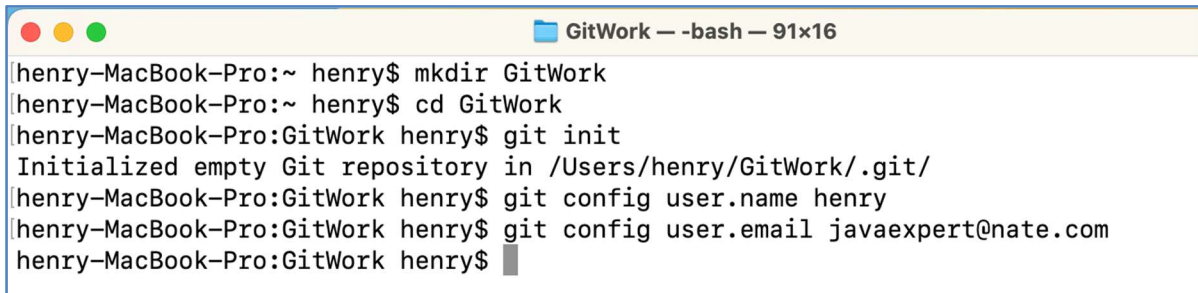


Git 기본적인 사용 방법

1. git init

- 1) 현재 Directory 를 Git Repository 로 초기화
- 2) Git 으로 Version 관리를 하려면 Repository 를 초기화해야 한다.
- 3) 실제로 Directory 를 생성하고 Repository 를 초기화하면 된다.

\$ git init

A terminal window titled "GitWork — -bash — 91x16" showing a series of commands and their outputs. The commands are: mkdir GitWork, cd GitWork, git init, git config user.name henry, and git config user.email javaexpert@nate.com. The output for git init is "Initialized empty Git repository in /Users/henry/GitWork/.git/".

```
[henry-MacBook-Pro:~ henry$ mkdir GitWork
[henry-MacBook-Pro:~ henry$ cd GitWork
[henry-MacBook-Pro:GitWork henry$ git init
Initialized empty Git repository in /Users/henry/GitWork/.git/
[henry-MacBook-Pro:GitWork henry$ git config user.name henry
[henry-MacBook-Pro:GitWork henry$ git config user.email javaexpert@nate.com
henry-MacBook-Pro:GitWork henry$
```

- 4) 초기화가 성공적으로 완료되면 **git init** 명령어를 실행한 directory 에 **.git** 이라는 이름의 Directory 가 만들어진다.
- 5) .git Directory 에는 현재 Directory 와 관련한 Repository 관리 정보가 저장된다.
- 6) Git 에는 이 Directory 이하의 내용을 해당 Repository 와 관련된 **Working Tree** 라고 부른다.

2. .gitignore

- 1) Git 은 **Working Directory** 안에 있는 모든 파일을 추적 관리한다.
- 2) 하지만, 개발과정에서 불필요한 파일과 중요한 파일들은 추적관리에서 배제해야 한다.
- 3) 이를 위해 Git 은 별도의 추적관리 배제하기 위한 **.gitignore** 라는 설정 파일을 제공한다.
- 4) **.gitignore** 에 정의된 파일은 **Staging Area** 에 올라가지 않기 때문에 Tracking 되지 않는다.
- 5) 따라서 **git status** 에서 보이지 않는다.
- 6) 불필요한 파일
 - C 언어와 같은 언어에서 컴파일하면 목적파일(.obj)등 다양한 부산물의 파일들이 같이 생성된다.
 - 코드의 이력을 관리할 때 이런 부수적인 파일까지 이력을 관리할 필요가 없다.
 - 불필요한 실행파일, 임시파일까지 모두 추적 관리하게 되면 저장소의 크기가 불필요하게 커지는 문제도 발생하고, 시스템의 동작을 느리게 하는 이유가 되기도 한다.
 - 이런 파일들의 목록, 디렉토리 등을 별도의 설정파일에 등록하여 Git 의 추적관리대상에서 제외하게 할 수 있다.

- 예를 들면,
 - NodeJS 로 개발시 npm module
 - Java 로 개발시 .class

7) 중요한 파일

- Database 접속 정보처럼 개발할 때는 필요하지만, 외부에 배포할 때 중요한 정보가 노출되는 경우가 있다.
- Git 은 중요한 파일들이 Git 에 의해 추적관리되지 않고, 외부에 노출되지 않도록 별도의 설정 파일에 등록하여 제외하게 할 수 있다.
- 예를 들면
 - AWS Secret Access Key
 - JWT 비밀 키

8) .gitignore 파일

- Git 은 이력 추적을 제외하게 하기 위해 별도의 목록 파일을 생성한다.
 - Working Directory 안에 .gitignore 파일을 생성한다.
- \$ touch .gitignore**

9) 파일 생성

- 텍스트 에디터를 통해 간단하게 파일을 생성한다.
- 확장자는 없다.
- Working Directory 루트에 생성한다.
- 파일명은 반드시 .gitignore 이다.
- 이 파일안에 추적을 배제하기 위한 파일, 폴더의 목록을 나열하면 된다.

10) 표기법

- 특정 규칙을 적용하여 Git 의 추적관리를 배제할 수 있다.
- 이를 위해서 Shell Globbing 문법을 이용하여 문자열을 지정해야 한다.
- 주석
 - 주석을 작성할 때에는 해당 라인이 #으로 시작하면 된다.

- 파일명

- 추적관리에서 제외될 파일들의 목록들을 넣는다.
- 이때 파일명은 확장자를 포함하여 완전한 형태로 기입해야 한다.
- 만일 확장자를 제외하거나 특정 패턴을 지정하고자 할 때에는 Shell Globbing 문자열을 같이 작성한다.
- 만일 파일을 지정할 때 경로가 있다면 경로명도 같이 입력해 준다.

DB 접속 파일을 제외함

dbinfo.php

- 전체 기호

- * 기호를 사용하여 패턴을 정의한다.
- * 기호는 모든 문자열을 대체한다.
- 이런 문자를 Shell Globbing 이라고 한다.
- Globbing 문자를 사용하여 패턴을 확장한다.

오브젝트 파일은 제외함

*.obj

- 포함하기

- ignore 패턴을 작성할 때 반드시 추적 관리를 제외하는 파일만 작성하는 것은 아니다.
- 제외하지 않는 파일과 필요한 파일은 파일 이름 앞에 !를 사용한다.
- 느낌표는 부정을 의미하는 not 과 같다.

환경 설정 파일은 제외하면 안됨.

!config.php

- 폴더

- 폴더는 OS 에 따라 Directory 를 표현하는 방법이 다를 수 있다.
- Git 에서 Directory 를 표현할 때는 Linux 와 같이 / 기호를 사용한다.

현재 디렉토리 안에 있는 파일 무시

/readme.txt

/pub/ 디렉토리 안의 모든 것 무시

/pub/

doc 디렉토리 아래의 모든 .txt 파일 무시

doc/**/*.*.txt

- 정규표현식

- Git 은 glob 패턴을 지원하기 때문에 정규표현식을 응용하여 작성 규칙을 넣을 수 있다.

```
# : comments

# no .a files
*.a

# but do track lib.a, even though you're ignoring .a files above
!lib.a

# only ignore the TODO file in the current directory, not subdir/TODD
/TODD

# ignore all files in the build/ directory
build/

# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt

# ignore all .pdf files in the doc/ directory
doc/**/*.*pdf
```

11) 주의할 점

- 이미 **Staging Area** 나 **Repository** 에 Commit 으로 올라간 파일은 gitignore 를 하기 위해서는 먼저 파일을 제거해야 한다.
- 다음과 같은 명령어가 가능하다.

```
$ git rm [파일명]
```

```
$ git commit -m "message"
```

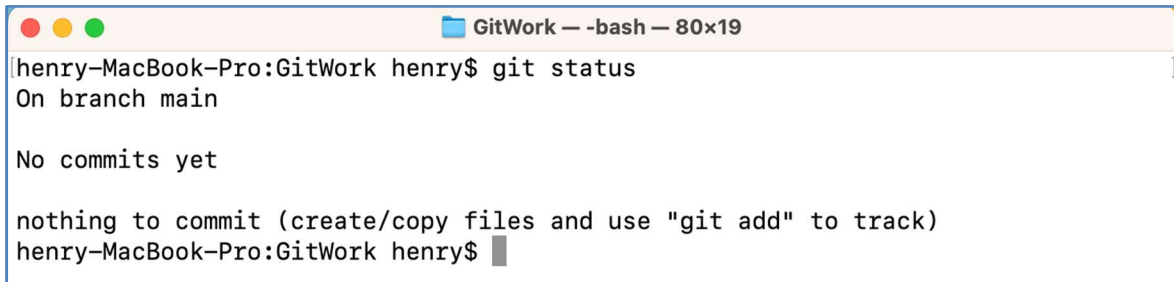
12) gitignore.io 이용하기

- <https://www.toptal.com/developers/gitignore>

3. git status

- 1) Git Repository 의 상태를 표시하는 명령이다.
- 2) Working Tree 또는 Repository 에 대응되는 조작을 하면 상태가 차례대로 변경된다.
- 3) **git status** 명령어로 현재 상태를 확인하면서 Git 의 명령어를 하나하나 입력하는 것이 기본이다.

\$ git status

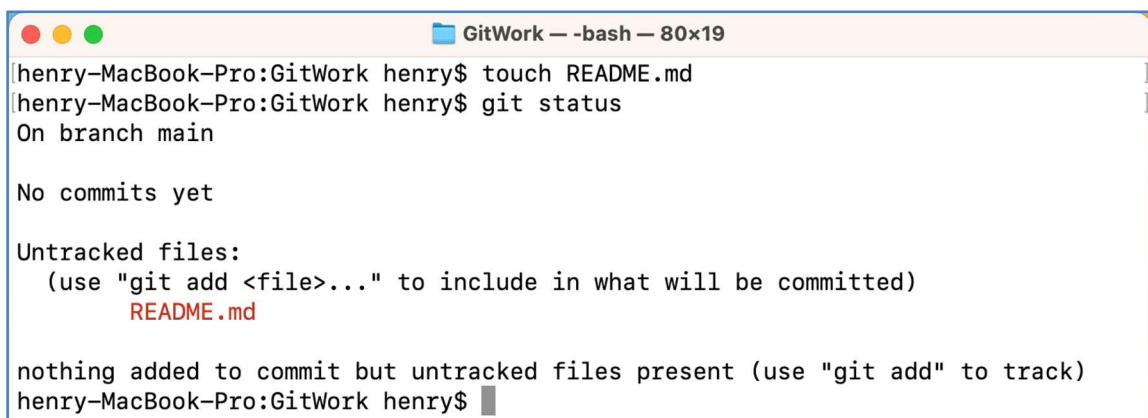


```
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
henry-MacBook-Pro:GitWork henry$
```

- 4) 현재 **main**(구 **master**)라는 이름을 가진 Branch 에 있는 것이 표시된다.
- 5) 또한 출력 결과를 보면 Commit 이 아직 없다는 것도 표시된다.
- 6) 간단하게 README.md File 을 작성해 보자.



```
henry-MacBook-Pro:GitWork henry$ touch README.md
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
henry-MacBook-Pro:GitWork henry$
```

- 8) README.md File 이 **Untracked files** 에 표시되는 것을 확인할 수 있다.
- 9) Git 의 입장에서 보면, 한번도 관리된 적이 없는 파일이라는 뜻.
- 10) 즉 Git 의 Working Tree 또는 Repository 에 명령이나 특별한 작업을 하면, **git status** 명령어의 출력이 변경되는 것이다.
- 11) 자주 사용되는 명령이기 때문에 꼭 기억해야 한다.

4. git add

- 1) Working Directory에 있는 File 을 Stage Area로 추가하는 명령이다.
- 2) Git Repository 의 Working Tree File 을 생성하는 것만으로는 Git Repository 의 Version 관리에 등록되지 않는다.
- 3) 여전히 **git status** 로 확인해 보면 위에 생성했던 README.md file 은 Untracked files로 표시된다.
- 4) File 을 Git Repository 에서 관리하도록 하려면 **git add** 명령어를 Stage Area(Index 라고 부르기도 한다)라고 불리는 장소에 등록해야 한다.
- 5) Stage Area이란 commit 하기 전의 임시 영역이다. 다음 3 가지 중 하나를 사용할 수 있다.

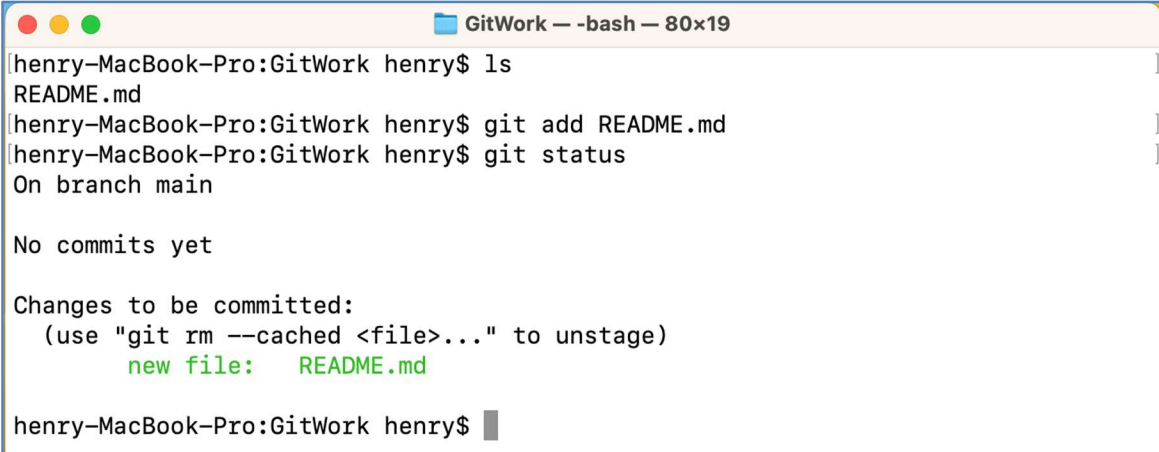
```
$ git add README.md
```

```
$ git add .
```

```
$ git add -i
```

- 6) README.md 파일을 **git add** 를 통해 Staging Area에 추가한다.

```
$ git add README.md
```



```
GitWork - -bash - 80x19
[henry-MacBook-Pro:GitWork henry$ ls
README.md
[henry-MacBook-Pro:GitWork henry$ git add README.md
[henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

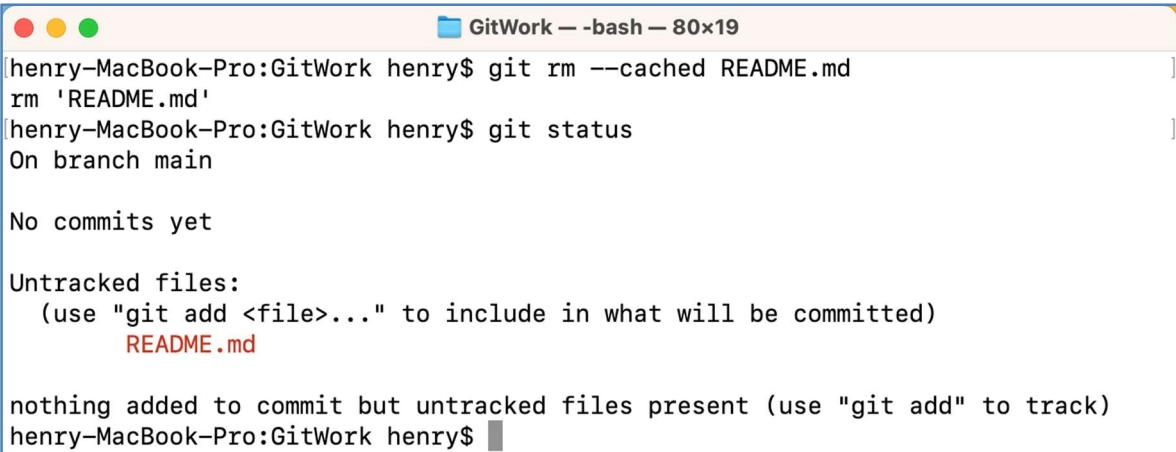
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md

henry-MacBook-Pro:GitWork henry$
```

- 7) **git status** 로 확인해 보면, Changes to be committed에 README.md File 이 표시되는 것을 확인할 수 있다.

- 8) Staging Area에 옮겨진 File 을 다시 Working Directory로 옮기는 명령어는 다음과 같다.

```
$ git rm --cached <file name>
```



```
GitWork - -bash - 80x19
[henry-MacBook-Pro:GitWork henry$ git rm --cached README.md
rm 'README.md'
[henry-MacBook-Pro:GitWork henry$ git status
On branch main

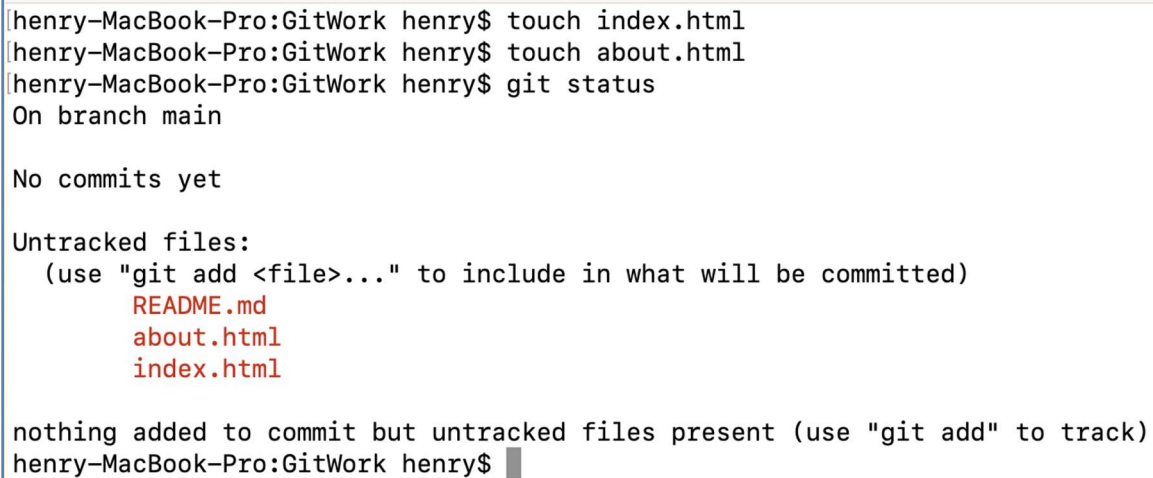
No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
henry-MacBook-Pro:GitWork henry$
```

9) 아래 예제에 따라 연습해 보자.

```
$ touch index.html
$ touch about.html
$ git status
```



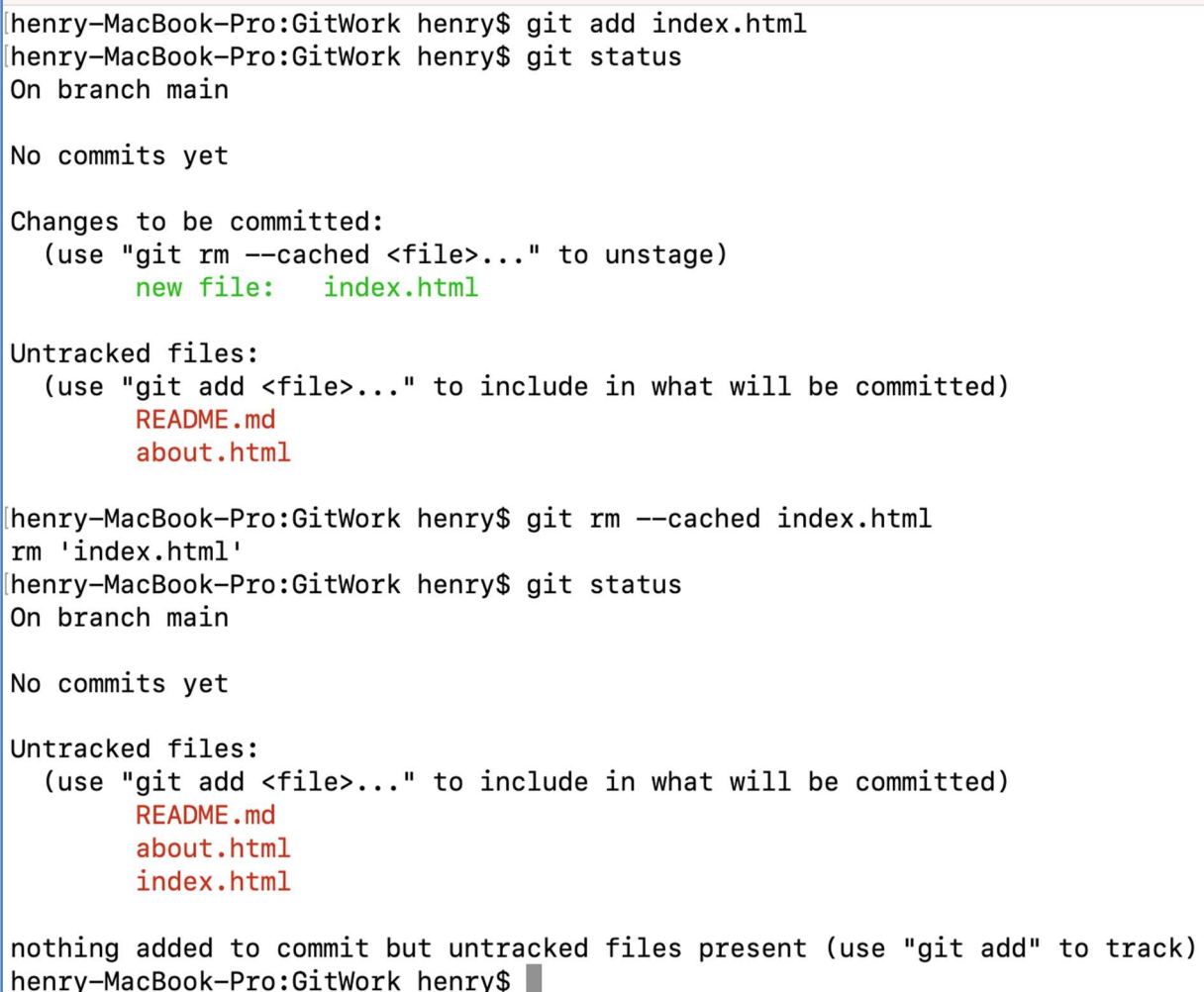
```
henry-MacBook-Pro:GitWork henry$ touch index.html
henry-MacBook-Pro:GitWork henry$ touch about.html
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        about.html
        index.html

nothing added to commit but untracked files present (use "git add" to track)
henry-MacBook-Pro:GitWork henry$
```

```
$ git add index.html
$ git status
$ git rm --cached index.html
$ git status
```



```
henry-MacBook-Pro:GitWork henry$ git add index.html
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        about.html

henry-MacBook-Pro:GitWork henry$ git rm --cached index.html
rm 'index.html'
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        about.html
        index.html

nothing added to commit but untracked files present (use "git add" to track)
henry-MacBook-Pro:GitWork henry$
```

```
$ git add .
$ git status
```

```
GitWork — -bash — 80x30
henry-MacBook-Pro:GitWork henry$ git add .
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
        new file:   about.html
        new file:   index.html

henry-MacBook-Pro:GitWork henry$
```

```
$ git rm -r --cached .
$ git status
```

```
GitWork — -bash — 80x30
henry-MacBook-Pro:GitWork henry$ git rm -r --cached .
rm 'README.md'
rm 'about.html'
rm 'index.html'
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        about.html
        index.html

nothing added to commit but untracked files present (use "git add" to track)
henry-MacBook-Pro:GitWork henry$
```



```
$ git add -i
```

```
*** Commands ***
```

1: status	2: update	3: revert	4: add untracked
5: patch	6: diff	7: quit	8: help

```
What now> 4
```

```
1: README.md
2: about.html
3: index.html
```

```
Add untracked>> 3
```

```
1: README.md
2: about.html
```

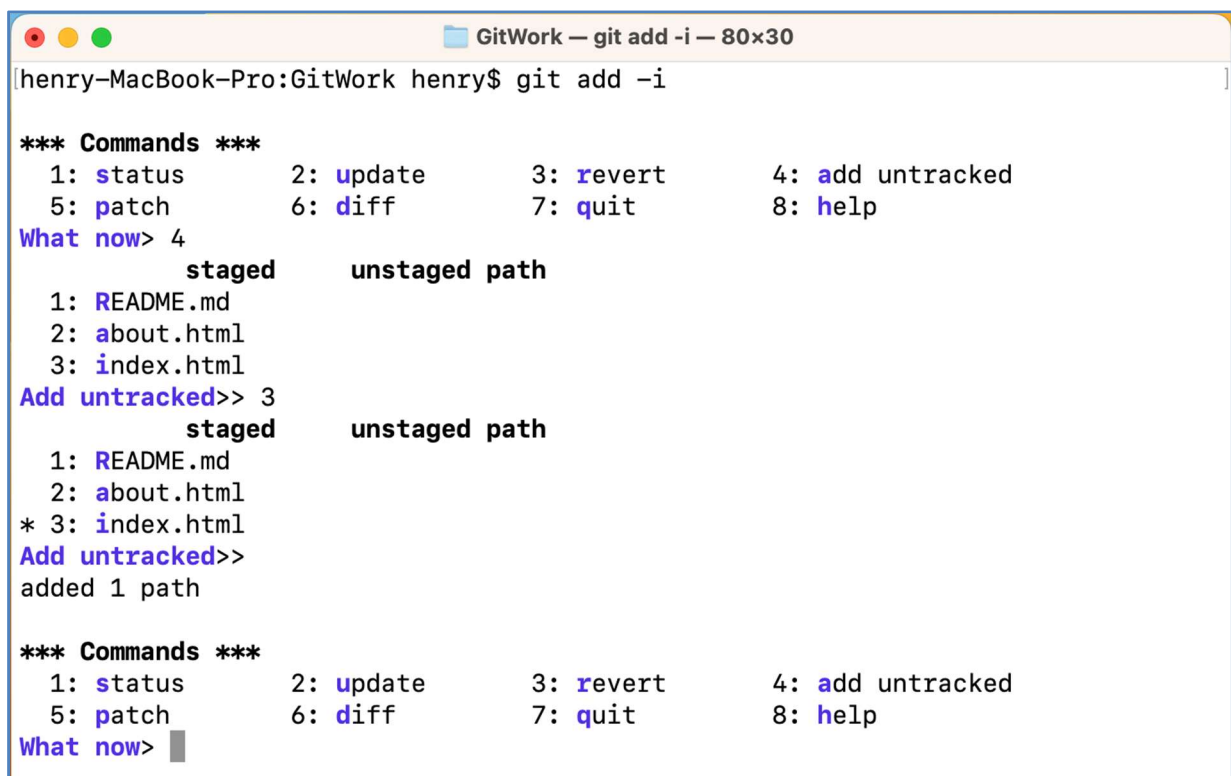
```
* 3: index.html
```

```
Add untracked>> [엔터키]
```

```
added 1 path
```

```
*** Commands ***
```

1: status	2: update	3: revert	4: add untracked
5: patch	6: diff	7: quit	8: help



```
henry-MacBook-Pro:GitWork henry$ git add -i

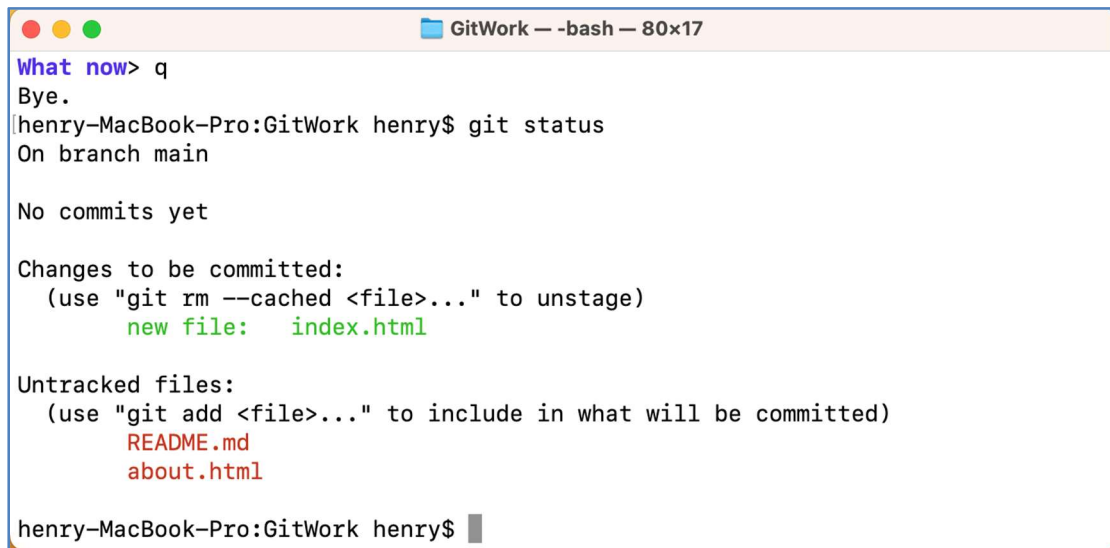
*** Commands ***
 1: status      2: update     3: revert     4: add untracked
 5: patch      6: diff       7: quit      8: help
What now> 4
      staged      unstaged path
 1: README.md
 2: about.html
 3: index.html
Add untracked>> 3
      staged      unstaged path
 1: README.md
 2: about.html
* 3: index.html
Add untracked>>
added 1 path

*** Commands ***
 1: status      2: update     3: revert     4: add untracked
 5: patch      6: diff       7: quit      8: help
What now> █
```

What now> q

Bye.

\$ git status

A terminal window titled "GitWork — -bash — 80x17" showing the output of the 'git status' command. The output indicates the current branch is 'main', there are no commits yet, and there are changes to be committed (a new file 'index.html') and untracked files ('README.md' and 'about.html').

```
What now> q
Bye.
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        about.html

henry-MacBook-Pro:GitWork henry$
```

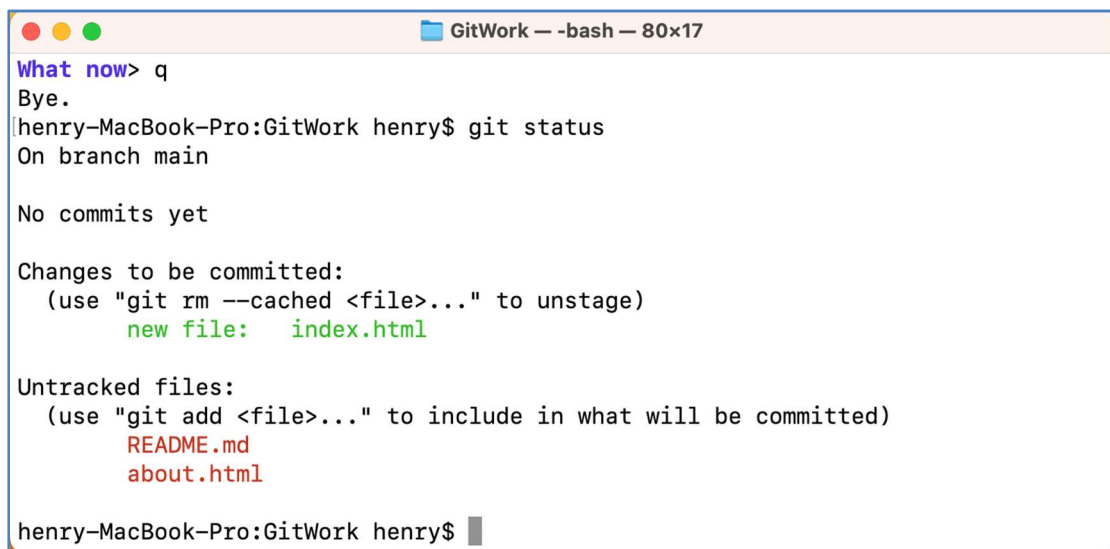
5. git commit

- 1) **Staging Area** 에 있는 File 을 Repository 에 저장하는 명령어
- 2) **Staging Area** 에 기록된 시점의 File 들을 실제 Repository 변경 내역에 반영하는 것
- 3) 이러한 기록을 기반으로 File 을 **Working Tree** 에 복원하는 것이 가능한 것이다.

4) git commit -m "message"

- 한 줄의 Commit Message 를 기록하기

\$ git commit -m "Message"

A terminal window titled "GitWork — -bash — 80x17" showing the output of the 'git status' command after a commit. The output is identical to the previous one, showing changes to be committed and untracked files.

```
What now> q
Bye.
henry-MacBook-Pro:GitWork henry$ git status
On branch main

No commits yet

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        about.html

henry-MacBook-Pro:GitWork henry$
```

- -m Option 을 사용하여 index.html 을 Commit 해 보자.

```
$ git commit -m "First commit"
```

```
henry-MacBook-Pro:GitWork henry$ git commit -m "First commit"
[main (root-commit) 7f32742] First commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 index.html
henry-MacBook-Pro:GitWork henry$
```

5) git commit

- 상세한 Commit Message 기록하기
- 한 개의 File 이라도 **git add** 를 해서 Staging Area 에 올려놓은 후 실행한다.

```
$ git add about.html
```

```
$ git commit
```

```
henry-MacBook-Pro:GitWork henry$ git add about.html
henry-MacBook-Pro:GitWork henry$ git commit
```

```
devex@DESKTOP-1BKHISM MINGW64 /c/gitwork (master)
$ git add about.html

devex@DESKTOP-1BKHISM MINGW64 /c/gitwork (master)
$ git commit
hint: Waiting for your editor to close the file... unix2dos: converting file C:/gitwork/.git/COMMIT_EDITMSG to DOS format...
dos2unix: converting file C:/gitwork/.git/COMMIT_EDITMSG to Unix format...
Aborting commit due to empty commit message.
```

- Git 를 설치할 때 지정한 Editor 가 실행된다.
- 기본 Editor 는 VIM 이다.
- 만일 Editor 가 설정되어 있지 않으면 다음과 같은 오류가 발생한다.

```
$ git commit
```

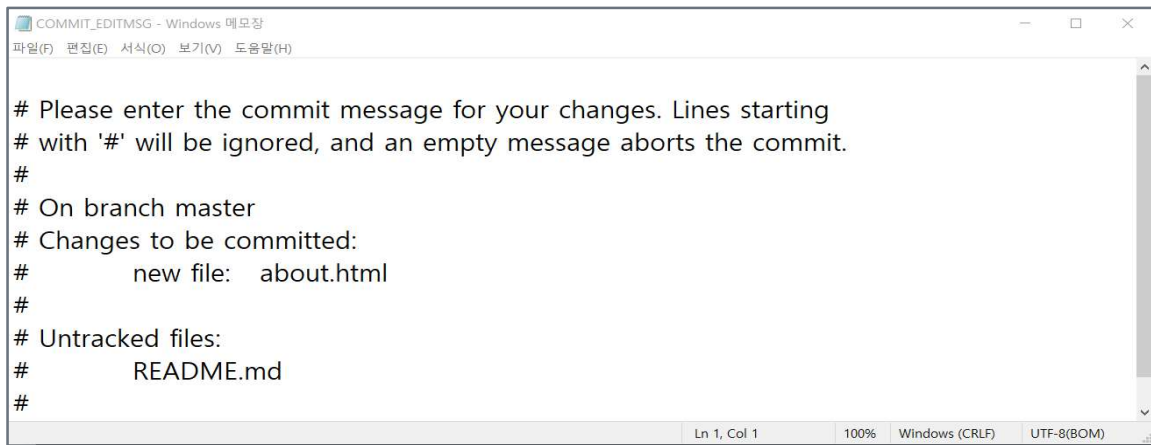
```
error : Terminal is dumb, but EDITOR unset
```

```
Please supply the message using either -m or -F option.
```

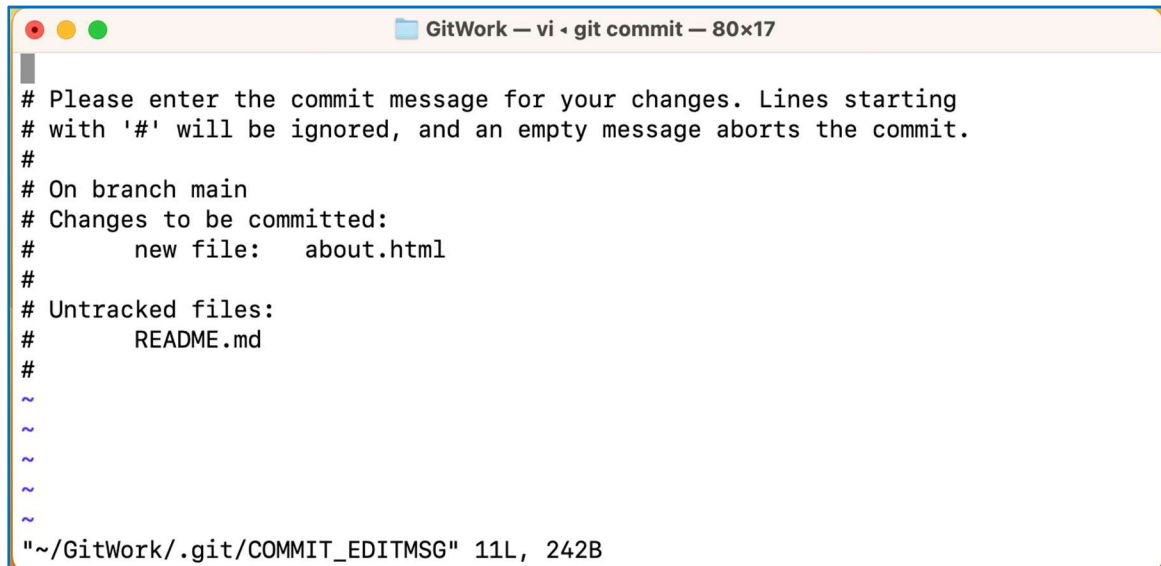
- 다음의 명령어로 Editor 를 등록한다.
- Editor 로 메모장(Windows OS)을 등록하기로 하자.
- 만일 다른 Editor 를 사용하려면 해당 Editor Program 의 경로를 입력하면 된다.

```
$ git config --local core.editor notepad
```

```
devex@DESKTOP-1BKHISM MINGW64 /c/gitwork (master)
$ git commit
hint: Waiting for your editor to close the file... unix2dos: converting file C:/gitwork/.git/COMMIT_EDITMSG to DOS format...
```



```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#   new file:   about.html
#
# Untracked files:
#   README.md
#
```



```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Changes to be committed:
#   new file:   about.html
#
# Untracked files:
#   README.md
#
#
#
#
#
~/GitWork/.git/COMMIT_EDITMSG" 11L, 242B
```

- **git commit** 은 미리 지정한 Editor 를 이용하는데, 보다 상세한 Commit Message 를 기록하기 위함이다.
- 기록하는 형식은 아래와 같다.
 - 첫 번째 줄 : commit 으로 인한 변경 내용을 한 줄로 요약해서 작성, Title 이라고 함.
 - 두 번째 줄 : 공백
 - 세 번째 줄 이후 : 변경과 관련된 내용을 상세하게 기록
- 이렇게 형식을 지켜서 작성하게 되면, Log 를 확인하는 명령어 혹은 Tool 등에서 자세한 Commit Message 가 출력된다.
- **#** 기호는 Comment 처리된다.
- 이제 about.html file 을 Commit 해보자.
- 아직 **Staging Area** 에 올려놓지 않았으면 **git add** 명령어로 **Staging Area** 에 올려놓은 다음, 보다 상세한 Commit Message 를 작성하기 위해 미리 지정된 VIM Editor 를 실행하도록 하자.

```
$ git add about.html
```

```
Instructor@DESKTOP-NU7GQVV MINGW64 /c/gitwork (master)
$ git add about.html

Instructor@DESKTOP-NU7GQVV MINGW64 /c/gitwork (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   about.html
```

```
$ git commit
```

[illegible]

```
MINGW64:/c/gitwork

첫 줄은 title이라고 부르는 간단한 message 처리부분이다.

두 번째 줄은 공백으로 놓는다.
세 번째 줄부터 보다 상세한 commit message를 넣는다.
모두 입력했으면 ESC > :wq를 입력해서 editor를 닫는다.
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#   new file:   about.html
#
~
~
~
~
~
~
~
~
~
~
C:/gitwork/.git/COMMIT_EDITMSG[+] [unix] (00:33 15/11/2018) 5,70-54 All
-- INSERT --
```

```
GitWork — vi < git commit — 80x17

첫 줄은 title이라고 부르는 간단한 message 처리부분이다.

두 번째 줄은 공백으로 놓는다.
세 번째 줄부터 보다 상세한 commit message를 넣는다.
모두 입력했으면 ESC >:wq를 입력해서 editor를 닫는다.
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Changes to be committed:
#   new file:   about.html
#
# Untracked files:
#   README.md
#
~
-- INSERT --
```

- 모두 입력이 마친 상태는 아래의 그림과 같다.

```
GitWork — -bash — 80x14

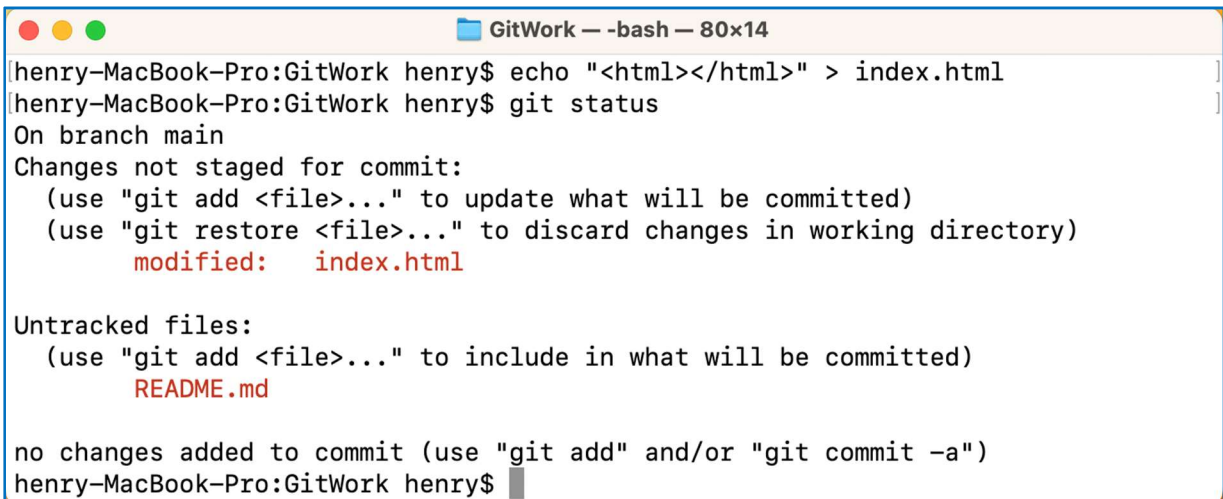
henry-MacBook-Pro:GitWork henry$ git commit
[main 5a6de60] 첫 줄은 title이라고 부르는 간단한 message 처리부분이다.
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 about.html
henry-MacBook-Pro:GitWork henry$
```

- 첫 줄 즉 Title 부분만 보이는 것을 알 수 있다.

6) git commit -a -m "Message"

- 이미 Commit 처리된 File 중에 내용이 수정이 된 File 은 다시 **Working Area** 로 내려오게 된다.
- 한번 Commit 했었기 때문에 이 File 은 **git add** 명령에 의해 **Staging Area** 에 옮길 필요 없이 바로 Commit 할 수 있다.
- 이럴 때 사용하는 Option 이다.
- 하지만, 한번도 **Staging Area** 에 옮겨본 적이 없는 File 은 이 Option 을 사용해서 바로 Commit 하려고 해도 Commit 되지 않는다.
- 이미 Commit 된 index.html 을 수정해 보자.

```
$ echo "<html></html>" > index.html
```

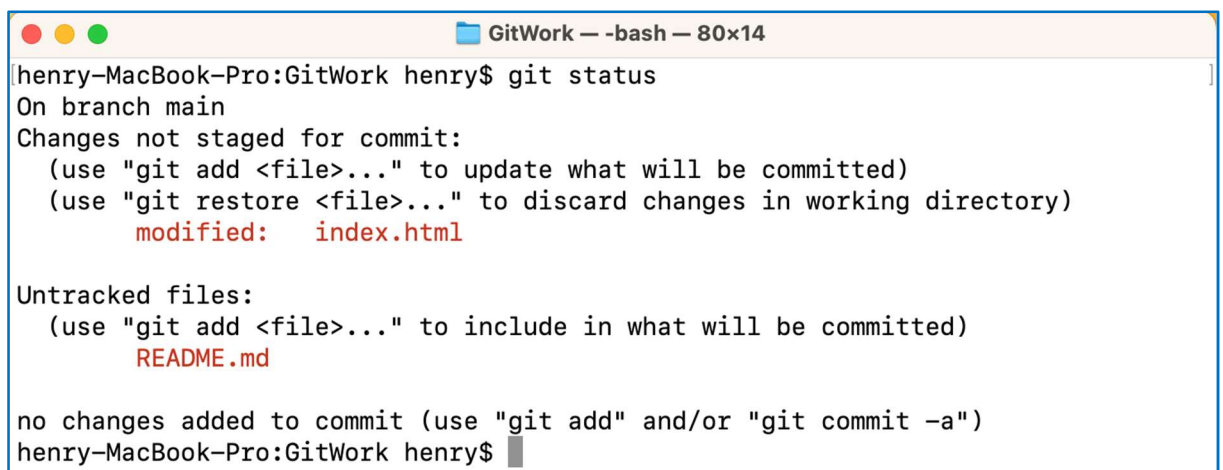


```
henry-MacBook-Pro:GitWork henry$ echo "<html></html>" > index.html
henry-MacBook-Pro:GitWork henry$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

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

no changes added to commit (use "git add" and/or "git commit -a")
henry-MacBook-Pro:GitWork henry$
```

- Index.html 을 수정했더니 다시 **Working Area** 에 있는 것을 알 수 있다.
- 먼저 **git status** 로 현 상태를 확인한다.



```
henry-MacBook-Pro:GitWork henry$ git status
On branch main
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:   index.html

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

no changes added to commit (use "git add" and/or "git commit -a")
henry-MacBook-Pro:GitWork henry$
```

- 이 File 을 Commit 해보자.
- 여기서 주의할 점은 한번도 **Staging Area** 에 옮겨진 적이 없는 File 은 Commit 이 안된다는 점이다.

```

henry-MacBook-Pro:GitWork henry$ git commit -a -m "Add html tag"
[main 5d99ecc] Add html tag
1 file changed, 1 insertion(+)
henry-MacBook-Pro:GitWork henry$

```

- Commit 을 한 후 상태를 확인해 보자.

```

henry-MacBook-Pro:GitWork henry$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README.md

nothing added to commit but untracked files present (use "git add" to track)
henry-MacBook-Pro:GitWork henry$

```

- 현재 **Working Tree** 의 상태는 Commit 이 된 최신 상태이다.
- 변경 내역이 이미 반영되었기 때문에 새로운 변경 내역이 없다는 것을 확인할 수 있다.

6. git log

- 1) Repository 에 있는 Commit 이력을 조회하는 명령어
- 2) 누가 언제 Commit 또는 Merge 를 했는지, 어떤 변경이 발생했는지 등을 확인할 수 있다.

3) git log

- Commit 이력 상세 조회

4) git log --oneline

- Commit 이력 중 Commit ID, Title Message 만 조회

```

henry-MacBook-Pro:GitWork henry$ git log --oneline
5d99ecc (HEAD -> main) Add html tag
5a6de60 첫 줄은 title이라고 부르는 간단한 message 처리부분이다.
7f32742 First commit
henry-MacBook-Pro:GitWork henry$

```


5) **git log --oneline --decorate --graph --all**

- 모든 Branch Commit 이력 조회

6) **git log <file name>**

- 특정 File 의 변경 Commit 조회

```
henry-MacBook-Pro:GitWork henry$ git log index.html
commit 5d99ecc52b1591347d935e1706698250143c8bb9 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:35:07 2023 +0900

    Add html tag

commit 7f3274217ae9b3e18a6473087a12fd853f8ef572
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:26:52 2023 +0900

    First commit
henry-MacBook-Pro:GitWork henry$
```

7) 이제까지의 Commit 이력을 확인해 보자.

\$ git log

```
henry-MacBook-Pro:GitWork henry$ git log
commit 5d99ecc52b1591347d935e1706698250143c8bb9 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:35:07 2023 +0900

    Add html tag

commit 5a6de60584ff88741e358626eaad8add5799de51
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:30:39 2023 +0900

    첫 줄은 title이라고 부르는 간단한 message 처리 부분이다.

    두 번째 줄은 공백으로 놓는다.
    세 번째 줄부터 보다 상세한 commit message를 넣는다.
    모두 입력했으면 ESC >:wq를 입력해서 editor를 닫는다.

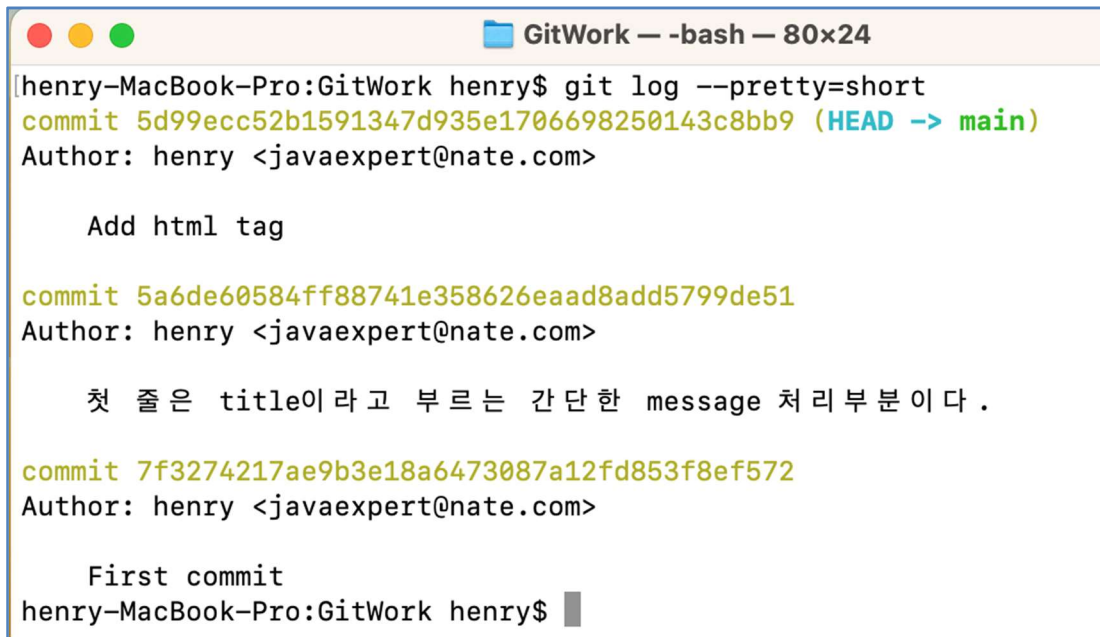
commit 7f3274217ae9b3e18a6473087a12fd853f8ef572
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:26:52 2023 +0900

    First commit
henry-MacBook-Pro:GitWork henry$
```

- 방금 전까지 Commit 했던 내용들을 확인할 수 있다.
- Commit 옆에 표시되는 글자들은 해당 Commit 을 나타내는 Hash 인데, 다른 Git 명령어에서 이러한 Hash 를 이용한다.
- Author 옆에는 Git 에서 설정한 이름과 Email 주소가 표시된다.
- Date 옆에는 Commit 했던 시간이 표시되고, 그 아래에는 이전에 입력했던 Commit Message 가 출력된다.

8) 이번에는 Commit Message 의 첫 번째 줄만 출력해보자.

\$ git log --pretty=short



```

henry-MacBook-Pro:GitWork henry$ git log --pretty=short
commit 5d99ecc52b1591347d935e1706698250143c8bb9 (HEAD -> main)
Author: henry <javaexpert@nate.com>

    Add html tag

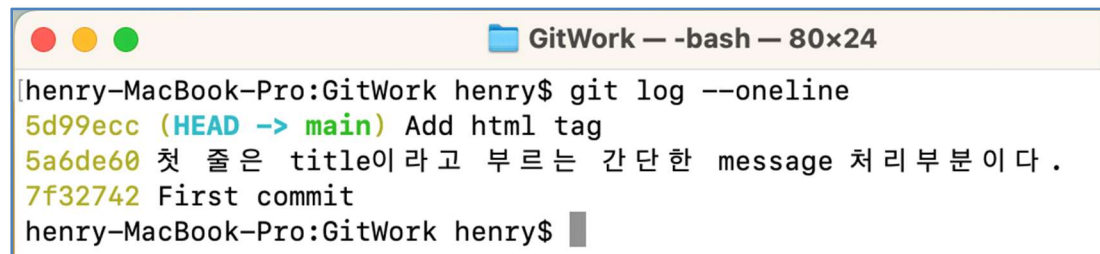
commit 5a6de60584ff88741e358626eaad8add5799de51
Author: henry <javaexpert@nate.com>

    첫 줄은 title이라고 부르는 간단한 message 처리 부분이다.

commit 7f3274217ae9b3e18a6473087a12fd853f8ef572
Author: henry <javaexpert@nate.com>

    First commit
henry-MacBook-Pro:GitWork henry$
  
```

- Commit Message 가 여러 줄이면 Commit 을 확인할 때 복잡해 보일 수 있다.
- 이럴 때 Commit Message 의 첫 번째 요약 줄만 표시하는 기능이다.
- **git log --oneline** 과 의미는 같다.
- 더 간단하게 출력한다.

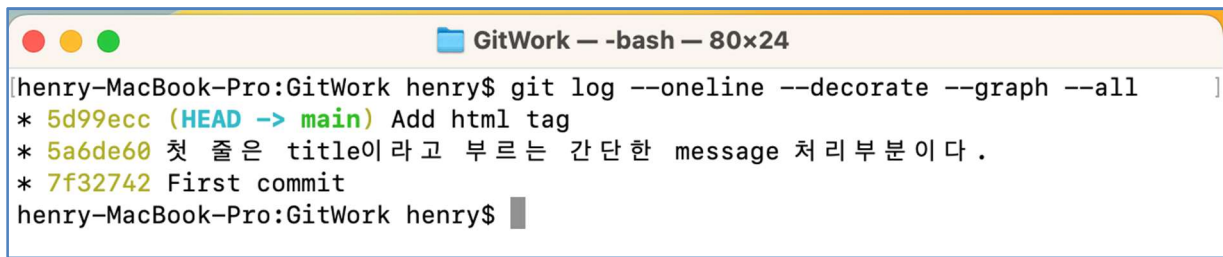


```

henry-MacBook-Pro:GitWork henry$ git log --oneline
5d99ecc (HEAD -> main) Add html tag
5a6de60 첫 줄은 title이라고 부르는 간단한 message 처리 부분이다.
7f32742 First commit
henry-MacBook-Pro:GitWork henry$
  
```

9) 아직 Branch 를 학습하지는 않았지만, 모든 Branch 의 Commit 이력을 조회하는 명령도 해 보자.

```
$ git log --oneline --decorate --graph -all
```

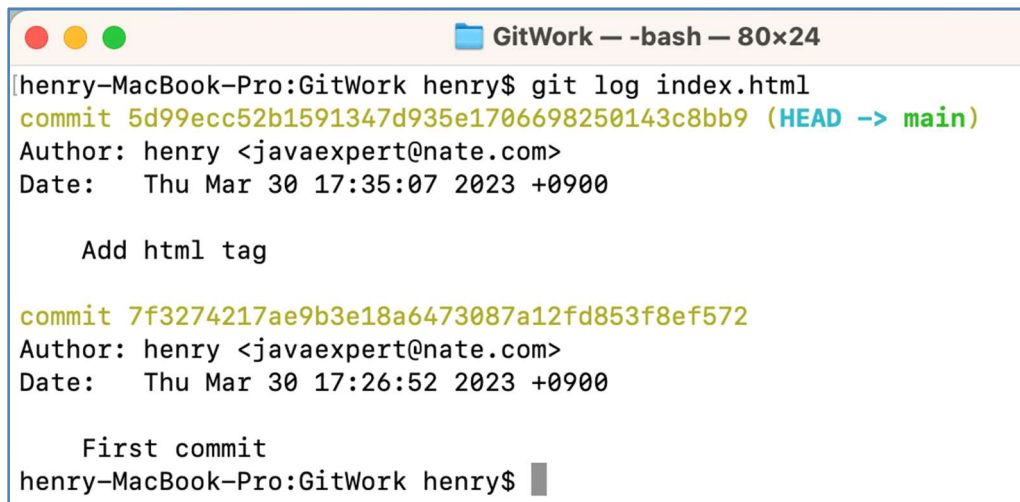
A terminal window titled 'GitWork - -bash - 80x24' showing the output of the command 'git log --oneline --decorate --graph --all'. The output lists three commits: 5d99ecc (HEAD -> main) 'Add html tag', 5a6de60 '첫 줄은 title이라고 부르는 간단한 message 처리 부분이다.', and 7f32742 'First commit'.

```
henry-MacBook-Pro:GitWork henry$ git log --oneline --decorate --graph --all
* 5d99ecc (HEAD -> main) Add html tag
* 5a6de60 첫 줄은 title이라고 부르는 간단한 message 처리 부분이다.
* 7f32742 First commit
henry-MacBook-Pro:GitWork henry$
```

10) 선택한 Folder 또는 File 의 Log 를 출력하는 방법

- **git log <file name>**
- 해당 Folder 나 File 의 Log 만 표시된다.

```
$ git log index.html
```

A terminal window titled 'GitWork - -bash - 80x24' showing the output of the command 'git log index.html'. It displays the commit history for the index.html file, including commit 5d99ecc52b1591347d935e1706698250143c8bb9 (HEAD -> main) with message 'Add html tag' and commit 7f3274217ae9b3e18a6473087a12fd853f8ef572 with message 'First commit'.

```
henry-MacBook-Pro:GitWork henry$ git log index.html
commit 5d99ecc52b1591347d935e1706698250143c8bb9 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:35:07 2023 +0900

    Add html tag

commit 7f3274217ae9b3e18a6473087a12fd853f8ef572
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:26:52 2023 +0900

    First commit
henry-MacBook-Pro:GitWork henry$
```

11) File 의 변경된 내용을 출력하는 방법

- Commit 에서 변경된 내용을 확인하고 싶을 때는 **-p** Option 을 사용한다.
- Commit Message 뒤에는 변경 내용이 함께 표시된다.

```
$ git log -p
```

```
GitWork — -bash — 80x25
henry-MacBook-Pro:GitWork henry$ git log -p
commit 5d99ecc52b1591347d935e1706698250143c8bb9 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:35:07 2023 +0900

    Add html tag

diff --git a/index.html b/index.html
index e69de29..18ecdcb 100644
--- a/index.html
+++ b/index.html
@@ -0,0 +1 @@
+<html></html>

commit 5a6de60584ff88741e358626eaad8add5799de51
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:30:39 2023 +0900

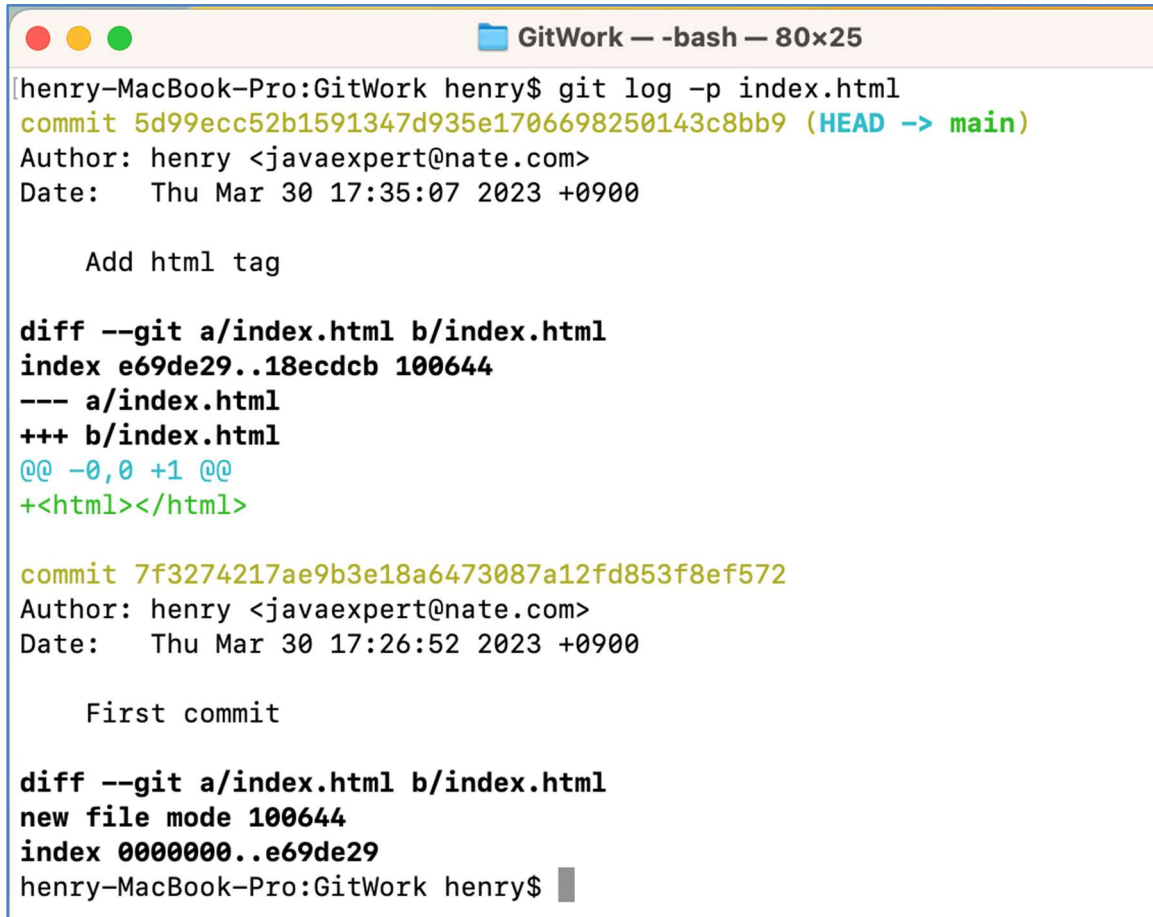
    첫 줄은 title이라고 부르는 간단한 message 처리 부분이다.

    두 번째 줄은 공백으로 놓는다.
    세 번째 줄부터 보다 상세한 commit message를 넣는다.
    모두 입력했으면 ESC >:wq를 입력해서 editor를 닫는다.

:...skipping...
```

- 만일 특정 파일의 변경 내용만 출력하려면 **-p** Option 뒤에 File 이름을 넣으면 된다.

\$ git log -p <file name>

A terminal window titled "GitWork — -bash — 80x25" showing the output of the command "git log -p index.html". The output displays two commits. The first commit, with hash 5d99ecc52b1591347d935e1706698250143c8bb9, is labeled "(HEAD -> main)" and shows a diff for "index.html" with the addition of an HTML tag. The second commit, with hash 7f3274217ae9b3e18a6473087a12fd853f8ef572, is labeled "First commit" and shows the initial creation of "index.html".

```
henry-MacBook-Pro:GitWork henry$ git log -p index.html
commit 5d99ecc52b1591347d935e1706698250143c8bb9 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:35:07 2023 +0900

    Add html tag

diff --git a/index.html b/index.html
index e69de29..18ecdcb 100644
--- a/index.html
+++ b/index.html
@@ -0,0 +1 @@
+<html></html>

commit 7f3274217ae9b3e18a6473087a12fd853f8ef572
Author: henry <javaexpert@nate.com>
Date: Thu Mar 30 17:26:52 2023 +0900

    First commit

diff --git a/index.html b/index.html
new file mode 100644
index 0000000..e69de29
henry-MacBook-Pro:GitWork henry$
```

- 12) 이렇게 **git log** 명령어는 과거의 Commit 내용을 파악하고자 다양한 Option 을 제공한다.
- 13) 한 번에 모든 Option 을 기억하는 것을 어렵기 때문에 필요할 때마다 자주 사용하도록 한다.