Git & Github

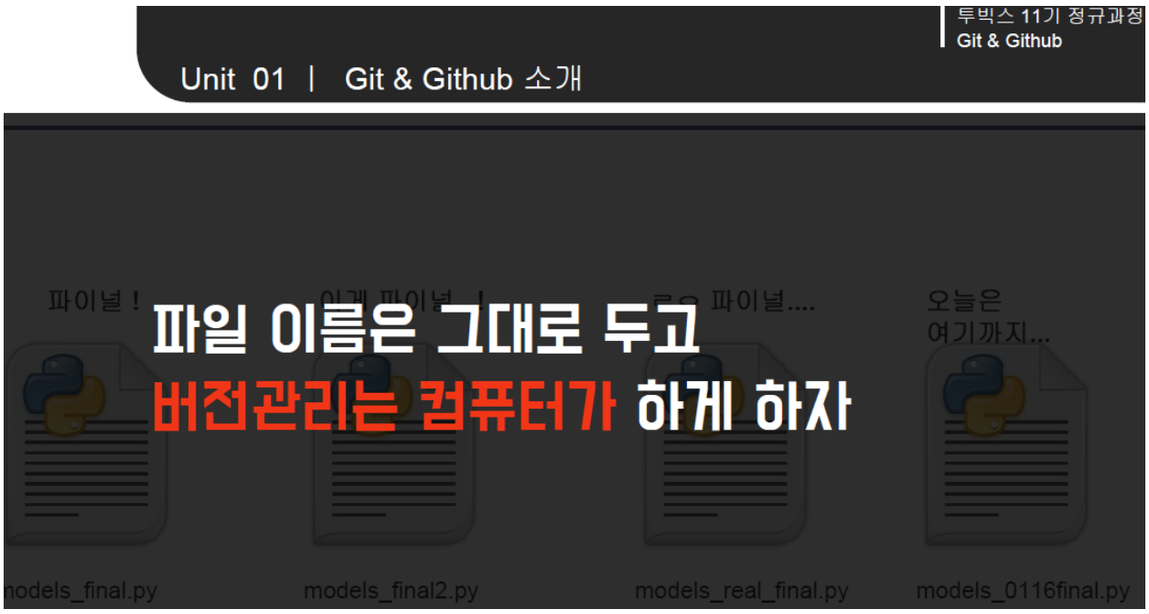
Files	
Tags	

#0. 다시 공부하는 Git & Github

- Remote 저장소에 있는 파일 저장 및 관리
- 협업이 가능하도록 저장소를 관리 하는 방법
- 기타 꼭 알아야하는 기본 지식부터 알아두면 좋은 팁까지!

⇒ 다시 공부하면서 하나하나 정복해나가자

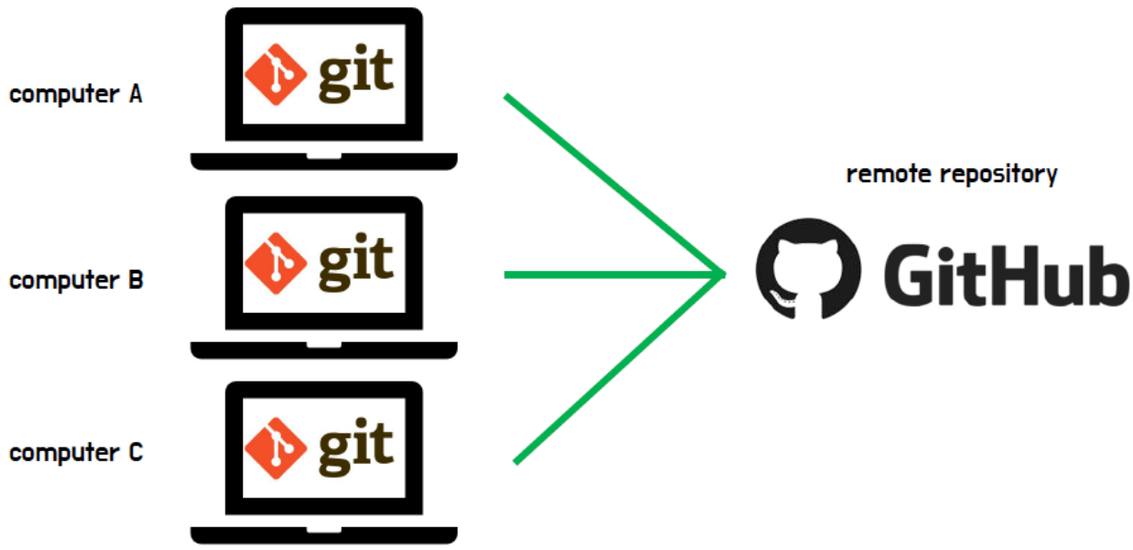
#1. Unit 1. Git & Github 소개



> 파일의 이름은 그대로 두고 버전 관리는 컴퓨터가 하게 하자

언제까지 model_v1.py, model_v2.py, model_v3.py... 이렇게 관리할래?

- Managing version
- Backup
- Recovery
- Collaboration



Summary

- Git is version management tool using local repository
- Github is remote repository for collaboration by multiple git users

생각을 해보자...

git은 로컬 저장소를 사용하는 버전관리 툴이야

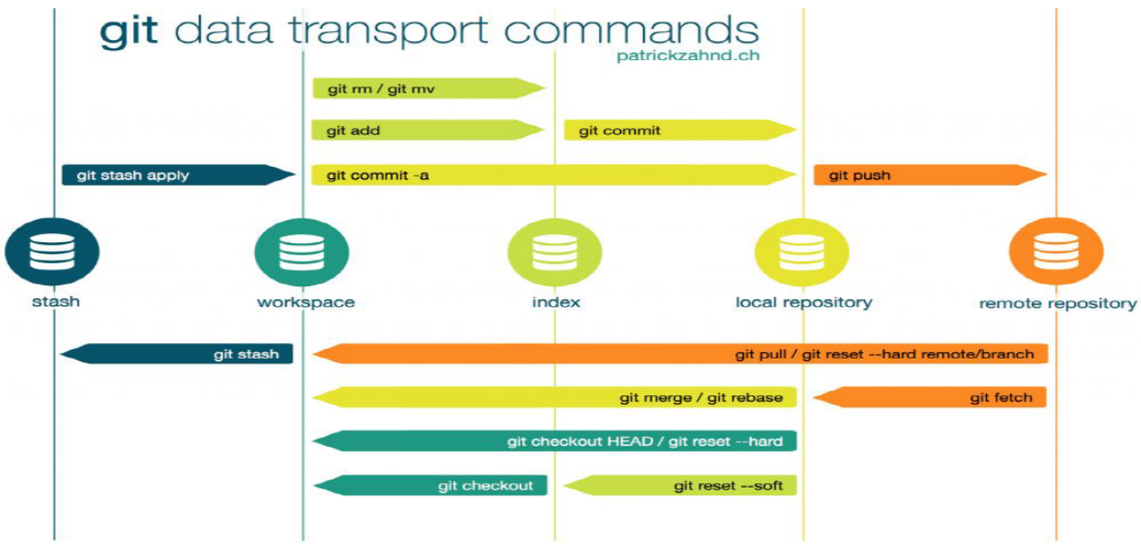
github은 원격 저장소야. 왜 원격 저장소지? 다양한 깃 유저들과 함께 협업하기 위함이야

keep going

#2. Git& Github 개념

git data transport commands

(git에서 데이터 전송 명령어들)

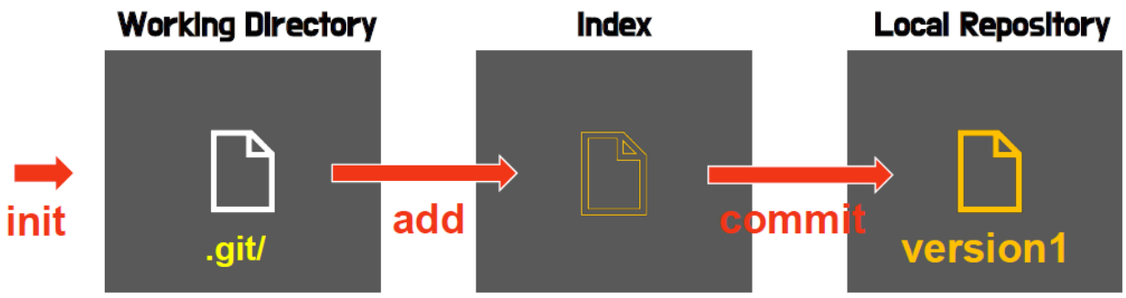


Working Directory, Local Repository, Remote Repository

⇒ 그래서 각각의 역할이 뭔데요?

- **Working Directory**: 내 PC안의 작업공간들 중 git을 사용하는 작업공간
- **Index**: 임시 버전들이 올라가는 공간
- **Local Repository**: 최종 확정본이 올라가는 공간개인 PC에 파일이 저장되는 개인 전용 저장소
- **Remote Repository**: 파일이 원격저장소 전용 서버에 올라가는 공간

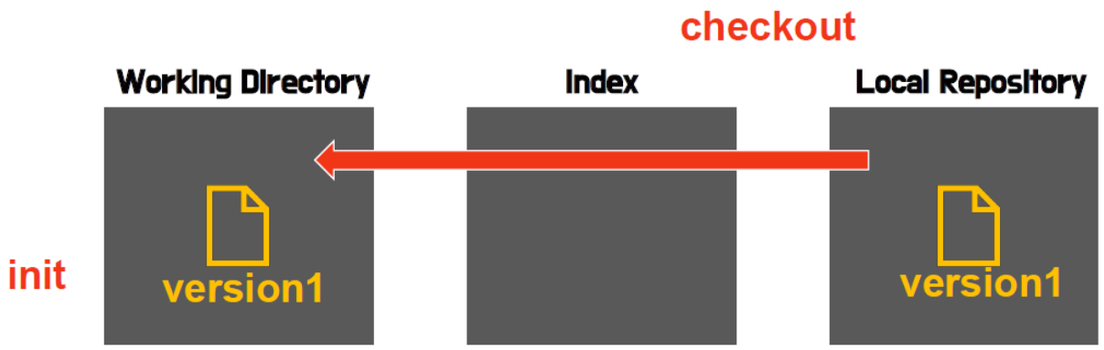
git init, add, commit



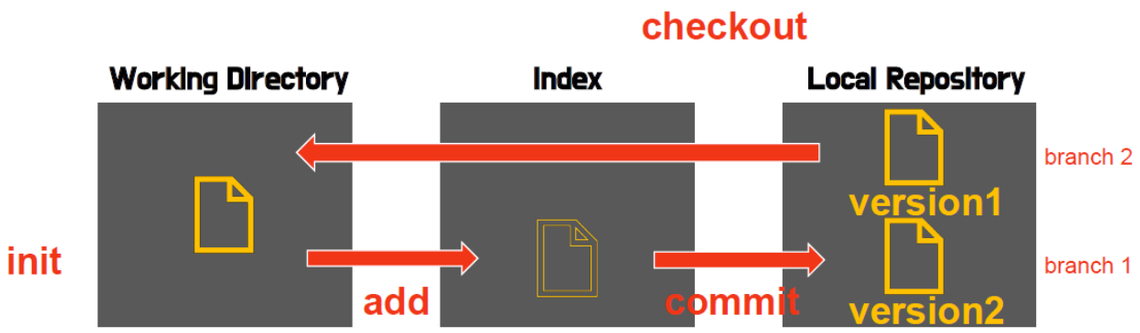
내 pc안에서 작업하다가 저장 또는 관리가 필요한 파일을을 add한다.

add한 파일들 중 최종 확정본을 commit을 통해 Local Repository에 올린다.

checkout

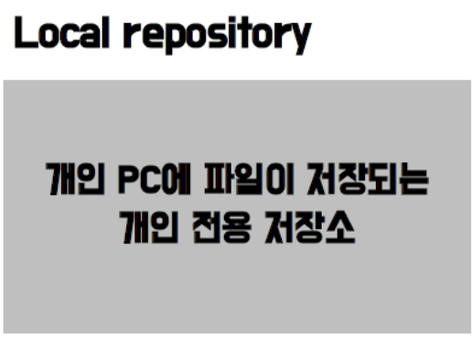


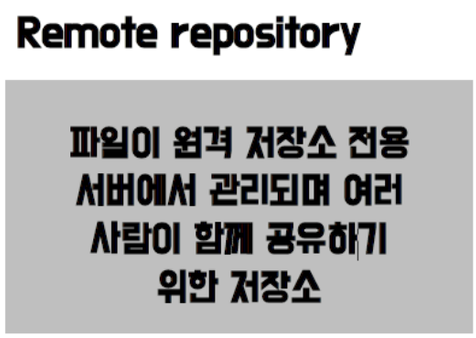
Checkout → add → commit



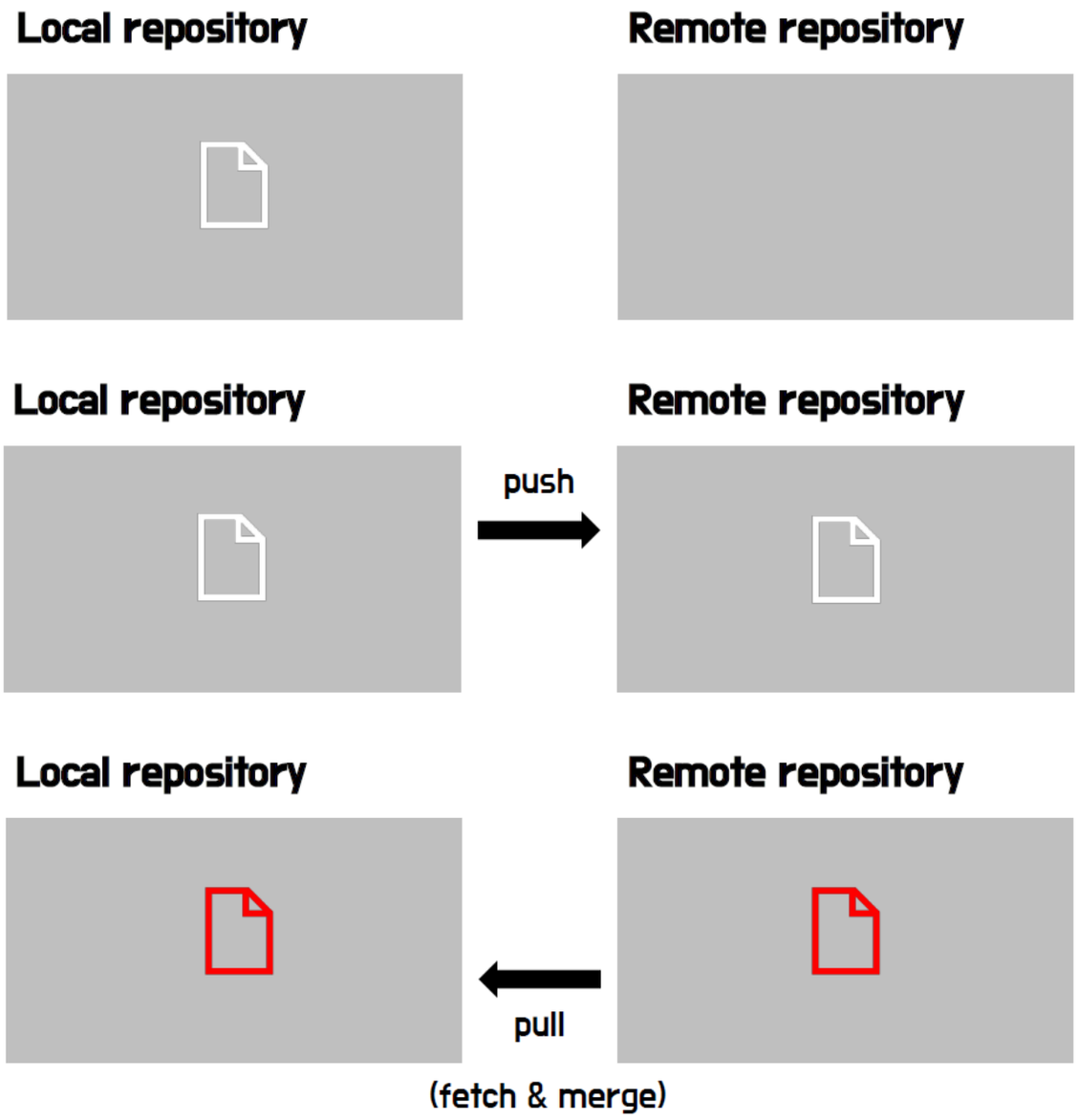
- Local repository와 Remote repository란 무엇인가?

→ 그래서 Local repository와 Remote repository가 뭔데?





- Example
  - Local repository: 연구실 컴퓨터, 집 데스크탑, 맥북
  - Remote repository: 여러 사람이 공유하기 위한 저장소, Steve-YJ github



Summary

이것 만큼은 알고 넘어가자

- Git이란 local repository의 버전관리를 위한 툴이다.
- Github은 원격서버를 통해 여러사람들이 협업할 수 있도록 파일을 저장하고 관리해준다.
- git init, git add, git commit

Git의 다양한 명령어

- git init : 현재 디렉토리에 git을 적용하겠다고 git에게 알림(start working area)
- git add : git에게 해당 파일을 버전 관리해라고 알림(staging area에 등록)
- git commit : 하나의 버전을 저장소 히스토리에 저장
- git push : 로컬 저장소에 있는 내용들을 원격 저장소에 저장
- git branch "branch_name" : "branch_name"의 브랜치를 생성
- git checkout "branch_name" : "branch_name"으로 이동
- git checkout -- "file_name" : 헌 브랜치에서 지웠던 파일 되살리기
- git merge "branch_name" : 현재 브랜치에서 "branch_name"을 merge
- git fetch : 원격 저장소에 있는 내용을 로컬 저장소에 반영
- git pull : 원격 저장소에 있는 내용을 로컬 저장소로 반영하고 브랜치를 병합함으로써 업데이트
- git log : 커밋 로그 상태를 확인
- git diff : 변경 사항 확인
- git reset "commit_ID" --hard : 해당 "commit_ID"의 상태로 돌아감
- git revert "commit_ID" : 해당 "commit_ID" 상태로 돌아가면서 새로운 버전으로 바꿈
- git "명령어" --help : 해당 명령어에 대한 설명, 옵션
- .git/ : 여러 파일들이 존재하는데 여기에 모든 로그 정보들이 담겨있고 이 정보들을 가지고 git이 실행됨
- index : 커밋대기상태(staging area)

Q. 내 깃에 있는 작업을 그대로 불러다가 하고 싶으면 git pull하면 되는거 아냐?!

⇒ 뒤에서 보다 자세히 배우게 됩니다.

#3. 실습

1) Git add와 commit

- git add

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git status
On branch master

No commits yet

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

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

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git add a.py
warning: LF will be replaced by CRLF in a.py.
The file will have its original line endings in your working directory

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ ^C

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git status
On branch master

No commits yet

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


Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$
```

- git commit

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git commit -m "first commit"
[master (root-commit) 805dfe6] first commit
 1 file changed, 2 insertions(+)
 create mode 100644 a.py

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git log
commit 805dfe68eaf0f65970c1ac2e9a18c062b8f9f25a (HEAD -> master)
Author: Steve-YJ <leeyoungjeon0511@gmail.com>
Date:   Sun Mar 8 18:36:34 2020 +0900

    first commit

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$
```

⇒ 이로써 Master branch에서 작업한 최종 확정본이 로컬 리퍼지토리(local repository)에 저장된다.

2) branch 생성과 merge

- test branch 생성과 Checkout 명령어를 통한 test 브랜치로 이동

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git branch test

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git branch -a
* master
  test

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git checkout test
Switched to branch 'test'

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (test)
$
```

- a.py 수정

```
2020.03.08.sun YJ git-learning
my name is Youngjeon Lee
```

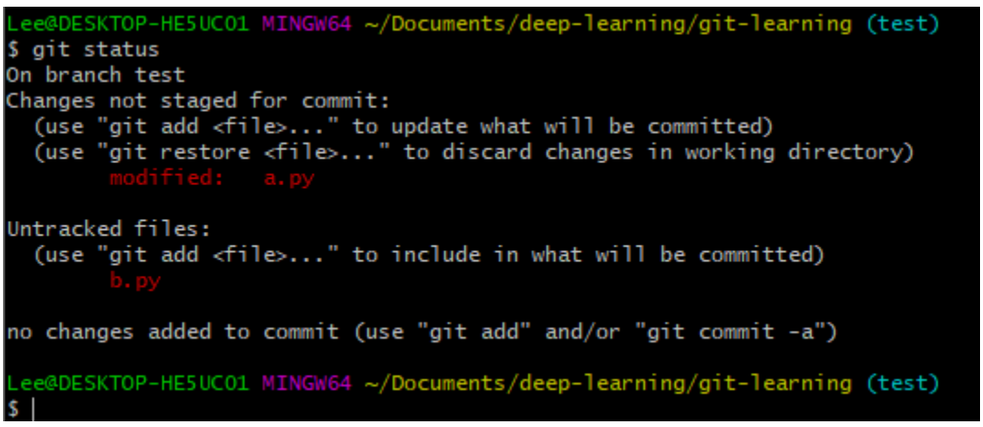
—my name is Youngjeon Lee 라는 기존 a.py파일에

—2020.03.08 sun YJ git-learning를 추가한다.

- b.py 작업

```
def tobigs(s):
    pass
```

- git status



⇒ Let's do it

해석해보자!

기존 master branch에서 test 브랜치로 이동을 한 상태에서 다음의 작업을 수행하였다.

1. a.py 파일 수정(맨 윗줄에 라인 추가)
2. b.py파일 생성

git status 명령어를 입력했을 때 위의 사진과 같은 출력을 얻게된다.

a.py의 경우 changes not staged for commit

업데이트 내역을 commit 하기 위해 git add 명령어 사용할 것을 권한다.

b.py의 경우 새로 생성했지만 git add를 해주지 않았기에 Untracked file이라는 메세지를 볼 수 있다.

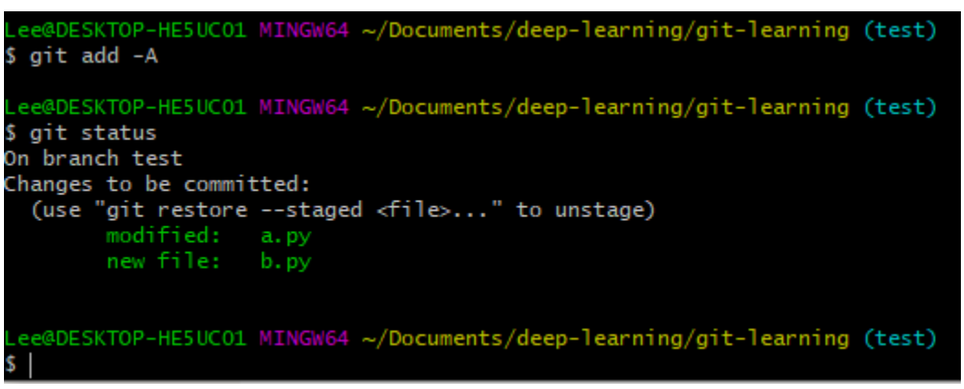
⇒ 그렇다면 이제 내가 해야할 일은

git add a.py b.py가 아닐까?

- git add -A

git add -A명령어를 통해 워킹디렉터리(working directory)내에 있는 모든 파일들을 추적할 수 있다.

master에서 했던 작업과 마찬가지로 commit을 해줘야 원격저장소를 통해 관리할 수 있게 된다.



- git commit -m "edit a.py and make b.py"

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (test)
$ git commit -m "edit a.py and make b.py"
[test 1b51f01] edit a.py and make b.py
 2 files changed, 3 insertions(+)
 create mode 100644 b.py

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (test)
$
```

—commit message로 나는 a.py파일을 수정했고 b.py를 만들었어! 라고 입력한다.

- git log —branches —decorate —graph —oneline

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (test)
$ git log --branches --decorate --graph --oneline
* 1b51f01 (HEAD -> test) edit a.py and make b.py
* 805dfe6 (master) first commit

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (test)
$
```

지금까지는 test branch에서 작업을 했다.

1. Master branch에서 a.py파일을 만들었고 test branch로 이동해서 작업을 했다.
2. test branch에서는 a.py파일을 수정했으며 b.py파일을 만들었다.
3. 다시 master branch로 이동한다.

- master 브랜치 이동/edit a.py
  - master 브랜치 이동 및 a.py 수정

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (test)
$ git checkout master
Switched to branch 'master'

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ vim a.py
```

```
my name is Youngjeon Lee
Let's git it!
```

- 수정된 a.py를 tracking하도록 하고 commit해준다.

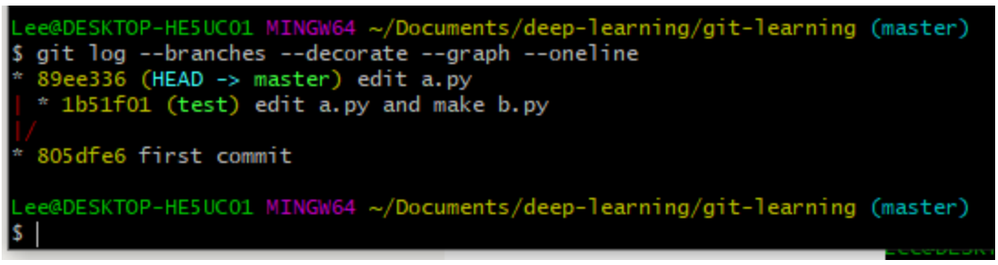
```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   a.py

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

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git add a.py

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git commit -m "edit a.py"
[master 89ee336] edit a.py
 1 file changed, 1 insertion(+)
```

- git log —branches —decorate —graph —oneline



위의 그림은 지금까지 한 작업들을 보여준다. 해석해보면 지금까지 작업한 내용들에 대해서 보다 이해가 잘 가리라 생각한다.

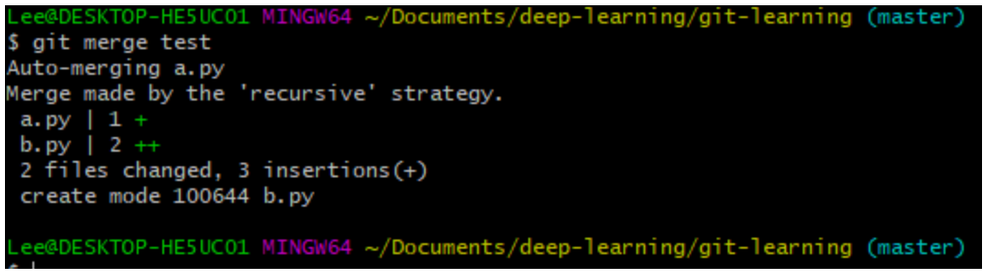
- 해석

1. Master branch: 우리는 master branch에서 처음 작업을 했다. a.py파일을 생성하고 git이 a.py파일을 추적하도록 하여 first commit을 했다.
2. Checkout to test branch: master branch에서 test branch로 체크아웃을 했다. 여기서 주목할 점은 master branch에서 생성한 a.py수정과 b.py생성이다. master branch에서 생성한 a.py파일을 test branch로 이동해 작업을 했으며 b.py라는 새로운 파이썬 파일을 생성했다. 작업한 내용을 'edit a.py and make b.py'라는 메세지와 함께 commit 했다.
3. Checkout from test branch to master branch: test branch에서 다시 master branch로 돌아왔다. 여기서 한 작업은 a.py 파일을 수정한 것이다.

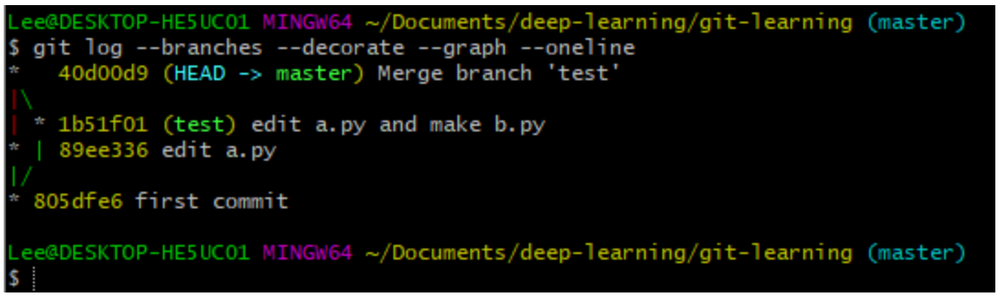
⇒ 이제 버전관리가 어떻게 이루어지는지 살펴보도록 하자.

---

master 브랜치에서 test 브랜치 merge



master브랜치에서 test 브랜치를 merge했다. git log —branches —decorate —graph —oneline 명령어를 통해 작업한 내용들을 살펴보자.



예전에는 이 부분이 이해가 잘 안갔었는데 이제 알것 같다.

브랜치를 보면 지금까지 했던 작업들을 직관적으로 보여준다.

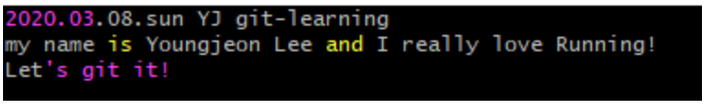
1. master branch - first commit
2. test branch — edit a.py and make b.py

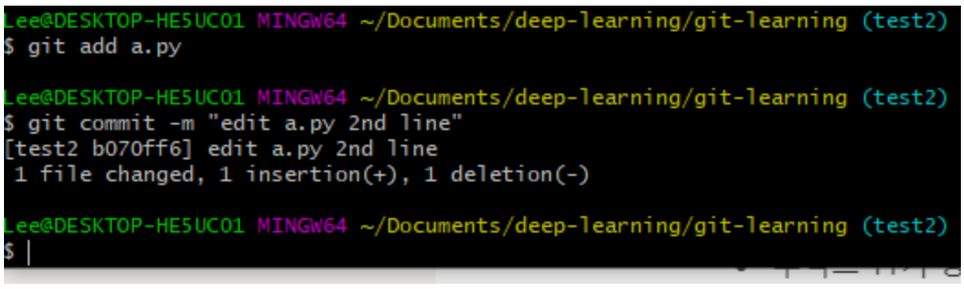
   master branch — edit a.py
3. masterbranch — Merge branch 'test'

항상 merge가 잘 될까?

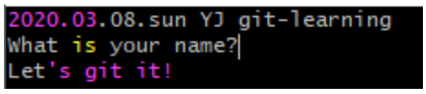
- 어떤 경우 merge가 되지 않는지 알아보자

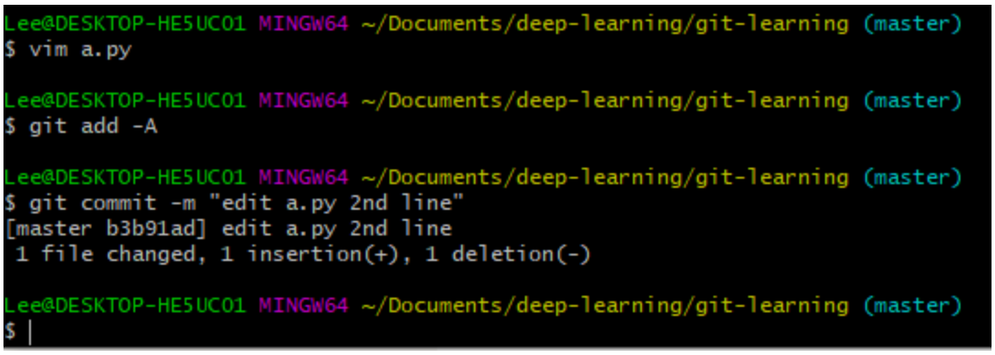
test2 브랜치 생성 후 이동 / edit a.py





master 브랜치로 이동/edit a.py





⇒ **주의!**

test2 브랜치와 master 브랜치 모두 a.py 파일의 2번째 줄을 수정했다.

생각해보자. 회의록을 작성하는데 나는 두 번째 줄에 "우리 회사의 수익은 10000이다."라고 작성했는데 나의 동료는 "우리 회사의 수익은 20000이다."라고 작성을 했다. 이 때 두 파일을 합치면 버전관리가 될까?

10000이야? 20000이야?!
같은 문제가 코드에서 발생하는 것이다. 충돌이 나는 것이다.

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git merge test2
Auto-merging a.py
CONFLICT (content): Merge conflict in a.py
Automatic merge failed; fix conflicts and then commit the result.

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master|MERGING)
$
```

> CONFLICT (content): Merge conflict in a.py

⇒ 다시 a.py에 들어가서 충돌이 일어난 부분을 수정해야 한다.

```
1 # 20190116 tobigs seminar
2 <<<<<<< HEAD
3 # what is your name?
4 =======
5 # my name is Yunho Jung and I love music
6 >>>>>>> test2
7 # Let's git it !
```

⇒ 충돌이 난 부분을 수정해주면 병합을 할 수 있게 된다.

```
2020.03.08.sun YJ git-learning
What is your name?
my name is Youngjeon Lee and I really love Running!
Let's git it!
```

```
Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master|MERGING)
$ vim a.py

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master|MERGING)
$ git add -A

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master|MERGING)
$ git commit -m "solve a conflict and merge"
[master 41f3cee] solve a conflict and merge

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$ git log --branches --decorate --graph --oneline
*   41f3cee (HEAD -> master) solve a conflict and merge
|\
| * b070ff6 (test2) edit a.py 2nd line
* | b3b91ad edit a.py 2nd line
|/
*   40d00d9 Merge branch 'test'
|\
| * 1b51f01 (test) edit a.py and make b.py
* | 89ee336 edit a.py
|/
* 805dfe6 first commit

Lee@DESKTOP-HE5UC01 MINGW64 ~/Documents/deep-learning/git-learning (master)
$
```

#4. 유용한 팁

1. 원격저장소와 연결

- 원격저장소를 관리할 수 있는 명령어 git remote를 이용하여 연결
- git remote add origin 사용자명@호스트:/원격/저장소/경로

(git clone 사용자명@호스트:/원격/저장소/경로)

2. 원격 저장소에서 로컬 저장소로 업데이트하는 방법

원격 저장소 → 로컬 저장소 pull(fetch & merge)

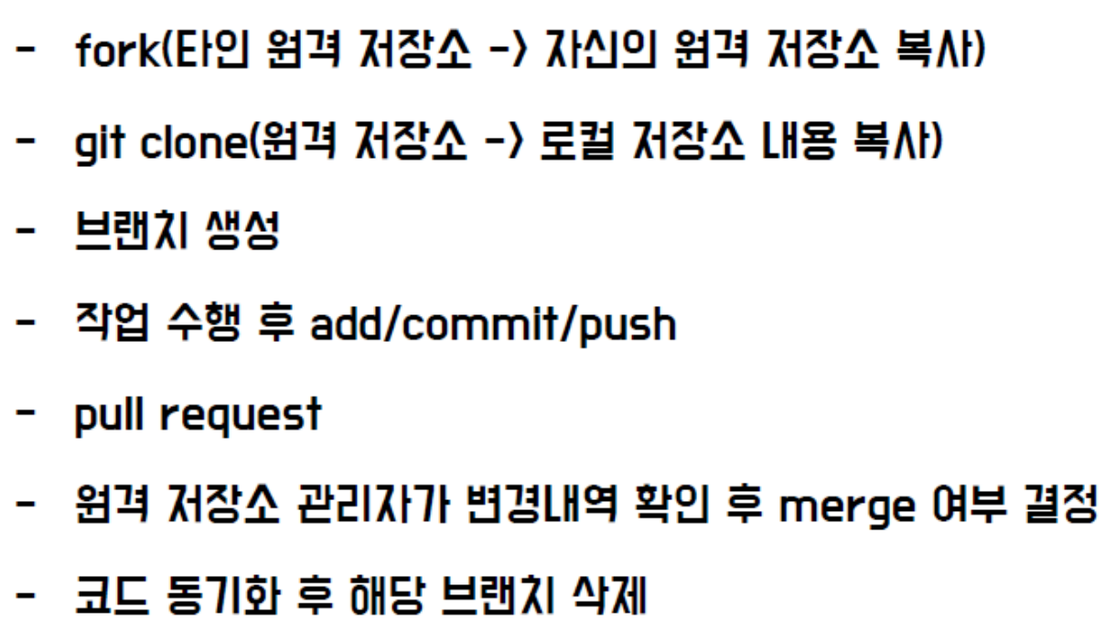
- 원격 저장소의 내용을 로컬 저장소로 업데이트(다른 사람이 수정한 것을 받기 위해)
- git pull origin master(브랜치명) 실행
- origin의 내용이 master로 반영(fetch)되고 브랜치를 병합(merge)

3. git configuration

- git pull을 할 때는 깃허브의 유저이름과 비밀번호를 쳐야 하는 경우가 많다.
- 매번 비밀번호를 입력하기 싫다면
  - git config —global credmential.helper 'store —file 경로'하면 해당 경로에 비밀번호가 저장된 파일이 생성된다.
  - 단, 파일로 저장되는 만큼 보안에 취약하기 때문에 주의해야 한다.

4. 많이 쓰이는 패턴

많이 쓰이는 패턴



타인의 원격 저장소를 복사해 온다든지

원격 저장소의 내용을 로컬로 복사해온다든지

브랜치를 생성후 작업을 한다든지

원격 저장소의 관리자가 변경 내역을 확인 후 merge여부를 결정한다든지!

#8. Reference

- 투빅스 11기 정규세션 - 정윤호의 Git&GitHub