

Git / Github

inky4832@daum.net

Git 개요

Git.

분산 버전 관리 시스템 중 하나.
2005년 리누스 토르발스에 의해서 만들어짐



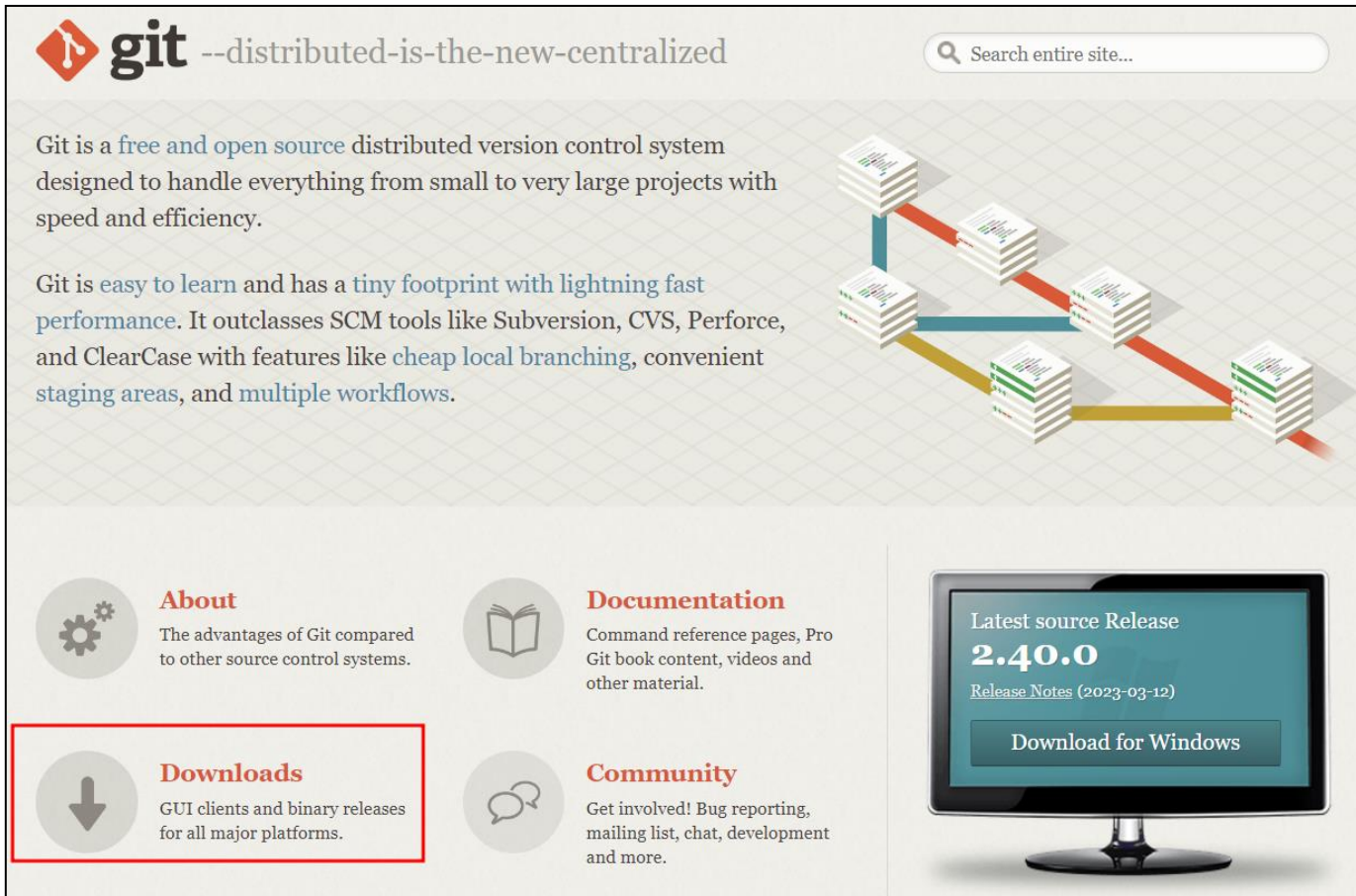
Git 특징

로컬 및 원격 저장소 생성
로컬 저장소에 파일생성 및 추가
수정 내역을 로컬 저장소에 제출
파일 수정내역 추적
원격 저장소에 제출된 수정 내역을 로컬 저장소에 적용
master에 영향을 끼치지 않는 브랜치(branch) 생성
브랜치 병합(merge)

2. Git 설치

1) Git 다운로드

<https://git-scm.com/>



The screenshot shows the Git website homepage. At the top left is the Git logo and the tagline "--distributed-is-the-new-centralized". To the right is a search bar. The main content area has two paragraphs describing Git as a free and open source distributed version control system, designed for speed and efficiency, and easy to learn with a tiny footprint and lightning fast performance. To the right of the text is a diagram showing a network of Git repositories represented as stacks of books connected by lines. Below the main text are four sections: "About" (advantages compared to other systems), "Documentation" (command reference, Pro Git book, videos), "Downloads" (GUI clients and binary releases, highlighted with a red border), and "Community" (bug reporting, mailing list, chat). On the right side of the page is a monitor displaying the "Latest source Release 2.40.0" with a "Download for Windows" button.

git --distributed-is-the-new-centralized

Search entire site...

Git is a **free and open source** distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is **easy to learn** and has a **tiny footprint with lightning fast performance**. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like **cheap local branching**, convenient staging areas, and **multiple workflows**.

About
The advantages of Git compared to other source control systems.

Documentation
Command reference pages, Pro Git book content, videos and other material.

Downloads
GUI clients and binary releases for all major platforms.

Community
Get involved! Bug reporting, mailing list, chat, development and more.

Latest source Release
2.40.0
[Release Notes \(2023-03-12\)](#)
Download for Windows

각 운영체제에 맞는 Git 프로그램 다운로드 하기

Downloads

 macOS

 Windows

 Linux/Unix

Older releases are available and the Git source repository is on GitHub.



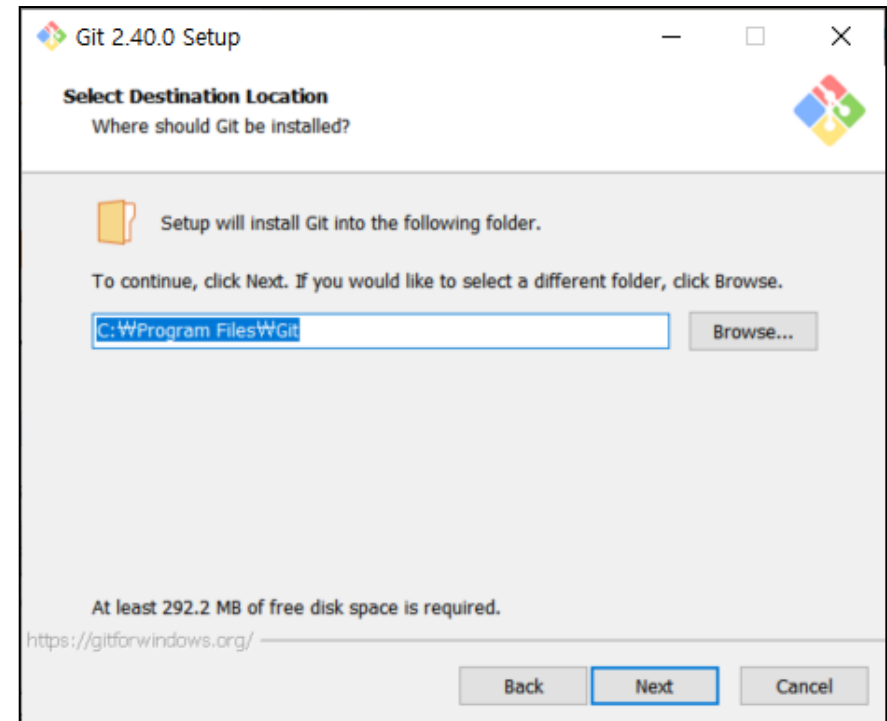
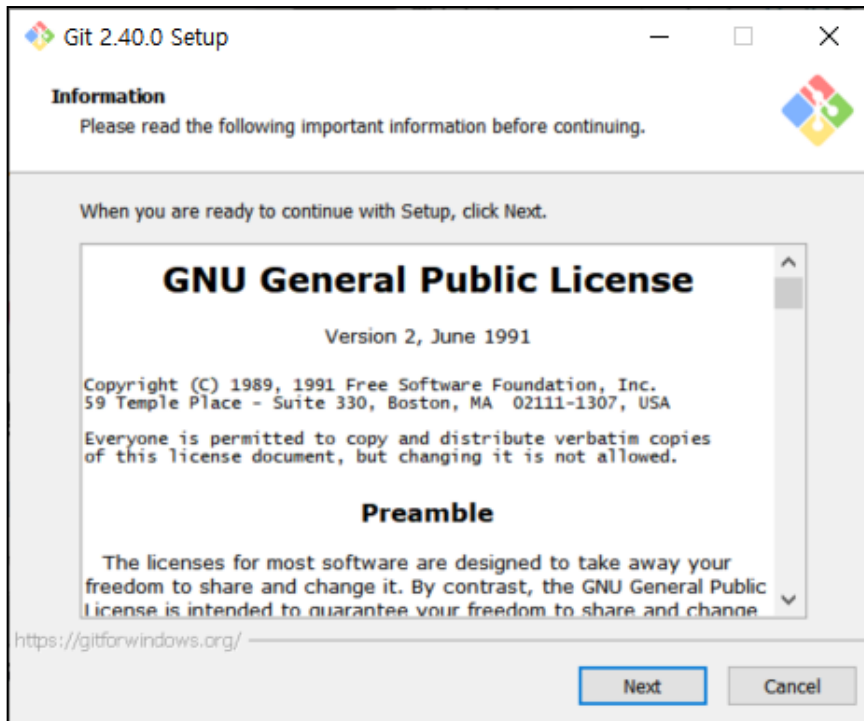
Latest source Release
2.40.0
[Release Notes \(2023-03-12\)](#)
[Download for Windows](#)

Download for Windows

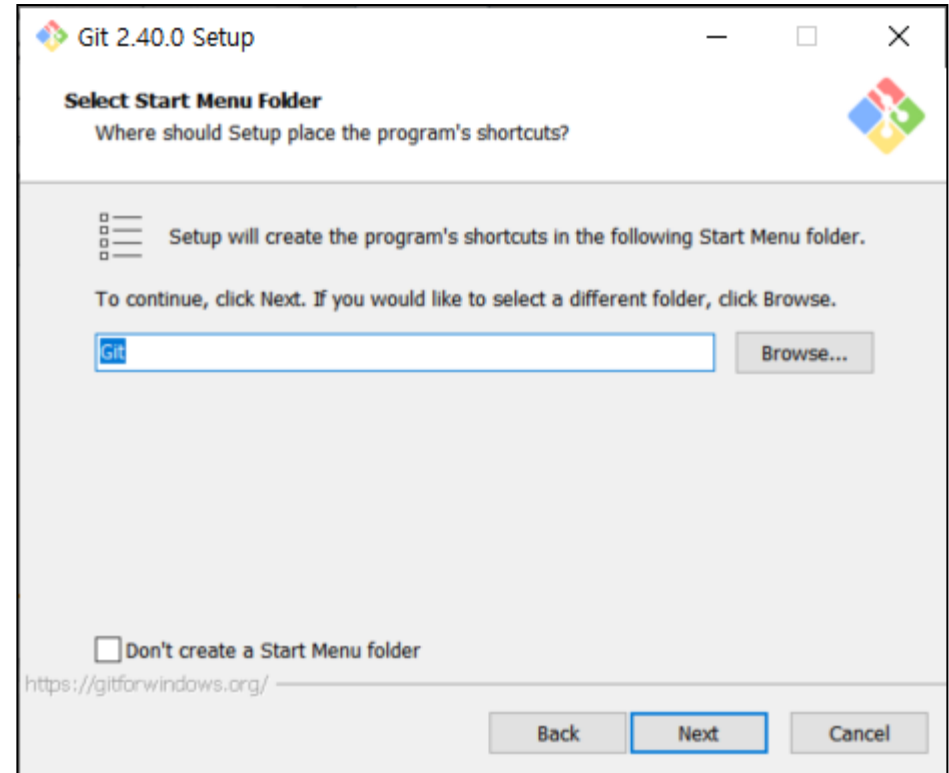
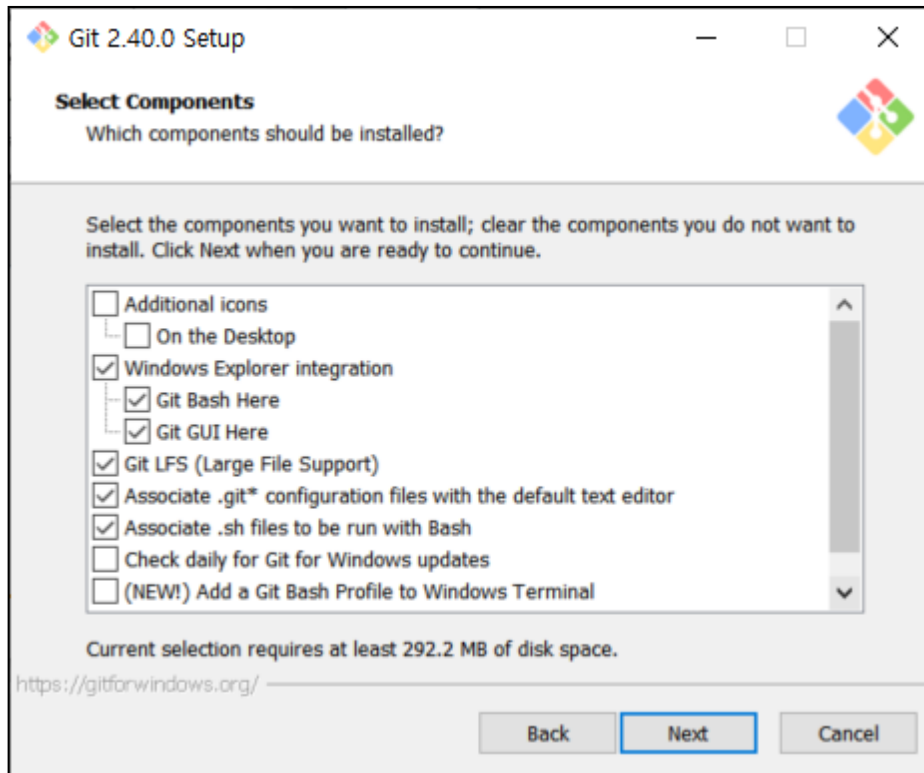
[Click here to download](#) the latest (2.40.0) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 18 days ago, on 2023-03-14.

Other Git for Windows downloads

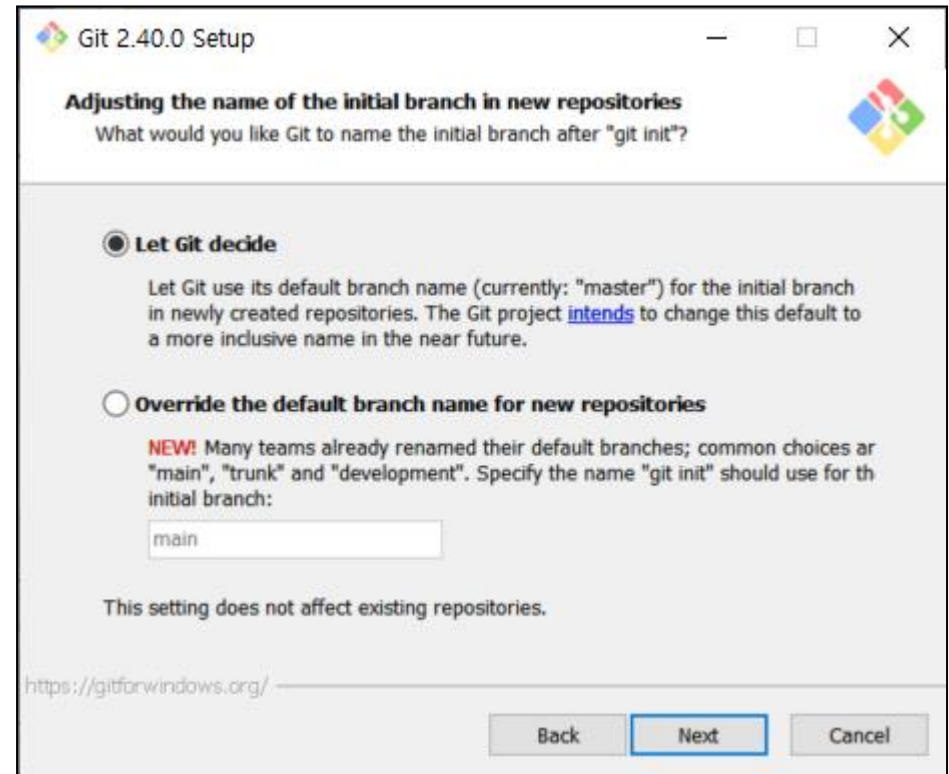
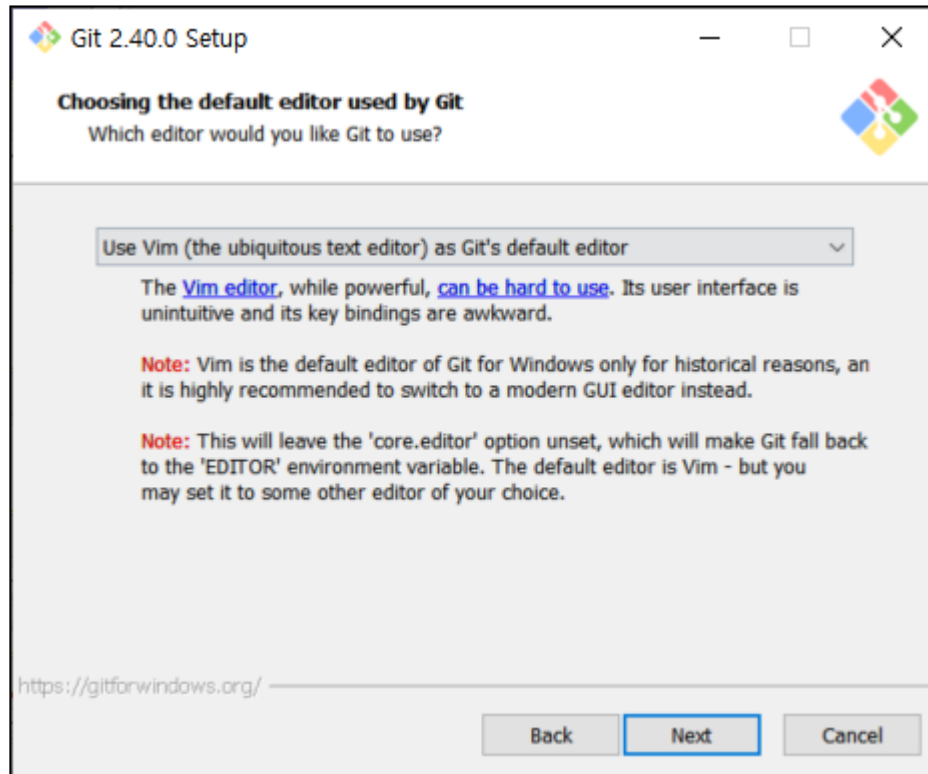
2) Git 설치



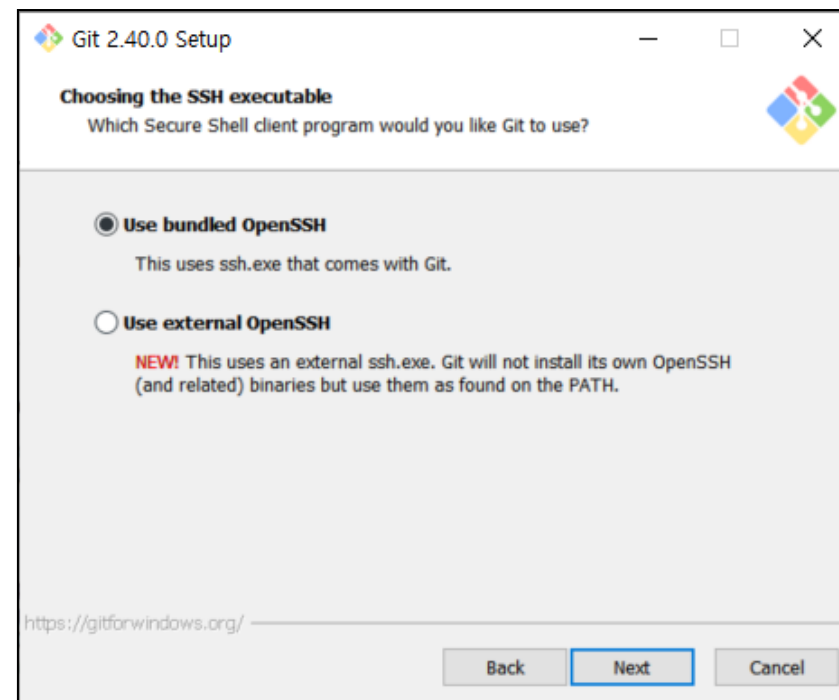
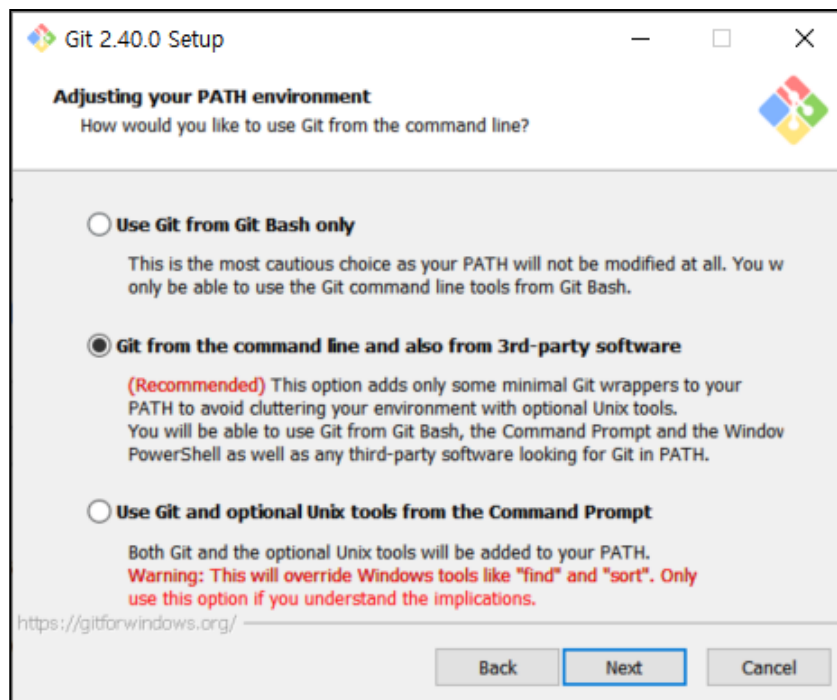
2. Git 설치



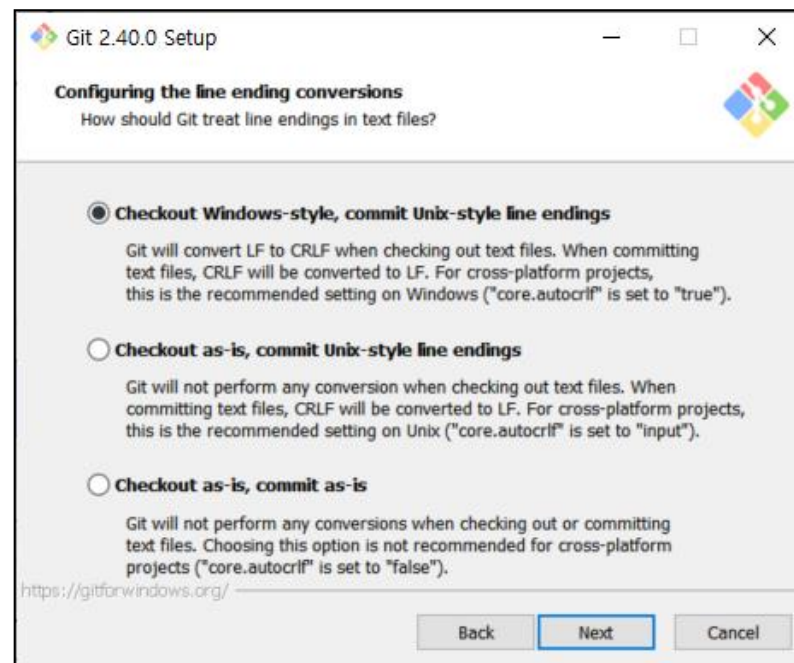
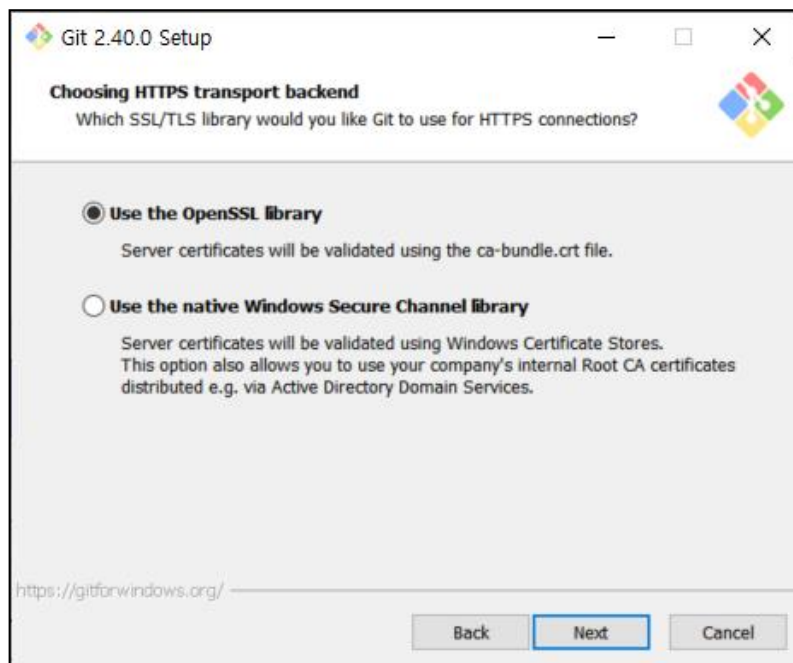
2. Git 설치



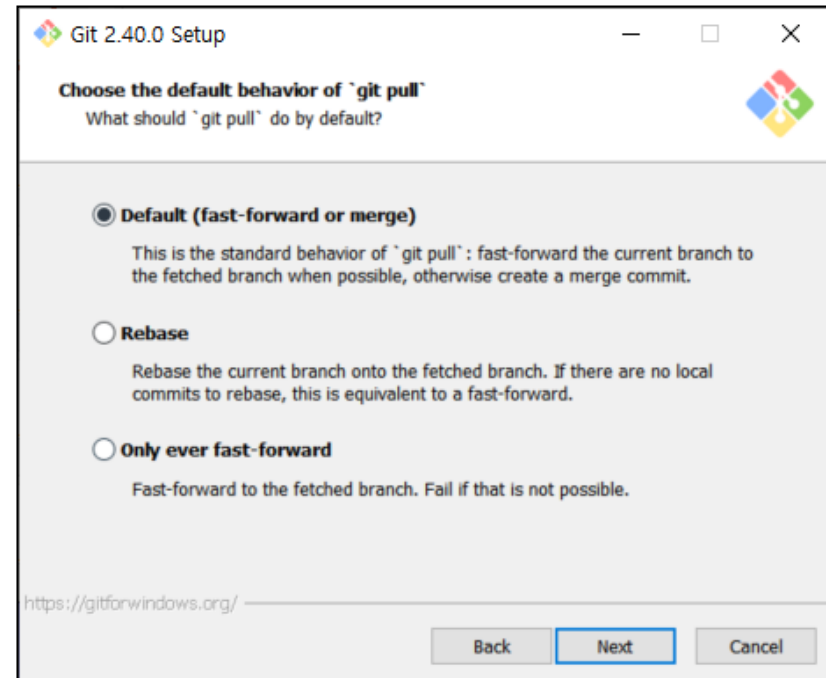
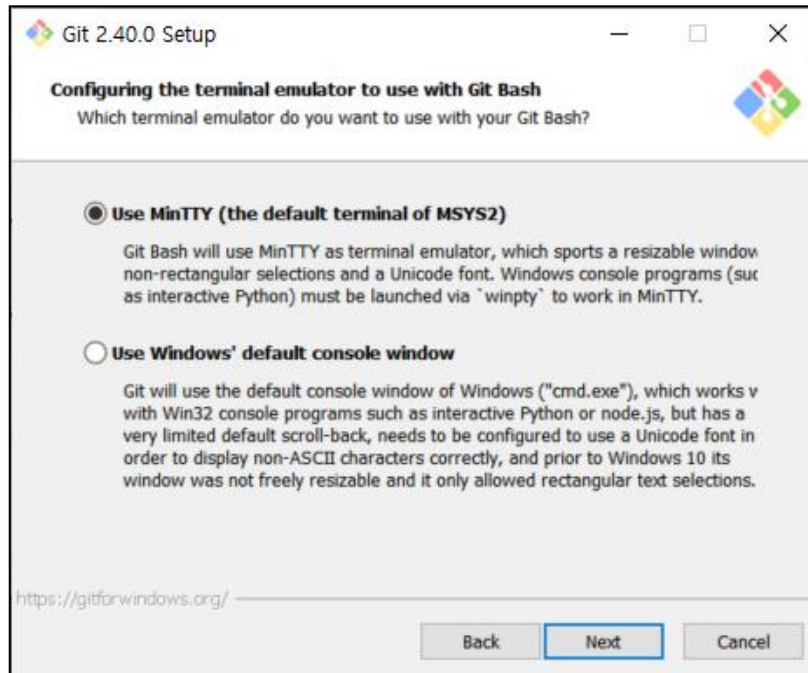
2. Git 설치



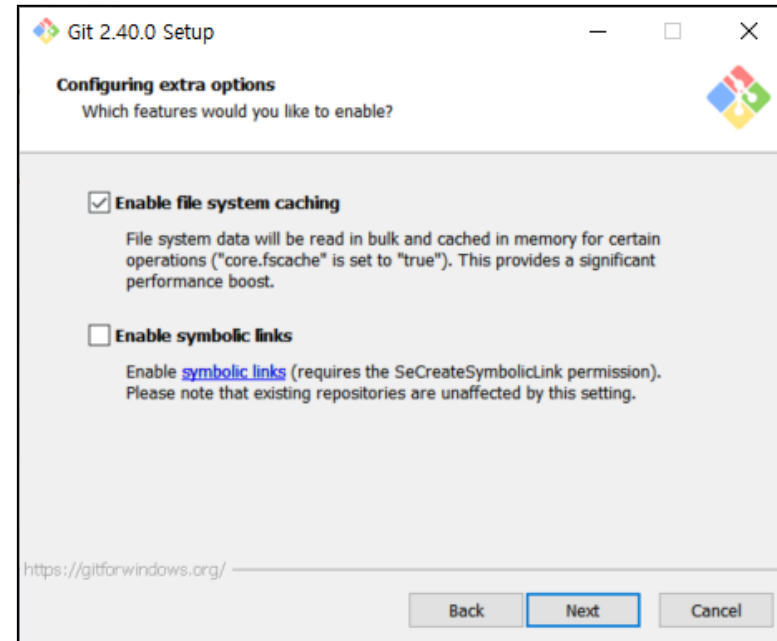
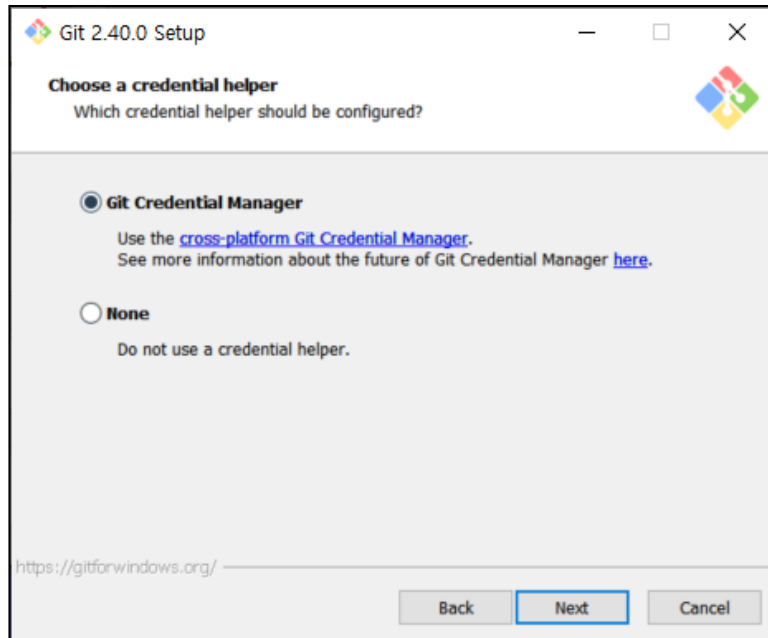
2. Git 설치



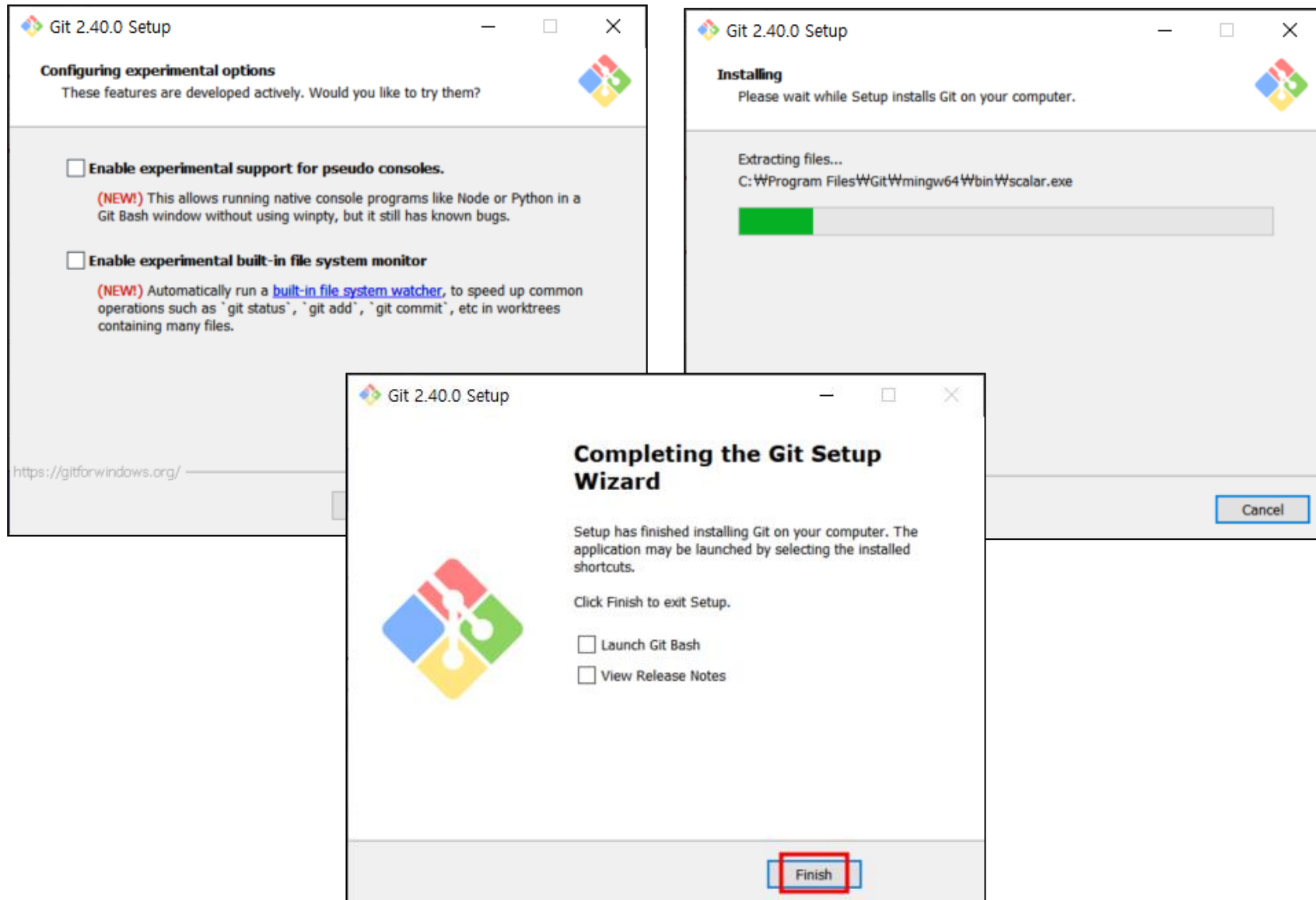
2. Git 설치



2. Git 설치



2. Git 설치



3) 사용자 정보 설정

터미널에서 git의 버전을 저장할 때 사용될 사용자 정보(이름,이메일)를 설정한다.



Git 버전 확인

```
MINGW64:/c/Users/inky4  
  
inky4@DESKTOP-410SFPS MINGW64 ~  
$ git -v  
git version 2.40.0.windows.1
```

사용자 정보 설정

```
inky4@DESKTOP-410SFPS MINGW64 ~  
$ git config --global user.name "kyin"  
  
inky4@DESKTOP-410SFPS MINGW64 ~  
$ git config --global user.email "inky4832@daum.net"
```

로컬 저장소 생성 및 관리

git 기본 명령어

저장소 관리에 필요한 git 기본 명령어

명령어	기능	설명
git init	저장소 생성	실행한 위치를 git 저장소로 지정
git add 파일명	저장소에 파일추가	해당파일을 git이 추적할 수 있도록 저장소에 추가
git commit	저장소에 수정내용 제출	변경된 파일을 저장소에 제출
git status	저장소 상태 확인	현재 저장소의 상태 출력

2. 로컬 저장소 생성

저장소 생성: git init

저장소를 만들고 싶은 디렉토리로 이동해서 git을 초기화하면 그때부터 해당 디렉터리에 있는 파일들을 버전관리 할 수 있다.

현재디렉토리 위치 확인

```
MINGW64:/c/Users/inky4  
  
inky4@DESKTOP-410SFPS MINGW64 ~  
$ pwd  
/c/Users/inky4
```

디렉토리생성 및 이동

Git저장소를 만들 디렉토리를 생성하고 해당 디렉토리로 이동

```
inky4@DESKTOP-410SFPS MINGW64 ~  
$ mkdir git_tutorial  
  
inky4@DESKTOP-410SFPS MINGW64 ~  
$ cd git_tutorial/
```

2. 로컬 저장소 생성

git 저장소 생성

git 저장소를 만들 디렉터리를 생성하고 해당 디렉터리로 이동한다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial
$ git init
Initialized empty Git repository in C:/Users/inky4/git_tutorial/.git/
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$
```

디렉터리 안의 내용을 확인하면 소스코드 버전이 저장될 저장소인 .git 디렉터리가 생성되어 있다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ ls -al
total 16
drwxr-xr-x 1 inky4 197609 0 Apr  2 10:37 ./
drwxr-xr-x 1 inky4 197609 0 Apr  2 10:11 ../
drwxr-xr-x 1 inky4 197609 0 Apr  2 10:37 .git/
```

저장소 상태 확인: git status

저장소의 상태를 확인하기 위한 방법이다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git status
On branch master

No commits yet

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

위 상태 메시지의 의미는 다음과 같다.

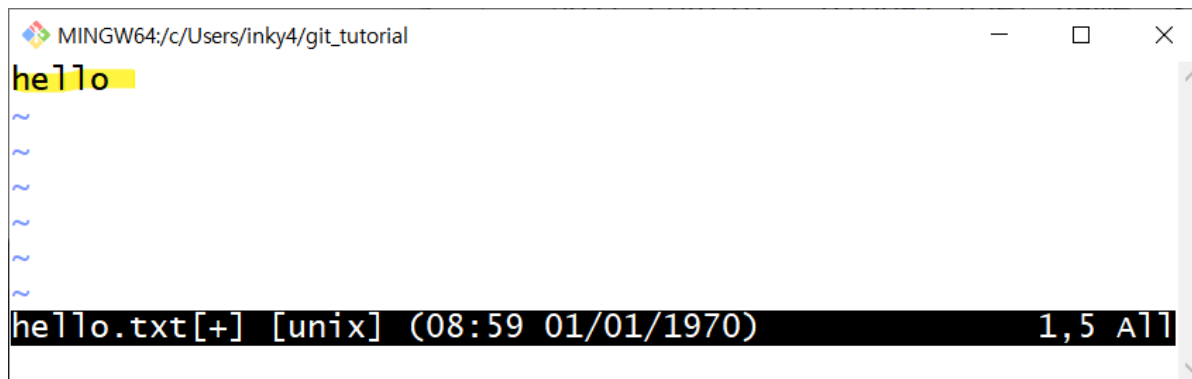
- 1) 현재 master 브랜치에 있다.
- 2) 아직 커밋한 파일이 없다.
- 3) 현재 커밋할 파일이 없다.

4. 버전으로 관리할 문서 작성

문서 작성

vim 에디터, 메모장등 편집기를 통해서 hello.txt 파일을 생성하고 파일내부에 hello 문자열을 입력한 후 저장한다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)  
$ vim hello.txt
```



- 1) a 또는 i 키보드 입력 (입력모드)
- 2) hello 문자열 입력
- 3) esc 키 (명령모드)
- 4) Shift + : 입력
- 5) :wq 입력하여 파일저장 및 종료

4. 버전으로 관리할 문서 작성

문서 작성 후 저장소 상태 확인

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git status
On branch master

No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
```

git에서는 아직 한번도 버전관리 되지 않은 파일을 untracked files라고 한다.
즉 저장소에 git이 아직 추적하지 않는 파일이 있음을 알려주는 화면이다.

5. 추적 시킬 파일 등록

추적 시킬 파일 등록: git add 파일명

git에게 버전 만들 준비를 하라고 알려주는 작업으로서 스테이징(staging) 또는 '스테이지에 올린다' 또는 인덱스(index)에 등록한다 ' 라고 부른다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git add hello.txt
warning: in the working copy of 'hello.txt', LF will
Git touches it
```

warning은 윈도우의 줄바꿈 문자와 리눅스의 줄바꿈 문자가 서로 다르기 때문에 출력되는 경고로서 무시해도 된다.

```
$ git status
On branch master

No commits yet

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

untracked files 문장이 changes to be committed로 바뀌었는데, 이것은 새파일 hello.txt 를 앞으로 커밋할 것이라는 의미이다.

버전 만들기: `git commit -m "메시지 "`

파일이 stage에 있으면 이제 버전을 만들 수 있다. git에서는 버전을 만드는 작업을 간단히 커밋(commit) 한다고 말한다. 커밋을 할 때는 그 버전에 어떤 변경사항이 있었는지 메시지를 함께 저장한다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git commit -m 'first commit'
[master (root-commit) e73900c] first commit
1 file changed, 1 insertion(+)
create mode 100644 hello.txt
```

커밋후의 결과 메시지를 보면 파일 1개가 변경되었고 파일 1개가 추가되었다. 즉 stage에 있던 hello.txt 파일이 저장소에 추가된 것이다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git status
On branch master
nothing to commit, working tree clean
```

현재 git상태를 보면 버전으로 만들 파일이 없고 working tree(작업디렉토리)도 수정사항없이 깨끗하다는 메시지가 출력된다.

버전 확인: git log 옵션

커밋 내역을 확인할 수 있는 명령어이다.

명령어	설명
git log -p	각 커밋에 적용된 실제 변경 내용을 출력
git log --stat	각 커밋에서 수정된 파일의 통계정보 출력
git log --name-only	커밋 정보중에서 수정된 파일의 목록만 출력
git log --relative-date	정확한 절대적인 시간이 아닌 상대적인 시간을 출력
git log --graph	브랜치 분기와 병합내역을 아스키 그래프로 출력
git log --oneline	한 줄에 한 커밋씩만 출력. 간략히 볼 때 사용

git log 명령어

```
$ git log
commit e73900c3a9306816201b55b9ea52194d73306433 (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 11:33:59 2023 +0900

    first commit
```

가장 최신 버전

방금 커밋한 버전에 대한 설명이 출력된다. 커밋을 만든 사람, 만든 시간, 커밋 메시지가 함께 출력된다. commit 옆의 문자는 '커밋해쉬' 라고 부르고 커밋을 구별하는 역할이다.

git log -p 명령어

```
$ git log -p
commit e73900c3a9306816201b55b9ea52194d73306433 (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 11:33:59 2023 +0900

    first commit

diff --git a/hello.txt b/hello.txt
new file mode 100644
index 0000000..ce01362
--- /dev/null
+++ b/hello.txt
@@ -0,0 +1 @@
+hello
```

git log --stat 명령어

```
$ git log --stat
commit e73900c3a9306816201b55b9ea52194d73306433 (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 11:33:59 2023 +0900

    first commit

hello.txt | 1 +
1 file changed, 1 insertion(+)
```

7. 버전 확인

git log --name-only 명령어

```
$ git log --name-only
commit e73900c3a9306816201b55b9ea52194d73306433 (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 11:33:59 2023 +0900

    first commit

hello.txt
```

git log --relative-date 명령어

```
$ git log --relative-date
commit e73900c3a9306816201b55b9ea52194d73306433 (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   83 minutes ago

    first commit
```

7. 버전 확인

git log --stat 명령어

```
$ git log --graph
* commit e73900c3a9306816201b55b9ea52194d73306433 (HEAD -> master)
   Author: kyin <inky4832@daum.net>
   Date:   Sun Apr 2 11:33:59 2023 +0900

       first commit
```

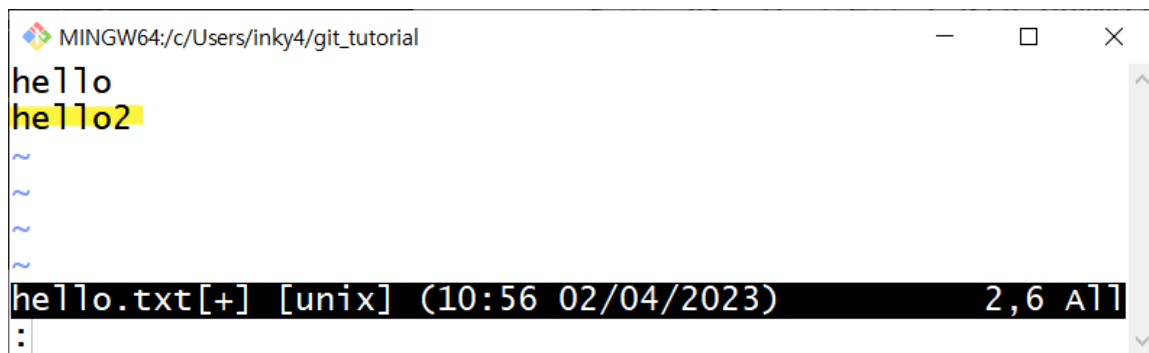
8. 스테이징과 commit 한꺼번에

```
git commit -am '메시지'
```

commit 명령어에 -am 옵션을 사용하면 파일을 스테이지에 올리고 커밋하는 과정을 한 번에 처리할 수 있다.

단 반드시 한번이라도 커밋한 적이 있어야 된다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ vim hello.txt
```



```
MINGW64:/c/Users/inky4/git_tutorial
hello
hello2
~
~
~
~
hello.txt[+] [unix] (10:56 02/04/2023) 2,6 A11
:
```

```
$ git commit -am 'second commit'
warning: in the working copy of 'hello.txt', LF will be replaced b
y CRLF the next time Git touches it
[master 3a828af] second commit
1 file changed, 1 insertion(+)
```

commit 명령어에 -am 옵션을 지정하여 스테이징과 커밋을 한 번에 처리한다.

8. 스테이징과 commit 한꺼번에

```
$ git log
commit 3a828af40ad4d0ef6af4bc8c2f8df95781/a21be (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 13:39:44 2023 +0900

    second commit

commit e73900c3a9306816201b55b9ea52194d73306433
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 11:33:59 2023 +0900

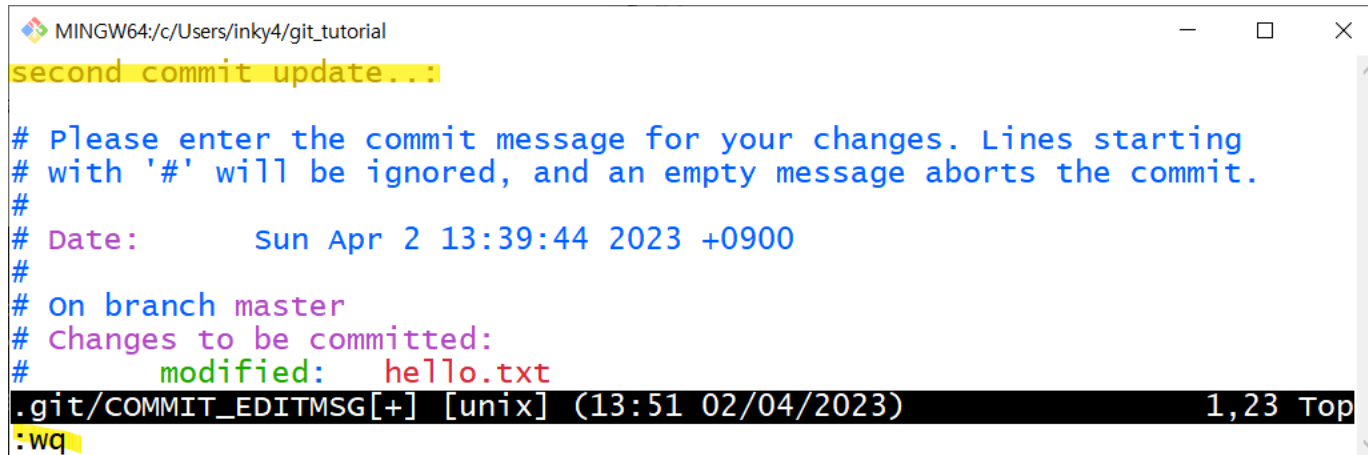
    first commit
```

9. 커밋 메시지 수정

```
git commit --amend
```

문서의 수정 내용을 기록하는 커밋 메시지를 잘못 입력했다면 커밋을 만든 즉시 커밋 메시지를 수정할 수 있다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git commit --amend
```

A screenshot of a Git commit message editor window titled 'second commit update...:'. The window shows a template for a commit message. It includes instructions to enter a message, with lines starting with '#' being ignored. It shows the current date as 'Sun Apr 2 13:39:44 2023 +0900' and the branch as 'master'. It lists the changes to be committed: 'modified: hello.txt'. At the bottom, there is a status bar showing '.git/COMMIT_EDITMSG[+] [unix] (13:51 02/04/2023) 1,23 Top' and a cursor at the end of the first line of the message body, which currently contains '.wq'.

```
second commit update...:

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun Apr 2 13:39:44 2023 +0900
#
# On branch master
# Changes to be committed:
#   modified:   hello.txt
.git/COMMIT_EDITMSG[+] [unix] (13:51 02/04/2023) 1,23 Top
.wq
```

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git commit --amend
[master 01fcab2] second commit update...
Date: Sun Apr 2 13:39:44 2023 +0900
1 file changed, 1 insertion(+)
```

9. 커밋 메시지 수정

```
$ git log
commit 01fcab284fadaa23552c6d0000ec4b7e0db4775f (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 13:39:44 2023 +0900

    second commit update..:

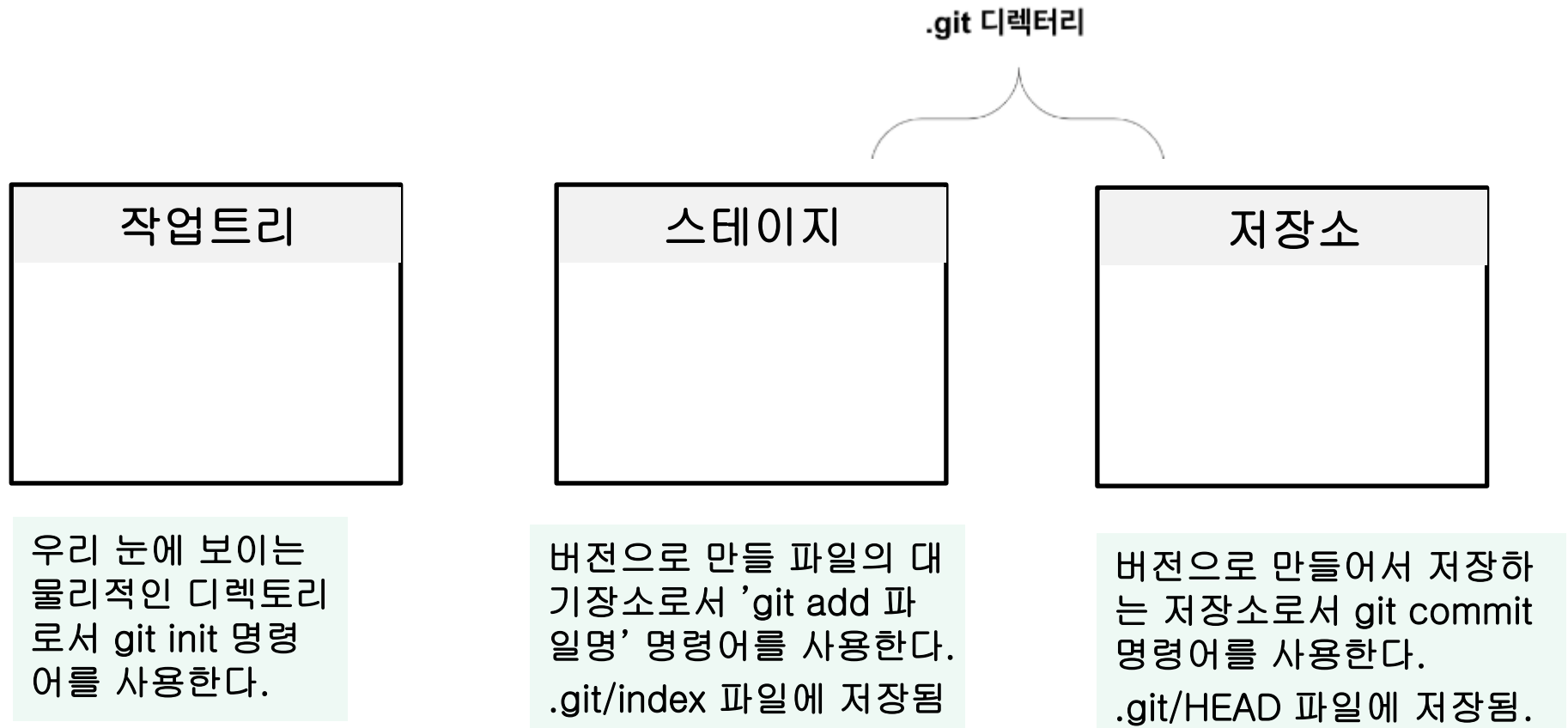
commit e73900c3a9306816201b55b9ea52194d73306433
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 11:33:59 2023 +0900

    first commit
```

원래 커밋 메시지를 수정하고 저장하면 커밋 메시지가 수정되면서 이전 커밋에 더해진다.

10. 저장소 구조

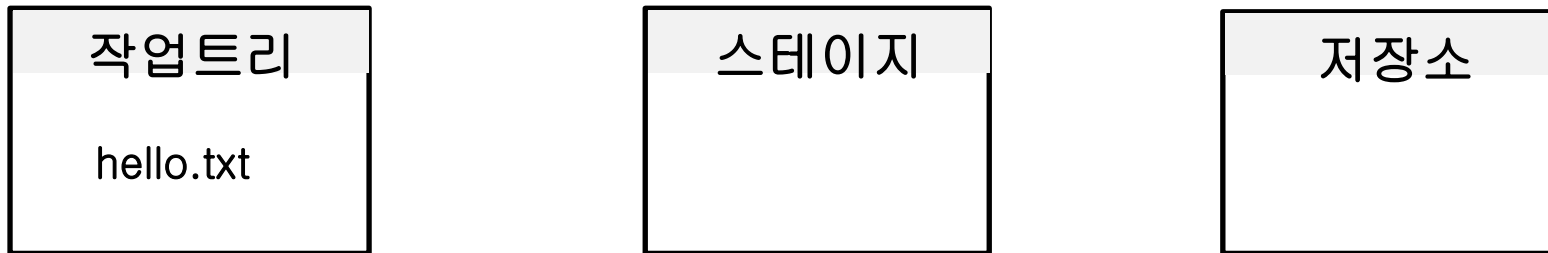
git은 관리할 파일들의 이름을 그대로 유지하면서 수정 내역을 기록한다.
이를 위해서 다음과 같은 구조를 사용한다.



10. 저장소 구조

스테이지와 저장소는 눈에 보이지 않는다. git을 초기화했을 때 만들어지는 .git 디렉터리안에 숨은 파일 형태로 존재하게 된다.

1) git init 으로 초기화하고 hello.txt 문서 작성한 후



git init

```
$ git status
On branch master

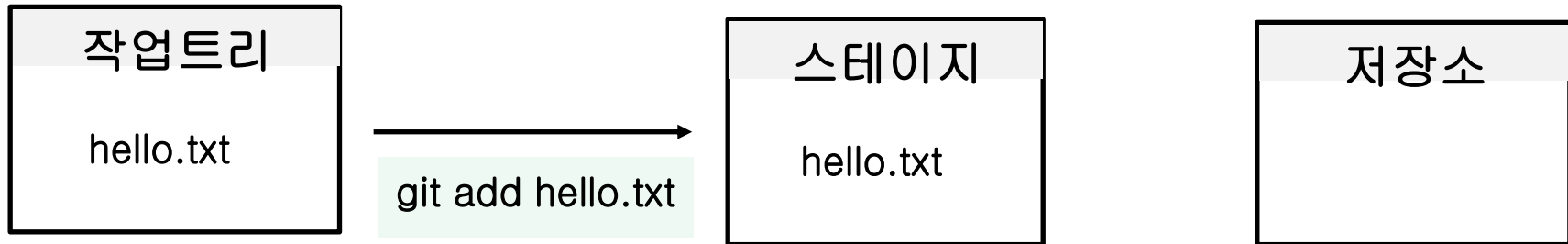
No commits yet

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

nothing added to commit but untracked files present (use "git add" to track)
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
```

10. 저장소 구조

2) git add hello.txt 명령어로 추적파일로 지정한 후



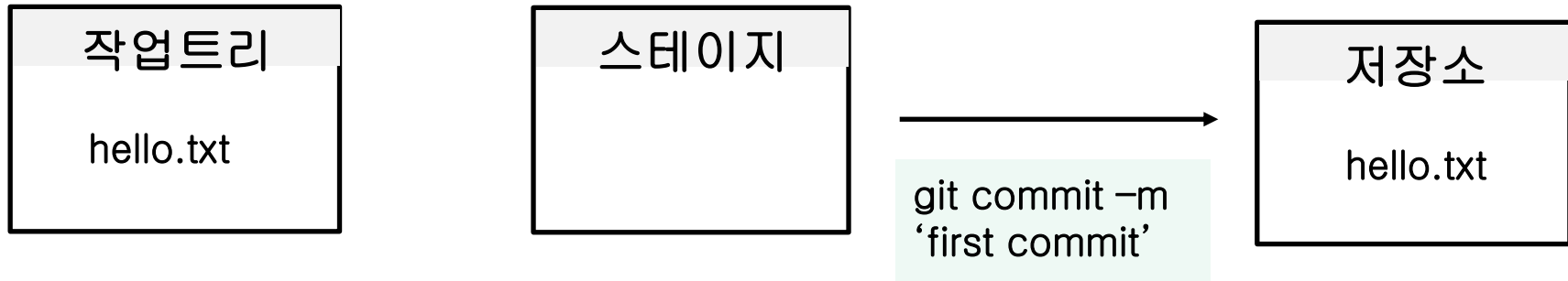
```
$ git status
On branch master

No commits yet

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

10. 저장소 구조

3) git commit -m “first commit” 명령어로 저장소에 제출한 후



```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git status
On branch master
nothing to commit, working tree clean
```

변경사항 비교

1. 변경사항 비교

git diff

git diff 명령어를 사용하면 수정내용이 포함된 작업트리에 있는 파일과 최신 커밋 내용을 비교해서 수정파일을 커밋하기 전에 최종적으로 검토할 수 있다.



1) 현재상태 확인

```
$ git status
On branch master
nothing to commit, working tree clean

inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ cat hello.txt
hello
hello2

inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
```

1. 변경사항 비교

2) 파일내용 수정

```
MINGW64:/c/Users/inky4/git_tutorial
hello
hello2
hello3
~
hello.txt[+] [unix] (13:39 02/04/2023) 3,6 All
:wq
```

작업트리

hello.txt

```
hello
hello2
hello3
```

저장소

hello.txt

```
hello
hello2
```

```
$ 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:   hello.txt

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

3) 변경사항 비교

작업트리

hello.txt

hello
hello2
hello3

저장소

hello.txt

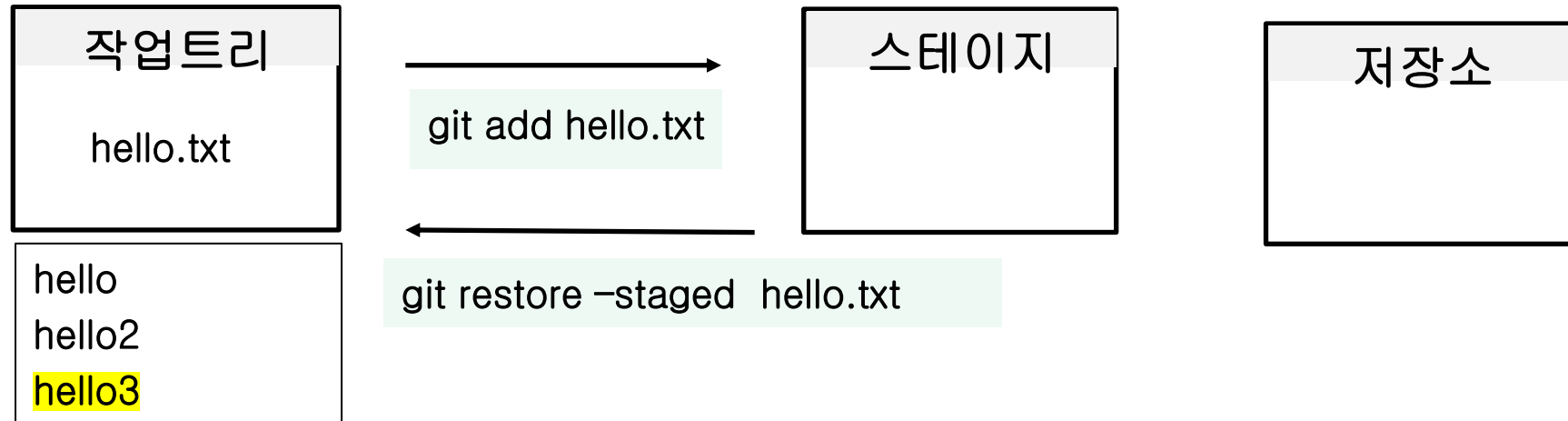
hello
hello2

```
$ git diff
warning: in the working copy of 'hello.txt', LF will be replaced by CRLF the next time
it
diff --git a/hello.txt b/hello.txt
index 97531f3..3cea797 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1,2 +1,3 @@
hello
hello2
+hello3
```

4) 최종처리 작업 (커밋 또는 무시)



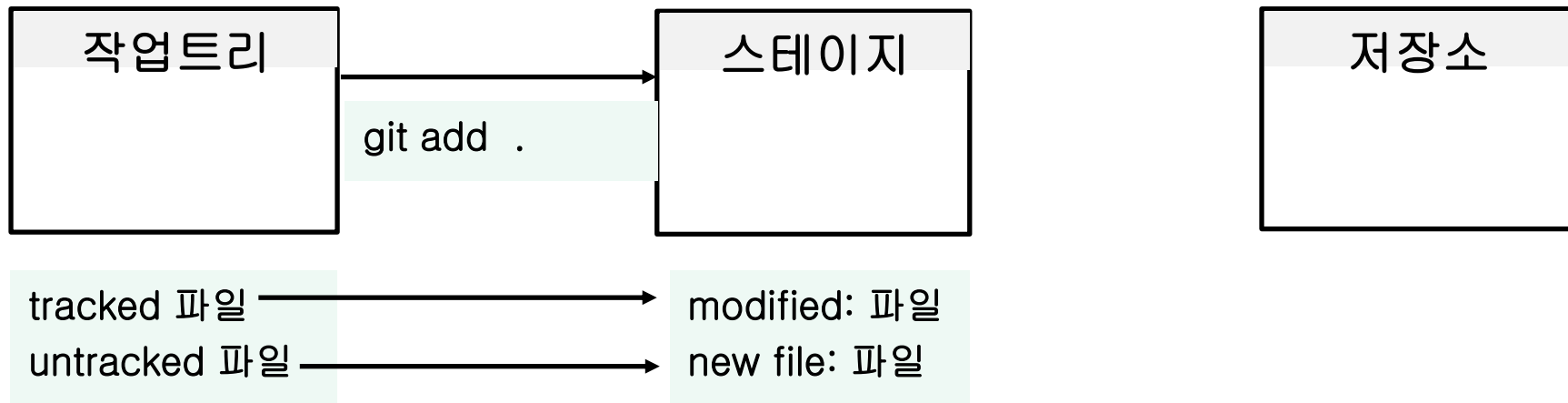
1. 변경사항 비교



버전 단계마다의 파일 상태 확인

1. 버전단계마다의 파일 상태

git에서는 버전을 만드는 각 단계마다 파일 상태를 다르게 표시한다.
따라서 파일의 상태를 이해하면 이 파일이 버전 관리의 여러 단계 중에서 어디에 있는지 알 수 있다.



2. tracked vs untracked 파일

tracked 파일은 git이 한번이라도 커밋을 한 파일의 수정여부를 계속 추적하는 파일을 의미한다.

untracked 파일은 한번도 git에서 버전관리를 하지 않은 파일을 의미한다.

1) 기존 hello.txt 파일 수정하고 새로운 world.txt 파일 추가

```
$ ls -al
total 26
drwxr-xr-x 1 inky4 197609  0 Apr  2 15:52 ./
drwxr-xr-x 1 inky4 197609  0 Apr  2 15:52 ../
drwxr-xr-x 1 inky4 197609  0 Apr  2 15:51 .git/
-rw-r--r-- 1 inky4 197609 23 Apr  2 15:50 hello.txt
-rw-r--r-- 1 inky4 197609  6 Apr  2 15:52 world.txt
```

작업트리

hello.txt

world.txt

2) 파일 상태 확인

```
$ 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:   hello.txt

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

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

hello.txt 파일은 changes not staged for commit 로서 변경된 파일이 스테이지에 올라가지 않았음을 알 수 있고 modified: 라고 표시되어 있어서 파일이 수정되었음을 알 수 있다. 이렇게 git은 한번이라도 커밋한 파일의 수정여부를 계속 추적하며 git이 추적하고 있다는 뜻에서 tracked 파일이라고 부른다.

반면에 world.txt 파일은 한번도 git이 버전관리를 하지 않았고 수정내역을 추적하지 않았기 때문에 untracked files 라고 출력된다.

2. Tracked vs untracked 파일

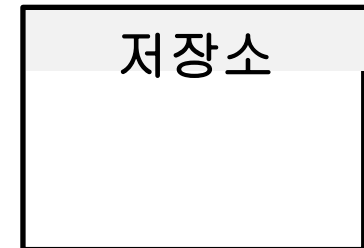
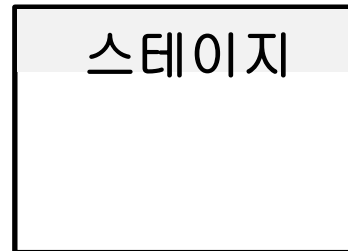
3) new file 및 modified 파일

```
$ git add .  
warning: in the working copy of 'world.txt', LF will be re  
t time Git touches it  
  
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)  
$ git status  
On branch master  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   hello.txt  
    new file:   world.txt  
  
$ git commit -m 'new file world.txt'  
[master 49d0628] new file world.txt  
2 files changed, 2 insertions(+)  
create mode 100644 world.txt
```

마지막 커밋 이후 수정된 hello.txt 파일(tracked)은 modified: 로 표시되고 버전관리가 처음인 world.txt 파일(untracked)은 new file:로 표시된다.

3. Tracked 파일의 상태 (unmodified, modified, staged)

tracked 파일은 작업트리에 있는지 스테이지에 있는지등 더 구체적인 상태를 알 수 있다.



1) 현재 수정사항이 없는 경우

```
$ git status      unmodified 상태
on branch master
nothing to commit, working tree clean
```

3. Tracked 파일의 상태 (unmodified, modified, staged)

2) 현재 수정사항이 있는 경우 (world.txt 내용 수정)

```
$ git status
On branch master
Changes not staged for commit:    modified 상태
    (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
        modified:   world.txt

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

3) 스테이징

```
$ git status
On branch master
Changes to be committed:    staged 상태
    (use "git restore --staged <file>..." to unstage)
        modified:   world.txt
```


불필요한 파일 및 폴더 무시

1. 버전 관리에서 제외

.gitignore 파일

버전 관리중인 디렉터리안에 버전 관리를 하지 않을 특정 파일이나 폴더가 있다면 .gitignore 파일을 만들고 그 안에 파일 또는 폴더명을 지정하면 된다.

1) 현재 디렉터리 목록보기

```
$ ls -al
total 26
drwxr-xr-x 1 inky4 197609  0 Apr  2 16:34 ./
drwxr-xr-x 1 inky4 197609  0 Apr  2 17:01 ../
drwxr-xr-x 1 inky4 197609  0 Apr  2 16:37 .git/
-rw-r--r-- 1 inky4 197609 23 Apr  2 15:50 hello.txt
-rw-r--r-- 1 inky4 197609 13 Apr  2 16:34 world.txt
```

2) tmp 폴더와 Test.java 파일 생성 (tmp폴더에 exam.txt 파일추가)

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ mkdir tmp

inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ touch Test.java
```

1. 버전 관리에서 제외

3) 디렉터리 목록보기

```
$ ls -al
total 26
drwxr-xr-x 1 inky4 197609  0 Apr  2 17:03 ./
drwxr-xr-x 1 inky4 197609  0 Apr  2 17:01 ../
drwxr-xr-x 1 inky4 197609  0 Apr  2 16:37 .git/
-rw-r--r-- 1 inky4 197609  0 Apr  2 17:03 Test.java
-rw-r--r-- 1 inky4 197609 23 Apr  2 15:50 hello.txt
drwxr-xr-x 1 inky4 197609  0 Apr  2 17:03 tmp/
-rw-r--r-- 1 inky4 197609 13 Apr  2 16:34 world.txt
```

4) 상태정보 확인

```
$ git status
on branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    Test.java
    tmp/

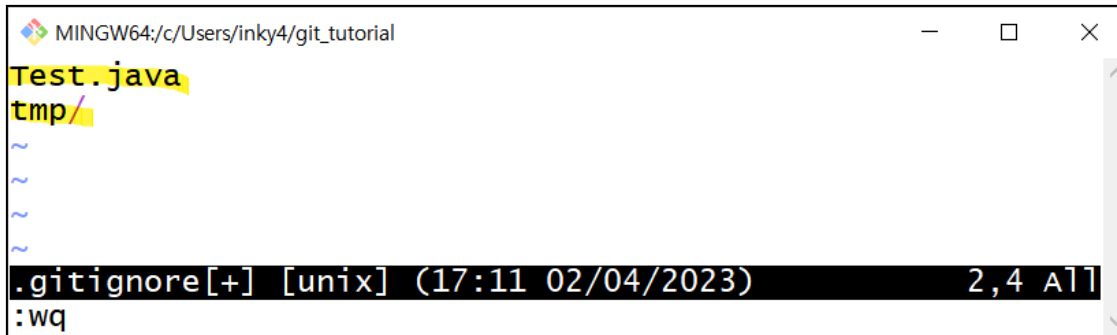
nothing added to commit but untracked files present (use "git add" to track)
```

1. 버전 관리에서 제외

5) .gitignore 파일 작성

```
inky4@DESKTOP-410SFPS MINGW64 ~/g
$ touch .gitignore

inky4@DESKTOP-410SFPS MINGW64 ~/g
$ vim .gitignore
```



```
Test.java
tmp/
~
~
~
~
.gitignore[+] [unix] (17:11 02/04/2023) 2,4 All
:wq
```

5) 상태정보 확인

```
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    .gitignore

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

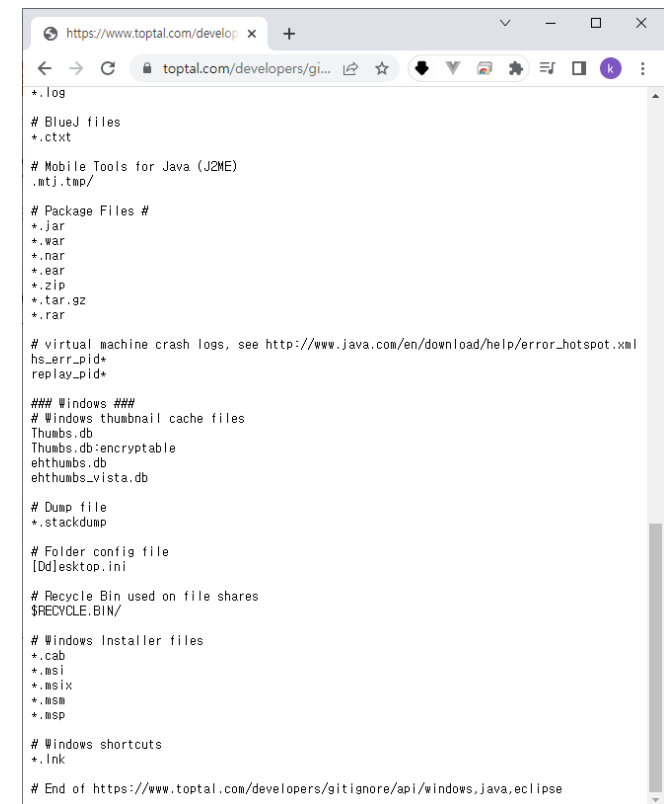
.gitignore 파일안에 명시된 Test.java 와 tmp 폴더가 버전 관리 목록에서 제외된 것을 확인할 수 있다.

2. 웹 사이트 이용

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



The screenshot shows the gitignore.io website. At the top, the logo "gitignore.io" is displayed in blue. Below it, the text "자신의 프로젝트에 꼭 맞는 .gitignore 파일을 만드세요" (Create a .gitignore file that fits your project) is shown. In the center, there is a configuration bar with three buttons: "Windows X", "Java X", and "Eclipse X". These buttons are enclosed in a red rectangular box. To the right of these buttons is a green button labeled "생성" (Generate). Below the configuration bar, there are two links: "소스 코드" (Source Code) and "커맨드라인 문서" (Command Line Docs). Two black arrows originate from the "생성" button: one points to the right and then down towards the code block, and the other points down towards the "소스 코드" link.



The screenshot shows the content of the generated .gitignore file. The text is as follows:

```
+.log
# BlueJ files
+.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
+.jar
+.war
+.nar
+.ear
+.zip
+.tar.gz
+.rar

# virtual machine crash logs, see http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
replay_pid*

### Windows ###
# Windows thumbnail cache files
Thumbs.db
Thumbs.db:encryptable
ehthumbs.db
ehthumbs_vista.db

# Dump file
+.stackdump

# Folder config file
[Dd]esktop.ini

# Recycle Bin used on file shares
$RECYCLE.BIN/

# Windows Installer files
+.cab
+.msi
+.msix
+.msm
+.msp

# Windows shortcuts
+.lnk

# End of https://www.toptal.com/developers/gitignore/api/windows,java,eclipse
```

6) 스테이징 및 커밋

```
$ git add .gitignore
warning: in the working copy of '.gitignore', LF will
    be replaced by CRLF in '.gitignore'.
    Use 'git config --local core.autocrlf true' to automatically
    convert to CRLF, or use --no-autocrlf to disable this message.

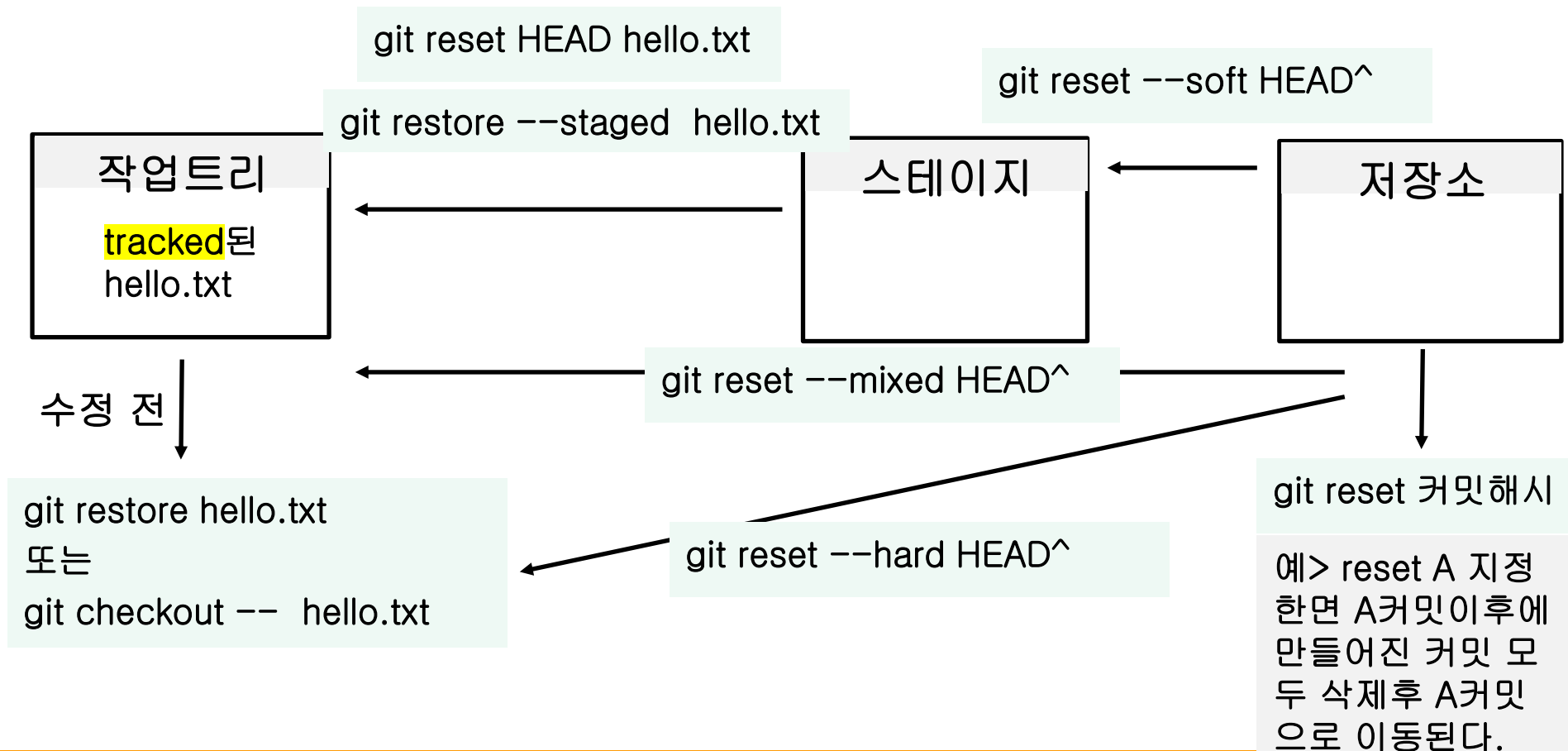
inky4@DESKTOP-410SFPS MINGW64 ~/git_tutorial (master)
$ git commit -m 'gitignore add'
[master 9ce1021] gitignore add
1 file changed, 2 insertions(+)
create mode 100644 .gitignore
```

작업 되돌리기

1. 작업 되돌리기

작업트리에서 수정했던 변경사항을 취소하거나 스테이지에 올렸던 파일을 내리거나 커밋을 취소하는등 각 단계에서 수행했던 작업을 되돌리는 방법이다.

주의할 점은 반드시 tracked 된 파일이어야 된다.



2. 작업트리에서 수정한 파일 되돌리기

```
git restore hello.txt
```

```
git checkout -- hello.txt
```

1) 새로운 저장소 생성

```
$ mkdir git_rollback  
$ cd git_rollback/  
$ git init
```

2) 파일생성 및 커밋

```
$ vim hello.txt
```

```
$ cat hello.txt  
hello
```

```
$ git add hello.txt  
$ git commit -m 'first commit'  
$ git status
```

2. 작업트리에서 수정한 파일 되돌리기

3) 파일 수정

```
$ vim hello.txt
```

```
$ cat hello.txt  
Hello  
hello2
```

4) 상태확인

```
$ 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:   hello.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")
```

2. 작업트리에서 수정한 파일 되돌리기

5) 파일 되돌리기

```
$ git checkout -- hello.txt

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git status
On branch master
nothing to commit, working tree clean

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ cat hello.txt
hello
```

3. 스테이징 파일 되돌리기

```
git reset HEAD hello.txt
```

```
git restore --staged hello.txt
```

1) 현재 파일 내용

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ cat hello.txt
hello
```

2) 파일수정후 스테이징

```
$ vim hello.txt
```

```
$ cat hello.txt
Hello
hello2
```

```
$ git add hello.txt
```

```
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello.txt
```

3) 파일 되돌리기

```
$ git reset HEAD hello.txt
Unstaged changes after reset:
M      hello.txt
```

```
$ 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:   hello.txt
```

4. 최신 커밋 되돌리기

git reset HEAD^

명령어	설명
--soft HEAD^	최근 커밋을 하기 전 스테이지 상태로 되돌리기
--mixed HEAD^	최근 커밋과 스테이징 하기 전 상태로 되돌리기
--hard HEAD^	최근 커밋,스테이징,파일수정 하기 전 상태로 되돌리기

1) 현재 파일 내용

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ cat hello.txt
hello
```

2) 파일수정후 커밋

```
$ vim hello.txt
```

```
$ cat hello.txt
Hello
hello2
```

```
$ git add hello.txt
$ git commit -m 'second commit'
[master a81877d] second commit
1 file changed, 1 insertion(+)
```

4. 최신 커밋 되돌리기

3) 커밋 내역 목록

```
$ git log
commit a81877da8653e83501d810726839f25a74524af9 (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 20:20:45 2023 +0900

    second commit

commit e96de0fa718e26e4a1a2cf8c1f9272fa17d72ccb
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 17:44:13 2023 +0900

    first commit
```

4) 최신 커밋 되돌리기 (스테이징도 함께 취소)

```
$ git reset HEAD^
Unstaged changes after reset:
M   hello.txt

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ 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:   hello.txt

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

4. 최신 커밋 되돌리기

5) 커밋 내역 목록

```
$ git log
commit e96de0fa718e26e4a1a2cf8c1f9272fa17d72ccb (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 17:44:13 2023 +0900

    first commit
```


5. 특정 커밋으로 되돌리기

```
git reset --hard 커밋해시
```

지정된 커밋해시 이후의 모든 커밋을 삭제하고 지정된 커밋으로 이동된다.
따라서 지정된 커밋이 최신 커밋이 된다.

1) 현재 파일 내용

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)  
$ cat hello.txt  
hello
```

2) 파일수정후 여러 번 커밋

```
$ cat hello.txt  
Hello  
Hello2  
hello3  
hello4
```

5. 특정 커밋으로 되돌리기

```
$ git log
commit 8e3584729dc20b02085bbf8f9f6be0b9bc1ba893 (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 21:21:28 2023 +0900

    fourth commit

commit 379b61868d800addf7791ba49c89ca0880fc4f60
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 21:20:39 2023 +0900

    third commit

commit 6cfcc3eeab7ee44dae59f5d98634540c4f7b0e5e
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 21:20:18 2023 +0900

    second commit

commit e96de0fa718e26e4a1a2cf8c1f9272fa17d72ccb
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 17:44:13 2023 +0900

    first commit
```

5. 특정 커밋으로 되돌리기

3) 특정 커밋으로 되돌리기 (second commit 으로 되돌리기)

```
commit 6cfcc3eeab7ee44dae59f5d98634540c4f7b0e5e
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 21:20:18 2023 +0900

    second commit
```



```
$ git reset --hard 6cfcc3eeab7ee44dae59f5d98634540c4f7b0e5e
HEAD is now at 6cfcc3e second commit
```

4) 커밋 내역 목록

```
$ git log
commit 6cfcc3eeab7ee44dae59f5d98634540c4f7b0e5e (HEAD -> master)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 21:20:18 2023 +0900

    second commit

commit e96de0fa718e26e4a1a2cf8c1f9272fa17d72ccb
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 17:44:13 2023 +0900

    first commit
```

5) 현재 파일 내용

```
$ cat hello.txt
Hello
Hello2
```

브랜치 (branch)

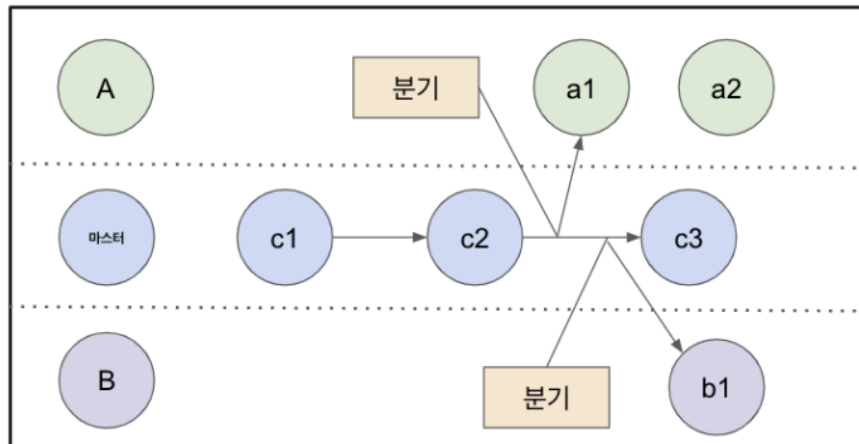
브랜치 용도

기존 파일에 새로운 기능을 추가하고자 할 때, 제대로 동작하는 소스는 그대로 둔 채 새로운 소스를 추가한 버전을 따로 만들어 관리하는 경우에 사용된다.

브랜치 기능

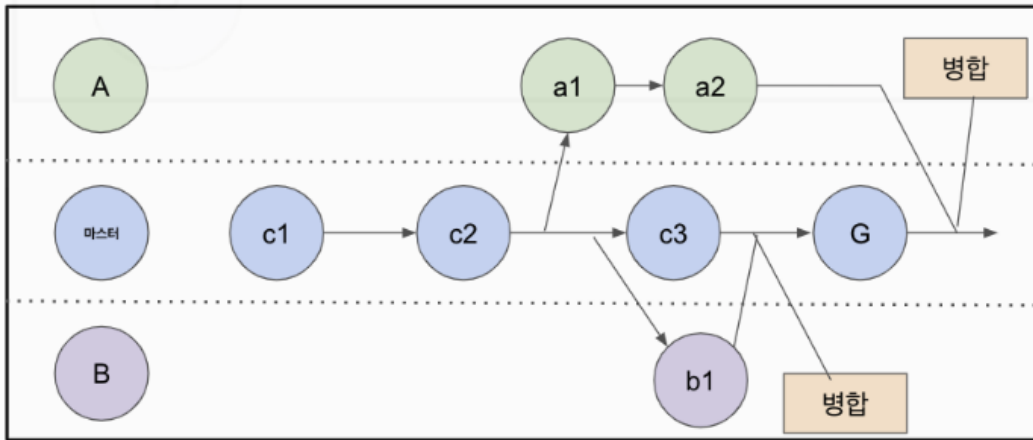
git으로 버전 관리를 시작하면 기본적으로 master 브랜치가 만들어진다. 사용자가 커밋 할 때마다 master 브랜치는 최신 커밋을 가리킨다.

이때 새로운 브랜치를 만들면 기존에 저장한 파일을 master 브랜치에 그대로 유지하면서 기존 파일 내용을 수정하거나 새로운 기능을 구현할 파일을 만들수 있다.



1. 브랜치 개요

분기했던 새로운 브랜치에서 원하는 작업을 모두 끝낸 후 새 브랜치에 있던 파일을 원래 master 브랜치에 합칠수도 있다. (병합, merge)



브랜치 관리에 필요한 git 기본 명령어

명령어	기능
git branch	브랜치 목록 보기
git branch 이름	지정된 이름의 새로운 브랜치 생성하기
git branch -d 이름	병합후에 기존 브랜치 삭제하기 병합전에 브랜치 삭제하기 위해서는 -D 옵션 사용한다.
git checkout 이름	브랜치 변경하기
git merge 이름	현재 작업중인 브랜치에서 지정된 이름의 브랜치를 가져와서 병합하기
git log --oneline --branches	모든 branch 커밋 이력을 한 줄로 출력하기
git log master..hotfix	master 브랜치 기준으로 hotfix 브랜치와의 차이점 출력

1) 브랜치 목록 보기

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git branch
* master
```

2) 브랜치 생성 (hotfix)

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git branch hotfix

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git branch
hotfix
* master
```

* 는 현재 사용중인 브랜치를 의미한다.

3) 브랜치 변경

master 브랜치에서 hotfix 브랜치로 변경한다.

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git checkout hotfix
Switched to branch 'hotfix'

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git branch
* hotfix
  master
```

지금부터 하는 모든 작업은 hotfix 브랜치만 영향을 미친다. (파일 수정, 삭제 등)
마음껏 작업하고 commi한 후에 master 브랜치로 checkout하면, hotfix에서 했던 모든 작업을 해당 브랜치의 최종 commit 상태로 보존한 후에 master 브랜치의 최종 커밋 상태로 파일들이 변경된다.

3. 브랜치 관리

4) hotfix 브랜치에서 파일 수정

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git branch
* hotfix
  master

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ cat hello.txt
hello
hello2
```

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ vim hello.txt

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ cat hello.txt
hello
hello2
hello3 hotfix
```

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git commit -am 'hello3 hotfix commit'
[hotfix bcd5be9] hello3 hotfix commit
1 file changed, 1 insertion(+)
```

3. 브랜치 관리

5) hotfix 브랜치에서 커밋 이력

```
$ git log
commit bcd5be9540757462397cbf13e4cfeab87737c145 (HEAD -> hotfix)
Author: kyin <inky4832@daum.net>
Date: Sun Apr 2 22:40:34 2023 +0900

    hello3 hotfix commit

commit 6cfcc3eeab7ee44dae59f5d98634540c4f7b0e5e (master)
Author: kyin <inky4832@daum.net>
Date: Sun Apr 2 21:20:18 2023 +0900

    second commit

commit e96de0fa718e26e4a1a2cf8c1f9272fa17d72ccb
Author: kyin <inky4832@daum.net>
Date: Sun Apr 2 17:44:13 2023 +0900

    first commit

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
```

6) 모든 브랜치에서 커밋 이력

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git log --online --branches
bcd5be9 (HEAD -> hotfix) hello3 hotfix commit
6cfcc3e (master) second commit
e96de0f first commit
```

7) master 브랜치 기준으로 hotfix 브랜치 차이점 출력

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git checkout master
Switched to branch 'master'

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git log master..hotfix
commit bcd5be9540757462397cbf13e4cfeab87737c145 (hotfix)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 22:40:34 2023 +0900

    hello3 hotfix commit
```

master 브랜치에는 없고 hotfix 브랜치에만 있는 커밋 정보 출력.

브랜치 병합(merge)

만들어진 각 브랜치에서 작업을 하다가 어느 시점에는 브랜치 작업을 마무리하고 기존 master 브랜치와 합해야 된다. 이것을 브랜치 병합(merge)이라고 부른다.

1) master 브랜치 변경

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git checkout master
Switched to branch 'master'
```

2) master 브랜치와 hotfix 브랜치 병합

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git merge hotfix
Updating 6cfcc3e..bcd5be9
Fast-forward
 hello.txt | 1 +
 1 file changed, 1 insertion(+)
```

만약 master 브랜치에서 분기한 후에 master 브랜치에서 변경 사항이 없다면 분기한 브랜치를 병합하는 것은 매우 간단하다. 분기한 브랜치에서 만든 최신 커밋을 master 브랜치가 참조하도록 하면 된다. 이것을 Fast-forward (빨리 감기 병합)라고 부른다.

3) master 커밋 이력 보기

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git branch
  hotfix
* master

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git log
commit bcd5be9540757462397cbf13e4cfeab87737c145 (HEAD -> master, hotfix)
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 22:40:34 2023 +0900

    hello3 hotfix commit

commit 6cfcc3eeab7ee44dae59f5d98634540c4f7b0e5e
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 21:20:18 2023 +0900

    second commit

commit e96de0fa718e26e4a1a2cf8c1f9272fa17d72ccb
Author: kyin <inky4832@daum.net>
Date:   Sun Apr 2 17:44:13 2023 +0900

    first commit
```

브랜치 충돌(conflict)

git에서는 줄 단위(line)로 변경 여부를 확인한다.

따라서 브랜치에 같은 파일의 이름을 가지고 있으면서 같은 줄을 수정하고 병합하면 브랜치 충돌(conflict)이 발생된다.

해결방법은 명시적으로 사용자가 충돌위치를 수정하고 커밋해서 처리한다.

1) 현재 상태 보기

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git log master..hotfix

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git log hotfix..master
```

2) master 브랜치에서 hello.txt 수정

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ cat hello.txt
hello
hello2
hello3 hotfix
hello4 master
```

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git commit -am 'hello4 master commit'
[master 473d6cd] hello4 master commit
1 file changed, 1 insertion(+)
```

3) hotfix 브랜치에서 hello.txt 수정


```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ cat hello.txt
hello
hello2
hello3 hotfix
hello4 hotfix
```

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git commit -am 'hello4 hotfix commit'
[hotfix 0fdb9f9] hello4 hotfix commit
1 file changed, 1 insertion(+)
```

Git 활용한 버전관리

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (hotfix)
$ git checkout master
Switched to branch 'master'

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git merge hotfix
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.
```



A screenshot of a Windows command prompt window. The title bar shows the path "MINGW64/c:/Users/inky4/git_rollback". The command prompt shows the following output:

```

hello
hello2
hello3 hotfix
<<<<<< HEAD
hello4 master
=====
hello4 hotfix
>>>>>> hotfix

```

The lines from "<<<<<< HEAD" to ">>>>>> hotfix" are enclosed in a red rectangular box. At the bottom of the window, a status bar displays "hello.txt [dos] (00:01 03/04/2023) 4,12 All".



A screenshot of a Windows command prompt window. The title bar shows the path "MINGW64/c/Users/inky4/git_rollback". The command prompt shows the command "git log" being executed. The output of the command is displayed in a monospaced font. The first three lines of the output are highlighted with a red rectangular box: "hello3 hotfix", "hello4 master", and "hello4 hotfix". The rest of the output is partially visible at the bottom of the window.

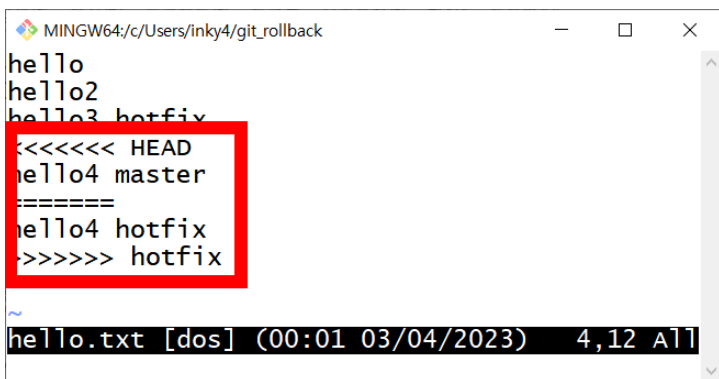
```
hello
hello2
hello3 hotfix
hello4 master
hello4 hotfix
~
~
~
<1lo.txt[+] [dos] (00:01 03/04/2023)6,0-1 All
:wg
```

- 80 -

5. 브랜치 충돌

4) 충돌시 명시적으로 해결

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master|MERGING)
$ vim hello.txt
```



5) 변경사항 커밋

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master|MERGING)
$ git commit -am 'master hotfix merge'
[master e60dded] master hotfix merge
```

6) 커밋 이력 보기

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git log --oneline
e60dded (HEAD -> master) master hotfix merge
0fdb9f9 (hotfix) hello4 hotfix commit
473d6cd hello4 master commit
bcd5be9 hello3 hotfix commit
6cfcc3e second commit
e96de0f first commit
```

6. 병합시킨 브랜치 삭제

병합시킨 hotfix 브랜치 삭제

```
inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git branch
  hotfix
* master

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git branch -d hotfix
Deleted branch hotfix (was 0fdb9f9).

inky4@DESKTOP-410SFPS MINGW64 ~/git_rollback (master)
$ git branch
* master
```



Thank you
