

GIT

1. 버전관리시스템의 개요

A. 정의

- 소프트웨어 개발에서 소스 코드와 파일의 변경 이력을 관리하는 도구
- VCS는 코드의 변경 내용을 추적하고 변경 이력을 관리하여 효율적인 개발과 협업을 가능하게 함
- 여러 명이 개발자가 동시에 작업하고 변경사항을 통합하며, 이전 버전으로 롤백이나 비교 등의 작업도 지원함

B. 분류

i. 중앙 집중식 버전 관리 시스템

- 중앙 서버에 저장소가 위치하고, 개발자들은 중앙 서버에서 파일을 받아서 작업
- 변경 이력과 파일은 중앙 서버에 집중되어 저장
- 예 : Subversion (SVN), CVS

ii. 분산 버전 관리 시스템

- 로컬 컴퓨터에 저장소를 복제하여 작업하고, 변경 이력 및 파일은 로컬 저장소에 기록
- 로컬 저장소에 기록된 변경 내용은 원격 서버에 공유될 수 있음
- 예 : Git

C. 발전 순서

i. 로컬 버전 관리 시스템 (Local VCS)

- 초기에 개발자들이 개별적으로 소스 코드 변경 이력을 관리하기 위해서 사용
- 변경 이력을 관리하기 위해서 패치(patch)를 생성
- RCS (Revision Control System)
 - ✓ 1982년 개발, 발표된 버전 관리 시스템
 - ✓ GNU 프로젝트로 개발

ii. 중앙 집중식 버전 관리 시스템

- CVS (Current Version System)
 - ✓ 1990년 발표 (1986년에 흔적이 있음), GPL 라이선스
 - ✓ SunOS의 커널 소스 코드를 관리하는데 사용되었음
- Subversion, SVN
 - ✓ 2000년 10월 20일 발표
 - ✓ 아파치 프로젝트, 아파치 라이선스
 - ✓ Current Version – 1.14.2 (2022. 4. 12)
 - ✓ CVS의 계승자

파일 단위 → 파일 및 디렉토리, 프로젝트 단위

중앙 저장소 → 로컬 및 중앙 저장소 (GIT의 로컬 저장소와는 개념이 다름)

- iii. 분산 버전 관리 시스템
 - Git
2. GIT 소개
- A. 간단한 역사
- 2002년 이후 리눅스는 비트키퍼(BitKeeper)를 소스 관리 시스템으로 사용
 - 비트키퍼의 자유 이용 라이센스 철회로 인해서 새로운 리눅스 소스 관리 시스템이 필요해졌음
 - 리누스 토발즈의 요구사항
- Take the Concurrent Versions System (CVS) as an example of what not to do; if in doubt, make the exact opposite decision. (CVS는 안됨. 비슷한 것도 안됨)
 - Support a distributed, BitKeeper-like workflow. (BitKeeper에서 사용하는 것과 유사한 분산 워크플로우를 지원해야 한다.)
 - Include very strong safeguards against corruption, either accidental or malicious. (우연이든, 고의든 깨짐은 발생하면 안된다.)
- 원하는 것을 찾지 못해서, 리누스 토발즈가 직접 개발을 시작 (2005. 4. 3) 했고, 2005. 4. 7에 최초의 릴리즈가 발표
 - 최초의 머지는 4. 18, 그리고 git으로 관리된 최초의 리눅스 커널은 6. 16 (2.6.12-rc2)
- The name "git" was given by Linus Torvalds when he wrote the very first version. He described the tool as "the stupid content tracker" and the name as (depending on your mood):
- random three-letter combination that is pronounceable, and not actually used by any common UNIX command. The fact that it is a mispronunciation of "get" may or may not be relevant.
 - stupid. contemptible and despicable. simple. Take your pick from the dictionary of slang.
 - "global information tracker": you're in a good mood, and it actually works for you. Angels sing, and a light suddenly fills the room.
 - "goddamn idiotic truckload of sh*t": when it breaks
 - 2005. 12. 21 1.0 버전 출시
 - ✓ 1.x에서는 Git의 기본 기능과 핵심 개념이 정리되고 반영
 - 2014. 5. 28 2.0 버전 출시
 - ✓ 2.0 : 대규모 프로젝트에서 성능을 향상시키기 위한 여러 개선 사항 반영
 - ✓ 2.6 : Git LFS 지원
 - ✓ 2.17 : Git Protocol 2.0 도입
 - 최신 버전 – 2.41 (2023. 6. 1)
- B. 개념
- 분산 버전 관리 시스템 – 개발자들이 소스 코드를 효과적으로 관리하고 협업하는데 사용되는 도구
 - 코드 변경의 스냅샷을 저장하는 방식으로 동작. 이를 통해 언제든지 이전 상태로 돌아갈 수 있고, 여러 개발자들이 동시에 작업하고 변경 내용을 공유할 수 있음
 - 로컬에 Git을 설치하고 사용할 수 있으며, Git 저장소는 프로젝트의 폴더나 디렉토리로 간주할 수 있음

- ‘커밋’ 개념 – 코드 변경의 스냅샷을 칭하며, 동시에 변경 사항을 저장소에 기록하는 작업. 각각의 커밋은 고유한 식별자를 가지고 있으며, 이를 통해 변경 이력을 추적
- 코드 변경을 추적하고 관리할 수 있음 – 파일이 변경되면 이를 ‘스테이징’으로 관리하고, ‘스테이징’을 커밋해서 저장소에 기록.
- 브랜치를 통한 병렬 작업 지원 – 브랜치는 독립적인 작업 흐름을 의미하고, 여러 개발자가 동시에 작업하고 변경 사항을 병합할 수 있음
- 원격 저장소와의 연동을 지원 (Fetch, Merge)

C. GIT vs. Subversion

	Git	SVN
분산 방식	분산	중앙집중식
저장소 위치	서버, 클라이언트	서버
로컬 저장소의 역할	하나의 완전한 저장소	중앙 서버와의 동기화를 위한 작업 환경
브랜치와 병합	브랜치 및 병합이 간단하고 유연	비교적 복잡한 브랜치 및 병합 프로세스
네트워크 의존성	로컬 작업에 인터넷 연결 필요 없음	인터넷 연결 필요
성능	빠른 속도 및 효율적인 처리	프로젝트가 커지면 느려질 수 있음
무결성	로컬 저장소에서 변경 내용의 무결성 보장	변경 내용이 중앙 서버에 의해 제어됨
작업 동기화	로컬 저장소에서 작업을 자유롭게 수행	중앙 서버에서 작업을 동기화
릴리즈 버전 관리	태그와 브랜치를 통해 릴리즈 버전을 관리	태그를 통해 릴리즈 버전을 관리
커뮤니티	대규모 개발자 커뮤니티 및 오픈 소스 생태계	상대적으로 사용자가 작고 보다 엔터프라이즈 지향적인 커뮤니티
통합	다양한 도구, 환경과 통합 가능	

3. GIT 설치

- <https://git-scm.com/> 에서 stable release를 OS별로 내려 받아 설치하거나, Source Code를 빌드해서 설치 가능
- 3rd party git 구현체를 사용
 - ✓ Github Desktop
 - ✓ Sourcetree
 - ✓ GitKraken
 - ✓ Tower
 - ✓ ...
- 공식 릴리즈를 기준으로 설명
- Windows 설치
 - ✓ 다운로드

The screenshot shows the official Git website. At the top, there's a search bar with the placeholder "Search entire site...". Below the search bar, there's a diagram illustrating "cheap local branching" with multiple branches and merges between them. The main content area has several sections:

- About**: Describes Git as a free and open source distributed version control system.
- Documentation**: Links to command reference pages, the Git book, videos, and other material.
- Downloads**: Provides links for GUI clients and binary releases for Mac, Windows, Linux, and other platforms.
- Community**: Encourages involvement through bug reporting, mailing lists, chat, development, and more.
- Books**: A link to "Pro Git" by Scott Chacon and Ben Straub.
- Latest source Release**: Shows "2.41.0" with a "Release Notes (2023-06-01)" link and a "Download for Mac" button.
- Build Options**: Buttons for "Mac GUIs", "Tarballs", "Windows Build" (which is highlighted with a red box), and "Source Code".
- Companies & Projects Using Git**: Logos for Google, Microsoft, LinkedIn, Netflix, PostgreSQL, Camel, PostgreSQL, and Android.

● MacOS

- ✓ Mac의 “Command Line Tools for XCode”를 설치하면 git을 포함한 기본적인 개발 도구가 설치됨 ('xcode-select -install')
- ✓ 만약 별도로 설치하고 싶다면 Homebrew를 설치하고 (homebrew에서 제공하는 설치 스크립트를 다운 받아서 실행) Homebrew를 이용해 git을 설치하면 된다. ('brew install git')

```
edberg:~ > /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
=> Checking for `sudo` access (which may request your password)...
Password:
=> This script will install:
```

```

edberg:~ > brew install git
==> Fetching dependencies for git: gettext and pcre2
==> Fetching gettext
==> Downloading https://ghcr.io/v2/homebrew/core/gettext/manifests/0.21.1
#####
100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/gettext/blobs/sha256:fd7e48065c
#####
100.0%
==> Fetching pcre2
==> Downloading https://ghcr.io/v2/homebrew/core/pcre2/manifests/10.42
#####
100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/pcre2/blobs/sha256:7f414ed9d561
#####
100.0%
==> Fetching git
==> Downloading https://ghcr.io/v2/homebrew/core/git/manifests/2.41.0_1
#####
100.0%
==> Downloading https://ghcr.io/v2/homebrew/core/git/blobs/sha256:6b23e127ff57cb
#####
100.0%
==> Installing dependencies for git: gettext and pcre2
==> Installing git dependency: gettext
==> Pouring gettext--0.21.1.ventura.bottle.tar.gz
🍺 /usr/local/Cellar/gettext/0.21.1: 1,983 files, 20.3MB
==> Installing git dependency: pcre2
==> Pouring pcre2--10.42.ventura.bottle.tar.gz
🍺 /usr/local/Cellar/pcre2/10.42: 230 files, 6.3MB
==> Installing git
==> Pouring git--2.41.0_1.ventura.bottle.tar.gz
==> Caveats
The Tcl/Tk GUIs (e.g. gitk, git-gui) are now in the `git-gui` formula.
Subversion interoperability (git-svn) is now in the `git-svn` formula.

zsh completions and functions have been installed to:
/usr/local/share/zsh/site-functions
==> Summary
🍺 /usr/local/Cellar/git/2.41.0_1: 1,633 files, 48.7MB
==> Running `brew cleanup git`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
==> Caveats
==> git
The Tcl/Tk GUIs (e.g. gitk, git-gui) are now in the `git-gui` formula.
Subversion interoperability (git-svn) is now in the `git-svn` formula.

zsh completions and functions have been installed to:
/usr/local/share/zsh/site-functions

```

- Linux
 - ✓ 각 배포본의 패키지 매니저에서 제공하는 설치 기능을 사용
 - ✓ Ubuntu: sudo apt-get install git
 - ✓ CentOS or RHEL: yum install git
- Git 설치 확인

```

edberg:~ > git --version
git version 2.39.2 (Apple Git-143)

```

4. 기본 설정

- 명령어 :git config

```
edberg:~ > git config  
사용법 : git config [<옵션>]
```

설정 파일 위치

--global	공통 설정 파일을 사용합니다
--system	시스템 설정 파일을 사용합니다
--local	저장소 설정 파일을 사용합니다
--worktree	use per-worktree config file
-f, --file <파일>	지정한 설정 파일을 사용합니다
--blob <블롭 -id>	지정한 블롭 오브젝트에서 설정을 읽습니다

동작

--get	get value: name [value-pattern]
--get-all	get all values: key [value-pattern]
--get-regexp	get values for regexp: name-regex [value-pattern]
--get-urlmatch	<URL>에 특정되는 값을 가져옵니다 : <섹션>[.<변수>] <URL>
>	
--replace-all	replace all matching variables: name value [value-pattern]
tern]	
--add	새 변수를 추가합니다 : <이름> <값>
--unset	remove a variable: name [value-pattern]
--unset-all	remove all matches: name [value-pattern]
--rename-section	섹션의 이름을 바꿉니다 : <옛-이름> <새-이름>
--remove-section	섹션을 제거합니다 : <이름>
-l, --list	전체 목록을 표시합니다
--fixed-value	use string equality when comparing values to 'value-pattern'
ttern'	
-e, --edit	편집 기능을 엽니다
--get-color	설정한 색을 찾습니다 : slot [<기본값>]
--get-colorbool	색 설정을 찾습니다 : slot [<표준 출력이 TTY인지 여부>]

값 종류

-t, --type <종류>	값이 해당 종류로 주어집니다
--bool	값이 "true" 또는 "false"입니다
--int	값이 실수입니다
--bool-or-int	값이 --bool 또는 --int입니다
--bool-or-str	value is --bool or string
--path	값이 경로(파일 또는 디렉토리 이름)입니다
--expiry-date	값이 만료 시각입니다

기타

-z, --null	값을 NUL 바이트로 끝냅니다
--name-only	변수 이름만 표시합니다
--includes	찾아볼 때 include 지시어를 고려합니다
--show-origin	설정의 출처를 표시합니다 (파일, 표준 입력, 블롭, 명령행)
)	
--show-scope command	show scope of config (worktree, local, global, system,
--default <값>	--get 옵션에서, 해당 항목이 없으면 기본값을 사용합니다

● --global과 --local

- ✓ global : 사용자 홈 디렉토리의 .gitconfig 파일의 설정이며, 로그인

- 한 사용자의 기본 전역 설정
 - ✓ local : 현재 위치의 저장소의 설정으로 저장소 단위의 설정이며, global 설정을 덮어 씁니다. 저장소 별로 다른 정보가 필요한 경우에 사용
- 설정 syntax
 - ✓ git config [--global|--local] 설정항목 “값” (default는 local)
- 기본 설정 항목
 - ✓ 사용자 이름 :user.name
 - ✓ 사용자 메일 주소 :user.email

```
edberg:~ > git config --global user.name "Sangeun Bak"
edberg:~ > git config --global user.mail "edberg.bak@airi.kr"
```

✓ 설정 확인 :git config --list

```
edberg:~ > git config --list
credential.helper=osxkeychain
user.name=Sangeun Bak
user.mail=edberg.bak@airi.kr
```

- ✓ 설정 수정
 - 설정을 엎어 쓰면 되는데
 - 삭제해야 하는 경우 :git config --unset 또는 .gitconfig 파일 수정

```
edberg:~ > git config --global --unset user.mail
edberg:~ > git config --global user.email "edberg.bak@airi.kr"
edberg:~ > git config -l
credential.helper=osxkeychain
user.name=Sangeun Bak
user.email=edberg.bak@airi.kr
```

5. Repository를 생성해 봅시다.

- 부모 폴더에서 git init repository_name 또는 저장소 폴더에서 git init

```
edberg:~/flyai > mkdir tutorial1
edberg:~/flyai > cd tutorial1
edberg:~/flyai/tutorial1 > git init
힌트: Using 'master' as the name for the initial branch. This default branch name
힌트: is subject to change. To configure the initial branch name to use in all
힌트: of your new repositories, which will suppress this warning, call:
힌트:
힌트:   git config --global init.defaultBranch <name>
힌트:
힌트: Names commonly chosen instead of 'master' are 'main', 'trunk' and
힌트: 'development'. The just-created branch can be renamed via this command:
힌트:
힌트:   git branch -m <name>
/Users/edbergbak/flyai/tutorial1/.git/ 안의 빈 깃 저장소를 다시 초기화했습니다
```

- 만들어진 후의 디렉토리의 상황

```

edberg:~/flyai/tutorial1 > ls
edberg:~/flyai/tutorial1 > ls -atl
total 0
drwxr-xr-x@ 9 edbergbak staff 288 7 7 20:08 .git
drwxr-xr-x@ 3 edbergbak staff 96 7 7 20:08 .
drwxr-xr-x@ 9 edbergbak staff 288 7 7 20:08 ..

```

- .git 디렉토리 : git 저장소의 모든 것이 담겨 있음

```

edberg:~/flyai/tutorial1 > cd .git
edberg:~/flyai/tutorial1/.git > ls -atl
total 24
drwxr-xr-x@ 4 edbergbak staff 128 7 7 20:08 objects
drwxr-xr-x@ 9 edbergbak staff 288 7 7 20:08 .
-rw-r--r--@ 1 edbergbak staff 137 7 7 20:08 config
-rw-r--r--@ 1 edbergbak staff 23 7 7 20:08 HEAD
drwxr-xr-x@ 4 edbergbak staff 128 7 7 20:08 refs
drwxr-xr-x@ 16 edbergbak staff 512 7 7 20:08 hooks
-rw-r--r--@ 1 edbergbak staff 73 7 7 20:08 description
drwxr-xr-x@ 3 edbergbak staff 96 7 7 20:08 info
drwxr-xr-x@ 3 edbergbak staff 96 7 7 20:08 ..

```

- ✓ config : global configuration과 구분되는 해당 저장소 만의 설정

```

edberg:~/flyai/devops/.git > cat config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    ignorecase = true
    precomposeunicode = true
[user]
    name = Crapas
    mail = crapas@gmail.com

```

- ✓ description : 저장소 설명
- ✓ info/exclude : git이 무시할 파일의 패턴을 기록. gitignore와는 달리 local repository에만 영향을 미치며, 협업자들에게 전달되지 않음
- ✓ info/refs : 참조 목록. 참조 정보를 제공하는 git 프로토콜에서 사용

```

c7a5bda88de4ad7f4645e6f8e44a78c5c2859d41 HEAD
c7a5bda88de4ad7f4645e6f8e44a78c5c2859d41 refs/heads/main
9e0233b6b9b89478386d5260f476a7422b1a9a0f refs/heads/feature
    ✓ info/sparse-checkout – 기록한 디렉토리의 파일만 가지고 옴
    ✓ info/attributes – 특정 파일에 대한 사용자 정의 속성을 정의
        ➤ *.jpg -diff : git에서 diff를 할 때 .jpg 파일은 제외
    ✓ COMMIT_EDITMSG : 임시파일, 현재의 커밋 메시지로 사용이 끝나면
        삭제됨
    ✓ index : 스테이징 영역 (커밋에 포함될 준비가 된 변경 사항들이 기록
        되는 영역) 을 관리하는데 사용되며, 변경된 파일들의 스냅샷과 해당

```

파일들의 메타데이터를 기록. git add 시점에 업데이트

- ✓ packed-refs : 참조의 전체 목록과 그에 대응되는 해시값을 기록
- ✓ objects/ : git 저장소의 객체들이 저장되는 디렉토리며 압축되어 저장
 - Blob 객체 : 파일의 내용을 나타내는 객체
 - Tree 객체 : 디렉토리 구조를 나타내는 객체.
 - Commit 객체 : 커밋에 대한 정보 객체로, 커밋의 메타데이터, 부모 커밋에 대한 참조, 트리 객체에 대한 참조를 포함
 - SHA-1 해시 함수를 사용하여 고유한 식별자를 가지고 있음

```
edberg:~/flyai/devops/.git/objects > cd c6
```

```
edberg:~/flyai/devops/.git/objects/c6 > ls
```

```
2133d1f7ddfb9473605d3fe980da48e2e8633
```

- ✓ ORIG_HEAD : HEAD 커밋을 조작하면 조작 이전의 위치를 참조하는 포인터로 사용
 - 브랜치 이동

```
edberg:~/flyai/devops/.git > cat ORIG_HEAD
```

```
4473ae35528732fbe1197c2ebde0f82c1d2857c5
```

```
edberg:~/flyai/devops/.git > git log
```

```
commit c62133d1f7ddfb9473605d3fe980da48e2e8633 (HEAD -> master)
```

```
Merge: 4473ae3 5f413ca
```

```
Author: Crapas <edberg.bak@airi.kr>
```

```
Date: Wed Jul 5 15:22:36 2023 +0900
```

```
fix : Resolve conflict
```

```
Resolving conflict between first and second branch while merging into master
```

```
.
```

```
Footer : master branch
```

```
commit 5f413cab4363120e5eb8c19a0c85b297405a282
```

```
Author: Crapas <edberg.bak@airi.kr>
```

```
Date: Wed Jul 5 15:13:10 2023 +0900
```

➤ git reset 명령 실행

- ✓ HEAD : 현재 작업 중인 브랜치를 가리키는 포인터
 - 브랜치 이름

```
edberg:~/flyai/devops/.git > cat HEAD
```

```
ref: refs/heads/master
```

➤ 혹은 커밋 해시값 (커밋이 브랜치와 분리된 경우)

- ✓ logs : git 사용의 기록
- ✓ hooks : 툭 정의. 나중에 다시 설명 합니다.

- 현재 저장소만의 설정 : git config --local

```
edberg:~/flyai/tutorial1 > git config --local user.name "Edberg"
edberg:~/flyai/tutorial1 > git config --local user.email "edberg.s@gmail.com"
edberg:~/flyai/tutorial1 > git config --local -l
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
core.ignorecase=true
core.precomposeunicode=true
user.name=Edberg
user.email=edberg.s@gmail.com
edberg:~/flyai/tutorial1 > cat .git/config
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    ignorecase = true
    precomposeunicode = true
[user]
    name = Edberg
    email = edberg.s@gmail.com
```

- 저장소 상태 보기 명령 :git status
 - ✓ 현재 저장소가 어떤 상태에 있는가?

```
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master

아직 커밋이 없습니다

커밋 할 사항 없음 (파일을 만들거나 복사하고 "git add"를 사용하면 추적합니다)
```

- ✓ 파일을 하나 추가해 본 후의 status

```
edberg:~/flyai/tutorial1 > touch newfile  
edberg:~/flyai/tutorial1 > git status  
현재 브랜치 master
```

아직 커밋이 없습니다

추적하지 않는 파일 :
(커밋할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)
newfile

커밋할 사항을 추가하지 않았지만 추적하지 않는 파일이 있습니다 (추적하려면 "git add"를 사용하십시오)

```
edberg:~/flyai/tutorial1 > rm newfile  
edberg:~/flyai/tutorial1 > git status  
현재 브랜치 master
```

아직 커밋이 없습니다

커밋할 사항 없음 (파일을 만들거나 복사하고 "git add"를 사용하면 추적합니다)

```
edberg:~/flyai/devops > git status  
현재 브랜치 master  
커밋할 사항 없음, 작업 폴더 깨끗함  
edberg:~/flyai/devops > touch garbage  
edberg:~/flyai/devops > git status  
현재 브랜치 master  
추적하지 않는 파일 :  
(커밋할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)  
garbage
```

커밋할 사항을 추가하지 않았지만 추적하지 않는 파일이 있습니다 (추적하려면 "git add"를 사용하십시오)

6. 파일 추가와 스테이징 (Staging)

- README.txt 파일 추가

```
edberg:~/flyai/tutorial1 > cat > README.txt  
This is sample txt file for tutorial  
edberg:~/flyai/tutorial1 > git status  
현재 브랜치 master
```

아직 커밋이 없습니다

추적하지 않는 파일 :
(커밋할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)
README.txt

커밋할 사항을 추가하지 않았지만 추적하지 않는 파일이 있습니다 (추적하려면 "git add"를 사용하십시오)

- README2.txt 추가

```
edberg:~/flyai/tutorial1 > cat > README2.txt
This is sample txt file for tutorial.
I'll remove it.
```

```
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master
```

아직 커밋이 없습니다

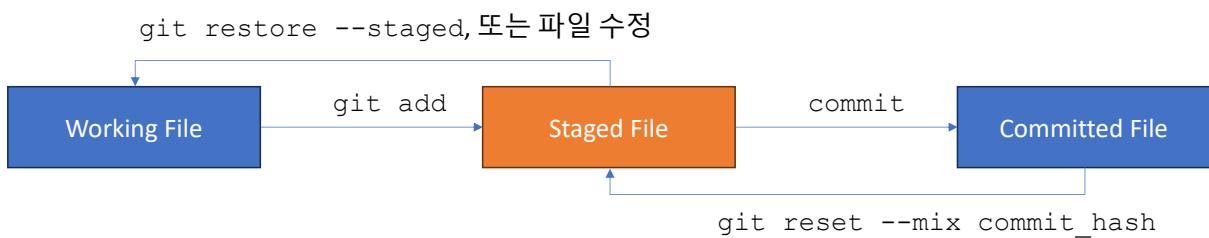
추적하지 않는 파일 :

(커밋할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)

README.txt
README2.txt

커밋할 사항을 추가하지 않았지만 추적하지 않는 파일이 있습니다 (추적하려면 "git add"를 사용하십시오)

- ✓ 이 상태에서는 어떠한 파일도 추적되지 않음
- 스테이징 영역에 추가 :git add
 - ✓ 스테이징 영역이란?
 - 변경된 파일 중 커밋할 준비가 된 파일을 선택적으로 추가하는 중간 영역
 - 스테이징 영역은 커밋할 파일의 스냅샷을 보관



✓ 스테이징 영역으로 보내기 :git add filename

```
edberg:~/flyai/tutorial1 > git add README.txt
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master
```

아직 커밋이 없습니다

커밋할 변경 사항 :

(스테이징 해제하려면 "git rm --cached <파일>..."을 사용하십시오)

새 파일 : README.txt

추적하지 않는 파일 :

(커밋할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)

README2.txt

✓ git add dir_path: 디렉토리와 그 하위 디렉토리 전체를 스테이징

```
edberg:~/flyai/tutorial1 > git add .
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master
```

아직 커밋이 없습니다

커밋할 변경 사항 :

(스테이지 해제 하려면 "git rm --cached <파일>..."을 사용하십시오)

새 파일 : README.txt
새 파일 : README2.txt

- ✓ git add --all: 모든 변경 사항을 스테이징
- ✓ 스테이징 된 파일은 이때부터 추적 시작

```
edberg:~/flyai/tutorial1 > rm README2.txt
```

```
edberg:~/flyai/tutorial1 > git status
```

현재 브랜치 master

아직 커밋이 없습니다

커밋할 변경 사항 :

(스테이지 해제 하려면 "git rm --cached <파일>..."을 사용하십시오)

새 파일 : README.txt
새 파일 : README2.txt

커밋하도록 정하지 않은 변경 사항 :

(무엇을 커밋할지 바꾸려면 "git add/rm <파일>..."을 사용하십시오)

(use "git restore <file>..." to discard changes in working directory)

삭제함 : README2.txt

- ✓ 추적은 되지만, 그래도 임시 영역일 뿐

```
edberg:~/flyai/tutorial1 > git add --all
```

```
edberg:~/flyai/tutorial1 > git status
```

현재 브랜치 master

아직 커밋이 없습니다

커밋할 변경 사항 :

(스테이지 해제 하려면 "git rm --cached <파일>..."을 사용하십시오)

새 파일 : README.txt

7. 커밋 (commit)

A. 개념

- git repository의 핵심 개념! → 프로젝트의 버전 관리와 협업의 키포인트
- git에서 변경 사항의 스냅샷을 저장하는 작업 단위
- 각 커밋은 프로젝트의 특정 시점의 상태를 나타내며, 프로젝트의 이력을 기록하는데 사용

B. 특징

- 변경 사항의 스냅샷을 가지고 있음
 - ✓ 변경된 파일의 상태를 스냅샷으로 저장
 - ✓ 작업 디렉토리와 스테이징 영역의 변경 사항이 포함됨
 - ✓ 변경된 파일의 내용, 커밋된 파일/디렉토리의 구조와 상태를 저장
- 고유한 식별자
 - ✓ 해시값 형태의 고유한 값
 - ✓ 이 값을 기준으로 모든 커밋은 구분되고 식별됨
- 변경 사항의 추적
 - ✓ 커밋은 파일의 변경 사항을 추적
 - ✓ 이전 커밋과 차이를 비교해서 확인할 수 있음. 이를 통해 변경 이력을 확인하고 복원할 수 있음
- 불변성
 - ✓ 한번 생성된 커밋은 수정 또는 삭제할 수 없음
 - ✓ 새로운 커밋으로 대체하는 수 밖에 없으며, 이를 통해 안전성과 무결성을 보장
- 커밋 그래프
 - ✓ 이어지는 커밋을 그래프 형태로 구성할 수 있으며
 - ✓ 이 그래프를 통해 커밋들 간의 관계를 파악하고, 변경 사항의 흐름을 이해할 수 있음

C. 커밋과 관련해서 알아야 할 것들

- 언제 커밋할 것인가?
 - ✓ 내가 만든 작업이 종료되어 (크건 작건) 세상에 나갈 준비가 되었을 때
 - ✓ 오류가 없거나, 오류가 있더라도 syntax에 해당하지 않고 협업이 가능한 시점 → 협업자와 합의된 수준의 테스트가 완료 되었을 때. (팀장님 미안해요? 아니라, 팀장님 미안해야 함)
- 커밋을 하는 단위는?
 - ✓ 하나의 단일한 작업
- 커밋을 협업자에게 알리는 수단은?
 - ✓ 커밋 메시지 (아래 참조)
- 한 번 완료된 커밋은 수정할 수 있나요?
 - ✓ 없음. 새로운 커밋으로 덮어 쓰는 수 밖에 없음

D. 커밋해 보기 :git commit

- 커밋의 대상은 현재 스테이징 된 대상

```
edberg:~/flyai/tutorial1 > git commit  
커밋을 중지합니다. 커밋 메시지가 비어 있습니다.
```

- git commit 뒤에 파일을 명시해서 스테이징 된 대상 중 일부만 커밋할 수 있음
- 편집기가 실행되며 commit message 작성을 요청

```

# 변경 사항에 대한 커밋 메시지를 입력하십시오. '#' 문자로 시작하는
# 줄은 무시되고, 메시지를 입력하지 않으면 커밋이 중지됩니다.
#
# 현재 브랜치 master
#
# 최초 커밋
#
# 커밋 할 변경 사항 :
#     새 파일 : README.txt
#

```

✓ 커밋 메시지가 작성되지 않으면 커밋이 이루어지지 않음!

E. 커밋 메시지

i. 커밋 메시지란?

- 개발자가 변경 사항을 기록하고 설명하는 메시지
- 코드 변경의 이력을 추적하고, 협업 과정에서 다른 개발자와 의사소통하는데 활용

ii. 커밋 메시지의 주요 목적

- 변경 이력 추적 : 각 커밋이 가진 고유 식별자와 스냅샷을 사용, 변경 이력을 추적해 이전 버전으로 롤백하거나 특정 시점의 상태를 확인
- 협업과 코드 리뷰 : 다른 개발자와 협업하는 과정에서 커밋 메시지를 통해 자신의 수해안 작업을 명확하게 설명하여 다른 개발자들의 이해를 돋고 이를 리뷰할 수 있도록 함.
- 버전 관리와 이력 분석 : 각 커밋은 변경된 코드와 해당 변경의 이유 또는 수정된 버그에 대한 정보를 제공하여, 개발자를 포함한 여러 관련자들이 프로젝트의 특정 시점에 어떤 변경이 이루어졌으며, 그 변경의 의도와 배경을 이해할 수 있음. 또한 버전 별 차이점을 비교하고 특정 기능 또는 버그 수정, 그리고 일정 기간에 거친 기술 수준의 변화도 추적할 수 있음

iii. 커밋 메시지는 어떻게 만들어야 할까요?

Git에서 권장하는 메시지 포맷

- 짧은 줄로 (50자 미만) 커밋 메시지를 시작하고 한 줄 뛰우고 더 자세한 내용을 표시하는게 좋다.
- 커밋 메시지의 첫 번째 빈 줄 까지의 텍스트는 GIT에서 일반적으로 제목으로 취급된다.
- 사용하는 문자의 인코딩에는 크게 구애 받지 않는다. (주. 이건 무조건 신뢰하면 안된다.)
- 유효한 UTF-8 문자열을 가급적 사용하라
- 한편 예전에는 아래와 같은 내용을 권장했던 적도 있음
<Type>: Subject
(행 구분)
<Body>
(행 구분)
<Footer>

Type: feat / fix / docs / style / refactor (설계와 동작 결과가 바뀌지 않는 코드 수정) / test / chore (빌드 스크립트, 패키지 관련 수정) / ci

Subject : 해당 커밋의 간결한 요약. 현재시제, 첫 글자는 대문자, 마침표 사용하지 않음

Body : 본문. 커밋의 자세한 설명이 들어가는 부분. 라인 당 72자 이내로 작성

Footer : 부가 정보 (ex: issue 번호 등)

- 제목
 - ✓ 길게 쓰지 말 것 (의미를 해치지 않는 한에서)
 - ✓ 메시지 각 단어의 첫 글자를 대문자로 작성 (Make a awesome code → Make a Awesome Code)
 - ✓ 명령어 스타일의 문장 (I changed loop for performance → Change loop for performance)
 - ✓ 문장 마침 부호(.) 사용하지 말 것

- 본문
 - ✓ 이번 커밋의 대상과 목적을 가장 잘 설명할 수 있는 형태로 구성
 - 무엇을, 왜에 집중
 - 어떻게는 꼭 필요하지 않으면 주석으로
 - ✓ 커밋 메시지에 링크를 삽입해야 한다면, 시스템 내에 별도의 환경에 이를 기록하고 이 링크를 넣어서 커밋 메시지 자체를 지저분하게 만들지 않음
 - ✓ 본문이 필요 없는 커밋 메시지는 이를 기입 (Ex. 냉무)

- 가장 중요한 것은 조직의 커밋 메시지 컨벤션을 따르는 것!
 - ✓ Tutorial에서는 사정 상 짧게 단문으로 입력하겠습니다.

F. 커밋해 보기 (다시)

- git commit -m "커밋 메시지"

```
edberg:~/flyai/tutorial1 > git commit -m "First Commit of README.txt"
[master (최상위 -커밋) 6754511] First Commit of README.txt
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```

- 간단한 수정 사항의 경우에는 스테이징 없이 커밋도 가능 :git commit -a
 - ✓ 단, 스테이징이 된 적이 없는 파일은 적용되지 않음
- 커밋을 수정 :git commit --amend
 - ✓ 가급적 하지 마시기 바랍니다.

```
edberg:~/flyai/tutorial1 > git commit --amend -m "First Commit of REMDME.txt(U)"

[master 3f1782d] First Commit of REMDME.txt(U)
 Date: Fri Jul 7 20:35:17 2023 +0900
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
```

- 커밋을 한 후의 git status

```
edberg:~/flyai/tutorial1 > git status  
현재 브랜치 master  
커밋 할 사항 없음 , 작업 폴더 깨끗함
```

- git log: 저장소의 커밋 로그를 조회

```
edberg:~/flyai/tutorial1 > git log  
commit 3f1782dd4f0f5070af6c43ffffd9f0ef7cdfdb3e (HEAD -> master)  
Author: Edberg <edberg.s@gmail.com>  
Date:   Fri Jul 7 20:35:17 2023 +0900  
  
First Commit of REMDME.txt(U)
```

- 계속된 커밋

- ✓ TEST.txt 추가, README.txt 수정

```
edberg:~/flyai/tutorial1 > cat > TEST.txt  
This is another file for tutorial.  
edberg:~/flyai/tutorial1 > cat > README.txt  
This is sample txt file for tutorial. (2nd ver.)  
edberg:~/flyai/tutorial1 > git status  
현재 브랜치 master  
커밋 하도록 정하지 않은 변경 사항 :  
(무엇을 커밋할지 바꾸려면 "git add <파일>..."을 사용하십시오)  
(use "git restore <file>..." to discard changes in working directory)  
수정함 : README.txt
```

추적하지 않는 파일 :

(커밋할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)
TEST.txt

커밋할 변경 사항을 추가하지 않았습니다 ("git add" 및 / 또는 "git commit -a"를 사용하십시오)

```
edberg:~/flyai/tutorial1 > git add .  
edberg:~/flyai/tutorial1 > git status  
현재 브랜치 master  
커밋 할 변경 사항 :  
(use "git restore --staged <file>..." to unstage)  
수정함 : README.txt  
새 파일 : TEST.txt
```

- 변경한 것들을 커밋

- ✓ 파일 단위로 커밋할 수도 있음

```
edberg:~/flyai/tutorial1 > git commit README.txt -m "Second Commit of README.txt"
"
[master 0ede892] Second Commit of README.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master
커밋 할 변경 사항 :
(use "git restore --staged <file>..." to unstage)
  새 파일 : TEST.txt

edberg:~/flyai/tutorial1 > git commit TEST.txt -m "First Commit of TEST.txt"
[master b5ef42e] First Commit of TEST.txt
 1 file changed, 1 insertion(+)
  create mode 100644 TEST.txt
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master
커밋 할 사항 없음, 작업 폴더 깨끗함
edberg:~/flyai/tutorial1 > git log
commit b5ef42e9619fef159f93a0cb6a86c02a55927fc (HEAD -> master)
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:48:16 2023 +0900

    First Commit of TEST.txt

commit 0ede892397d985429b26269e0b2fddcca88ad898
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:47:50 2023 +0900

    Second Commit of README.txt

commit 3f1782dd4f0f5070af6c43ffffd9f0ef7cdfdb3e
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:35:17 2023 +0900

    First Commit of REMDME.txt(U)
```

✓ git log --oneline: 커밋 별 한 줄씩 표시

```
edberg:~/flyai/tutorial1 > git log --oneline
b5ef42e (HEAD -> master) First Commit of TEST.txt
0ede892 Second Commit of README.txt
3f1782d First Commit of REMDME.txt(U)
```

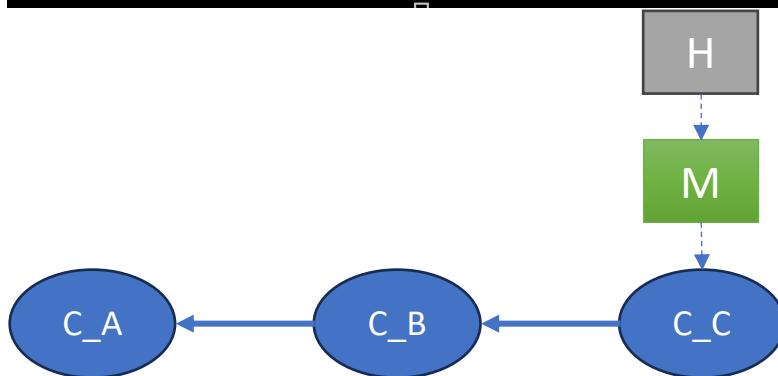
✓ git log --graph: 커밋 그래프 표시

```

edberg:~/flyai/tutorial1 > git log --graph
* commit b5ef42e9619fef159f93a0cb6a86c02a55927fc (HEAD -> master)
| Author: Edberg <edberg.s@gmail.com>
| Date:   Fri Jul 7 20:48:16 2023 +0900
|
|     First Commit of TEST.txt
|
* commit 0ede892397d985429b26269e0b2fddcca88ad898
| Author: Edberg <edberg.s@gmail.com>
| Date:   Fri Jul 7 20:47:50 2023 +0900
|
|     Second Commit of README.txt
|
* commit 3f1782dd4f0f5070af6c43ffffd9f0ef7cdfdb3e
  Author: Edberg <edberg.s@gmail.com>
  Date:   Fri Jul 7 20:35:17 2023 +0900
  First Commit of REMDME.txt(U)

edberg:~/flyai/tutorial1 > git log --graph --oneline
* b5ef42e (HEAD -> master) First Commit of TEST.txt
* 0ede892 Second Commit of README.txt
* 3f1782d First Commit of REMDME.txt(U)

```



✓ git log --patch : 무엇이 바뀌었는지

```
edberg:~/flyai/tutorial1 > git log --patch
commit b5ef42e9619fef159f93a0cb6a86c02a55927fcb (HEAD -> master)
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:48:16 2023 +0900

    First Commit of TEST.txt

diff --git a/TEST.txt b/TEST.txt
new file mode 100644
index 0000000..34ba264
--- /dev/null
+++ b/TEST.txt
@@ -0,0 +1 @@
+This is another file for tutorial.

commit 0ede892397d985429b26269e0b2fddcca88ad898
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:47:50 2023 +0900

    Second Commit of README.txt

diff --git a/README.txt b/README.txt
index b005a97..c52169e 100644
--- a/README.txt
+++ b/README.txt
@@ -1 +1 @@
-This is sample txt file for tutorial
+This is sample txt file for tutorial. (2nd ver.)

commit 3f1782dd4f0f5070af6c43ffffd9f0ef7cdfdb3e
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:35:17 2023 +0900

    First Commit of REMDME.txt(U)

diff --git a/README.txt b/README.txt
new file mode 100644
index 0000000..b005a97
--- /dev/null
+++ b/README.txt
@@ -0,0 +1 @@
+This is sample txt file for tutorial
```

✓ git log --stat: 커밋 통계 정보

```

edberg:~/flyai/tutorial1 > git log --stat
commit b5ef42e9619fef159f93a0cb6a86c02a55927fcb (HEAD -> master)
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:48:16 2023 +0900

    First Commit of TEST.txt

TEST.txt | 1 +
1 file changed, 1 insertion(+)

commit 0ede892397d985429b26269e0b2fddcca88ad898
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:47:50 2023 +0900

    Second Commit of README.txt

README.txt | 2 +-+
1 file changed, 1 insertion(+), 1 deletion(-)

commit 3f1782dd4f0f5070af6c43ffffd9f0ef7cdfdb3e
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:35:17 2023 +0900

    First Commit of REMDME.txt(U)

README.txt | 1 +
1 file changed, 1 insertion(+)

```

✓ 어떻게 저장되어 있을까?

```

edberg:~/flyai/tutorial1 > cd .git/objects
edberg:~/flyai/tutorial1/.git/objects > ls -atl
total 0
drwxr-xr-x@ 12 edbergbak  staff  384  7  7 20:48 ..
drwxr-xr-x@ 15 edbergbak  staff  480  7  7 20:48 .
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:48 b5
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:47 0e
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:47 95
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:47 57
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:45 34
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:45 c5
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:36 3f
drwxr-xr-x@  4 edbergbak  staff  128  7  7 20:36 b0
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:35 67
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:33 3a
drwxr-xr-x@  3 edbergbak  staff   96  7  7 20:29 fb
drwxr-xr-x@  2 edbergbak  staff   64  7  7 20:08 info
drwxr-xr-x@  2 edbergbak  staff   64  7  7 20:08 pack
edberg:~/flyai/tutorial1/.git/objects > cd b5
edberg:~/flyai/tutorial1/.git/objects/b5 > ls
ef42e9619fef159f93a0cb6a86c02a55927fcb

```

G. 수정과 되돌리기

- i. 스테이징 영역을 사용해 커밋 수정 : `git commit --amend`
 - 가장 마지막 커밋을 수정 또는 보충
 - 변경된 스테이징 영역을 반영

```
edberg:~/flyai/tutorial3 > git init
힌트: Using 'master' as the name for the initial branch. This default branch name
힌트: is subject to change. To configure the initial branch name to use in all
힌트: of your new repositories, which will suppress this warning, call:
힌트:
힌트:     git config --global init.defaultBranch <name>
힌트:
힌트: Names commonly chosen instead of 'master' are 'main', 'trunk' and
힌트: 'development'. The just-created branch can be renamed via this command:
힌트:
힌트:     git branch -m <name>
/Users/edbergbak/flyai/tutorial3/.git/ 안의 빈 깃 저장소를 다시 초기화했습니다
edberg:~/flyai/tutorial3 > touch abc
edberg:~/flyai/tutorial3 > touch def
edberg:~/flyai/tutorial3 > ls
abc      def
edberg:~/flyai/tutorial3 > git add abc
edberg:~/flyai/tutorial3 > git status
현재 브랜치 master
```

아직 커밋이 없습니다

커밋 할 변경 사항 :

(스테이지 해제하려면 "git rm --cached <파일>..."을 사용하십시오)

새 파일 : abc

추적하지 않는 파일 :

(커밋 할 사항에 포함하려면 "git add <파일>..."을 사용하십시오)

def

```
edberg:~/flyai/tutorial3 > git commit -m "Initial Commit"
[master (최상위 -커밋) 0e63526] Initial Commit
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 abc
edberg:~/flyai/tutorial3 > git add def
edberg:~/flyai/tutorial3 > git commit --amend -m "Initial Commit 2"
[master 03b6850] Initial Commit 2
  Date: Sat Jul 8 14:39:27 2023 +0900
  2 files changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 abc
  create mode 100644 def
edberg:~/flyai/tutorial3 > git status
현재 브랜치 master
커밋 할 사항 없음, 작업 폴더 깨끗함
edberg:~/flyai/tutorial3 > git log
commit 03b6850ac79d19edcb7bf4bbcf2d40d7a57582cf (HEAD -> master)
Author: Sangeun Bak <edberg.bak@airi.kr>
Date:   Sat Jul 8 14:39:27 2023 +0900
```

Initial Commit 2

- 기록에 남지 않음 - 이전의 커밋의 존재가 사라진다는 의미
- 커밋 기록을 깔끔하게 만드는 용도이지만, 잘못 사용하면 안됨

- ✓ 아차, 커밋에 빠뜨린 파일이 있어
 - ✓ 아차, 커밋 메시지에 오타가 났어
- ii. 커밋 후 수정 사항을 되돌리고 싶어 :git restore path 또는 file
- 복구할 수 없으니 주의할 것!

```
edberg:~/flyai/tutorial3 > cat > abc
I'm very handsome man.
edberg:~/flyai/tutorial3 > cat > def
I'm also very smart.
edberg:~/flyai/tutorial3 > git restore abc
edberg:~/flyai/tutorial3 > cat abc
edberg:~/flyai/tutorial3 > git restore .
edberg:~/flyai/tutorial3 > cat def
```

8. Branch와 Tag I

A. 브랜치(branch)란?

- 코드 변경 작업을 분리하여 병렬로 진행하고 독립적인 작업 공간을 제공하는 기능
- 브랜치를 사용해 동시에 여러 기능을 개발하거나 버그를 수정할 수 있음
- 각각의 브랜치에서 작업한 내용을 병합(merge)하여 하나의 통합된 코드로 만들 수 있음
- 상위 브랜치의 특정 커밋을 가리키는 참조 지점

B. 브랜치와 관련된 주요 개념

i. 기본 브랜치

- git 저장소 생성 시에 기본적으로 생성되는 브랜치
- 일반적으로 안정적인 코드의 버전이 유지되는 브랜치로 사용
- 최신 버전의 커밋을 이 브랜치에 병합하는게 일반적인 흐름

ii. 새로운 브랜치의 생성

- git branch 명령으로 새로운 브랜치를 생성
- 브랜치를 생성하면 현재 작업 중인 커밋을 기준으로 새로운 저장소 공간이 생기며
- 이 새로운 저장소 공간은 기존 또는 이후의 다른 브랜치와 독립적인 작업 공간을 가짐

iii. 브랜치 전환

- git checkout 명령어를 사용해 브랜치 전환 가능
- 일종의 ‘업무를 전환’한다는 의미

iv. 브랜치 병합

- git merge 명령어를 사용해 한 브랜치의 변경 사항을 다른 브랜치로 병합
- 이를 통해 다른 브랜치에서 작업한 내용을 기준 브랜치로 통합

v. 브랜치 관리

- 브랜치의 목록을 확인하거나 삭제
- 브랜치 병합 시 커밋 기록 유지 등

C. 태그(tag)란?

- 브랜치와 마찬가지로 특정 커밋을 가리키는 참조 지점

- 주로 특정 버전을 나타내기 위해 사용됨
 - ✓ 소프트웨어 릴리즈 버전
 - ✓ 마일스톤
 - ✓ 중요한 이벤트 등
- 태그의 유형
 - ✓ Annotated 태그
 - 태그를 만든 사람의 이름, 이메일과 태그를 만든 날자, 태그 메시지를 저장
 - ✓ Lightweight 태그
 - 브랜치와 유사하며, 단순히 특정 커밋을 가리키는 ‘참조’

D. 태그의 용도

- 버전 관리
- 중요한 이벤트, 마일스톤에 태그를 달아, 이력을 추적
- 배포 관리

E. 브랜치와 태그의 관계

	브랜치	태그
목적	독립적인 작업 공간 제공	특정 버전 식별 및 릴리즈
업데이트	새로운 커밋 생성 시 업데이트	불변적 정적 참조
변경 가능성	수정, 삭제, 브랜치 간 병합 가능	수정 불가능, 고정된 참조
관련 작업	코드 변경 작업	버전 식별, 릴리즈 관리

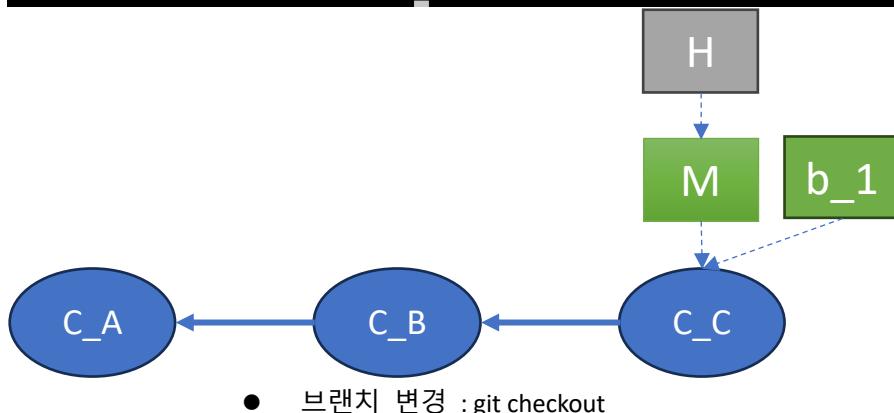
F. 브랜치 상에서의 작업

- 브랜치 확인 :git branch

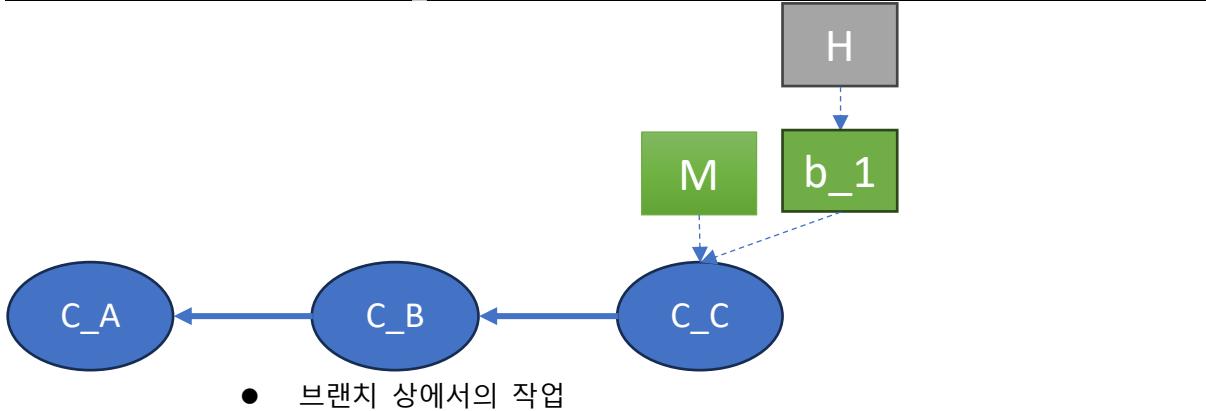
```
edberg:~/flyai/tutorial1 > git branch
* master
```

- 브랜치 생성 :git branch branch_name

```
edberg:~/flyai/tutorial1 > git branch branch_1
edberg:~/flyai/tutorial1 > git branch
branch_1
* master
```



```
edberg:~/flyai/tutorial1 > git checkout branch_1  
'branch_1' 브랜치로 전환합니다  
edberg:~/flyai/tutorial1 > git branch  
* branch_1  
  master
```



커밋할 변경 사항을 추가하지 않았습니다 ("git add" 및 / 또는 "git commit -a"를 사용하십시오)

```
edberg:~/flyai/tutorial1 > git commit -a -m "Commit of Removal of TEST.txt"  
[branch_1 136f9f6] Commit of Removal of TEST.txt  
 1 file changed, 1 deletion(-)  
 delete mode 100644 TEST.txt
```

```

edberg:~/flyai/tutorial1 > git status
현재 브랜치 branch_1
커밋할 사항 없음 , 작업 끌더 깨끗함
edberg:~/flyai/tutorial1 > git log
commit 136f9f673dac9cb25263d73e9032aeb6309f9218 (HEAD -> branch_1)
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 21:04:00 2023 +0900

    Commit of Removal of TEST.txt

commit b5ef42e9619fef159f93a0cb6a86c02a55927fc (master)
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:48:16 2023 +0900

    First Commit of TEST.txt

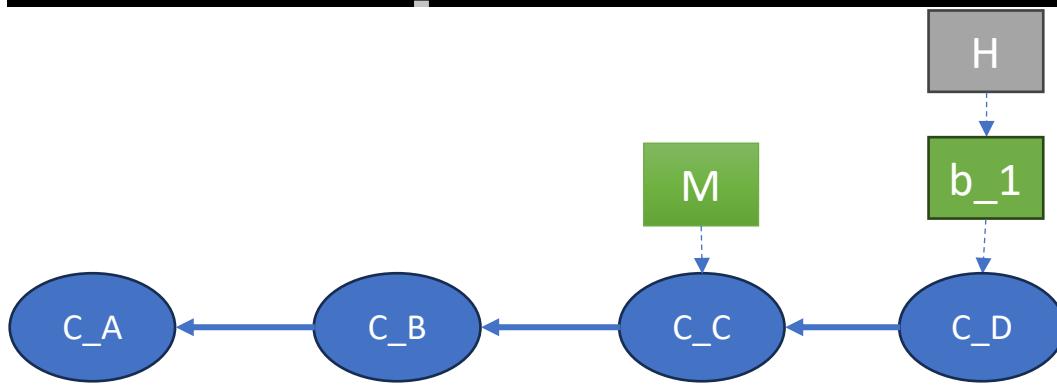
commit 0ede892397d985429b26269e0b2fddcca88ad898
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:47:50 2023 +0900

    Second Commit of README.txt

commit 3f1782dd4f0f5070af6c43ffffd9f0ef7cdfdb3e
Author: Edberg <edberg.s@gmail.com>
Date:   Fri Jul 7 20:35:17 2023 +0900

    First Commit of REMDME.txt(U)

```



```

edberg:~/flyai/tutorial1 > ls
README.txt
edberg:~/flyai/tutorial1 > git checkout master
'master' 브랜치로 전환합니다
edberg:~/flyai/tutorial1 > ls
README.txt      TEST.txt

```

- master, branch_1 각각 비교

```
edberg:~/flyai/tutorial1 > git checkout master
'master' 브랜치로 전환합니다
edberg:~/flyai/tutorial1 > git log --oneline --graph
* 57a767c (HEAD -> master) First Commit of TEST.txt
* 77e383c Second Commit of README.txt
* 8f345db First Commit of README.txt(U)
edberg:~/flyai/tutorial1 > git checkout branch_1
'branch_1' 브랜치로 전환합니다
edberg:~/flyai/tutorial1 > git log --oneline --graph
* a7c0c4c (HEAD -> branch_1) Commit of Removal of TEST.txt
* 57a767c (master) First Commit of TEST.txt
* 77e383c Second Commit of README.txt
* 8f345db First Commit of README.txt(U)
```

- 새로운 브랜치를 생성
 - ✓ git checkout -b branch_name : git branch + git checkout

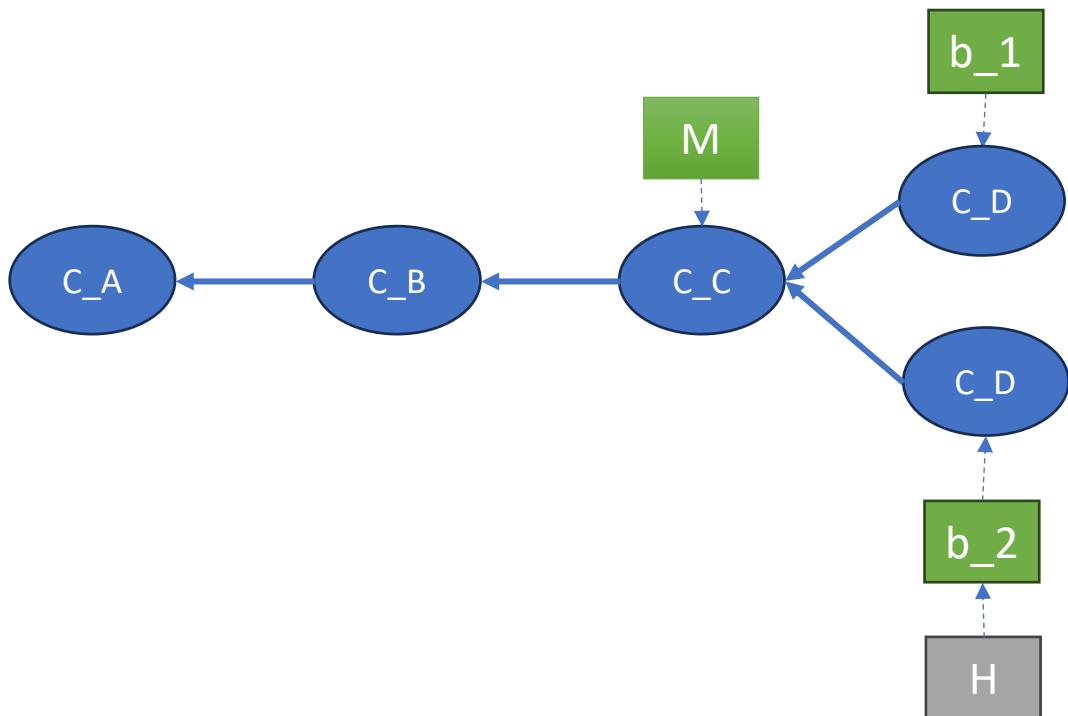
```
edberg:~/flyai/tutorial1 > git checkout -b branch_2
새로 만든 'branch_2' 브랜치로 전환합니다
```

- 새로운 브랜치에서 파일 수정, 커밋

```
edberg:~/flyai/tutorial1 > cat > README.txt
This is sample txt file for tutorial. (3rd ver. in branch_2)
edberg:~/flyai/tutorial1 > git commit -a -m "Third Commit of README.txt"
[branch_2 5774dab] Third Commit of README.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

- 이 상황에서의 저장소 상태와 커밋 그래프

```
edberg:~/flyai/tutorial1 > git log --graph --oneline
* de33516 (HEAD -> branch_2) Third Commit of README.txt
* 57a767c (master) First Commit of TEST.txt
* 77e383c Second Commit of README.txt
* 8f345db First Commit of README.txt(U)
```



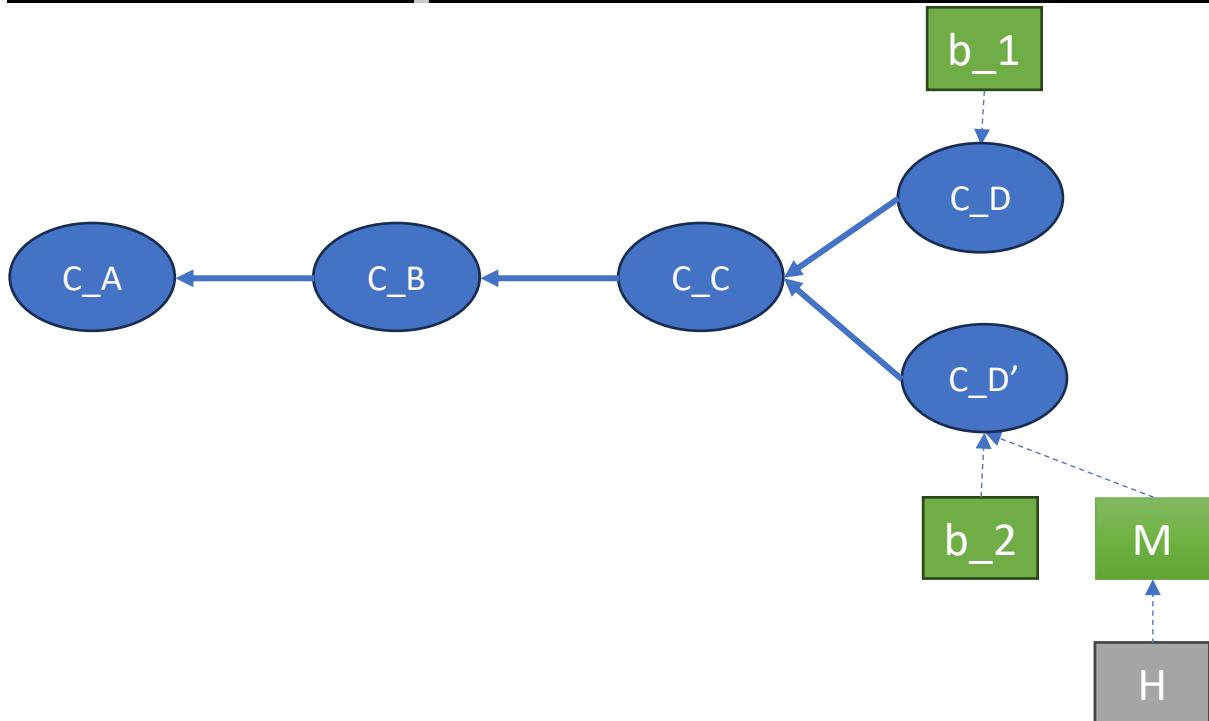
G. 브랜치 병합

- i. 브랜치 병합이란?
 - 두 개 이상의 브랜치에서 개발한 변경 사항을 하나로 통합
 - 동시에 진행된 작업을 합치고, 다른 브랜치에서 작업한 내용을 현재 브랜치로 가지고 옴
- ii. 브랜치 병합의 일반적인 단계
 - 목표 브랜치 확인
 - 병합 명령 실행
 - 병합 충돌 해결
 - 병합 커밋 생성
 - 병합된 내용 확인
- iii. 병합작업 : 병합할 브랜치 상에서 git merge 병합될 브랜치

```

edberg:~/flyai/tutorial1 > git branch
  branch_1
* branch_2
  master
edberg:~/flyai/tutorial1 > git checkout master
'master' 브랜치로 전환합니다
edberg:~/flyai/tutorial1 > git log --graph --oneline
* 57a767c (HEAD -> master) First Commit of TEST.txt
* 77e383c Second Commit of README.txt
* 8f345db First Commit of README.txt(U)
edberg:~/flyai/tutorial1 > git merge branch_2
업데이트 중 57a767c..de33516
Fast-forward
  README.txt | 2 +-
  1 file changed, 1 insertion(+), 1 deletion(-)
edberg:~/flyai/tutorial1 > git log --graph --oneline
* de33516 (HEAD -> master, branch_2) Third Commit of README.txt
* 57a767c First Commit of TEST.txt
* 77e383c Second Commit of README.txt
* 8f345db First Commit of README.txt(U)
edberg:~/flyai/tutorial1 > cat README.txt
This is sample txt file for tutorial. (3rd ver. in branch_2)

```



- 이런 형태의 merge를 Fast-Forward Merge라고 함

H. 충돌 해결

- 충돌은 언제 발생하는가?

- 두 개 이상의 브랜치에서 동일한 파일의 동일한 부분을 수정
- git이 자동으로 병합할 수 없는 상황을 의미하며, 수동으로 충돌을 해결해야 함

ii. 충돌 확인 방법

- 충돌이 발생하면 해당 파일에 “merge marker” 구문을 삽입
- '<<<', '===','>>>'와 같은 형식으로 표시

iii. 충돌의 실제 예

- 브랜치를 이동한 후 다른 브랜치에서 수정한 파일을 수정후 커밋

```
edberg:~/flyai/tutorial1 > git branch
branch_1
branch_2
* master
edberg:~/flyai/tutorial1 > git checkout branch_1
'branch_1' 브랜치로 전환합니다
edberg:~/flyai/tutorial1 > ls
README.txt
edberg:~/flyai/tutorial1 > cat > README.txt
This is sample txt file for tutorial. (3rd ver. in branch_1)
edberg:~/flyai/tutorial1 > git commit -a -m "Third Commit of README.txt"
[branch_1 2707bfc] Third Commit of README.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

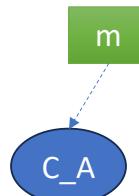
- 이 시점에 병합을 실시하면 충돌 상황이 발생

```
edberg:~/flyai/tutorial1 > git checkout master
'master' 브랜치로 전환합니다
edberg:~/flyai/tutorial1 > git merge branch_1
자동 병합 : README.txt
충돌 (내용) : README.txt에 병합 충돌
자동 병합이 실패했습니다. 충돌을 바로잡고 결과물을 커밋하십시오.
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master
병합하지 않은 경로가 있습니다.
(충돌을 바로잡고 "git commit"을 실행하십시오)
(병합을 중단하려면 "git merge --abort"를 사용하십시오)

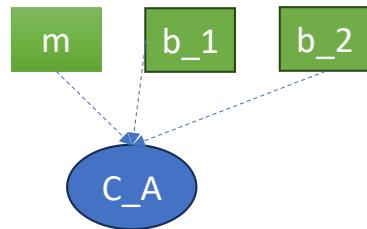
커밋할 변경 사항 :
    삭제함 :      TEST.txt

병합하지 않은 경로 :
(해결했다고 표시하려면 "git add <파일>..."를 사용하십시오)
    양쪽에서 수정 : README.txt
```

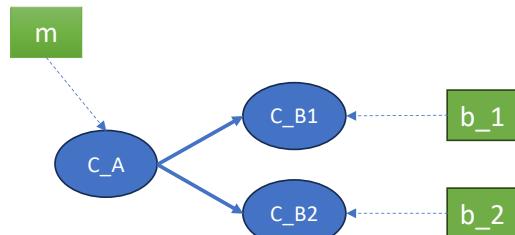
- 왜 충돌이 났을까?



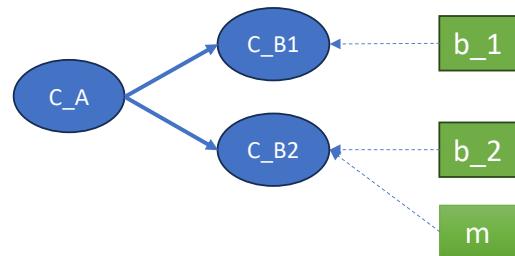
- ✓ branch_1 생성 : master의 README.txt → branch_1의 README.txt
- ✓ branch_2 생성 : master의 README.txt → branch_2의 README.txt



- ✓ branch_1, branch_2의 README.txt의 동일한 위치를 각자 수정, 커밋 (충돌!)



- ✓ 둘 중 하나를 병합하는 것은 문제가 되지 않음.



- ✓ 나머지 하나를 병합 할 때 문제를 확인하게 됨

● Merge Marker

```

edberg:~/flyai/tutorial1 > cat README.txt
<<<<< HEAD
This is sample txt file for tutorial. (3rd ver. in branch_2)
=====
This is sample txt file for tutorial. (3rd ver. in branch_1)
>>>>> branch_1

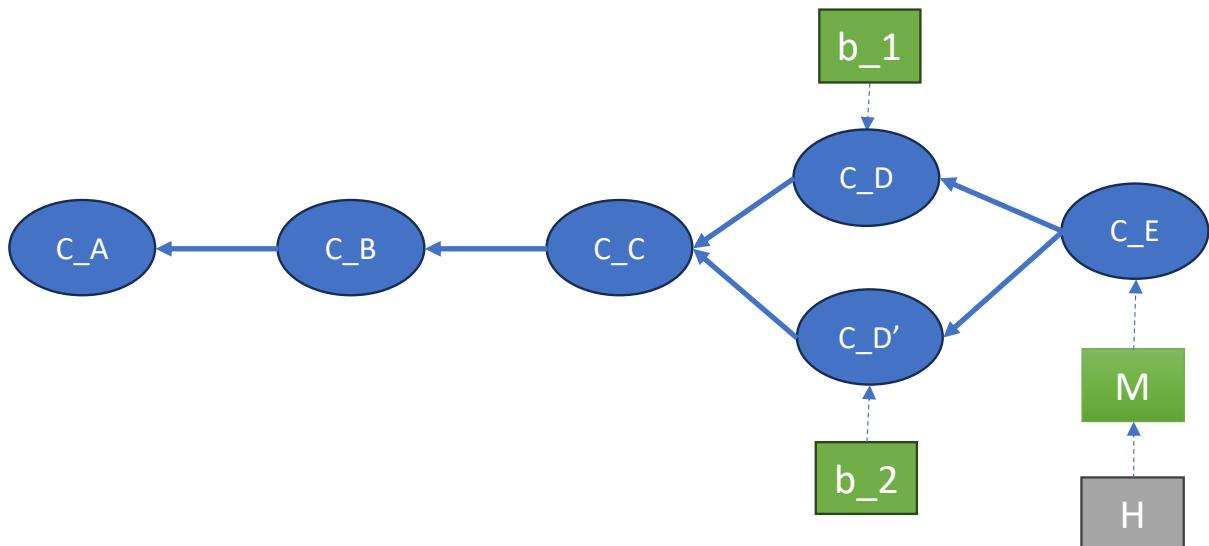
```

iv. 충돌을 해결하는 방법 : 직접 수정 후 스테이징, 커밋

```

edberg:~/flyai/tutorial1 > cat > README.txt
This is sample txt file for tutorial. (4th ver. with branch_2 and branch_1)
edberg:~/flyai/tutorial1 > git add .
edberg:~/flyai/tutorial1 > git commit -m "Reslove Confilct"
[master 75adf3b] Reslove Confilct
edberg:~/flyai/tutorial1 > ls
README.txt      TEST.txt
edberg:~/flyai/tutorial1 > cat README.txt
This is sample txt file for tutorial. (4th ver. with branch_2 and branch_1)
edberg:~/flyai/tutorial1 > git status
현재 브랜치 master
커밋 할 사항 없음 , 작업 풀 더 깨끗 함

```



- ✓ 부모가 여러 개인 병합
- ✓ 3-way merge로 가장 일반적인 병합의 형태 (충돌 유무와 무관하게)
- 브랜치 간에도 동일하게 병합, 충돌 해결 가능
- v. 문어발 병합 (Merge by octopus strategy)
 - 한 번에 여러 개의 브랜치 머지 가능.
 - 단 충돌 발생 시 해결이 매우 어렵기 때문에 하지 않는 것을 추천

```

edberg:~/flyai/tutorial2 > ls
test1  test2
edberg:~/flyai/tutorial2 > cat test1
abc
edberg:~/flyai/tutorial2 > cat test2
def
edberg:~/flyai/tutorial2 > git branch
* master
edberg:~/flyai/tutorial2 > git branch b1
edberg:~/flyai/tutorial2 > git branch b2
edberg:~/flyai/tutorial2 > git checkout b1
'b1' 브랜치로 전환합니다
edberg:~/flyai/tutorial2 > cat > test1
123
edberg:~/flyai/tutorial2 > git commit -a -m "Modify test1"
[b1 8ace911] Modify test1
 1 file changed, 1 insertion(+), 1 deletion(-)
edberg:~/flyai/tutorial2 > git checkout b2
'b2' 브랜치로 전환합니다
edberg:~/flyai/tutorial2 > cat > test2
456
edberg:~/flyai/tutorial2 > git commit -a -m "Modify test2"
[b2 a1654f1] Modify test2
 1 file changed, 1 insertion(+), 1 deletion(-)
edberg:~/flyai/tutorial2 > git checkout master
'master' 브랜치로 전환합니다
edberg:~/flyai/tutorial2 > git merge b1 b2
다음으로 정방향 진행: b1
b2에 간단한 병합 시도합니다
Merge made by the 'octopus' strategy.
 test1 | 2 +-
 test2 | 2 +-
 2 files changed, 2 insertions(+), 2 deletions(-)
edberg:~/flyai/tutorial2 > cat test1 test2
123
456
edberg:~/flyai/tutorial2 > git log --graph --oneline
*   8dfde70 (HEAD -> master) Merge branches 'b1' and 'b2'
|\ \
| * a1654f1 (b2) Modify test2
* | 8ace911 (b1) Modify test1
| /
* a617665 Initial Commit
* eadd666 Initial Commit

```

- 충돌 해결 도구 :git mergetool

vim

abc
def

abc

abc
xyz

./test1_LOCAL_2586 ./test1_BASE_2586 ./test1_REMOTE_2586

abc
=<<<< .merge_file_66mULw
def
=====
xyz
>>>> .merge_file_dLgDkX

test1
"test1" 6L, 74B

- 병합 여부에 따른 브랜치 확인 : git branch --merged 또는 git branch --no-merged

```
edberg:~/flyai/tutorial6 > git branch --merged
branch_1
branch_2
* master
edberg:~/flyai/tutorial6 > git branch --no-merged
edberg:~/flyai/tutorial6 > cd branch_1
cd: no such file or directory: branch_1
edberg:~/flyai/tutorial6 > git checkout branch_1
'branch_1' 브랜치로 전환합니다
edberg:~/flyai/tutorial6 > git branch --merged
* branch_1
edberg:~/flyai/tutorial6 > git branch --no-merged
branch_2
master
```

I. 브랜치 삭제

i. 삭제와 관련된 주의 사항

- 삭제를 하길 해야 함
 - ✓ 작업을 완료한 브랜치
 - ✓ 실험적인 브랜치의 실험이 끝난 경우
 - ✓ 내 부끄러움을 감추기 위한 용도로는 사용하면 안됨
- 삭제를 해도 되는지 확인
 - ✓ 작업 중인 브랜치인지의 여부 - 남은 일이 있는 브랜치인지 확인
 - ✓ 필요한 경우 백업이나 필요한 정보를 추출 후 삭제
- 리모트 저장소의 공유 브랜치인 경우 협업자들과 합의후 삭제 여부를 결정
- '-D' 옵션은 조심해서 사용

ii. 삭제 작업 : git branch -d

-

```
edberg:~/flyai/devops > git branch -d first-branch
first-branch 브랜치 삭제 (과거 5f413ca).
edberg:~/flyai/devops > git branch
* master
second-branch
```

- git branch -D : 강제로 브랜치를 삭제하는 옵션
 - ✓ '-d' 옵션은 삭제하고자 하는 브랜치의 커밋이 다른 브랜치에 다 포함된 경우 (즉 완전히 병합이 된 경우) 에만 삭제 가능
 - ✓ '-D' 옵션은 이와 무관하게 강제 삭제

```
edberg:~/flyai/tutorial1 > git branch -d branch_1
branch_1 브랜치 삭제 (과거 79655b8).
edberg:~/flyai/tutorial1 > git branch -D branch_2
branch_2 브랜치 삭제 (과거 d5da91a).
edberg:~/flyai/tutorial1 > git branch
* master
```

9. 리모트 저장소 (기초 사용 편)

A. 리모트 저장소란?

- 인터넷이나 네트워크 또는 다른 어딘가에 있는 저장소
- 여러 사용자가 동시에 작업하고 변경 사항을 공유하며 협업할 수 있는 곳
- 일반적으로 웹 상에서 호스팅 되거나 회사 내에서 서버 관리자가 운영
- github, gitlab 등
- 왜 필요한가?
 - ✓ 백업 및 복원
 - ✓ 공유 및 협업
 - ✓ 다양한 환경에서 작업
 - ✓ 이력 관리
 - ✓ CI/CD

B. 리모트 저장소 추가하기

- i. `git clone url` : 리모트 리파지토리를 복제
 - 자동으로 remote 저장소를 ‘origin’이라는 이름으로 추가

```
edberg:~/flyai > git clone https://github.com/crapas/dp.git
'dp'에 복제합니다...
remote: Enumerating objects: 289, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 289 (delta 1), reused 6 (delta 1), pack-reused 281
오브젝트를 받는 중: 100% (289/289), 70.55 KiB / 10.08 MiB/s, 완료.
델타를 알아내는 중: 100% (132/132), 완료.
edberg:~/flyai > git remote
edberg:~/flyai > cd dp
edberg:~/flyai/dp > ls git remote
ls: git: No such file or directory
ls: remote: No such file or directory
edberg:~/flyai/dp > git remote
origin
```

- ii. 현재 작업 공간에 추가하기 :`git remote add name url`
 - `git fetch name`

```
edberg:~/flyai/dp > git remote -v
origin  https://github.com/crapas/dp.git (fetch)
origin  https://github.com/crapas/dp.git (push)
edberg:~/flyai/dp > git remote add eep https://github.com/crapas/ep.git
edberg:~/flyai/dp > git remote -v
eep      https://github.com/crapas/ep.git (fetch)
eep      https://github.com/crapas/ep.git (push)
origin  https://github.com/crapas/dp.git (fetch)
origin  https://github.com/crapas/dp.git (push)
edberg:~/flyai/dp > ls
Appendix      ch1          ch3          ch5
README.md      ch2          ch4
edberg:~/flyai/dp > git fetch eep
remote: Enumerating objects: 307, done.
remote: Counting objects: 100% (307/307), done.
remote: Compressing objects: 100% (237/237), done.
remote: Total 307 (delta 9), reused 303 (delta 5), pack-reused 0
오브젝트를 받는 중 : 100% (307/307), 2.19 MiB | 6.13 MiB/s, 완료 .
델타를 알아내는 중 : 100% (9/9), 완료 .
https://github.com/crapas/ep URL에서
 * [새로운 브랜치]  main      -> eep/main
edberg:~/flyai/dp > ls
Appendix      ch1          ch3          ch5
README.md      ch2          ch4
```

- 리모트 브랜치 확인 : `git branch -r`
- ✓ 리모트 브랜치에 대한 상세한 내용은 뒤에 다시...

```
edberg:~/flyai/dp > git branch
* master
edberg:~/flyai/dp > git branch -r
  eep/main
  origin/HEAD -> origin/master
  origin/master
edberg:~/flyai/dp > git checkout eep/main
Note: switching to 'eep/main'.
```

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -c with the switch command. Example:

```
git switch -c <new-branch-name>
```

Or undo this operation with:

```
git switch -
```

Turn off this advice by setting config variable `advice.detachedHead` to `false`

HEAD의 현재 위치는 7b6640f so what

```
edberg:~/flyai/dp > ls
001_030      091_120      331_360      661_690      781_810
031_060      096          481_510      690_720      README.md
061_090      181_210      601_630      719          algoritms
```

c. 리모트 저장소에서 데이터 가지고 오기

i. git fetch

- 로컬에는 없지만 리모트에는 있는 모든 데이터를 가지고 옴

```

edberg:~/flyai > git clone https://github.com/crapas/ep.git
'ep'에 복제합니다...
remote: Enumerating objects: 307, done.
remote: Counting objects: 100% (307/307), done.
remote: Compressing objects: 100% (237/237), done.
remote: Total 307 (delta 9), reused 303 (delta 5), pack-reused 0
오브젝트를 받는 중 : 100% (307/307), 2.19 MiB | 8.07 MiB/s, 완료 .
델타를 알아내는 중 : 100% (9/9), 완료 .
edberg:~/flyai >
edberg:~/flyai >
edberg:~/flyai >
edberg:~/flyai > cd ep
edberg:~/flyai/ep > git branch
* main
edberg:~/flyai/ep > git branch -r
zsh: command not found: git
edberg:~/flyai/ep > git branch -r
  origin/HEAD -> origin/main
  origin/main

```

이후 ep를 수정 (README 수정을 main에서, 브랜치 추가해서 각각)

```

edberg:~/flyai/ep > git fetch origin
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 2), reused 0 (delta 0), pack-reused 0
오브젝트 뮐음 푸는 중 : 100% (6/6), 1.28 KiB | 261.00 KiB/s, 완료 .
https://github.com/crapas/ep URL에서
  7b6640f..e58670c main          -> origin/main
  * [새로운 브랜치]  crapas-patch-1 -> origin/crapas-patch-1
edberg:~/flyai/ep > git branch
* main
edberg:~/flyai/ep > git branch -r
  origin/HEAD -> origin/main
  origin/crapas-patch-1
  origin/main

```

- 단, 브랜치를 가지고 오기만 하며 병합하지 않음
- origin/main과 main 간의 병합이 필요

```
edberg:~/flyai/ep > git merge origin/main
```

업데이트 중 7b6640f..e58670c

Fast-forward

README.md | 2 ++

1 file changed, 2 insertions(+)

```
edberg:~/flyai/ep > cat README.md
```

zsh: command not found: cat

```
edberg:~/flyai/ep > cat README.md
```

ep

ep

Euler Project

ii. git pull

- 자동으로 로컬 브랜치와 병합

```
edberg:~/flyai > git clone https://github.com/crapas/ep.git
'ep'에 복제합니다...
remote: Enumerating objects: 313, done.
remote: Counting objects: 100% (313/313), done.
remote: Compressing objects: 100% (241/241), done.
remote: Total 313 (delta 11), reused 302 (delta 5), pack-reused 0
오브젝트를 받는 중: 100% (313/313), 2.19 MiB | 7.77 MiB/s, 완료.
델타를 알아내는 중: 100% (11/11), 완료.
edberg:~/flyai > cd ep
edberg:~/flyai/ep > ls
001_030      091_120      331_360      661_690      781_810
031_060      096          481_510      690_720      README.md
061_090      181_210      601_630      719          algorithms
edberg:~/flyai/ep > git branch -r
origin/HEAD -> origin/main
origin/crapas-patch-1
origin/main
edberg:~/flyai/ep > git pull origin
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
오브젝트 뮐음 푸는 중: 100% (3/3), 661 bytes | 330.00 KiB/s, 완료.
https://github.com/crapas/ep URL에서
e58670c..4358b6b main      -> origin/main
업데이트 중 e58670c..4358b6b
Fast-forward
 README.md | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
edberg:~/flyai/ep > git branch
* main
edberg:~/flyai/ep > git branch -r
origin/HEAD -> origin/main
origin/crapas-patch-1
origin/main
edberg:~/flyai/ep > git status
현재 브랜치 main
브랜치가 'origin/main'에 맞게 업데이트된 상태입니다.

커밋할 사항 없음, 작업 폴더 깨끗함
edberg:~/flyai/ep > cat README.md
# ep
# ep

Euler Project new
```

- D. 리모트 저장소에 내 변경사항 올리기 :git push [리모트저장소이름] [브랜치 이름]
 - i. 단, 내 리파지토리가 가장 최신일 때만 가능
 - ii. Fetch → Merge 후 push

```
edberg:~/flyai/ep > cat > TEST.txt
It's file for push test.
edberg:~/flyai/ep > git add *
edberg:~/flyai/ep > git commit -m "ADD File for Push Test"
[main 5c8a5cc] ADD File for Push Test
 1 file changed, 1 insertion(+)
 create mode 100644 TEST.txt
edberg:~/flyai/ep > git push origin main
Username for 'https://github.com': edberg.s@gmail.com
Password for 'https://edberg.s@gmail.com@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/crapas/ep.git/'
```

E. 리모트 저장소 정보 / 수정 / 삭제

i. 정보보기 : git remote show 저장소이름

```
edberg:~/flyai/ep > git remote show origin
* 리모트 origin
  가져오기 URL: https://github.com/crapas/ep.git
  푸시 URL: https://github.com/crapas/ep.git
  HEAD 브랜치: main
  리모트 브랜치:
    crapas-patch-1 추적됨
    main           추적됨
  'git pull'에 사용할 로컬 브랜치를 설정:
    main 병합: 리모트 main
  로컬 레퍼런스를 'git push'로 미러링:
    main에서 main(으)로 푸시 (로컬이 뒤떨어짐)
```

ii. 이름 수정 : git remote rename 저장소이름 바꿀이름

```
edberg:~/flyai/ep > git remote rename origin oriiiiigin
Renaming remote references: 100% (4/4), 완료.
edberg:~/flyai/ep > git remote show oriiiiigin
* 리모트 oriiiiigin
  가져오기 URL: https://github.com/crapas/ep.git
  푸시 URL: https://github.com/crapas/ep.git
  HEAD 브랜치: main
  리모트 브랜치:
    crapas-patch-1 추적됨
    main           추적됨
  'git pull'에 사용할 로컬 브랜치를 설정:
    main 병합: 리모트 main
  로컬 레퍼런스를 'git push'로 미러링:
    main에서 main(으)로 푸시 (로컬이 뒤떨어짐)
```

iii. 리모트 저장소 삭제 git remote remove 저장소이름

```

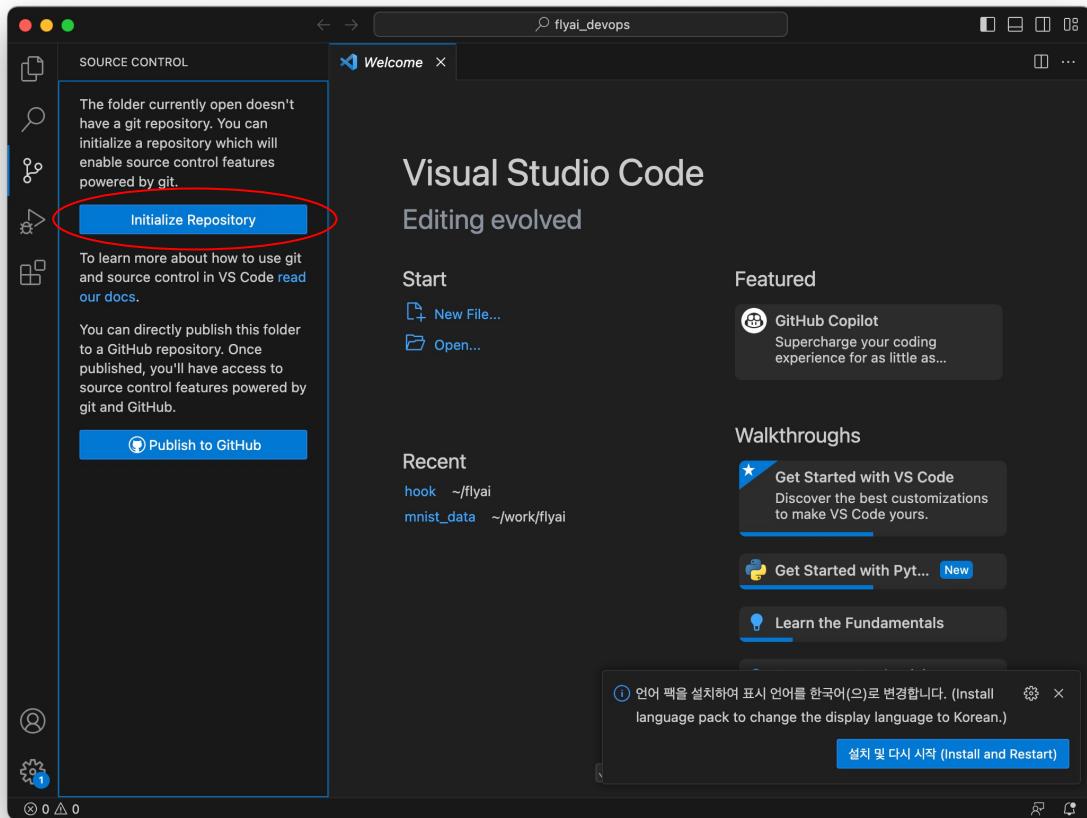
edberg:~/flyai/ep > git remote remove oriiiiigin
edberg:~/flyai/ep > ls
001_030      096      601_630      781_810
031_060      181_210    661_690      README.md
061_090      331_360    690_720      TEST.txt
091_120      481_510    719          algoritms
edberg:~/flyai/ep > git remote
edberg:~/flyai/ep > git status
현재 브랜치 main
커밋 할 사항 없음, 작업 풀 더 깨끗함
edberg:~/flyai/ep > git branch
* main

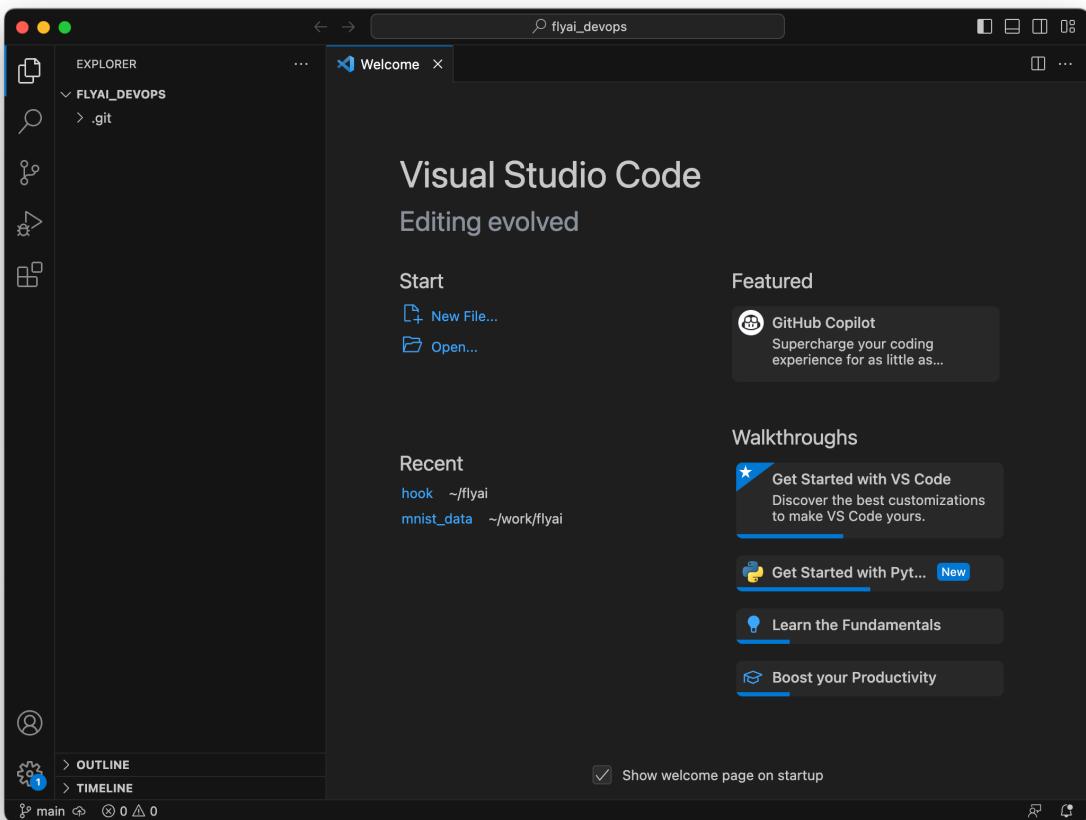
```

10. Visual Studio Code에서 git 사용하기

A. 시작

- git이 포함된 폴더를 열면 자동으로 사용 준비
- 혹은 Initialize Repository





✓ .git 폴더를 보려면?

➤ Settings > Files > Watcher Exclude에서 수정

B. 사용하기

- 파일 추가

C. Remote Repository 사용하기

github에 publish

수정 후 push - Sync Changes 1

github에서 한 줄 추가

vs에서 한 줄 추가

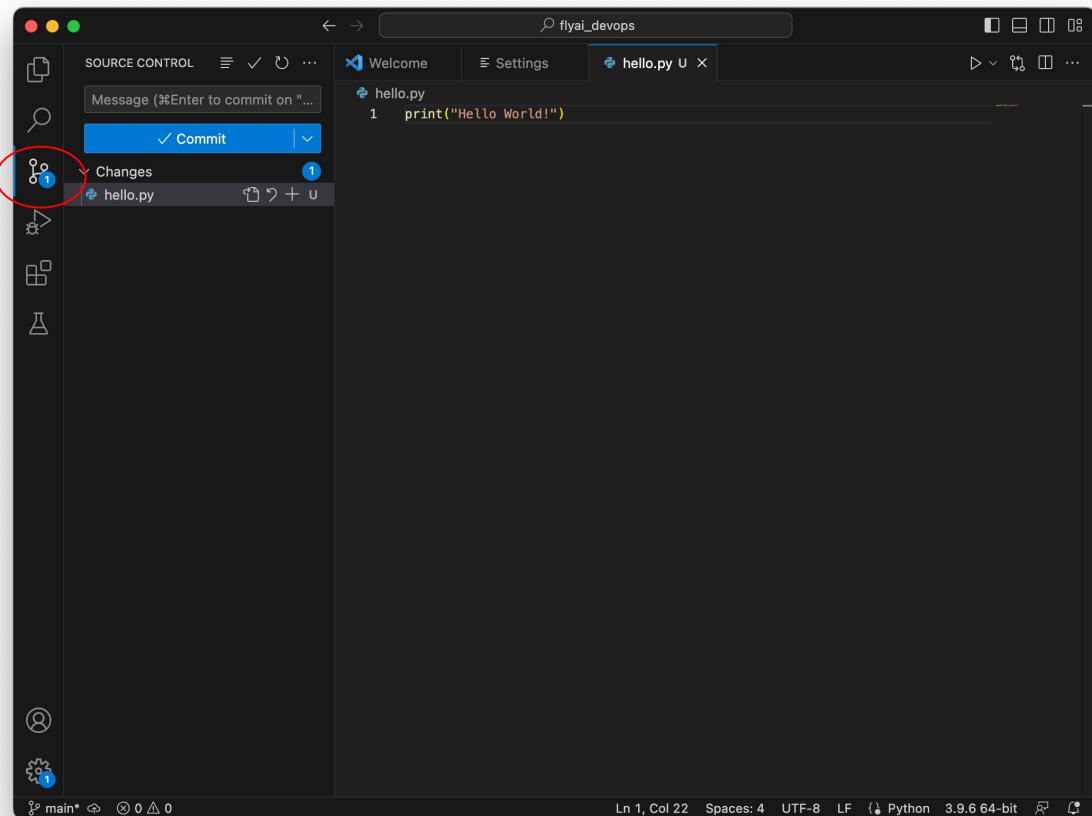
이번엔 push로...

The screenshot shows the Visual Studio Code interface with a dark theme. The Explorer sidebar on the left lists a project folder 'FLYAI_DEVOPS' containing '.git' and 'hello.py'. The 'hello.py' file is selected and has a red circle around its status icon in the list, which is labeled 'U' for 'Untracked'. The main editor area displays the following Python code:

```
1 print("Hello World!")
```

A yellow box highlights the text "추적 안되고 있다는 의미" (meaning "not tracked") in the bottom right corner of the editor area.

추적 안되고 있다는 의미



+ 눌러서 스테이징

Commit 눌러서 Commit - 메시지도 넣고

아래쪽 main 눌러서 branch 만들고 전환 (b1, b2)

main에서 파일 수정

```
print("Hello World!")
print("Hello branch_1!")
print("Hello branch_2!")
```

main에서 병합 : branch -> merge

b1에서 파일 수정

b2에서 파일 수정

main

11. Git Hook

A. 흑에 대해서

- 이벤트에 따라 자동으로 특정 스크립트를 실행하도록 하는 기능
- .git/hooks 디렉토리

B. 흑 실습 1 : pre-commit을 이용한 unit test

- commit 직전에 실행되는 흑

- 결과값이 non-0인 경우 commit이 진행되지 않음
- python unittest module
- pre-commit 설정

foo.py

```
def foo(param):
    if param == 0:
        return 1
    elif param == 1:
        return 0
    else:
        return -1
```

test.py

```
import unittest
from foo import foo

class TestFoo(unittest.TestCase):
    def test_0(self):
        result = foo(0)
        self.assertEqual(result, 1)

    def test_1(self):
        result = foo(1)
        self.assertEqual(result, 0)

    def test_other(self):
        result = foo(2)
        self.assertLess(result, 0)

if __name__ == "__main__":
    unittest.main()
```

pre-commit

```
#!/bin/sh

python3 -m unittest discover -s .

result=$?

if [ $result -ne 0 ]; then
    echo "Tests failed. Aborting commit."
    exit 1
fi

exit 0
```

```
edberg:~/flyai/hook > git commit -m "First commit of this tutorial"
...
-----
Ran 3 tests in 0.000s

OK
[master (최 상위 -커밋) 2e2d51c] First commit of this tutorial
  2 files changed, 25 insertions(+)
   create mode 100644 foo.py
   create mode 100644 test.py
```

수정 foo.py

```
def foo(param):
    if param == 0:
        return 1
    elif param == 1:
        return 0
    else:
        return 2
```

```
edberg:~/flyai/hook > git status
현재 브랜치 master
커밋하도록 정하지 않은 변경 사항 :
(무엇을 커밋할지 바꾸려면 "git add <파일>..."을 사용하십시오)
(use "git restore <file>..." to discard changes in working directory)
    수정함 :          foo.py

커밋할 변경 사항을 추가하지 않았습니다 ("git add" 및 /또는 "git commit -a"를
사용하십시오)
edberg:~/flyai/hook > git commit -a -m "fix : Make error on test"
..F
=====
FAIL: test_other (test.TestFoo)
-----
Traceback (most recent call last):
  File "/Users/edbergbak/flyai/hook/test.py", line 15, in test_other
    self.assertLess(result, 0)
AssertionError: 2 not less than 0

-----
Ran 3 tests in 0.001s

FAILED (failures=1)
Tests failed. Aborting commit.
```

```

edberg:~/flyai/hook > git add *
edberg:~/flyai/hook > git status
현재 브랜치 master
커밋 할 변경 사항 :
  (use "git restore --staged <file>..." to unstage)
    수정 함 :          foo.py

edberg:~/flyai/hook > git commit -m "fix : Make error on test"
.. F
=====
FAIL: test_other (test.TestFoo)
-----
Traceback (most recent call last):
  File "/Users/edbergbak/flyai/hook/test.py", line 15, in test_other
    self.assertLess(result, 0)
AssertionError: 2 not less than 0

-----
Ran 3 tests in 0.001s

FAILED (failures=1)
Tests failed. Aborting commit.
edberg:~/flyai/hook > git status
현재 브랜치 master
커밋 할 변경 사항 :
  (use "git restore --staged <file>..." to unstage)
    수정 함 :          foo.py

```

그래서 다시 수정

```

edberg:~/flyai/hook > git commit -a -m "fix : Make success of test"
...
-----
Ran 3 tests in 0.000s

OK
[master 7c46f60] fix : Make success of test
  1 file changed, 1 insertion(+), 1 deletion(-)

```

C. 흑 실습 2: post-merge을 이용한 pulling 후 데이터 변경 확인 흑

- merge 이후에 실행되는 흑
 - ✓ pull은 merge를 동반하는 작업
- 결과값이 non-0인 경우 commit0| 진행되지 않음
- when_changed.py
- post-merge 추가
 - ✓ 권한 확인 필요

이번엔 post-merge

파일 추가

```
when_changed.py
def changed():
    print("There's some change in data folder")

if __name__ == "__main__":
    changed()

post-merge 추가
```

```
#!/bin/sh

# Check if there are changes in the 'data' directory after a pull
changed_files=$(git diff-tree -r --name-only --no-commit-id ORIG_HEAD HEAD)
data_directory_changed=$(echo "$changed_files" | grep "^data/")

if [ -n "$data_directory_changed" ]; then
    python3 when_changed.py
fi
```

data 폴더 및 가짜 데이터 생성

commit

...

다른 폴더에 가서 clone

pull

```
edberg:~/flyai2/hook > cd .git/hooks
edberg:~/flyai2/hook/.git/hooks > ls
applypatch-msg.sample          pre-push.sample
commit-msg.sample              pre-rebase.sample
fsmonitor-watchman.sample     pre-receive.sample
post-update.sample             prepare-commit-msg.sample
pre-applypatch.sample         push-to-checkout.sample
pre-commit.sample              sendemail-validate.sample
pre-merge-commit.sample       update.sample
```

헐? 없다?

이유는... hook은 일종의 메타데이터 성격이라 push/pull등의 영향을 받지 않아요

그래서 가지고 와서 써야 해요. (copy)

```
edberg:~/flyai2/hook > git config advice.ignoreHook false
edberg:~/flyai2/hook > git pull
remote: 오브젝트 나열하는 중 : 5, 완료 .
remote: 오브젝트 개수 세는 중 : 100% (5/5), 완료 .
remote: 오브젝트 압축하는 중 : 100% (3/3), 완료 .
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
오브젝트 둑음 푸는 중 : 100% (3/3), 371 bytes | 185.00 KiB/s, 완료 .
file:///Users/edbergbak/flyai/hook URL에서
  a03d427..cde1269 master      -> origin/master
업데이트 중 a03d427..cde1269
Fast-forward
  data/pseudo-data2 ! 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 data/pseudo-data2
힌트 : '.git/hooks/post-merge' 후크가 실행 가능하도록 설정되지 않아서, 무시됩니다.

.
힌트 : 이 경고는 `git config advice.ignoreHook false` 명령으로 끌 수 있습니다.
edberg:~/flyai2/hook > cd ..
edberg:~/flyai2 > ls
devops hook
edberg:~/flyai2 > cd hook
edberg:~/flyai2/hook > ls
data          foo.py        test.py        when_changed.py
edberg:~/flyai2/hook > cd .git/hooks
edberg:~/flyai2/hook/.git/hooks > ls -atl
total 144
drwxr-xr-x@ 14 edbergbak staff  448 7 6 14:16 ..
-rw xr-xr-x@ 1 edbergbak staff  143 7 6 14:12 pre-commit
drwxr-xr-x@ 18 edbergbak staff  576 7 6 14:12 .
-rw-r--r--@ 1 edbergbak staff  291 7 6 14:12 post-merge
-rw xr-xr-x@ 1 edbergbak staff  2783 7 6 13:50 push-to-checkout.sample
-rw xr-xr-x@ 1 edbergbak staff  3650 7 6 13:50 update.sample
-rw xr-xr-x@ 1 edbergbak staff  1374 7 6 13:50 pre-push.sample
-rw xr-xr-x@ 1 edbergbak staff  424 7 6 13:50 pre-applypatch.sample
-rw xr-xr-x@ 1 edbergbak staff  416 7 6 13:50 pre-merge-commit.sample
-rw xr-xr-x@ 1 edbergbak staff  189 7 6 13:50 post-update.sample
-rw xr-xr-x@ 1 edbergbak staff  1492 7 6 13:50 prepare-commit-msg.sample
-rw xr-xr-x@ 1 edbergbak staff  544 7 6 13:50 pre-receive.sample
-rw xr-xr-x@ 1 edbergbak staff  4726 7 6 13:50 fsmonitor-watchman.sample
-rw xr-xr-x@ 1 edbergbak staff  478 7 6 13:50 applypatch-msg.sample
-rw xr-xr-x@ 1 edbergbak staff  1643 7 6 13:50 pre-commit.sample
-rw xr-xr-x@ 1 edbergbak staff  2308 7 6 13:50 sendemail-validate.sample
-rw xr-xr-x@ 1 edbergbak staff  4898 7 6 13:50 pre-rebase.sample
-rw xr-xr-x@ 1 edbergbak staff  896 7 6 13:50 commit-msg.sample
edberg:~/flyai2/hook/.git/hooks > chmod +x post-merge
edberg:~/flyai2/hook/.git/hooks > ls -atl
total 144
drwxr-xr-x@ 14 edbergbak staff  448 7 6 14:16 ..
-rw xr-xr-x@ 1 edbergbak staff  143 7 6 14:12 pre-commit
drwxr-xr-x@ 18 edbergbak staff  576 7 6 14:12 .
-rw xr-xr-x@ 1 edbergbak staff  291 7 6 14:12 post-merge
-rw xr-xr-x@ 1 edbergbak staff  2783 7 6 13:50 push-to-checkout.sample
-rw xr-xr-x@ 1 edbergbak staff  3650 7 6 13:50 update.sample
-rw xr-xr-x@ 1 edbergbak staff  1374 7 6 13:50 pre-push.sample
-rw xr-xr-x@ 1 edbergbak staff  424 7 6 13:50 pre-applypatch.sample
-rw xr-xr-x@ 1 edbergbak staff  416 7 6 13:50 pre-merge-commit.sample
-rw xr-xr-x@ 1 edbergbak staff  189 7 6 13:50 post-update.sample
-rw xr-xr-x@ 1 edbergbak staff  1492 7 6 13:50 prepare-commit-msg.sample
-rw xr-xr-x@ 1 edbergbak staff  544 7 6 13:50 pre-receive.sample
```

그래도 안되는 이유 → 이건 실행 권한 때문

```
edberg:~/flyai2/hook > git pull
remote: 오브젝트 나열하는 중: 5, 완료.
remote: 오브젝트 개수 세는 중: 100% (5/5), 완료.
remote: 오브젝트 압축하는 중: 100% (3/3), 완료.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
오브젝트 뭍음 푸는 중: 100% (3/3), 375 bytes | 187.00 KiB/s, 완료.
file:///Users/edbergbak/flyai/hook URL에서
  cde1269..58b9dc8 master    -> origin/master
업데이트 중 cde1269..58b9dc8
Fast-forward
  data/pseudo-data4 | 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 data/pseudo-data4
There's some change in data folder
edberg:~/flyai2/hook > git pull
이미 업데이트 상태입니다.
```

12. GIT 서버

A. 정의

- 협업을 위해서 제공되는 git 저장소
- 로컬 저장소에서는 이 Git 서버를 연결해서 사용 → 리모트 저장소

B. 서버 운영 방법

i. 프로토콜 결정

- Local
 - ✓ 가장 기본적인 저장소
 - ✓ 아무 설정을 할 필요가 없음
 - ✓ url 대신 파일 경로
 - git clone path
 - git clone [file:///\(path\)](file:///(path)) - 네트워크 상의 리모트 저장소와 동일하게 동작
- HTTP
 - ✓ HTTP 프로토콜을 통해서 동작하는 git repository
 - ✓ 두 종류의 HTTP protocol 지원
 - Smart HTTP : SSH나 Git Protocol처럼 동작하며, 경로가 HTTP인 것일 뿐, 현재 github등에서 제공되는 대표적인 방법
 - Dumb HTTP : 파일을 제공하는 웹 서버처럼 동작, 저장소를 http 디렉토리의 root에 넣으면 끝
- SSH
 - ✓ SSH 프로토콜을 통해서 동작
 - ✓ ssh:// 로 시작하는 URL을 사용
 - git clone ssh://user@server/xxx.git, git clone <user@server:xxx.git>

- ✓ SSH 인증 사용
- ✓ SSH 데몬만 설정되어 있으면 그대로 사용 가능
- Git
 - ✓ GIT 데몬을 사용하는 서버 (일반적으로 9418 포트 사용)
 - ✓ 전송 속도가 가장 빠르지만, 인증 메카니즘이 없음
- ii. 프로토콜에 따른 필요한 도구 설치
- iii. 리파지토리 생성하고 운영
- C. SSH 설치/설정
 - i. 설치
 - (물론, git은 설치되어 있어야 함)
 - 만들어진 git 디렉토리를 git 서버의 원하는 디렉토리에 복사
 - git clone id@server:path
 - ii. 인증 설정
 - (생략)
- D. SMART HTTP 설치/설정
 - i. 웹 서버 설치
 - ii. 사용하고자 하는 git directory를 HTTP에서 Serve 가능한 폴더로 등록 (그룹 권한 설정)
 - iii. 원하는 URL로 해당 디렉토리가 접근이 되도록 HTTP 서버를 설정
 - iv. 인증 설정 (Apache라면 htpasswd 파일 인증 등)

GIT은 혼자서는 의미가 없어요

항상 이용가능한 서버 상의 리파지토리가 있어야 하겠죠

13. Github

- A. 소개
 - i. 가장 유명한 Git 호스팅
 - ii. 호스팅 이외에 협업, 코드 검토, 이슈 추적, 그리고 CI/CD를 지원하기 위한 추가 기능 등을 제공
 - iii. 2008년 출시되었으며, Ruby로 개발
 - iv. SSH와 https 프로토콜을 지원
 - v. Github Enterprise : Github의 자체 구축 솔루션
 - vi. MS에서 인수해서 운영 중
- B. 둘러보기
 - i. 계정 생성
 - ii. Project Fork : 프로젝트에 기여하기

fork 누르고

로컬에서 clone

clone 한 녀석을 VS에 열기

브랜치 만들고

무엇인가 수정

로컬에 커밋

Push - git push origin branch

permission denied가 뜨면

токен 생성 (developer menu)

token을 기록하시고

이를 등록해서 암호로 사용

pull request 보내기

원래 repository로 가서

살펴보고 Merge pull request

C. 이슈와 토론

D. github action