

CLI 환경에서의 버전관리

로컬저장소 생성

- CLI 명령어만으로 로컬저장소를 생성하는 것이 목표
- Git bash 창에서 로컬저장소를 만들 위치로 이동
 - ex) C:\₩에 들어가기 위해 아래 명령어 입력
 - `cd C:`
- 폴더 생성
 - ex) GitDirectory1 폴더를 만들기 위해 아래 명령어 입력
 - `mkdir GitDirectory1`
- 해당 폴더로 이동
 - `cd GitDirectory1`
 - 그러면 아래와 같이 비어있는 GitDirectory1으로 들어온 상황이 됨

```
for@Eunsang MINGW64 /c/GitDirectory1
$ |
```

로컬저장소 생성

- 아래 명령어로 git 저장소 생성
 - git init
- "Initialized empty Git repository"라는 텍스트가 나오면 성공

```
for@Eunsang MINGW64 /c/GitDirectory1
$ git init
Initialized empty Git repository in C:/GitDirectory1/.git/
```

- ls -all 명령어로 .git 폴더가 만들어졌는지 확인

```
for@Eunsang MINGW64 /c/GitDirectory1 (master)
$ ls -all
total 12
drwxr-xr-x 1 for 197121 0 Mar 14 16:03 ./
drwxr-xr-x 1 for 197121 0 Mar 14 16:00 ../
drwxr-xr-x 1 for 197121 0 Mar 14 16:03 .git/
```

로컬저장소 생성

- 아직까지는 특별히 변경사항 없음.
- git status로 상태를 보면 다음과 같음

```
for@Eunsang MINGW64 /c/GitDirectory1 (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

- git status -s 명령어는 git status보다 짧게 요약해서 상태를 보여주는 명령어

용어정리

- 워킹트리
 - 로컬저장소(.git)이 있는 현재 디렉토리
 - 즉, 일반적인 작업 디렉토리
 - ex) C:\GitDirectory1
 - 워킹트리, 워킹 디렉토리, 작업 디렉토리, 작업 폴더 모두 같은 뜻으로 사용됨
 - 일반적으로 사용자가 파일과 하위폴더 만들고 작업 결과물을 저장하는 곳
 - 정확하게는 작업 폴더에서 .git 폴더(로컬저장소)를 뺀 나머지 부분이 워킹트리임

용어정리

- 로컬저장소
 - .git은 git으로 생성한 버전들의 정보와 원격저장소 주소 등이 들어있는 숨겨진 폴더
 - 이 .git 폴더를 로컬저장소라 부름
 - 커밋, 커밋을 구성하는 객체, 스테이지가 모두 이 폴더에 저장됨

용어정리

- 원격저장소
 - 로컬저장소를 업로드하는 곳
 - GitHub 저장소가 바로 원격저장소
- Git 저장소
 - git 명령으로 관리할 수 있는 폴더 전체를 일반적으로 git 프로젝트 혹은 git 저장소라 부르며 정의가 모호함
 - 공식 문서에서는 로컬저장소와 git 저장소를 같은 뜻으로 사용

Config

- config란
 - git을 사용하기 위해 옵션을 설정해주는 것
 - 옵션은 지역 옵션, 전역 옵션, 시스템 환경 옵션 세 종류가 있으며 우선순위는 지역 옵션 > 전역 옵션 > 시스템 환경 옵션
- 지역옵션 (local)
 - 현재 git 저장소에만 유효한 옵션
- 전역옵션 (global)
 - 현재 사용자를 위한 옵션
- 시스템 환경 옵션 (system)
 - PC 전체의 사용자를 위한 옵션

Config

```
git config --global <옵션명>
```

지정한 전역 옵션의 내용을 살펴봅니다.

```
git config --global <옵션명> <새로운 값>
```

지정한 전역 옵션의 값을 새로 설정합니다.

```
git config --global --unset <옵션명>
```

지정한 전역 옵션을 삭제합니다.

```
git config --local <옵션명>
```

지정한 지역 옵션의 내용을 살펴봅니다.

```
git config --local <옵션명> <새로운 값>
```

지정한 지역 옵션의 값을 새로 설정합니다.

```
git config --local --unset <옵션명>
```

지정한 지역 옵션의 값을 삭제합니다.

```
git config --system <옵션명>
```

지정한 시스템 옵션의 내용을 살펴봅니다.

```
git config --system <옵션명> <값>
```

지정한 시스템 옵션의 값을 새로 설정합니다.

```
git config --system --unset <옵션명> <값>
```

지정한 시스템 옵션의 값을 삭제합니다.

```
git config --list
```

현재 프로젝트의 모든 옵션을 살펴봅니다.

Config

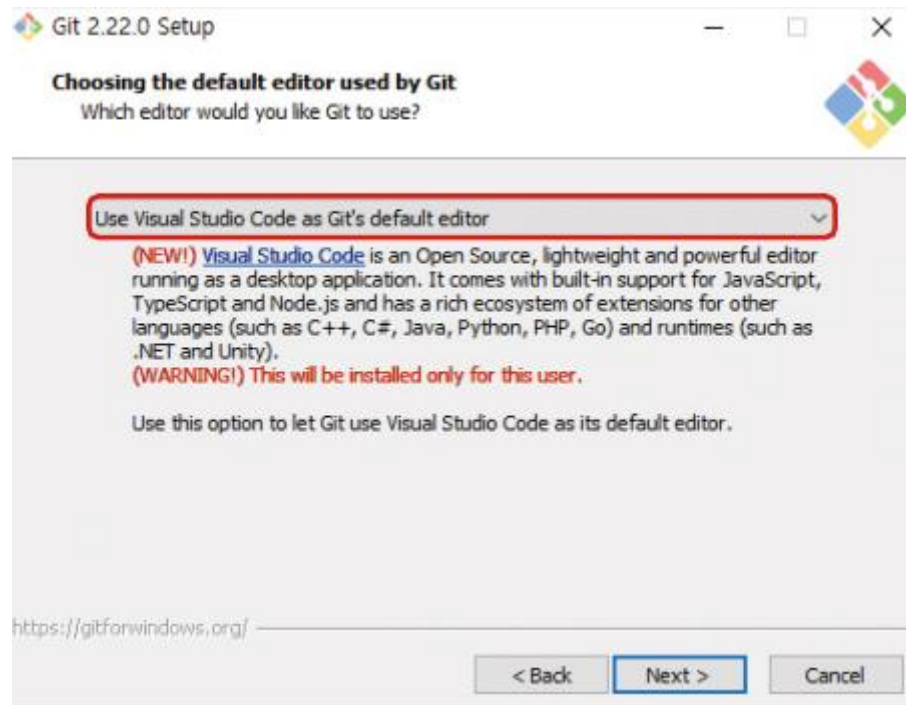
- 로컬저장소에서 커밋을 하기 위해서는 내 정보를 등록해야함
- `git config --global user.email "이메일 주소"`
- `git config --global user.name "유저이름"`
- ex)
 - `git config --global user.email "eslee3209@sejong.ac.kr"`
 - `git config --global user.name "eslee3209"`

기본 에디터

- Git의 기본 에디터를 vscode로 설정 가능
- git 기본 에디터 확인
 - `git config core.editor`
 - `git config --global core.editor`
 - `git config --system core.editor`

기본 에디터

- git 기본 에디터 설정 방법
 - git config로 설정하는 것도 가능하지만 git을 재설치해도 됨
 - 아래와 같이 기본 에디터 선택이 나오면 vscode로 선택



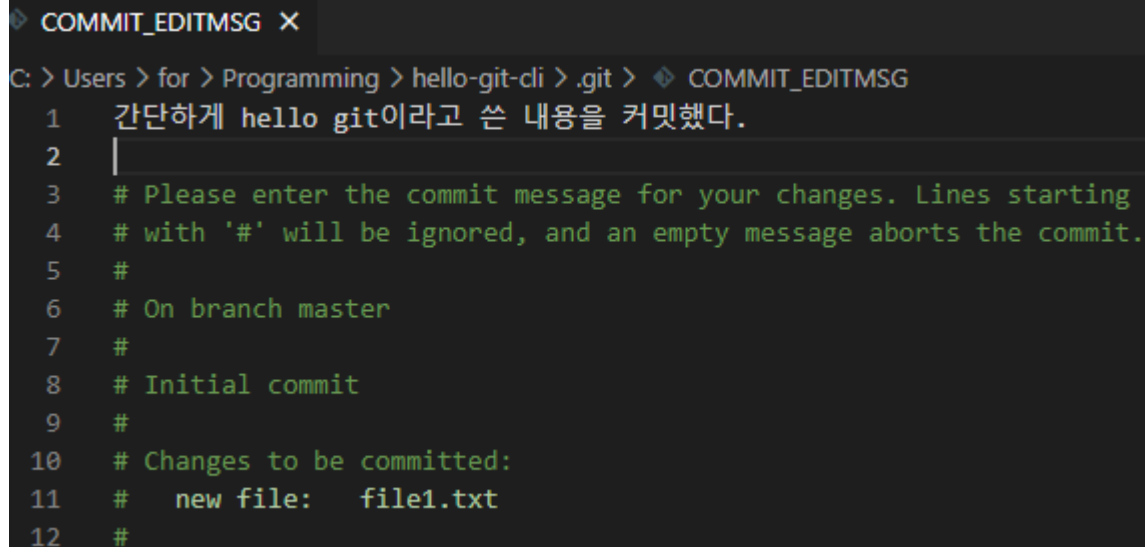
스테이징

- 스테이징(staging)
 - 아래 명령어로 커밋에 파일을 추가할 수 있음. 즉, 스테이징(staging) 가능
 - `git add [파일명]`

커밋

- 커밋

- 아래 명령어로 커밋 가능
- `git commit -m "커밋 설명"`
- 만약 그냥 `git commit` 이라고만 하면 커밋 메시지를 적을 수 있는 창이 뜬(ex. vscode 창)
- 커밋 메시지를 잘 적고 저장. 이 때, 첫 줄과 둘째 줄 사이는 반드시 한 줄 비워야함
- 첫 줄에는 작업 내용의 요약, 다음 줄에는 자세하게 작업내용을 기록



```
COMMIT_EDITMSG X
C: > Users > for > Programming > hello-git-ci > .git > COMMIT_EDITMSG
1  간단하게 hello git이라고 쓴 내용을 커밋했다.
2  |
3  # Please enter the commit message for your changes. Lines starting
4  # with '#' will be ignored, and an empty message aborts the commit.
5  #
6  # On branch master
7  #
8  # Initial commit
9  #
10 # Changes to be committed:
11 #   new file:   file1.txt
12 #
```

스테이징, 커밋 실습

- GitDirectory1 폴더에서 GitFile1.txt 생성
- 파일 내용은 Contents1이라고 작성
- 다음 명령어 실행
 - git add GitFile1.txt
 - git commit -m "Commit1"
- 그러면 커밋이 성공적으로 됨

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit 5f0c3494e77d705a806b13295fbb934df9a41880 (HEAD -> master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

언스태이징

- 언스태이징
 - 스테이징을 취소하고 싶은 경우 다음 명령어로 가능
 - `git reset <파일명>`

언스태이징 실습

- 새로운 파일 GitFile2.txt를 생성하고 내용을 Contents1이라고 함
- 그리고 git status로 상태를 보면 untracked 상태가 됨

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    GitFile2.txt

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

- 이 때, git add GitFile2.txt를 통해 스테이징을 하면 다음과 같이 됨

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git add GitFile2.txt
warning: in the working copy of 'GitFile2.txt', LF will be replaced by CRLF the next time
Git touches it

USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   GitFile2.txt
```

언스태이징 실습

- 방금 스테이징한 파일 GitFile2.txt를 언스태이징 하기 위해 다음 명령어 입력
 - git reset GitFile2.txt
- 그러면 다음과 같이 다시 untracked 상태가 됨. 즉, 성공적으로 언스태이징이 됨

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git reset GitFile2.txt

USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    GitFile2.txt

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

커밋 취소

- 방금 한 커밋을 취소하려면 다음 명령어로 가능
 - `git reset --hard HEAD~`
- 임의의 원하는 커밋으로 이동하려면 다음과 같이 가능
 - `git reset --hard [이동할 커밋 체크섬]`

커밋 취소 실습

- 현재, GitFile1.txt만 생성된 상황(GitFile2.txt는 없음)
- 두 개의 커밋 Commit1, Commit2 가 있는 상황이라 가정

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit 14ec0800bf0583ae7bbe4b9d61e961b46fbbabe1 (HEAD -> master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:42:24 2023 +0900

    Commit2

commit 5f0c3494e77d705a806b13295fbb934df9a41880
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

- 이 때, 다음 명령어를 수행하면 커밋 취소 가능
 - git reset --hard HEAD~

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git reset --hard HEAD~
HEAD is now at 5f0c349 Commit1

USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit 5f0c3494e77d705a806b13295fbb934df9a41880 (HEAD -> master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

커밋 히스토리 확인

- git log
 - 이 명령어로 현재 브랜치의 커밋 이력을 볼 수 있음

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit c6062eea3000a6ff3ccbef0d29524a39f69527c8 (HEAD -> master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 22:49:24 2023 +0900

    Commit2

commit 5f0c3494e77d705a806b13295fbb934df9a41880
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

- git log --oneline
 - 간단히 커밋 해시와 제목만 봄

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log --oneline
c6062ee (HEAD -> master) Commit2
5f0c349 Commit1
```

커밋 히스토리 확인

- `git log -n<숫자>`
 - 전체 커밋 중에서 최신 n개의 커밋만 봄
 - ex) `git log -n3`
- `git log --oneline -n<숫자>`
 - 전체 커밋 중에서 최신 n개의 커밋만 간단히 봄
 - ex) `git log --oneline -n5`
- `git log --oneline --graph --decorate`
 - HEAD와 관련된 커밋들을 조금 더 자세하게 보고 싶을 때
 - graph: 커밋 옆에 브랜치의 흐름을 그래프로 보여줌
 - decorate: 브랜치와 태그 등의 참조를 간결히 표시
- `git log --oneline --graph --all --decorate`
 - `--all`이 들어가면 모든 브랜치들을 볼 수 있음

커밋 히스토리 확인

- git log
 - commit: 커밋 아이디
 - author: 누가 커밋했는지. 유저이름 및 이메일
 - date: 커밋한 일자
 - 메모: 커밋할 때 메모한 내용

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit c6062eea3000a6ff3ccbef0d29524a39f69527c8 (HEAD -> master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 22:49:24 2023 +0900

    Commit2

commit 5f0c3494e77d705a806b13295fbb934df9a41880
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

도움말

- help
 - git 도움말
 - 다음 형태로 해당 명령어의 도움말을 볼 수 있음
 - git help <명령어>
 - ex)
 - git help status
 - git help commit
 - git help add

Push

- 로컬저장소에 만들었던 커밋들을 원격저장소에 push하기 위해서는 로컬저장소에 원격저장소 주소를 알려주어야함
- 이 때, 비어있는 원격저장소에 대해 등록을 해주어야 이후 커밋할 때 충돌이 생기지 않음
- `git remote add origin <원격저장소 주소>`
 - ex) `git remote add origin https://github.com/eslee3209/GitRepository1.git`

Push 방법1

- 아래와 같은 형태로 push 가능
 - git push [원격저장소명] [브랜치명]
 - ex) git push origin master

Push 방법1 실습

- 로컬저장소에서 커밋이 2개 되고 아직 push는 되지 않은 상황이라 생각
- 다음 명령어 입력
 - git push origin master
- 그러면 정상적으로 push가 완료됨

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit c6062eea3000a6ff3ccbef0d29524a39f69527c8 (HEAD -> master, origin/master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 22:49:24 2023 +0900

    Commit2

commit 5f0c3494e77d705a806b13295fbb934df9a41880
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

Push 방법2

- 매번 원격저장소와 브랜치명을 입력하지 않아도 되는 방법이 있음
- 한번만 다음과 같은 형태로 업스트림을 지정하면서 push를 진행
 - `git push -u origin master` 혹은
 - `git push --set-upstream origin master`
- 이는 로컬저장소의 현재 브랜치와 원격저장소의 브랜치를 연결시켜줌
- 그러면 그 이후부터는 단순히 `git push` 명령 만으로 push가 가능

Push 취소

- 최근 push까지 진행한 커밋을 취소하고 취소한 결과를 원격저장소에도 반영하고자 함
- 이 때, 먼저 reset 명령어를 사용하여 로컬저장소에서 커밋을 취소한 후 다음 명령어로 원격저장소에 push한 결과도 취소할 수 있음
 - `git push -u <원격저장소명> <브랜치명> --force`
 - ex) `git push -u origin master --force`

Push 취소 실습

- 커밋 2개를 push까지 진행한 상황 생각

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit c6062eea3000a6ff3ccbef0d29524a39f69527c8 (HEAD -> master, origin/master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 22:49:24 2023 +0900

    Commit2

commit 5f0c3494e77d705a806b13295fbb934df9a41880
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

- 이 때, 로컬저장소에서 커밋을 취소함
 - git reset --hard HEAD^

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git reset --hard HEAD^
HEAD is now at 5f0c349 Commit1

USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit 5f0c3494e77d705a806b13295fbb934df9a41880 (HEAD -> master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

    Commit1
```

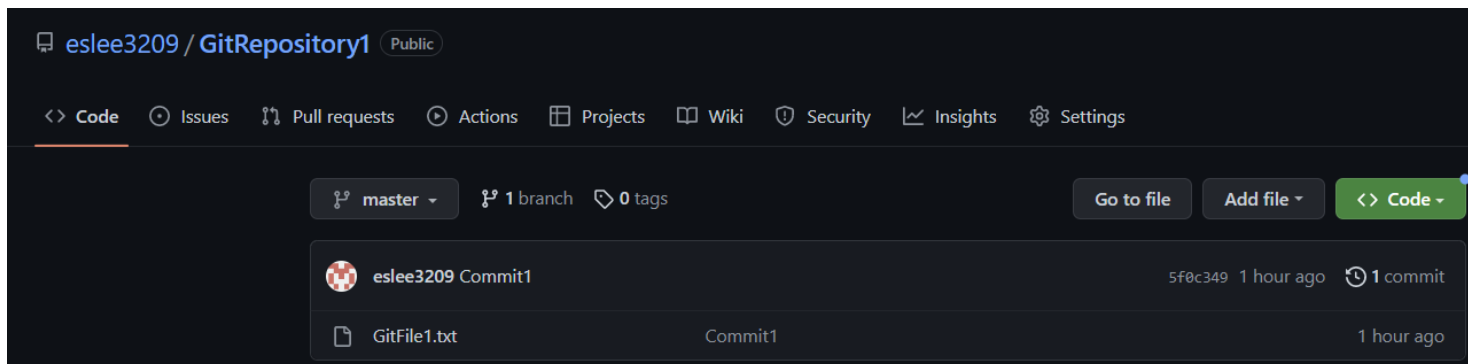
Push 취소 실습

- 다음 명령어로 강제푸시를 진행하여 원격저장소에도 반영시킴
 - `git push -u origin master --force`

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git push -u origin master --force
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/eslee3209/GitRepository1.git
+ c6062ee...5f0c349 master -> master (forced update)
branch 'master' set up to track 'origin/master'.
```

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git log
commit 5f0c3494e77d705a806b13295fbb934df9a41880 (HEAD -> master, origin/master)
Author: eslee3209 <eslee3209@sejong.ac.kr>
Date: Tue Mar 14 21:31:45 2023 +0900

Commit1
```



eslee3209 / GitRepository1 Public

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 1 branch 0 tags Go to file Add file <> Code

eslee3209 Commit1 5f0c349 1 hour ago 1 commit

GitFile1.txt Commit1 1 hour ago

Clone

- Clone(클론)이란 원격저장소의 코드와 버전 전체를 내 컴퓨터로 내려받는 것
- 모든 버전 및 원격저장소 등이 내 로컬저장소에 저장됨
- 아래 명령어로 현재 폴더에 clone 가능
 - `git clone [원격저장소주소]` .
- 다음 명령어로 clone 진행 시 [새 폴더명] 이름의 폴더가 생성되면서 그 안에 내용이 저장됨
 - `git clone [원격저장소주소] [새 폴더명]`

Clone 생성 실습

- 방법1

- 로컬에서 GitDirectory2 폴더를 생성하고 그 안에 들어감
- 아래 명령어 입력
 - git clone <https://github.com/eslee3209/GitRepository1.git> .

- 방법2

- 로컬에서 GitDirectory2 폴더를 만들 위치에서 아래 명령어 입력
 - git clone <https://github.com/eslee3209/GitRepository1.git>
GitDirectory2

Pull

- 원격저장소는 최근 커밋 업데이트 되었지만 로컬 폴더에서는 커밋이 반영되어있는 경우 있음
- 이 경우 pull을 통해 원격저장소의 새 커밋을 로컬저장소에 갱신함
- 사실 `git fetch + git merge`의 합성

Pull 실습

- GitDirectory1에서 GitFile1.txt가 있고 커밋이 1개 된 상태에서 push되었다고 함
- GitDirectory2에서 해당 원격저장소를 clone한 후 커밋을 한 개 추가한 후 push
- 그 다음에 GitDirectory1으로 돌아가면 원격저장소의 커밋이 로컬저장소보다 빠르므로 pull이 필요한 상황이 됨
- 아래 명령어를 입력하면 pull 완료
 - git pull

```
USER@BOOK-1C7BTP524F MINGW64 /c/GitDirectory1 (master)
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 227 bytes | 56.00 KiB/s, done.
From https://github.com/eslee3209/GitRepository1
   5f0c349..c5dce7b  master      -> origin/master
Updating 5f0c349..c5dce7b
Fast-forward
   GitFile1.txt | 2 +
1 file changed, 1 insertion(+), 1 deletion(-)
```

명령어 요약

- 1. 로컬저장소 생성 방법
 - 폴더A를 로컬저장소로 만듦
 - 터미널에서 폴더A로 들어감
 - `git init`
- 2. 스테이징
 - 파일 A 스테이징
 - `git add A`
 - ex) `git add GitFile1.txt`
 - 모든 파일 스테이징
 - `git add .`

명령어 요약

- 3. 언스태이징
 - 파일A 언스태이징
 - `git reset A`
- 4. 커밋
 - 커밋A의 파일 상태가 되도록 작업/스태이징/커밋(메시지는 B)
 - 커밋A의 파일상태가 되도록 VScode로 파일 작업함
 - `git add .`
 - `git commit -m "B"`

명령어 요약

- 5. 커밋 취소
 - 브랜치A의 최근 커밋을 커밋B로 Hard모드로 돌림
 - `git checkout A`
 - `git reset --hard [커밋B체크섬]`
 - 브랜치A의 가장 최신 커밋을 Hard모드로 취소함
 - `git checkout A`
 - `git reset --hard HEAD~`
- 6. 원격저장소 등록
 - 원격저장소(주소A)를 로컬저장소B에 C라는 닉네임으로 등록
 - 터미널에서 해당 디렉토리 B에 들어감
 - `git remote add C A`
 - ex) `git remote add origin`
<https://github.com/eslee3209/GitRepository1.git>

명령어 요약

- 7. Push
 - 브랜치 A를 원격저장소에 push
 - `git push origin A`
 - ex) `git push origin master`
- 8. 강제 Push
 - 브랜치 A를 원격저장소에 강제 push
 - `git push -u origin A --force`
 - ex) `git push -u origin A --force`

명령어 요약

- 9. 클론
 - 원격저장소 A(주소)를 폴더 B에 클론함
 - 터미널에서 폴더 B에 들어감(없으면 빈 폴더 생성)
 - git clone A .
 - ex) git clone <https://github.com/eslee3209/GitRepository1.git> .
- 10. pull
 - 브랜치 A를 pull
 - git pull origin A
 - (혹은 브랜치 A로 체크아웃한 후 git pull)

명령어 요약

- 11. fetch
 - 브랜치A를 fetch함
 - `git fetch origin A`
 - ex) `git fetch origin master`
 - 원격저장소의 모든 커밋/브랜치/태그를 fetch함
 - `git fetch`