



GIT & GITHUB

BASIC

문성훈

강사소개



- 문성훈
- (주) ATGLab. 개발 / 컨설팅 / IT직무교육 전문기업.
- Ph.D. in Computer Science
- 대구 가톨릭대학교 IT 공학부 산학교수(현)
과학기술정보통신부 - 국가초고성능 컴퓨팅위원회 실
무위원회 민간위원(현)
- 삼성전자 / LGCNS / KT 직무교육
한국IBM / SK C&C 직무교육
LGCNS 신입사원교육 및 과정설계 / 직무평가 컨설팅
CJON 직무교육
한화시스템/ICT 신입사원교육
- Email : moon9342@gmail.com
- Blog : <https://moon9342.github.io>

과정목적 (1)



최종 Report.doc



최종 Report(0307).doc



최종 Report(최종).doc



최종 Report(완성본).doc

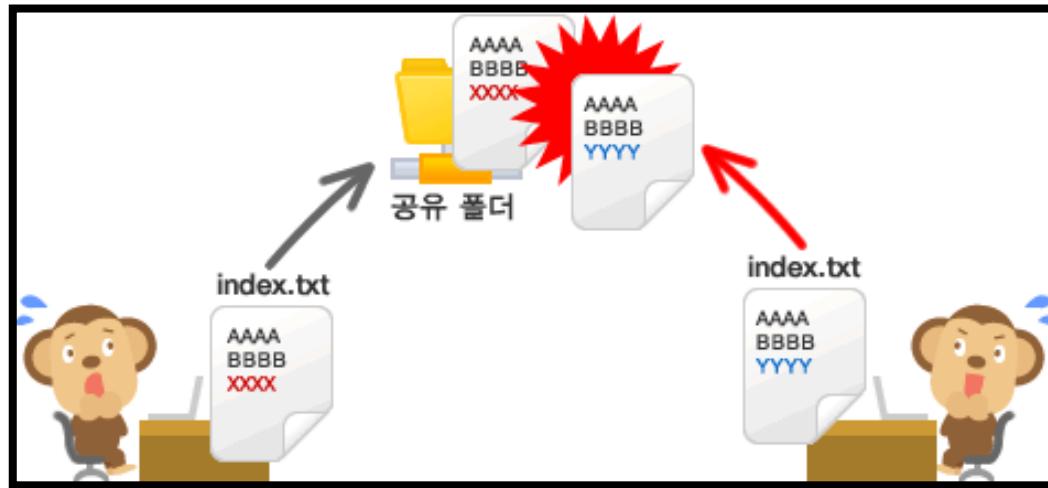


최종 Report(진짜 최종).doc



과정목적 (2)

- 여러 명이 공동으로 같은 파일을 작업할 때 파일 내용을 덮어쓰는 문제가 생길 수 있다.
즉, 다른 사람이 작업한 사항을 유실할 수 있다.
- 이와 같은 문제가 항상 빈번하게 발생하게 되고 이를 해결하기 위해 나온 시스템이 바로
VCS(Version Control System)



VCS (VERSION CONTROL SYSTEM) 개요

❖ VCS란?

- 파일의 변화를 시간에 따라 기록했다가 나중에 특정 시점의 버전을 다시 꺼내올 수 있는 시스템

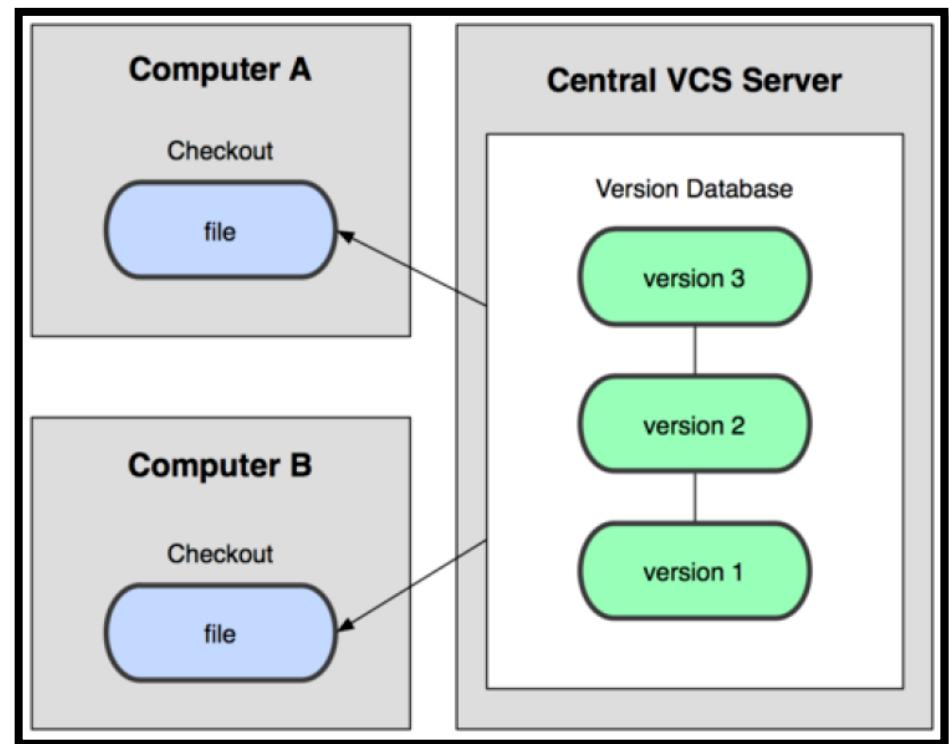
- VCS의 특징

- 개별 파일 혹은 프로젝트 전체를 이전 상태로 되돌리거나 시간에 따른 변경사항을 쉽게 검토할 수 있다.
- 문제가 되는 부분을 추적해서 누가 마지막으로 수정했는지를 알 수 있다.
- 파일을 잃어버리거나 잘못되었을 경우 쉽게 복구할 수 있다.

VCS (VERSION CONTROL SYSTEM) 개요

❖ Centralized Version Control System (CVCS)

- 중앙집중식 버전 관리 시스템
- CVS, Subversion(SVN)이 대표적 System
- 파일을 관리하는 서버가 별도로 존재하고 클라이언트는 중앙 서버에서 파일을 받아 사용 (Checkout)



VCS (VERSION CONTROL SYSTEM) 개요

❖ Centralized Version Control System (CVCS)

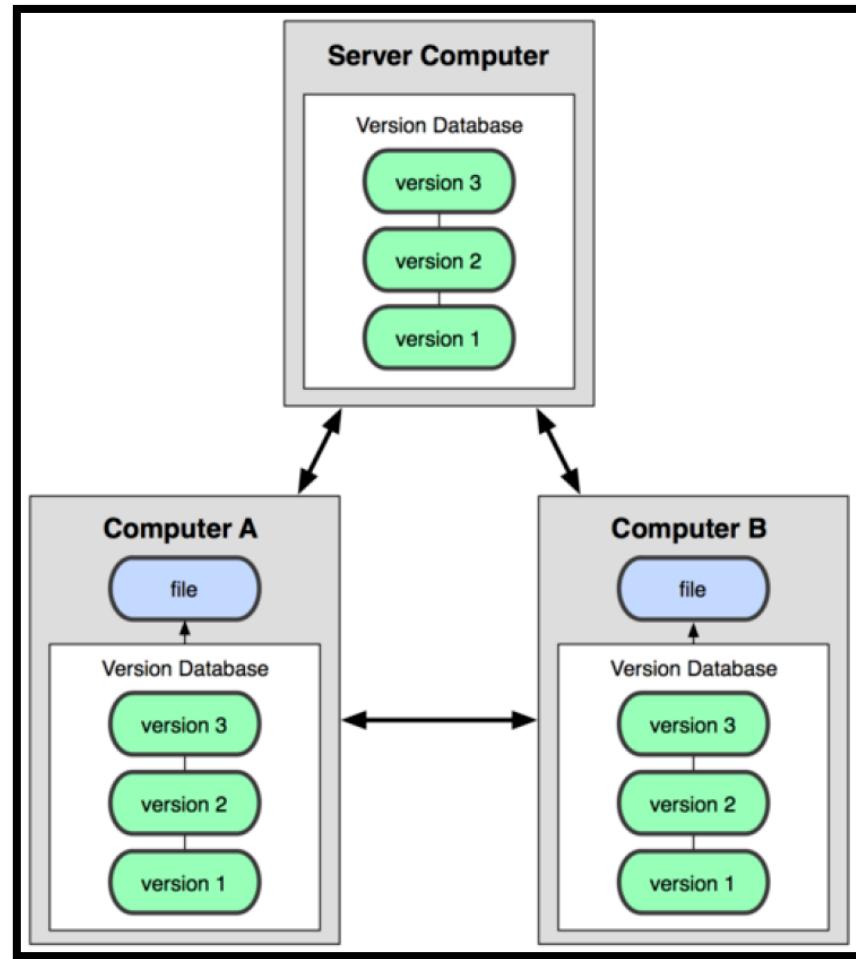
- CVCS 환경의 장점과 단점

- 누구나 다른 사람이 무엇을 하고 있는지 알 수 있고,
관리자는 누가 무엇을 할 수 있는지 세세하게 관리 가능
- 중앙서버에 문제가 발생하면 다른 사람과의 협업과 백업 등이 불가능
- 중앙서버의 하드디스크에 문제가 발생하면 프로젝트의 모든 History를 잃어버릴 수 있다.

VCS (VERSION CONTROL SYSTEM) 개요

❖ Distributed Version Control System (DVCS)

- 분산 버전 관리 시스템
- Git이 대표적 제품
- 클라이언트가 파일의 마지막 Snapshot을 Checkout하는 방식이 아닌 저장소(Repository)를 통째로 복제해서 사용



VCS (VERSION CONTROL SYSTEM) 개요

- ❖ Git의 탄생
- Linux kernel은 대규모의 open source project
- 2002년도에 Project를 관리하기 위해서 BitKeeper 상용 DVCS 사용. 2005년도에 BitKeeper의 무료사용이 제고되고 Linux 개발 커뮤니티가 자체 도구를 제작 (리누스 토발즈)
- Git의 특징
 - 빠른 속도
 - 단순한 구조
 - 완벽한 분산
 - 동시 다발적인 branch를 사용하여 비선형적 개발 가능
 - 속도나 데이터 크기 면에서 대형 Project에 적합

GIT MECHANISM (LOCAL REPOSITORY)

❖ Git의 파일 관리

- Git은 파일을 3가지 상태로 관리 (**Committed, Modified, Staged**)

- Modified : 수정한 파일을 아직 로컬 데이터베이스에 commit하지 않은 것을 의미
- Staged : 파일을 수정한 후 수정된 파일을 곧 commit할 것이라고 표시한 상태를 의미
- Committed : 데이터가 로컬 데이터베이스(Local Repository)에 안전하게 저장되었다는 것을 의미.

GIT MECHANISM (LOCAL REPOSITORY)

❖ Git이 사용하는 3가지 영역

● Git Directory (Local Repository)

- Git이 프로젝트의 메타데이터와 객체 데이터베이스를 저장하는 공간.
- 만약 특정 폴더를 Git directory(Local Repository)로 설정하려면 “git init” 명령을 이용.
- Local Repository로 설정되면 .git이라는 숨김 폴더가 생성되고 이 안에 Git 관리 정보들이 생성.

● Working Directory

- 프로젝트의 특정 버전(branch)을 checkout한 Directory. Local Repository의 압축된 데이터베이스에서 파일을 가져와서 Working Directory를 생성.

● Staging Area

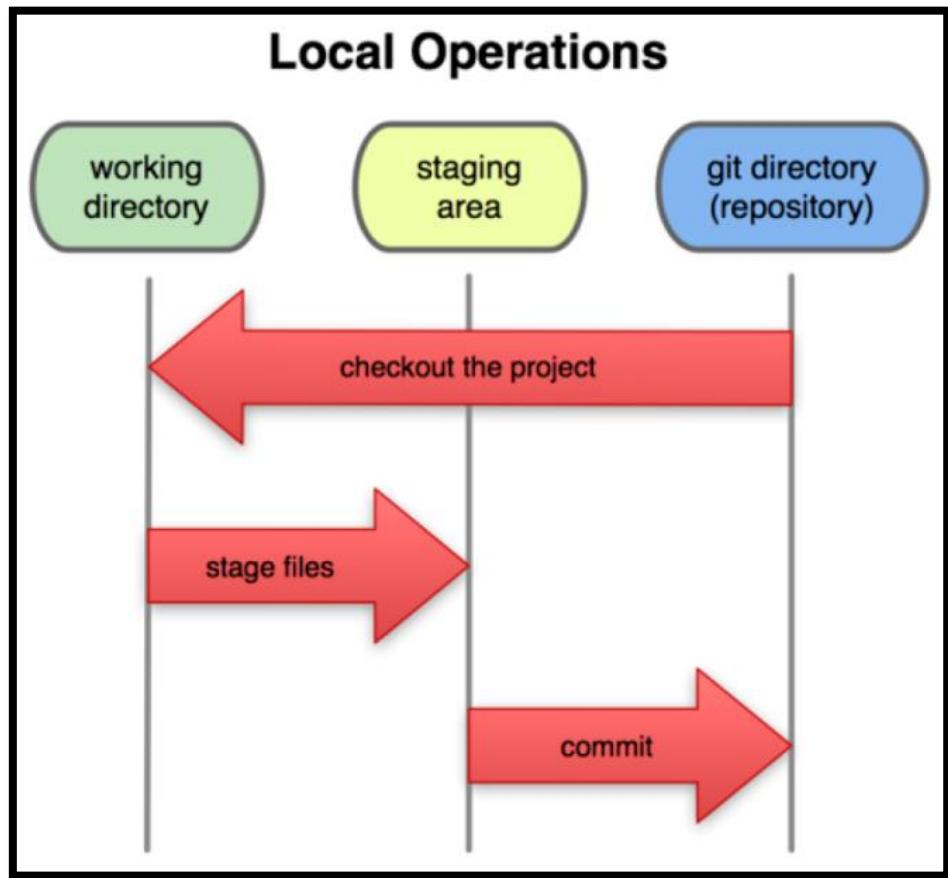
- Git Directory(Local Repository)안에 파일로 존재. 곧 commit할 파일에 대한 정보를 저장.

GIT MECHANISM (LOCAL REPOSITORY)

❖ Working Directory, Staging Area, Git Directory

● Git 작업의 기본 순서

- Working Directory에서 파일 수정.
- Staging Area에 파일을 Stage해서 commit할 Snapshot생성
- Staging Area에 있는 파일들을 commit해서 Git Directory에 영구적인 Snapshot으로 저장



GIT MECHANISM (LOCAL REPOSITORY)

❖ Git에서 관리하는 File의 Lifecycle

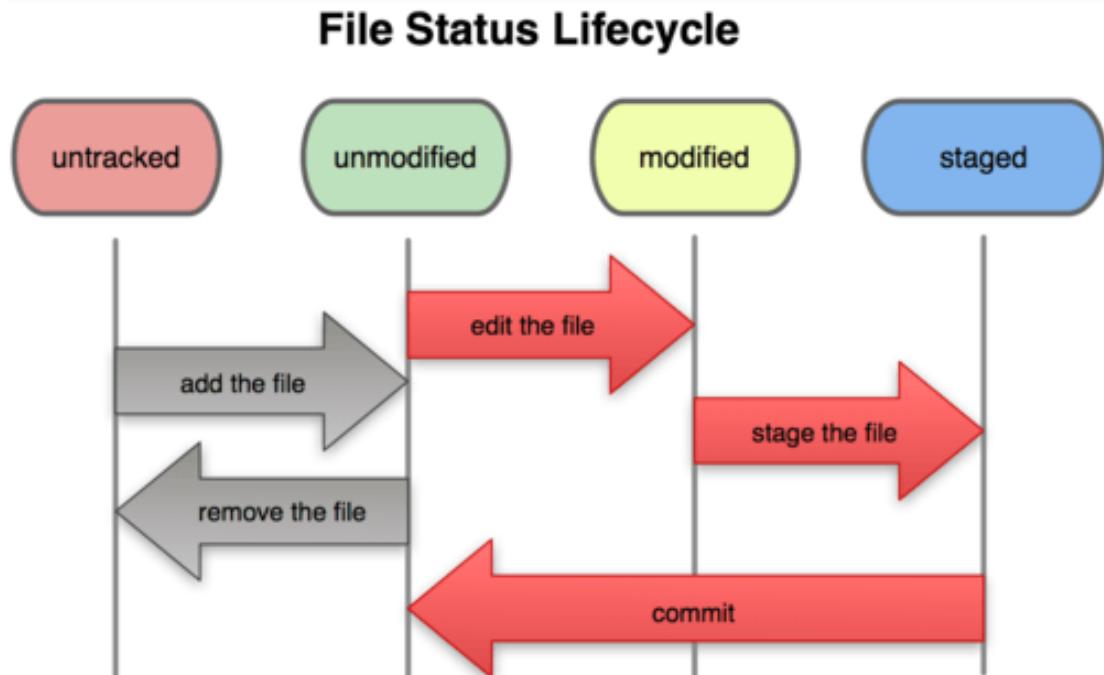
- Working Directory의 모든 파일은 크게 2가지로 구분.

- Tracked file :

Repository의 Snapshot에 포함되어 있던 파일

- Tracked file은 Unmodified, modified, staged 3가지 상태 중 하나.

- 그 이외의 모든 파일은 Untracked file



GIT 설치 및 기본설정

❖ Git download & install

- <http://git-scm.com>

➤ 기본 설정으로
install

The screenshot shows the official website for Git (<http://git-scm.com>). At the top, there's a large red Git logo followed by the word "git" in a bold, lowercase sans-serif font. Below the logo is the tagline "--distributed-even-if-your-workflow-isn't". A search bar with a magnifying glass icon and placeholder text "Search entire site..." is positioned to the right. The main content area features several sections: "About" (describing Git as a free and open source distributed version control system), "Documentation" (linking to command reference pages and the Pro Git book), "Downloads" (providing links for GUI clients and binary releases), and "Community" (inviting users to get involved with bug reporting and mailing lists). To the right, there's a graphic illustrating the distributed nature of Git with multiple interconnected nodes represented by stacks of books. Below this is a monitor icon displaying a teal box with the text "Latest source Release 2.21.0" and a "Download 2.21.0 for Windows" button.

GIT 설치 및 기본설정

❖ Git 환경설정

- git config를 이용하여 설정내용을 확인하고 변경
- 설정파일은 크게 3가지 종류가 존재 (Windows system)
 - {\$GIT_HOME}/etc/mingw64/etc/gitconfig : 시스템의 모든 사용자와 모든 저장소에 적용되는 설정 (git config --system 으로 사용)
 - {\$USER_HOME}/.gitconfig : 특정 사용자에게만 적용되는 설정 (git config --global 으로 사용)
 - .git/config : Git directory안에 있고 특정 저장소에만 적용
 - 각각의 설정은 위에 나열된 순서의 역순으로 우선 적용된다.

GIT 설치 및 기본설정

❖ Git 환경설정

● 실습 진행

- Git Bash 실행
- 사용자 이름과 Email 설정
(Git에서 commit을 할 때마다 사용하는 사용자 이름과 Email 설정)

```
MINGW64:/c/Users/shmoon

shmoon@DESKTOP-JLVQBG9 MINGW64 ~
$ git config --global user.name "SungHoon Moon"

shmoon@DESKTOP-JLVQBG9 MINGW64 ~
$ git config --global user.email "moon9342@gmail.com"

shmoon@DESKTOP-JLVQBG9 MINGW64 ~
$
```

GIT 사용 방법

❖ Git Repository (Git Directory) 생성

- Git을 이용해 Git Repository를 만들어서 사용해 보자.
- 가지고 있는 Project가 없기 때문에 간단하게 폴더를 하나 생성하고 그 폴더를 Project 폴더로 간주하고 진행.
- Git의 기본 명령과 개념을 이해하는 것이 목적이기 때문에 command 창에서 명령어를 이용해서 작업 진행.
- 추후에 실제 Project에 적용할 경우 IDE의 기능을 이용하거나 SourceTree와 같은 GUI툴을 이용.

GIT 사용 방법

❖ Git Repository (Git Directory) 생성

- 윈도우 탐색기를 이용해 먼저 프로젝트 폴더를 생성 (C:/MyProject)
- Git으로 관리할 파일을 하나 생성 (readme.txt)
- readme.txt의 내용 : This is a sample Text

GIT 사용 방법

❖ Git Repository (Git Directory) 생성

- project 폴더로 이동한 후 console을 통해 “git init” 을 실행해 Git Repository를 생성.



The screenshot shows a terminal window titled "MINGW64:/c/MyProject". The command \$ cd "C:/MyProject" is run, followed by \$ git init, which initializes an empty Git repository in C:/MyProject/.git/. The final prompt shows the user is now in the master branch of the repository.

```
MINGW64:/c/MyProject
shmoon@DESKTOP-JLVQBG9 MINGW64 ~
$ cd "C:/MyProject"

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject
$ git init
Initialized empty Git repository in C:/MyProject/.git/
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$
```

- .git 폴더가 생성되고 그 안에 Git Repository가 생성. 또한 Repository에 필요한 각종 Skeleton들도 같이 생성된다.

GIT 사용 방법

❖ Git Repository (Git Directory) 사용

- git status를 실행.

- Git이 아직 추적하고 있지 않은 readme.txt가 존재한다고 출력.

```
MINGW64:/c/MyProject
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/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)
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ |
```

GIT 사용 방법

❖ Git Repository (Git Directory) 사용

- Git이 파일을 관리하게 하려면 Repository에 git add를 이용해 파일을 추가하고 git commit을 이용해 commit까지 해야 한다.
- Repository에 파일 추가, 확인, commit까지 진행.
(-m option은 commit message를 작성하기 위해서 사용.)
- 다음 슬라이드의 그림 참조

GIT 사용 방법

❖ Git Repository (Git Directory) 사용

```
MINGW64:/c/MyProject
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git add readme.txt

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/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

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git commit -m "initial Commit"
[master (root-commit) 34ef419] initial Commit
 1 file changed, 1 insertion(+)
 create mode 100644 readme.txt

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ |
```

GIT 사용 방법

❖ Git Repository (Git Directory) 사용

- readme.txt 파일의 내용을 에디터를 이용해 수정.
- git status를 실행.
- Git은 해당 파일이 변경(Modified)되었다는 것을 인식해서 해당내용을 출력.
- 변경된 내용을 적용해 Repository에 저장하려면 다시 git add를 이용해 해당 파일을 staging 한 후 git commit을 실행.
- 다음 슬라이드의 그림 참조

GIT 사용 방법

❖ Git Repository (Git Directory) 사용

```
MINGW64:/c/MyProject
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   readme.txt

no changes added to commit (use "git add" and/or "git commit -a")

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git add readme.txt

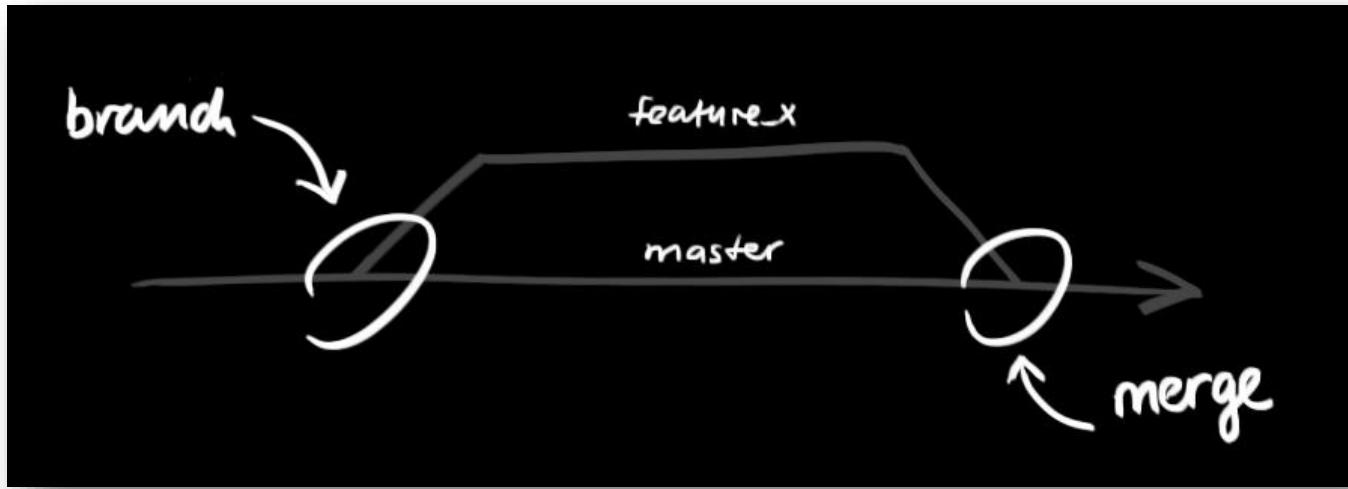
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git commit -m "modify readme.txt"
[master b7d28f6] modify readme.txt
 1 file changed, 2 insertions(+), 1 deletion(-)

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ |
```

GIT 사용 방법

❖ Branch 활용

- branch는 기본 project에 영향이 가지 않는 상태에서 새로운 기능을 추가하거나 기존 기능을 변경해야 하는 경우에 사용.
- 이런 각각의 branch들은 서로 간의 영향을 받지 않는다.
그렇기 때문에 여러 작업을 동시에 진행시킬 수 있다. (비선형적)



GIT 사용 방법

❖ Branch 활용

- Repository를 처음 생성하게 되면 Git은 master라는 이름의 branch를 생성.
- 이전 예제에서 우리가 readme.txt 파일을 Repository에 추가하고 내용을 변경해서 commit까지 진행했는데 모두 이 master라는 branch에서 처리.
- master가 아닌 다른 branch를 생성할 수 있다.
또한 “이후로의 작업은 xxx branch에서 진행한다!!!” 라는 식으로 명령을 줄 수 있는데 이걸 checkout이라고 한다.
- 즉, checkout은 특정 branch의 내용을 가져와서 Working directory를 설정하는 작업. 특정 branch에서 일어나지 않는 모든 작업은 당연히 master branch에서 일어나게 된다.

❖ Branch 활용

- 현재 어떤 branch가 존재하는지 알아보려면 git branch 명령을 이용.
- 만약 새로운 branch를 생성하고 싶으면 git branch branch_name 형태로 branch 이름을 명시하시면 새로운 branch가 생성.
- 현재 작업중인 branch는 * 기호로 표시된다.
- 다음 슬라이드의 그림은 새로운 branch를 생성하고 git checkout 명령을 이용하여 hotfix branch 를 Working directory에 가져오는 것으로 이후부터 하는 작업은 모두 hotfix branch에서 발생하는 것이고 master branch와는 무관하게 동작한다.

GIT 사용 방법

❖ Branch 활용

```
MINGW64:/c/MyProject
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git branch
* master

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git branch hotfix

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git branch
  hotfix
* master

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git checkout hotfix
Switched to branch 'hotfix'

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (hotfix)
$ git branch
* hotfix
  master

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (hotfix)
$
```

GIT 사용 방법

❖ Branch 활용

- 파일을 변경하거나 추가해서 hotfix branch에서 작업한 후 다시 master branch를 checkout 해 보면 아까 했던 작업이 master branch에는 영향을 미치지 않는다는 것을 확인하실 수 있다.
- 이 부분은 개별적으로 실습

❖ Branch Merge

- Merge 작업은 현재 작업중인 branch에 다른 branch를 가져와서 병합하는 작업을 의미.
- git merge branch_name 을 이용하여 현재 branch에 명시된 이름의 branch를 가져와 파일을 병합.
- 만약 두개의 branch에서 같은 파일의 같은 곳을 수정했을 경우 해당 파일을 병합할 때 당연히 문제가 발생하게 된다.
- conflict가 발생했을 경우 내용을 살펴보고 수동으로 해결해야 한다.
(Git은 병합이 실패했을 때 그 해결을 모두 사용자에게 일임.)

GIT 사용 방법

❖ Branch Merge 실습

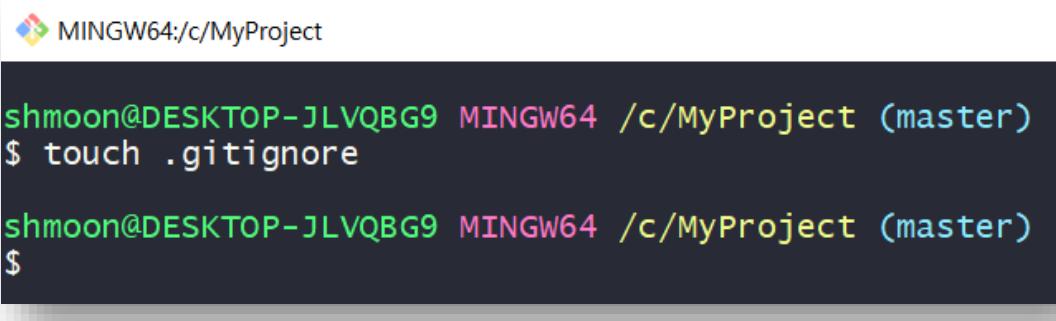
```
MINGW64:/c/MyProject  
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)  
$ git merge hotfix  
Auto-merging hotfix.readme.txt  
CONFLICT (content): Merge conflict in hotfix.readme.txt  
Automatic merge failed; fix conflicts and then commit the result.  
  
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master|MERGING)  
$ git commit hotfix.readme.txt  
fatal: cannot do a partial commit during a merge.
```

```
MINGW64:/c/MyProject  
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)  
$ git commit -i hotfix.readme.txt
```

GIT 사용 방법

❖ .gitignore 파일

- Project 폴더 안에 굳이 추적할 필요가 없는 파일들도 존재.
 - 입출력 데이터파일이나 로그파일, 혹은 .class와 같은 파일들은 굳이 Git을 이용해서 관리할 필요가 없다. 즉, 임시로 사용되거나 환경설정파일, 결과물로 생성되는 파일들이 이 범주에 들어간다.
- 이런 경우 .gitignore 파일을 이용해 추적 관리할 필요가 없는 파일을 배제시킬 수 있다.
- <https://www.gitignore.io/> 이곳에서 자신의 설정에 맞는 .gitignore 파일의 내용을 생성 한 후 파일로 저장.



The screenshot shows a terminal window titled 'MINGW64:/c/MyProject'. It displays the following command being run:

```
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ touch .gitignore
```

After the command is run, the terminal shows the prompt again:

```
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$
```

❖ Remote Repository

- Git은 혼자 사용할 수도 있지만 기본적으로 다른 사람과 협업을 하기 위한 도구.
- 협업 도구로서 Git의 가장 큰 유용함은 Remote Repository(원격 저장소)에 있다.
- 이 Remote Repository는 우리가 따로 구축해서 사용할 수 있다. 또한 이런 Remote Repository를 서비스하는 회사도 굉장히 많이 있는데 Git 기반의 Remote Repository 중 가장 대표적인 것이 바로 GitHub.



GIT 사용 방법

❖ Remote Repository

- GitHub에 있는 이런 Remote Repository는 크게 public repository와 private repository로 구분
 - public repository : 아무나 파일들을 열람할 수 있도록 공개되어 있는 repository
 - private repository : 권한을 가진 사람들만 사용할 수 있는 repository입니다.
- GitHub는 Fork와 Pull Request라는 기능을 제공.
이 Fork와 Pull Request는 Git이 제공하는 것이 아니라 GitHub가 제공하는 서비스.
 - Fork : 다른 사람의 Repository를 통째로 내 GitHub 계정으로 복사해 오는 기능.
즉, GitHub 계정간 Repository를 서로 복사해 갈 수 있는 기능이라고 생각하면 된다.
 - Pull Request : 다른 사람의 Repository를 Fork한 후 그 내용을 수정한 다음 원본 Repository에 수정된 내용을 보내 병합을 요청할 수 있는데 이 작업을 Pull Request라고 한다. 아무나 Repository를 수정할 수 있는 권한을 주게 되면 Repository는 금방 엉망이 될 테니 READ기능(Fork)만 제공하고 병합시에는 요청을 받아서 처리하도록 한다.

GIT 사용 방법

❖ GitHub에 Remote Repository 생성

- GitHub에 계정을 생성하고 새로운 Repository를 생성하는 버튼을 클릭.
- Repository이름을 입력하고 간단한 설명을 입력합니다.
- Repository의 종류를 선택.
- 이전에 만들어 놓은 Local Repository와 연결시키는 목적으로 사용할 것이기 때문에 README 파일을 만들지 않고 Repository를 생성.

❖ GitHub에 Remote Repository 생성

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

Owner Repository name *

 moon9342 / MyProjectRemote ✓

Great repository names are short and memorable. Need inspiration? How about `scaling-octo-invention`?

Description (optional)
MyProject에 대한 GitHub Remote Repository

Public
 Anyone can see this repository. You choose who can commit.

Private
 You choose who can see and commit to this repository.

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.

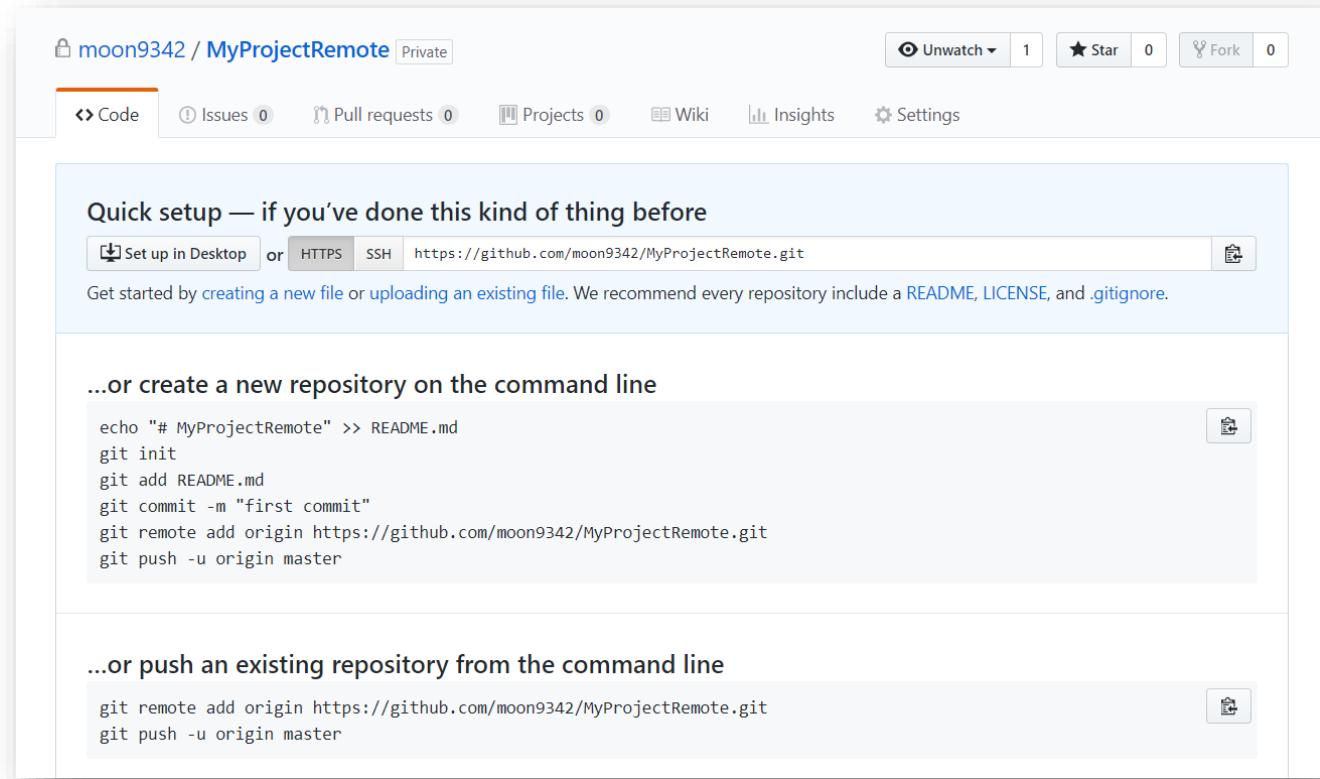
Add .gitignore: None ▾ | Add a license: None ▾ ⓘ

Create repository

GIT 사용 방법

❖ GitHub에 Remote Repository 생성

- Repository를 생성하면 다음과 같은 화면을 볼 수 있다. Repository에 현재 파일이 존재하지 않기 때문에 이렇게 나오는 것이고 만약 특정 파일들이 저장되어 있으면 파일의 목록들이 보여지게 된다.



GIT 사용 방법

❖ GitHub에 Remote Repository 관리 명령어

- GitHub에 생성한 Remote Repository를 관리하기 위해서 Git은 몇몇개의 명령어를 제공.
 - git clone : Remote Repository의 모든 내용을 Local Repository로 복사.
 - git remote : Local Repository를 특정 Remote Repository와 연결시킬 때 사용.
 - git push : Local Repository의 변경 사항을 연결된 Remote Repository에 적용하기 위해서 사용.
 - git fetch : Remote Repository와 Local Repository의 변경 사항이 다를 때 이를 비교 대조해서 충돌을 해결하고 최신 데이터를 반영하기 위해서 사용.
 - git pull : 연결된 Remote Repository의 최신 내용을 Local Repository로 가져오면서 merge.
git push와 반대의 개념으로 생각하면 되지만 merge할 때 문제가 발생했을 때 추적이 어렵다.
- git pull을 이용하는 것 보다는 일단 git fetch로 변경사항을 받고 이를 확인해서 코드를 수정한 후 Local Repository에 commit한 다음 git push로 최종 변경 사항을 Remote Repository에 반영하는 게 더 좋은 방법.

GIT 사용 방법

❖ GitHub에 Remote Repository 관리 명령어 – git clone

- git clone.

➤ Remote Repository에 있는 project를 내 컴퓨터로 가져올 때 사용. 즉, GitHub에서 Local 환경으로 복사하는 작업. 이전에 생성했던 MyProjectRemote란 GitHub Remote Repository를 Local로 clone해 보자.

- 먼저 Remote Repository의 주소를 복사.

- 로컬 컴퓨터에서 프로젝트를 저장할 폴더를 생성한 후 git clone을 이용해 저장소를 복사.

C:/GitHub 폴더를 생성한 후 이 폴더 안에서 다음의 명령을 실행.

- git clone <https://github.com/moon9342/MyProjectRemote.git>

GIT 사용 방법

❖ GitHub에 Remote Repository 관리 명령어 – git clone

- 정상적으로 clone이 진행되면 저장소 이름으로 폴더가 하나 생성.
- 당연히 이 폴더는 Remote Repository와 연결되어 있는 Local Repository가 된다.

```
MINGW64:/c/GitHub
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub
$ git clone https://github.com/moon9342/MyProjectRemote.git
Cloning into 'MyProjectRemote'...
warning: You appear to have cloned an empty repository.

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub
$ dir
MyProjectRemote

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub
$
```

- ❖ GitHub에 Remote Repository 관리 명령어 – git remote
- GitHub에 빈 Remote Repository를 생성한 후 git remote를 이용하여 Local Repository와 연결할 수 있다.
- git remote add origin <https://github.com/moon9342/MyProjectRemote.git>
- 연결이 성공했는지를 다음의 명령어를 이용해서 확인해보자.
- git remote -v

GIT 사용 방법

❖ GitHub에 Remote Repository 관리 명령어 – git remote

```
MINGW64:/c/MyProject
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git remote add origin https://github.com/moon9342/MyProjectRemote.git

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git remote -v
origin  https://github.com/moon9342/MyProjectRemote.git (fetch)
origin  https://github.com/moon9342/MyProjectRemote.git (push)

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$
```

GIT 사용 방법

- ❖ GitHub에 Remote Repository 관리 명령어 – git push
- git remote를 이용하여 Local Repository와 Remote Repository가 연결되었으면 이제 자신이 작업한 내용을 Remote Repository에 upload 할 수 있다.
- 이 때 git push 명령을 이용하면 파일을 upload할 수 있는데 기본적으로 Remote Repository의 master branch에 upload되게 된다. 따라서 만약 다른 branch의 내용을 upload하려 하면 다음과 같이 명령을 실행시켜야 한다.
- git push origin “local branch명”
- 위의 명령에서 origin은 원격 저장소의 별칭. 이전에 git remote를 이용하여 Remote Repository를 연결할 때 이 origin이라는 별칭을 이용.

GIT 사용 방법

- ❖ GitHub에 Remote Repository 관리 명령어 – git push
- git remote를 이용하여 Local Repository와 Remote Repository가 연결되었으면 이제 자신이 작업한 내용을 Remote Repository에 upload 할 수 있다.
- 이 때 git push 명령을 이용하면 파일을 upload할 수 있는데 기본적으로 Remote Repository의 master branch에 upload되게 된다. 따라서 만약 다른 branch의 내용을 upload하려 하면 다음과 같이 명령을 실행시켜야 한다.
- git push origin “local branch명”
- 위의 명령에서 origin은 원격 저장소의 별칭. 이전에 git remote를 이용하여 Remote Repository를 연결할 때 이 origin이라는 별칭을 이용.

GIT 사용 방법

- ❖ GitHub에 Remote Repository 관리 명령어 – git push
- 만약 origin 저장소에 Local의 모든 branch를 push하려면 다음과 같은 명령을 수행.
 - git push origin --all
- git push가 진행될 때 Remote Repository에 같은 이름의 branch가 존재한다면 내용이 변경되고 만약 해당 branch가 존재하지 않는다면 새로운 branch를 Remote Repository에 생성. 같은 이름의 branch가 존재하지만 내역이 다르다면 당연히 push 작업은 거부된다.
- 예를 들자면 Local Repository의 master branch의 내용을 Remote Repository에 push하려면 다음과 같이 실행.
 - git push origin master

GIT 사용 방법

❖ GitHub에 Remote Repository 관리 명령어 – git push

```
MINGW64:/c/MyProject
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git remote add origin https://github.com/moon9342/MyProjectRemote.git

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git remote -v
origin  https://github.com/moon9342/MyProjectRemote.git (fetch)
origin  https://github.com/moon9342/MyProjectRemote.git (push)

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ git push origin master
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 8 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (18/18), 1.70 KiB | 434.00 KiB/s, done.
Total 18 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/moon9342/MyProjectRemote.git
 * [new branch]      master -> master

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/MyProject (master)
$ |
```

GIT 사용 방법

- ❖ GitHub에 Remote Repository 관리 명령어 – git fetch & pull
- Remote Repository를 이용하다 보면 다른 누군가가 먼저 commit할 경우가 있다.
이런 경우 git은 당연히 파일의 내용이 서로 상이하기 때문에 push를 허용하지 않는다.
- Remote Repository와 Local Repository를 적절히 서로 맞춰야 하며 이럴 때 사용하는 것이
git fetch. 즉, fetch는 Remote Repository의 commit들을 Local Repository로 가져오는 역할을 하
고 사용자는 Local Repository로 가져온 commit들을 자신의 작업과 적절히 병합하여 Remote
Repository에 push해야 한다.
- git pull은 Remote Repository의 정보를 가져오면서 자동으로 Local branch에 병합하는 명령어.
편하게 사용할 수는 있지만 만약 conflict가 발생하면 내역 확인이 쉽지 않은 단점이 있다.

GIT 사용 방법

- ❖ GitHub에 Remote Repository 관리 명령어 – git fetch & pull
- 만약 Local Repository에 Remote Repository의 branch가 존재하지 않는다면 git pull을 이용한다 해도 자동으로 Remote Repository에 있는 branch를 가져오지 못한다.
- Remote Repository에 있는 branch를 Local Repository로 가져오려면 다음과 같이 처리한다.
 - 우선 git remote에 대한 내역을 갱신한다. 해당 명령을 하지 않을 경우 나중에 branch를 찾지 못한다는 오류가 발생할 수 있기 때문
 - git remote update
 - 그 다음은 Remote Repository의 branch내역을 확인.
-a option을 이용하면 Remote Repository와 Local Repository의 모든 branch를 확인할 수 있다.
 - git branch -r

GIT 사용 방법

❖ GitHub에 Remote Repository 관리 명령어 – git fetch & pull

```
MINGW64:/c/GitHub/MyProjectRemote
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub/MyProjectRemote (master)
$ git remote update
Fetching origin
remote: Enumerating objects: 18, done.
remote: Counting objects: 100% (18/18), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 18 (delta 4), reused 18 (delta 4), pack-reused 0
Unpacking objects: 100% (18/18), done.
From https://github.com/moon9342/MyProjectRemote
 * [new branch]      master      -> origin/master
 * [new branch]      hotfix      -> origin/hotfix

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub/MyProjectRemote (master)
$ git branch -r
origin/hotfix
origin/master

shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub/MyProjectRemote (master)
$ |
```

GIT 사용 방법

❖ GitHub에 Remote Repository 관리 명령어 – git fetch & pull

```
MINGW64:/c/GitHub/MyProjectRemote  
  
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub/MyProjectRemote (master)  
$ git checkout -t origin/hotfix  
Switched to a new branch 'hotfix'  
Branch 'hotfix' set up to track remote branch 'hotfix' from 'origin'.  
  
shmoon@DESKTOP-JLVQBG9 MINGW64 /c/GitHub/MyProjectRemote (hotfix)  
$ |
```

❖ GitHub에 Remote Repository 관리 명령어 – git fetch & pull

- 이제 Local Repository에 동일 이름의 branch를 생성하면서 Remote Repository의 branch를 가져온 후 해당 branch로 checkout을 해보자.
- `git checkout -t <Remote_Branch_Name>`
- 만약 Local Repository에 같은 이름의 branch가 아니라 다른 이름의 branch를 생성하고 싶을 경우는 아래처럼 -b option을 이용.
- `git checkout -b <Local_Branch_Name> <Remote_Branch_Name>`

THANKS FOR LISTENING

A blurred background photograph of a person sitting at a desk, looking at a tablet. The tablet screen shows a blue and white bar chart. A person's hand is visible on the right side, pointing towards the tablet with their index finger. The overall scene suggests a presentation or a meeting.