1. git 관련 문서를 찾아 읽고 git 에 있는 개념들과 할 수 있는 것들에 대해 과제 본문에 정리한다. 다음 링크들을 (모두) 읽어보고 정리한다.

(1) git

- 깃이란? 깃은 프로젝트의 어떤 부분도 겹쳐쓰지 않게 프로젝트의 변경을 관리하는 버전관리 소프트웨어이다.

- 왜 깃을 사용해야 하는가? 깃과 같은 버전고나리 앱은 공동작업을 할 때 누군가의 작업이 겹쳐쓰여지거나 지워지는 것을 방지한다.

각각이 같은 페이지에 각자의 수정사항을 업로드 할 수 있고 나중에 그대로 어떤 작업도 잃어버리지 않고 병합할 수 있다.

깃은 이전에 만들어진 모든 변경사항의 스냅샷을 저장하기 때문에 이전 시점의 어떤 버전으로 되돌릴 수도 있다.

- 깃허브(GitHub)란? 분산 버전 관리 룰인 깃을 사용하는 프로젝트를 지원하는 웹 호스팅 서비스이다. (오픈 소스 코드 저장소)

- 깃허브가 깃을 어떻게 편리하게 해주는가?

ㄱ. 깃허브 소트프웨어를 다운로드하면, 로컬에서 각자의 프로젝트를 관리할 수 있게하는 비주얼 인터페이스를 제공

ㄴ. Github.com에 계정을 생성하면 웹에서 프로젝트를 버전관리할 수 있음

- 다른 깃허브 사용자의 프로젝트도 둘러볼 수 있다. 또한 다른 사용자도 나의 공개 프로젝트에 대해 같은 걸 할 수 있고 에러를 발견하여 해결책을 제안할 수도 있다.

- 깃허브의 특징

ㄱ. 저장소에 팀원초대는 무제한

ㄴ. 개인 저장소는 유료(공개되지 않는 저장소), 무료로 개인 저장소를 제공하는 다른 Git웹 서비스들이 존재

ㄷ. Pull->Pull Request형식이다.

- 깃허브의 기본 용어

ㄱ. 커맨트 라인(Command Line): 깃 명령어를 입력할 때 사용하는 컴퓨터 프로그램. 프롬프트로 알려진 텍스트 기반 명령어를 입력한다.

ㄴ. 저장소(Repository): 프로젝트가 거주할 수 있는 디렉토리나 저장 공간. 컴퓨터 안의 로컬 폴더가 될 수도 있고, 깃허브나 다른 온라인 호스트의 저장 공간이 될 수도 있다.

저장소 안에 코드 파일, 텍스트 파일, 이미지 파일을 저장하고, 이름을 붙일 수 있다.

ㄷ. 버전관리(Version Control): 기본적으로, 깃이 서비스되도록 고안된 목적으로, 프로젝트 히스토리의 모든 시점의 스냅샷을 유지하므로, 잃어버리거나 겹쳐쓰지 않을 수 있다.

ㄹ. 커밋(Commit): 깃에게 파워를 주는 명령. 커밋을 한 시점의 저장소의 스냅샷을 찍어, 프로젝트를 이전의 어떠한 상태로든 재평가하거나 복원할 수 있는 체크포인트를 가질 수 있다.

ㅁ. 브랜치(Branch): 작업자들은 메인 프로젝트의 브랜치를 따와서(branch off), 자신이 변경하고 싶은 자신만의 버전을 만든다.

작업을 끝낸 후, 프로젝트의 메인 디렉토리인 master에 브랜치를 다시 Merge한다.

- 깃허브 주요 명령어

ㄱ. git init: 깃 저장소를 초기화한다. 저장소나 디렉토리 안에서 이 명령을 실행하기 전까지는 그냥 일반 폴더이고, 이것을 입력한 후에야 추가적인 깃 명령어들을 줄 수 있다.

ㄴ. git config: 처음에 깃을 설정할 때 가장 유용

ㄷ. git help: 명려어가 기억이 나지 않을 때 커맨드 라인에 이걸 타이핑 하며 21개의 가장 많이 사용되는 명령어들이 나타난다.

(더 자세히 “git help init”이나 다른 용어를 타이핑하여 특정 깃 명령어를 사용하고 설정하는 법을 이해할 수도 있다.)

ㄹ. git status: 저장소 상태를 체크. (어떤 파일이 저장소 안에 있느지, 커밋이 필요한 변경사항이 있는지-현재 저장소에 있는 내용과 로컬 컴퓨터에 있는 내용이 얼마나 다른지에 대해서 알려줌 , 현재 저장소의 어떤 브랜치에서 작업하고 있는지 알 수 있음. )

따라서, 로컬 컴퓨터에서 작업한 내용을 저장소에 올리기 전에 확인할 때, 주로 많이 사용될 명령어이다.

ㅁ. git add: 이 명령이 저장소에 새 파일을 추가하는 것이 아니라 대신, 깃이 새 파일들을 지켜보게 한다. 곧, 수정된 파일을 저장소에 올릴 준비를 하는 것이다. (git에 수정된 사항을 덧붙이겠다라는 의미)

파일을 추가하면, 깃의 저장소 스냅샷에 포함된다.

ㅂ. git commit: 깃의 가장 중요한 명령어. 어떤 변경사항이라도 만든 후, 저장소의 스냅샷을 찍기 위해 이것을 입력한다. commit 하면 저장소에 올릴 것을 최종확정 하는 것이다.

또, 몇 개의 파일이 바뀌었는지, 얼마나 수정되었는지 출력이 된다. 하지만, 아직 저장소에 반영이 되진 않았고, 로컬 컴퓨터에서만 확정이 된 것.

ㅅ. git branch: 공동 작업하고 자신만의 변경을 원할 때 사용하는 명령어이다. 이 명령어는 새로운 브랜치를 만들고, 자신만의 변경사항과 파일 추가 등의 커밋 타임라인을 만든다.

내가 지은 제목이 명령어 다음에 온다. 새 브랜치를 “cats”로 부르고 싶으면, git branch cats를 타이핑하면 된다.

ㅇ. git checkout: 글자 그대로, 현재 위치하고 있지 않은 저장소를 체크아웃할 수 있다. 이것은 체크하길 원하는 저장소로 옮겨가게 해주는 탐색 명령이다.

master 브랜치를 들여다 보고 싶으면, git checkout master를 사용할 수 있고, git checkout cats로 또 다른 브랜치를 들여다 볼 수 있다.

ㅈ. git merge: 브랜치에서 작업을 끝내고, 공동 작업자들이 볼 수 있는 master 브랜치로 병합할 수 있다.

(git merge cats는 “cats” 브랜치에서 만든 모든 변경사항을 master로 추가한다.)

ㅊ. git push(수정 사항을 저장소에 반영하기): 로컬 컴퓨터에서 작업하고 당신의 커밋을 깃허브에서 온라인으로도 볼 수 있기를 원할 때 사용한다. git remote -v 명령어를 통해 저장소의 URL을 확인하고, 그 이름을 통해서 push(저장소에 실제 반영)를 하게 됨

ㅋ. git pull (수정된 저장소 내용 불러오기): 수정된 부분만 받는 명령어(공동 작업을 하는데, 다른 사람이 수정한 내용을 불러와 로컬 컴퓨터의 파일들을 최신화 시킬 때. 즉, 로컬 컴퓨터에서 작업할 때 작업하고 있는 저장소의 최신 버전을 원하면 이 명령어로 깃허브로부터 변경사항을 다운로드)

ㅌ. git clone(저장소 가져오기): 이 명령어를 통해 저장소에 있는 내용을 로컬 컴퓨터로 불러올 수 있다. clone을 하게 되면 git init까지 자동으로 설정

ㅍ. git remote(저장소 URL 설정하기): 이 명령어는 git url을 설정할 때 주로 사용

- 깃/깃허브 설정하기

로컬 컴퓨터에서 작업하려면 깃을 설치해야 한다. 커맨드 라인으로 넘어가서 git config--global user.name "SoHyun"을 입력하고

다음엔 git config--global user.email "chsh981127@naver.com" 나의 이메일을 입력한다. 깃허브를 가입할때 사용한 이메일로 해야한다.

- 깃허브 저장소 만들기

온라인 저장소 만들기: Repository name에 저장소 이름을 쓰고, Description에 저장소에 대한 설명을 작성할 수 있다.

Public, Private으로 공공 저장소를 만들지, 개인 저장소를 만들지 선택 가능(개인 저장소는 유료)

README를 만들면, 저장소에 대한 상세 설명 등을 쓸 수 있다.

-로컬 저장소 만들기

컴퓨터에서 작업할 것이므로, 로컬 디렉토리에 만들 저장소에 실제로 미러링해야 한다.

mkdir ~/MyProject를 타이핑 하고 -> 디렉토리를 만들었고

cd ~/Myproject를 타이핑한다. (cd: 탐색 명령어) -> 그 디렉토리로 옮겨 들어갔다.

다음 줄에 git init을 입력하여 렉토리를 로컬 깃 저장소라고 컴퓨터에게 말해준다. 컴퓨터는 이제 이 디렉토리를 Git-ready로 인식하고, 깃 명령어를 입력할 수 있다.

즉, 폴더를 하나 만들고 그 안에서 git init 명령을 실행하면 새로운 git저장소가 만들어진다.

- 저장소 받아오기

\*로컬 저장소를 복제(clone)하려면 아래 명령을 실행하면 된다.

[git clone /로컬/저장소/경로]

\* 원격 서버의 저장소를 복제하려면 아래 명령을 실행하세요.

git clone 사용자명@호스트:/원격/저장소/경로

- 작업의 흐름

로컬 저장소는 git이 관리하는 세 그루의 나무로 구성되어 있다.

첫번째 나무인 작업 디렉토리(Working directory)는 실제 파일들로 이루어져있고, 두번째 나무인 인덱스(Index)는 준비 영역(staging area)의 역할을 하며, 마지막 나무인 HEAD는 최종 확정본(commit)을 나타낸다.

(1) 추가와 확정(commit)

변경된 파일은 아래 명령어로 (인덱스에) 추가할 수 있다.

git add <파일 이름>, git add \*

이것이 바로 git의 기본 작업 흐름에서 첫 단계. 하지만 실제로 변경 내용을 확정하려면 [git commit -m "이번 확정본에 대한 설명"] 이 명령어를 내려야 한다.

이제 변경된 파일이 HEAD에 반영되었으나 원격 저장소에는 아직 반영이 안 되었다. 따라서, 변경 내용을 발행(push)한다.

현재의 변경 내용은 아직 로컬 저장소의 HEAD 안에 머물고 있다. 이 변경 내용을 git push origin master 이 명령어를 사용하여 원격 서버로 올리면 원격저장소에 저장된다.

(다른 가지를 발행하려면 master를 원하는 가지 이름으로 바꿔주면 된다.)

만약 기존에 있던 원격 저장소를 복제한 것이 아니라면, 원격 서버의 주소를 git에게 알려줘야 한다.

=> git remote add origin <원격 서버 주소>

이제 변경 내용을 원격 서버로 발행할 수 있게 되었다.

(2) 가지(branch)치기

가지는 안전하게 격리된 상태에서 무언가를 만들 때 사용한다. 저장소를 새로 만들면 기본으로 master 가지가 만들어진다.

이제 다른 가지를 이용해서 개발을 진행하고, 나중에 개발이 완료되면 master 가지로 돌아와 병합하면 된다.

git checkout -b feature\_x 명령으로 "feature\_x"라는 이름의 가지를 만들고 갈아탄다.

git checkout master 명령으로 master 가지로 돌아올 수 있다.

git branch -d feature\_x 명령으로는 가지를 삭제할 수 있다.

내가 새로 만든 가지를 원격 저장소로 전송하기 전까지는 다른 사람들이 접근할 수 없다.

=>git push origin <가지 이름>

(3) 갱신과 병합(merge)

나의로컬 저장소를 원격 저장소에 맞춰 갱신하려면 git pull 명령을 실행한다.

이렇게 하면 원격 저장소의 변경 내용이 로컬 작업 디렉토리에 받아지고(fetch), 병합(merge)된다.

다른 가지에 있는 변경 내용을 현재 가지(예를 들면, master 가지)에 병합하려면 git merge <가지 이름> 명령을 실행하면 된다.

첫번째 명령이든 두번째 명령이든, git은 자동으로 변경 내용을 병합하려고 시도하는데, 항상 성공하는 게 아니라 가끔 충돌(conflicts)이 일어나기도 한다.

이렇게 충돌이 발생하면, git이 알려주는 파일의 충돌 부분을 직접 수정해서 병합이 가능하도록 해야 한다.

충돌을 해결했다면, git add <파일 이름>명령으로 git에게 아까의 파일을 병합하라고 알려주면 된다.

변경 내용을 병합하기 전에, 어떻게 바뀌었는지 비교해볼 수도 있다. [git diff <원래 가지> <비교 대상 가지>]

(4) 꼬리표(tag) 달기

소프트웨어의 새 버전을 발표할 때마다 꼬리표를 달아놓으면 좋다.

[git tag 1.0.0 1b2e1d63ff] 명령을 실행하면 새로운 꼬리표인 1.0.0을 달 수 있다.

위 명령에서 1b2e1d63ff 부분은 꼬리표가 가리킬 확정본 식별자이다.

git log 명령으로 확정본 식별자를 얻을 수 있다. 확정본 식별자의 앞부분 일부만 입력해도 꼬리표를 붙일 수 있지만,

그 일부분이 반드시 고유하다는 조건이 필요.

(5) 로컬 변경 내용 되돌리기

실수로 무언가 잘못한 경우, [git checkout -- <파일 이름>] 명령으로 로컬의 변경 내용을 되돌릴 수 있다.

위 명령은 로컬의 변경 내용을 변경 전 상태(HEAD)로 되돌려주지만, 이미 인덱스에 추가된 변경 내용과 새로 생성한 파일은 그대로 남는다.

만약, 로컬에 있는 모든 변경 내용과 확정본을 포기하려면,

[git fetch origin, git reset --hard origin/master] 명령으로 원격 저장소의 최신 이력을 가져오고, 로컬 master 가지가 저 이력을 가리키도록 할 수 있어요.

2. 파이썬 기초 (1) 문서를 읽고 코드를 따라 쳐 본다. 다르게 변형해서도 해 보도록 한다.

- 해 본 결과를 본문에 입력하고 각각에 대한 느낀점을 적도록 한다.

[결과]

Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC v.1924 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

>>># 더하기, 나누기, 나누기, 나머지 연산

>>> 1+2

3

>>> 100-1

99

>>> 7\*52

364

>>> 1024/8

128.0

>>> 1/2

0.5

>>> 1//2

0

>>> 4/3

1.3333333333333333

>>> 4//3

1

>>> 7%3

1

>>> 7.5%3

1.5

>>> 4%2.1

1.9

>>> #원의 넓이 구하기

>>> 3.141592653589793\*(3\*3)

28.274333882308138

>>> 3.141592653589793\*3\*\*2

28.274333882308138

>>> 3.141592653589793\*(9\*\*2)

254.46900494077323

>>> 3.141592653589793\*(20\*\*2)

1256.6370614359173

>>> 3.141592542489793\*(1.5\*\*2)

7.068583220602034

>>> #변수

>>> pi=3.141502653589793

>>>

>>> pi\*4\*\*2

50.26404245743669

>>> pi\*2.5\*\*2

19.634391584936207

>>> pi\*8\*\*2

201.05616982974675

>>> #print 함수

>>> print(pi)

3.141502653589793

>>>

>>> pi=3.141592653589793

>>>

>>> print(pi)

3.141592653589793

>>>

>>> r=7

>>>

>>> area=pi\*r\*\*2

>>>

>>> print(area)

153.93804002589985

>>>

>>> r=11

>>>

>>> area=pi\*r\*\*2

>>>

>>> print(area)

380.132711084365

>>> #변수에는 문자열을 담을 수 있다.

>>> name="Trump"

>>> name1="Trump"

>>> name2= '강다니엘'

>>>

>>>

>>> print(name1)

Trump

>>> print(name2)

강다니엘

>>>

>>> order=4

>>> pi=3.141592

>>> name1="Trump"

>>> name2='Daehyun'

>>>result=True

>>> #자료형 구하기

>>> type(order)

<class 'int'>

>>> type(pi)

<class 'float'>

>>> type(name1)

<class 'str'>

>>> type(name2)

<class 'str'>

>>> type(result)

<class 'bool'>

>>> myname=sohyun

Traceback (most recent call last):

File "<pyshell#59>", line 1, in <module>

myname=sohyun

NameError: name 'sohyun' is not defined

>>> myname"소현"

SyntaxError: invalid syntax

>>> myname="소현"

>>> print(myname)

소현

>>> type(myname)

<class 'str'>

>>>

>>> #비교 연산

>>> 4>3

True

>>> 3.5<-1.3

False

>>> 100==100

True

>>> 10>=10

True

>>> a=365>366

>>> print(a)

False

>>> type(a)

<class 'bool'>

>>>

>>>

>>> first="Daehyun"

>>> first="Sohyun"

>>> last='cho'

>>> last='Cho'

>>> name=first+' '+last

>>> name

'Sohyun Cho'

>>> print(name)

Sohyun Cho

>>>

>>> name\*2

'Sohyun ChoSohyun Cho'

>>> name\*3

'Sohyun ChoSohyun ChoSohyun Cho'

>>>

>>> name[0]

'S'

>>> name[2]

'h'

>>> name[-1]

'o'

>>> name[-2]

'h'

>>>

>>> # name[start:stop:step]

>>> # 슬라이스- 문자열의 일부분을 잘라내는 기법

>>> title="Python 2D Game Programming"

>>> title[0:6]

'Python'

>>> title[7:9]

'2D'

>>> title[10:14]

'Game'

>>> title[:6]

'Python'

>>> title[-11:]

'Programming'

>>> title[::2]

'Pto DGm rgamn'

>>> title[::-1]

'gnimmargorP emaG D2 nohtyP'

>>> #List

>>> twice=['momo', 'sana', 'zwi', 'nayun', 'dahyun']

>>> bts=['jungkook', 'v', 'jimin', 'j-hope', 'suga', 'jin', 'rm']

>>> twice

['momo', 'sana', 'zwi', 'nayun', 'dahyun']

>>> twice.append('jihyo')

>>> twice

['momo', 'sana', 'zwi', 'nayun', 'dahyun', 'jihyo']

>>> twice.sort()

>>> twice

['dahyun', 'jihyo', 'momo', 'nayun', 'sana', 'zwi']

>>> len(twice)

6

>>> unite=twice+bts

>>> unite

['dahyun', 'jihyo', 'momo', 'nayun', 'sana', 'zwi', 'jungkook', 'v', 'jimin', 'j-hope', 'suga', 'jin', 'rm']

>>> unite.remove('momo')

>>> unite

['dahyun', 'jihyo', 'nayun', 'sana', 'zwi', 'jungkook', 'v', 'jimin', 'j-hope', 'suga', 'jin', 'rm']

>>>#List에서 Slice가 적용됨

>>> unite[0]

'dahyun'

>>> unite[-1]

'rm'

>>> unite[:3]

['dahyun', 'jihyo', 'nayun']

>>> unite[-3:]

['suga', 'jin', 'rm']

>>> # Dictionary

>>> score={'momo':80,'zwi':85,'sana':98}

>>> type(score)

<class 'dict'>

>>> score['momo']

80

>>> score['nayun']

Traceback (most recent call last):

File "<pyshell#123>", line 1, in <module>

score['nayun']

KeyError: 'nayun'

>>> score['nayun']=100

>>> score

{'momo': 80, 'zwi': 85, 'sana': 98, 'nayun': 100}

>>> del score['momo']

>>> score

{'zwi': 85, 'sana': 98, 'nayun': 100}

>>> score.keys()

dict\_keys(['zwi', 'sana', 'nayun'])

>>> score.values()

dict\_values([85, 98, 100])

>>> 'zwi' in score

True

>>> 'momo'in score

False

>>>

>>> score.clear()

>>> score

{}

>>>

>>>#Tuple

>>> t1=(1,2,3)

>>> t2=(1, )

>>> t3=()

>>> t4=1,2,3,4

>>> t4

(1, 2, 3, 4)

>>> type(t4)

<class 'tuple'>

>>> t5=(1,'a',"park",(1,2))

>>> t1[1:]

(2, 3)

>>> t1[0:]

(1, 2, 3)

>>> t1[0:1]

(1,)

>>> t1[0:2]

(1, 2)

>>> t1+t5

(1, 2, 3, 1, 'a', 'park', (1, 2))

>>> t4\*t4

Traceback (most recent call last):

File "<pyshell#149>", line 1, in <module>

t4\*t4

TypeError: can't multiply sequence by non-int of type 'tuple'

>>> t4\*2

(1, 2, 3, 4, 1, 2, 3, 4)

>>>

>>> #set 집합자료형

>>> #중복 허용되지 않음, 순서가 없음

>>> s1={1,2,3}

>>> type(s1)

<class 'set'>

>>> s1={1,2,2,4}

>>> s1

{1, 2, 4}

>>> l1=[1,2,2,2,2,3,3,3,3,5,5,5,5,5]

>>> s1=set(l1)

>>> s1

{1, 2, 3, 5}

>>> s2={3,5,6,7}

>>> s1+s2

Traceback (most recent call last):

File "<pyshell#162>", line 1, in <module>

s1+s2

TypeError: unsupported operand type(s) for +: 'set' and 'set'

>>> s1|s2

{1, 2, 3, 5, 6, 7}

>>> s1&s2

{3, 5}

>>> s2-s1

{6, 7}

>>> s1-s2

{1, 2}

>>> s1.add(8)

>>> s1

{1, 2, 3, 5, 8}

>>> s2.remove(6)

>>> s2

{3, 5, 7}

>>>

[느낀점]

- 스크립트 언어다 보니 한 줄씩 번역하고 실행할 수 있어 더 편리한 점이 있었습니다.

또한 가장 신기했던 것은 별다른 자료형을 사용하지 않고 바로 변수에 값을 대입할 수 있었던 점이 신기하면서도 편리하였습니다.

예제를 따라해봤듯이 만약 c나 c++에서 pi에 값을 대입하려면 int pi로 변수 선언을 했어야 하는데 별다른 자료형 없이도 pi에 곧바로 값을 넣을 수 있어서 편했습니다.

출력하는 것도 마찬가지였습니다. c에서는 printf함수를, c++에서는 cout<< 을 사용하여 조금은 번거롭게 출력이 가능했는데, 여기서는 print()를 통하여서 다른 것 없이 편하게 원하는 것들을 출력할 수 있어 편한 부분이 있었습니다.

비교연산이나 일반 연산은 이전에 배웠던 부분과 비슷하였으나, 제곱에 대한 연산기호가 존재하는 것이 새로웠습니다.

또 새로웠던 내용은 슬라이스였습니다. 문자열의 일부를 잘라내는 것이었는데 숫자를 통하여 원하는 부분부터 원하는 곳까지 혹은 단위를 정하여서 원하는 부분까지만 문자열을 자를 수 있다는 점이 가장 새로웠던 부분이었습니다.

dictionary도 배열과 비슷한듯 하였으나 새로웠던 것은 score.keys()를 사용하여서 score.values() 를 사용하여서 원하는 값들만 추출해 낼 수 있음 역시 새로웠으며 bool과 같이 참과 거짓을 판단할 때도 긴 코드보다는

쉽게 읽을 수 있는 간단한 영어문장으로써 참 거짓을 판단할 수 있어 새롭기도 하고 컴파일 언어보다 편리했던 점이 많았던 것 같습니다.

Tuple과 집합자료형인 set까지도 매우 신선하고 재미있던 내용이었습니다. 데이터 타입에 list, dictionary, tuple, set과 같이 다양하게 각기 다른 용도로 다른 모습과 특징을 가지며 구분되어 사용되고 있음에 신기하면서도 확실히 편리하게 사용할 수 있는 언어라고 생각이 들었습니다.

이제껏 컴파일언어만 접해보다가 스크립트 언어를 처음 접해보았는데 컴파일언어와는 또 다른 매력이 존재하는 것 같습니다. 일단, 초반 부분 기본적인 내용을 접해서 그런 것일 수도 있지만 컴파일언어보다 스크립트 언어가 확실히 편하구나 라고 생각이 들었고, 컴파일 언어와는 또 어떤 차이가 있을 것이며, 앞으로 배우면서도 확실히 파이썬이 쉽고 편리할 것인가 궁금증을 가지게 되었습니다.

처음 스크립트 언어 '파이썬'을 접해보았는데 신선하고 앞으로 배우는 것에 있어서 기대를 가지게 되는 계기였습니다.