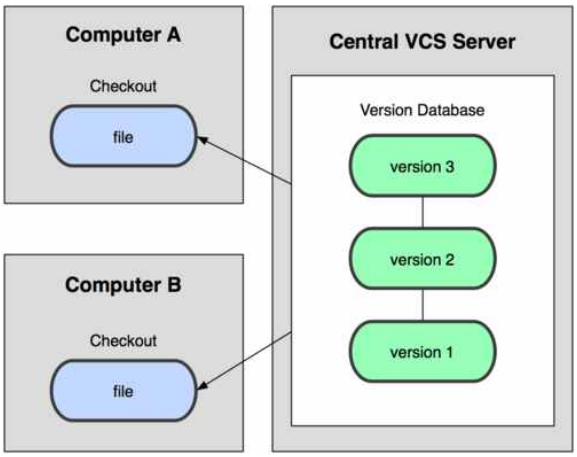
Git과 GitHub의 간단한 사용법

● VCS(Version Control System) : 파일의 변화를 시간에 따라 기록했다가 나중에 특정 시점의 버전을 다시 꺼내올 수 있는 시스템을 의미한다. VCS의 특징 4가지
- 각 파일을 이전 상태로 되돌릴 수 있다.
- 시간에 따라 수정 내용을 비교해 볼 수 있다.
- 누가 문제를 일으켰는지 쉽게 추적할 수 있다.
- 파일을 잘못 고쳤을 때 쉽게 복구할 수 있다.

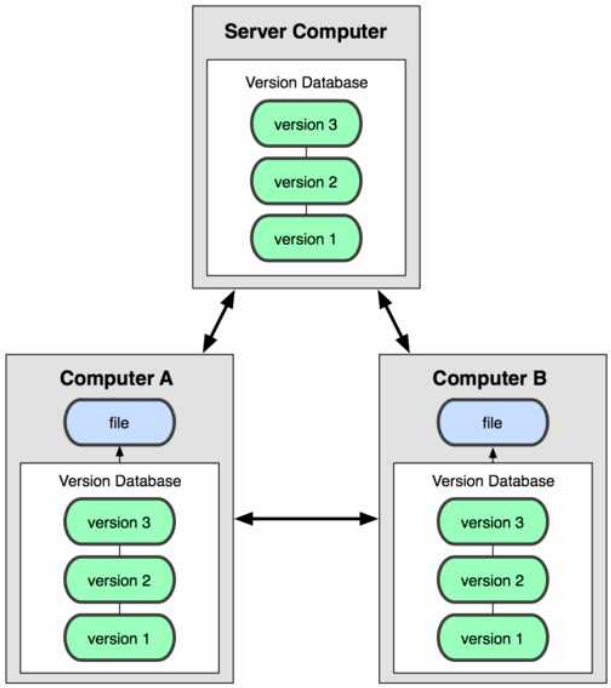
● CVCS(Centralized Version Control System) : 중앙집중식 버전 관리 시스템. 이전에 사용했던 CVS나 Subversion같은 제품이 이 범주에 들어간다. 파일을 관리하는 서버가 별도로 존재하고 , 클라이언트는 중앙 서버에서 파일을 받아서 사용하는 개념.



cf) 여기서 나오는 checkout이라는 용어는 Git의 checkout과는 다른 개념.
이런 CVCS는 중앙서버에 문제가 발생하면, 다른 사람과의 협업 자체가 불가능해진다. 또한 중앙서버의 하드디스크에 문제가 발생하면, 프로젝트의 모든 History를 잃어버리게 된다.
(백업서버 운영시 이런 문제를 일시적으로 해결할 수는 있지만 본질적인 문제는 남아있게 된다.)

● 분산형 VCS (DVCS, Distributed Version Control System)
- DVCS는 분산 버전 관리 시스템. Git이 대표적 제품이다. 이 방식은 CVCS처럼 클라이언트가 파일의 마지막 Snapshot을 Checkout하는 방식이 아니다. 클라이언트는 서버 저장소를 통째로 로컬에 복제해서 사용한다.

(다음 장 그림 참조)



● Git의 탄생

- Linux kernel은 대규모의 오픈소스 프로젝트이다. 이 프로젝트의 버전 관리를 위해 초기에는 BitKeeper라는 상용 DVCS를 사용했었는데 2005년도에 이 BitKeeper의 무료사용이 제고되면서 리눅스 토발즈의 주도로 Linux 개발 커뮤니티 자체 VCS를 개발했는데 이게 Git.

● Git의 특징

- 단순한 구조에서 오는 빠른 속도
- 완벽한 분산처리
- branch를 사용한 비선형적 개발 가능
- 속도나 크기면에서 대형 Project에 적합

● Git의 기본 : Committed, Modified, Staged 의 3가지 상태로 파일을 관리하게 됨

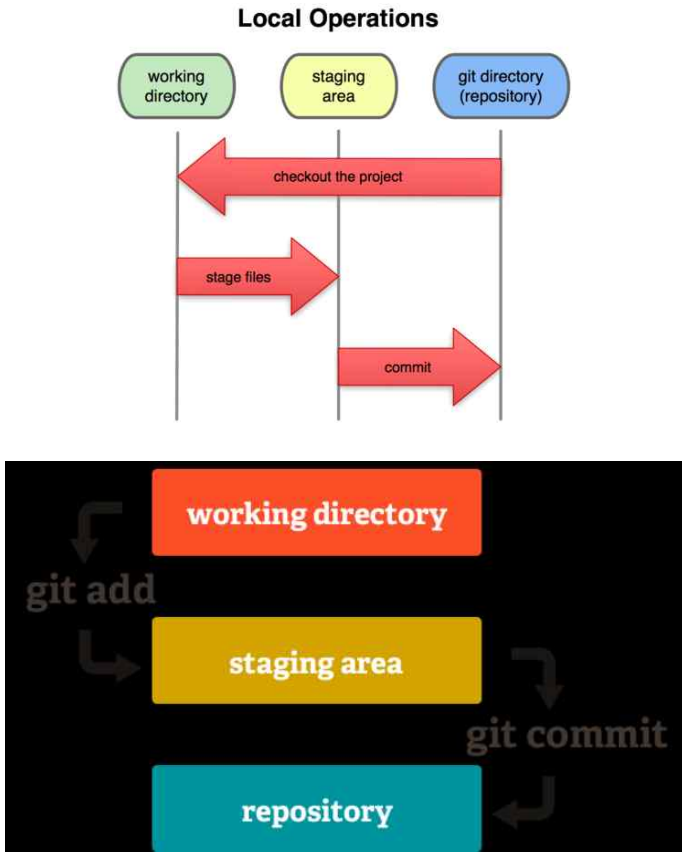
- Committed : 파일을 수정한 후 해당 파일에 대해 commit 명령을 실행해 파일을 로컬 데이터베이스 (로컬 Repository) 에 안전하게 저장한 상태를 의미한다.
- Modified : 파일을 수정한 후 아직 로컬 데이터베이스에 commit 하지 않은 상태를 의미한다.
- Staged : 파일을 수정한 후 수정할 파일을 곧 commit할 것이라고 표시한 상태.

● Git은 파일상태 관리와 더불어 3가지 영역을 사용한다.

- Git directory : Git이 project의 메타데이터와 객체 데이터베이스를 저장하는 곳을 의미한다. 다른말로 Local Repository라고 하며 만약 특정 폴더를 Git directory(Local Repository)로 설정하려면 git init 명령을 이용하면 된다. Repository로 설정되면 .git 이라는 숨김 폴더가 생성되고 이 안에 Git 관리 정보들이 생성된다.

- Working directory : project의 특정 branch를 checkout 한 내용이 들어있는 폴더.
- Staging Area : Git directory에 존재하며 단순한 파일입니다. 곧 commit할 파일에 대한 정보를 가지고 있게 된다.

● Git으로 하는 작업의 기본순서
- Working directory에서 파일 수정
- Staging Area에 수정한 파일을 Stage해서 commit할 Snapshot 생성(git add)
- Staging Area에 있는 수정된 파일을 commit해서 Git directory에 영구적인 Snapshot으로 저장.(git commit)



● Git 설치 & 기본 설정
- http://git-scm.com 을 클릭해서 Git Official HomePage로 이동해서 Git을 다운로드한 후 기본설정으로 install 한다.
- Git을 설치한 후 git config를 이용해 기본적인 환경설정을 한다. 설정파일은 크게 3종류
- 1) $GIT_HOME/mingw64/etc/gitconfig : 시스템의 모든 사용자와 모든 저장소에 적용되는 설정으로 git config -- system 으로 설정한다.
- 2) $USER_HOME/.gitconfig : 특정 사용자에게만 적용되는 설정. git config --global로 설정한다.
- 3) .git/config : Git directory 안에 위치하며 특정 저장소에만 적용되는 설정

- 윈도우 시스템에서 git을 설치하고나면 Git Bash 메뉴가 생성되는데 이를 실행해 console

을 실행시킨 후 사용자 이름과 Email주소를 설정하면 된다.

```
kangs@LAPTOP-GKP6OJ8H MINGW64 ~
$ git config --global user.name "sangwoo0727"

kangs@LAPTOP-GKP6OJ8H MINGW64 ~
$ git config --global user.email "kangsw9395@gmail.com"

kangs@LAPTOP-GKP6OJ8H MINGW64 ~
$
```

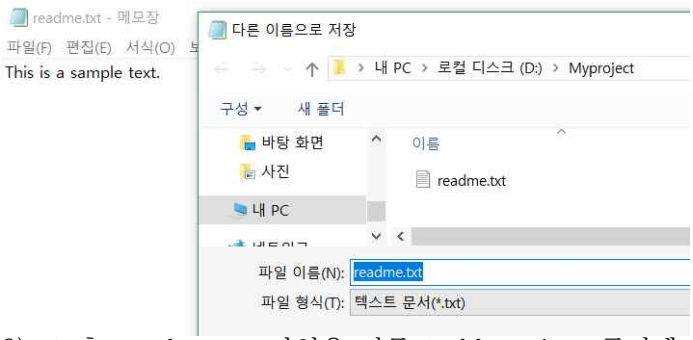
cf) mingw64 는 윈도우시스템으로 porting한 GNU 소프트웨어 도구모음이다.

● Git Local Repository 생성

- Git을 이용해 Git Repository를 만들어서 사용하는 과정을 설명하는 순서. 가지고 있는 project가 없기 때문에 간단하게 폴더를 하나 생성하고 그 폴더를 project 폴더로 간주하고 진행.
- Git의 기본 명령과 개념을 이해하는 목적이기 때문에 command 창에서 명령어를 이용해서 작업을 진행한다. 추후에 실제 project에 적용할때는 IDE의 기능을 이용하거나 SourceTree와 같은 GUI 툴을 이용하는게 좋다.

```
kangs@LAPTOP-GKP6OJ8H MINGW64 ~
$ cd /d

kangs@LAPTOP-GKP6OJ8H MINGW64 /d
$ cd Myproject

kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject
$
```

1) 먼저 command 창에서 다음과 같이 입력해서 D드라이브 아래에 Myproject라는 폴더를 생성한다. 그 후 실제로 그 위치로 가서 Myproject가 생성되었는지 확인한다.



2) 그 후 readme.txt 파일을 만들고, Myproject 폴더에 저장한다.

```
kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject
$ dir
readme.txt

kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject
$ git init
Initialized empty Git repository in D:/Myproject/.git/
```

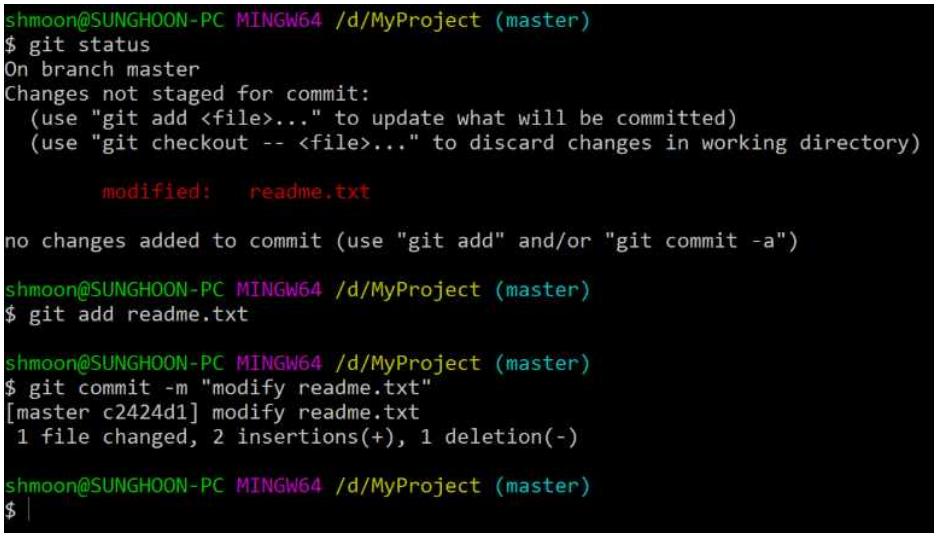
3) dir 명령어를 통해 Myproject 폴더 안에 readme.txt 파일이 있는 것을 확인할 수 있다.
4) git init 명령어를 실행하여 Git Repository를 생성한다. 위와 같은 메시지가 출력된다. (Myproject 아래에 .git이라는 폴더가 생성되는 것을 확인 할 수 있고, 그 안에 Git Repository가 생성된다. 또한 Repository에 필요한 것들도 생성된다.

```
kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        readme.txt

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

kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject (master)
$
```

5) 이 상태에서 git status 명령어를 실행하면, Git이 아직 추적하고 있지 않은 readme.txt가 존재한다는 문구가 뜬다.
- Git Repository를 생성했지만 아직 어떠한 파일도 관리를 하고 있지 않다. 이제 Git이 파일을 관리하게 하려면 Repository에 git add를 이용해 파일을 추가하고 git commit을 이용해 commit까지 진행해야한다.

```
kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject (master)
$ git add readme.txt

kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject (master)
$ git status
On branch master

No commits yet

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

        new file:   readme.txt


kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject (master)
$ git commit -m "initial commit"
[master (root-commit) 917a03b] initial commit
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt

kangs@LAPTOP-GKP6OJ8H MINGW64 /d/Myproject (master)
$
```

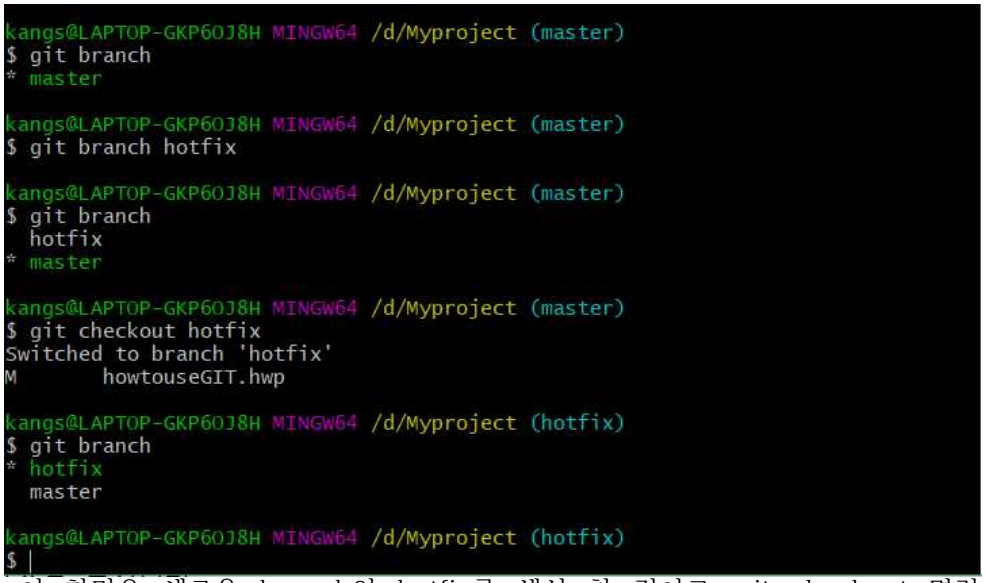
6) 다음 그림처럼 명령을 이용하여 Repository에 파일 추가, 확인 , commit까지 진행한다. (-m 옵션은 commit message를 작성하기 위해서 사용된다.)



7) 위 그림은 메모장을 통해 readme.txt의 내용을 바꾼후 다시 git status 명령어를 실행시켜서 Git이 해당 파일이 변경(Modified)되었다는 것을 인식해서 보여주는 것을 알 수 있다. 변경된 내용을 적용해 Repository에 저장하려면 다시 git add를 이용해 해당 파일을 staging 한 후 git commit을 실행해야 한다.

● Branch 생성

- branch 는 기본 project에 영향이 가지 않는 상태에서 새로운 기능을 추가하거나 기존 기능을 변경해야 하는 경우에 유용하게 사용할 수 있다. 필요에 의해 만들어지는 이런 각각의 branch들은 서로간의 영향을 받지 않습니다. 그렇기 때문에 여러 작업을 동시에 진행할 수 있다.
- Repository를 처음 생성하게 되면 Git는 master라는 이름의 branch를 만들어준다. 위에서 우리가 readme.txt 파일을 repository에 추가하고 내용을 변경해서 commit까지 진행했었는데 모두 이 master라는 branch에서 처리했던 것.
- master가 아닌 다른 branch를 생성할 수 있다. 또한 “이제부터의 작업은 xxx branch에서 진행할꺼야!” 라는 식으로 명령을 줄 수 있는데 이걸 checkout 이라고 한다. 즉 checkout은 특정 branch의 내용을 가져와서 Working directory를 설정하는 작업이라고 보면 된다. 특정 branch에서 일어나지 않는 모든 작업은 당연히 master branch에서 일어나게된다.
- 현재 어떤 branch가 존재하는지 알아보려면 git branch 명령을 이용하면 된다. 만약 새로운 branch를 생성하고 싶으면 git branch branch_name 형태로 branch 이름을 명시하면 된다.
- 현재 작업중인 branch는 *모양으로 표시된다.



<이 화면은 새로운 branch인 hotfix를 생성 한 것이고, git checkout 명령을 이용하여, hotfix branch를 working directory에 가져온 것이다. 지금부터 하는 작업은 모두 hotfix branch에서 발생하는 것이고, master와 무관하게 동작한다>

- 실제로 파일을 변경하거나 추가해서 hotfix branch에서 작업한 후 다시 master branch를 checkout 해보면 아까 했던 작업이 master branch에 영향을 미치지 않는다는 것을 확인할 수 있다.
- 참고로 모든 branch를 확인하기 위해서는 다음의 명령을 실행하면 된다.

git branch -a

● Branch Merge

- Merge 작업은 현재 작업중인 branch에 다른 branch를 가져와서 병합하는 작업을 의미한다. git merge branch_name을 이용하여 현재 branch에 명시된 이름의 branch를 가져와 파일을 병합하게 된다.
- 만약 두 개의 branch에서 같은 파일의 같은 곳을 수정했을 경우 해당 파일을 병합할 때 당연히 문제가 발생하게 된다. 그냥 합쳐질 수가 없기 때문이다. conflict가 발생했다고 표현한다. 이런 경우 충돌이 일어난 내용을 살펴보고 수동으로 해결해야 한다.
- 수동으로 파일을 수정한 후 다시 commit 작업을 진행해야 한다. merge 작업은 Remote Repository를 설명하는 부분에서 다시 다룬다

● .gitignore 파일

- project 폴더 안에서 굳이 추적할 필요가 없는 파일들도 존재한다. 입출력 데이터 파일이나 로그파일, 혹은 .class 와 같은 파일들은 굳이 Git을 이용해서 관리할 필요가 없다. 즉, 임시로 사용되거나 결과물로 생성되는 파일들이 이 범주에 들어간다.
- 이런 경우 .gitignore 파일을 이용해서 추적 관리할 필요가 없는 파일을 배제 시킬 수 있다. 그냥 만들어도 되지만 https://www.gitignore.io/ 이 사이트에 들어가서 조금 더 쉽게 .gitignore 파일의 내용을 만들어 복사해서 사용할 수 있다.

● Remote Repository

- Git은 혼자 사용할 수 도 있지만 기본적으로 다른 사람과 협업을 하기 위한 도구이다. 협업 도구로서 Git의 가장 큰 유용함은 Remote Repository(원격 저장소)에 있다.
- 이 Remote Repository는 우리가 따로 구축해서 사용할 수도 있다. 또한 이런 Remote Repository를 서비스하는 회사도 굉장히 많이 있다. Git 기반의 Remote Repository중 가장 대표적인 것이 바로 GitHup 이다.
- Remote Repository를 쉽게 생각하자면 로컬에서 작업한 Git Local Repository가 외부에 있는 거라고 생각하면 된다. GitHub는 이런 Remote Repository를 전세계적으로 서비스하고 있고 굉장히 많은 사람들이 GitHub를 자신들이 수행하고 있는 project의 Remote Repository로 이용하고 있다.
- GitHub에는 이런 Remote Repository는 크게 public과 private로 구분된다. 말그대로 public은 아무나 파일을 열람할 수 있도록 공개되어있는 것이고, private는 권한은 가진 사람들만 사용할 수 있는 것
- GitHub은 Fork와 Pull Request 라는 기능을 제공하고 있다. 정확히 말하자면 이 Fork와 Pull Request는 Git이 제공하는 것이 아니라 GitHub이 제공하는 서비스.

  (1) Fork : 다른 사람의 Repository를 통째로 내 GitHub 계정으로 복사해 오는 기능.

  (2) Pull Request : 다른 사람의 Repository를 Fork 한 후 그 내용을 수정한 다음 원본 Repository에 수정된 내용을 보내 병합을 요청할 수 있는데 이 작업을 Pull Request 라고 한다. 아무나 Repository를 수정할 수 있는 권한을 주게 되면 Repository는 금방 엉망이 될테니 READ 기능(Fork)만 제공하고 병합시에는 요청을 받아서 처리하도록 한다.

● GitHub에서 Remote Repository 생성

- GitHub 계정을 생성했으면, Repository를 만들면 된다.

● Remote Repository 관리 명령어

- GitHub에서 생성한 Remote Repository를 관리하기 위해서 Git은 몇몇개의 명령어를 제공한다.

(1) git clone : Remote Repository 의 모든 내용을 Local Repository로 복사한다,

(2) git remote : Local Repository를 특정 Remote Repository와 연결시킬 때 사용한다,

(3) git push : Local Repository에 추가된 파일이나 변경 사항을 연결된 Remote Repository에 저장하기 위해서 사용한다.

(4) git fetch : Remote Repository와 Local Repository의 변경사항이 다를 때 이를 비교 대조해서 충돌을 해결하고 최신 데이터를 반영하기 위해서 사용한다.

(5) git pull : 연결된 Remote Repository의 최신 내용을 Local Repository로 가져오면서 merge한다. git push와 반대개념이고, merge할 때 문제 발생하면 추적 어렵다.

● git clone

- git clone은 Remote Repository에 있는 project를 내 컴퓨터로 가져올 때 사용한다. 즉, GitHub에서 Local 환경으로 복사하는 작업이다.

- 저장할 폴더를 생성한 후, 명령어를 이용해 복사한 것을 확인할 수 있다.
- 정상적으로 clone이 진행되면, 저장소 이름으로 폴더가 하나 생성된다. 당연히 이 폴더는 Remote Repository와 연결되어 있는, Local Repository가 된다.

- 이해를 돕기위한 순차적 과정

(1) 협업을 책임지는 사람(PM)이 GitHub에 빈 Remote Repository를 생성한다.
(2) PM은 자신의 Local Repository에 project에 필요한 기본 구조와 여러 가지 환경설정 그리고 개발한 필요한 기타 사항들을 만들어서 저장한다.
(3) PM은 project의 뼈대가 담겨있는 이 Local Repository를 GitHub에 생성해 놓은 빈 Remote Repository와 연결한다. 이 때 git remote 명령을 이용하게 된다.
(4) PM은 자신이 Local Repository에 가지고 있는 내용을 Remote Repository에 push한다.
(5) 협업하는 사람 모두가 이 Remote Repository를 clone해서 로컬로 복사해 간 다음 자신이 해야하는 작업을 진행한다.

● git remote

- GitHub에 빈 Remote Repository를 생성한 후 git remote를 이용하여 Local Repository와 연결 할 수 있다.
- git remote add origin “Remote Repository URL” 명령어를 입력한다.
- git remote -v를 이용하여 연결이 성공했는지를 알 수 있다.

● git push

- git remote를 이용하여 Local Repository와 Remote Repository가 연결되었으면 이제 자신이 작업한 내용을 Remote Repository에 upload 할 수 있다.
- git push 명령을 이용하면 파일을 upload 할 수 있는데 기본적으로 Remote Repository의 master branch에 upload되게 된다. 따라서 만약 다른 branch의 내용을 upload 하려면 다음과 같은 명령을 실행해야 한다.
- git push origin “local branch 명”
- 위의 명령에서 orgin은 원격 저장소의 별칭이다. git remote를 이용하여 Remote Repository를 연결할 때 이 origin이라는 별칭을 이용해서 원격 연결을 했었다.
- 만약 origin 저장소에 Local의 모든 branch를 push 하려면 다음과 같이 명령을 수행하면

된다.

- git push origin --all
- git push가 진행될 때 Remote Repository에 같은 이름의 branch가 존재한다면 내용이 변경될 것이고 만약 해당 branch가 존재하지 않는다면 새로운 branch를 Remote Repository에 생성하게 된다. 같은 이름의 branch가 존재하지만 내역이 다르다면 당연히 push는 일어나지 않고 작업이 거부된다.

ex) Local Repository의 master branch의 내용을 Remote Repository에 push하려면
git push origin master
ex) Local Repository의 hotfix branch의 내용을 Remote Repository에 push 하려면
git push origin hotfix

<< 출처 >> https://moon9342.github.io/git-github