10월 31일 (일)

1. Hello Git

2. git init

init 은 initialize (시작하다.) 를 뜻한다.

git init은 Git이 프로젝트 변경 사항을 추적하는데 필요한 모든 도구를 설정한다.

```
$ git init
Initialized empty Git repository in /home/ccuser/workspace/sorcerers-code/.git/
$
```

3. Git workflow

깃은 3가지 부분으로 나뉠 수 있다.

(1) working directory : 파일 생성, 편집, 삭제 및 구성 등 모든 작업을 수행 할 수 있는 곳

(2) Staging Area : 작업 디렉토리에 대한 변경 사항을 나열하는 곳

(3) Repository : 변경 사항을 프로젝트의 다른 버전으로 영구적으로 저장하는 저장소

깃에서 commit 으로 변경사항을 저장한다.



4. git status

작업 내용에 대한 변경사항을 확인할 수 있는 명령어이다.

```
$ git init
Initialized empty Git repository in /home/ccuser/workspace/sorcerers-code/.git/
$ git status
On branch master

Initial commit

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

        init_test.rb
        scene-1.txt

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

위의 Untracked files: 아래에 나오는 빨간색 파일들이 아직 Git에 저장되지 않은 항목들을 나타내고 있다.

5. git add - (스테이지 영역에 파일 추가하기)

git이 빨간색 파일 즉 secene-1.txt 추적을 시작하려면 파일이 Staging area(준비 영역)에 추가 되어야한다.

git add filename 을 통해 스테이징 에리어에 추가시켜준다.

```
$ git add scene-1.txt
$ git status
On branch master

Initial commit

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

        new file:   scene-1.txt

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

        add_test.rb
        init_test.rb

$
```

추가한 후 git status로 상태를 확인해보면 초록색 글씨로 파일이 추가 된 것을 확인할 수 있다.

6. git diff

작업 파일의 변경사항 즉, 내용을 확인할 수 있다.

```
$ git diff scene-1.txt
diff --git a/scene-1.txt b/scene-1.txt
index a02a36b..d1e46a3 100644
--- a/scene-1.txt
+++ b/scene-1.txt
@@ -1 +1,3 @@
 Harry Programmer and the Sorcerer's Code: Scene 1
+
+Dumblediff: I should've known you would be here, Professor McGonagit.
\ No newline at end of file
$
```

+표시가 생기고 추가된 내용이 초록색 글씨로 나타내어진다.

변경된 내용을 다시 스테이징 영역에 git add 를 이용하여 저장한다.

7. git commit

commit(커밋)은 Git 워크 플로우의 마지막 단계이다. 스테이지 영역의 변경 사항을 레포지토리에 영구적으로 저장하는 과정이다.

git commit -m "Complete first line of dialogue" —> 커밋 공식

(1) 따옴표 안에 적어야한다.

(2) 현재시제로 적어야한다.

(3) -m을 사용할 때는 50자 이하로 적어야한다.

m " " 은 하나의 커밋에 대해 남겨두는 메세지 인 것 같다.

```
$ git commit -m "Complete first commit"
[master (root-commit) b40a74d] Complete first commit
 1 file changed, 3 insertions(+)
 create mode 100644 scene-1.txt
$
```

8. git log

종종 git을 사용하는 경우 이전 버전의 프로젝트를 다시 참조 해야하는 경우가 있다. 커밋은 저장소에 시간순으로 저장이 되며 git log를 사용하여 확인이 가능하다.

```
$ git log
commit b40a74d33ab6e27eb693fcac5b9060cb8bb04b8e
Author: codecademy <ccuser@codecademy.com>
Date:   Mon Mar 22 09:42:52 2021 +0000

    Complete first commit
$
```

노란색 글씨는 커밋의 고유식별 코드이다. SHA 라고 하며 40자로 된 코드이다.

Author 은 커밋 작성자를 뜻한다.ㅇ

아래에 Date 는 커밋이 작성된 날짜를 뜻한다.

그 아래는 커밋 메세지를 뜻한다.

9. Generalizations (일반화)

깃은 웹 개발자를 위한 업계 표준 버전 제어 시스템이다.

깃 명령을 통해 프로젝트 변경 사항을 추적할 수 있다.

git init 는 새로운 git 저장소를 만든다.

git status는 작업 디렉토리 및 스테이징 영역의 내용을 검사한다.

git add는 작업 디렉토리의 파일을 스테이징 영역에 추가한다.

git diff 는 작업 디렉토리와 스테이징 영역의 차이점을 보여준다.

git commit 은 저장소에 스테이지 영역의 파일 변경 사항을 영구적으로 저장한다.

git log는 모든 이전의 커밋 목록을 보여준다.

Appendix) Set up with Git and Github

깃과 깃허브 사용법에 대해 알아보는 단원이다.

깃과 깃허브를 연결하여 편리하게 사용할 수 있다.

1. What are Git and Github?

깃은 코드 관리에 널리 사용되는 버전 제어 시스템이다.

깃을 사용하면 코드 초안을 저장할 수 있으므로 이전 버전을 되돌아보고 잠재적으로 복잡한 오류를 실행 취소 할 수 있다. Git으로 관리되는 프로젝트를 Git 저장소라고 한다.

깃허브는 깃 레포지토리(저장소)를 위한 인기있는 호스팅 서비스이다. 깃허브를 이용하면 로컬 Git 레포지토리를 클라우드에 저장할 수 있다.

깃허브를 사용하면 개인 파일을 백업하고 코드를 공유하고 다른 사람들과 공동 작업을 할 수 있다.

즉 간단히 말해서 깃허브는 깃 작업을 위한 도구이다. 깃 레포지토리를 호스팅하는 다른 여러 서비스가 있지만, 깃허브는 전 세계적으로 신뢰할 수 있는 무료 서비스이다.

2. 맥 유저 깃 세팅

(1) 터미널을 실행한다.

(2) 터미널에 git을 입력한다

(3) 설치가 안되어있다면 자동으로 설치 메세지가 뜨고 설치를 한다.

(4) Git 사용자 이름 및 이메일 설정에 대한 Github의 문서로 이동하여 터미널 사용에 대한 지침을 따른다.

(5) 깃허브는 작업을 안전하게 유지하기 위해 https, SSH 의 두 가지 인증 옵션을 제공한다. 이는 권한이 없는 사람이 깃 허브 저장소를 변경하지 못하도록 방지하기 위한 보안 조치이다. 이 아티클에서는 https를 사용한다. 비밀번호 캐싱에 대한 깃허브의 문서로 이동하고 안내에 따라 https를 사용할 수 있도록 컴퓨터를 구성하세요.

3. 시도해봐!

연습을 위한 로컬 깃 초기화 또는 만들기

(1) mkdir git_practice : 새로운 연습용 디렉토리 생성

(2) cd git_practice : 작업 디렉토리로 이동

(3) git init : 현재 빈 디렉토리를 새로운 깃 저장소로 전환한다.

(4) echo "Hello Git and GitHub" >> README.txt : 샘플 텍스트가 있는 새 리드미 파일을 만든다.

(5) git add README.txt : 새로운 파일을 깃 스테이징 에리어에 추가한다.

(6) git commit -m "First commit" : 새 리드미 파일로 첫번째 커밋을 만든다.

4. Github의 첫 번째 원격 저장소

마지막으로 깃허브에 저장소를 만든 다음 컴퓨터의 로컬 저장소에 연결한다. 이를 통해 작업을 지속적이고 안전하게 백업할 수 있으므로 작업을 다시 잃어 버릴 염려가 없다! 이제 로컬 깃 저장소를 깃허브에 연결해보자.

(1) 커맨드 라인 인터페이스에서 현재 작업 디렉토리가 새 Git 리포지토리인지 확인해라. 그렇지 않은 경우 거기로 이동하세요

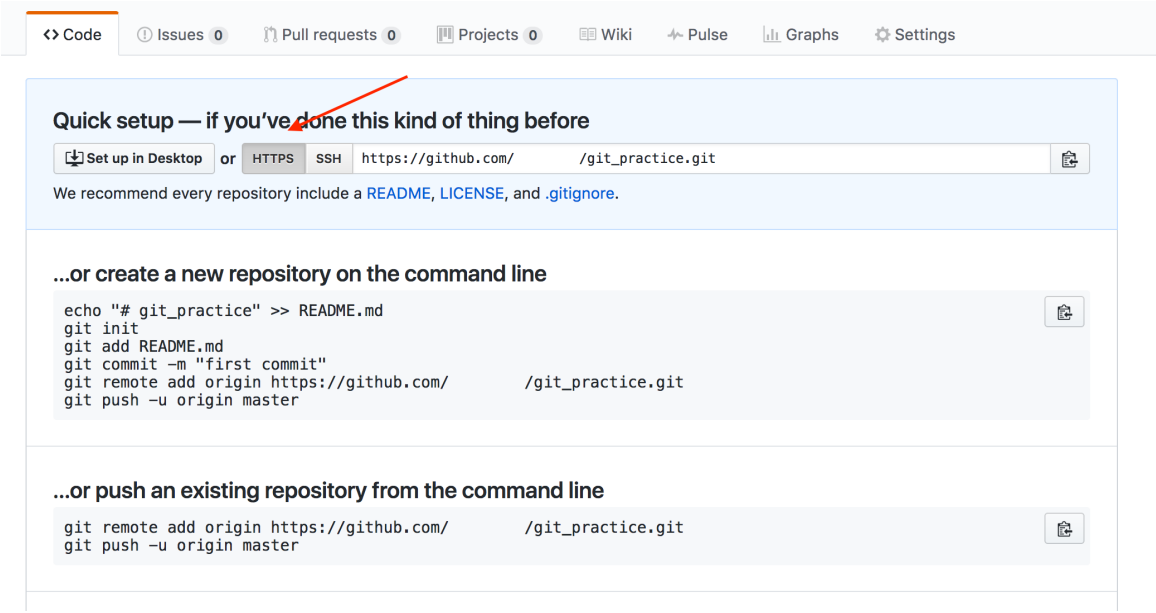
(2) 어떤 파일이나 폴더가 새롭게 편집된 상태인지 확인해라. 연결 전 수정된 파일이 없어야 한다.

—> git status 를 사용하여 확인

(3) 깃허브에서 new repository 를 클릭

(4) 레포지토리 의 이름을 깃 디렉토리와 같게 하는 것이 편리하다.

(5) 레포지토리가 생성되면 깃허브에 리포지토리 페이지가 표시된다. 페이지 상단에서 "HTTPS" 가 선택되어 있는지 확인한다.



(6) 저장소가 비어 있으므로 기존 작업에 연결해보자. 깃허브 페이지의 "...or push an existing repository from the command line" 라는 제목 아래에 깃 명령을 복하사여 커맨드 라인 인터페이스에 붙여 넣는다. 이러한 명령을 실행하면 원격 저장소가 추가된 다음 로컬 저장소가 원격 저장소로 푸쉬된다.

사용자 이름과 암호를 묻는 메세지가 표시되면 깃허브 사용자 이름과 암호를 입력하고 엔터키를 누른다. 입력중인 문자가 보이지 않더라도 놀라지 말자. 보안 조치를 위해 의도적으로 숨겨져 있기 때문이다.

(7) 터미널에 푸시가 완료됐다고 하면 깃허브에서 페이지를 새로고침한다. 이제 이전에 작성한 리드미 파일을 볼 수 있다.

깃허브는 저장소에 있는 리드미 파일의 내용을 자동으로 표시한다. 리드미파일은 프로젝트에 대한 설명을 작성하기에 완벽한 장소이다.