**버전 관리 시스템**

문서나 코드의 변경사항을 저장하여 과거의 상태를 열람, 복원할 수 있도록 함

협업시에 협업자들의 변경사항을 자동으로 붙여주고, 충돌 방지

충돌이 일어난 부분 고지하고 직접 고치도록 권유

**Git**

분산형 버전관리 시스템

**Git ssh 설정**

**Root folder**

웹 상의 git과 연결될 local folder

git init

git remote add origin *repository address*

**Commit**

타임캡슐… 소스코드에 변경된 파일들이 들어갈 것

커밋을 네번 했으면 전으로 돌아갈 수 있고 Ctrl+Z 느낌

**Branch**

마스터로 시작했는데 *서브* 브랜치로 갈 수 있음

모두 마스터에서 작업한다고 하면 합치려고 할 때 시간이 맞지 않음. 충돌 가능성

애초에 브랜치를 파주게 되면

서로 이미 작업된 상태에서 브랜치를 만들고 그때부터 각자 만듦

나중에는 이걸 하나로 합침

Git merge! 마스터 브랜치에!

* 이 기능 때문에 깃을 쓰는 것

**Remote add origin**

클론을 했기 때문에 안썼다

이 주소를 가리킨다는 명령어

**Push**

git push -u origin master

이 주소에다가 마스터라는 브랜치를 동기화하겠다

**Function이라는 브랜치를 만들 것**

Git branch => 현재 브랜치들 보여줌

Git branch *Function => 만듦*

브랜치를 만들었으면 옮겨타는 게 필요!

* **git checkout *Function***

마스터 브랜치는 변경이 안되어있어서 깃헙에 반영된 건 아님!

바로 반영되는 건 아님~ 실수가 있을 수도 있고…

**Pull request**

이 프로젝트를 주도하는 사람들에게 마스터 브랜치에 반영해달라고 요청하는 것

보고서라고 생각하면 된다

Merge는 누구나 누를 수 있지만 …!

Merge하면 그 브랜치는 필요가 없어지니까 바로 삭제할 수 있음

**Conflict (Merge)가 안되는 경우**

두명이 작업한다고 가정

Git reset –hard origin/master

오리진에 있는 마스터에 있는 ㅐㄴ용을 이 브랜치로 완전 탈바꿈

Hard, soft, default

Cd Desktop

Git clone ssh 주소 Example

* Example 폴더 만들어짐. 깃과 연동되는 !

Cd Example

Git log

Git status

Git add .

Git commit -m “created test.py”

Git status

Git log

Git push origin master

Git branch

Git branch function

Git checkout function

* Function 브랜치로 전환합니다

Git status

Git add .

Git commit -m “function was created”

Git push origin function

Git status

Git checkout master

Git fetch

Git fetch --all

* 로컬에 적용시킴

Git log

Git reset --hard origin master

* 깃헙에 있는 걸 로컬에 그대로 복사! 토씨하나 안틀리고
* Hard, soft, (default) 버전있음
* Staging에 대한 개념 이해 필요

Git branch A

Git checkout A

Git status

Git add .

Git commit -m “msg is A”

Git push origin A

Git log

Git checkout master

Git branch B

Git checkout B

Git log

Git status

Git add .

Git commit -m “msg is B”

Git push origin B

Git checkout master

Git fetch --all

Git reset --hard origin/master

Git log