1. Git

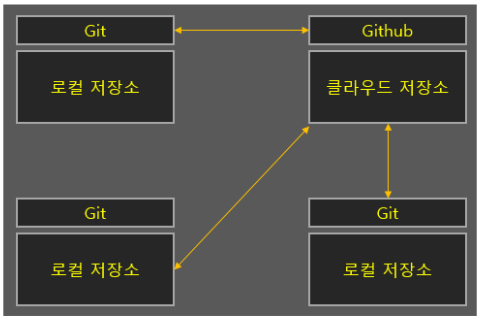
Git 은 기본적으로 소프트웨어 프로그램이자 시스템

> Git은 실제로 여러 개의 작은 실행 파일이나 스크립트로 구성된 도구 모음입니다. 예를 들어, Windows 환경에서는 Git을 설치하면 **git.exe**가 존재하며, 이 파일이 Git 명령어들을 호출하는 프론트엔드 역할을 합니다.
>
> 반면에 Unix/Linux 같은 시스템에서는 Git 명령어들이 별도의 바이너리 파일들로 제공되어 터미널에서 직접 사용할 수 있습니다.
> 즉, Git은 하나의 단일 .exe 파일이 아니라 여러 실행 파일과 스크립트들이 모여 전체 기능을 제공하는 구조로 구현되어 있습니다.

참고) https://yganalyst.github.io/etc/git_github_summary/

2. Git 과 GitHub, Cloud 의 관계

Git 은 분산형 버전 관리 시스템으로 파일, 특히 소스 코드의 변경 이력을 시간 순서대로 로컬 저장소에 기록하고, commit & push 로 필요시 원격 저장소에 업로드한다. 이 때, 업로드되는 원격 저장소를 호스팅하고 협업을 가능하게 해주는 플랫폼이 GitHub 이며, 여기서 원격 저장소를 운영하는데 Cloud 기술이 적용되는 것이다.(Git = 프로그램, GitHub = 플랫폼, Cloud = 인프라)



출처: https://www.elancer.co.kr/blog/detail/179

3. Git 과 Repo 의 물리적 위치

Git과 Repository의 물리적 위치는 사용 방식에 따라 다르며, 로컬과 원격으로 나눌 수 있습니다.

1. Git의 물리적 위치

- **Git 자체**는 소프트웨어로, 사용자의 컴퓨터(로컬 머신)에 설치됩니다. Git 클라이언트는 로컬 저장소와 상호작용하며, 커밋, 푸시, 풀 등 다양한 명령어를 통해 버전 관리 기능을 제공합니다.

2. Repository의 물리적 위치

Repository(저장소)는 Git에서 버전 관리가 이루어지는 디렉토리이며, 두 가지 형태로 나눌 수 있습니다:

1. **로컬 저장소(Local Repository)**
   - 로컬 저장소는 사용자의 개인 컴퓨터에 위치합니다. `git init` 명령어로 폴더를 Git 저장소로 초기화하면, 해당 폴더 내 `.git` 디렉토리가 생성됩니다. 이 `.git` 디렉토리 안에는 프로젝트의 모든 버전 정보(커밋, 브랜치 등)가 저장됩니다.
   - 로컬 저장소의 물리적 위치는 사용자가 Git 저장소를 설정한 디렉토리입니다. 예를 들어, `C:/projects/my-repo/` 디렉토리 안에 `.git` 폴더가 있는 경우, 해당 디렉토리가 로컬 저장소입니다.
2. **원격 저장소(Remote Repository)**
   - 원격 저장소는 GitHub, GitLab, Bitbucket과 같은 호스팅 서비스 또는 개인 서버에 위치합니다. 이는 로컬 저장소와 다르게 네트워크를 통해 접근할 수 있는 위치에 있으며, 협업을 위해 여러 사용자가 공유할 수 있습니다.
   - 물리적으로는 클라우드 서버 또는 데이터 센터의 물리적 서버에 저장됩니다. 사용자는 `git push`, `git pull` 등의 명령어를 통해 원격 저장소에 접근하고, 데이터를 송수신할 수 있습니다.
   - 원격 저장소의 주소는 일반적으로 `https://`, `git@` 등의 형식을 통해 접근합니다. 예를 들어, `https://github.com/user/repo.git` 과 같은 URL로 저장소에 접근합니다.

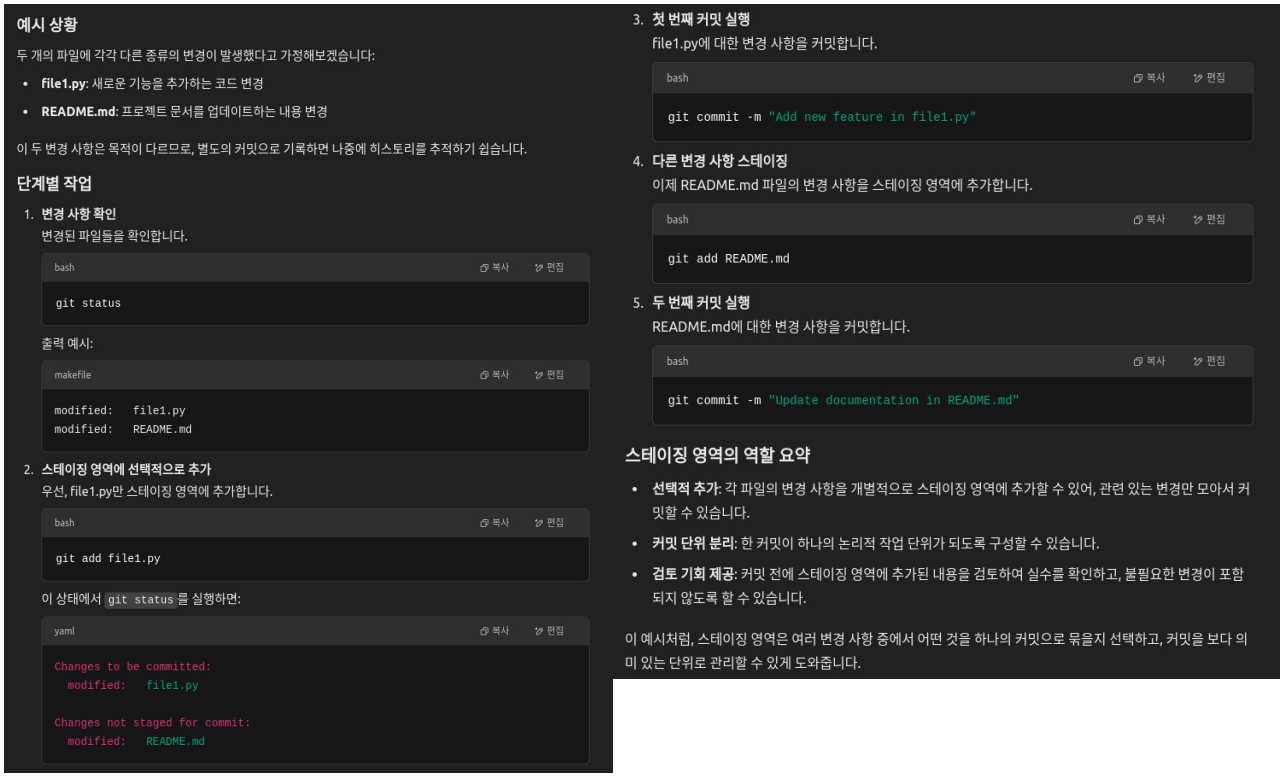
요약

- **Git** 자체는 사용자의 로컬 컴퓨터에 설치되는 소프트웨어입니다.
- **로컬 저장소**는 사용자의 컴퓨터 디렉토리 내에 위치하며, `.git` 폴더가 해당 저장소의 물리적 위치입니다.
- **원격 저장소**는 클라우드 호스팅 서비스나 서버에 위치하며, 네트워크를 통해 접근할 수 있는 저장소입니다.

4. add vs commit

add 는 임시 저장소(stage = .git/index 에 존재하는 물리적 저장위치)에 변경이력을 저장 / commit 은 로컬저장소 .git 에 최종적으로 변경이력을 저장

stage 와 stagging 이 필요한 이유(예시)



예시 상황

두 개의 파일에 각각 다른 종류의 변경이 발생했다고 가정해보겠습니다:

- **file1.py**: 새로운 기능을 추가하는 코드 변경
- **README.md**: 프로젝트 문서를 업데이트하는 내용 변경

이 두 변경 사항은 목적이 다르므로, 별도의 커밋으로 기록하면 나중에 히스토리를 추적하기 쉽습니다.

단계별 작업

1. **변경 사항 확인**
   변경된 파일들을 확인합니다.

```
bash
git status
```

   출력 예시:

```
makefile
modified:   file1.py
modified:   README.md
```

2. **스테이징 영역에 선택적으로 추가**
   우선, file1.py만 스테이징 영역에 추가합니다.

```
bash
git add file1.py
```

   이 상태에서 `git status`를 실행하면:

```
yaml
Changes to be committed:
  modified:   file1.py

Changes not staged for commit:
  modified:   README.md
```

3. **첫 번째 커밋 실행**
   file1.py에 대한 변경 사항을 커밋합니다.

```
bash
git commit -m "Add new feature in file1.py"
```

4. **다른 변경 사항 스테이징**
   이제 README.md 파일의 변경 사항을 스테이징 영역에 추가합니다.

```
bash
git add README.md
```

5. **두 번째 커밋 실행**
   README.md에 대한 변경 사항을 커밋합니다.

```
bash
git commit -m "Update documentation in README.md"
```

스테이징 영역의 역할 요약

- **선택적 추가**: 각 파일의 변경 사항을 개별적으로 스테이징 영역에 추가할 수 있어, 관련 있는 변경만 모아서 커밋할 수 있습니다.
- **커밋 단위 분리**: 한 커밋이 하나의 논리적 작업 단위가 되도록 구성할 수 있습니다.
- **검토 기회 제공**: 커밋 전에 스테이징 영역에 추가된 내용을 검토하여 실수를 확인하고, 불필요한 변경이 포함되지 않도록 할 수 있습니다.

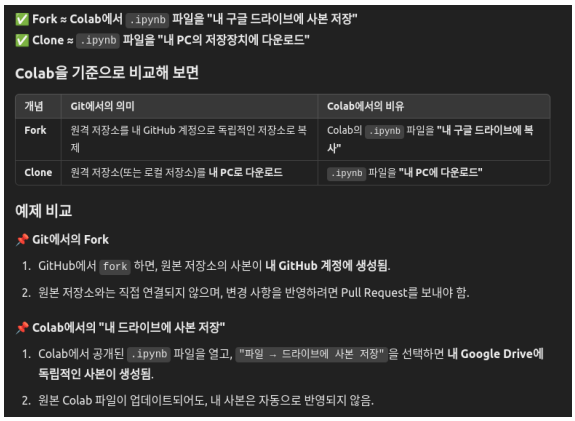
이 예시처럼, 스테이징 영역은 여러 변경 사항 중에서 어떤 것을 하나의 커밋으로 묶을지 선택하고, 커밋을 보다 의미 있는 단위로 관리할 수 있게 도와줍니다.

5. repo?

repository 는 local, remote 둘 다 있지만, 보통 remote 를 가리킴
ex) "repo 찾아봐"(프로젝트의 github 원격 repo 링크를 들어가보라는 뜻)

6. fork vs clone

fork 는 내 계정의 remote repo 에 타 사용자의 repo 를 복사해오는 것(GitHub 서버 내에서 복사)이고, clone 은 내 pc 의 local repo 에 복사해오는 것(pc 저장장치로 다운로드)



✅ **Fork ≈ Colab에서 `.ipynb` 파일을 "내 구글 드라이브에 사본 저장"**
✅ **Clone ≈ `.ipynb` 파일을 "내 PC의 저장장치에 다운로드"**

Colab을 기준으로 비교해 보면

개념	Git에서의 의미	Colab에서의 비유
Fork	원격 저장소를 내 GitHub 계정으로 독립적인 저장소로 복제	Colab의 `.ipynb` 파일을 **"내 구글 드라이브에 복사"**
Clone	원격 저장소(또는 로컬 저장소)를 **내 PC로 다운로드**	`.ipynb` 파일을 **"내 PC에 다운로드"**

예제 비교

📌 **Git에서의 Fork**

1. GitHub에서 `fork` 하면, 원본 저장소의 사본이 **내 GitHub 계정에 생성됨**.
2. 원본 저장소와는 직접 연결되지 않으며, 변경 사항을 반영하려면 Pull Request를 보내야 함.

📌 **Colab에서의 "내 드라이브에 사본 저장"**

1. Colab에서 공개된 `.ipynb` 파일을 열고, `"파일 → 드라이브에 사본 저장"`을 선택하면 **내 Google Drive에 독립적인 사본이 생성됨**.
2. 원본 Colab 파일이 업데이트되어도, 내 사본은 자동으로 반영되지 않음.

7. commit message(commit 규칙)

좋은 협업환경을 구성하기 위해서 필요한 전략 중 하나로 commit, pull-request 관리 전략

참고) https://github.com/SpaceStationLab/git-commit

8. Git-flow

좋은 협업환경을 구성하기 위해서 필요한 전략 중 하나로 branch, version, release 관리 전략

참고) https://techblog.woowahan.com/2553/