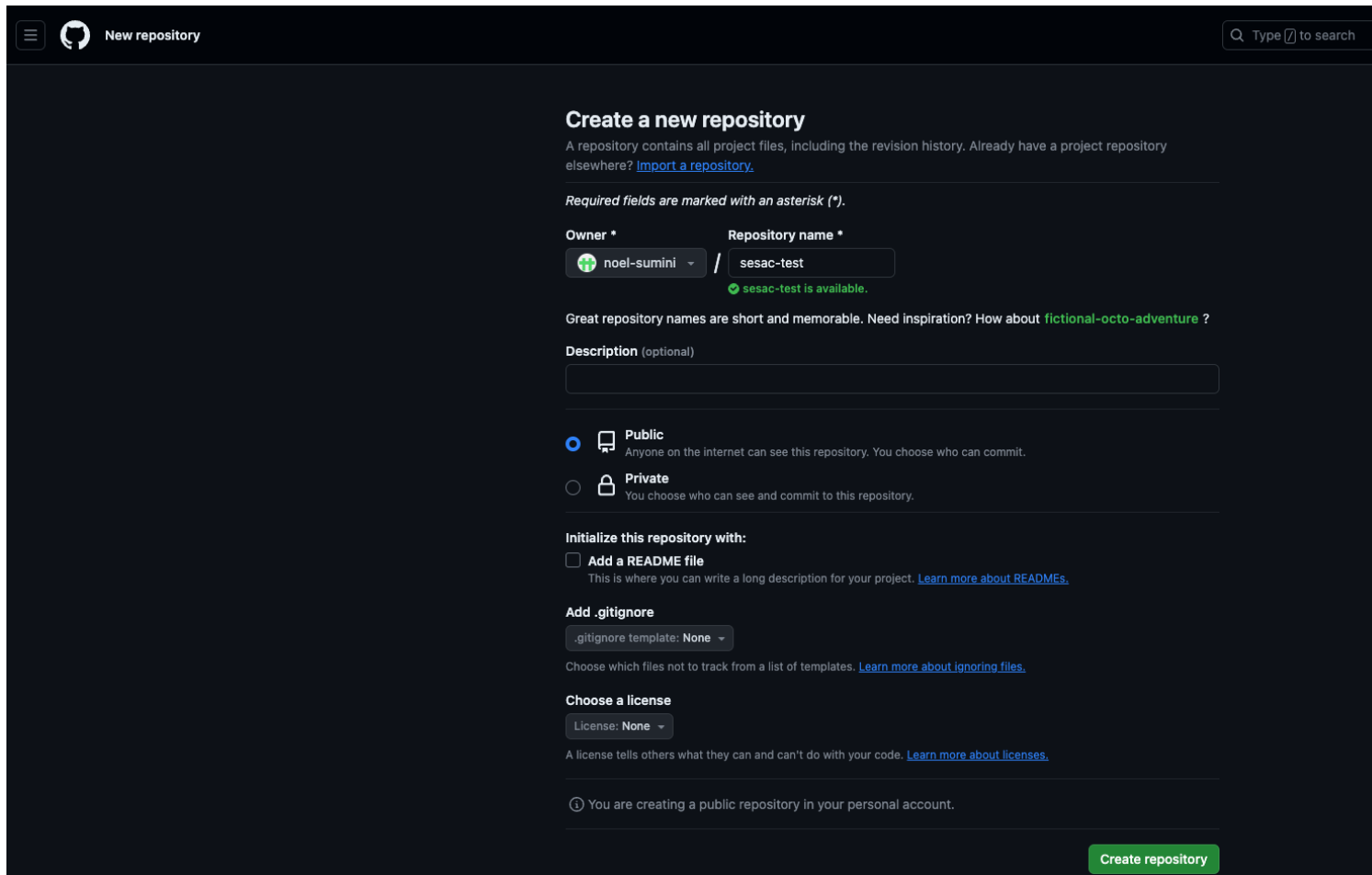


```
MINGW64:/c/work/project  
  
dedur@DESKTOP-9F838L4 MINGW64 /c/work/project  
$ pwd  
/c/work/project  
  
dedur@DESKTOP-9F838L4 MINGW64 /c/work/project  
$ git init  
Initialized empty Git repository in C:/work/project/.git/  
  
dedur@DESKTOP-9F838L4 MINGW64 /c/work/project (master)  
$
```

내 디렉토리에 git init을 수행하면  
Git으로 관리가 시작되며  
master branch가 생성됩니다.

# 간단한 Git의 Flow

## 아주 간단한 Git workflow



The screenshot shows the GitHub 'Create a new repository' page. At the top, there's a header with the GitHub logo and 'New repository' text. A search bar is on the right. The main content area is titled 'Create a new repository' and includes a sub-header 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this, a note states 'Required fields are marked with an asterisk (\*)'. The 'Owner' field is set to 'noel-sumini' and the 'Repository name' field is 'sesac-test', with a green checkmark indicating 'sesac-test is available.'. A tip suggests 'Great repository names are short and memorable. Need inspiration? How about [fictional-octo-adventure](#) ?'. The 'Description' field is optional and empty. Under 'Visibility', 'Public' is selected, with a note 'Anyone on the internet can see this repository. You choose who can commit.'. The 'Private' option is also visible. The 'Initialize this repository with:' section has 'Add a README file' checked, with a note 'This is where you can write a long description for your project. [Learn more about READMEs.](#)'. The '.gitignore' section shows a template of 'None'. The 'Choose a license' section shows a license of 'None'. A footer note states 'You are creating a public repository in your personal account.'. A green 'Create repository' button is at the bottom right.

Create a new repository

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

Required fields are marked with an asterisk (\*).

Owner \* Repository name \*

noel-sumini / sesac-test

sesac-test is available.

Great repository names are short and memorable. Need inspiration? How about [fictional-octo-adventure](#) ?

Description (optional)

☒ 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.

Initialize this repository with:

☒ Add a README file  
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None

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

Choose a license

License: None

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

You are creating a public repository in your personal account.

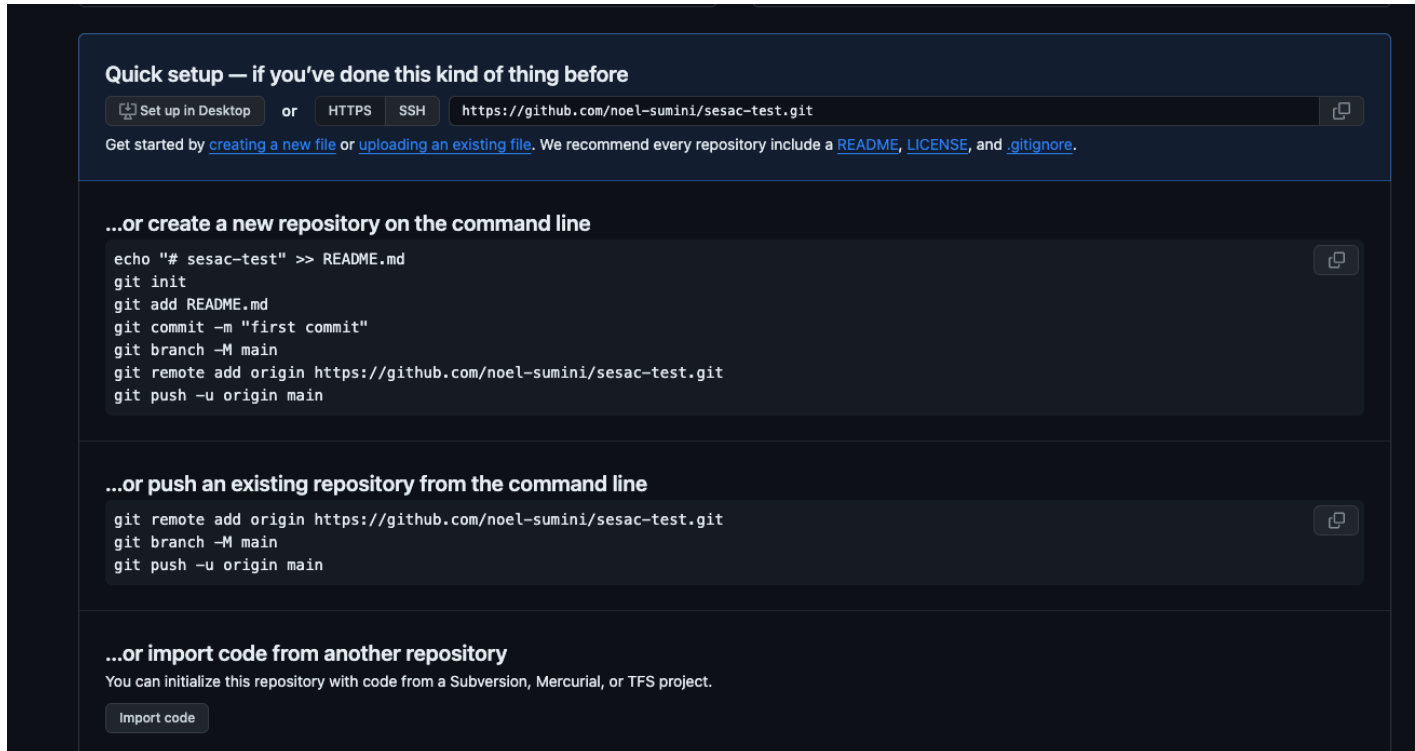
Create repository

Github에 새로운 repository를  
생성하고자 한다면  
이런 화면을 볼 수 있습니다.



# 간단한 Git의 Flow

## 아주 간단한 Git workflow

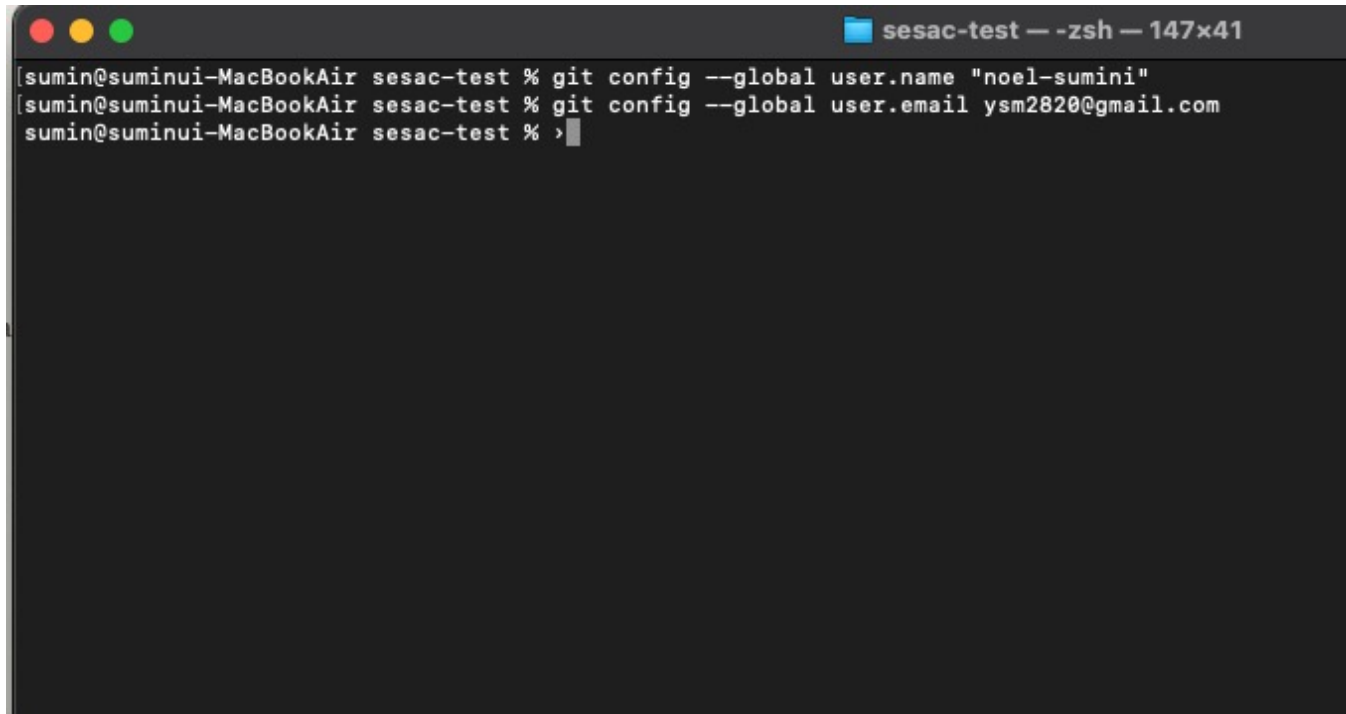


그리고 생성을 완료한다면  
이런 화면을 보실 수 있습니다.

저희는 이미 디렉토리에 git init을 수행했으므로  
해당 디렉토리에 존재하는 git에서 원격저장소인 github repository를 연결시켜주는 작업을 수행합니다.

# 간단한 Git의 Flow

## 아주 간단한 Git workflow

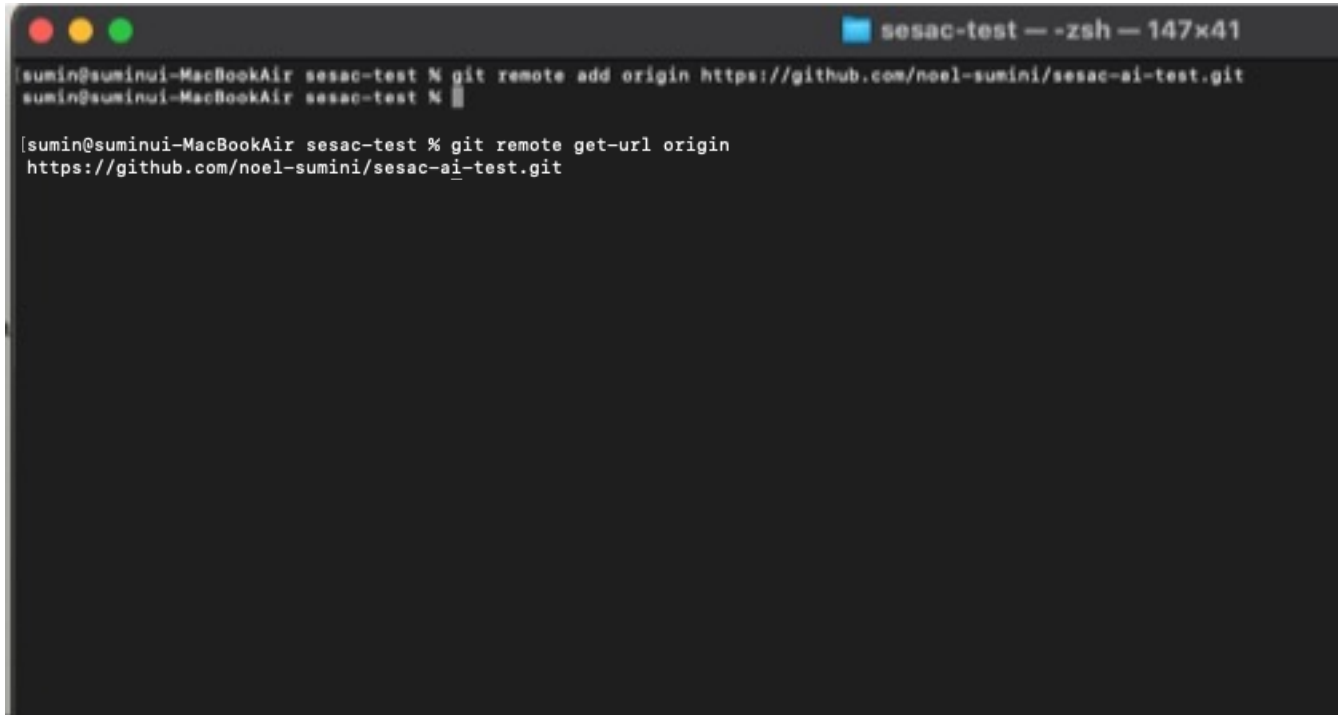
A terminal window titled 'sesac-test — zsh — 147x41' showing three lines of commands and their execution. The first line sets the global user name to 'noel-sumini', the second sets the global user email to 'ysm2820@gmail.com', and the third shows a prompt character '>' with a cursor.

```
[sumin@suminui-MacBookAir sesac-test % git config --global user.name "noel-sumini"]  
[sumin@suminui-MacBookAir sesac-test % git config --global user.email ysm2820@gmail.com]  
sumin@suminui-MacBookAir sesac-test % >
```

단, remote url을 등록하기 전에 꼭!!  
git config에 user name과 user email을 등록해주세요!!

# 간단한 Git의 Flow

## 아주 간단한 Git workflow

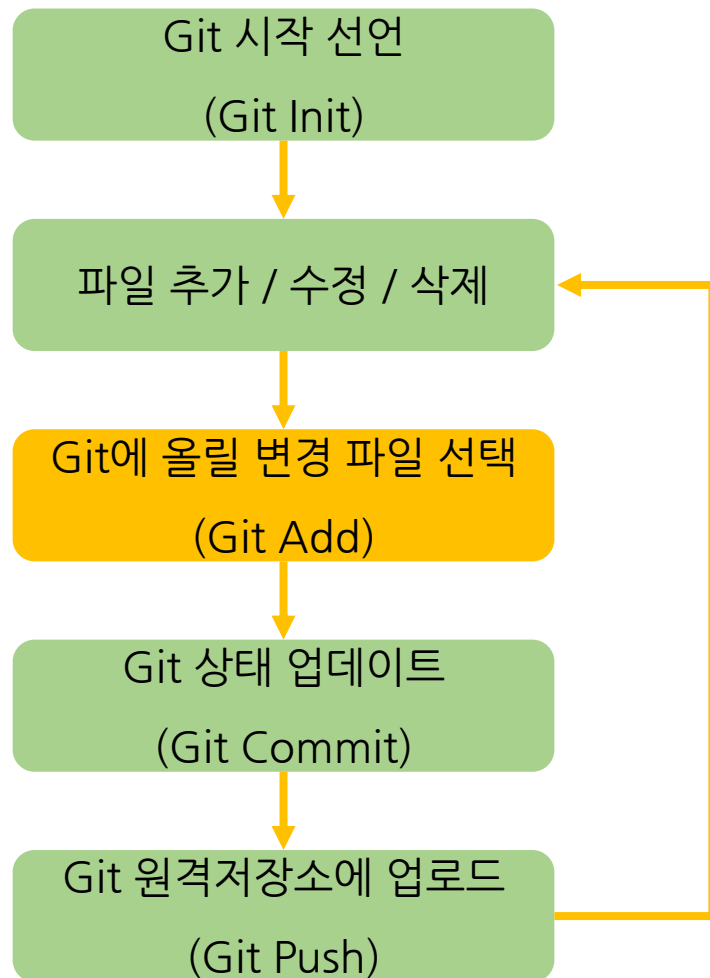
A terminal window titled 'sesac-test - zsh - 147x41' showing two Git commands being executed. The first command is 'git remote add origin https://github.com/noel-sumini/sesac-ai-test.git' and the second is 'git remote get-url origin', which outputs the same URL. The terminal background is dark with light-colored text.

```
[sumin@suminui-MacBookAir sesac-test % git remote add origin https://github.com/noel-sumini/sesac-ai-test.git
sumin@suminui-MacBookAir sesac-test % 
[sumin@suminui-MacBookAir sesac-test % git remote get-url origin
https://github.com/noel-sumini/sesac-ai-test.git
```

Remote add로 원격 repository 저장소를 등록하면  
내 git directory와 github 저장소가 연동이 된 상태가 됩니다.  
이제 수정사항을 원격저장소에 등록할 수 있습니다!!

# 간단한 Git의 Flow

## | 아주 간단한 Git workflow



## [ Git 의 File Tracking 3단계 ]

Unmodified

이전 버전과 비교하여  
수정한 부분이 없는 상태

Modified

이전 버전과 비교하여  
수정한 부분이 있는 상태

Staged

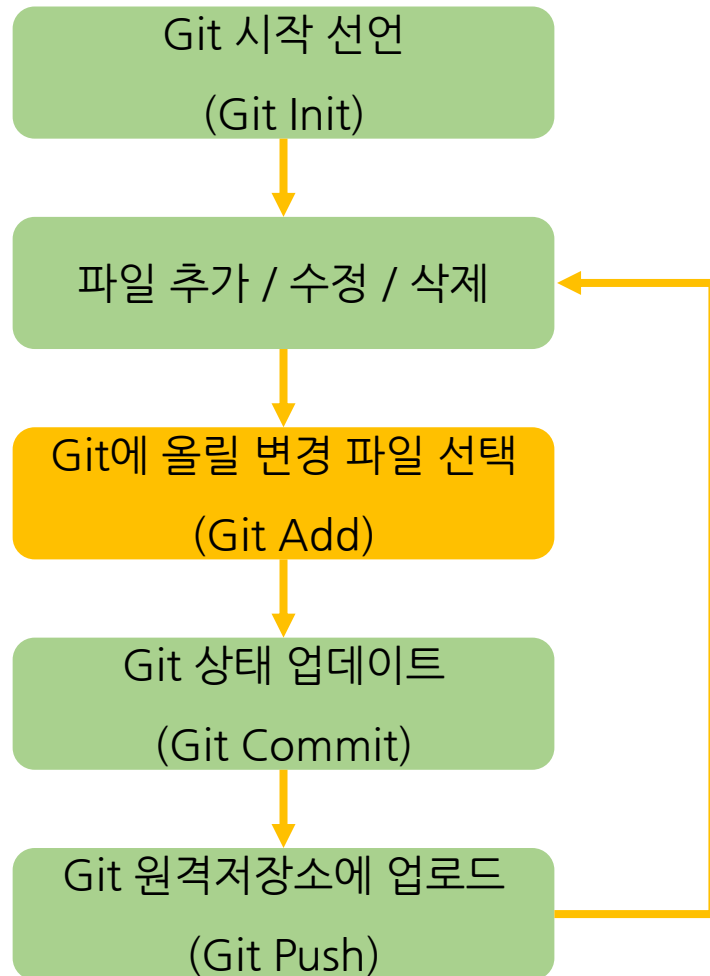
저장을 위해 준비된 상태  
(Commit 직전)

Git Add를 하게되면 staged 상태가 됩니다.



# 간단한 Git의 Flow

## 아주 간단한 Git workflow



```
sesac-test — zsh — 118x27
sumin@suminui-MacBookAir sesac-test % git add .
sumin@suminui-MacBookAir sesac-test % git status
On branch main

No commits yet

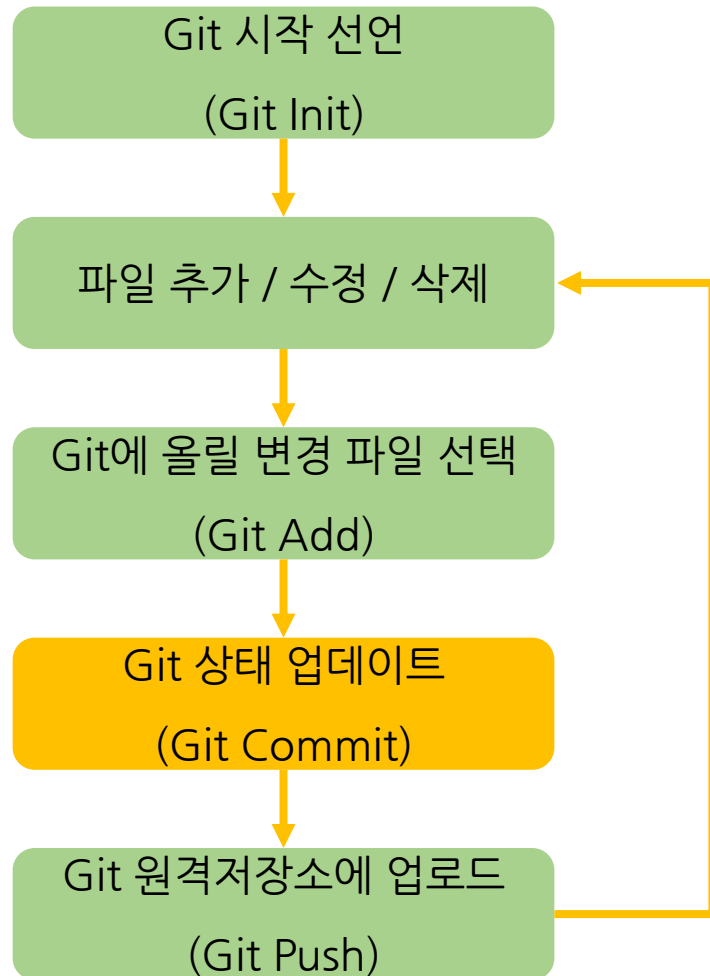
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
```

git add를 하고 git status 명령을 통해 상태를 확인해보면  
현재 commit된 내역은 없으며  
Commit을 위해 변화된 내역은 있다며  
New file을 알려줍니다.

Git add를 할때는 현재 디렉토리를 add하는  
( git add . ) 으로 add하거나  
File단위로 add할 수도 있습니다.

# 간단한 Git의 Flow

## | 아주 간단한 Git workflow



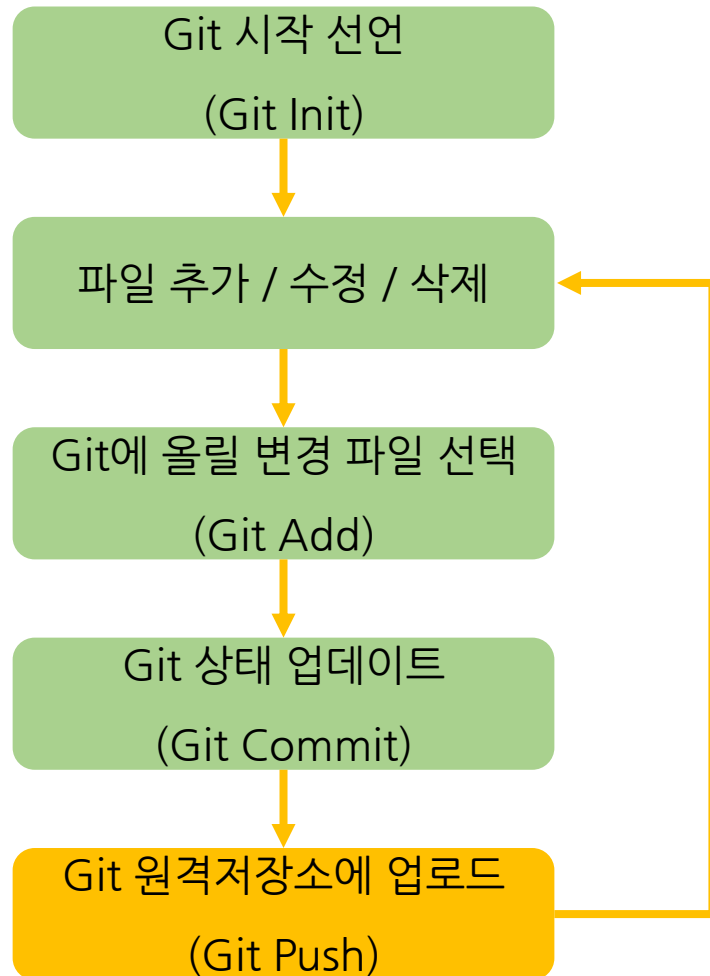
```
sesac-test — zsh — 118x27
sumin@suminui-MacBookAir sesac-test % git commit -m "first commit"
[main (root-commit) f886405] first commit
1 file changed, 1 insertion(+)
create mode 100644 README.md
sumin@suminui-MacBookAir sesac-test % git status
On branch main
nothing to commit, working tree clean
sumin@suminui-MacBookAir sesac-test %
```

Git Commit은 내 수정사항을 Git에 최종 등록하는 절차입니다.  
수정사항을 메시지형태로 함께 기록하며  
( `git commit -m "commit message"` )의 형태로 사용합니다.

Commit을 하면, 몇개의 파일이 수정되었고  
어떤 파일이 수정되었는지를 함께 보여줍니다.

# 간단한 Git의 Flow

## 아주 간단한 Git workflow



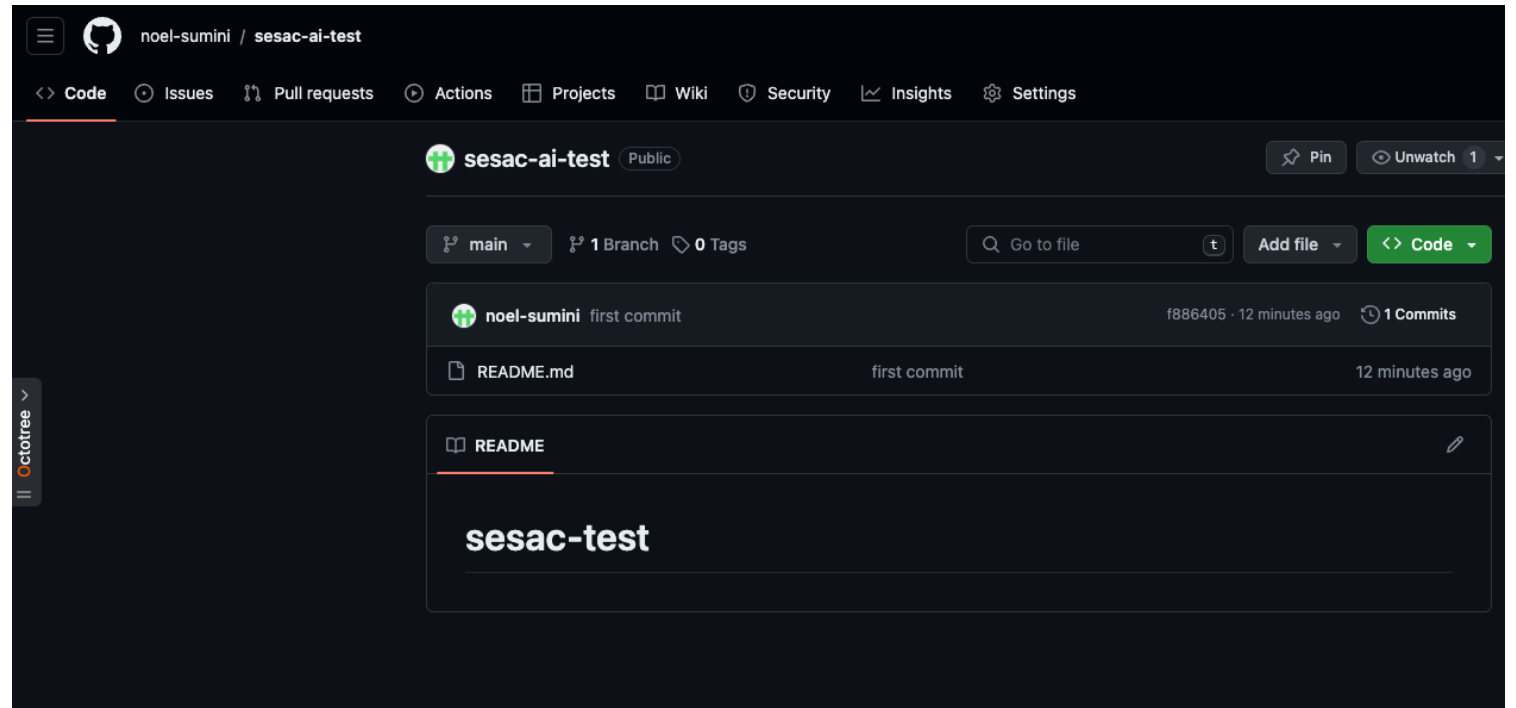
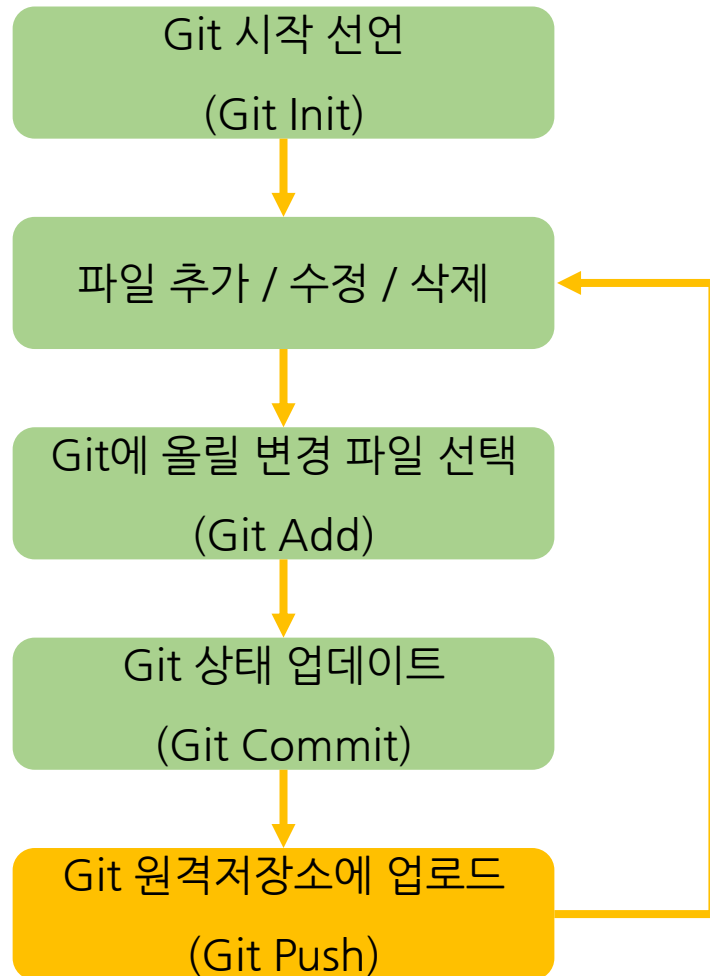
```
sesac-test — -zsh — 130x27
[sumin@suminui-MacBookAir sesac-test % git push origin main
Username for 'https://github.com': noel-sumini
[Password for 'https://noel-sumini@github.com':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 224 bytes | 224.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/noel-sumini/sesac-ai-test.git
 * [new branch]      main -> main
sumin@suminui-MacBookAir sesac-test %
```

git push를 하면 최초 push시에는 github user name과 password를 입력하라고 나옵니다.

다만, 21년 이후 password 인증방식이 사라져서 Token을 발급받아 입력하셔야합니다.

# 간단한 Git의 Flow

## 아주 간단한 Git workflow



Push를 완료한 후 github에 들어가보시면  
Commit한 이력이 보입니다!



# 간단한 Git의 Flow

## | Git 버전을 과거 버전으로 되돌려봐요!

```
sesac-test — -zsh — 130x27
[sumin@suminui-MacBookAir sesac-test % git log
commit 2ea169905b3d532a852e21e2d0c77cfd0e7eb4d9 (HEAD, origin/main, main)
Author: noel-sumini <ysm2820@gmail.com>
Date: Sat Dec 30 03:47:16 2023 +0900

    second commit

commit f886405650d49ef2f511a389bef5c65f3ddbccc34
Author: noel-sumini <ysm2820@gmail.com>
Date: Sat Dec 30 03:34:00 2023 +0900

    first commit
[sumin@suminui-MacBookAir sesac-test % git reset f886405650d49ef2f511a389bef5c65f3ddbccc34 --mixed
Unstaged changes after reset:
M README.md
[sumin@suminui-MacBookAir sesac-test % cat README.md
# sesac-test22
[sumin@suminui-MacBookAir sesac-test % git log
commit f886405650d49ef2f511a389bef5c65f3ddbccc34 (HEAD)
Author: noel-sumini <ysm2820@gmail.com>
Date: Sat Dec 30 03:34:00 2023 +0900

    first commit
```

- mixed option : 리셋 이후의 커밋 내용들이 unstage 상태로 존재
- soft option : 리셋 이후의 커밋 내용들이 stage 상태로 존재
- hard option : 리셋 이후의 커밋 내용들이 전부 삭제

Git Reset 명령을 사용하면 과거 commit으로  
Commit 이력을 되돌릴 수 있습니다.

# 간단한 Git의 Flow

## 아주아주 간단한 Git Workflow



여기까지가 혼자 개발을 위한  
Git / Github 사용이었습니다.

이제는 협업을 위한 Git 사용을 배워볼 차례입니다.

A silver laptop is open on a light-colored wooden desk. A semi-transparent white rectangular box is centered over the laptop screen, containing the title text. A solid yellow horizontal line is positioned above the white box. In the background, to the left of the laptop, are three small, modern, geometric wooden sculptures. To the right, there is a small potted plant with green leaves in a white square pot, and a small, clear, geometric wooden object.

## Part 2. 협업을 위한 Git/Github workflow

# 협업을 위한 Git/Github Workflow

| Git에서 현재 commit version을 내려받자

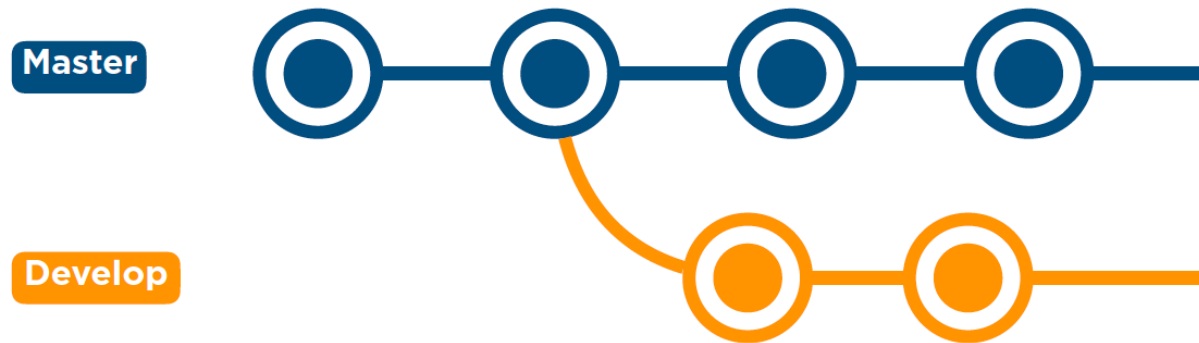
```
[sumin@suminui-MacBookAir sesac-test % git pull origin develop
From https://github.com/noel-sumini/sesac-ai-test
 * branch          develop      -> FETCH_HEAD
Updating 269d010..bd93831
Fast-forward
 README.md | 1 -
 1 file changed, 1 deletion(-)
```

Git에서 현재 버전을 내려받아서 작업을 이어나가야 conflict을 막을 수 있습니다.

Git에서 현재 버전을 내려받는 절차를 pull이라고 하며  
“git pull [remote저장소] [branch명]”으로 내려받습니다.

# 협업을 위한 Git/Github Workflow

Branch를 만들어보자!



물론 hotfix patch등의 이슈해결에도  
branch을 새로 따서 작업합니다.

일반적으로 IT환경에서는  
운영서버와 개발서버가 존재합니다.

검증되지 않은 코드를 그대로 개발환경에  
올리면 Risk가 매우매우 크기 때문에  
개발환경에서 테스트를 거쳐 운영서버에  
배포합니다.

운영환경의 코드를 관리하는 Branch가  
Master branch (Production Branch)  
개발환경의 코드를 관리하는 Branch가  
Develop branch 입니다.



# 협업을 위한 Git/Github Workflow

Branch를 만들고, Branch를 전환해보자

```
sumin@suminui-MacBookAir sesac-test % git branch develop
sumin@suminui-MacBookAir sesac-test % git branch -v
* (HEAD detached from 2ea1699) b7f0810 second commit
   develop                    b7f0810 second commit
   main                      2ea1699 second commit
sumin@suminui-MacBookAir sesac-test % git checkout develop
Switched to branch 'develop'
sumin@suminui-MacBookAir sesac-test %
```

git branch [branch명]

git checkout [branch명]

git branch 명령을 통해 develop branch를 새로 생성하였고,  
git checkout 명령을 통해 branch를 전환하였습니다.

# 협업을 위한 Git/Github Workflow

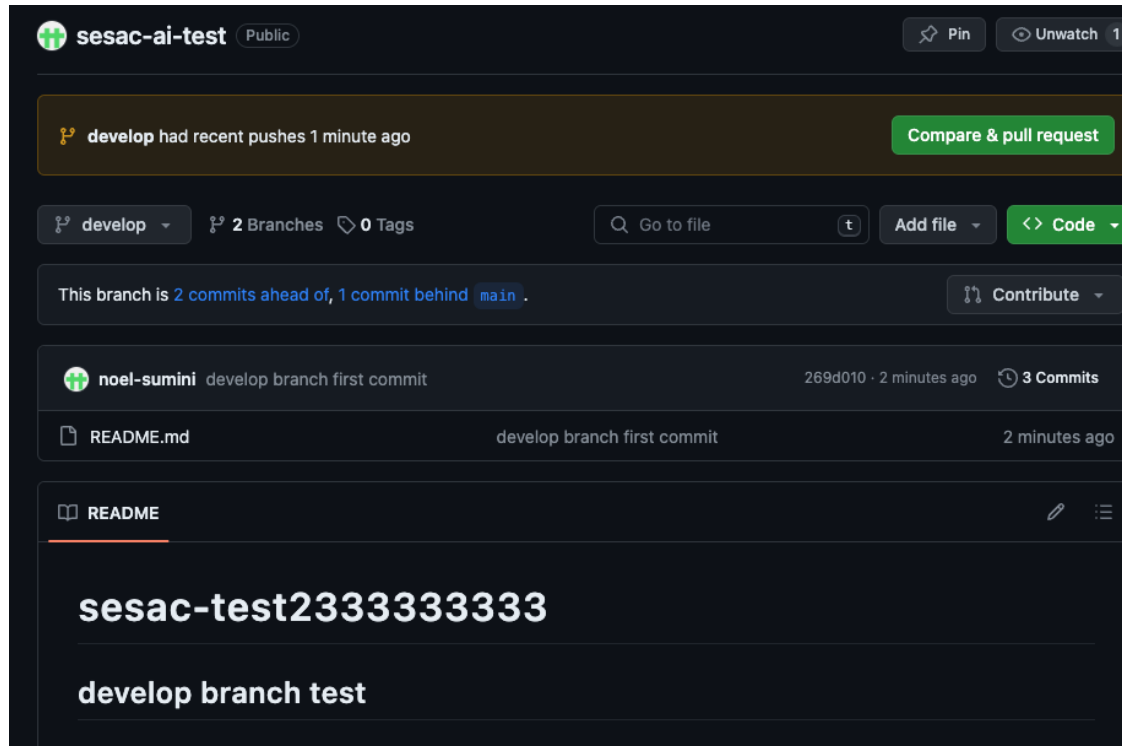
| 새로운 branch에 commit / push해보자!

```
sumin@suminui-MacBookAir sesac-test % git add .
sumin@suminui-MacBookAir sesac-test % git commit -m "develop branch first commit"
[develop 269d010] develop branch first commit
 1 file changed, 2 insertions(+), 1 deletion(-)
sumin@suminui-MacBookAir sesac-test % git push origin develop
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (6/6), 513 bytes | 513.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'develop' on GitHub by visiting:
remote:   https://github.com/noel-sumini/sesac-ai-test/pull/new/develop
remote:
To https://github.com/noel-sumini/sesac-ai-test.git
 * [new branch]      develop -> develop
sumin@suminui-MacBookAir sesac-test %
```

이제는 main branch에 push하는게 아닌 develop branch에 push 해야하므로,  
Push할때 이점에 유의해주세요!!

# 협업을 위한 Git/Github Workflow

새로운 branch에 commit / push해보자!



github에 가보면 "develop had recent pushes"라고 새로운 push내역이 있음을 알려주고,  
develop branch에 가보면 새로운 commit 이력을 확인할 수 있습니다.

# 협업을 위한 Git/Github Workflow

## Pull Request

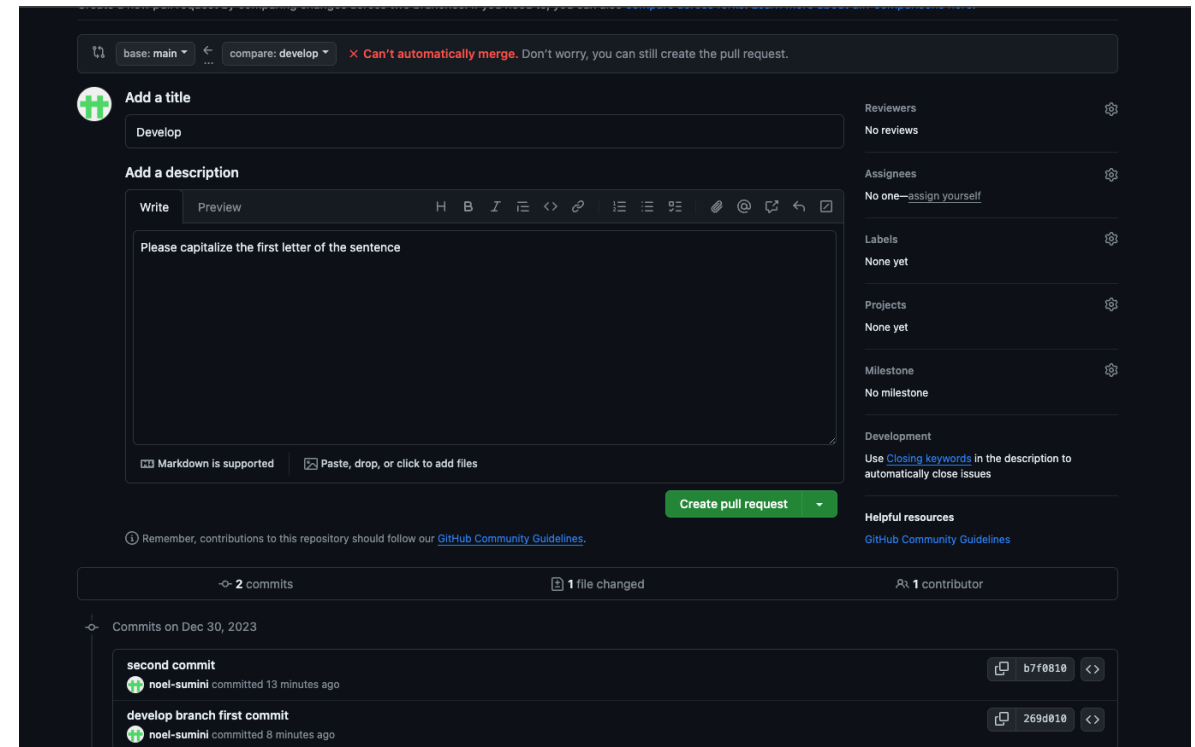
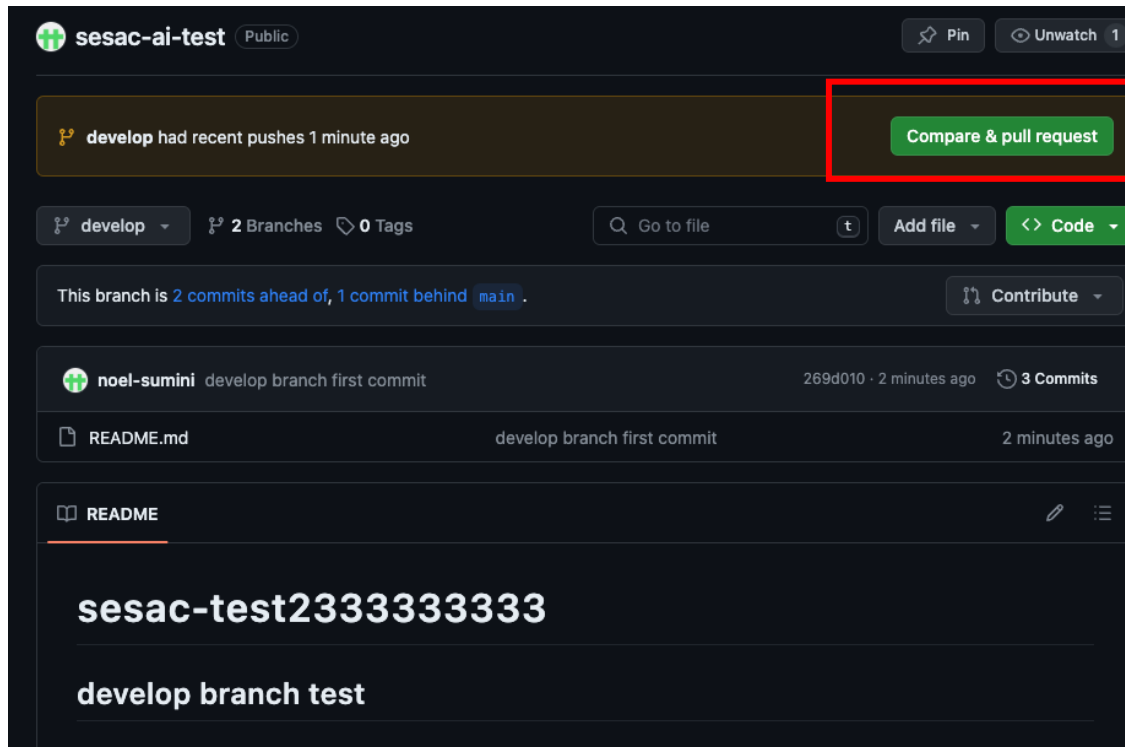


운영 중인 Branch에 내가 Develop branch에  
commit했던 이력을 반영하는 절차를  
Branch를 merge한다고 합니다.  
(Dev branch → Main branch merge)

다만, 무지성으로 main branch에  
merge하는 것을 막고  
Code quality를 확보하기 위해  
Merge 전 Pull Request(PR)절차를 거쳐  
Code Review를 수행 후  
Approve가 되면  
Main Branch에 Merge를 하게됩니다.

# 협업을 위한 Git/Github Workflow

## Pull Request



Code review의 내용을 message로 남겨놓을 수 있습니다.

# 협업을 위한 Git/Github Workflow

## Merge

```
[sumin@suminui-MacBookAir sesac-test % git add .
[sumin@suminui-MacBookAir sesac-test % git commit -m "develop branch second commit"
[develop d8ea9d6] develop branch second commit
 1 file changed, 1 insertion(+), 1 deletion(-)
[sumin@suminui-MacBookAir sesac-test % git push origin develop
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 273 bytes | 273.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/noel-sumini/sesac-ai-test.git
   bd93831..d8ea9d6  develop -> develop
[sumin@suminui-MacBookAir sesac-test % git checkout main
Switched to branch 'main'
[sumin@suminui-MacBookAir sesac-test % git merge develop
Updating 2ea1699..d8ea9d6
Fast-forward
 README.md | 2 +--
 1 file changed, 1 insertion(+), 1 deletion(-)
sumin@suminui-MacBookAir sesac-test %
```

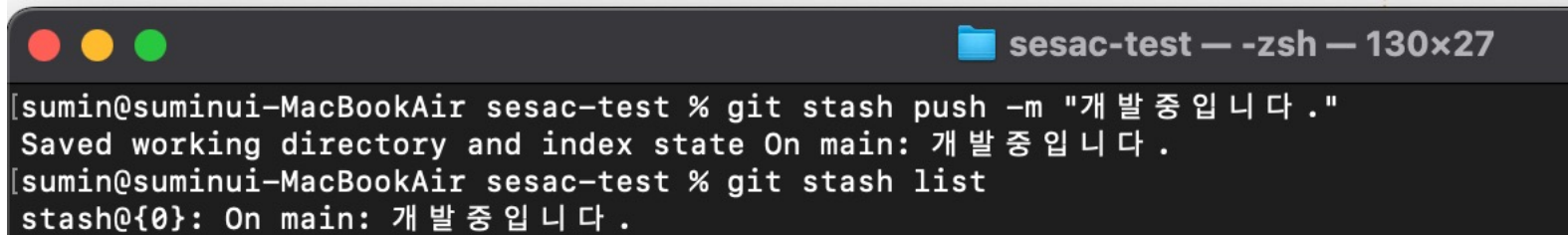
Develop Branch → Main branch로 merge하기 위해서는

Checkout 명령을 통해 다시 Main branch로 이동한 후

“git merge develop” 명령을 통해 develop branch의 변경내역을 main branch에 merge해줘야 합니다.

# 협업을 위한 Git/Github Workflow

## Stash

A terminal window titled 'sesac-test — -zsh — 130x27' showing the execution of git stash commands. The first command is 'git stash push -m "개발 중입니다."' which outputs 'Saved working directory and index state On main: 개발 중입니다.'. The second command is 'git stash list' which outputs 'stash@{0}: On main: 개발 중입니다.'.

```
[sumin@suminui-MacBookAir sesac-test % git stash push -m "개발 중입니다."]
Saved working directory and index state On main: 개발 중입니다.
[sumin@suminui-MacBookAir sesac-test % git stash list
stash@{0}: On main: 개발 중입니다.
```

`git stash push -m "save message"`

현재 작업중인 개발내역이 아직 commit하기는 부족한데,  
운영 중인 코드에 bug가 생겨서 해결해야하는 등  
Branch전환이 필요한 상황일 경우 따로 임시저장 해두는게 필요할 수 있습니다.  
이런 경우 stash 기능을 사용합니다.



# 협업을 위한 Git/Github Workflow

## Stash

```
[sumin@suminui-MacBookAir sesac-test % git stash apply stash@{0}
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
sumin@suminui-MacBookAir sesac-test %
```

`git stash apply stash@{index}`

Stash apply를 하면 stash 내역을 가져올 수 있습니다.

이때, unstaged상태로 가져오므로 git add / commit이 필요합니다.