GitHub 포트폴리오

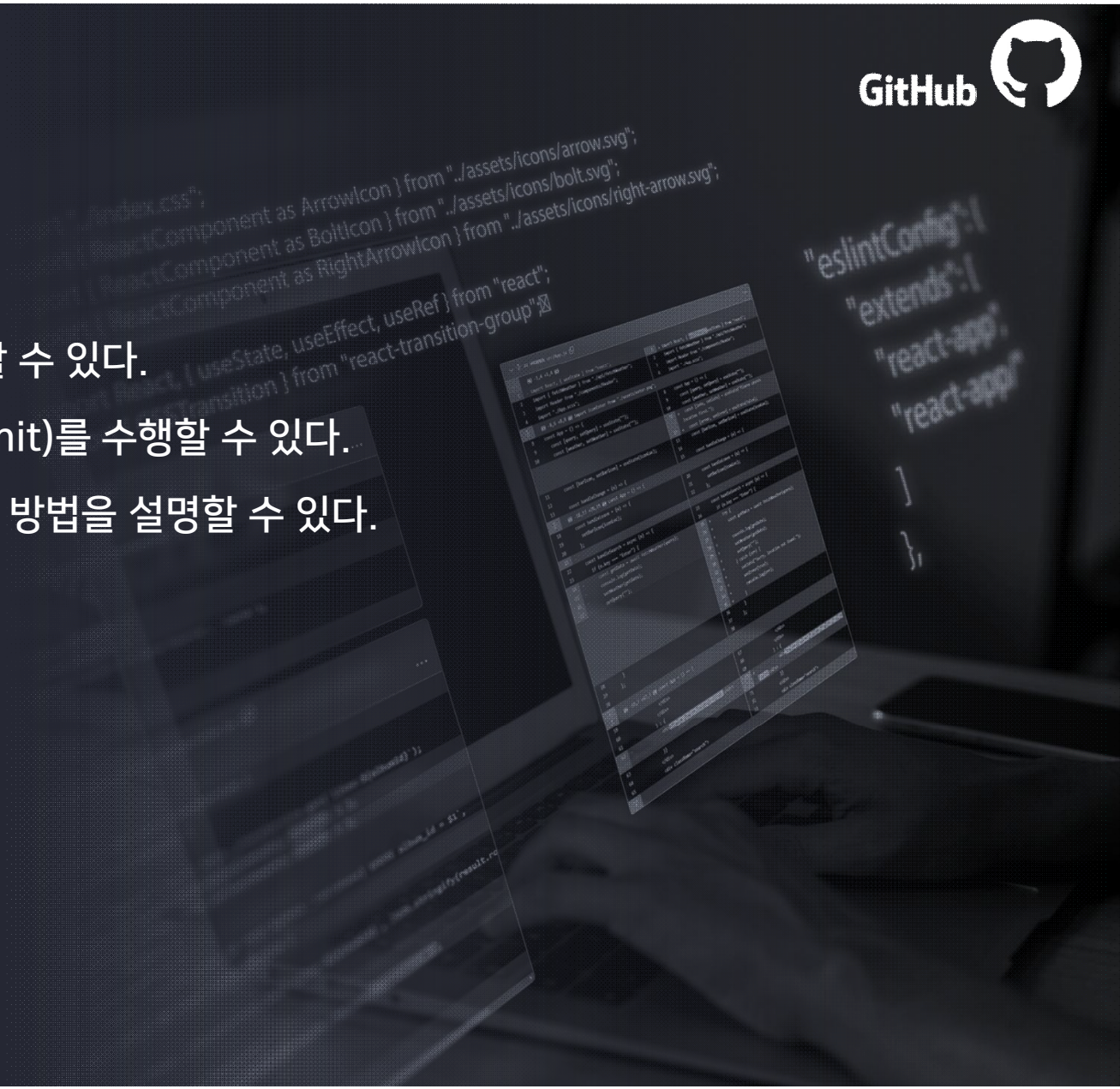
2주차. Git 기본 사용법 I

서울사이버대학교

지난 주 복습

버전관리 이해	버전	소프트웨어나 문서 등의 특정 상태를 구별하는 고유한 식별자
	버전관리 시스템	파일 변화를 시간에 따라 기록 및 특정 시점의 버전을 다시 꺼내올 수 있는 시스템
	장점	• 파일 복원 또는 복구 • 파일 변경 전과 후의 비교 • 문제 추적
Git과 Github	Git	분산 버전 관리 시스템
	Github	Git 버전 관리 시스템을 사용하여 소스 코드 관리를 돕는 웹 호스팅 서비스의 일종
Github 계정 생성과 기본 설정	Git	계정 불필요 & 다운로드 후 설치
	Github	GitHub 서비스에 가입하여 계정 생성
	기본 설정	SSH 키 생성 및 등록, git config 환경설정

학습개요



학습목표

01. Git 설치 및 설정 방법에 대해 설명할 수 있다.

02. Git의 기본 명령어(init, add, commit)를 수행할 수 있다.

03. 로컬 저장소를 효과적으로 관리하는 방법을 설명할 수 있다.

학습내용

- Git 설치 및 설정 방법
- Git의 기본명령어
- 로컬 저장소 관리

학습에 앞서...

GitHub

Git 설치 및 설정 방법

Git의 기본 명령어

로컬 저장소 관리

- 분산 버전 관리 시스템으로의 Git은 많은 기능에도 불구하고 설치 등 접근성이 매우 좋다고 알려져 있음
- 이번 시간은 Git의 기본 사용법을 익히는 데 중점을 둠
- Git은 버전 관리 시스템의 표준으로, 프로젝트의 변경 이력을 관리하고 협업을 원활하게 함
- Git의 기본 사용법을 숙지하는 것은 향후 프로젝트 관리와 팀 협업 시 필수적이며, 실무에서 많이 요구되는 기술임

Git 설치 및 설정 방법

UNIT

Git 설치

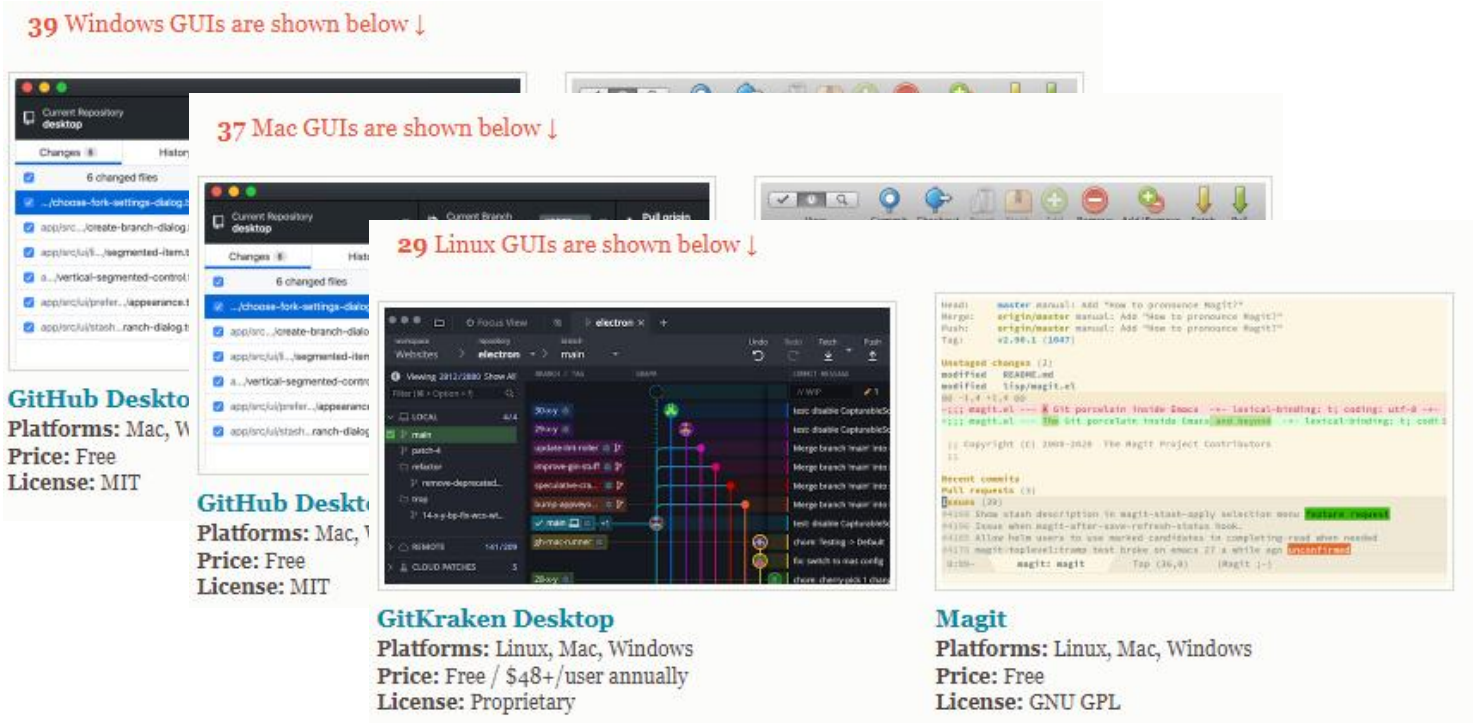
설치 방법 (CLI, 강의내용)

- Windows :
  Git 공식 웹사이트(https://git-scm.com/downloads) 최신 윈도우 버전 다운로드 & 설치
- macOS :
  보통 기본적으로 설치되어 있음
  (단, 최신버전이 아닐 수 있으니 brew install git 으로 업데이트)
- Linux (패키지 관리자)
  - Ubuntu/Debian: sudo apt-get install git
  - Fedora: sudo yum install git
  - Arch Linux: sudo pacman -S git

Git 설치

설치 방법 (GUI)

- Git 공식 웹사이트(https://git-scm.com/download/gui/windows) 에서 OS별로 다양하게 선택 가능



Git 설치

설치 방법 (GUI)

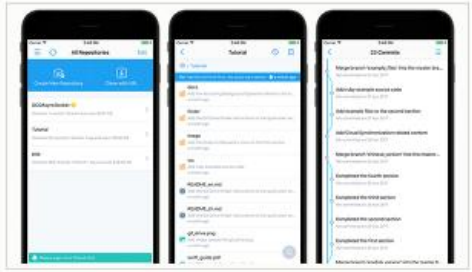
- Git 공식 웹사이트(https://git-scm.com/download/gui/windows) 에서 OS별로 다양하게 선택 가능

1 Android GUIs are shown below ↓

Pocket Git
Platforms: Android
Price: €2.49/$1.99
License: Proprietary

4 iOS GUIs are shown below ↓

Working Copy
Platforms: iOS
Price: Free / $19.99
License: Proprietary



GitDrive
Platforms: iOS
Price: Free / $6.99
License: Proprietary

Git 설치

GitHub

접근성과 용어 장벽

오소리님, 프로젝트 Fork로 로컬에 clone 하신 다음에 develop 브런치 아래에 오소리님 브런치 생성해서 한 번 커밋 푸시해보시겠어요?

Fork요...?
(Fork로 어떻게 clone하고 브런치까지 만들라는 거지..?)

네. Fork 이용하시면 돼요!

??? Fork 한 후에 clone 하고 브런치 생성하라는거죠??

아뇨, Fork라는 툴이 있어요!

!!

Git 설정 및 기본 검증

Git 설치 확인 (= 버전 확인)

- 버전을 확인하는 것으로 설치 여부 점검

CODE

```
git --version
```

⇢ git version 2.39.2

⇢ "command not found" 또는 유사한 에러 메시지

Git 설정 및 기본 검증

Git 설정

- 사용자 이름과 전자우편 설정 :
  Git 커밋 기록에 저장되어 작업한 사람을 식별하는 데 사용

CODE

```
git config --global user.name "Your Name“
git config --global user.email "your@email.com"
```

Git 설정 확인

CODE

```
git config --list     # 전체 확인
git config [--global] user.name    # [전역정보에서] 개별정보 확인
git config [--global] user.email    # [전역정보에서] 개별정보 확인
```

Git 설정 및 기본 검증

Git 설정 파일 위치

- Windows : C:₩Users₩user₩Username.gitconfig
- macOS/Linux : ~/.gitconfig

> 위 경로에서 gitconfig 파일을 직접 열어보면 설정된 값들을 확인할 수 있음

> Git 설정은 기본적으로 한번만 설정
(단, 프로젝트나 상황에 따라 다른 이름과 이메일 주소를 사용해야 하면 매번 변경)

Git 의 기본명령어

UNIT

Git 저장소 초기화

GitHub

저장소 신규 생성 : git init [경로]

- git init : 현재 디렉토리에 Git 저장소가 생성
- 생성결과 메시지 :
  Initialized empty Git repository in /path/to/directory/.git/
- git init /path/to/new/repository
- 생성결과 메시지 :
  Initialized empty Git repository in /path/to/new/repository/.git/

> git init 는 한번만 실행 가능
(기존에 있으면 '.git 디렉토가 있다고 메시지 출력')

> git add, git commit, git push 등을 사용하여 본격적인 버전관리 가능

> git 저장소 삭제는 ".git" 폴더 삭제
(단, 프로젝트 소스 코드 파일은 존재함)

Git 저장소 초기화

GitHub

저장소로 전환

- 기존에 프로젝트 소스가 있는 폴더를 git 저장소로 전환

1. cd /path/to/existing-project # 기존 프로젝트 디렉토리로 이동
2. git init # 저장소 생성 초기화
3. git add . # 프로젝트 디렉토리 내의 모든 파일을 Git 저장소에 추가
   (특정 파일 추가 git add file1.ext file2.ext)
4. git commit -m "Initial commit"
   # 추가된 파일들을 커밋. -m 옵션 뒤에 커밋 메시지를 작성

⇢ 기존 프로젝트가 Git 저장소로 전환

⇢ git add, git commit 등을 사용하여 본격적인 버전관리 가능

> .gitignore" 파일을 생성하여 특정 파일이나 디렉토리를 Git 저장소에서 제외 가능

Git 저장소 초기화

원격 저장소로 전송

- 기존에 프로젝트 소스가 있는 폴더를 git 저장소로 전환

1. cd /path/to/existing-project # 기존 프로젝트 디렉토리로 이동
2. git init # 저장소 생성 초기화
3. git add . # 프로젝트 디렉토리 내의 모든 파일을 Git 저장소에 추가
   (특정 파일 추가 git add file1.ext file2.ext)
4. git commit -m "Initial commit"
   # 추가된 파일들을 커밋. -m 옵션 뒤에 커밋 메시지를 작성
5. git remote add origin https://github.com/username/repo.git
   # GitHub나 GitLab 등의 원격 저장소에 연결
6. git push -u origin master
   # 원격 저장소에 최초로 push. -u 옵션은 git push만 입력해도 원격 저장소 push
7. git pull <원격 저장소 별칭> <브랜치이름>
   # 원격 저장소의 변경사항을 가져와 로컬의 브랜치와 병합

파일 상태 추적

GitHub

git add, git status

- “git add <파일명>”, “git add .” : 파일을 스테이징 영역에 추가
  - 새로운 파일(untracked file)의 상태를 staging are(준비영역)에 추가함으로 추적 개시
- “git status” : 파일 상태 확인
  - 추적되지 않은 파일(untracked file) : Git 저장소에 추가되지 않은 새 파일
  - 추적된 파일(tracked file)
    * Unmodified : 최신 커밋과 동일한 상태
    * Modified: 수정되었지만 아직 staging area에 추가되지 않은 상태
    * Staged: 수정된 파일이 staging area에 추가된 상태로, 다음 커밋에 포함될 예정

파일 상태 추적

GitHub

"git status" 결과 해석과 작업(action)

```
$ 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:   file1.txt

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

    file2.txt
```

- file1.txt
  - (추적된 vs 추적되지 않은) 파일이며 수정되었기 때문에 (git add vs git status)로 staged 상태로 만든 후 (git commit vs git push) 하여 커밋
  - 커밋한 파일은 (git commit vs git push)하여 저장소에 반영

파일 상태 추적

"git status" 결과 해석과 작업(action)

```
$ 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:   file1.txt

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

    file2.txt
```

- file2.txt
  - (추적된 vs 추적되지 않은) 파일이며 수정되었기 때문에 (git add vs git status)로 staging area에 추가하여 추적하며, (git commit vs git push) 하여 커밋
  - 커밋한 파일은 (git commit vs git push)하여 저장소에 반영

변경 사항 커밋

GitHub

commit 의 정의와 중요성

commit	파일의 변경 사항을 기록하고 버전 관리하는 역할

- 파일의 변경 사항을 스냅샷 형태로 기록
- 변경된 파일, 변경 내용, 작성자, 날짜, 커밋 메시지 등의 메타데이터 포함
- 변경 이력 추적 정보
- 특정 시점을 버전으로 저장
- 필요한 경우 이전 버전으로 되돌아갈 수 있음
- 특정 버전 간의 차이를 비교하고 병합
- 논리적인 작업 단위로 변경 사항을 분리
- 작은 커밋 단위로 나누면 코드 리뷰와 문제 추적이 용이

변경 사항 커밋

GitHub

git commit -m "커밋 메시지“ : 변경사항을 커밋하여 이력을 저장

- 커밋 메시지
  - 간결하고 명확한 문구 사용
  - 딱! 보면 기억할 내용
  - 타 시스템과 연결되는 값
  - 이슈트래커ID 작성하면 좋음 (“Fix issue #42”)

변경 사항 커밋

GitHub

git log : 변경사항 이력 확인

- git log : 전체 커밋 로그가 출력
- git log --stat : 변경된 파일의 통계 정보 출력
- git log --oneline : 커밋 해시값과 제목만 출력
- git log --graph --oneline :
  브랜치와 병합 이력을 약간 그래프스럽게 출력
- git log -- <파일/디렉토리 경로> :
  특정 파일이나 디렉토리의 변경 내역만 확인
- git log --author="작성자명" : 특정 작성자의 로그만 확인
- git log --since="2020-01-01" --until="2020-06-30" :
  특정 기간 동안의 로그 확인

변경 사항 커밋

GitHub

git commit --amend : 가장 최근의 커밋 정보 수정

git revert

- 특정 커밋을 취소하고, 그 변경 사항의 반대 작업을 새로운 커밋으로 생성
- 커밋 히스토리를 지우지 않고 안전하게 되돌릴 수 있음
- 원격 저장소에 push된 커밋을 취소할 때 유용

Ex > git revert HEAD (가장 최근 커밋 취소),
git revert [커밋 해시] (특정 커밋 취소)

변경 사항 커밋

GitHub

git reset

- 현재 브랜치를 특정 커밋 상태로 완전히 되돌리기
- 커밋 이력이 변경되며, 지정한 커밋 이후의 모든 커밋이 제거
- 작업 파일과 스테이징 영역의 변경 사항도 함께 제거
- 로컬 저장소에서만 사용하는 것을 권장

Ex > git reset [커밋 해시] (지정 커밋 상태로 되돌리기),
git reset --hard HEAD (최신 커밋 상태로 되돌리기)

로컬 저장소 관리

UNIT

Git 디렉토리 구조 이해

Git 디렉토리와 Working 디렉토리

Git 디렉토리	프로젝트의 모든 파일과 디렉토리, 그리고 Git에 의해 관리되는 메타데이터가 저장

- 숨김 폴더인 .git 디렉토리에 저장소의 모든 데이터가 포함
- 프로젝트의 모든 버전 기록과 변경 사항을 추적하고 관리

Git 디렉토리 구조 이해

Git 디렉토리와 Working 디렉토리

Working 디렉토리	실제로 파일을 편집하고 작업하는 프로젝트 디렉토리

- Git 디렉토리와 별개로 존재
- 개발자가 직접 파일을 수정하고 변경 사항을 Git에 추가하는 장소
- 일반적으로 Git 저장소에 아직 커밋되지 않은 상태의 파일이 존재
- (git add vs git commit) 명령으로 staging area에 추가하고,
- (git add vs git commit)으로 Git 디렉토리에 변경 기록

Git 디렉토리 구조 이해

GitHub

작업과 디렉토리

1. 워킹 디렉토리에서 파일을 수정하거나 새로 생성
2. git add 명령으로 변경 사항을 스테이징 영역에 추가
3. git commit 명령으로 스테이징 영역의 변경 사항을 Git 저장소에 영구적으로 기록

> 워킹 디렉토리는 실제 작업 공간

> 스테이징 영역은 Git 저장소에 기록할 변경 사항을 준비하는 중간 영역

> Git 디렉토리 내부에 있는 Git 저장소는 프로젝트의 버전 기록을 안전하게 보관하는 영역

파일 상태 변화 관리

파일의 상태

상태	설명
Untracked 상태	Git 저장소에 아직 추적되지 않은 새 파일의 상태 (git add)
Modified 상태	이미 추적 중인 파일을 수정한 상태 (git add)
Staged 상태	Staging Area에 추가된 파일의 상태 (git commit)

> git diff : 워킹 디렉토리와 Staging Area의 차이점 확인

> git diff --staged : Staging Area와 최신 커밋 간의 차이점 확인

> git rm --cached <파일명> : Staging Area에서 파일 제거