

Git Tutorial

1. Git 소개

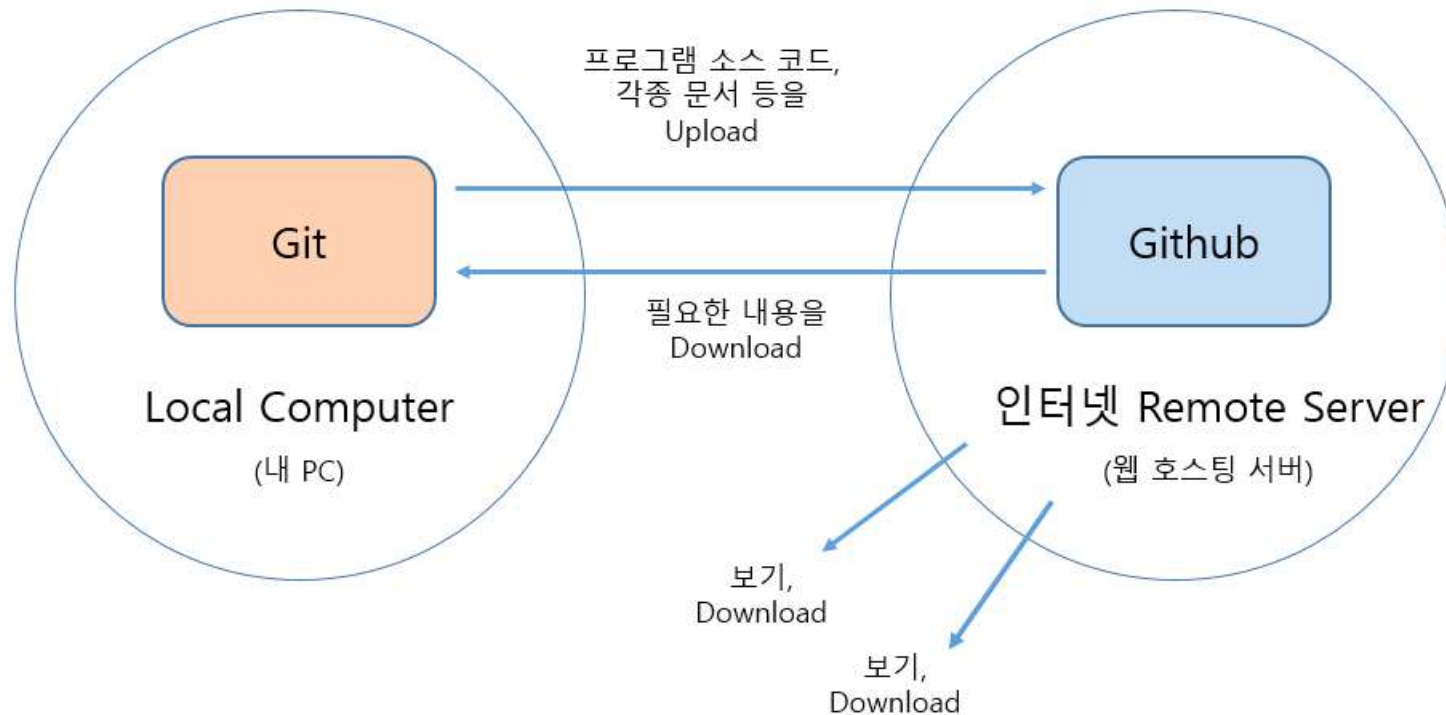
- Git은 전세계적으로 가장 널리 사용되는 버전 관리 시스템(Version Control System)입니다.
- 2005년에 Linus Torvalds에 의해 만들어졌으며, 그 이후로 Junio Hamano에 의해 유지 관리하고 있습니다.
- 다음 용도로 주로 사용됩니다
 - 코드 변경 추적
 - 코딩 협업
 - 누가 변경했는지 추적 (코딩 협업 시)
 - 프로그램 코드 및 자료의 Backup & 공유 (현재 가장 많은 용도로 사용되고 있음)

◆ Git으로 무엇을 할 수 있나?

- 리포지토리(Repositories)로 프로젝트 관리
- 스테이징(Staging) 및 커밋(Committing) 으로 변경 사항 제어 및 추적
- 브랜치(Branch) 및 병합(Merge)을 통해 프로젝트의 다른 버전 작업 가능
- 로컬 리포지토리를 GitHub로 푸시한다(Push)
- Github에 있는 프로젝트의 최신 버전을 로컬 복사본으로 가져온다.(Pull)
- GitHub에 있는 프로젝트를 로컬 복사본에서 작업하기 위해 프로젝트를 복제(Clone)

◆ Github란?

- 분산 버전 관리 툴인 Git 저장소(Repository)를 호스팅하기 위해 지원하는 웹 서비스입니다.
- Github는 영리적인 서비스와 오픈소스를 위한 무상 서비스를 모두 제공합니다.
- Git과 GitHub는 다릅니다
- GitHub는 세계에서 가장 큰 소스 코드 호스트이며 2018년부터 Microsoft가 소유하고 있습니다.
- 본 학습서에서는 GitHub와 함께 Git을 사용하는 데 중점을 두기로 합니다.



◆ Why Git?

- 전세계 70% 이상의 개발자가 Git을 사용!
- 개발자는 전 세계 어디에서나 함께 작업 가능
- 개발자는 프로젝트의 전체 기록을 볼 수 있음
- 개발자는 프로젝트의 이전 버전으로 되돌릴 수 있음

2. Git 시작하기

◆ Git 설치하기

- 다음 웹사이트에서 Git을 무료로 다운로드하여 설치할 수 있습니다. <https://www.git-scm.com/>

◆ Command Line에서 Git 사용하기

- Git 사용을 시작하려면 먼저 Windows 명령 프롬프트를 엽니다. (Windows 왼쪽아래 검색창에서 cmd를 입력합니다)
- Windows의 파일탐색기에서 폴더를 선택하고, 오른쪽 버튼을 눌러 Windows용 Git에 포함된 Git bash를 사용할 수 있습니다
- 가장 먼저 해야 할 일은 Git이 제대로 설치되었는지 확인하는 것입니다.

```
$ git --version
```

- Git이 제대로 설치된 경우, 다음과 같이 표시되어야 한다. `git version X.Y`

◆ Git 구성 (Configure)

- 이제 Git에게 당신이 누구인지 알려주어야 합니다.
- 나중에 사용할 각 Git commit 명령어가 다음의 정보를 사용하기 때문에, 이것은 매우 중요합니다. (나중에 GitHub를 사용할 때, 같은 id와 email을 사용하는 것이 좋음)

```
$ git config --global user.name "사용자 id"
$ git config --global user.email "email 주소"
```

*** 참고 :**

컴퓨터의 모든 레포지토리에서 동일한 id와 email을 사용하려면, 위와 같이 --global 옵션을 사용하고, 특정 레포지토리에서만 사용하려면 --global 옵션을 빼면 됩니다.

- 설정된 내용을 확인할 경우에는 아래의 명령어를 사용합니다.

```
$ git config --list          # 전체 항목 확인
$ git config --global 항목   # 항목 하나만 확인
```

- 또한, 설정된 config 항목을 삭제하기 위해서는 아래의 명령어를 사용합니다.

```
$ git config --unset 항목          # local 항목
$ git config --unset --global 항목 # global 항목
```

◆ 폴더에 Git 적용 하기

- 프로젝트 폴더에 Git을 적용할 폴더로 이동합니다. 여기서는 앞에서 생성한 D: 파티션에 있는 webdevelop 폴더에 적용해 보겠습니다.

```
$ d:  
$ cd academy\swbae\webdevelop
```

* 참조 :

Windows에서는 Git을 적용할 폴더가 이미 생성되어 있을 경우, 파일 탐색기에서 오른쪽 마우스 버튼을 클릭하고, 'Git Bash Here'를 선택하면, Git 명령어를 입력할 수 있는 Bash Shell이 나타난다

◆ Git 초기화(Initialize)

- 초기화하고자 하는 폴더로 이동하여 아래의 명령어를 입력합니다.

```
$ git init
```

- 그러면 git을 이용할 수 있는 준비가 끝났습니다.
- git 초기화를 실행하면, 해당 폴더에 .git 이라는 숨김 폴더가 하나 생성됩니다.
- 이 폴더를 이용하여 git은 모든 정보를 저장하고 처리합니다.

3. Git 하기

◆ 새 File을 Git Add 하기

- 앞에서 local Git 레포지토리(webdevelop)를 하나 지정하였습니다. (\$ Git init 사용)
- webdevelop 폴더에 아래의 파일을 하나 생성합니다. (우선 메모장을 사용함)

webdevelop/intex.html

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
</head>
<body>

<h1>Hello world!</h1>
<p>This is the first file in my new Git Repo.</p>

</body>
</html>
```

- 파일이 잘 만들어졌는지, webdevelop 폴더에서 아래의 명령으로 확인한다 (명령 프롬프트에서 혹은 파일 탐색기에서)


```
$ dir
```

- 그 다음, 아래의 명령으로 현재의 Git 상태를 한번 확인해 보겠습니다.

```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add ..." to include in what will be committed)
```

```
index.html
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

- 위 결과를 보면, Git은 index.html 파일을 인식은 하지만 아직 레포지토리에 추가 하지는 않았다고 메시지가 나타납니다.
- Git 리포지토리 폴더의 파일은 다음 두 가지 상태 중 하나일 수 있습니다
 - Tracked(추적됨) - Git이 인식하고 레포지토리에 추가된 파일
 - Untracked(추적되지 않음) - 작업 디렉토리에 있지만 레포지토리에 추가되지 않은 파일
- 빈 저장소(Repository)에 파일을 처음 추가하면 Untracked 상태가 됩니다. Git에서 추적(Tracked)하도록 하려면 스테이징하거나 스테이징 환경에 추가해야 합니다.

4. Git 스테이징 환경(Staging Environment)

◆ 스테이징 환경에 추가하기

- Git의 핵심 기능 중 하나는 스테이징 환경(Staging Environment)과 커밋(Commit)의 개념입니다.
- 작업하면서 파일을 추가, 편집 및 제거할 수 있습니다.
- 그러나 특정 작업진도(Milestone)에 도달하거나 작업의 일부를 완료할 때마다 스테이징 환경에 파일을 추가해야 합니다.
- 스테이징(Staged)된 파일은 작업중인 리포지토리에 커밋(Commit) 할 준비가 된 파일입니다.
- 지금까지는 index.html에 대한 중간 작업을 마친 단계라고 가정합니다.
- 그래서 스테이징 환경에 추가해 보기위해 아래의 명령을 입력합니다.

```
$ git add index.html
```

- 이제 index.html 파일이 스테이징 되었습니다. git status 명령어로 상태를 확인해 보면,

```
$ git status  
  
On branch master  
  
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached ..." to unstage)
```

```
new file: index.html
```

◆ 새로운 파일을 두개 더 만들기

- 아래 두개의 파일을 myproject 폴더에 새로 생성한다

README.md

```
# hello-world
```

```
Hello World repository for Git tutorial
```

```
This is an example repository for the Git tutoial on https://www.w3schools.com
```

```
This repository is built step by step in the tutorial.
```

bluestyle.css

```
body {
```

```
background-color: lightblue;
```

```
}
```

```
h1 {
```

```
color: navy;
```

```
margin-left: 20px;
}
```

- 그리고, 기존의 index.html 파일을 아래와 같이 수정한다

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<link rel="stylesheet" href="bluestyle.css">
</head>
<body>

<h1>Hello world!</h1>
<p>This is the first file in my new Git Repo.</p>

</body>
</html>
```

참고 : 지금의 상태를 'git status' 로 확인해 보자

- 위의 3개의 파일들을 한번에 스테이징 하기 위해 아래의 명령을 실행합니다

```
$ git add --all      # 혹은 $ git add -A
```

- 이제 Git 상태를 확인해 보겠습니다.

```
$ git status

On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached ..." to unstage)

    new file:   README.md
    new file:   bluestyle.css
    new file:   index.html
```

- 이렇게 해서 3개의 file을 모두 스테이징 환경에 추가하였다. 이제 Commit 할 준비가 되었습니다.

참고 : 'git add --all' 명령대신 'git add -A'를 실행해도 똑 같음

5. Git Commit

◆ Commit 하기

- 앞에서 지금까지의 작업을 완료했기 때문에, 3개의 파일을 스테이징 환경으로 옮겼고, 이제는 커밋을 할 단계입니다.
- 커밋을 추가하게 되면, 작업 진행 상황과 변경 사항을 추적하거나, 특정 커밋 시점으로 작업을 되돌릴 수 있습니다.
- Git은 각 커밋을 각 파일들의 변경에 대한 "Save Point"로 저장하고 관리합니다.
- 다시 말해서 필요 시, 특정 커밋 시점으로 프로젝트를 되 돌릴수 있는 Point가 됩니다.
- 커밋할 때에는 반드시 메시지를 포함해야 합니다.
- 각 커밋에 명확한 메시지를 추가하면 변경된 내용과 시기를 쉽게 확인할 수 있습니다.
- 아래의 명령으로 Commit을 실행합니다.

```
$ git commit -m "First release of Hello World!"  
[master (root-commit) 221ec6e] First release of Hello World!  
3 files changed, 26 insertions(+)  
create mode 100644 README.md  
create mode 100644 bluestyle.css  
create mode 100644 index.html
```

- `git commit -m "message"` 로 커밋을 실행하며 메시지를 추가합니다.
- 지금까지의 결과로 스테이징 환경은 위와 같은 메시지와 함께 레포지토리 커밋 되었습니다.

- 'git status'를 이용하여 지금까지의 git 상태를 확인해 봅니다.

◆ index.html 파일을 수정하고, 다시 Commit 하기

- 새로운 커밋을 실행하기 위해 index.html 파일을 아래와 같이 수정하고 저장 합니다.

index.html

```
<!DOCTYPE html>
<html>
<head>
<title>Hello World!</title>
<link rel="stylesheet" href="bluestyle.css">
</head>
<body>

<h1>Hello world!</h1>
<p>This is the first file in my new Git Repo.</p>
<p>A new line in our file!</p>

</body>
</html>
```

- 파일을 저장한 상태에서 'git status'로 상태를 확인해 봅니다.
- 다시 커밋 하기 위해 아래의 명령어를 실행합니다.

```
$ git add -A
$ git commit -m "index.html Updated"
[master 09f4acd] index.html Updated
1 file changed, 1 insertion(+)
```

- 새로운 변경사항이 커밋되었습니다.

◆ Commit Log 확인하기

- 레포지토리의 커밋 히스토리를 확인하기 위해, 'git log' 명령어를 사용할 수 있습니다.
- 이를 통해 커밋한 사용자와 커밋 시점을 확인할 수 있습니다.

```
$ git log
commit 09f4acd3f8836b7f6fc44ad9e012f82faf861803 (HEAD -> master)
Author: swbae01 <swbae77@naver.com>
Date:   Fri Mar 26 09:35:54 2021 +0100

    index.html Updated
```



```
commit 221ec6e10aeedbfd02b85264087cd9adc18e4b26
```

```
Author: swbae01 <swbae77@naver.com>
```

```
Date:   Fri Mar 26 09:13:07 2021 +0100
```

```
    First release of Hello World!
```

6. Git Commit 취소, 되돌리기, 덮어쓰기

◆ git reset : 커밋 취소하기

- 다음과 같이 97f1e31 -> d678197 -> 12741e5 차례로 커밋을 했다고 가정하면

```
$ git log --oneline
12741e5 (HEAD -> master) Add ThisIsaFile
d678197 Add .gitignore
97f1e31 Readme.md
```

- 이 때, 12741e5 커밋을 취소하고, d678197 커밋으로 되 돌리고자 하는 경우에 아래와 같이 할 수 있습니다

```
$ git reset --hard d678197
```

- 결과를 확인해 보면 아래와 같습니다.

```
$ git log --oneline
d678197 (HEAD -> master) Add .gitignore
97f1e31 Readme.md
```

- 여기서, --hard 옵션은 현재 HEAD에서 추가된 변경사항들을 모두 되돌려 줍니다
- --hard 옵션을 사용할 때 Git 저장소에서 관리 하지 않는 파일들(Untracked files)을 추가한 경우 이 파일들은 reset 이후에도 그대로 유지됩니다

- 이 파일들까지 삭제하고자 하는 경우에는 git clean 명령어를 따로 실행해야 한다

```
# 스테이징 환경에 포함되지 않은(Untracked files) 파일 목록 확인
```

```
$ git clean -n
```

```
# 스테이징 환경에 포함되지 않은(Untracked files) 파일 삭제
```

```
$ git clean -f
```

◆ git revert : 커밋 되돌리기

- git reset은 HEAD 위치를 바꿔버려서, 로컬 저장소의 상태를 커밋 이전 상태로 강제 변경한다
- 하지만 커밋을 협업중인 원격 저장소에 push 해버린 경우, 로컬 저장소에서 커밋을 취소해버리면 원격 저장소와 상태가 틀어져버린다. 특히 저장소에서 force push를 금지하는 경우, 로컬 저장소의 변경사항을 push할 수 없게 되어버린다
- 이런 경우에는 git revert로 특정 커밋의 내용을 되돌리는 커밋을 하는 방법을 사용해야한다

```
$ git log --oneline
```

```
12741e5 (HEAD -> master) Add ThisIsaFile
```

```
d678197 Add .gitignore
```

```
97f1e31 Readme.md
```

- 이 상태에서 12741e5 커밋을 되돌리는 경우 다음 명령어를 실행한다

```
$ git revert 12741e5 -m "Add ThisisaFile"
```

- 결과를 확인해 보면 다음과 같이 나타난다

```
$ git log --oneline
2bc76a3 (HEAD -> master) Revert "Add ThisIsaFile"
12741e5 (origin/master) Add ThisIsaFile
d678197 Add .gitignore
97f1e31 Readme.md
```

◆ git commit -amend: 커밋 덮어쓰기

- 커밋의 내용을 덮어쓸 때는 git commit의 --amend 옵션을 사용한다
- 수정할 내용을 스테이징에 반영하고 다음 명령어를 실행한다. (새로운 Commit 히스토리를 생성하지 않고, 맨 마지막 Commit을 덮어 쓴다.)

```
$ git commit --amend -m "Recommit"
```

- amend의 옵션의 경우 스테이징에 추가된 내용을 반영해주는 동시에 커밋 메시지도 변경해 준다
- 따라서 변경할 내용이 없을 때도 커밋 메시지를 변경하고 싶을 때 자주 사용한다

7. Git Branch

◆ Git Branch란?

- Git에서 branch는 기본 리포지토리의 새로운 혹은 별도의 버전이다
- branch를 사용하면 기본 branch에 영향을 주지 않고 프로젝트의 다른 부분에서 작업할 수 있다
- 작업이 완료되면 branch를 메인 프로젝트와 병합할 수 있다
- branch 간에 전환하고 서로 간섭하지 않고 다른 프로젝트에서 작업할 수도 있다
- Git에서 branch는 매우 가볍고 빠르다!

◆ Git Branch 새로 생성

- 기존의 프로젝트의 현재의 상태로 그대로 저장해둔 다음에, 새로운 작업을 계속 이어가고 싶을 경우에는 아래와 같이 새로운 branch를 생성 한다

```
$ git branch hello-world-images
```

- 현재의 branch들을 모두 확인할 수 있다

```
$ git branch  
  hello-world-images  
* master
```

- branch를 이동하고 싶을 때에는 아래의 명령어를 사용한다

```
$ git checkout hello-world-images  
Switched to branch 'hello-world-images'
```

- 현재 작업 branch가 'master'에서 'hello-world-images'로 변경되었다
- 다시 현재의 branch들을 모두 확인해 보면, 아래와 같이 현재 branch가 'hello-world-images'로 바뀐 표시가 나타난다

```
$ git branch  
* hello-world-images  
master
```

- 이제 부터 변경된 branch에서 작업해 보자
- 메모장에서 test.txt 파일을 하나 생성하고, 현재의 디렉토리(webdevelop)에 저장한다

test.txt

```
This is a test File.
```

- 그러면, 'git status'명령어로 확인해 보면, Untracked 파일로 test.txt 파일이 존재하게 된다
- test.txt 파일을 스테이징 환경으로 옮기고, Commit 해 보자

```
$ git add -A  
$ git commit -m "test file commit"
```

- 그리와 나서, dir를 해보면, test.txt 파일을 포함한 현재 File 들이 List 된다
- 그런데, 'master' branch에서는 어떨까?
- 확인하기 위해 master branch로 checkout하고 file들을 List 해 보자

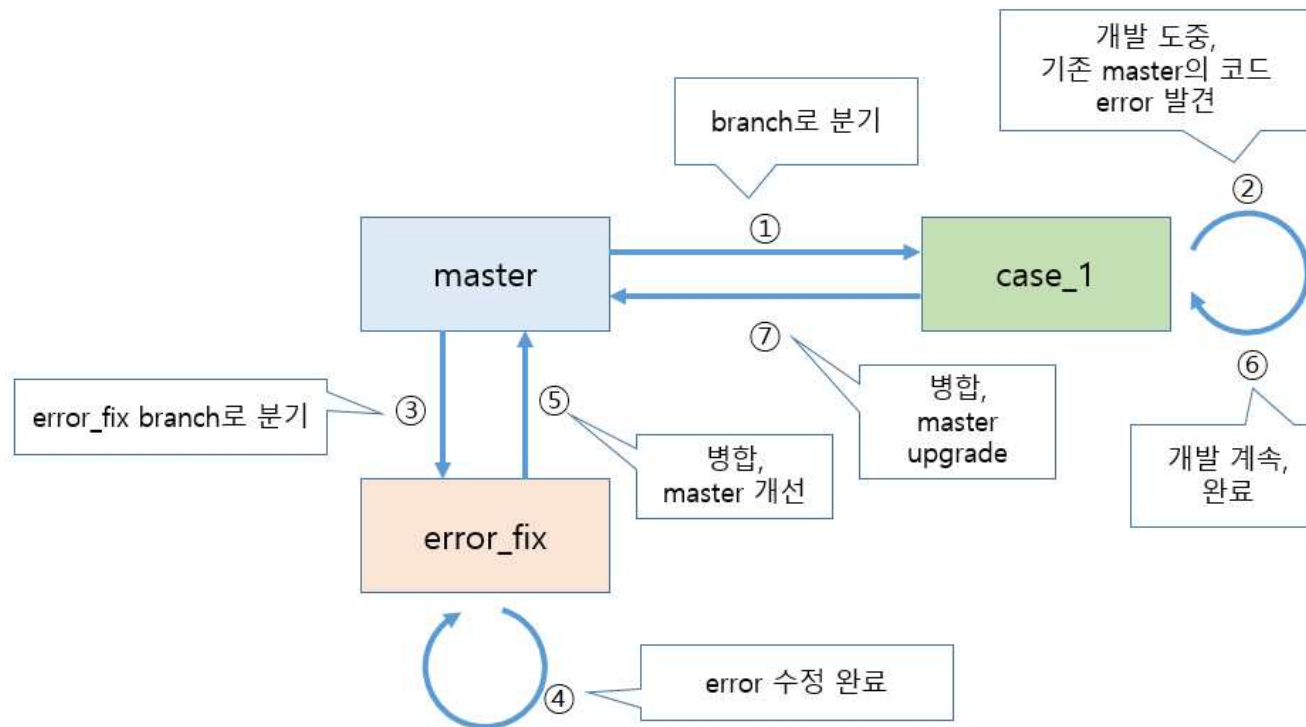
```
$ git checkout master
```

```
$ dir
```

- 결과는 'hello-world-images' branch에서는 보였던 test.txt 파일이 'master' branch에서는 볼 수가 없다
- 이처럼, 한 레포지토리에서 branch로 개발 과정의 Milestone 관리나 여러개의 버전으로 개발을 진행할 수 있다

8. Git Branch 병합(Merge)

- Git Branch 병합은 master branch와 그로 부터 분기된 branch와의 두 branch를 병합하는 것을 의미합니다.
- 아래 그림과 같은 경우를 가정해 보면,



- Master 병합을 위해서는 master branch로 checkout을 먼저하고, 상대 branch를 병합한다.


```
$ git checkout master  
$ git merge error_fix
```

- 그리고, 더 이상 필요없는 branch는 삭제 해 준다.

```
$ git branch -d error-fix
```

◆ Merge 충돌(Conflict) 해결

- 위의 시나리오에서 case_1 branch에서 commit을 통한 작업을 완료한 이후 통합한다.

```
$ git checkout master  
$ git merge case_1
```

- master branch와 통합하는 과정에서 충돌이 발생할 수 있다.
- 그럴 경우, editor를 열어서 충돌된 파일의 내용을 수정해 주어야 한다. (충돌 내용을 포함한 파일이 open 된다.)
- 충돌 내용을 수정한 파일을 스테이징 환경에 포함시키고, 다시 Commit을 수행한다.

```
$ git add -A  
$ git commit -m "메시지..."
```

- 위와 같은 과정을 통하여 새로운 버전의 master branch를 확보할 수 있다

- 그리고 기존의 branch도 필요 없으면 삭제해 준다

```
$ git branch -d case_1
```

참조 :

branch 병합(Merge)는 상당히 복잡하고, 조심해야 하는 기능이다. 병합 기능을 원할히 잘 이용하기 위해서는 별도의 더 많은 학습이 반드시 필요하다

이상으로, Git을 활용하기 위한 기본적인 사항들에 대해 학습하였다. 더 많은 기능의 학습을 위해서는 <https://git-scm.com/> 를 참조하면 된다.

- 다음 장에서는 Git와 GitHub를 연계하여 사용하는 방법에 대해 학습하도록 한다.

Git And GitHub

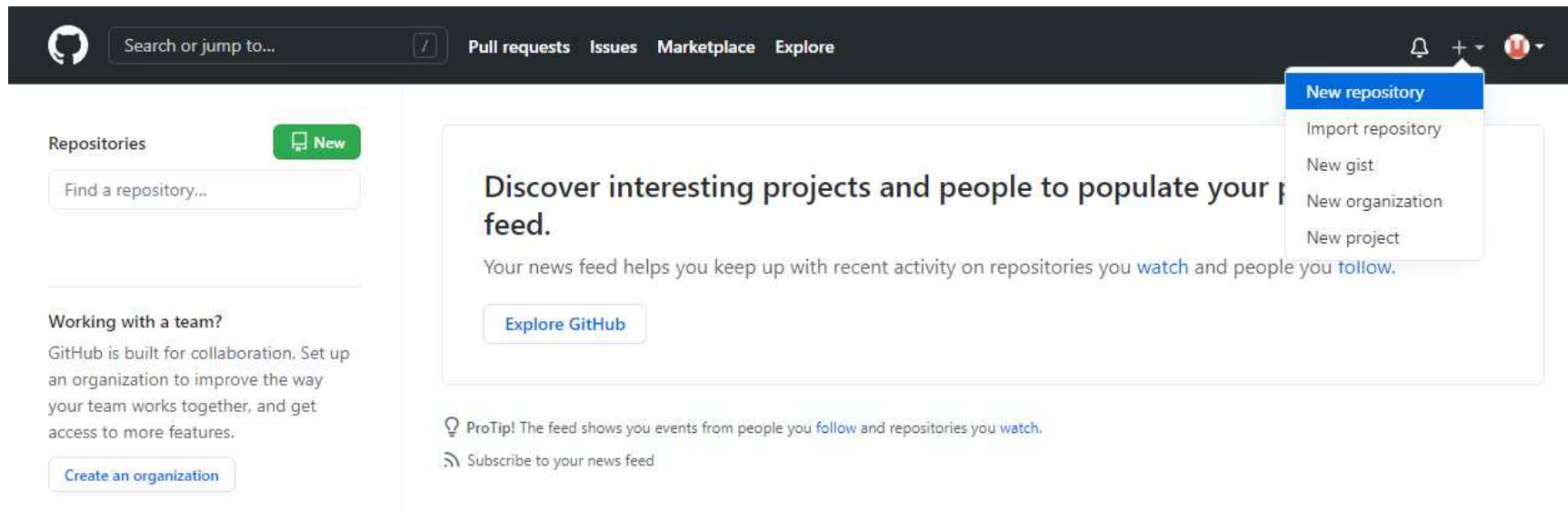
1. GitHub 시작하기

- 먼저 GitHub에 계정을 생성한다. <https://github.com/>
- 계정 생성을 했다면, GitHub에 로그인 한다.





* 참고 : Github id와 Password를 로컬의 Git id와 Password와 동일하게 하는 것이 바람직함.

◆ Repository 생성하고, 로컬 파일들을 Push하기

- 오른쪽 + 아이콘을 눌러 'New Repository'를 선택한다



- 레포지토리 관련 상세 정보를 입력한다.


 / [Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)   

Create a new repository


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

Owner *

Repository name *


 swbae01

 /


hello-world 

Great repository names & **hello-world** is available. le. Need inspiration? How about **didactic-dollop**?

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:
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.](#)

.gitignore template: **None** ▼

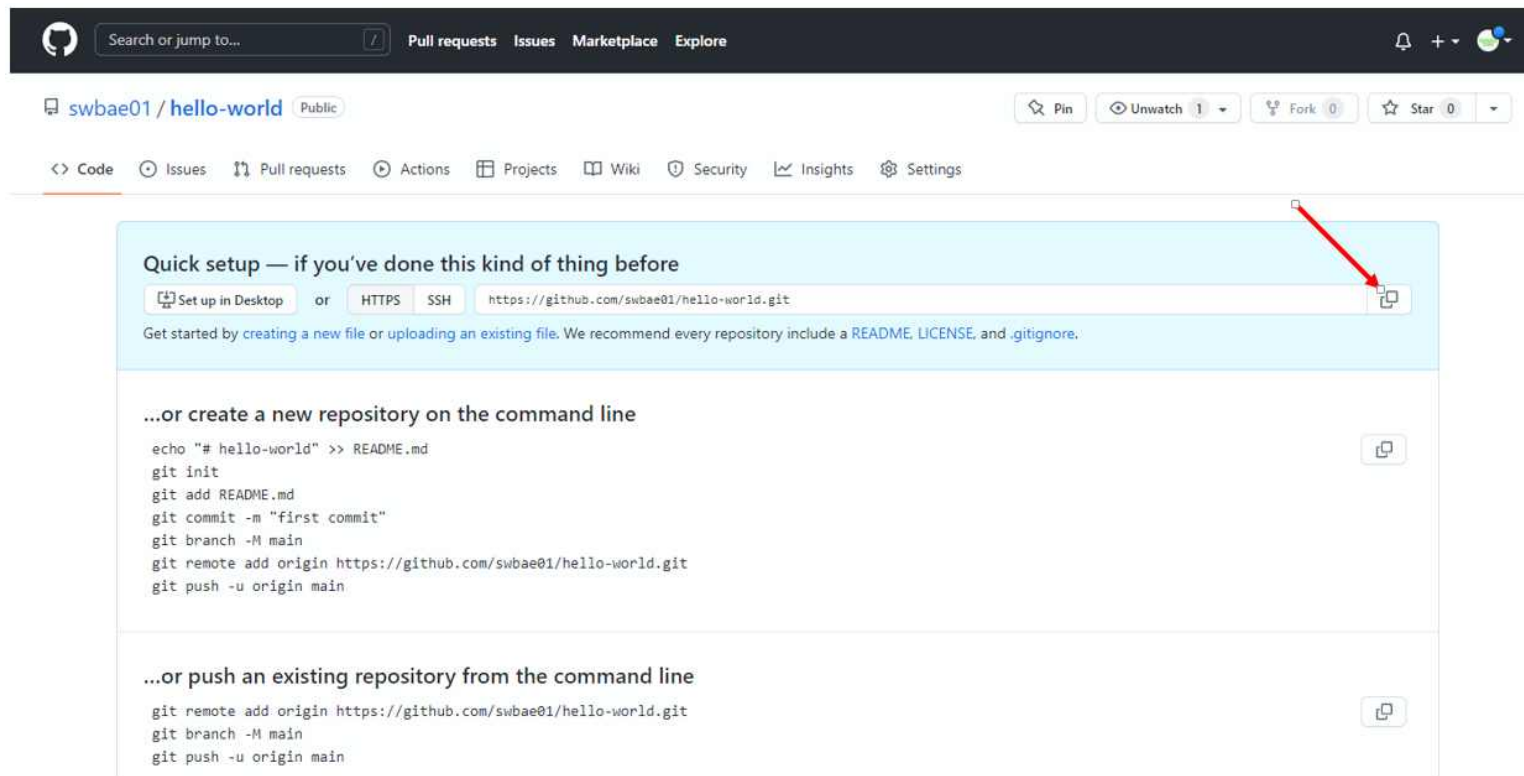
Choose a license
A license tells others what they can and can't do with your code. [Learn more.](#)

License: **None** ▼

 You are creating a public repository in your personal account.

Create repository

- 이제 local 컴퓨터에 있는 Git 레포지토리의 파일들을 GitHub에 Push 해 보자



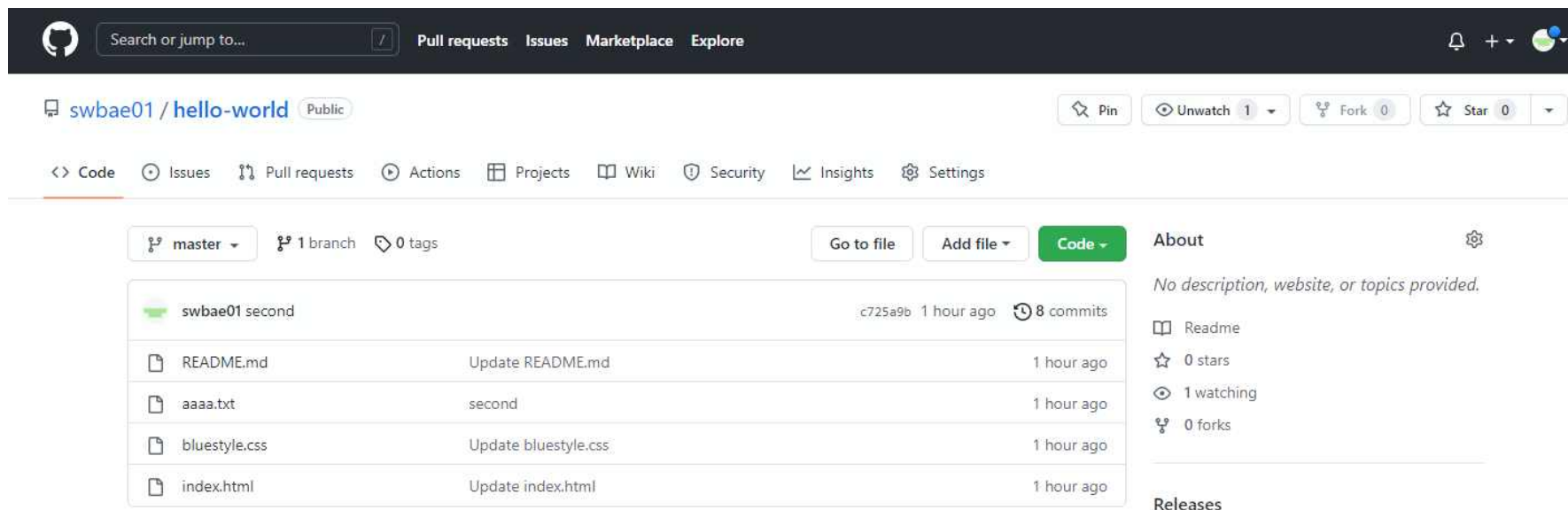
- 위 그림에 있는 버튼을 눌러 GitHub URL을 복사한다
- 그리고, 아래의 명령으로 GitHub의 'hello-world'라는 Remote 레포지토리를 로컬 컴퓨터의 Git에 'origin'이라는 이름으로 추가한다

```
$ git remote add origin https://github.com/swbae01/hello-world
```

- 이제 로컬 컴퓨터 Git의 master branch를 origin url로 Push 할 수 있다.

```
$ git push -u origin master
```

- 그러면 그 결과로 GitHub의 'hello-world' 레포지토리에 아래 그림과 같이 File들이 Push 된 것을 확인할 수 있다



◆ 변경된 내용을 GitHub로 다시 Push하기

- 프로젝트를 진행하면서, 새로운 파일이 생성되었거나, 기존 파일의 내용이 변경되었을 때에는 로컬 Git에서 스테이징과 커밋을 한 후, 다시 GitHub로 최신의 내용을 Push 할 수 있다
- 아래 명령을 순서대로 실행하면 된다

```
$ git add -A
```

```
$ git commit -m "메시지..."  
$ git push origin
```

◆ GitHub에서 변경된 내용을 로컬로 다시 가져오기

- 여러사람과 협업을 하여, 다른 사람이 GitHub 레포지토리의 내용을 갱신하였거나, 혹은 GitHub에서 직접 파일을 수정한 경우, 그 수정된 내용을 다시 일괄적으로 가져오기 위해서는 pull 명령을 사용한다

```
$ git pull origin
```

- 그러면, GitHub 레포지토리의 변경된 내용이, 로컬 컴퓨터의 Git 레포지토리에 반영됩니다.

◆ Remote로 연결되지 않은 GitHub의 레포지토리를 복사만해 올 경우

- 경우에 따라 로컬 Git 레포지토리와 연결되어 있지는 않지만, 특정 GitHub의 레포지토리를 복사해와야 하는 경우가 있다.
- 아래의 명령으로 쉽게 레포지토리를 복사해 올 수 있다.

```
$ git clone https://github.com/swbae01/hello-world
```