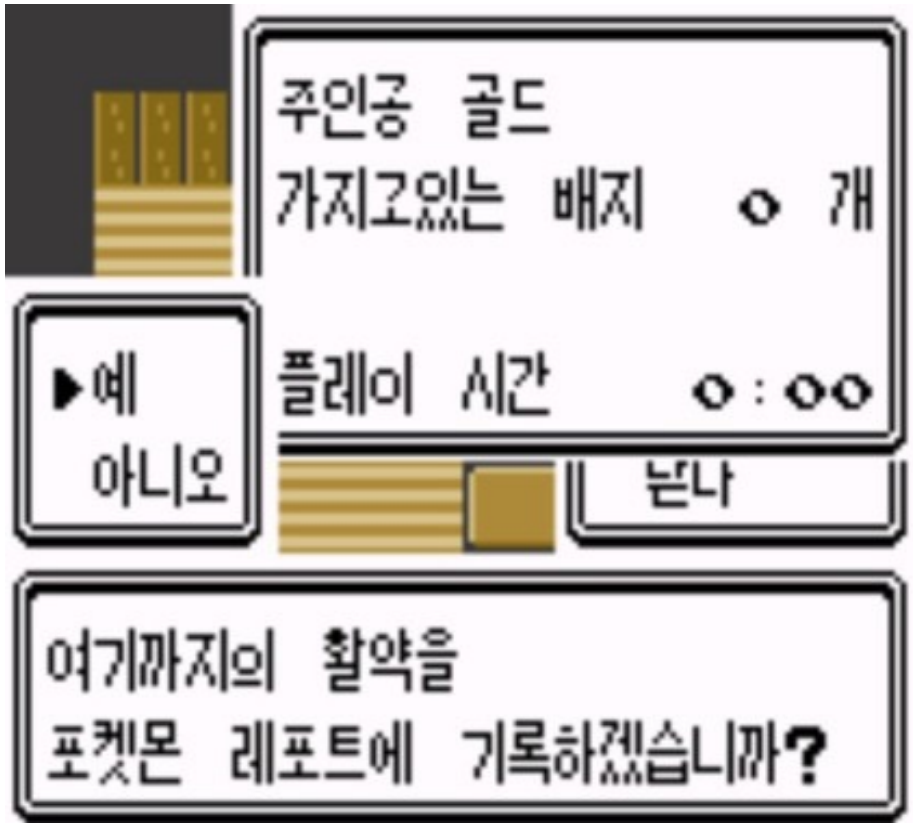
git + GitHub

On your mark, git set, go

강사 최인규

만약에 Git, GitHub가 없다면?

- 게임을 켜면 '이어하기'가 없고, '새로하기'만 있다!?
- 마치 SAVE 기능 없이 게임을 하는 것!
- Git과 GitHub는 개발을 이어서 계속할 수 있게 한다.

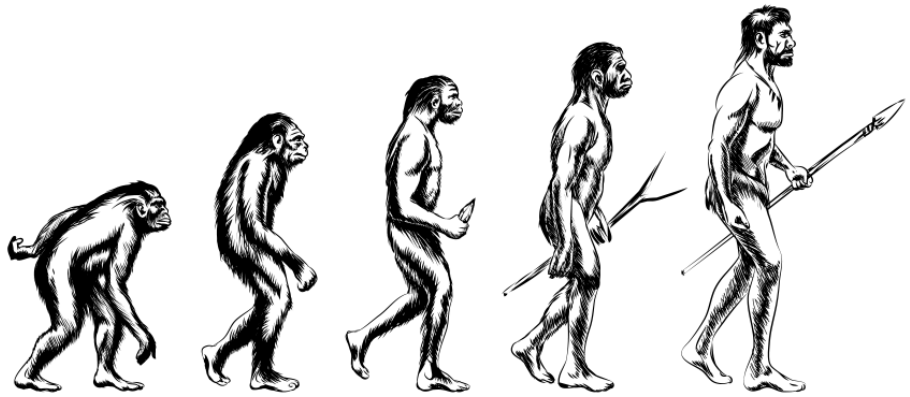


Git과 버전관리

- 버전이란?
  - 유의미한 변화가 결과물로 나온 것 또는 이를 구분하는 지점

TIP

패치 : 시급한 오류 해결이나 비교적 규모가 작은 버전



Windows 1

Windows 3.1

Windows 95

Windows XP

Windows Vista

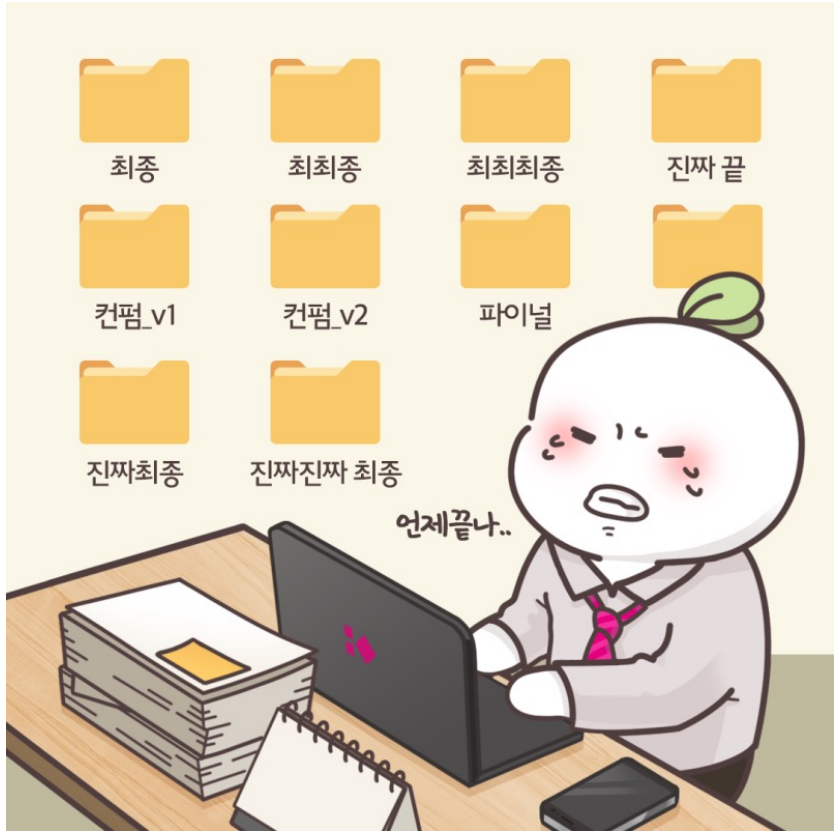
Windows 7

Windows 8

Windows 10

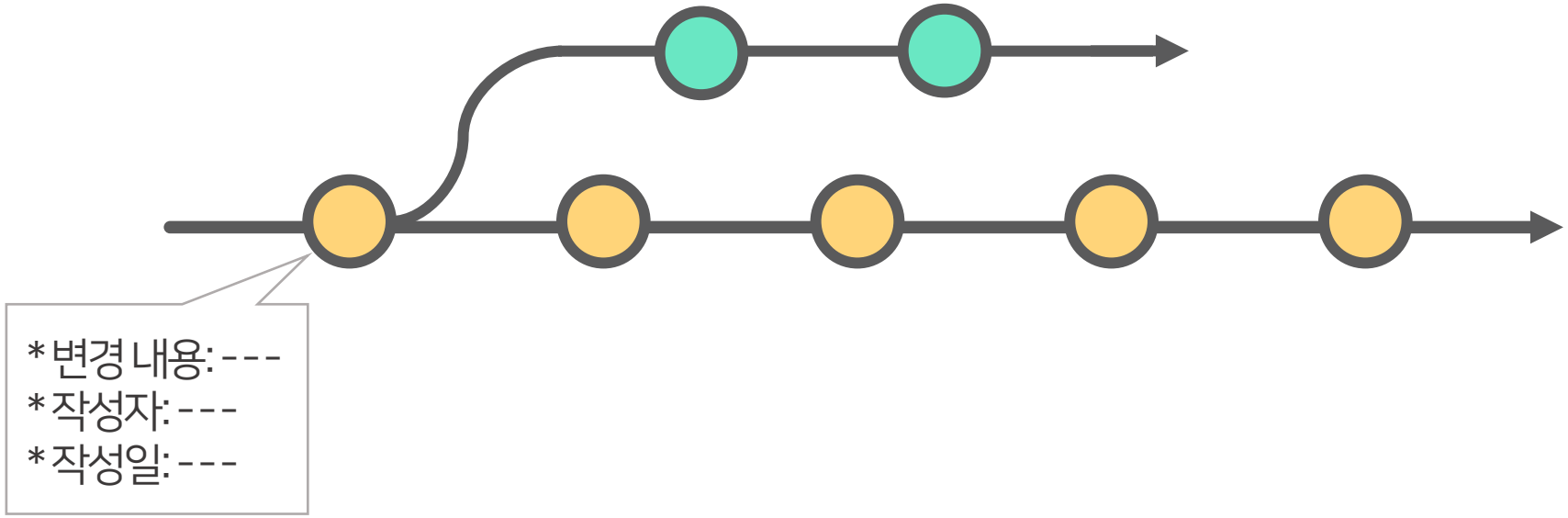
Windows 11

Git과 버전관리



Git과 버전관리

- 버전 관리 시스템을 이용하면, 버전 관리는 물론 변경점 관리, 백업 및 복구, 협업에 큰 이점이 있다.
  - 언제, 누가, 어떤 부분을, 어떻게 변경했는지 확인할 수 있다.
  - 특정 변경지점에서부터 다양한 버전으로 개발할 수 있다.
  - 수정된 내용을 쉽게 공유할 수 있다.
  - 특정 시점으로 돌아갈 수 있다.



Git과 버전관리

버전 관리는

"누가, 언제, 어떻게 변경했는지 **변경 내역**을 **기억**하고,

필요하다면 **특정 시점의 버전으로 되돌릴 수 있다**는 특징을 바탕으로,

협업하는 과정에서 코드를 **쉽게** 나누고 합치며 **개발할 수 있도록 도와주는 것**"이다.

Git과 버전관리

- 만약 버전 관리 시스템이 없다면?

    1. 각자 개발을 진행한다. (정해진 날짜[Merge Day]에 파일을 주고 받아 합치기로 약속)
    2. 혹시 합치면서 에러가 발생할 수 있으니, 미리 백업본을 만들어 둔다.
    3. 팀원이 내가 작업한 파일을 고친 경우에는 꼭 말해달라고 한다.
    4. 코드를 합친 후에는 내 컴퓨터에 반영을 시켜줘야 한다.

- 변경 내역을 일일이 체크하는 것도 번거롭고, 백업을 직접하지 않으면 되돌리는 것도 쉽지 않아 협업이 어려워진다.

Git과 버전관리

- 만약 버전 관리 시스템이 있다면?

원할 때 언제든지 코드를 간편하게 합칠 수 있고, 백업도 쉽게 가능해지게 된다. 

Git과 버전관리

- Git
  - 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 분산 버전 관리 시스템

- 소프트웨어 개발에서 소스 코드 관리에 주로 사용되지만
  어떠한 파일 집합이든 변경사항을 지속적으로 추적하기 위해 사용될 수 있다.

깃

소프트웨어

깃은 컴퓨터 파일의 변경사항을 추적하고 여러 명의 사용자들 간에 해당 파일들의 작업을 조율하기 위한 스냅샷 스트림 기반의 분산 버전 관리 시스템이다. 또는 이러한 명령어를 가리킨다. 위키백과

개발: 리누스 토르발스, Junio C Hamano

프로그래밍 언어: 파이썬, C, C++, 셸 스크립트, 펄, Tcl

개발자: 주니오 하마노(Junio Hamano), 리누스 토르발스 등

라이선스: GNU 일반 공중 사용 허가서 v2

안정화 버전: 2.43.0 / 2023년 11월 20일

언어: 영어

종류: 버전 관리

Git과 버전관리

- Git 다운로드 받기

https://git-scm.com/에서 다운로드 진행

Git과 버전관리

페이지 1 ~ 3을 작성해 'Alice ver.1' 저장

'Alice ver.1'을 다운로드 받은 뒤,
이어서 페이지 4 ~ 6을 작성해 'Bob ver.1' 저장

페이지 2를 수정한 후, 'Alice ver.2' 저장

'Alice ver.2'을 다운로드 받은 뒤,
'Bob ver.1'과 차이점(diff)을 반영한 'Bob ver.2' 저장

Git과 버전관리

- Git은 코딩을 하면서 원하는 시점마다 깃발을 꽂고, 자유롭게 돌아다닐 수 있게 해준다.

- Git은 저장할 공간만 있다면 어디서나 사용이 가능하다.

- Git은 CLI(Command Line Interface) 방식과 GUI(Graphic User Interface) 방식 모두 활용 가능하다.

TIP

컴퓨터와 상호작용하는 2가지 방식
CLI : 텍스트 기반 인터페이스 (키보드를 통해 명령어 입력)
GUI : 그래픽 기반 인터페이스 (마우스를 통해 그래픽 요소 클릭)

Git과 버전관리

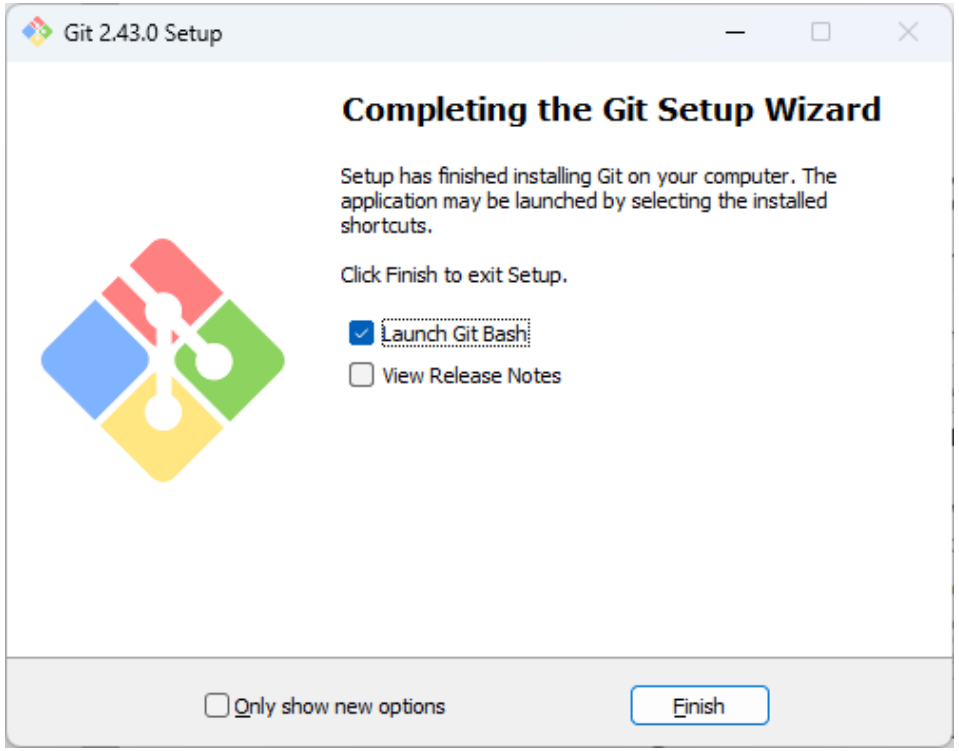

Git 2.43.0 Setup
Information
Please read the following important information before continuing.
When you are ready to continue with Setup, click Next.
GNU General Public License
Version 2, June 1991
Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.
Preamble
The licenses for most software are designed to take away your
freedom to share and change it. By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change
https://gitforwindows.org/
Only show new options
Next
Cancel

Git과 버전관리

- C:₩Program Files에 git 폴더를 생성하고 설치
- 바탕화면 아이콘 생성 체크, Git bash 체크, LFS 체크, 기본 텍스트 에디터 체크, Bash 연결 체크, 기본 터미널에 추가 체크
- 시작 메뉴 폴더에 생성
- 기본 편집기 선택
- 기본 브랜치 이름 (main)
- 다른 쉘에서도 git 사용
- OpenSSH 그대로 사용
- OPENSSL 그대로 사용
- 개행방식 조정
- 기본 터미널 에뮬레이터 MinTTY
- git pull 기본 동작 설정
- 자격증명도우미 사용
- 시스템 캐싱 통한 성능 향상
- 실험실 옵션

Git과 버전관리

- 이제 bash 터미널을 이용할 수 있다.



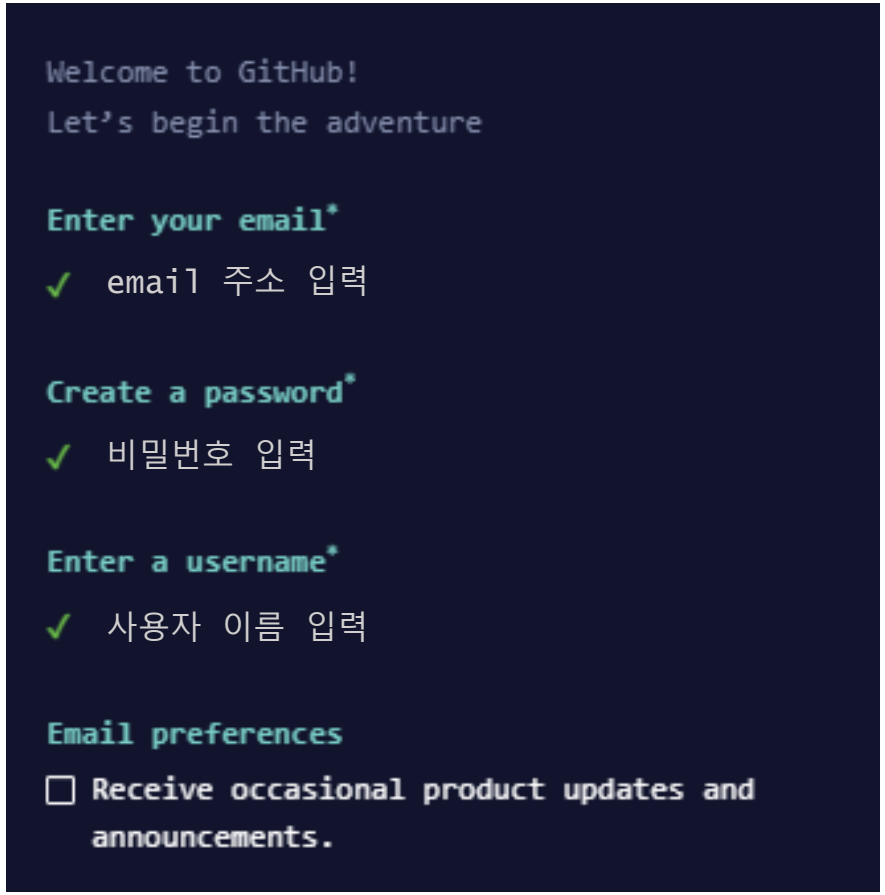
Git과 버전관리

- GitHub 회원가입 진행 (https://github.com/signup)
- 여기서 사용되는 이메일 주소와 사용자 이름은 고유한 값으로 git, gitHub에서 쓰이게 된다.

TI

GitHub의 사용자 이름은 개발자로서의 정체성을 나타내고,
온라인 포트폴리오로서 중요한 역할을 한다.

(inkyu, DevInkyu, InkyuCoder, Inkyu2000, inkyu_developer)



Git과 버전관리

```
Username@DESKTOP MINGW64 ~
$ git config --global user.name "사용자이름"

Username@DESKTOP MINGW64 ~
$ git config --global user.email "이메일주소"

Username@DESKTOP MINGW64 ~
$ git config --global --list
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
user.name=사용자이름
user.email=이메일주소
```

bash 쉘의 기초 명령어

1. 자동완성 : TAB 키 활용

2. 현재 작업 중인 경로 확인 : pwd (Print Working Directory)

3. 현재 디렉토리 내부 리스트 출력 : ls (List)
   ① -l 옵션 : 자세히 보기
   ② -a 옵션 : 숨김파일 포함 보기

4. 파일 내용 확인 : cat 파일명.확장자

5. 작업 중인 경로 변경 : cd 경로 (Change Directory)
   ① .. : 상위 폴더
   ② . : 현재 위치
   ③ ~ : 홈 경로

6. 폴더 생성 : mkdir 폴더명 (Make Directory)

bash 쉘의 기초 명령어

7. 삭제 : rm (Remove)
   ① 파일 삭제 : rm 파일명
   ② 강제 삭제 : rm -f 파일명
   ③ 폴더 삭제 : rm -r 폴더명 [-r 옵션을 사용해 재귀적으로 하위 파일까지 삭제]

8. 파일 생성 : touch 파일명.확장자

9. 복사 : cp (Copy)
   ① 파일 복사 : cp 해당파일명 복사할위치경로
   ② 폴더 복사 : cp -r 해당폴더명 복사할위치경로 [-r 옵션을 사용해 재귀적으로 하위 파일까지 복사]

10. 이동 : mv (Move)
    ① mv 파일명 이동할위치경로
    ② mv 폴더명 이동할위치경로

Git과 버전관리

```
Username@DESKTOP MINGW64 ~
$ pwd
/c/Users/Username

Username@DESKTOP MINGW64 ~
$ mkdir my_story

Username@DESKTOP MINGW64 ~
$ cd my_story/

Username@DESKTOP MINGW64 ~/my_story
$ pwd
/c/Users/Username/my_story

Username@DESKTOP MINGW64 ~/my_story
$ start .
```

Git & GitHub 기초 (CLI)

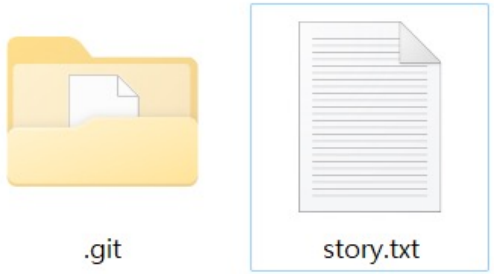
- git init : 깃 저장소(git repository)를 생성 [초기화]
- 다음 명령 입력부터는 기본 브랜치 이름이 나타난다. 이를 통해 git 저장소가 설정된 경로임을 인지할 수 있다.

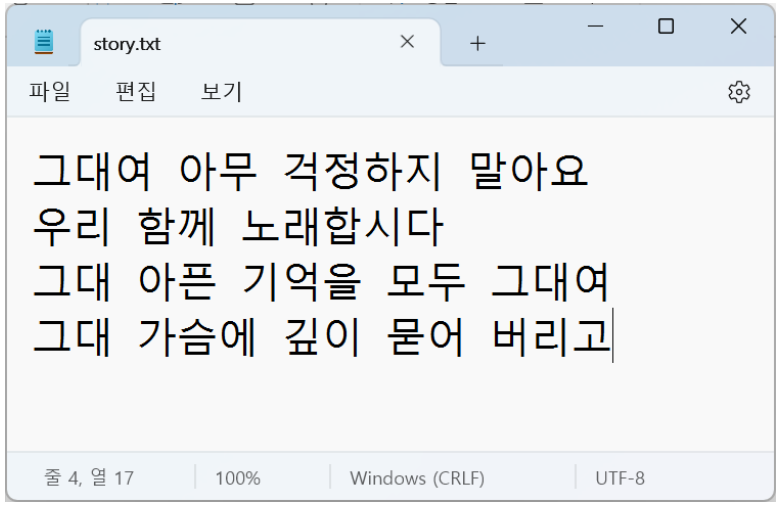
```
Username@DESKTOP MINGW64 ~/my_story
$ git init
Initialized empty Git repository in C:/Users/Administrator/my_story/.git/

Username@DESKTOP MINGW64 ~/my_story (main)
$ ls -a
./  ../  .git/
```

- .git은 "로컬 저장소", 내가 만든 버전의 정보, 원격 저장소의 주소 등을 저장한다.
- .git이 있는 경로 아래에 새로운 .git이 존재하면 안된다.

Git & GitHub 기초 (CLI)

- 해당 폴더 안에서 story.txt를 생성하고, 내용 작성 및 저장





Git & GitHub 기초 (CLI)

- git status : 현재 git repository의 상태를 확인
  - No commits yet : 아직 어떠한 커밋(저장)도 없는 상태
  - Untracked files : 추적되고 있지 않는 파일이 존재

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ ls -a
./  ../  .git/ story.txt

Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch master

No commits yet

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

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

Git & GitHub 기초 (CLI)

- git add 파일명 : 특정 파일을 스테이징 영역(staging area)에 추가한다.
  - staging area : 변경된 내용이 저장소에 기록되기 전에 대기하는 장소
  - git rm --cached <file> : 스테이징 영역에 추가한 파일에 대해서 스테이징 취소

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git add story.txt

Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   story.txt

Username@DESKTOP MINGW64 ~/my_story (main)
$ git rm --cached story.txt
```

Git & GitHub 기초 (CLI)

- git add . : 현재 경로에서 스테이징이 가능한 모든 파일을 스테이징 영역(staging area)에 추가한다.
- git commit -m "커밋 메시지" : git에 변경사항을 기록
  - nothing to commit, working tree clean : 커밋 이후 변경 사항이 없다는 것을 의미

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git add .

Username@DESKTOP MINGW64 ~/my_story (main)
$ git commit -m "first commit"
[master (root-commit) 290b8a8] first commit
 1 file changed, 4 insertions(+)
 create mode 100644 story.txt

Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch master
nothing to commit, working tree clean
```

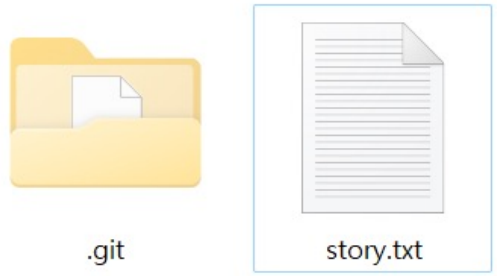
Git & GitHub 기초 (CLI)

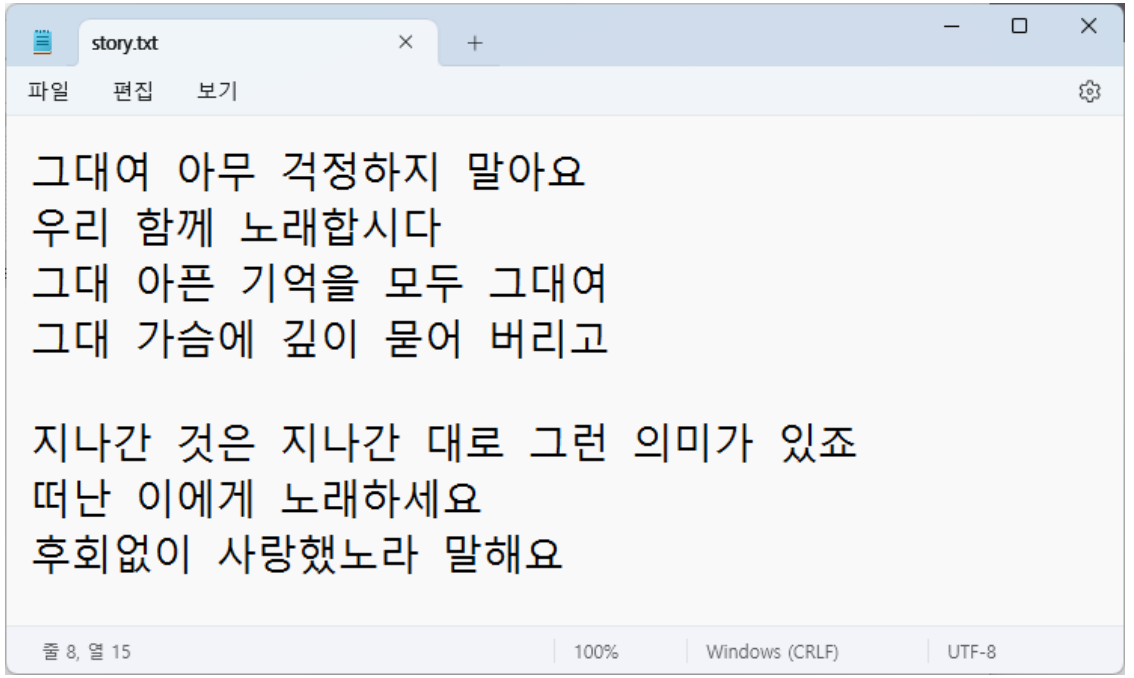
- git log : 커밋 기록 모두 확인
  - commit 11657~~~ : 커밋 식별 ID값
  - Author : 커밋 생성자
  - Date : 커밋 일자
  - 커밋 메시지

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git log
commit 11657497ead0802191e012d8c6497a2d9d1cd96c (HEAD -> master)
Author: 사용자이름 <이메일주소>
Date:   Thu Dec 26 16:37:26 2023 +0900

    first commit
```

Git & GitHub 기초 (CLI)

- story.txt를 수정하고 저장





Git & GitHub 기초 (CLI)

- 추적 중인 파일(story.txt)에 변경사항이 있는 것을 감지
  - git add를 이용해 스테이징을 할 수 있다.
  - git restore을 이용해 파일의 변경 내용을 가장 최근 커밋했던 때의 내용으로 복구할 수 있다.

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   story.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Git & GitHub 기초 (CLI)

- git diff : 파일 내용의 변화를 확인할 수 있다.
  - q를 입력하면 다음 명령어 입력 가능

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git diff
diff --git a/story.txt b/story.txt
index 016a23c..c4982ea 100644
--- a/story.txt
+++ b/story.txt
@@ -2,3 +2,7 @@
 우리 함께 노래합시다
 그대 아픈 기억들 모두 그대여
 그대 가슴에 깊이 묻어 버리고
+
+지나간 것은 지나간 대로 그런 의미가 있죠
+떠난 이에게 노래하세요
+후회없이 사랑했노라 말해요
```

Git & GitHub 기초 (CLI)

- git commt -am "커밋 메시지" : tracked files를 스테이징하고, 커밋까지 한 번에 진행 가능

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git commit -am "second commit"
[master bd295f3] second commit
 1 file changed, 5 insertions(+), 1 deletion(-)
```

Git & GitHub 기초 (CLI)

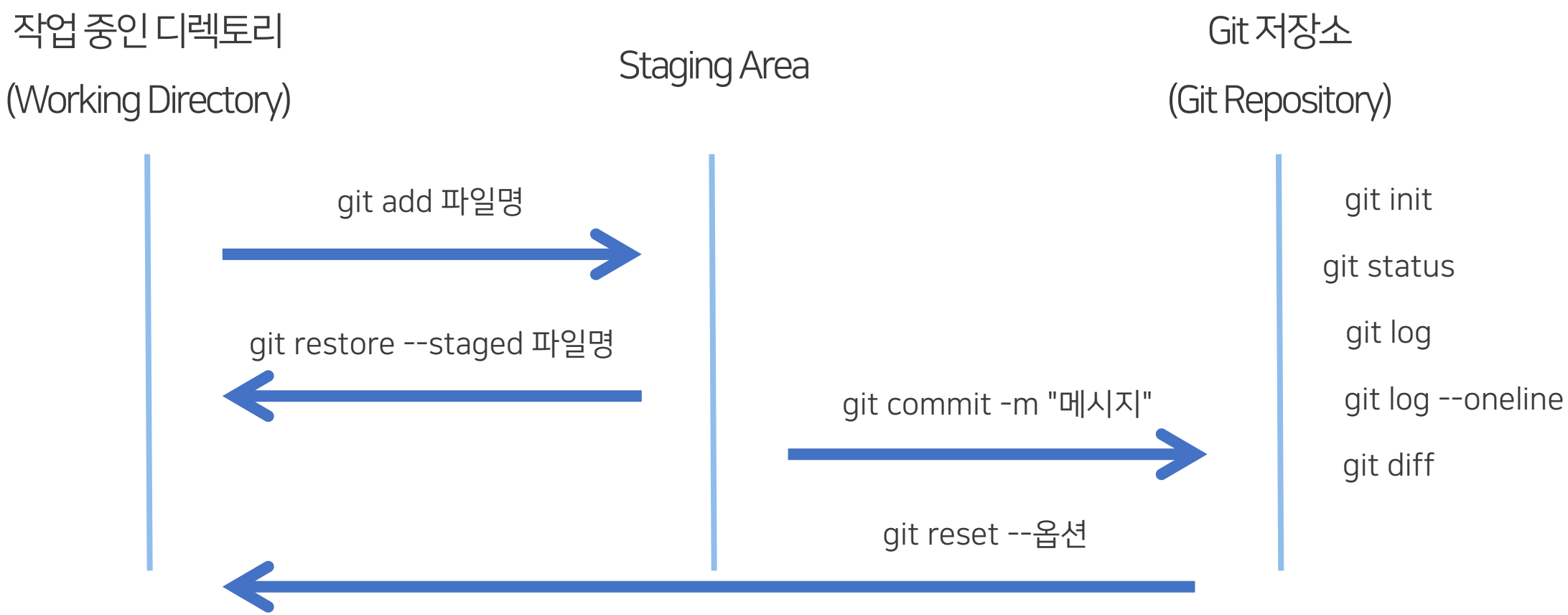
- git reset : 이전 commit으로 되돌린다. (다른 사람과 코드 공유가 되어 있지 않은 상태에서 사용)

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git reset --옵션 커밋ID
```

종류	설명
git reset --soft 커밋ID	staging 상태는 유지, 파일 변경도 유지 [commit 직전]
git reset --mixed 커밋ID	unstaging 상태, 파일 변경은 유지 [add 직전]
git reset --hard 커밋ID	파일 변경도 되돌린다 [이전 commit 직후]

Git & GitHub 기초 (CLI)

- 지금까지 학습한 명령어는 아래와 같다.



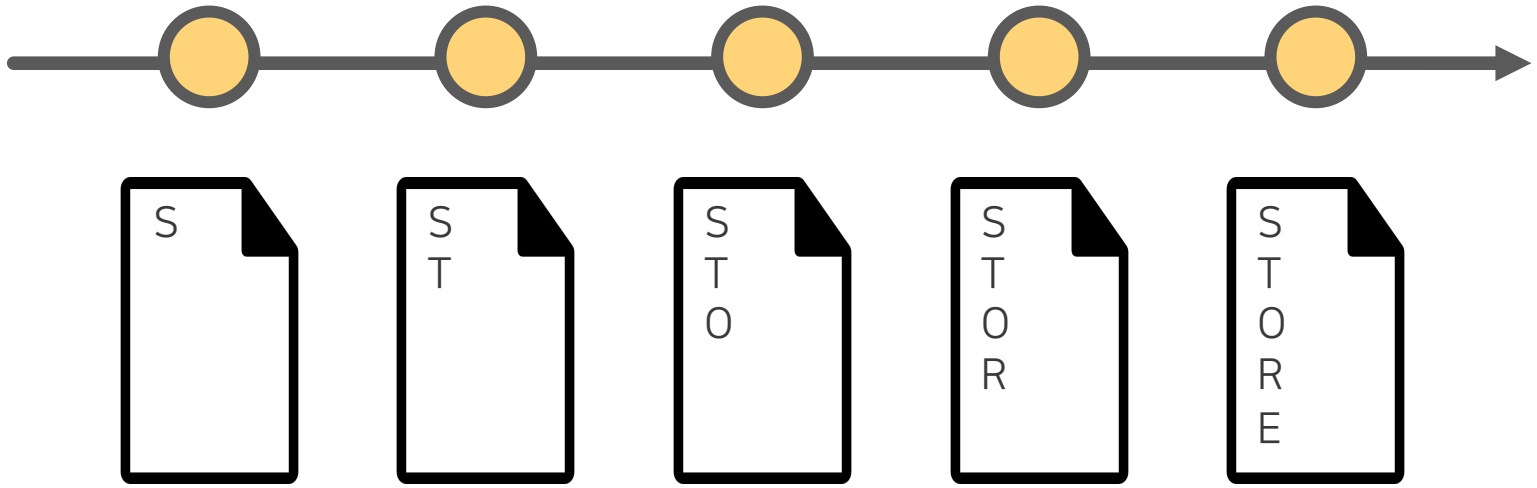
Git & GitHub 기초 (CLI) [실습]

- 앞서 진행한 명령어를 이용하여
  - word.txt 파일을 생성하고, 커밋을 진행 [commit message : add word.txt]
  - word.txt에 S를 입력한 후 저장하고, 커밋을 진행 [commit message : edit S]
  - word.txt에 T를 입력한 후 저장하고, 커밋을 진행 [commit message : edit T]
  - word.txt에 O를 입력한 후 저장하고, 커밋을 진행 [commit message : edit O]
  - word.txt에 R를 입력한 후 저장하고, 커밋을 진행 [commit message : edit R]
  - word.txt에 E를 입력한 후 저장하고, 커밋을 진행 [commit message : edit E]

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git log --oneline
0681292 (HEAD -> master) edit E
8b4a3a6 edit R
df51dbc edit O
e288f45 edit T
65fd425 edit S
s340d23 add word.txt
bd295f3 second commit
1165749 first commit
```

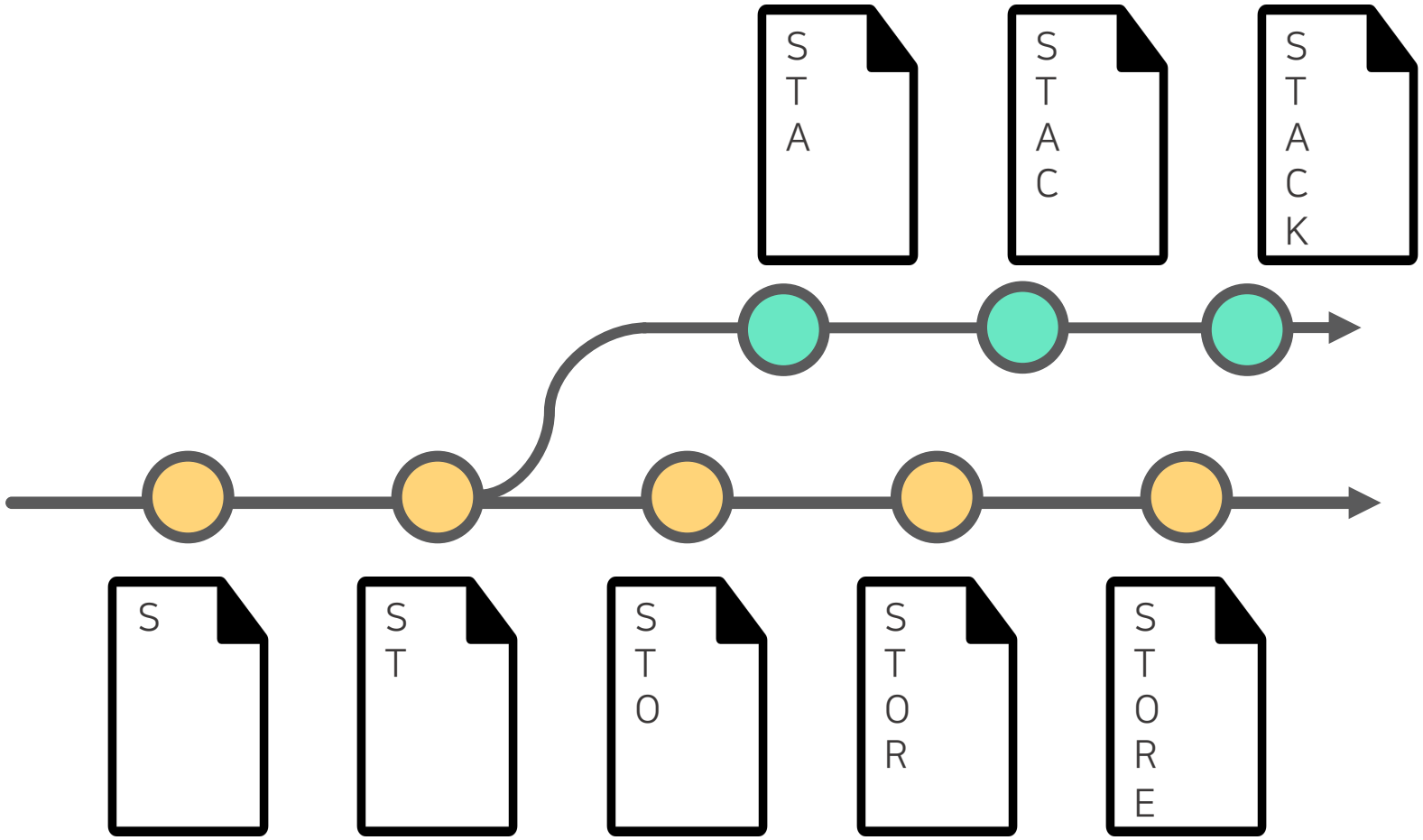
Git & GitHub 기초 (CLI) [실습]

- main 브랜치에 다섯개의 커밋이 추가되었고, 변경된 기록 또한 확인할 수 있게 되었다.
- 이제 ST까지 작성되어 있는 두번째 커밋[edit T]를 분기점으로 하여, STACK으로 수정해보겠다.
- 다만, STORE라는 내용도 가치가 있기 때문에 남겨둘 필요가 있다.
  즉, git reset 명령을 이용할 수 없게 되었다.



Git & GitHub 기초 (CLI) [실습]

- 이 때는 git reset이 아닌 git checkout을 통해 해당 분기점으로 돌아갈 수 있다.



Git & GitHub 기초 (CLI) [실습]

- git checkout <커밋ID> : main 브랜치는 그대로 유지하면서 해당 커밋 ID로 돌아갈 수 있다.
  - HEAD에서 분리된 상태로, 실험적인 변경이나 그러한 변경들을 커밋할 수 있고,
  - 다른 브랜치에 영향을 주지 않고 상태를 유지시킬 수 있다.

```
Username@DESKTOP MINGW64 ~/my_story (main)
$ git checkout e288

Note: switching to 'e288'.

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.

...

Username@DESKTOP MINGW64 ~/my_story ((e288f45...))
$
```

Git & GitHub 기초 (CLI) [실습]

- word.txt를 열면, ST까지 작성되어 있는 상태로 돌아간 것을 확인할 수 있다.
- 이제 앞서 실습한 것과 마찬가지로 커밋을 진행해보자.
  - word.txt에 A를 입력한 후 저장하고, 커밋을 진행 [commit message : edit A]
  - word.txt에 C를 입력한 후 저장하고, 커밋을 진행 [commit message : edit C]
  - word.txt에 K를 입력한 후 저장하고, 커밋을 진행 [commit message : edit K]

- 커밋이 기록될 때마다 가장 최근 커밋의 ID로 브랜치가 변경되는 것을 확인할 수 있다.

```
Username@DESKTOP MINGW64 ~/my_story ((15b785b...))
$ git log --oneline
15b785b (HEAD) K
243f7e8 C
98762c0 A
e288f45 T
65fd425 S
bd295f3 second commit
1165749 first commit
```

Git & GitHub 기초 (CLI) [실습]

- git switch : HEAD가 참조하는 브랜치를 변경한다.
- git switch -c <브랜치명> : branch를 생성하고, 생성된 branch로 현재 branch를 변경

```
Username@DESKTOP MINGW64 ~/my_story ((15b785b...))
$ git switch -c dev

Username@DESKTOP MINGW64 ~/my_story (dev)
$
```

Git & GitHub 기초 (CLI) [실습]

- switch 이전과 비교해보면 HEAD가 참조하고 있는 branch가 생성된 것을 확인할 수 있다.

```
Username@DESKTOP MINGW64 ~/my_story ((15b785b...))
$ git log --oneline
15b785b (HEAD) K
243f7e8 C
98762c0 A
e288f45 T
65fd425 S
bd295f3 second commit
1165749 first commit
```

```
Username@DESKTOP MINGW64 ~/my_story (dev)
$ git log --oneline
15b785b (HEAD -> dev) K
243f7e8 C
98762c0 A
e288f45 T
65fd425 S
bd295f3 second commit
1165749 first commit
```

Git & GitHub 기초 (CLI) [실습]

- 현재 word.txt에는 STACK이 작성되어 있을 것이다.
- 이제 기본 branch였던 main으로 돌아가 STORE를 확인해보자.

- git switch <브랜치명> : 해당 branch로 이동
- git branch -l : 로컬 branch 리스트 확인

```
Username@DESKTOP MINGW64 ~/my_story (dev)
$ git switch main
Switched to branch 'main'

Username@DESKTOP MINGW64 ~/my_story (main)
$ git branch -l
  dev
* main
```

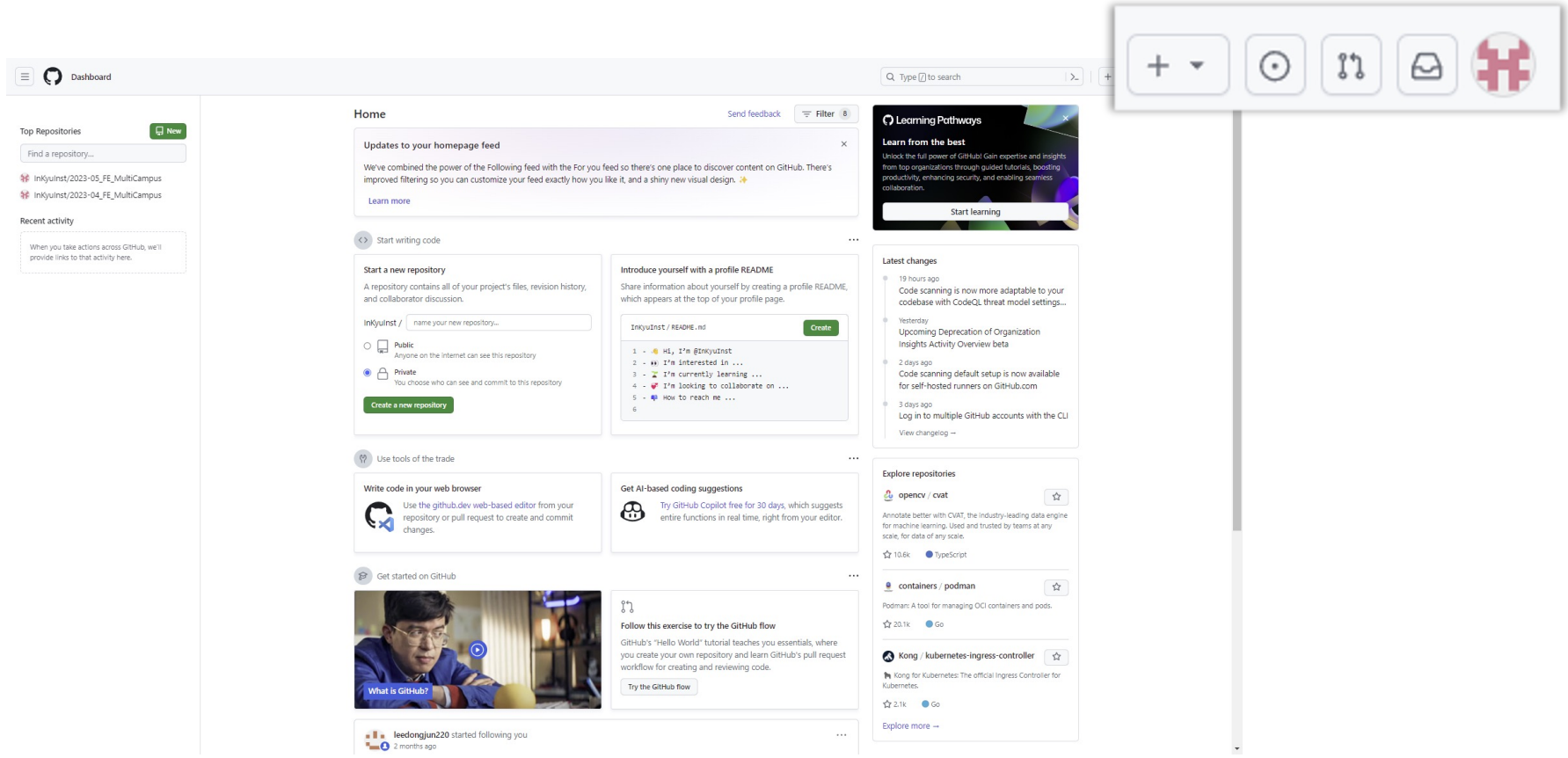
Git & GitHub 기초 (CLI)

- 앞서 작업한 내용은 로컬 저장소(Local Repository)만을 이용한 작업이었다.
- 이제는 Git Repository의 온라인 저장 사이트, GitHub를 이용해 원격 저장소(Remote Repository) 개념을 도입해보자.

- GitHub는 원격 Git Repository를 제공하는 서비스로,
  내 컴퓨터에 있는 Local Repository의 기록들을 업로드하고, 다운로드 받으며
  웹을 통해 언제 어디서나 버전관리를 할 수 있도록 해준다.

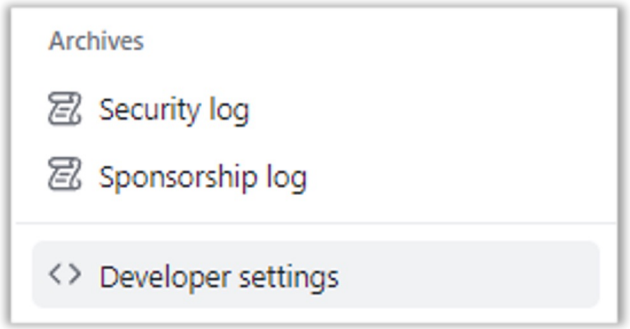
Git & GitHub 기초 (CLI)

- 내 컴퓨터에서 GitHub와 연동을 하기 위해서는 GitHub에 로그인을 해야 한다.
- 보안 정책에 따라 "내 컴퓨터"에서 로그인을 하기 위해서는 비밀번호가 아닌 Token을 통해 로그인(인증)을 해야한다.
- 프로필 이미지 클릭 후, Setting 클릭



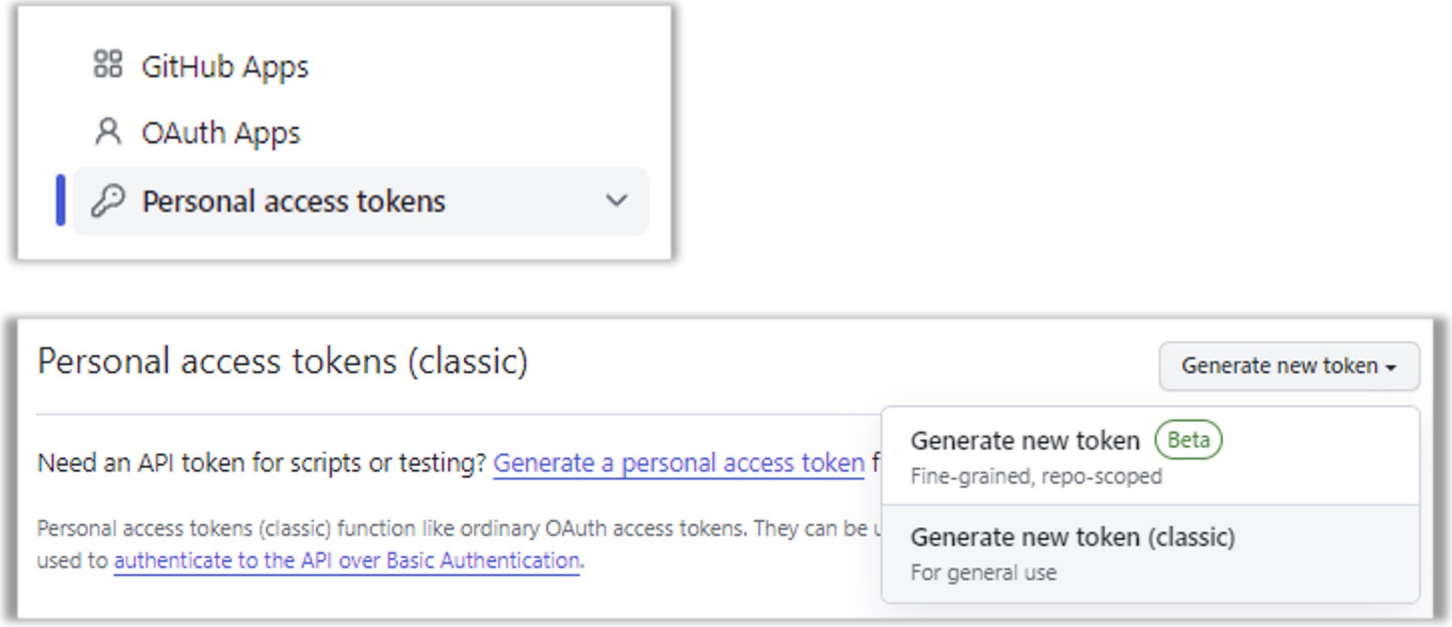
Git & GitHub 기초 (CLI)

- setting에서는 프로필 정보 수정이 가능하며, 좌측 메뉴를 이용해 Developer settings으로 이동한다.



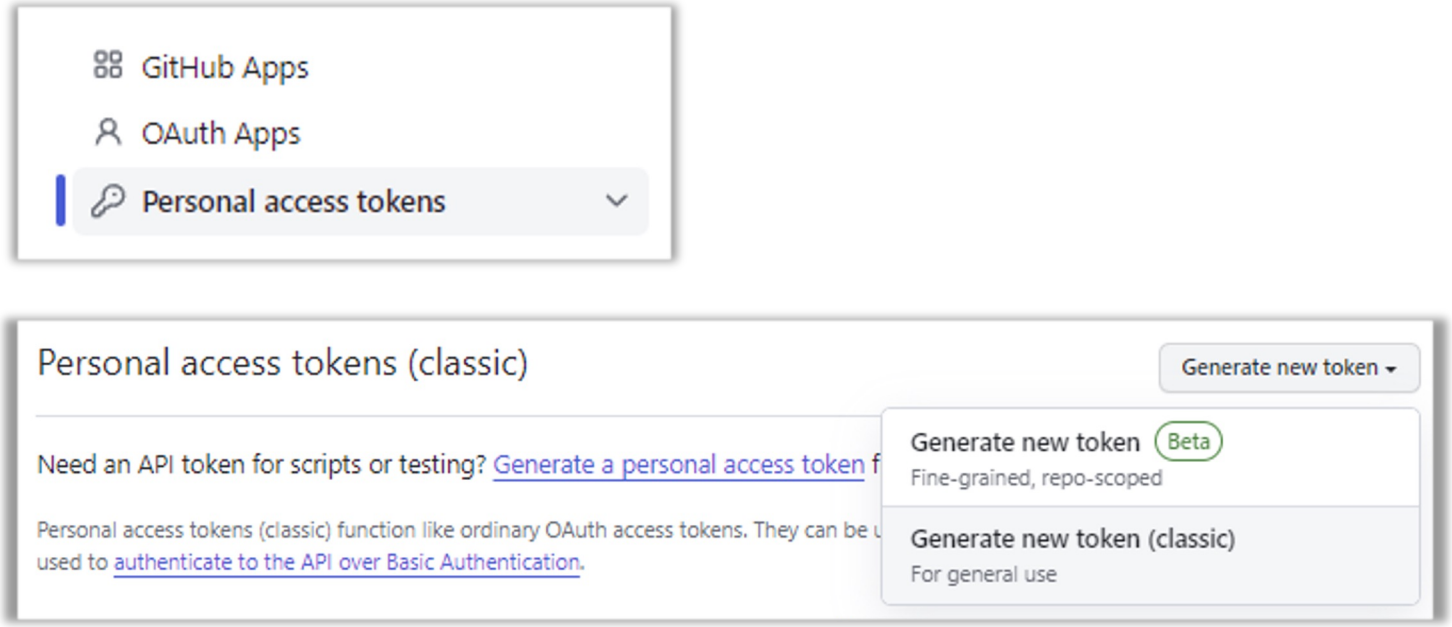
Git & GitHub 기초 (CLI)

- Personal access tokens를 선택하고, Generate new token (classic)을 선택하여 토큰 생성



Git & GitHub 기초 (CLI)

- Personal access tokens를 선택하고, Generate new token (classic)을 선택하여 토큰 생성



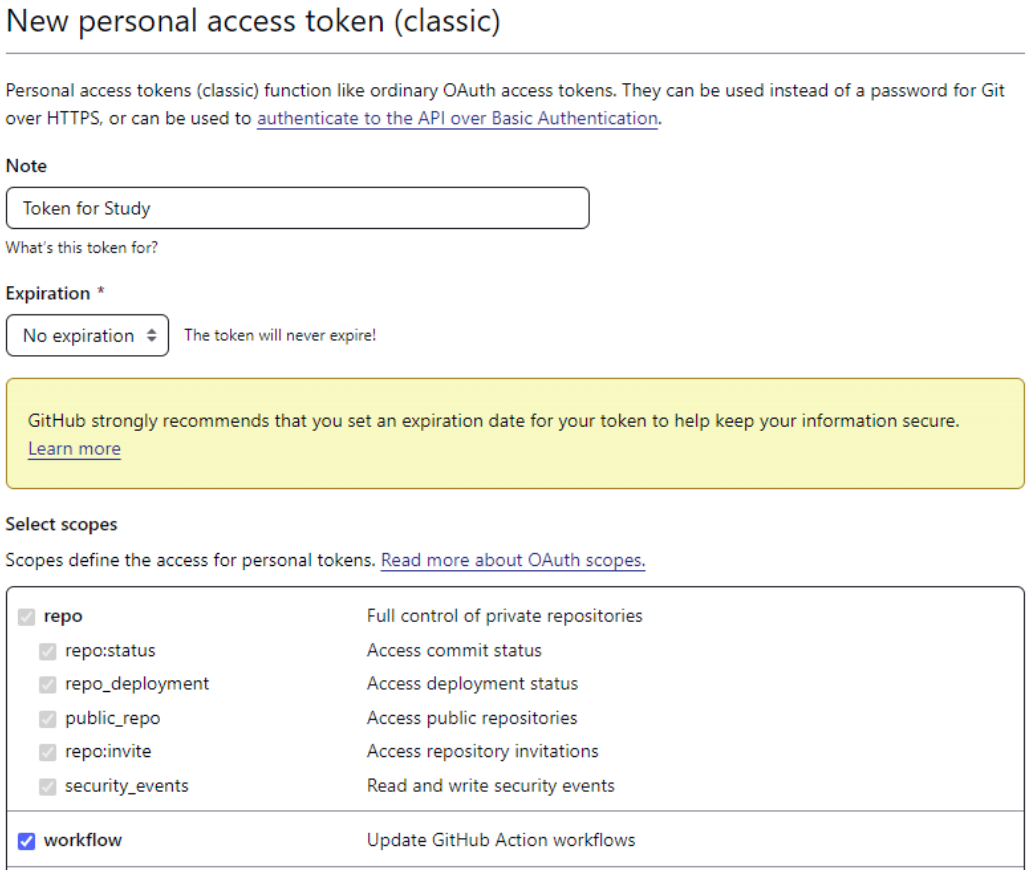
Settings👉 Developer settings👉 Personal access tokens 👉 Generate new token👉 Generate new token (classic)

Git & GitHub 기초 (CLI)

- Personal access tokens를 선택하고, Generate new token (classic)을 선택하여 토큰 생성

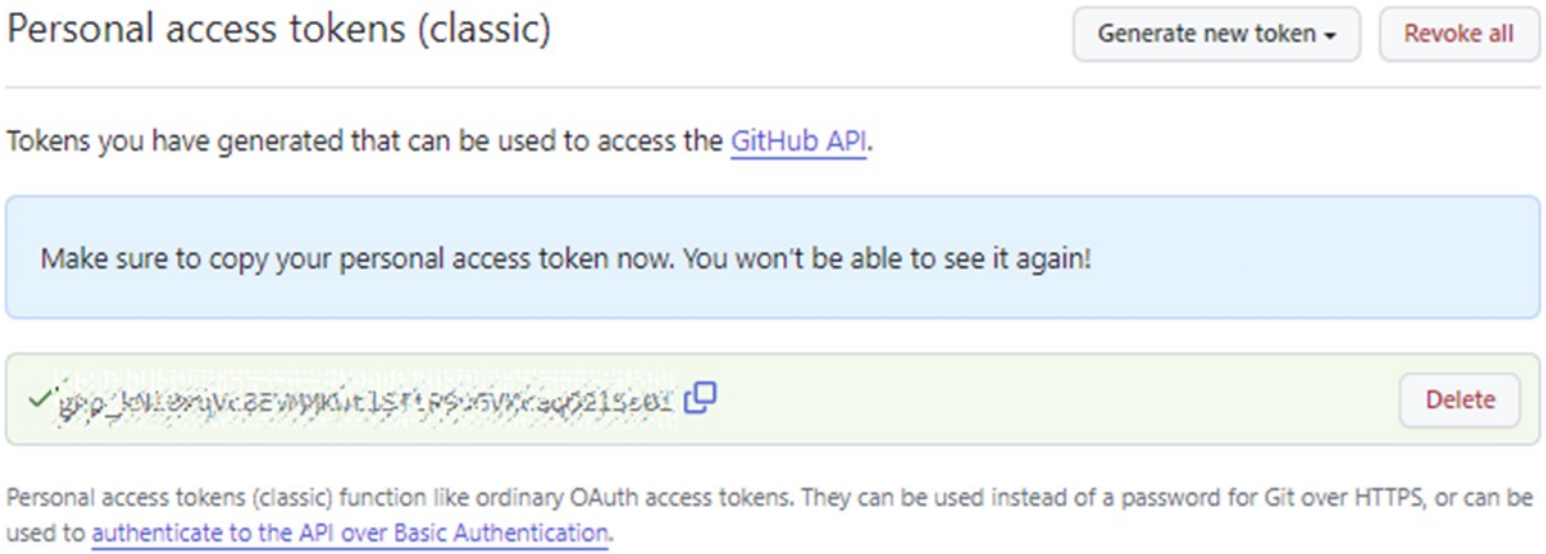
  - Note : Token for Study (토큰을 식별하기 위한 메모)
  - Expiration : No expiration (토큰 만료기간)
  - Select scopes : repo, workflow (해당 토큰 접근 범위 설정)

Generate token



Git & GitHub 기초 (CLI)

- 생성된 토큰을 복사하여 기록해두자.



Git & GitHub 기초 (CLI)

- GitHub에 개발한 코드를 올릴 때에는 아래와 같은 순서를 따른다.

1. 내 컴퓨터 안에 존재하는 프로젝트 폴더에서 "이제 여기에서 Git을 사용할거야!" 라고 명령하기 | (git init)
2. 즐거운 마음으로 코딩하기
3. 내 파일들 중에 GitHub에 올릴 파일들을 선택하기 | (git add)
4. 선택한 파일들을 한 덩어리로 만들어서 설명 적기 | (git commit)
5. GitHub에서 프로젝트 저장소를 생성하기 |
6. 내 컴퓨터 프로젝트 폴더에 "내 GitHub 저장소 주소는 여기야!" 라고 알려주기 | (git remote add)
7. "덩어리들을 GitHub에 올려줘!" 라고 명령하기 | (git push)

Git & GitHub 기초 (CLI) [실습]

company

home

출근

1. 로컬 저장소를 생성
2. 원격 저장소를 생성
3. 로컬 저장소에 원격 저장소를 연결
4. 기능 단위로 add과 commit을 진행
5. 원격 저장소로 push

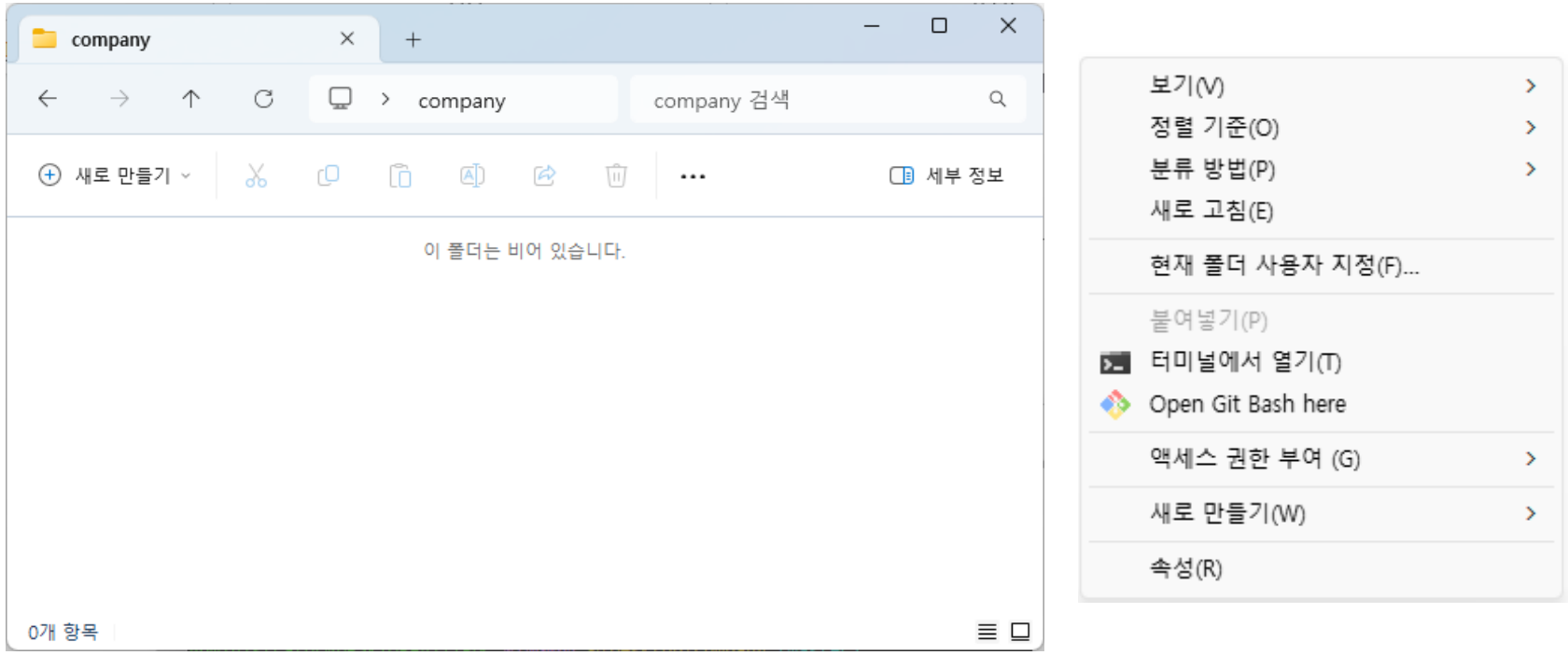
퇴근

귀가

6. 원격 저장소를 복제 또는 pull
7. 이어서 작업 진행 (기능 단위로 add과 commit)
8. 원격 저장소로 push

취침

Git & GitHub 기초 (CLI) [실습]



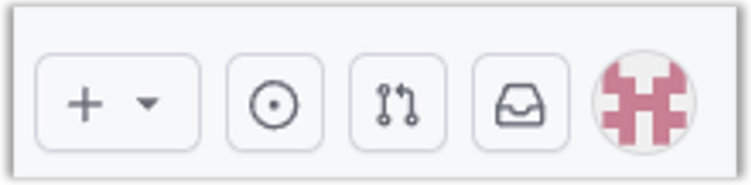
Git & GitHub 기초 (CLI) [실습]

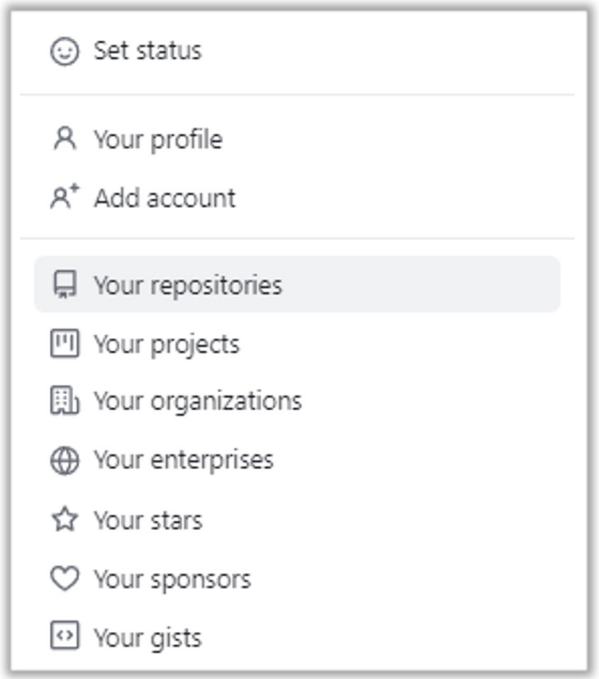
- 프로젝트 폴더에서 "이제 여기에서 Git을 사용할거야!" 라고 명령하기 | (git init)
- 즐거운 마음으로 코딩하기
- 내 파일들 중에 GitHub에 올릴 파일들을 선택하기 | (git add)
- 선택한 파일들을 한 덩어리로 만들어서 설명 적기 | (git commit)

```
Username@DESKTOP MINGW64 ~/Desktop/company
$ git init
Initialized empty Git repository in C:/Users/UserName/Desktop/company/.git/

$ echo "# 끝말잇기" >> README.md

$ git add .

$ git commit -m "first commit"
```

Git & GitHub 기초 (CLI) [실습]

- GitHub에서 프로젝트 저장소를 생성하기

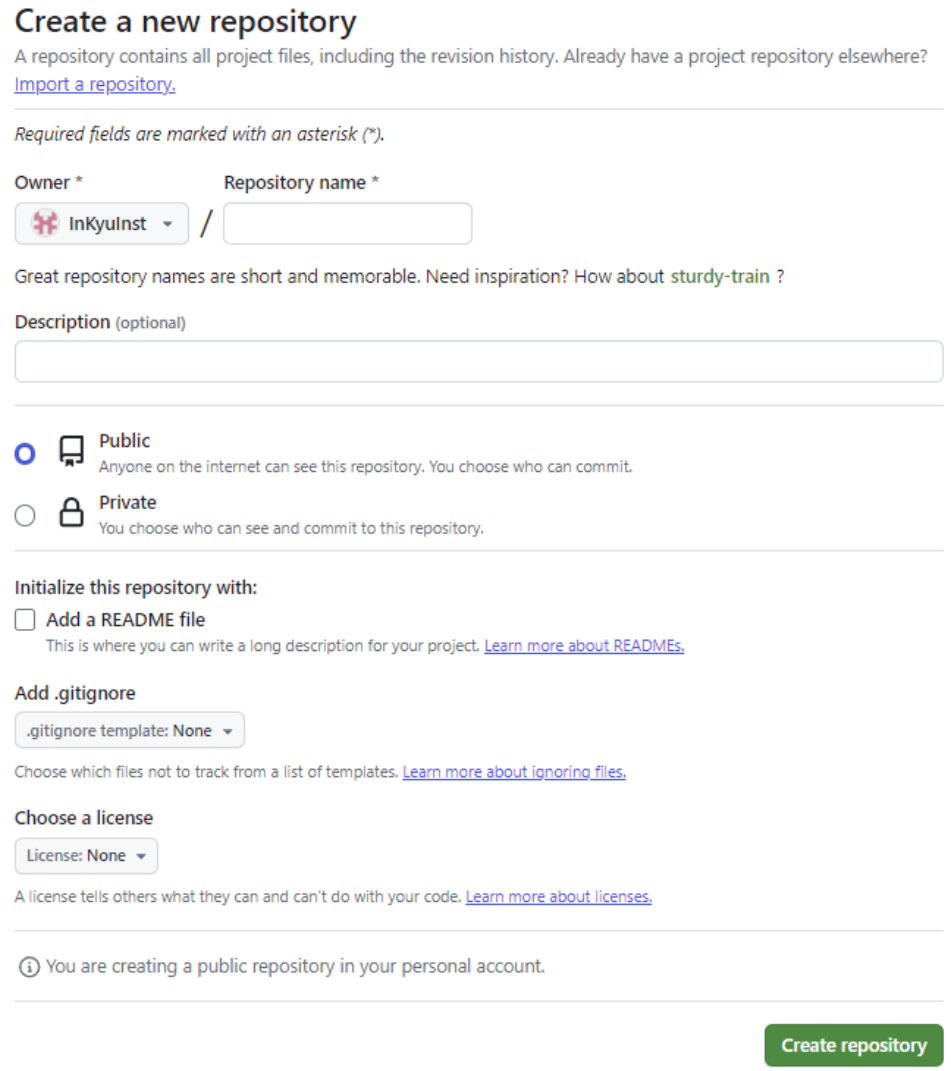




Git & GitHub 기초 (CLI) [실습]

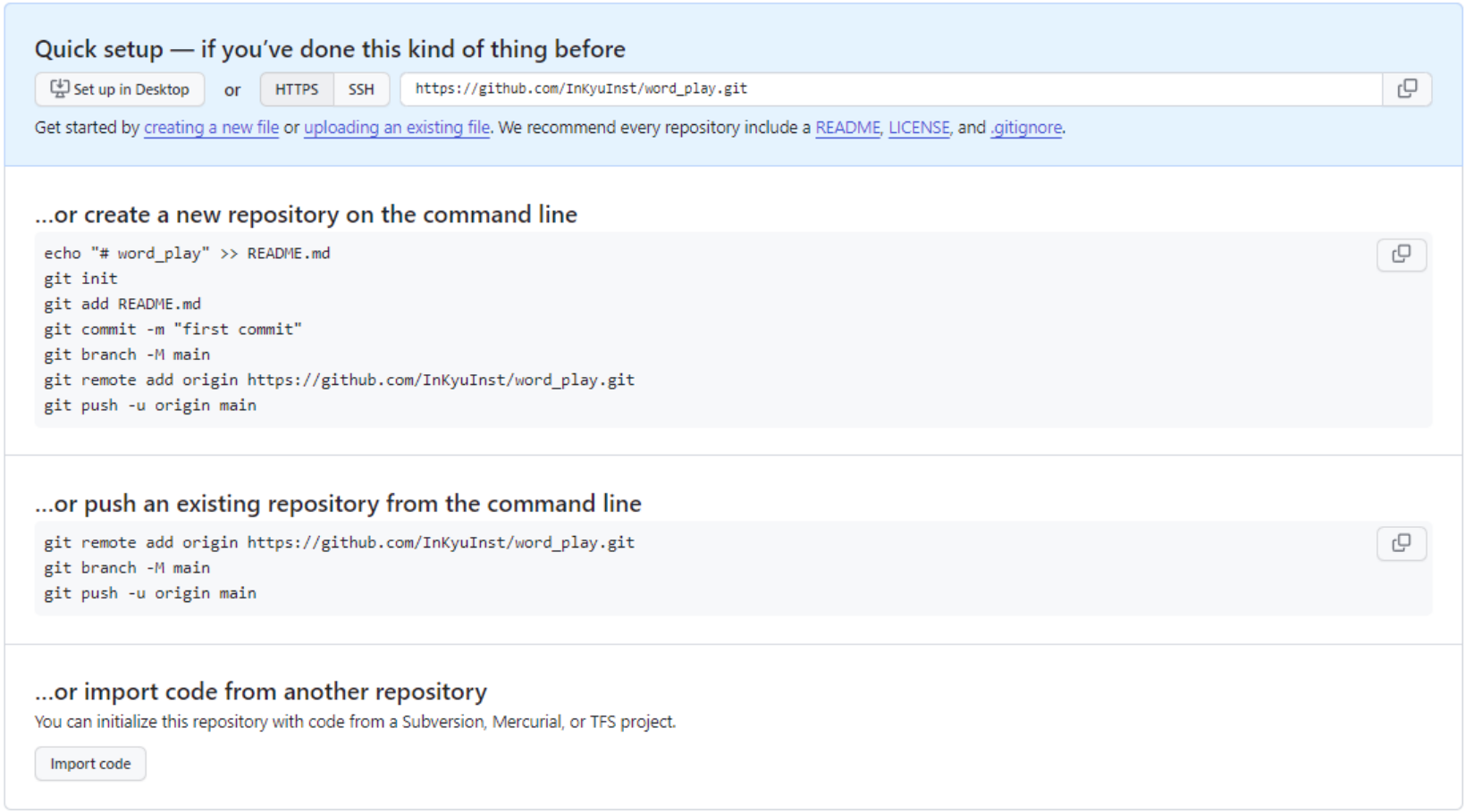
- GitHub에서 프로젝트 저장소를 생성하기

  - Repository name (저장소 명) : word_play
  - Description (저장소 설명) : 끝말잇기
  - public / private (공개 범위) : public
  - Add a README file (저장소 설명 파일 추가 여부) : X
  - Add .gitingnore (Git에서 관리하지 않을 파일 지정) : None
  - Choose a license (라이선스 안내) : None



Git & GitHub 기초 (CLI) [실습]

- GitHub에서 프로젝트 저장소를 생성하기



Git & GitHub 기초 (CLI) [실습]

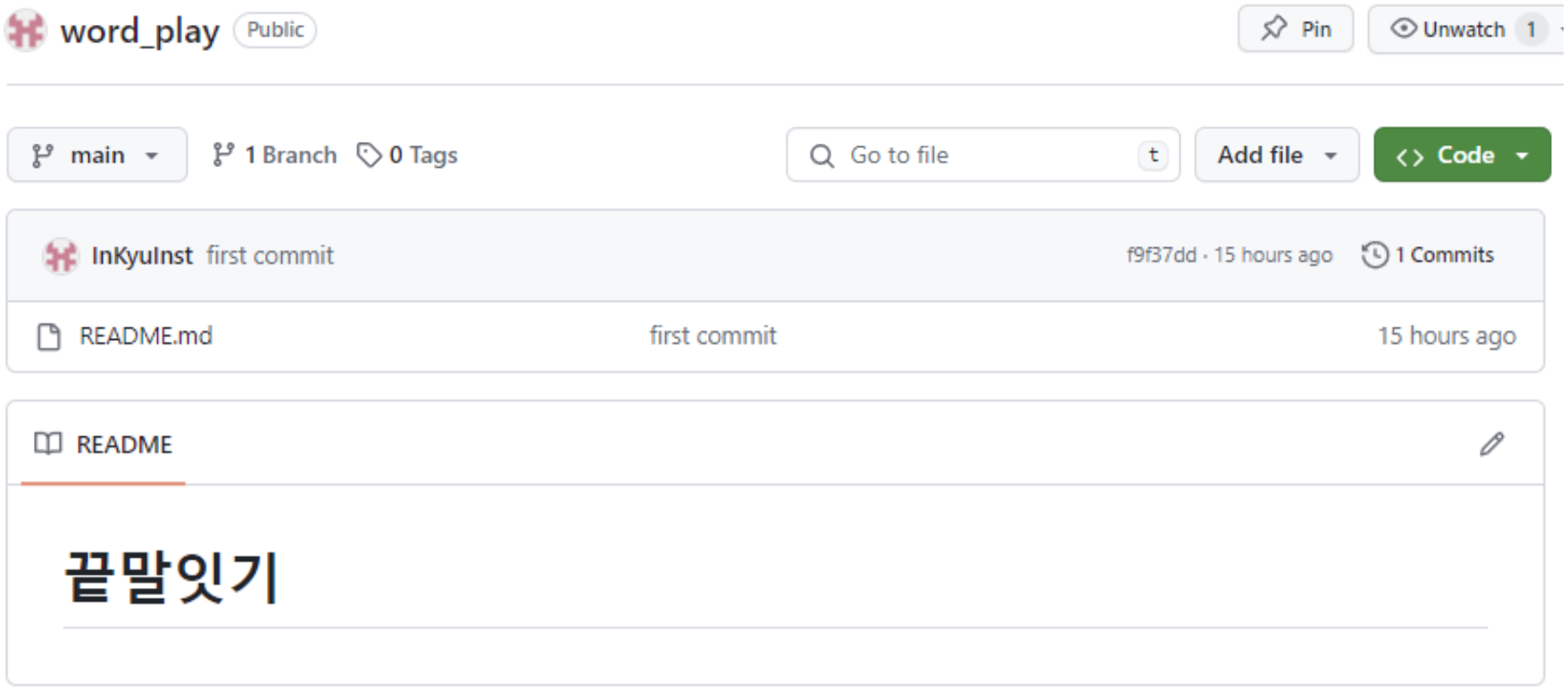
- 내 컴퓨터 프로젝트 폴더에 "내 GitHub 저장소 주소는 여기야!" 라고 알려주기 | (git remote add)
- "덩어리들을 GitHub에 올려줘!" 라고 명령하기 | (git push)

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git remote add origin https://github.com/사용자이름/word_play.git

$ git remote -v
origin https://github.com/사용자이름/word_play.git (fetch)
origin https://github.com/사용자이름/word_play.git (push)

$ git push -u origin main
```

TIP

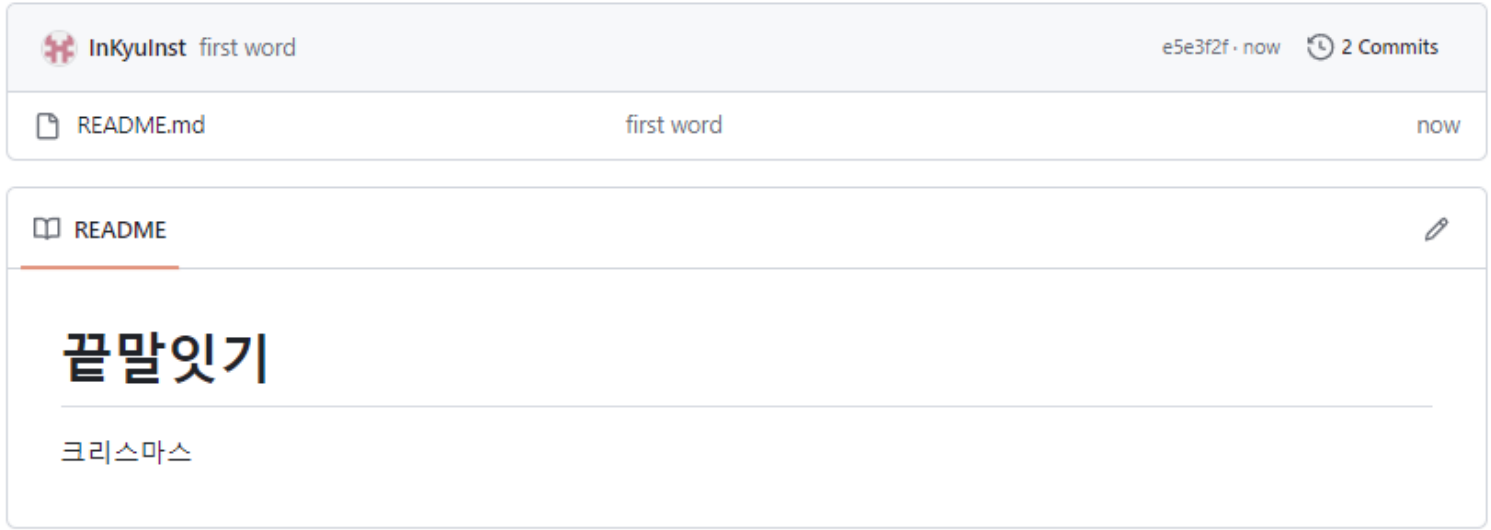
origin : 원격 저장소를 가리키는 가장 일반적인 별칭
remote -v 옵션 : 로컬 저장소와 연결된 원격 저장소의 주소 확인
push -u 옵션 : 로컬 브랜치와 연결된 원격 브랜치 지정 (로컬의 main 브랜치와 원격의 main 브랜치 연결)

Git & GitHub 기초 (CLI) [실습]



Git & GitHub 기초 (CLI) [실습]

```
Username@DESKTOP MINGW64 ~/Desktop/company (master)
$ echo "크리스마스" >> README.md

$ git add .

$ git commit -m "first word"

$ git push
```



Git & GitHub 기초 (CLI) [실습]

```
Username@DESKTOP MINGW64 ~/Desktop/company (master)
$ cd ..

Username@DESKTOP MINGW64 ~/Desktop
$ cd home

Username@DESKTOP MINGW64 ~/Desktop/home
$ git clone https://github.com/사용자이름/word_play.git

Username@DESKTOP MINGW64 ~/Desktop/home
$ ls
word_play/

Username@DESKTOP MINGW64 ~/Desktop/home
$ cd word_play

Username@DESKTOP MINGW64 ~/Desktop/home/word_play (main)
$
```

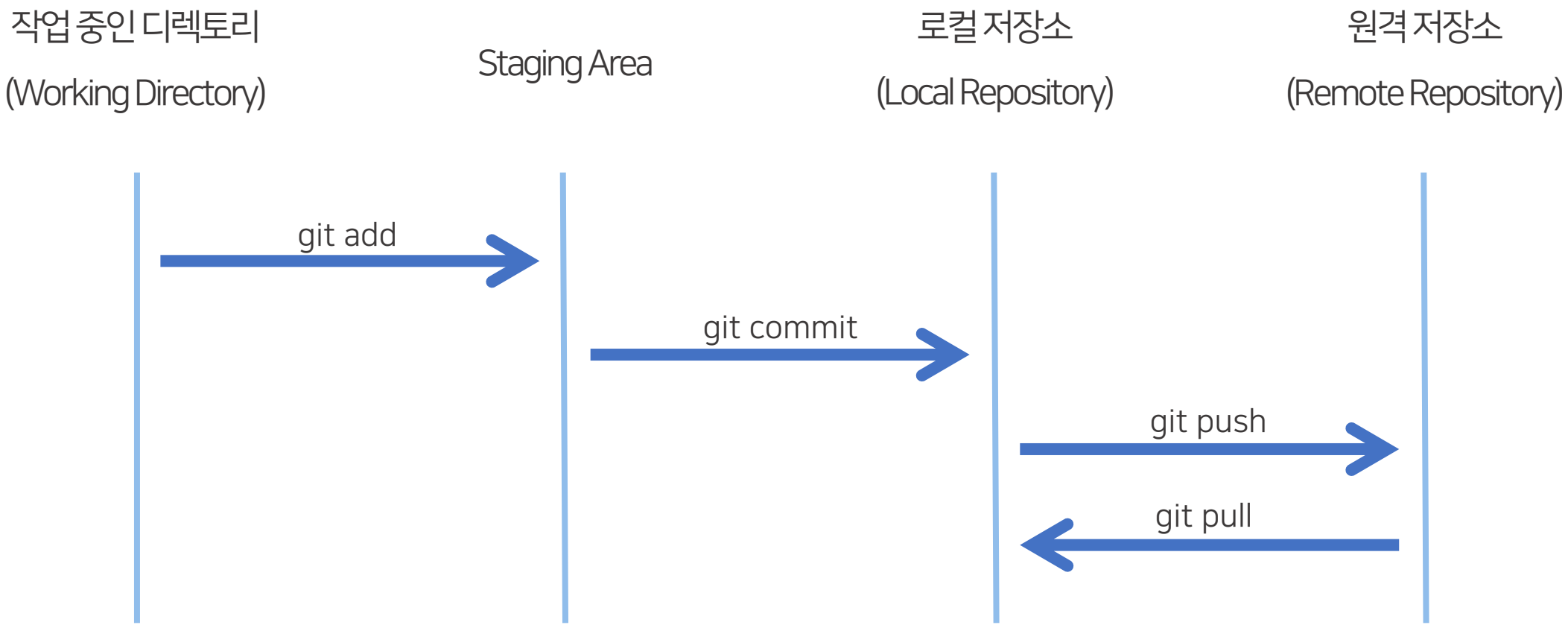
Git & GitHub 기초 (CLI) [실습]

```
Username@DESKTOP MINGW64 ~/Desktop/home/word_play (main)
$ echo "스위스" >> README.md

$ git add .

$ git commit -m "second word"

$ echo "스트레스" >> README.md

$ git commit -a -m "third word"

$ git push
```

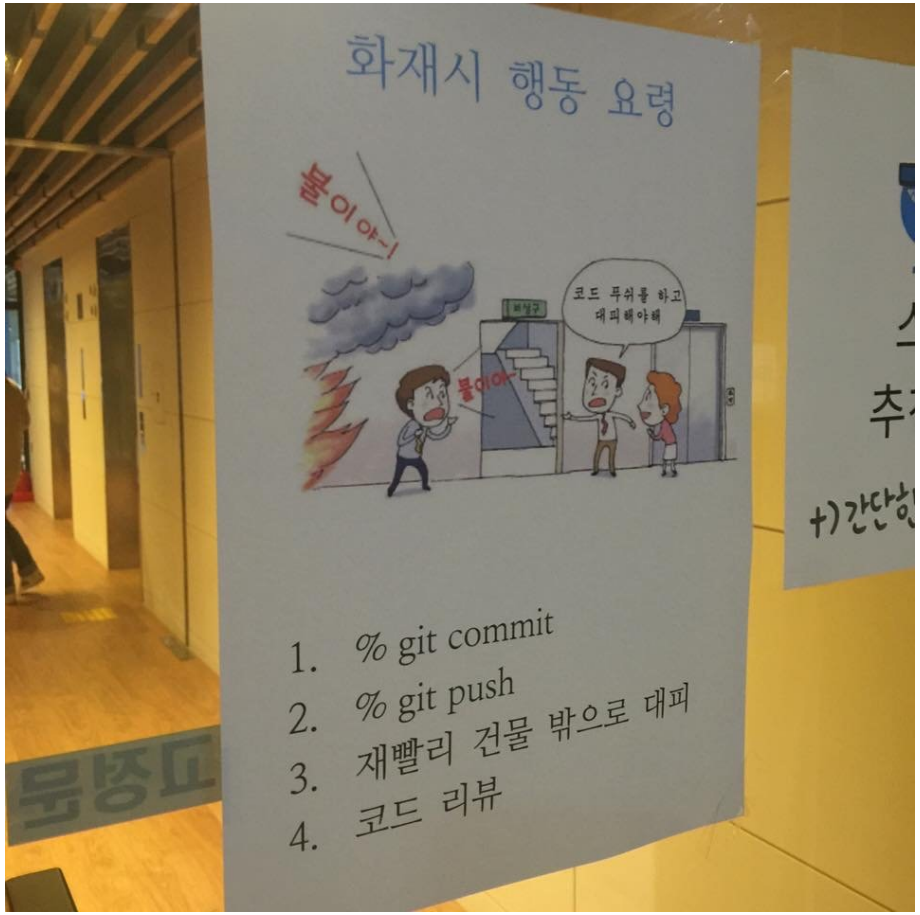
Git & GitHub 기초 (CLI) [실습]

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ 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), 264 bytes | 20.00 KiB/s, done.
From https://github.com/사용자이름/word_play
   e5e3f2f..03d5260  main     -> origin/main
Updating e5e3f2f..03d5260
Fast-forward
 README.md | 1 +
 1 file changed, 1 insertion(+)
```

Git & GitHub 기초 (CLI)

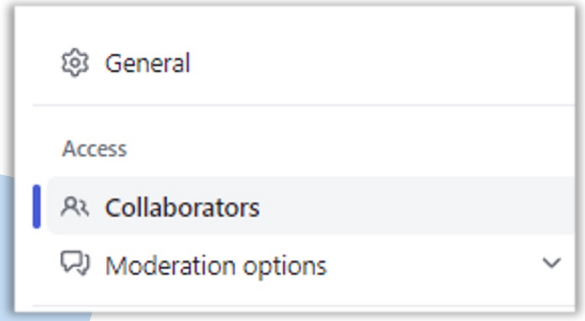


Git & GitHub 기초 (CLI)



Git & GitHub 기초 (CLI)

제 원격 저장소를 이용해 작업하시죠!
Collaborator로 추가해놓을게요.
이메일 확인해주세요!



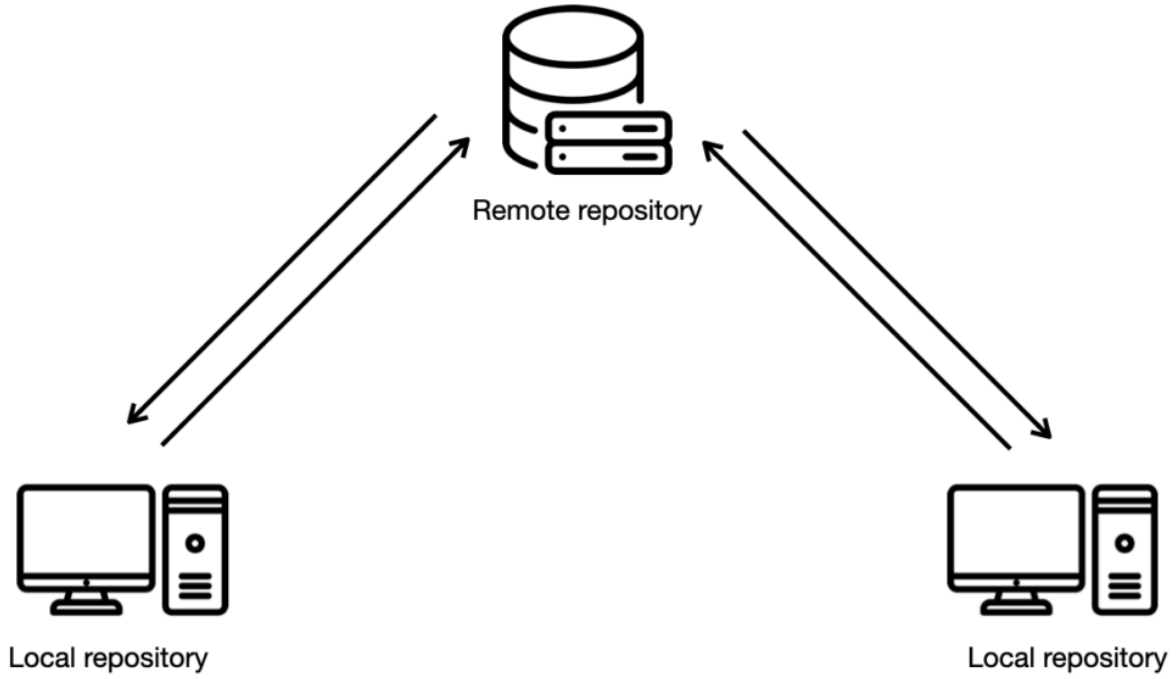
네! 감사합니다. clone해서 작업하겠습니다.

Alice

Bob

Git & GitHub 심화 (GUI)

- 원격 저장소로부터 각각의 로컬에서 작업을 진행하게 되면, 같은 파일을 변경하게 되는 경우가 생길 수 있다.
- 같은 내용이 동시에 수정하여 발생하는 문제를 "충돌(conflict)"이라 한다.
- 협업에서 충돌이 발생하는 것은 피할 수 없다.
- 프로젝트의 규모가 커지면 커질수록, 협업하는 인원이 많아질수록 충돌은 더 많이 발생하게 된다.

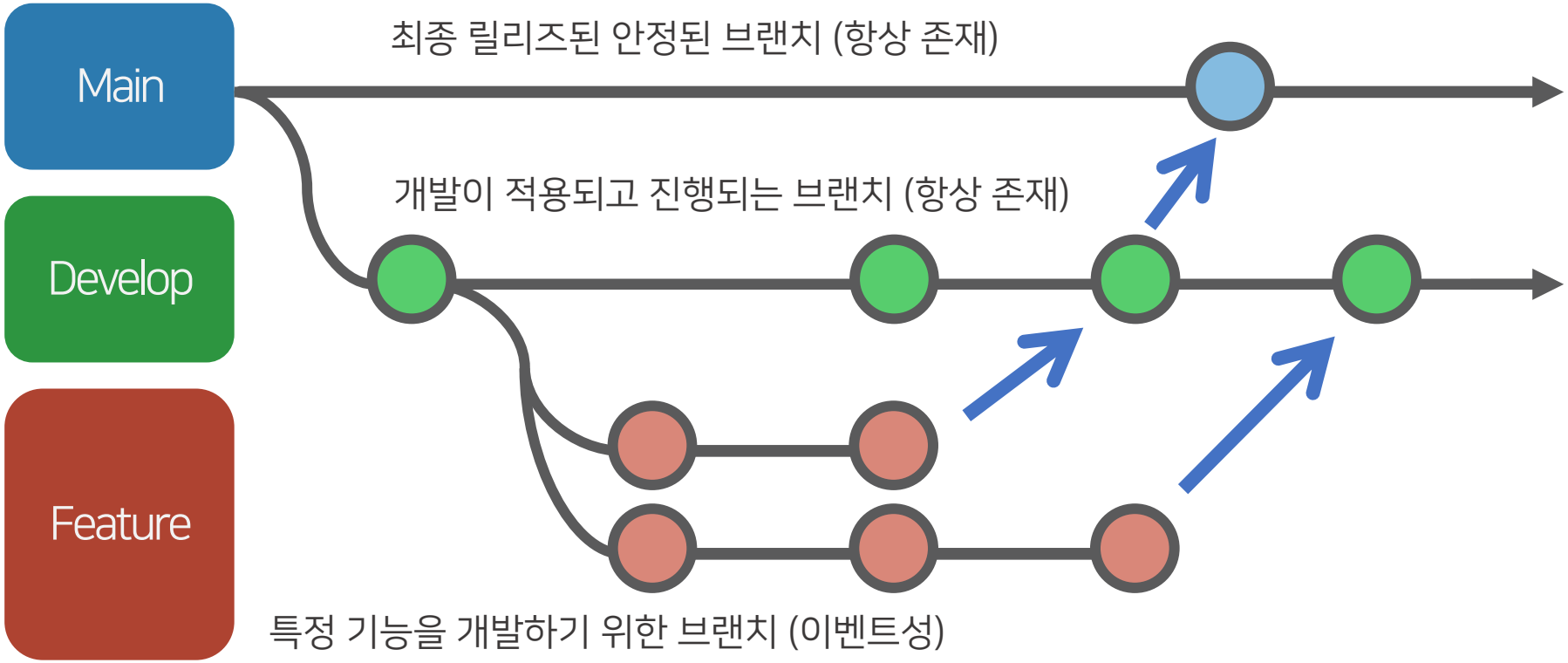


Git & GitHub 심화 (GUI)

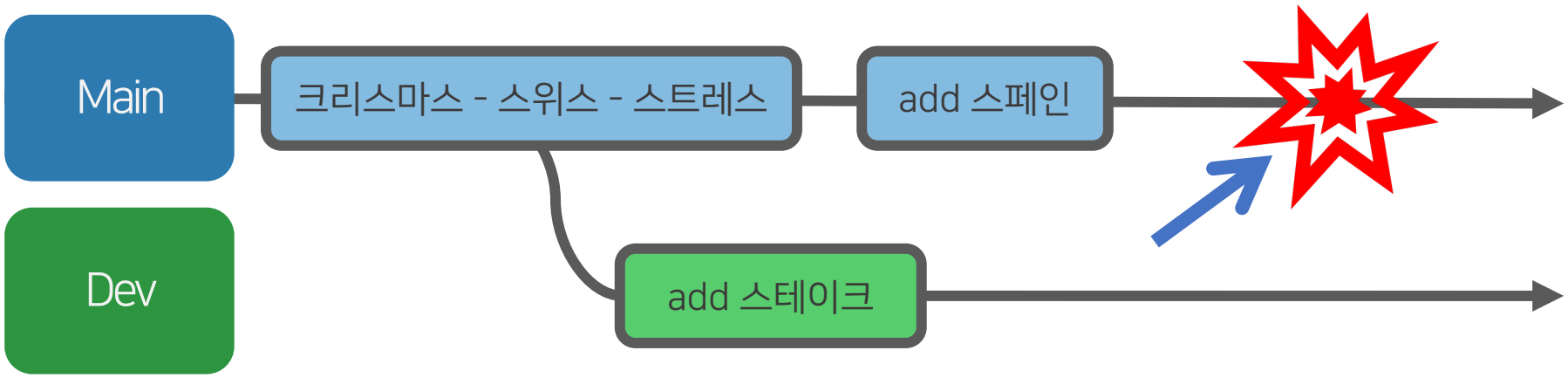
- 수시로 코드 통합을 거쳐 큰 충돌을 미리 방지할 수 있으며, Branch를 활용하여 충돌을 최소화할 수 있다.
- Branch 전략은 협업 환경에서 효율성을 높일 수 있는 중요한 요소이다.

TIP

브랜치가 합쳐지는 파란색 화살표를 merge라고 한다.



Git & GitHub 심화 (GUI)



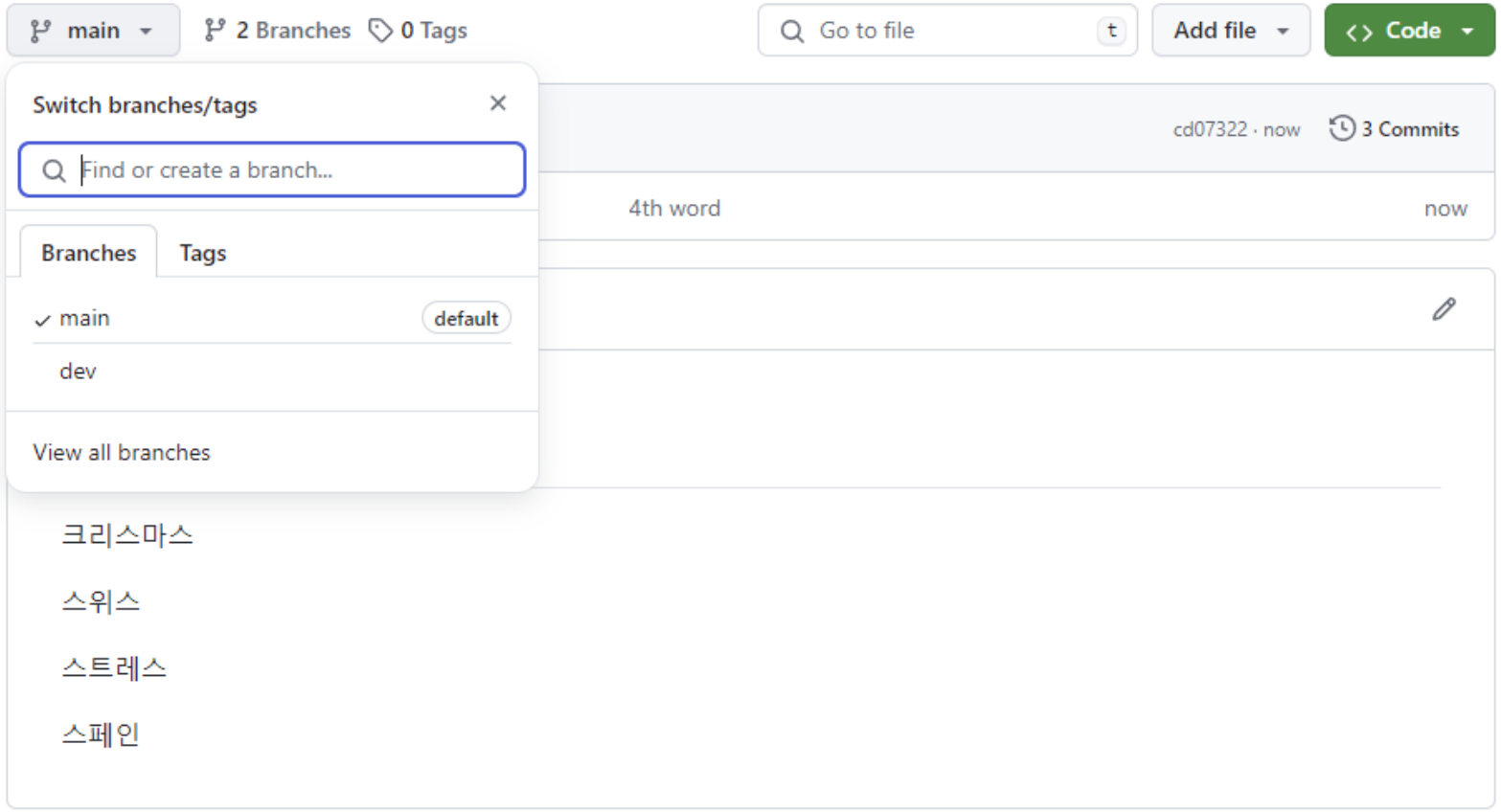
Git & GitHub 심화 (GUI)

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git switch -c dev

Username@DESKTOP MINGW64 ~/Desktop/company (dev)
$ echo "스테이크" >> README.md

$ git commit -am "4th word"

$ git push -u origin dev
```

Git & GitHub 심화 (GUI)

```
Username@DESKTOP MINGW64 ~/Desktop/company (dev)
$ git switch main

Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ echo "스페인" >> README.md

$ git commit -am "4th word"

$ git push
```

Git & GitHub 심화 (GUI)



Git & GitHub 심화 (GUI)

- git merge <브랜치명> : 현재 브랜치에 특정 브랜치를 병합한다.
  - README.md를 자동 병합을 하려고 했으나, 충돌이 발생하였으므로 직접 해결하고 commit을 해야 하는 상황!

```
Username@DESKTOP MINGW64 ~/Desktop/company (main)
$ git merge dev
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.

Username@DESKTOP MINGW64 ~/Desktop/company (main|MERGING)
```

Git & GitHub 심화 (GUI)

- git merge <브랜치명> : 현재 브랜치에 특정 브랜치를 병합한다.
  - README.md를 자동 병합을 하려고 했으나, 충돌이 발생하였으므로 직접 해결하고 commit을 해야 하는 상황!

```
Username@DESKTOP MINGW64 ~/Desktop/company (main|MERGING)
$ cat README.md
# 끝말잇기
크리스마스
스위스
스트레스
<<<<<<< HEAD (Current Change)
스페인
=======
스테이크
>>>>>>> dev (Incoming Change)
```

Git & GitHub 심화 (GUI)

- 충돌을 해결한 후, 다시 add와 commit을 진행한 뒤에 push까지 하면 충돌이 수정되고 merge가 진행된다.

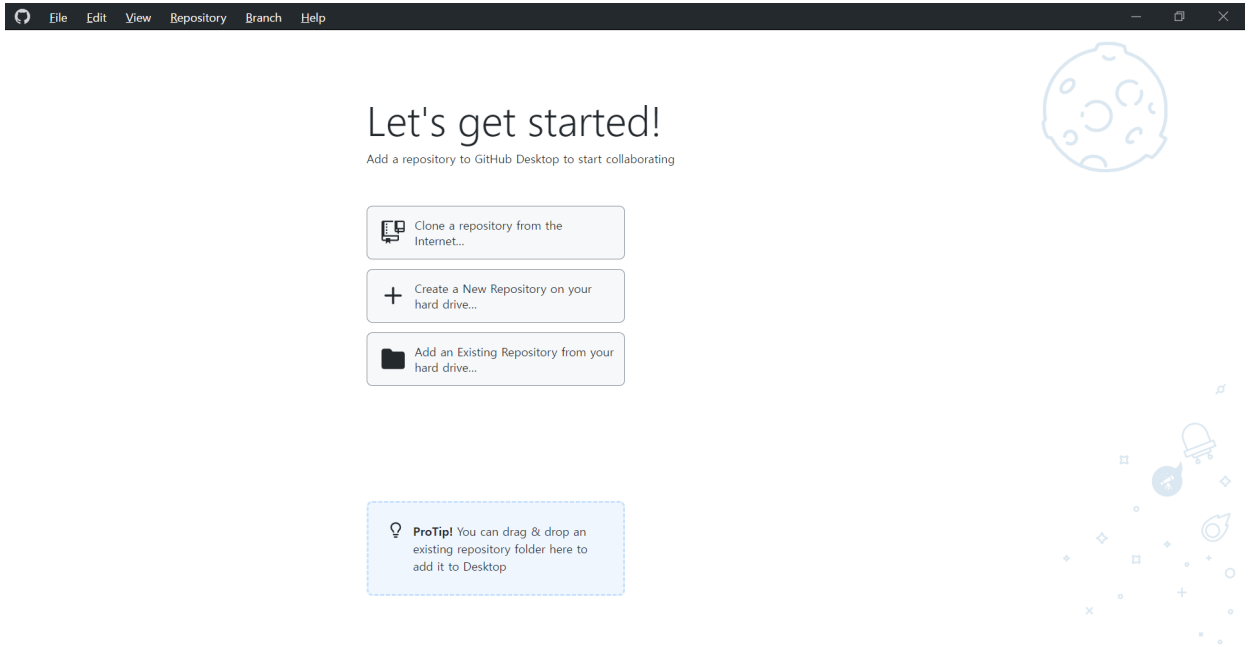
```
Username@DESKTOP MINGW64 ~/Desktop/company (main|MERGING)
$ git add .

Username@DESKTOP MINGW64 ~/Desktop/company (main|MERGING)
$ git commit -m "fix conflict"

Username@DESKTOP MINGW64 ~/Desktop/company (main|MERGING)
$ git push

Username@DESKTOP MINGW64 ~/Desktop/company (main)
$
```

Git & GitHub 심화 (GUI)

- https://desktop.github.com/ GitHub Desktop 다운로드

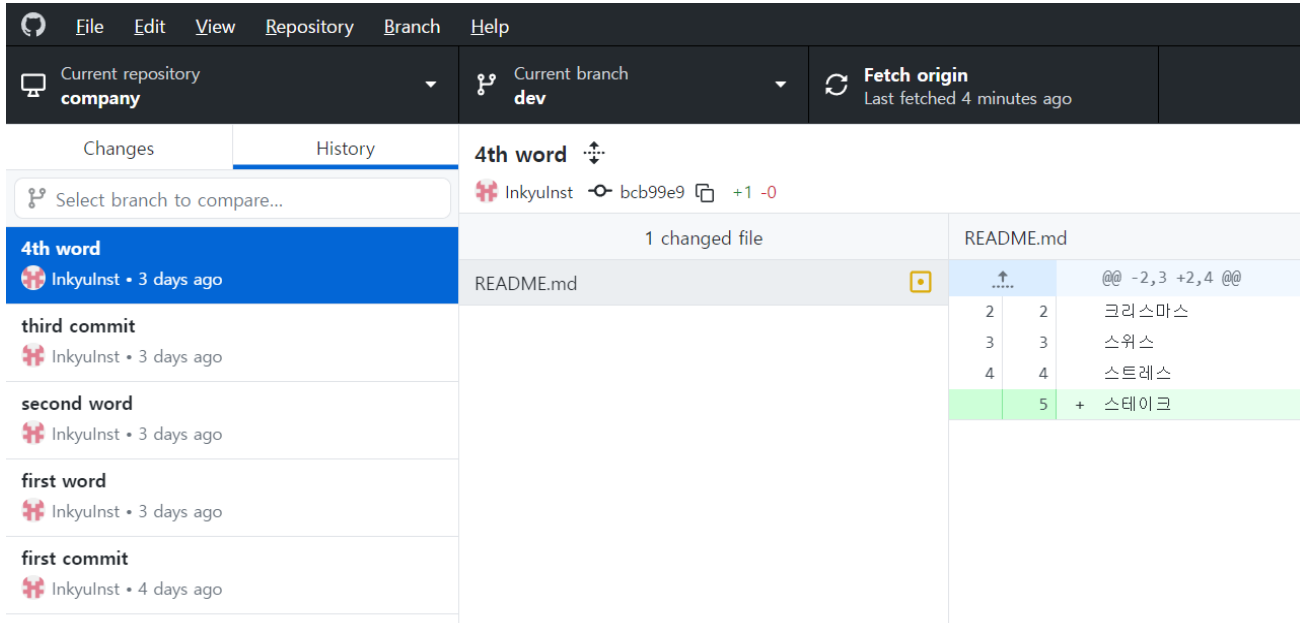
Git & GitHub 심화 (GUI)

- GitHub Desktop 실행
  - Clone a repository from the Internet : 원격 저장소 Clone
  - Create a New Repository : 새 원격 저장소 생성
  - Add an Existing Repository from your hard drive : 로컬 저장소 가져오기



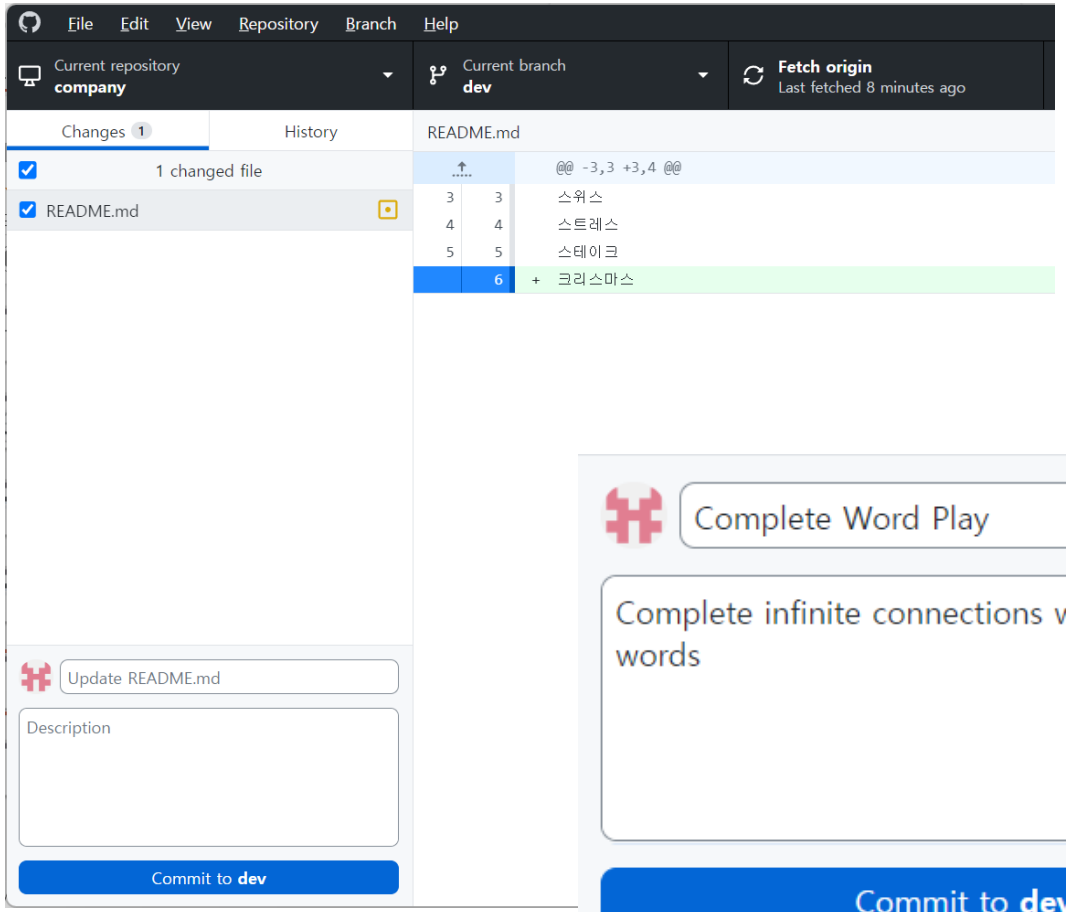
Git & GitHub 심화 (GUI)

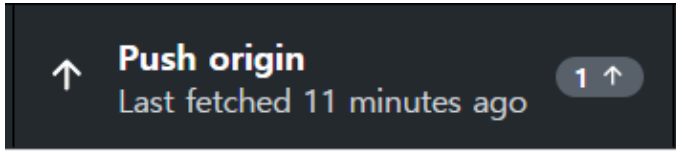
- GitHub Desktop 실행



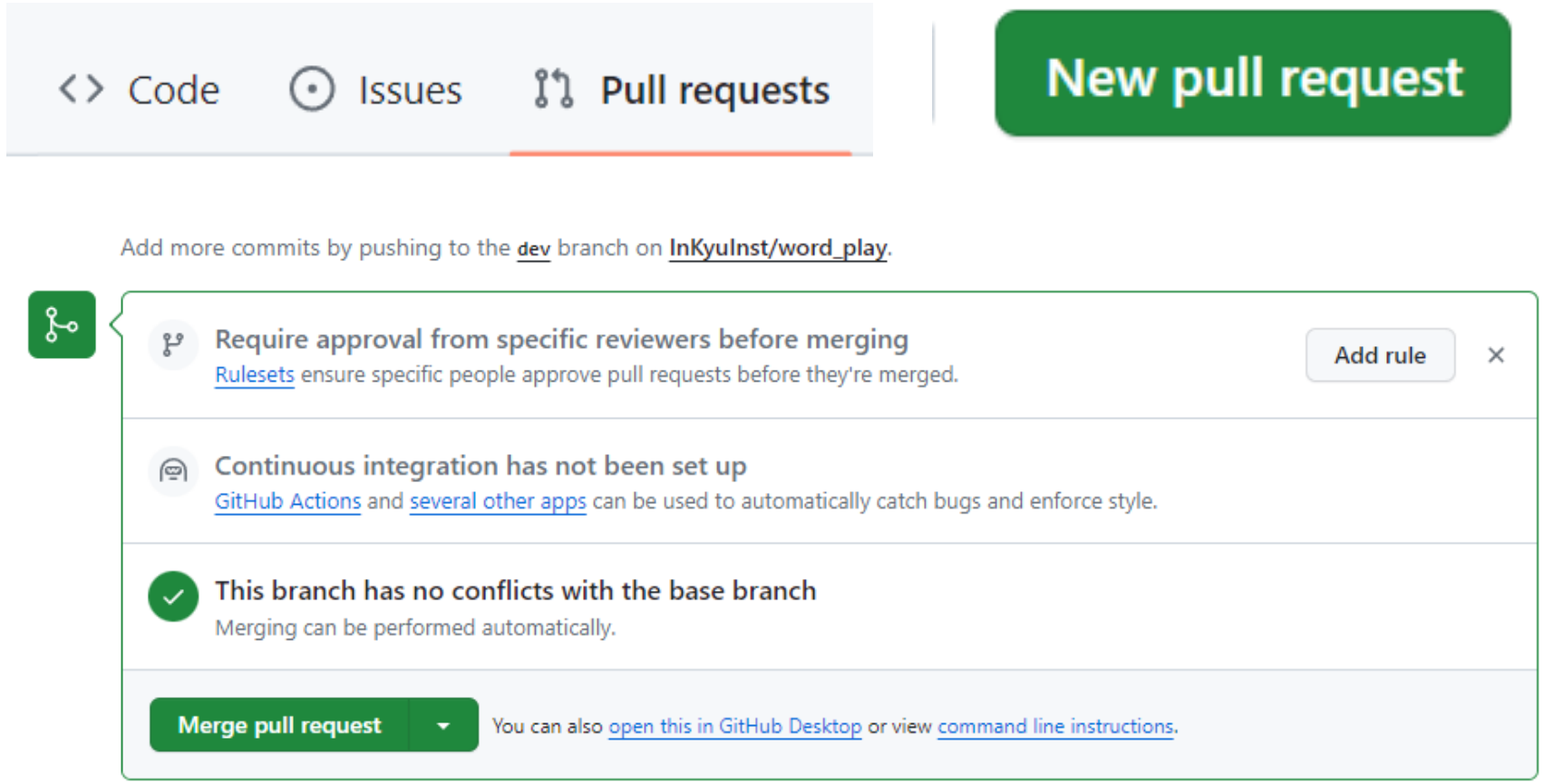
Git & GitHub 심화 (GUI)

- GitHub Desktop 실행

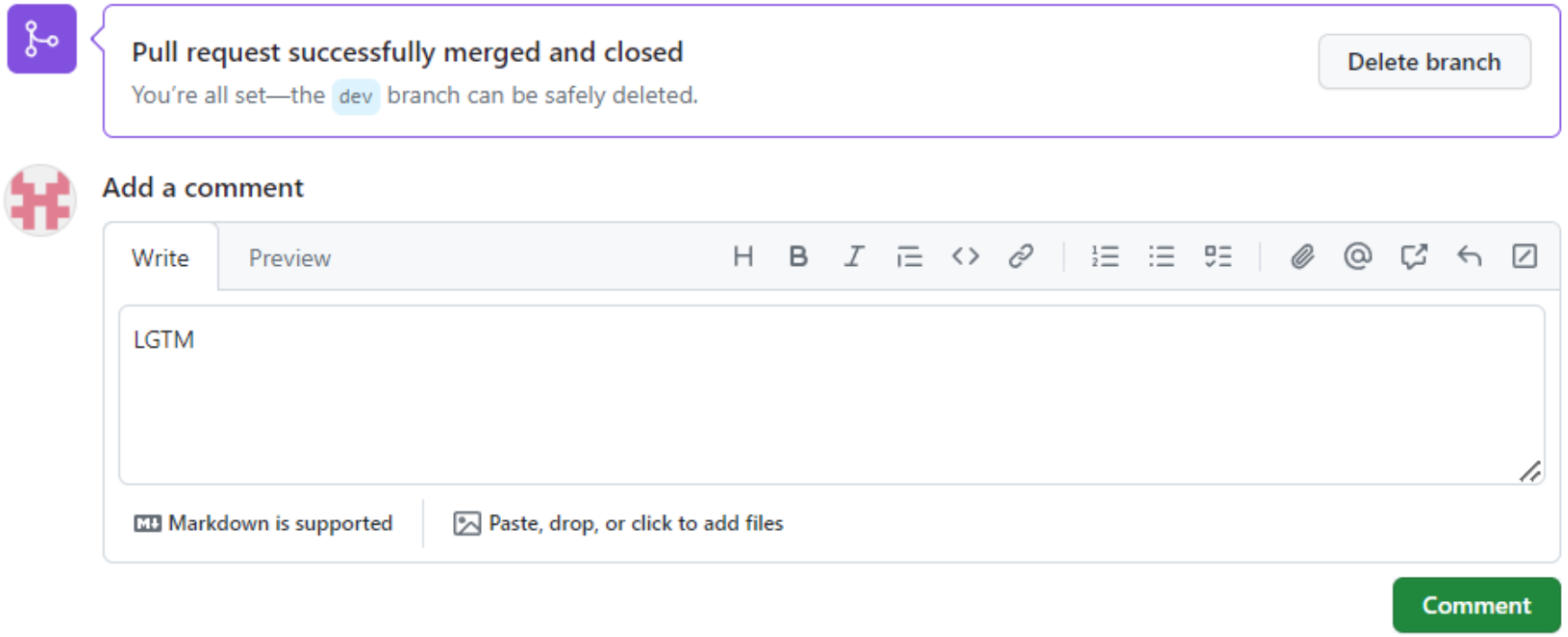




Git & GitHub 심화 (GUI)



Git & GitHub 심화 (GUI)



TIP

LGTM은 "Looks Good To Me"의 약어로,
다른 사람이 제안한 것이나 코드 변경 사항 등을 검토한 후 "이것은 좋아 보입니다"라는 뜻으로 사용

Git & GitHub 심화 (GUI)

- Code Review는 코드를 기반으로 피드백을 주고 받는 과정이다.
- 기술 부채를 줄이고, 코드의 부작용과 오류를 사전에 찾아낼 수 있도록 해주는 과정이기 때문에 반드시 필요한 과정이다.
- 만약 Code Review할 게 없다면? 칭찬해주기 👍

TIP

AFAIK	- “As Far As I Know” [내가 알기로는~]
IMO (IMHO)	- “In My (Humble) Opinion” [내 생각에는 ~]
FYI	- “For Your Information” [참고로 ~]
GOTCHA	- “I’ve Got You” [알았다!]
SSIA	- “Subject Says It All” [제곧내]
TIA	- “Thanks In Advance” [미리 감사드립니다]

Git & GitHub 심화 (GUI)

- 프로그래밍을 하면 Naming Convention(네이밍 컨벤션)을 지키게 되는데,
  네이밍 컨벤션이란 변수, 함수 등의 이름을 짓는 방식을 의미한다.
- Commit 메시지 또한 협업에 알맞게 커뮤니케이션에 유용하게, 깔끔한 가독성을 갖도록
  Commit Convention(커밋 컨벤션)을 만들고 지켜 나가는 것이 좋다.

Feat	새로운 기능을 추가	- feat: 로그인 기능 추가
Fix	버그 수정	- fix: 로그인 기능에 null 값 체크 로직 추가
Style	코드 포맷팅	- style: 들여쓰기 개선
Refactor	프로덕션 코드 리팩토링	- refactor: 로그인 로직 간소화
Comment	필요한 주석 추가 및 변경	- comment: 로그인 함수 주석 추가
Docs	문서 수정	- docs: API 문서에 로그인, 로그아웃 기능 설명 추가
Rename	파일 혹은 폴더명을 수정하거나 옮기는 작업만인 경우	- rename: user_service.java > UserService.java
Remove	파일을 삭제하는 작업만 수행한 경우	- remove: UserService2.java

MarkDown

- MarkDown은 텍스트 기반의 마크업 언어로, 문서를 포맷팅하고 스타일을 지정하는 간단한 방법을 제공한다.
- Github과 같은 버전 관리 플랫폼에서 주로 사용한다.

MarkDown

- MarkDown의 기본문법

# 큰 제목	**큰 제목**
## 작은 제목	**작은 제목**

TIP

TAB 키를 이용해 하위 항목을 생성할 수 있고,
SHIFT + TAB 키를 눌러서 상위 항목으로 이동 가능

1. 순서가 있는 목록 : 1.을 누르고 스페이스바를 눌러 생성.

- 순서가 없는 목록 : - 또는 * 을 쓰고 스페이스바를 눌러 생성.

MarkDown

- MarkDown의 기본문법

코드 블록

인라인 코드 블록 : 인라인 블럭으로 처리하고 싶은 부분을 `(백틱)으로 감싼다.

코드 블록 : `(백틱) 을 세 번 입력하고 Enter 를 눌러 생성한다.

MarkDown

- MarkDown의 기본문법

링크

[]()를 작성하고 () 안에 링크 주소를 작성하고 []안에 어떤 링크 주소인지 입력

이미지

를 작성하고, () 안에는 이미지 주소를 입력

MarkDown

- MarkDown의 기본문법

표

|(파이프) 사이에 컬럼을 작성하고 엔터를 입력

마지막 컬럼을 작성하고 |(파이프) 입력

강조

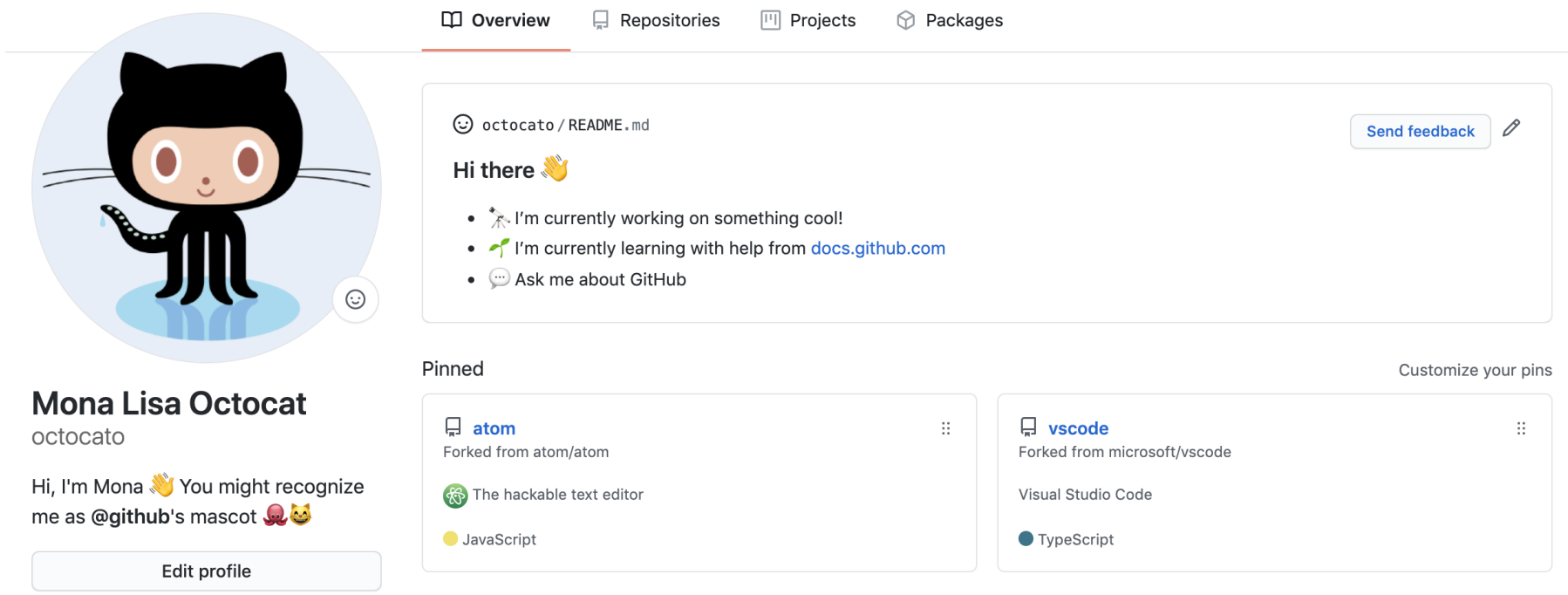
이탤릭체 : * 혹은 _ 로 감싸준다.

볼드체 : ** 또는 __로 감싸준다.

취소선: ~~으로 앞 뒤를 감싸준다.

MarkDown

- 자신의 UserName과 동일한 Repository를 생성하고, README.md 파일을 꾸미면, 자기소개 화면을 만들 수 있다.



MarkDown

- 개발 프로젝트의 README.md는 어떻게 구성하면 좋을까?

  ⭐ New Features

  🐞 Bug Fixes

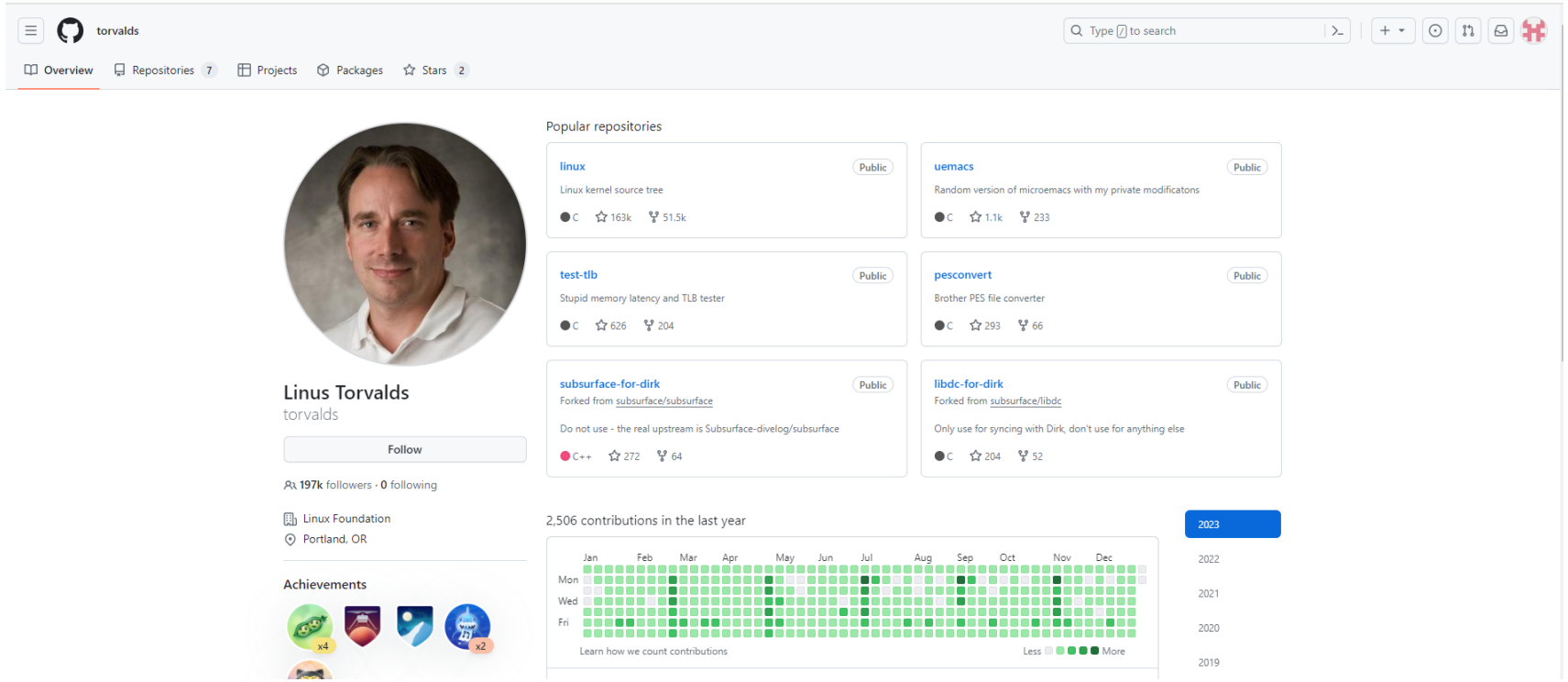
  📔 Documentation

  🔨 Dependency Upgrades

  ❤️ Contributors

MarkDown

- GitHub의 잔디는 성실한 개발자의 척도가 되어준다.



GitHub Pages 만들기

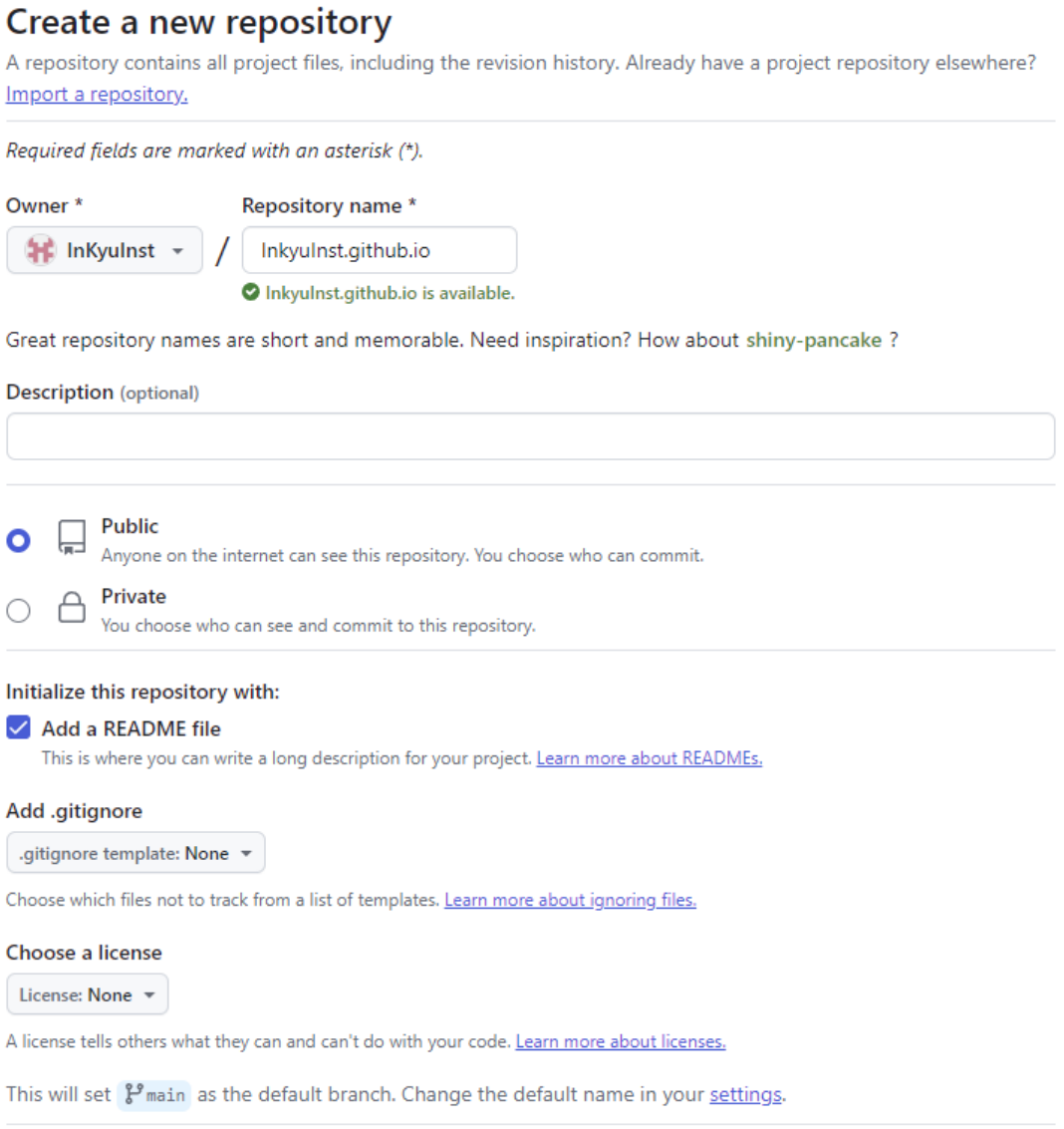
- GitHub Pages는 GitHub의 리포지토리에서 HTML, CSS 및 JavaScript 파일을 직접 가져와서
  필요에 따라 빌드 프로세스를 통해 파일을 실행하고 웹 사이트를 게시하는 정적 사이트 호스팅 서비스이다.

- 이를 통해 나만의 웹 페이지를 무료로 만들 수 있다.

- GitHub Pages는 사용자 사이트, 프로젝트 사이트, 조직 사이트를 제공하는데
  각 계정 당 하나의 사용자 또는 조직 사이트를 만들 수 있으며, 프로젝트 사이트는 제한이 없이 이용 가능하다.

GitHub Pages 만들기

- 생성 후 브라우저에서 <사용자이름>.github.io 접속



GitHub Pages 만들기

- 생성 후 브라우저에서 <사용자이름>.github.io 접속

```
Username@DESKTOP MINGW64 ~/Desktop
$ git clone https://github.com/사용자이름/사용자이름.github.io.git

Username@DESKTOP MINGW64 ~/Desktop
$ cd 사용자이름.github.io

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (main)
$ rm -rf *

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (main)
$ code .
```

GitHub Pages 만들기

- index.html 및 css 작성하기

```html
<!DOCTYPE html>
<html lang="ko">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>자기소개 페이지</title>
    <link rel="stylesheet" href="index.css">
</head>
<body>
    ~~~~~~
</body>
</html>
```

```css
body {
    margin: 0;
    padding: 0;
    line-height: 1.6;
    background-color: #f4f4f4;
}

~~~
```

GitHub Pages 만들기

```
Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (main)
$ git add .

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (main)
$ git commit -m 'add index.html and index.css'

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (main)
$ git push -u origin main
```

GitHub Pages 만들기

- 생성 후 브라우저에서 <사용자이름>.github.io 접속

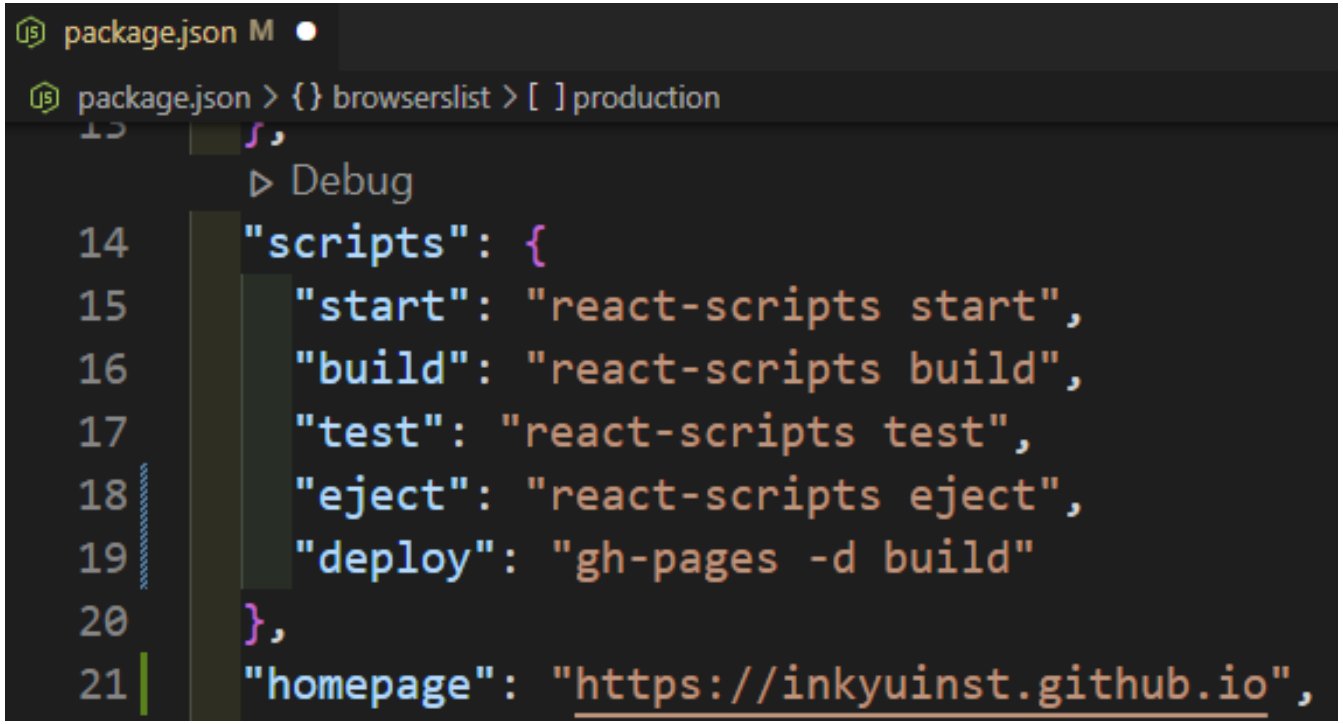
```
Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (main)
$ git switch -c react
Switched to a new branch 'react'

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ rm -rf *

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn create react-app .

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn add gh-pages -D
```

GitHub Pages 만들기

- 생성 후 브라우저에서 <사용자이름>.github.io 접속

```
Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (main)
$ git switch -c react
Switched to a new branch 'react'

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ rm -rf *

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn create react-app .

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn add gh-pages -D
```

GitHub Pages 만들기



GitHub Pages 만들기

```
Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ git add .

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ git commit -m '<커밋메시지>'

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ git push -u origin react

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn build

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn deploy
```

GitHub Pages 만들기

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at https://inkyuinst.github.io/
Last deployed by InKyuInst 38 minutes ago

Visit site ···

Build and deployment

Source

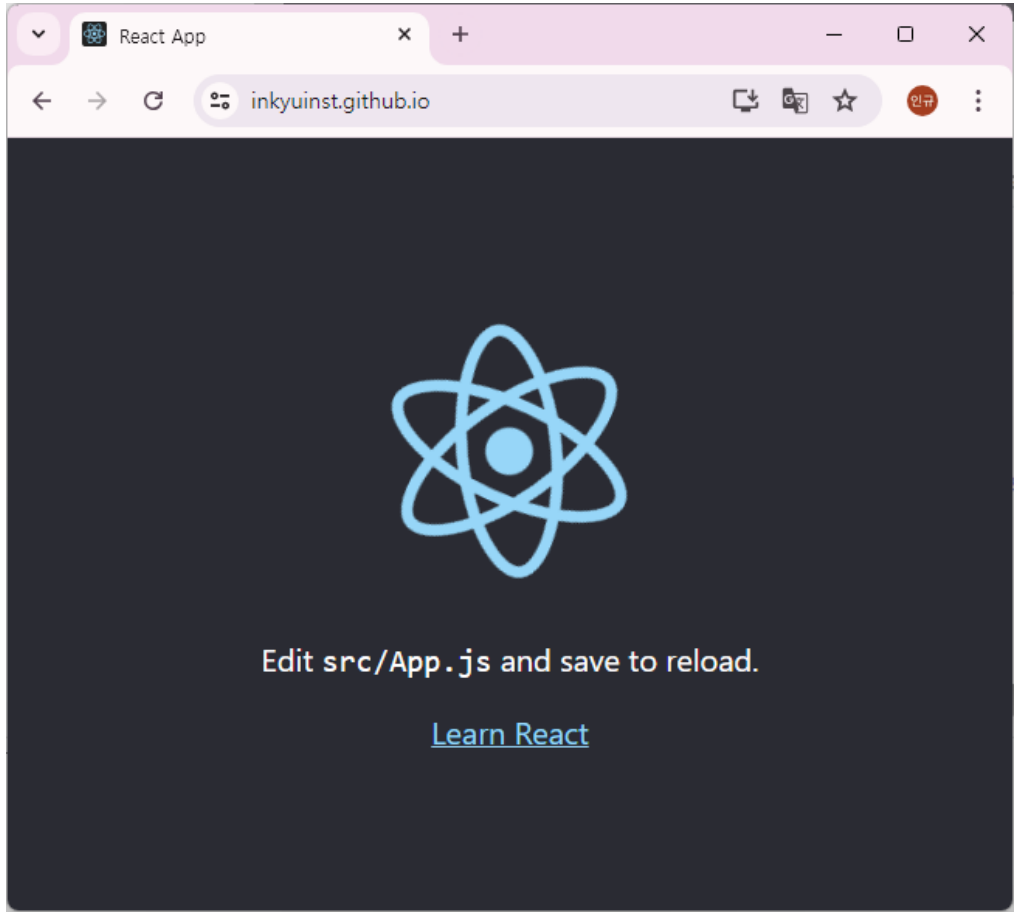
Deploy from a branch

Branch

Your GitHub Pages site is currently being built from the `main` branch. Learn more about configuring the publishing source for your site.

gh-pages / (root) Save

GitHub Pages 만들기



GitHub Pages 만들기

- React 코드 수정 후

```
Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn build

Username@DESKTOP MINGW64 ~/Desktop/inkyuinst.github.io (react)
$ yarn deploy
```

시작하기 전에

- 완성된 애플리케이션은 배포되어 서비스되어야 한다.

- 혼자가 아닌 수많은 개발자가 모인다면, 코드를 합치고, 빌드하고, 테스트하고, 배포되는 과정이 반복된다.
  그리고 해당 과정을 수동으로 하나씩 진행하는 것은 많은 단점이 있다.

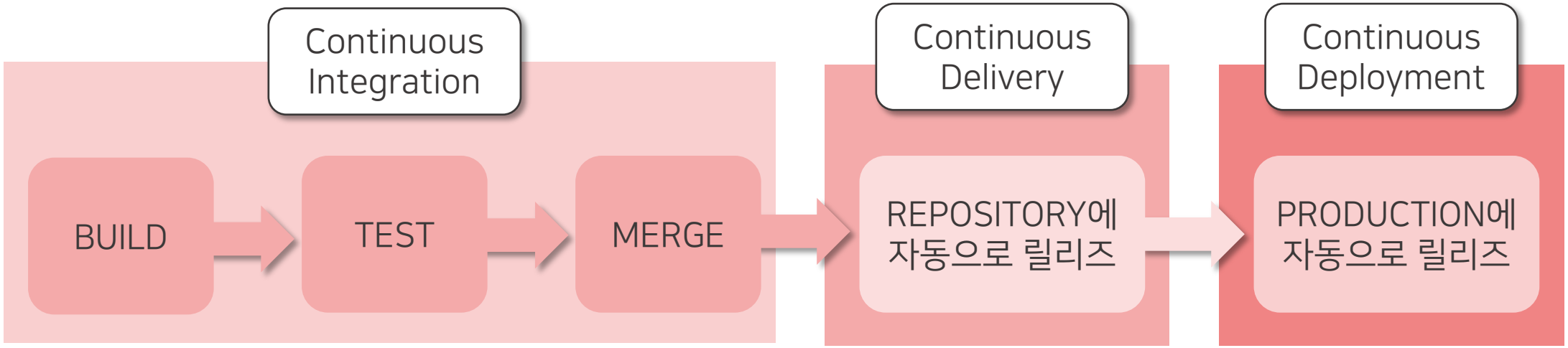
  - 각 과정에 시간이 오래 걸리며,
  - 오류 발생 가능성이 높아져 서비스 품질이 낮아질 수 있으며,
  - 개발자의 생산성이 저하된다.

TIP

빌드(Build) : 컴퓨터에서 실행할 수 있는 소프트웨어 산출물로 만드는 과정
배포(Deploy) : 빌드의 결과물을 사용자가 접근할 수 있는 환경에 배치하는 것

CI/CD

- 아래와 같이 빌드부터 배포까지의 일련의 과정을 CI/CD 라고 한다.



(지속적 통합)
새로운 코드 변경 사항이
빌드 및 테스트가 되어 합쳐지는 것

(지속적 제공)
최종 코드 산출물을
Repository에 저장하는 것

(지속적 배포)
실제 서비스에
배포하는 것

CI/CD

- CI : Continuous Integration (지속적인 통합)
  - 코드 변경사항을 주기적으로 빈번하게 통합해야 한다.
  - 통합을 위한 Build, Test, Merge 단계를 자동화한다.

- CI의 목적은 소프트웨어 개발 과정에서의 효율성, 품질, 협업을 향상시키는 것에 있다.
  - 작은 변경이라도 즉시 통합하고 테스트하여 오류를 조기에 발견한다.
  - 통합 및 테스트 주기를 단축하여 개발 속도를 높인다.
  - 자동화된 테스트로 코드의 품질을 유지하고 향상시킨다.
  - 다수의 개발자가 동시에 작업할 때, 코드 통합 문제를 줄이고 협업을 원활하게 한다.

CI/CD

- Continuous Delivery과 Continuous Deployment는 최종단계인 배포를 자동으로 하는지, 수동으로 하는지에 따라 달라질 수 있다.

- CD의 목적은 소프트웨어를 빠르고 안정적으로 사용자에게 제공하는 것에 있다.

    - 소프트웨어의 새로운 기능이나 수정 사항을 신속하게 사용자에게 제공한다.
    - 자동화된 배포 과정으로 일관된 배포를 보장한다.
    - 잦은 배포로 대규모 배포의 리스크를 줄일 수 있다.