

id: ros-how-to-use-github title: Github 이해 및 사용방법 소개 sidebar_label: Github 이해 및 사용방법 소개

- 작성 기준일: 2019. 07. 25.
- 작성자: 김지형 (100kimch@naver.com)

변경 사항

- 본 문서를 수정한 이력이 있다면 다음 표에 작성해주시기 바랍니다.

순번	일자	수정자	수정자 이메일	수정 내용	비고
1	2019. 07. 25	김지형	100kimch@naver.com	Initial Documentation	

개요

본 문서는 Git 및 Github 처음이용자를 위한 Git 및 Github 설명과 사용 설명서입니다.

Git과 Github

Git과 Github를 우선 구분할 필요가 있다.



Git은 유명한 소프트웨어 개발자 리누스 토발즈가 만든 소프트웨어로, 프로젝트를 진행할 때 여러 명의 팀원이 어떤 부분도 겹쳐 쓰지 않게 프로젝트의 변경을 관리하는 버전관리 소프트웨어이다. 왜 Git을 사용해야 할까? 당신의 동료가 동시에 같은 웹사이트에서 페이지를 변경하고 업데이트하고 있다고 하자. 당신이 무언가를 변경하고 저장한 다음 웹사이트에 그것을 업로드한다. 이때 동료가 동시에 같은 페이지에서 같은 내용을 작업할 때 누군가의 작업은 겹쳐쓰여질 것이고 지워지는 문제가 발생할 수 있다. Git과 같은 버전관리 앱은 그런 일을 방지한다. 당신과 동료는 같은 페이지에 각자의 수정사항을 업로드할 수 있고, Git은 두 개의 복사본을 저장한다. 나중에 당신들은 그대로 어떤 작업도 잃어버리지 않고 변경사항들을 병합할 수 있다. Git은 이전에 만들어진 모든 변경사항을 사진을 찍듯 "스냅샷"을 저장하기 때문에 어떠한 버전으로도 되돌릴 수도 있다.

```

egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/a (master)
$ git 1
* 6997660 (HEAD -> master, origin/master) work 3a
* abf35d9 Merge branch 'master' of github.com:egoingsb/git4-collaboration-c
11
| \
| * 0ef37a1 work 2a
| * 1a5381a work 2b
| /
* 33f1754 work 1
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/a (master)
$ |

egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ nano work.txt
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ git commit -am "work 4b"
[master 672ae5e] work 4b
1 file changed, 1 insertion(+)
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ nano work.txt
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ git commit -am "work 5b"
[master e1eede3] work 5b
1 file changed, 1 insertion(+)
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ git format-patch ^C
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ git 1
* e1eede3 (HEAD -> master) work 5b
* 672ae5e work 4b
* 6997660 (origin/master, origin/HEAD) work 3a
* abf35d9 Merge branch 'master' of github.com:egoingsb/git4-collaboration-c
11
| \
| * 0ef37a1 work 2a
| * 1a5381a work 2b
| /
* 33f1754 work 1
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ git format-patch 6997660
0001-work-4b.patch
0002-work-5b.patch
egoing@DESKTOP-RRNVULG MINGW64 ~/Documents/git4-collaboration/cli/b (master)
$ |

```

Git을 사용할 때 어려운 점은 90년대 해커와 같이 코드를 타이핑하는 까만 창에 흰 글씨로 명령어를 사용하여 접근해야 하는 것이다. 이것은 GUI 인터페이스에 익숙한 요즘 컴퓨터 사용자에게 까다로운 일일 수 있다.



Github는 이 까다로운 Git을 편리하게 해준다. Github.com에 계정을 생성하면 웹에서 프로젝트를 버전관리를 할 수 있다. 원하는 파일을 다운로드 받거나, 누가 언제 어디를 수정했는 지에 대해 비주얼적으로 쉽게 표현되어있으며, 까만 창(터미널)에서 할 수 있는 모든 일을 할 수 있다. 또한 다른 Github 사용자의 프로젝트를 둘러볼 수 있고, 그것들을 변경하거나 배우기 위해 자신만의 복사본을 다운로드할 수도 있다. 다른 사용자도 당신의 공개 프로젝트에 대해 같은 걸 할 수 있으며 에러를 발견해서 해결책을 제안할 수도 있다. 어느 경우든 Git이 모든 변경사항에 대한 "스냅샷"을 저장하기 때문에 어떠한 데이터도 잃어버리지 않는다. Git을 배우지 않고 Github를 사용할 수 있지만, 사용하는 것과 이해하는 것은 큰 차이가 있다. Git을 이해하기 전에도 Github를 이용할 수 있으니 처음엔 다른 프로젝트들을 복제해서 이용해보는 것을 먼저 해보길 바란다.

참고 사이트: [CLI\(Command-Line Interface\) 와 GUI\(Graphic User Interface\) ... 그리고 NUI\(Natural User Interface\)](#)

Git 설치

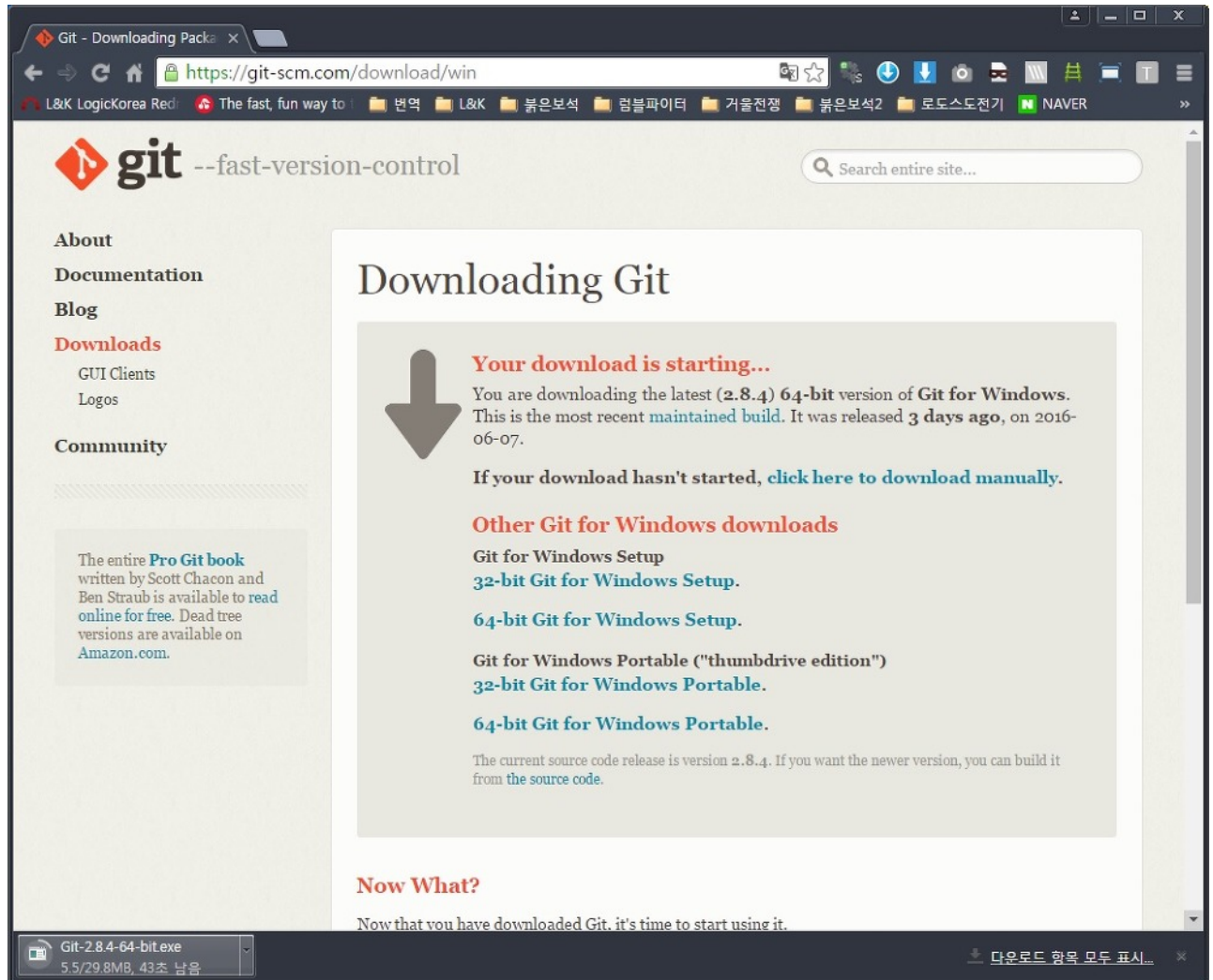
다음 프로그램들을 설치하려 한다.

- `git`
- `python3`, `pip3`

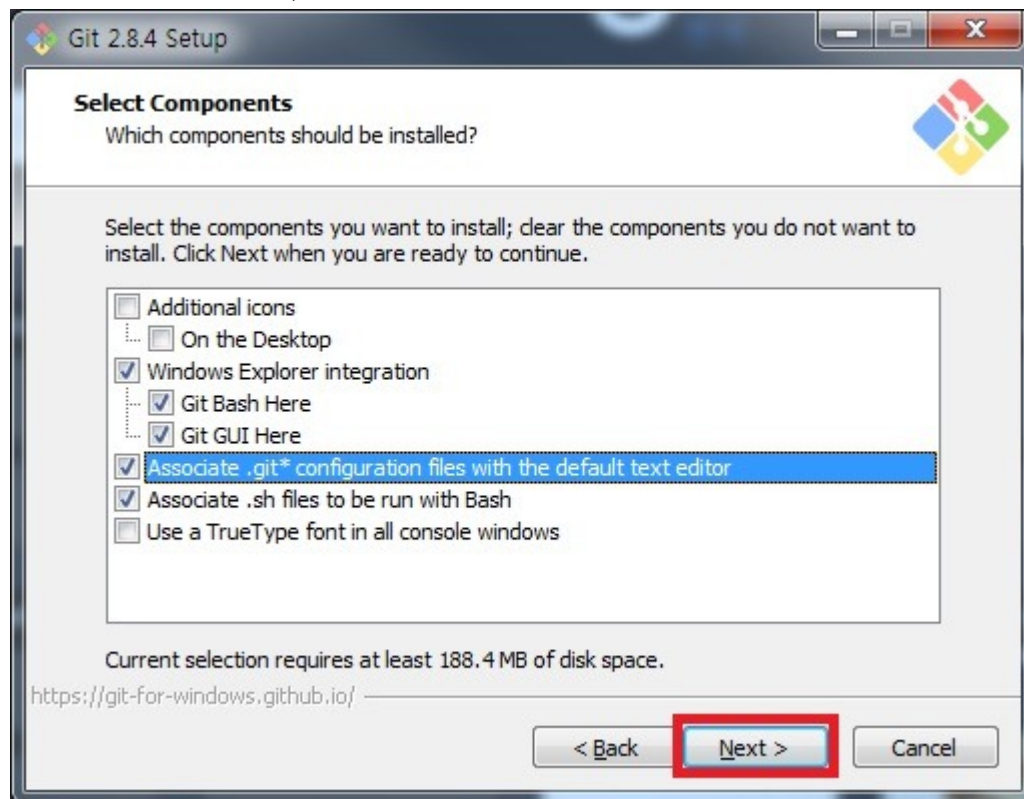
다음 내용은 시간이 지나면서 잘 작동하지 않을 수 있습니다. 설치에 실패할 경우 별도의 검색을 통해서 설치하 시기 바랍니다.

Windows에서 사용하기

1. [다음 링크](#)에서 본인 PC 사양에 맞게 '32-bit Git for Windows Setup' 또는 '64-bit Git for Windows Setup'을 선택한다

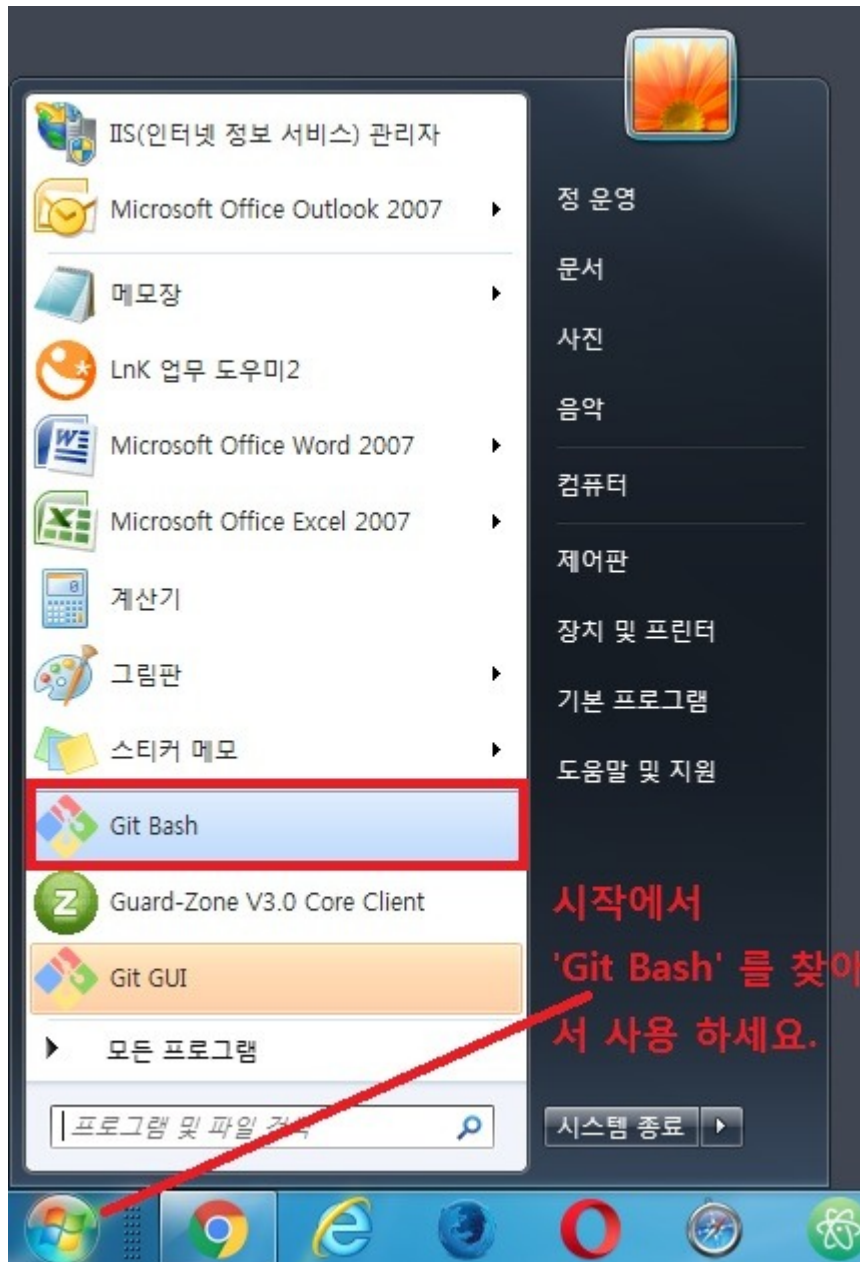


2. 설치 중 'Git Bash Here', 'Associate .sh files to be run with Bash'는 꼭 체크한다.



3. 나머지는 기본 설정값대로 설치 진행한다.

4. 다음 'Git Bash' 프로그램을 통해 Git을 실행시킬 수 있다.



5. Git Bash에서 `git --version`을 쳐서 올바르게 설치되었는지 확인한다.
6. Git Bash에서 `pip3 --version`을 쳐서 pip3가 올바르게 설치되었는지 확인한다.

macOS에서 사용하기

1. macOS 터미널에서는 기본적으로 Git이 설치되어 있다. `CMD + Space`를 눌러 "Terminal"을 쳐서 터미널을 실행한다.
2. `git --version`을 쳐서 올바르게 설치되었는지 확인한다.
3. `pip3 --version`을 쳐서 올바르게 설치되었는지 확인한다.

Linux에서 사용하기

1. Linux 계열에서는 기본적으로 Git이 설치되어 있다. `Ctrl + Alt + T`를 눌러 터미널을 실행한다.
2. `git --version`을 쳐서 올바르게 설치되었는지 확인한다.
3. 설치되어있지 않을 경우 다음을 쳐서 Git을 설치한다.

```
sudo apt install git
```

1. `pip3 --version`을 쳐서 `pip3`가 설치되어있는지 확인한다.
2. 설치되어있지 않을 경우 다음을 쳐서 Git을 설치한다.

```
sudo apt install pip3
```

오픈소스 Git 프로젝트 복제 후 사용하기

1. 다음을 쳐서 Github에서 오픈소스 프로젝트 하나를 다운받는다.

```
https://github.com/nguyenannie/cat_vs_fish
```

1. 다음을 쳐서 Git 폴더로 이동한다.

```
cd cat_vs_fish/
```

1. 다음을 쳐서 해당 프로젝트 프로그램 실행에 필요한 `pygame` 모듈을 설치한다.

```
sudo pip3 install pygame
```

1. 다음을 쳐서 해당 프로젝트의 파이썬 게임을 실행한다.

```
python3 cat_vs_fish.py
```

기본 용어

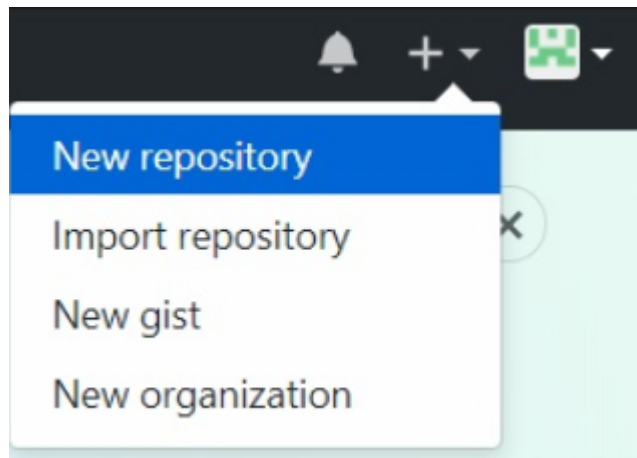
- **저장소**: 프로젝트가 있는 폴더. 깃허브 사용자는 종종 "repo"로 줄여서 사용한다. 당신의 컴퓨터 안의 로컬 폴더가 될 수도 있고, 깃허브나 깃랩 같은 온라인 저장 공간이 될 수도 있다. 저장소 안에 코드 파일, 텍스트 파일, 이미지 파일을 저장하고 이름붙일 수 있다.
- **버전관리**: 기본적으로, 깃이 서비스되도록 고안된 목적. MS 워드 작업할 때, 저장하면 이전 파일 위에 겹쳐 쓰거나 여러 버전으로 나누어 저장한다. 깃을 사용하면 그럴 필요가 없다. 프로젝트 기록에 모든 시점의 **스냅샷**을 유지하므로 결코 잃어버리거나 겹쳐쓰지 않을 수 있다. 커밋: 깃에게 파워를 주는 명령이다. 커밋하면, 그 시점의 당신의 저장소의 **스냅샷**을 찍어, 프로젝트를 이전의 어떠한 상태로든 재평가하거나 복원할 수 있는 체크포인트를 가질 수 있다.
- **브랜치**: 여러 명이 하나의 프로젝트에서 깃 없이 작업하는 것은 매우 혼란스러운 일이다. 일반적으로, 작업자들은 메인 프로젝트의 `master` 브랜치를 따와서, 자신이 변경하고 싶은 자신만의 버전을 만든다. 작업을 끝낸 후, 프로젝트의 메인 디렉토리인 `master`에 다시 합친다.

Git 기초 사용법

Git 프로젝트 생성하기

Git 프로젝트를 생성할 때 `git init`을 이용하지만, 설정을 최대한 간편하게 하기 위해 필자는 다음과 같은 순서대로 깃 프로젝트를 생성하고 이용한다.

1. github.com에 접속한다. 계정이 없는 경우 회원가입한다.



2. 우측 상단에 'New Repository'를 선택한다.
3. Repository 정보를 입력하고 Create repository 버튼을 클릭한다.

Create a new repository
A repository contains all the files for your project, including the revision history.

Owner: wjdxo513 / Repository name: 주소

Great repository names are short and memorable. Need inspiration? How about `potential-memory`.

Description (optional): 설명

☒ **Public**
Anyone can see this repository. You choose who can commit.
 ☐ **Private**
You choose who can see and commit to this repository.
 Public : 공개
Private : 비공개

☐ Initialize this repository with a README
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: Add a license: ⓘ

4. 생성된 프로젝트(Repository)의 주소를 기억한다. 깃허브에서 생성된 프로젝트의 주소 체계는 다음과 같다.



```
https://github.com/${계정 아이디}/${프로젝트 이름}
```

5. Git Bash 또는 터미널을 실행시킨다.

6. 다음을 쳐서 온라인에서 생성된 프로젝트를 다운로드 받는다.

```
git clone https://github.com/${계정 아이디}/${프로젝트 이름}
```

7. 해당 폴더로 이동한다.

```
cd ${프로젝트 이름}
```

8. 테스트 파일을 하나 생성한다.

```
touch hello_world.txt
```

`touch` 명령어를 통해 빈 파일을 작성할 수 있다.

Github에 업로드하기 위한 3단계

- 다음을 쳐서 현재까지 수정된 파일의 현황을 확인할 수 있다.

```
git status
```

- 파일을 업로드하고자 할때 다음 3가지 명령어를 습관적으로 친다.

```
git add .  
git commit -m "어떻게 변경하였는지 설명"  
git push origin master
```

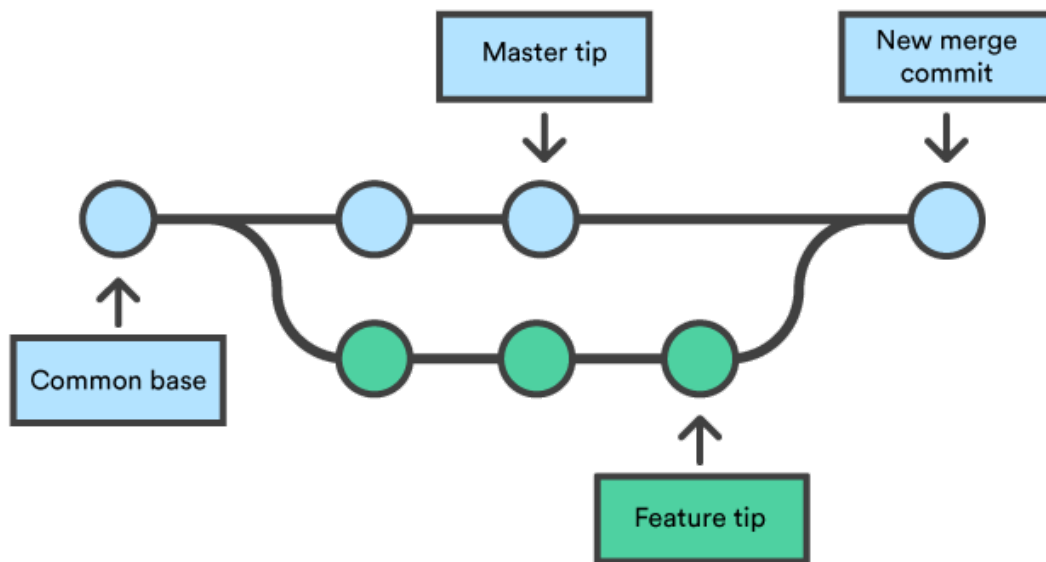
- `git add`는 업로드할 파일들을 선택하는 작업이다. `.`는 현재 경로를 의미하니, 현재 경로 하위에 있는 모든 파일들을 업로드한다.
- `git commit`은 변경 내용을 확정하는 작업입니다. `-m` 옵션을 이용해 이번 확정본에 대한 설명을 붙인다.
- `git push`는 변경된 내용을 발행하는 단계로, 원격 저장소로 올리는 작업을 한다.

Github에서 변경된 내용 갱신

- 다른 사용자가 내용을 변경하여 내 컴퓨터에 새로 변경된 내용을 업데이트할 필요가 있다. 다음 한줄로 원격 저장소에서 변경된 내용을 반영할 수 있다.


```
git pull
```

가지(branch) 치기



- 여러 명의 사용자가 같이 협업하여 프로젝트를 진행하기 위해서는 여러 branch를 만들어 개발을 진행하는 것이 좋다.
- 프로젝트 생성 시 기본으로 만들어지는 branch는 `master`이다. 다른 branch를 생성하여 개발한 후 완료되면 `master` 가지로 돌아와 작업을 한다.
- branch를 생성하고 해당 branch로 갈아탄다.

```
git checkout -b ${branch 이름}
```

- 다시 `master` branch로 이동한다.

```
git checkout -b ${branch 이름}
```

- branch를 삭제한다.

```
git branch -d ${branch 이름}
```

- 다음 명령어를 쳐야만 원격 저장소에 해당 branch의 변경 내역이 반영된다.

```
git push origin ${branch 이름}
```

- 다음을 쳐서 현재 어느 branch에 위치해 있는지 확인할 수 있다.

```
git branch
```

- 다른 branch에 있는 변경 내용을 현재 가지에 병합할 수 있다.

```
git merge ${branch 이름}
```

`git merge` 진행 시 같은 파일 내 내용을 동시에 두 branch가 수정했을 때 충돌(conflict)이 일어날 수도 있다. 이렇게 충돌이 발생하면, git이 알려주는 파일의 충돌 부분을 여러분들이 직접 수정 후 병합해야 한다.

- 변경 내용을 병합하기 전에, 어떻게 바뀌었는지 다음 명령어를 통해 비교해볼 수도 있다.

```
git diff ${원래 branch} ${비교 대상 branch}
```

꼬리표(tag) 달기

- 커밋을 한번 할 때 마다 해당 커밋은 `1b2e1d63ff`와 유사한 식별자가 달리게 된다. 다음 명령을 통해 커밋한 확정본 식별자를 확인할 수 있다.

```
git log
```

- `1b2e1d63ff`와 같은 식별자가 다소 직관적이지 않은데, 아래 명령을 통해 새로운 꼬리표인 `1.0.0`을 달 수 있다.

```
git tag 1.0.0 1b2e1d63ff
```

로컬 변경 내용 되돌리기

- 최대한 일어나지 않게 하는 것이 좋지만 무언가 잘못된 경우 아래 명령어를 통해 로컬 변경 내용을 되돌릴 수 있다.

```
git checkout -- ${파일 이름}
```

- 위 명령은 로컬의 변경 내용을 변경 전 상태(HEAD)로 되돌려준다. 단, 이미 인덱스에 추가 변경된 내용과 새로 생성한 파일은 그대로 남는다.
- 만약 로컬에 있는 모든 변경 내용과 확정본을 포기하려면, 아래 명령으로 원격 저장소의 최신 이력을 가져오고, 로컬 `master` 가지가 저 이력을 가리키도록 할 수 있다.

```
git fetch origin  
git reset --hard origin/master
```