Git_02_LogTag

Git 온라인 리소스

Git - git-log Documentation

Git - git-tag Documentation

Log : 저장소의 수정 사항을 검토 , 로컬 저장소에서 현재 체크아웃한 브랜치의 모든커밋 정보와 메시지 확인

Log 문제를 통한 실습

Q1. 전체 커밋 로그를 확인해보자

Git 저장소에서 모든 커밋 내역을 확인하려면 git log 명령어를 사용한다.
이 명령어를 실행하면 각 커밋의 **해시 값, 작성자, 날짜, 커밋 메시지**가 출력된다.

Q2. 한 줄로 요약된 커밋 로그를 확인해보자

Git 로그를 한 줄씩 간략하게 표시하려면 git log --oneline 명령어를 사용한다.
이 명령어를 실행하면 각 커밋이 **한 줄로 출력되며, 커밋 해시와 메시지만 표시**된다.

Q3. 최근 3 개의 커밋 로그만 확인해보자

최근 N 개의 커밋만 조회하려면 git log -n 형식으로 사용할 수 있다.

Q4. 각 커밋에서 변경된 내용을 함께 확인해보자

각 커밋이 적용된 코드 변경 사항(diff)까지 보고 싶다면 git log -p 옵션을 사용한다.

- 커밋 해시: 963c0f8d8d9e1ba8ee865420fa0d1da406e53030
- 현재 HEAD(가장 최근 커밋)이며 master 브랜치에 있음.
- 커밋 메시지: [edit] hello.txt (hello.txt 파일을 수정했다는 의미)

MINGW64:/d/myTest/hello

```
$ git log  -p
commit 963c0f8d8d9e1ba8ee865420fa0d1da406e53030 (HEAD -> master)
Author:
Date:   Sun Mar 2 20:23:53 2025 +0900

    [edit] hello.txt
```

파일 변경사항(diff)

- hello.txt 파일에서 변경된 내용을 표시하고 있음.
- **+ 기호가 붙은 줄은 새로 추가된 줄**을 의미함.

```
diff --git a/hello.txt b/hello.txt
index c46cf28..02d38b2 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1,3 @@
 hello world, git!
+
+this is hello.txt
commit 4a7a6451297cdfad9afc042d7c47b46fe1f746b9
Author:
Date:   Sun Mar 2 20:21:23 2025 +0900

    [initialize] hello.txt
diff --git a/hello.txt b/hello.txt
new file mode 100644
index 0000000..c46cf28
--- /dev/null
+++ b/hello.txt
@@ -0,0 +1 @@
:
commit 963c0f8d8d9e1ba8ee865420fa0d1da406e53030 (HEAD -> master)
Author:
Date:   Sun Mar 2 20:23:53 2025 +0900

    [edit] hello.txt
```

새로 추가된 내용

초기 커밋

초기 커밋 메시지

파일 변경
새롭게 생성됨

원래 이 파일이 없었으므로, 기존 상태는 /dev/null (없음).

Q5. 최근 2 개의 커밋에 대한 변경 내용을 확인해보자

git log -p -n 옵션을 사용하면 특정 개수의 커밋에 대해 변경 내용을 확인할 수 있다.

Q6. 변경된 파일 개수 및 라인 수를 확인해보자

각 커밋에서 변경된 **파일 개수, 추가/삭제된 코드 줄 수**를 요약해서 보려면 git log --stat 옵션을 사용한다.

Q7. 특정 사용자[같은 회고록팀]가 만든 커밋만 확인해보자

특정 사용자(committer)의 커밋 로그만 보고 싶다면 git log --author="이름" 옵션을 사용한다.

Q8. 특정 날짜 이후의 커밋만 확인해보자

Git 로그에서 특정 날짜 이후의 커밋을 보려면 git log --since="YYYY-MM-DD" 옵션을 사용한다.
예를 들어, 2025 년 3 월 1 일 이후의 커밋을 보려면 다음 명령어를 실행한다.

GIT LOG --SINCE="2025-03-01"

EX) 2024 년 1 월 1 일 이후의 **커밋 로그만 출력하는 명령어를 작성하시오.**

Q9. 특정 파일이 변경된 커밋만 확인해보자

특정 파일이 변경된 커밋만 보고 싶다면 git log -- <파일명> 옵션을 사용한다.

Q10. 탐색기에서 작업폴더에서 변경상태를 확인 해보자.

1. D:₩myTest₩hello₩hello.txt 파일을 복사해서 D:₩myTest₩hello₩hello_add.txt 로 탐색기에서 작업 한다.
2. 상태 확인 한다.

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hello_add.txt

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

3. Git status –short 로 파일명만 확인 한다.

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git status --short
?? hello_add.txt
```

- ?? : **Untracked file (추적되지 않는 파일)** : hello_add.txt 는 Git 이 아직 추적하지 않는 새로운 파일임. 즉, **Git 에 추가되지 않은 파일**이며 git add 명령어를 실행해야 스테이징할 수 있음.

git status --short에서 나올 수 있는 다른 상태 코드

상태 코드	의미	해결 방법
??	Git이 추적하지 않는 파일	git add <파일명> 후 git commit -m "메시지"
A	스테이징된 새 파일	git commit -m "메시지"
M	수정됨, 스테이징됨	git commit -m "메시지"
M	수정됨, 스테이징되지 않음	git add <파일명> 후 git commit -m "메시지"
D	스테이징된 파일 삭제됨	git commit -m "삭제"
D	삭제됨, 스테이징되지 않음	git add <파일명> 후 git commit -m "삭제"
R	이름이 변경됨	git commit -m "파일 이름 변경"

다음과 같은 예시를 확인할 수 있다.

```
$ git status --short

M index.html # index.html이 수정되었으나 스테이징되지 않음

M style.css # style.css이 수정되어 스테이징됨

?? newfile.txt # newfile.txt가 Git에 추가되지 않음

 D oldfile.txt # oldfile.txt가 삭제되고, Git에 스테이징됨
```

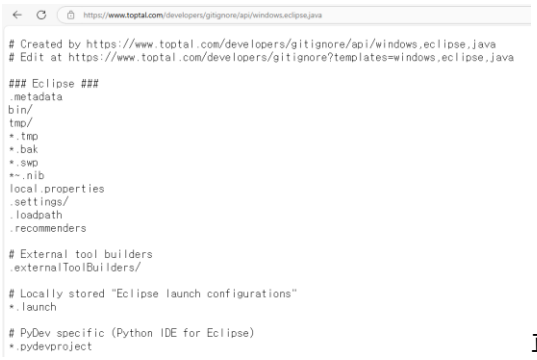
Q11. Git diff –staged 는 어떤 명령인지 Git diff –cached 의 의미를 생각해보자.

Git Ingore

gitignore.io - 자신의 프로젝트에 꼭 맞는 .gitignore 파일을 만드세요

1. 사이트 접속 , 입력

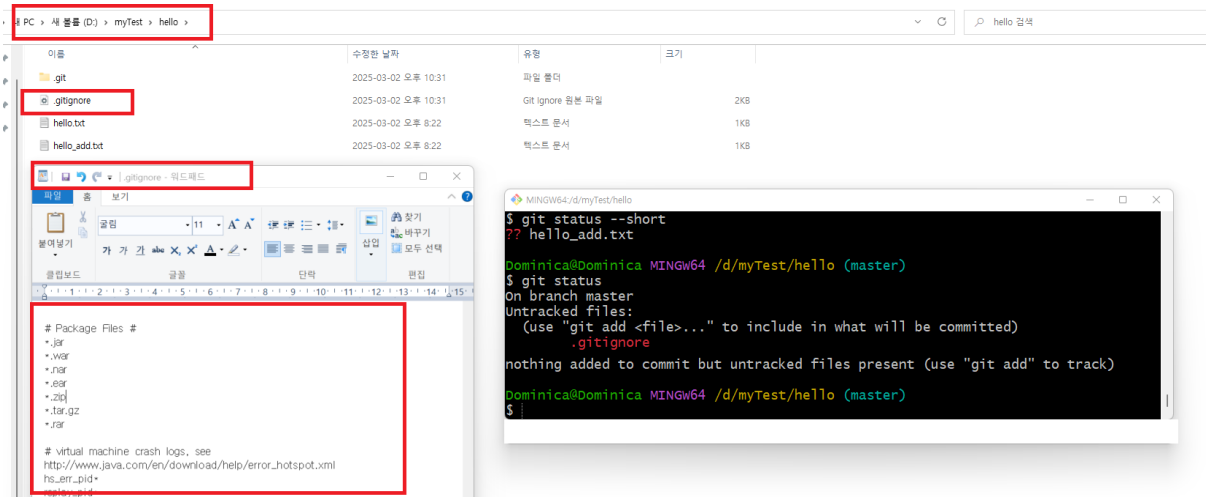
2. 파일 생성 후 저장



```
# Created by https://www.toptal.com/developers/gitignore/api/windows,eclipse,java
# Edit at https://www.toptal.com/developers/gitignore?templates=windows,eclipse,java

### Eclipse ###
.metadata
bin/
tmp/
*.tmp
*.bak
*.swp
*~.nib
local.properties
.settings/
.loadpath
.recommenders

# External tool builders
.externalToolBuilders/

# Locally stored "Eclipse launch configurations"
*.launch

# PyDev specific (Python IDE for Eclipse)
*.pydevproject
```

파일이 생성된다.

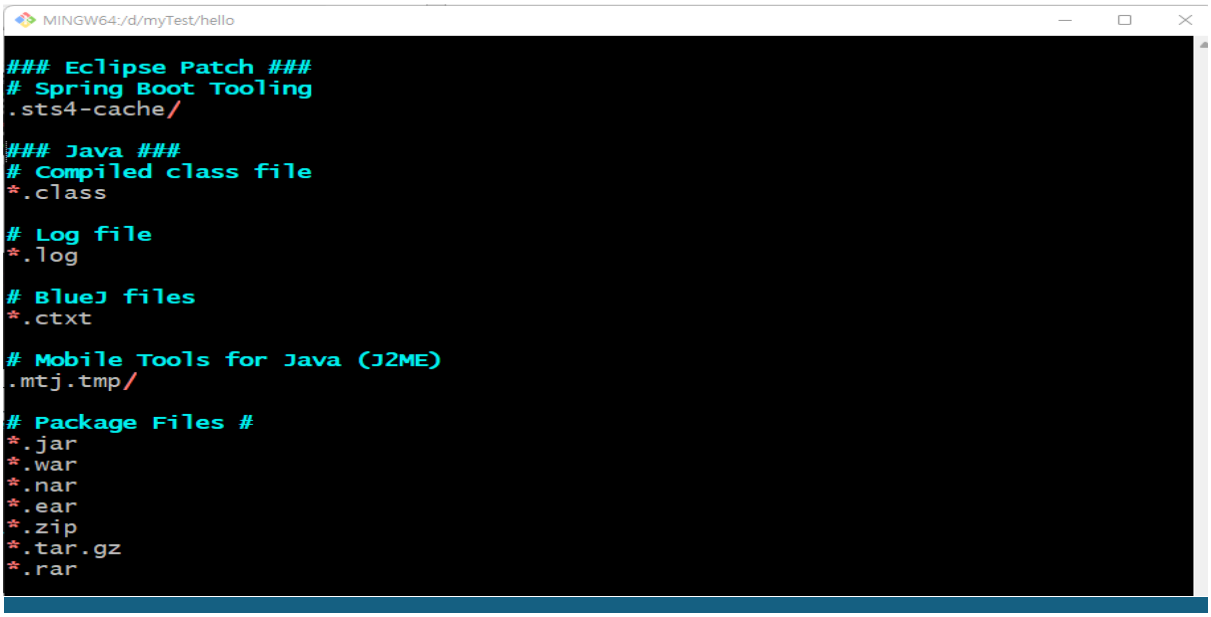
파일의 내용 전체 복사 후 .gitignore 로 저장 후 확인 한다.



3. Add 후 상태 확인

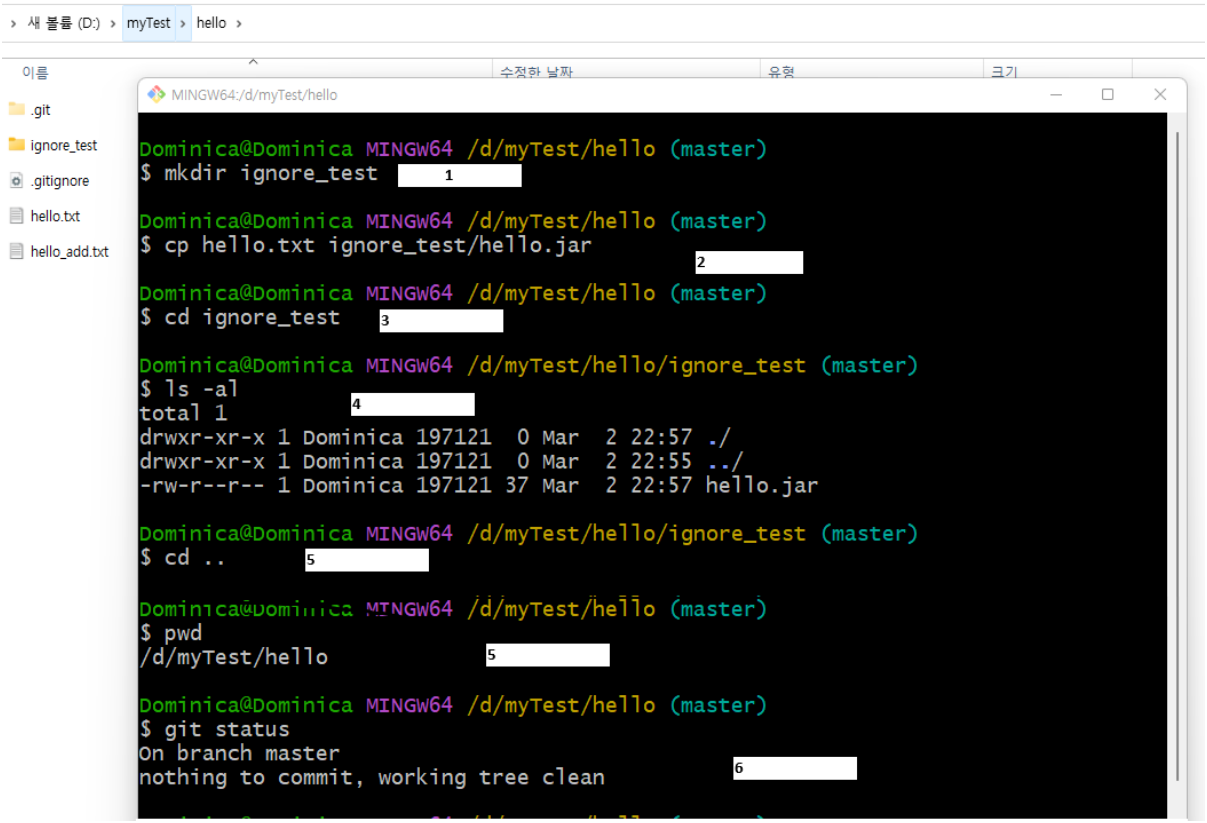
```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git commit -m "[initialize]  .gitignore"
[master 81b57e0] [initialize]  .gitignore
   files changed, 125 insertions(+)
 create mode 100644 .gitignore

Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git status
On branch master
nothing to commit, working tree clean
```

4. 내용확인 : 중간 부분에 자바로 인한 확장자는 깃에서 관리 무시 확인



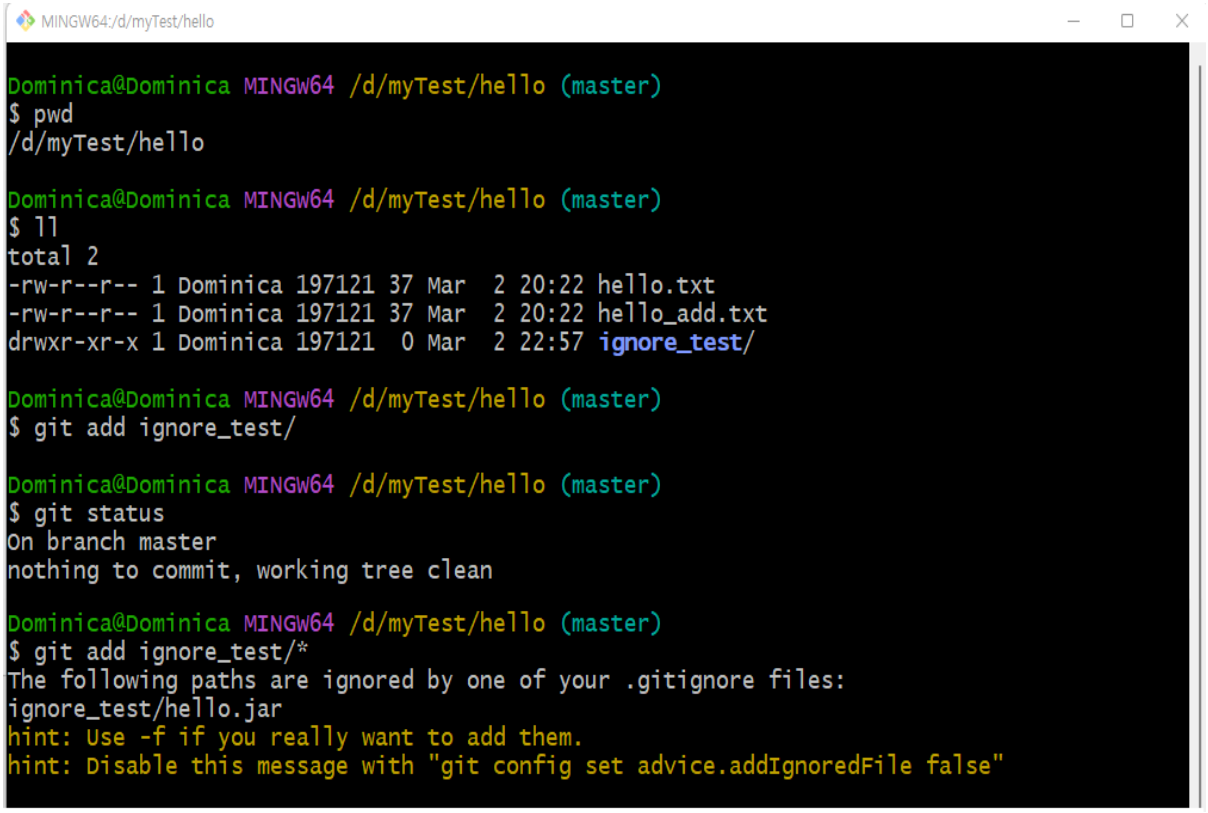
Git 실습: .gitignore 테스트 단계별 실행 명령어 및 설명

1. mkdir ignore_test : `ignore_test` 폴더 생성.
2. cp hello.txt ignore_test/hello.jar : `hello.txt`를 `hello.jar`로 복사.
3. cd ignore_test : `ignore_test` 폴더로 이동.
4. ls -al : 현재 폴더(`ignore_test`) 내 파일 목록 확인.
5. cd .. : 상위 디렉토리(`/d/myTest/hello`)로 이동.
6. pwd : 현재 디렉토리 확인.
7. git status : Git 상태 확인, 변경 사항이 없는지 확인.



Git .gitignore 적용 및 파일 추가 관련 분석

실행 로그에서 발생한 사항을 하나씩 분석하면 **Git이 특정 파일을 무시(.gitignore 적용) 하고 있는 상황**임을 확인



- .gitignore 에서 *.jar 이 무시되고 있어 hello.jar 파일이 추가되지 않은 상태.
- .gitignore 에서 해당 규칙을 삭제하거나, git add -f 로 강제 추가하면 해결 가능.
- git check-ignore -v 명령어로 .gitignore 규칙 적용 여부를 확인할 수 있음.
- git config 명령을 사용하여 경고 메시지 비활성화
  - git config --global advice.addIgnoredFile false
- Git이 무시하는 파일이라도 추가하려면 강제 추가 (-f 옵션 사용)
  - git add -f ignore_test/hello.jar

Tag : 특정위치 커밋을 찾는 책갈피

Git 태그는 크게 두 가지 종류로 나눌 수 있다.

1. Lightweight 태그 (경량 태그)

- 특정 커밋을 가리키는 단순한 포인터
- 태그 생성자, 생성 날짜, 메시지 등의 추가 정보를 저장하지 않는다.
- 빠르고 간편하게 태그를 생성할 수 있다.
- 주로 임시 태그나 개인 태그로 사용된다.

2. Annotated 태그 (주석 태그)

- 태그 생성자, 생성 날짜, 메시지 등의 추가 정보를 저장한다.
- GPG 서명을 추가하여 태그의 신뢰성을 높일 수 있다
- 협업 환경에서 태그에 대한 정보를 명확하게 전달할 수 있다.
- 주로 릴리스 버전 관리나 중요한 버전 관리에 사용된다.

명령어/옵션	설명
git tag	태그 목록을 보여줍니다.
git tag -l, --list	태그 목록을 필터링합니다.
git tag -a, --annotate	태그에 메시지를 추가합니다.
git tag -m, --message	태그 메시지를 지정합니다.
git tag -d, --delete	태그를 삭제합니다.
git tag -v, --verify	태그의 서명을 확인합니다.
git tag -s, --sign	GPG 서명을 사용하여 태그를 생성합니다.
git tag -f, --force	기존 태그를 덮어씁니다.
git tag -n, --list-n	태그 메시지의 처음 n줄을 보여줍니다.
git tag --contains	특정 커밋을 포함하는 태그를 보여줍니다.
git tag --merged	현재 브랜치에 병합된 태그를 보여줍니다.
git tag --sort	태그 목록을 정렬합니다.
git push --tags	원격 저장소에 태그를 푸시합니다.
git push origin :refs/tags/<태그 이름>	원격 저장소에서 태그를 삭제합니다.

Q1. hello.txt 파일의 변경 사항을 포함하는 커밋에 "v1.0"이라는 lightweight 태그를 생성하세요.

git tag v1.0

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git tag v1.0
```

Q2. hello_add.txt 파일을 추가한 커밋에 "feature/add_hello"라는 annotated 태그를 생성하고, 태그 메시지는 "hello_add.txt 파일 추가"로 설정하세요.

git tag -a feature/add_hello -m "hello_add.txt 파일 추가"

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git tag -a feature/add_hello -m "hello_add.txt 파 일  추 가 "

Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git show feature/add_hello
tag feature/add_hello
Tagger:
Date:   Sun Mar 2 23:39:48 2025 +0900

hello_add.txt 파 일  추 가

commit 81b57e0fb6581bc0ba88ae6e20046798f5eec8f9 (HEAD -> master, tag: v1.0, tag: feature/add_h
ello)
```

git show feature/add_hello 를 실행하면 태그 생성자, 생성 날짜, 메시지, 태그가 가리키는 커밋 정보 등을 확인할 수 있다.

- git log --decorate -oneline 로그와 함께 태그 정보 확인
- 태그 파일 디렉토리 확인 ls .git/refs/tags

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git log --decorate --oneline feature/add_hello
81b57e0 (HEAD -> master, tag: v1.0, tag: feature/add_hello) [initialize]  .gitignore
963c0f8 [edit] hello.txt
4a7a645 [initialize] hello.txt

Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ ls .git/refs/tags
feature/  v1.0
```

Q4. 생성한 모든 태그 목록을 보여주는 명령어를 작성하세요.

git tag

Q5. "v1.0" 태그를 삭제하는 명령어를 작성하세요.

git tag -d v1.0

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git tag
feature/add_hello
v1.0

Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git tag -d v1.0
Deleted tag 'v1.0' (was 81b57e0)
```

Q6. 로그 확인 후 다시 태그 생성

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git log --oneline
81b57e0 (HEAD -> master, tag: feature/add_hello) [initialize]  .gitignore
963c0f8 [edit] hello.txt
4a7a645 [initialize] hello.txt
```

- 로그 확인후 hello.txt edit 상태에 태그를 달고 커밋 정보확인

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git log --oneline
81b57e0 (HEAD -> master, tag: feature/add_hello) [initialize]  .gitignore
963c0f8 [edit] hello.txt
4a7a645 [initialize] hello.txt

Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git tag mytag 963c0f8

Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git tag
feature/add_hello
mytag

Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git show mytag
commit 963c0f8d8d9e1ba8ee865420fa0d1da406e53030 (tag: mytag)
Author:
Date:   Sun Mar 2 20:23:53 2025 +0900

    [edit] hello.txt

diff --git a/hello.txt b/hello.txt
index c46cf28..02d38b2 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1,3 @@
```

확인

위 해시값으로 mytag라는 책갈피 지정

확인

Q7. mytag 태그가 가리키는 커밋을 시작점으로 새로운 브랜치가 생성해보자.

```
Git checkout -b [브랜치이름][태그이름]
```

git checkout -b mytag mytag 명령어를 실행하면 mytag 태그가 가리키는 커밋을 기준으로 mytag 라는 이름의 새로운 브랜치가 생성되고, 현재 작업 브랜치가 해당 브랜치로 전환된다

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git log --oneline
81b57e0 (HEAD -> master, tag: feature/add_hello) [initialize]  .gitignore
963c0f8 (tag: mytag) [edit] hello.txt
4a7a645 [initialize] hello.txt
```

1. 81b57e0 (HEAD -> master, tag: feature/add_hello) [initialize] .gitignore: 81b57e0 커밋은 현재 master 브랜치의 HEAD이고, feature/add_hello 태그가 가리키는 커밋. .gitignore 파일을 초기화
2. 963c0f8 (tag: mytag) [edit] hello.txt: 963c0f8 커밋은 mytag 태그가 가리키는 커밋으로 . hello.txt 파일을 수정
3. 4a7a645 [initialize] hello.txt: 4a7a645 커밋은 hello.txt 파일을 초기화함

```
Dominica@Dominica MINGW64 /d/myTest/hello (master)
$ git checkout -b mytag mytag
Switched to a new branch 'mytag'

Dominica@Dominica MINGW64 /d/myTest/hello (mytag)
$ ls -l
total 1
-rw-r--r-- 1 Dominica 197121 37 Mar  2 20:22 hello.txt
drwxr-xr-x 1 Dominica 197121  0 Mar  2 22:57 ignore_test/
```

Switched to a new branch 'mytag' 메시지를 통해 mytag 브랜치로 전환

mytag 브랜치의 작업 디렉토리 상태를 확인

---

master 브랜치를 병합하는 방법

/d/myTest/hello (mytag) 상태에서 hello_add.txt 파일을 복사하여 mytag 브랜치에 추가하고 커밋하는 방법과 master 브랜치를 병합하는 방법

Q1. master 브랜치에서 hello_add.txt 파일 복사 및 추가하자

=> mytag 브랜치에는 mytag 태그가 가리키는 커밋 이후의 변경 사항이 포함되지 않으므로 hello_add.txt 파일이 사라진 것처럼 보인다.

hello_add.txt 파일을 다시 추가하려면 2가지 방법

- **master 브랜치에서 hello_add.txt 파일 복사: <<실습해보기>>**
  - master 브랜치로 전환.
  - hello_add.txt 파일을 복사.
  - mytag 브랜치로 전환.
  - 복사한 hello_add.txt 파일을 붙여넣는다.
  - 변경 사항을 커밋한다.
- **master 브랜치 병합: <<아래 확인>>**
  - mytag 브랜치에서 master 브랜치를 병합한다.
  - 충돌이 발생하면 충돌을 해결하고 커밋한다.

<<병합한 결과 >>

mytag 브랜치에 master 브랜치의 변경 사항이 Fast-forward 방식으로 병합

.gitignore 파일과 hello_add.txt 파일이 추가되었고, 총 125줄이 추가됨

```
Dominica@Dominica MINGW64 /d/myTest/hello (mytag)
$ git merge master
Updating 963c0f8..81b57e0
Fast-forward
 .gitignore    | 122 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 hello_add.txt |   3 ++
 2 files changed, 125 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 hello_add.txt
```