

<복제물에 대한 경고>

본 저작물은 **저작권법 제25조** 수업목적 저작물 이용 보상금제도에 의거, **한국복제전송저작권협회**와 약정을 체결하고 적법하게 이용하고 있습니다. 약정범위를 초과하는 사용은 저작권법에 저촉될 수 있으므로

저작물의 재 복제 및 수업 목적 외의 사용을 금지합니다.

2020. 03. 30.

건국대학교(서울)한국복제전송저작권협회

<전송에 대한 경고>

본 사이트에서 수업 자료로 이용되는 저작물은 **저작권법 제25조** 수업목적 저작물 이용 보상금제도에 의거,

한국복제전송저작권협회와 약정을 체결하고 적법하게 이용하고 있습니다.

약정범위를 초과하는 사용은 저작권법에 저촉될 수 있으므로

수업자료의 대중 공개 공유 및 수업 목적 외의 사용을 금지합니다.

2020. 03. 30.

건국대학교(서울)한국복제전송저작권협회



CHAPTER 04

커밋

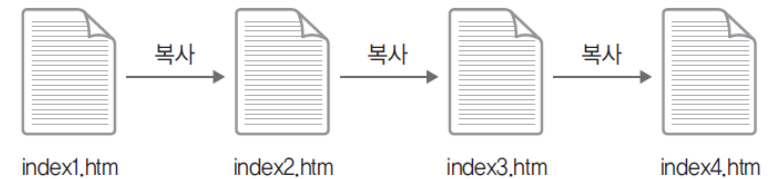
4.1 코드의 변화

1. 코드의 변화



> 파일 관리 방법

- 보통 우리는 의미 있는 변경을 할 때 파일을 복사함
- 복사한 새 파일에는 추가하거나 변경하고 싶은 내용을 적용함
- 파일을 복사하는 형태는 파일의 변경 내역을 기록하는 것보다 더 많은 파일을 생성하고 관리해야 하는 부작용이 있음
- 모든 내용이 중복되기 때문에 용량도 많이 차지함



1. 코드의 변화

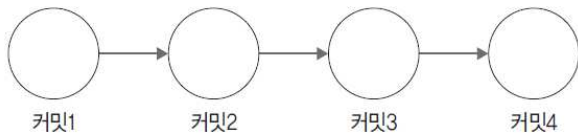


➤ 파일 관리 방법

- 깃의 커밋은 새로 변경된 부분만 추출하여 저장함
- 파일 이름을 변경하지 않고도 동일한 파일 이름으로 하나로 관리가 가능함
- 커밋:
시간에 따라 변화되는 내용만 관리하고, 코드가 변화된 시간 순서에 따라서 영구적으로 저장함



index1.htm



4.2 새 파일 생성 및 감지

6

2. 새 파일 생성 및 감지



➤ 새 파일 생성

- 실습을 위해 간단한 HTML 파일을 하나 작성함
- 에디터를 이용하여 코드를 작성하면 됨
- 필자는 VS Code를 이용함

```
$ mkdir gitstudy04 ----- 새 폴더 만들기
```

```
$ cd gitstudy04 ----- 만든 폴더로 이동
```

```
$ git init ----- 저장소를 깃으로 초기화
```

```
Initialized empty Git repository in E:/gitstudy04/.git/
```

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ code index.htm ----- VS Code를 사용하여 파일 작성
```

2. 새 파일 생성 및 감지



➤ 새 파일 생성

index.htm

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Page Title</title>
</head>
<body>

</body>
</html>
```

2. 새 파일 생성 및 감지



➤ 깃에서 새 파일 생성 확인

- 워킹 디렉터리에 새 파일이 생성됨
- 워킹 디렉터리에 새 파일이 추가되면 깃은 변화된 상태를 자동으로 감지함
- 이때 깃 상태를 확인할 수 있는 명령어가 status임
- status 명령어를 입력하면 메시지가 출력되는 것을 볼 수 있음

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git status ----- 상태 확인
```

```
On branch master
```

```
No commits yet
```

Untracked files:

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

```
index.htm ----- 새로운 파일이 등록된 것을 확인
```

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

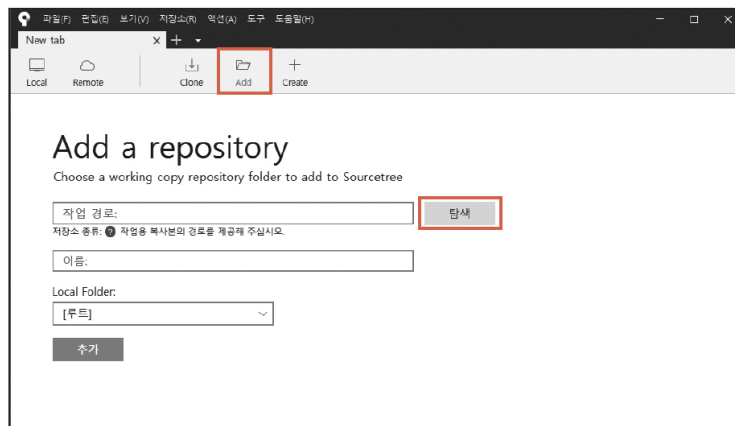
2. 새 파일 생성 및 감지



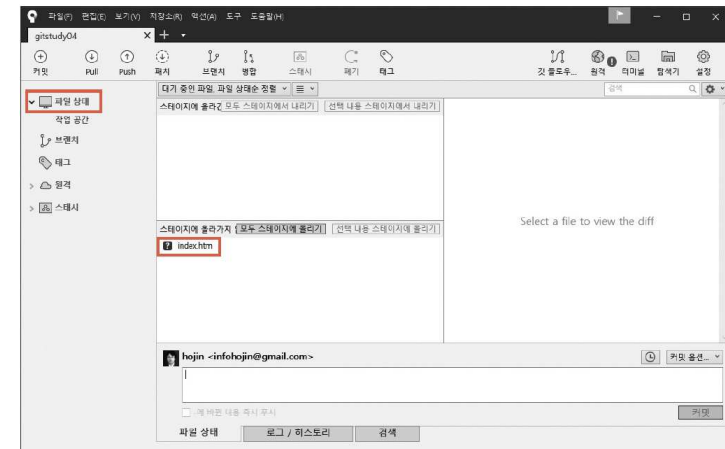
➤ 소스트리에서 새 파일 감지

- 소스트리를 사용하여 깃의 status 명령어와 동일한 상태를 확인할 수 있음
- 아직 gitstudy04 폴더와 소스트리를 연동하지 않음
- 새 탭에서 Add 버튼을 클릭함
- 탐색을 눌러 깃 배스에서 만든 gitstudy04 폴더를 찾아 선택한 후 추가를 누름

2. 새 파일 생성 및 감지



2. 새 파일 생성 및 감지



4.3 깃에 새 파일 등록

13

3. 깃에 새 파일 등록



➤ 깃에 새 파일 등록

- 워킹 디렉터리에 있는 파일은 깃이 자동으로 추적 관리하지 않음
- 커밋을 하려면 **파일의 상태가 추적 가능해야 함**
- 등록:
 - 워킹 디렉터리에 새로 추가된 untracked 상태의 파일을 추적 가능 상태로 변경하는 것
- 파일을 등록하면 워킹 디렉터리의 파일이 스테이지 영역에 추가됨
- 스테이지 영역의 관리 목록에 추가된 파일만 깃에서 이력을 추적할 수 있음



3. 깃에 새 파일 등록



➤ 스테이지에 등록

명령어로 등록: add 명령어

- 현재는 커밋 명령어를 실행하기 이전의 중간 단계임
- 깃의 add 명령어는 워킹 디렉터리의 파일을 스테이지 영역으로 등록
- 깃은 안정적인 커밋을 할 수 있도록 add 명령어를 기준으로 이전과 이후 단계를 구별함
- 터미널에서는 다음 형태의 명령어를 입력

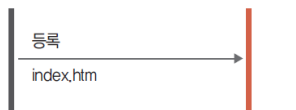
```
$ git add 파일이름
```

infoh@hojin MINGW64 /e/gitstudy04 (master) 워킹 디렉터리

```
$ git add index.htm ----- 스테이지에 등록
```

- 전체 파일 등록

```
$ git add .
```




3. 깃에 새 파일 등록



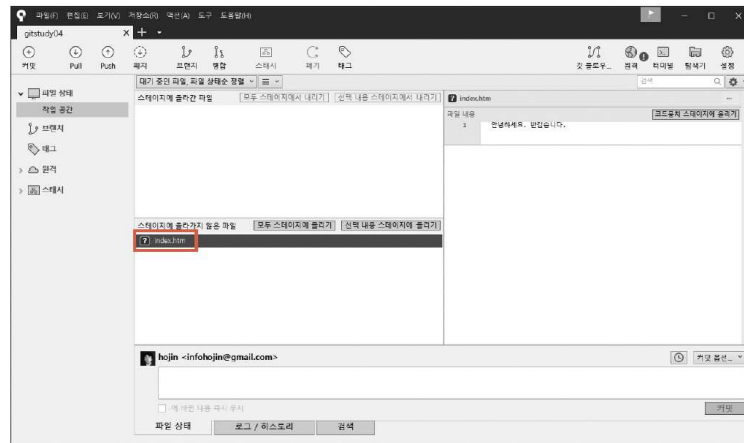
➤ 스테이지에 등록

소스트리에서 등록

- 소스트리는 스테이지 영역에 등록되는 파일을 직관적으로 확인할 수 있음
- untracked 상태인 파일은 소스트리의 **스테이지에 올라가지 않은 파일** 영역에서 확인할 수 있음
- 파일을 선택하여 상위 영역의 **스테이지에 올라간 파일** 부분으로 옮김
- 소스트리는 추적 상태와 추적하지 않음을 쉽게 구별할 수 있도록 파일 이름 앞에 아이콘을 함께 표시함
- 보라색 아이콘  index.htm 은 untracked 상태의 파일임
- 새로 생성된 파일을 의미하기도 함

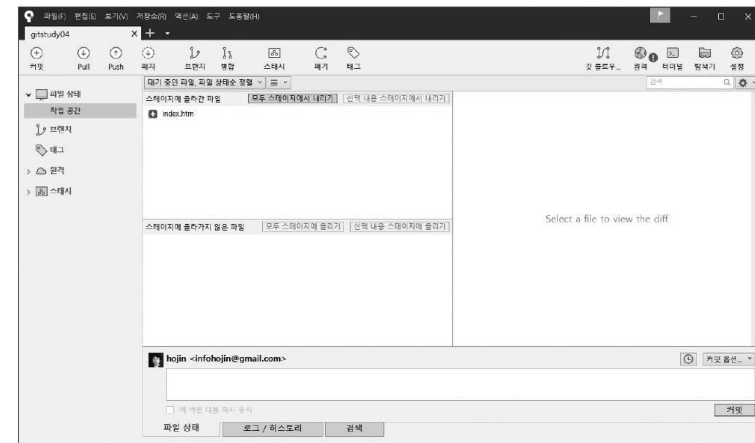
3. 깃에 새 파일 등록

Git
교과서



3. 깃에 새 파일 등록

Git
교과서



3. 깃에 새 파일 등록

Git
교과서



➤ 파일 등록 취소

- 워킹 디렉터리에 있는 새로운 파일이 스테이지 영역에 등록됨
infoh@hojin MINGW64 /e/gitstudy04 (master)
\$ git status ----- 상태 확인
On branch master
No commits yet
Changes to be committed:
 (use "git rm --cached <file>..." to unstage)
 new file: index.htm ----- 스테이지에 등록, 새 파일 상태
- 파일을 등록한 후 커밋하지 않고 바로 삭제(워킹디렉토리에 원본은 두고 스테이지 영역에서만 삭제)하려면 rm --cached 명령어를 사용함
infoh@hojin MINGW64 /e/gitstudy04 (master)
\$ git rm --cached index.htm
rm 'index.htm'

3. 깃에 새 파일 등록

Git
교과서



➤ 파일 등록 취소

- 스테이지의 캐시 목록에서 파일이 삭제됨
infoh@hojin MINGW64 /e/gitstudy04 (master)
\$ git status ----- 상태 확인
On branch master
No commits yet
Untracked files: ----- 추적하지 않음
 (use "git add <file>..." to include in what will be committed)
 index.htm ----- 스테이지 삭제
nothing added to commit but untracked files present (use "git add" to track)

3. 깃에 새 파일 등록



➤ 등록된 파일 이름이 변경되었을 때

- 작업 도중 파일 이름도 변경할 수 있음
- 깃에서도 파일 이름을 변경할 때 mv 명령어를 사용함

```
$ git mv 파일이름 새파일이름
```

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git mv index.htm home.htm ----- 파일 이름 변경
```

```
$ git status
```

```
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   home.htm ----- 변경된 파일 이름
```

```
$ mv index.htm home.htm
```

```
$ git rm index.htm
```

```
$ git add home.htm
```

4.4 첫 번째 커밋

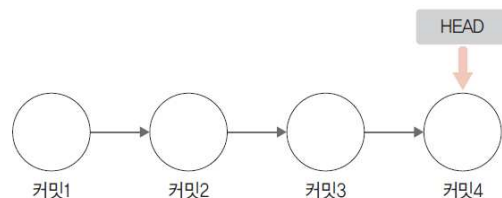
22

4. 첫 번째 커밋



➤ HEAD

- HEAD는 최종적인 커밋 작업의 위치를 가리킴
- 깃을 설치하고 처음 커밋할 때는 HEAD의 포인터가 없음
- 최소한 한 번 이상 커밋을 해야만 HEAD가 존재함
- HEAD는 커밋될 때마다 한 단계씩 이동함
- 마지막 커밋 위치를 가리킴

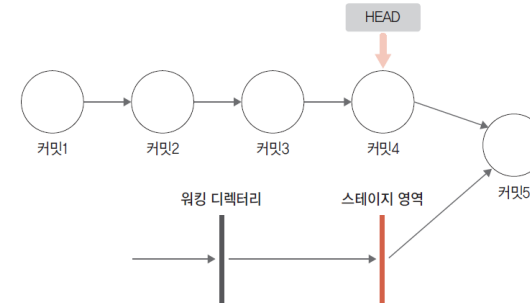


4. 첫 번째 커밋



➤ 스냅샷

- 깃은 이러한 시스템적인 단점을 해결하려고 변경된 파일 전체를 저장하지 않고, 파일에서 변경된 부분을 찾아 수정된 내용만 저장함
- 깃의 스냅샷은 HEAD가 가리키는 커밋을 기반으로 사진을 찍음
- 스테이지 영역과 비교하여 새로운 커밋으로 기록함
- 깃은 스냅샷 방식을 이용하여 빠르게 버전의 차이점을 처리하고, 용량을 적게 사용함



4. 첫 번째 커밋



➤ 파일 상태와 커밋

- 커밋하기 전에는 status 명령어로 항상 상태를 확인하는 습관이 필요함
- 워킹 디렉터리가 깨끗하게 정리되어 있지 않으면 커밋 명령어를 수행할 수 없음
- 커밋을 하려면 스테이지 영역에 새로운 변경 내용이 있어야 함
- 수정된 내용이 스테이지 영역으로 등록되지 않으면 커밋을 할 수 없음
- 커밋은 수정된 내용을 한 번만 등록함
- 깃은 스테이지 영역의 변경된 내용을 기준으로 스냅샷을 만들어 커밋하기 때문에 스테이지 영역의 파일이 변경되지 않았다면 커밋을 두 번 실행할 수 없음

4. 첫 번째 커밋



➤ 파일 상태와 커밋

명령어로 커밋: commit 명령어

- 수정된 파일 이력을 커밋하려면 commit 명령어를 사용함

```
$ git commit
```

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git commit -help
```

```
usage: git commit [<options>] [--] <pathspec>...
```

-q, --quiet	suppress summary after successful commit
-v, --verbose	show diff in commit message template

Commit message options

4. 첫 번째 커밋



➤ 파일 상태와 커밋

- "git commit" 명령어를 입력하면 커밋 메시지 작성을 요구하며, 메시지를 작성할 수 있는 화면이 나오며, 지정된 에디터인 vi가 열림

```
MINGW64: e/gitstudy04
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   new file:   index.htm
#
~
~
~
<test_git/g1/.git/COMMIT_EDITMSG [unix] (18:27 04/01/2019)1, 0-1 모두
```

4. 첫 번째 커밋



➤ 파일 상태와 커밋

- 원하는 메시지를 입력한 후 저장함
- vi 에디터에서 새로운 내용을 입력할 때는 [Esc]를 누른 후 [i]를 누름
- 작성한 후 저장과 종료는 [Esc]를 누른 후 [:+w, q]를 입력함
- 가끔씩 커밋 메시지를 작성하다 vi 에디터를 중지하고 싶을 때도 있을 것임
- 이때는 아무것도 작성하지 않고 [Esc]를 누른 후 [:+q]를 누름

- vi 에디터에서 커밋 메시지를 작성할 때는 요약 내용과 상세 내용을 분리하여 기록하면 좋음
- 보통 첫째 줄에는 '제목'을 적고, 다음 줄에는 상세 내용을 작성하곤 함
- 중간에 빈 줄로 구분해 주는 것도 좋음
- 첫째 줄을 분리하여 작성하는 것은 로그 출력을 간단하게 하기 위해서임
- 소스트리나 일부 간략한 로그들은 커밋 메시지의 첫째 줄만 표시하기 때문임

4. 첫 번째 커밋



➤ 파일 상태와 커밋

파일 등록과 커밋을 동시에 하는 방법

- 커밋을 하려면 반드시 워킹 디렉터리를 정리해야 함
- add 명령어로 추가되거나 수정된 파일들을 스테이지 영역에 등록해야 함
- 가끔씩 add 명령어를 미리 수행하는 것을 깜빡 잊을 때가 있는데, commit 명령어를 바로 수행하면 오류가 발생함
- -a 옵션을 commit 명령어와 같이 사용하면 이를 한 번에 해결할 수 있음

```
$ git commit -a
```

- -a 옵션은 커밋을 하기 전에 자동으로 모든 파일을 등록하는 과정을 미리 수행함
- 파일 등록과 커밋을 동시에 실행하는 것임

4. 첫 번째 커밋



➤ 파일 상태와 커밋

소스트리에서 커밋 메시지 작성

- 소스트리를 이용하면 좀 더 쉽게 커밋할 수 있음
- 소스트리에서 왼쪽의 파일 상태 탭을 선택하면 아래쪽과 같이 커밋 메시지를 입력할 수 있는 창이 열림



4.5 커밋 확인

5. 커밋 확인



➤ 스테이지 초기화

- 먼저 터미널에서 status 명령어를 실행해서 상태를 확인함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git status ----- 상태 확인
```

```
On branch master
```

```
nothing to commit, working tree clean ----- 워킹 트리 정리됨
```

- 이전과 달리 working tree clean 메시지를 볼 수 있음
- 커밋을 하면 스테이지 영역은 초기화됨
- 더 이상 추가된 새로운 파일과 수정된 파일이 없다는 의미임
- 항상 커밋 전후에 status 명령어로 상태를 확인하는 것이 좋음

5. 커밋 확인



로그 기록 확인

- 깃은 커밋 목록을 확인할 수 있는 log 명령어를 별도로 제공함

```
$ git log
```

- log 명령어는 시간 순으로 커밋 기록을 출력하는데, 최신 커밋 기록부터 내림차순으로 나열함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git log
```

```
commit e2bce41380691b0a34aeab7db889a6c30fed8287 (HEAD -> master)
```

```
Author: hojin <infohojin@gmail.com>
```

```
Date: Sat Jan 5 18:24:50 2019 +0900
```

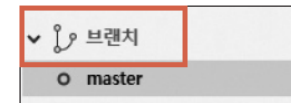
인덱스 페이지 레이아웃

5. 커밋 확인



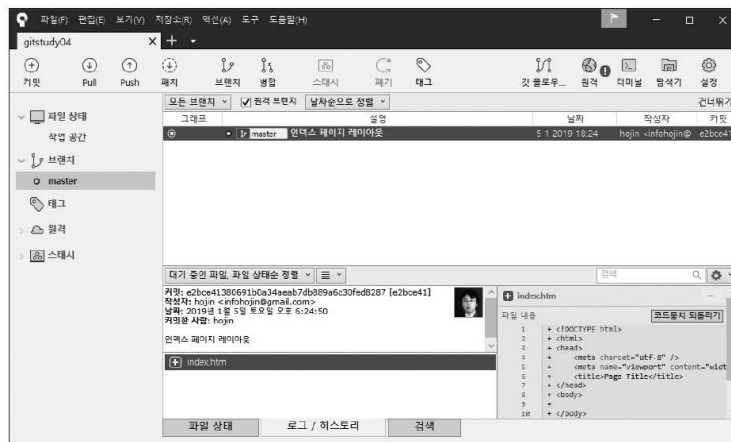
소스트리에서 로그 기록 확인

- 터미널로 로그 기록을 확인하는 것은 가독성이 좋지 않음
- 커밋 횟수가 많을수록 보기도 어려움
- 소스트리를 이용하면 좀 더 직관적으로 커밋 기록을 확인할 수 있음



- 깃을 처음 생성하면 자동으로 master 브랜치 1개를 생성함
- 커밋은 master 브랜치 안에 기록됨

5. 커밋 확인



4.6 두 번째 커밋

6. 두 번째 커밋

Git 교과서



➤ 파일 수정

- index.htm 파일의 <body></body> 태그 안에 <h1> 태그를 추가하여 간단한 인사말을 넣음

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ code index.htm ----- VS Code 실행
```

index.htm

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Page Title</title>
</head>
<body>
  <h1>hello GIT world!</h1>
</body>
</html>
```

6. 두 번째 커밋

Git 교과서



➤ 파일 변경 사항 확인

- 터미널에서 status 명령어를 다시 한 번 실행함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git status ----- 상태 확인
```

On branch master

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

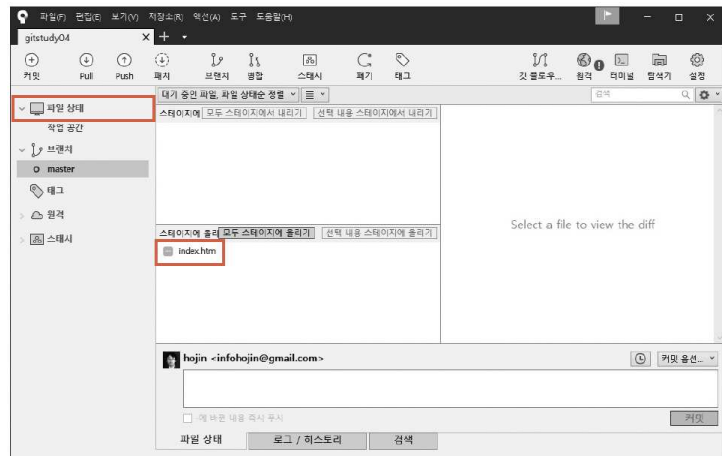
(use "git checkout -- <file>..." to discard changes in working directory)

modified: index.htm ----- 파일 수정

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

6. 두 번째 커밋

Git 교과서



6. 두 번째 커밋

Git 교과서



➤ 수정된 파일 되돌리기

- 깃을 이용하면 수정한 파일을 커밋 전 마지막 내용으로 쉽게 되돌릴 수 있음

```
$ git checkout -- 수정파일이름
```

- 수정 파일을 되돌리면 이전 커밋 이후에 작업한 수정 내역은 모두 삭제함

6. 두 번째 커밋

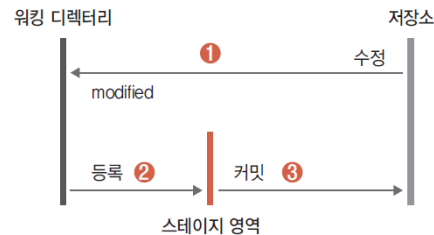
Git
교과서



➤ 스테이지에 등록

- ① 기존 파일을 수정하면 해당 파일은 modified 상태로 변경됨
- 다시 워킹 디렉터리로 이동함
- ② 파일이 수정되면 반드시 **add 명령어로 스테이지 영역에 재등록**해야 함
- 파일을 수정할 때마다 **등록 작업을 반복**해야 한다는 것
- 소스트리에서 모두 스테이지에 올리기를 사용함

```
$ git add 수정파일이름
```



6. 두 번째 커밋

Git
교과서



➤ 스테이지에 등록

- 수정한 파일을 스테이지에 재등록함
- 다시 한 번 status 명령어를 실행함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git add index.htm ----- 스테이지에 재등록
$ git status ----- 상태 확인

On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.htm ----- 파일 수정
```

- 수정된 파일 이름이 빨간색에서 녹색으로 변경된 것을 확인할 수 있음

6. 두 번째 커밋

Git
교과서



➤ 두 번째 커밋

- vi 에디터나 소스트리에서는 커밋 메시지를 여러 줄 작성할 수 있음
- 커밋과 동시에 간단하게 한 줄짜리 커밋 메시지도 작성할 수 있음
- 커밋할 때는 -m 옵션을 사용함
- 실제 작업할 때는 에디터를 여는 대신 간편한 -m 옵션을 많이 사용함

```
$ git commit -m "커밋메시지"
```

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git commit -m "hello git world 추가" ----- 커밋 메시지를 같이 입력
[master aa1dd51] hello git world 추가
1 file changed, 1 insertion(+), 1 deletion(-)
```

6. 두 번째 커밋

Git
교과서



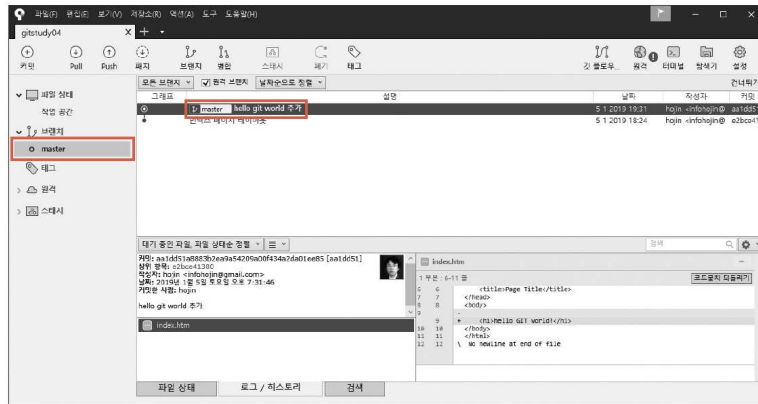
➤ 두 번째 커밋 확인

- 터미널에서 log 명령어를 실행함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git log ----- 로그 확인
commit aa1dd51a8883b2ea9a54209a00f434a2da01ee85 (HEAD -> master)
Author: hojin <infohojin@gmail.com>
Date:   Sat Jan 5 19:31:46 2019 +0900
    hello git world 추가 ----- 추가된 커밋 확인

commit e2bce41380691b0a34aeb7db889a6c30fed8287
Date:   Sat Jan 5 18:24:50 2019 +0900
    인덱스 페이지 레이아웃
```

6. 두 번째 커밋



4.7 메시지가 없는 빈 커밋

46

7. 메시지가 없는 빈 커밋



➤ 메시지가 없는 빈 커밋

- 커밋을 할 때는 반드시 커밋 메시지를 같이 작성해야 함
- 의미가 없는 커밋이라 커밋 메시지를 생략하고 싶을 때도 있음
- 빈 커밋:
특별한 상황에 대비하여 깃은 메시지가 없는 커밋 작성도 허용함

7. 메시지가 없는 빈 커밋



➤ 세 번째 커밋

- 실습을 위해 index.htm 파일 내용을 좀 더 수정하여 세 번째 커밋을 하겠음
 - 간략하게 <title>~</title> 사이의 내용을 JINYGIT으로 수정한 후 저장함
- ```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ code index.htm ----- VS Code 실행
```

index.htm

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <title>JINYGIT</title>
</head>
<body>
 <h1>hello GIT world!</h1>
</body>
</html>
```

## 7. 메시지가 없는 빈 커밋



### ➤ 세 번째 커밋

- 파일을 수정한 후에는 반드시 다음과 같이 다시 수정된 파일을 스테이지 영역에 **재등록**함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git add index.htm ----- 스테이지에 등록
```

- 터미널에서 메시지가 없는 빈 커밋을 작성하려면 --allow-empty-message 옵션을 사용함

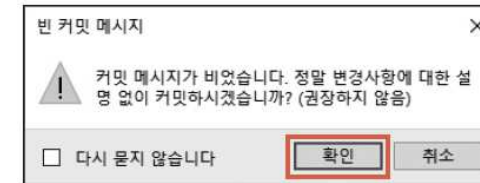
```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git commit --allow-empty-message -m "" ----- 커밋 메시지를 작성하지 않음
[master 42250c6] ----- 빈 커밋
1 file changed, 1 insertion(+), 1 deletion(-)
```

## 7. 메시지가 없는 빈 커밋



### ➤ 소스트리에서 빈 커밋

- 소스트리에서는 메시지가 없는 빈 커밋을 하기가 더 쉬움
- 예를 들어 **파일 상태** 탭을 선택한 후 아래쪽의 커밋 메시지 입력란을 비워 두고 **커밋**을 누르면 됨
- 경고문을 알리는 것은 커밋 작업이 반드시 메시지를 작성하는 것을 원칙으로 하고 있기 때문임



## 7. 메시지가 없는 빈 커밋



### ➤ 빈 커밋 확인

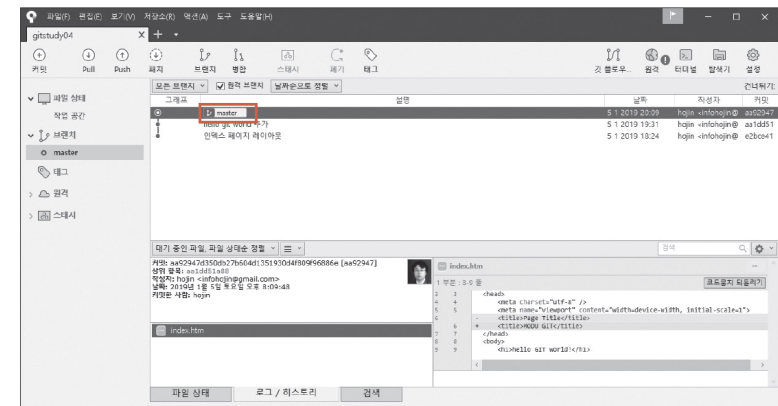
- 터미널에서 log 명령어를 입력함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git log ----- 로그 확인
commit aa92947d350db27b604d1351930d4f809f96886e (HEAD -> master)
Author: hojin <infohojin@gmail.com> ----- 빈 커밋
Date: Sat Jan 5 20:09:48 2019 +0900
```

```
commit aa1dd51a8883b2ea9a54209a00f434a2da01ee85
Author: hojin <infohojin@gmail.com>
Date: Sat Jan 5 19:31:46 2019 +0900
 hello git world 추가
```

```
commit e2bce41380691b0a34aeb7db889a6c30fed8287
Author: hojin <infohojin@gmail.com>
Date: Sat Jan 5 18:24:50 2019 +0900
 인덱스 페이지 레이아웃
```

## 7. 메시지가 없는 빈 커밋



## 4.8 커밋 아이디

53

## 8. 커밋 아이디



### > 커밋 아이디

- 다음과 같이 터미널에서 log 명령어를 실행하면 로그 정보를 볼 수 있음

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git log
commit aa92947d350db27b604d1351930d4f809f96886e (HEAD -> master) ----- 아이디
Author: hojin <infohojin@gmail.com>
Date: Sat Jan 5 20:09:48 2019 +0900
이하 생략
```

- 각 커밋에는 aa92947d350db27b604d1351930d4f809f96886e 같은 이상한 영문과 숫자가 있는데 이를 **커밋 아이디**라고 함
- 특정 커밋을 가리키는 절대적 이름이고, 명시적 참조 값임
- SHA1이라는 해시 알고리즘을 사용하여 중복되지 않는 고유키 생성 → 충돌 방지
- SHA1 해시키는 매우 큰 숫자이기 때문에 고유 접두사로 간략하게 사용할 수 있는데, 해시의 앞쪽 **7자만으로도 중복을 방지**하면서 전체 키 값을 사용할 수 있음

## 4.9 커밋 로그

55

## 9. 커밋 로그



### > 간략 로그

- 커밋 메시지를 여러 줄 작성했다면 일반적인 로그 정보를 복잡하게 느낄 수 있음
- 로그 옵션 중에서 **--pretty=short**를 사용하면 로그를 출력할 때 첫 번째 줄의 커밋 메시지만 출력함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git log --pretty=short ----- 로그 확인
commit aa92947d350db27b604d1351930d4f809f96886e (HEAD -> master)
Author: hojin <infohojin@gmail.com>
```

```
commit aa1dd51a8883b2ea9a54209a00f434a2da01ee85
Author: hojin <infohojin@gmail.com>
hello git world 추가
```

```
commit e2bce41380691b0a34aeab7db889a6c30fed8287
Author: hojin <infohojin@gmail.com>
```

인덱스 페이지 레이아웃

## 9. 커밋 로그



### ➤ 간략 로그

- 특정 커밋의 상세 정보도 확인할 수 있음
- 특정 커밋의 상세 정보를 확인하고 싶다면 show 명령어를 사용함

```
$ git show 커밋ID
```

## 9. 커밋 로그



### ➤ 특정 파일의 로그

- 전체 커밋과 달리 특정 파일의 로그 기록만 볼 수도 있음
- log 명령어 뒤에 파일 이름을 적어 주면 됨

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git log index.htm ----- 파일의 로그 확인
commit aa92947d350db27b604d1351930d4f809f96886e (HEAD -> master)
Author: hojin <infohojin@gmail.com>
Date: Sat Jan 5 20:09:48 2019 +0900
```

## 4.10 diff 명령어

## 10. diff 명령어



### ➤ diff 명령어

- diff 명령어는 커밋 간 차이를 확인함
- 보통 리눅스나 macOS 같은 유닉스 계열의 운영 체제에는 유용한 diff 명령어가 있음
- 깃 또한 초기 시작은 리눅스 커널을 개발하려는 것이었기 때문에 유사한 기능을 하는 diff 명령어를 제공함

## 10. diff 명령어

Git  
교과서



### ➤ 워킹 디렉터리 vs 스테이지 영역

- 아직 add 명령어로 파일을 추가하지 않은 경우, 워킹 디렉터리와 스테이지 영역 간 변경 사항을 비교할 수 있음
- index.htm 파일을 열어 <h1> 태그 밑에 <h2> 태그와 내용을 추가함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ code index.htm
```

## 10. diff 명령어

Git  
교과서



### ➤ 워킹 디렉터리 vs 스테이지 영역

index.htm

```
<!DOCTYPE html>
<html>
<head>
 <meta charset="utf-8" />
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <title>JINYGIT</title>
</head>
<body>
 <h1>hello GIT world!</h1>
 <h2>깃을 이용하면 소스의 버전 관리를 쉽게 할 수 있습니다.</h2>
</body>
</html>
```

## 10. diff 명령어

Git  
교과서



### ➤ 워킹 디렉터리 vs 스테이지 영역

- diff 명령어를 실행해보자

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git diff ----- 스테이지 vs 워킹 디렉터리 비교
```

```
diff --git a/index.htm b/index.htm
```

```
index f5097d9..56af0de 100644
```

```
--- a/index.htm
```

```
+++ b/index.htm
```

```
@@ -7,5 +7,6 @@
```

```
</head>
```

```
<body>
```

```
 <h1>hello GIT world!</h1>
```

```
+ <h2>깃을 이용하면 소스의 버전 관리를 쉽게 할 수 있습니다.</h2> ----- 추가
```

```
</body>
```

```
</html>
```

```
\ No newline at end of file
```

원본의 7번째 줄부터 5줄,  
수정본의 7번째 줄부터 6줄

## 10. diff 명령어

Git  
교과서



### ➤ 워킹 디렉터리 vs 스테이지 영역

- 워킹 디렉터리 내용과 스테이지 내용의 차이점을 출력함
- 이번에는 변경한 파일을 add 명령어로 추가하여 스테이지 영역에 등록함

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git add index.htm
```

- 다음 다시 diff 명령어를 수행함
- 아무 내용도 출력되지 않음

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
```

```
$ git diff
```

- 이것은 등록 과정을 거쳐 워킹 디렉터리의 수정 내역을 스테이지 영역에 반영했기 때문임
- 아무 내용도 출력되지 않음



## 10. diff 명령어

Git 교과서



### ➤ 커밋 간 차이

- 워킹 디렉터리와 마지막 커밋을 비교해보자

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git diff head ----- head 포인터 입력
diff --git a/index.htm b/index.htm
index f5097d9..56af0de 100644
--- a/index.htm
+++ b/index.htm
@@ -7,5 +7,6 @@
</head>
<body>
 <h1>hello GIT world!</h1>
+ <h2>깃을 이용하면 소스의 버전 관리를 쉽게 할 수 있습니다.</h2>
</body>
</html>
\ No newline at end of file
```

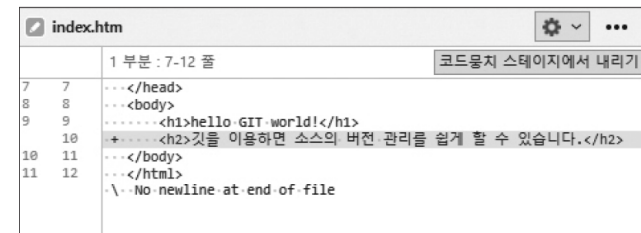
## 10. diff 명령어

Git 교과서



### ➤ 소스트리에서 간단하게 변경 이력 확인

- 소스트리에서 수정한 파일 이름을 클릭하면 오른쪽 창에 파일 변경 내역들이 출력됨



## 10. diff 명령어

Git 교과서



### ➤ 소스트리에서 간단하게 변경 이력 확인

- index.htm에서 <h1>hello GIT world!</h1>을 삭제한 후 다시 확인해 보면 삭제된 항목은 붉은색으로 표시함



## 10. diff 명령어

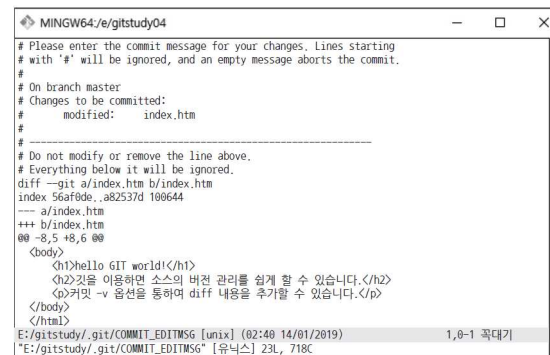
Git 교과서



### ➤ diff 내용을 추가하여 커밋

- 커밋 메시지를 작성할 때 -v 옵션을 같이 사용하면 vi 에디터에서 diff 내용을 추가할 수 있음

```
infoh@hojin MINGW64 /e/gitstudy04 (master)
$ git commit -v ----- diff 내용 추가
```



## 정리 & 실습

69

## 정리



### > 정리

- 커밋 작업은 깃에서 소스 코드를 관리하는 첫 단추임
- 너무 많은 코드를 수정한 후 커밋하는 것보다는 작은 단위로 코드를 수정한 후 커밋하는 것을 추천함
- 커밋의 수정 부분이 적을수록 검토하기 쉽고, 오류도 쉽게 찾을 수 있음

## 실습 내용



?

## 꼬고마(kkma) 형태소 분석기 설치



### > kkma 형태소 분석기

- 서울대학교 IDS 연구실에서 자연어 처리를 하기 위한 다양한 모듈 및 자료를 구축하기 위한 과제로 만든 한글 형태소 분석기

### 1) kkma 한글 형태소 분석기 다운로드

- <http://kkma.snu.ac.kr/documents/index.jsp>

### 라이브러리 내려받기 및 사용하기

꼬고마 한글 형태소 분석기는 Java 라이브러리로서 jar 파일 형태로 배포한다. 배포하는 jar 파일을 내려받아 형태소 분석기를 사용할 프로젝트의 classpath에 이 파일을 추가하면 형태소 분석기를 사용할 수 있다. Java 1.5 이상의 가상 머신 (Virtual Machine)에서 무리 없이 동작한다. 그러나 기분석 사건을 이용하기 때문에, 사건을 메모리에 적재하기 위한 충분한 힙메모리를 지정해주어야 한다. 따라서 512MB 이상의 메모리를 지정할 것을 권장한다.

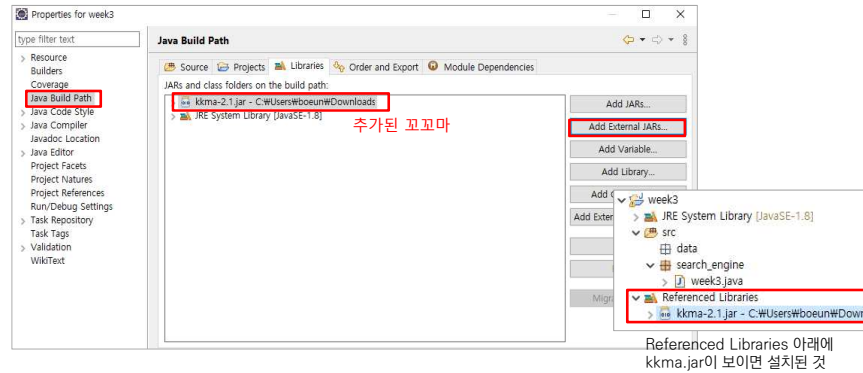
• 버전 2.1 내려받기

## 꼬꼬마(kkma) 형태소 분석기 설치



### 2) 라이브러리에 kkma 형태소 분석기 추가

- 해당 프로젝트 우클릭 → Properties → Java Build Path → Add External JARs → 다운받은 Jar파일 추가



## 꼬꼬마(kkma) 형태소 분석기 사용법



### ➤ 키워드(색인어) 추출하는 방법

```
String testString = "꼬꼬마형태소분석기를테스트하고있어요. 테스트결과를볼게요.";
// init KeywordExtractor
KeywordExtractor ke = new KeywordExtractor();
// extract keywords
KeywordList kl = ke.extractKeyword(testString, true);
// print result
for(int i = 0; i < kl.size(); i++) {
 Keyword kwr = kl.get(i);
 System.out.println(kwr.getString() + "\t" + kwr.getCnt());
}
```

### 코드 실행 결과

꼬꼬마	1
테스트	2
꼬꼬마형태소분석기	1
형태소	1
분석기	1
테스트결과	1
결과	1

## 질의응답



# Q&A

Homepage: <http://nlp.konkuk.ac.kr>  
E-mail: [nlprkim@konkuk.ac.kr](mailto:nlprkim@konkuk.ac.kr)