

# Git Tutorial

'24W










송 인 식

# Outline

- 버전 관리 시스템
- Git
- 저장소와 커밋
- 브랜치
- 병합
- Git 고급 기능

# 버전 관리 시스템

- 파일을 편집 전 상태로 되돌리고 싶을 때 가장 간단한 방법: 편집하기 전에 파일을 미리 복사하기
  - 파일과 폴더명 뒤에 편집한 날짜를 붙여주는 방식.
  - 파일을 편집할 때마다 매번 복사하는 일은 번거롭기도 하고 실수할 가능성도 많음

Name	
	120525_document_updated.txt
	120604_document.txt
	120605_document_amended.txt
	120605_document_John.txt
	120605_document_latest.txt
	120605_document_latestcopy.txt
	120605_document.txt
	1200602_document.txt
	document_meeting.txt

# 버전 관리 시스템

- 이렇게 여러 명이 공유한 파일을 동시에 편집하는 바람에 다른 사람이 먼저 변경하고 있던 내용을 지워버릴 수 있음



버전 관리 시스템 사용 전

버전 관리 시스템 사용 후

# 버전 관리 시스템

- (주로 텍스트)파일들을 누가 어떻게 바꿨는지 기록하고
- 기록한 내용을 쉽게 조회하고
- 특정 버전으로 되돌리고 복구하고
- 여럿이 함께 개발하고 공유하기 편리한 시스템

# Outline

- 버전 관리 시스템
- Git
- 저장소와 커밋
- 브랜치
- 병합
- Git 고급 기능
- GitHub

# Git

- 사실상 표준
- “분산” 버전 관리 시스템
- 가볍고 빠르고 효과적
- 버전간 차이가 아니라 (똑똑하게) 전체 사본(snapshot)을 보관
- 거의 모든 작업을 로컬에서 할 수 있음
- 강력한 브랜치 기능

# 분산 버전 관리와 중앙 버전 관리

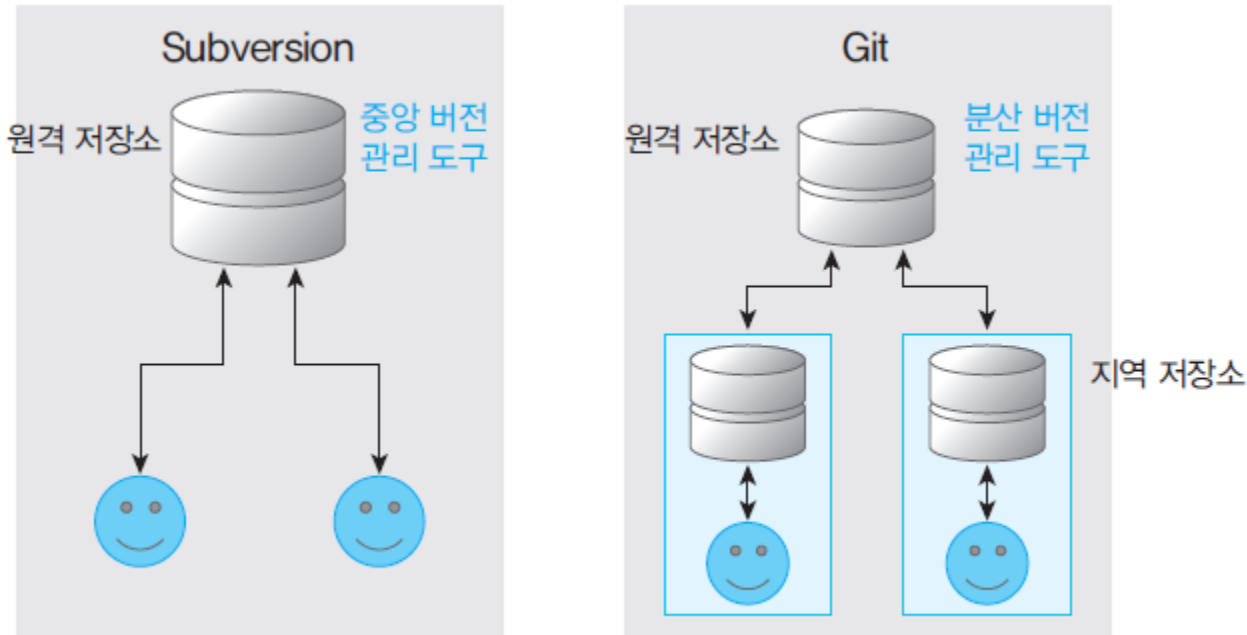


그림 9-1 Git은 원격 저장소와 지역 저장소 두 개가 존재한다.

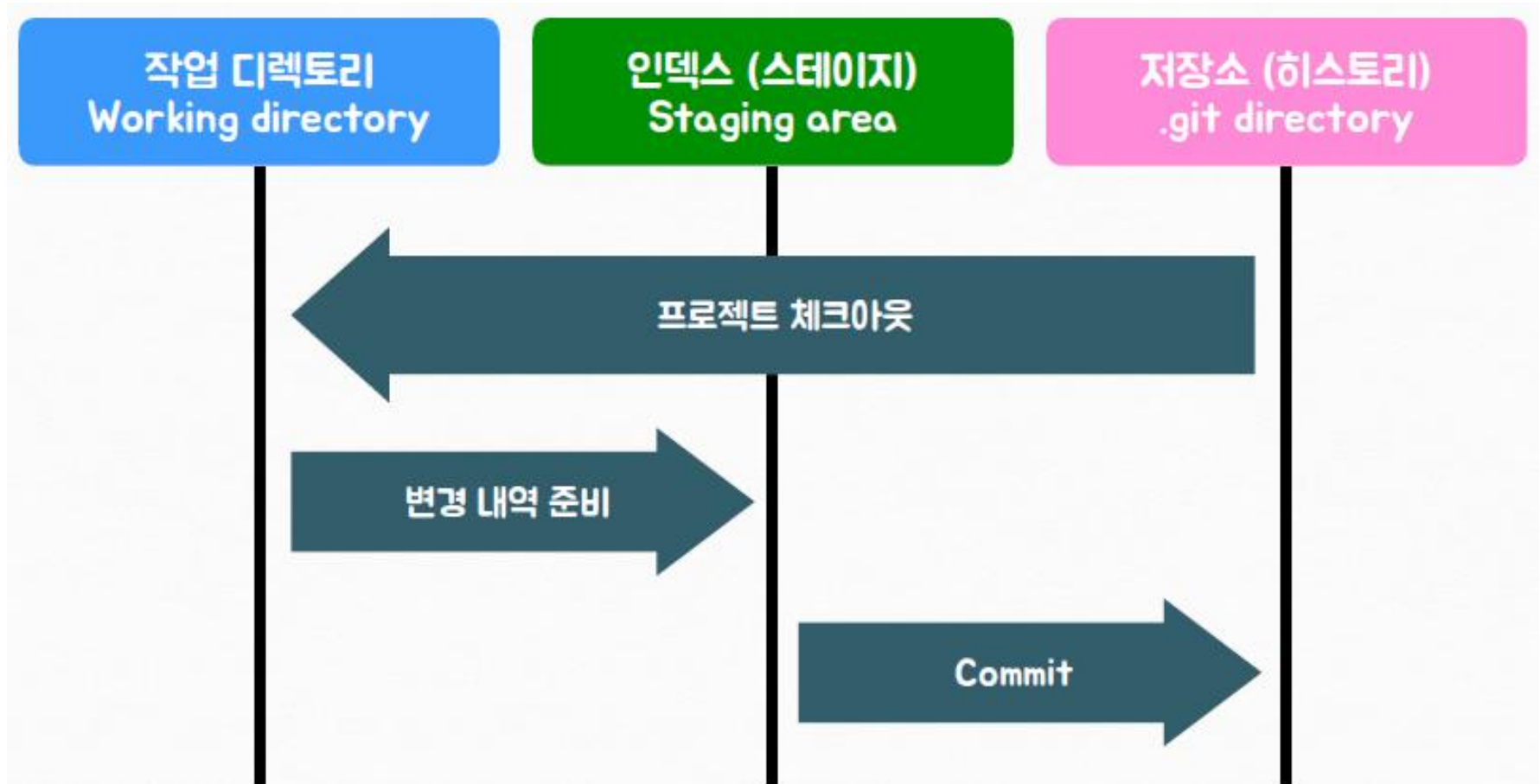
Git은 따라서 메인 서버나 통신에 문제가 있어도 지역 저장소를 이용하여 작업을 계속 수행할 수 있다



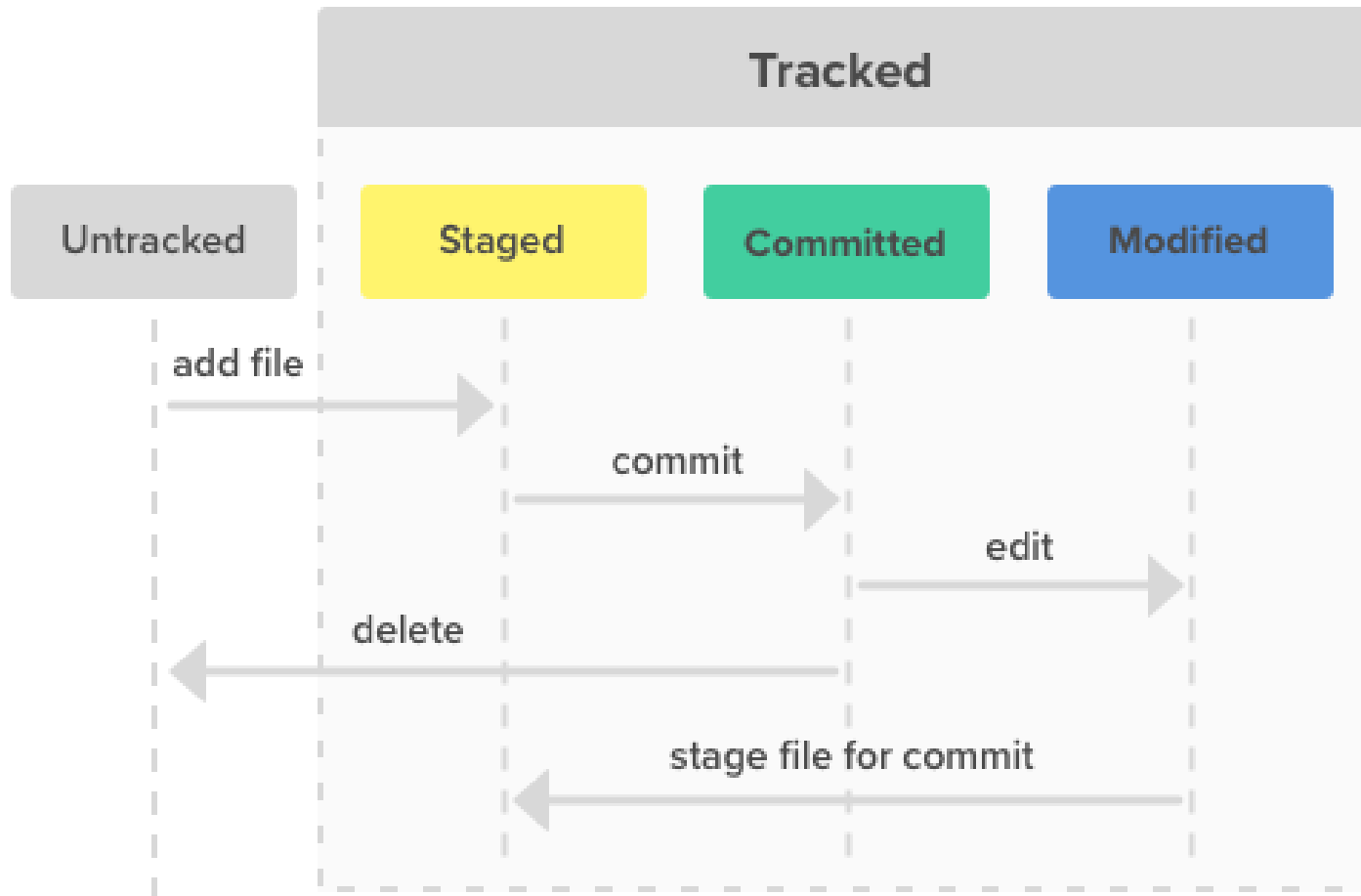
# DVCS 장점

- 네트워크 단절 OK
- 로컬에 빠르고 가볍게 작업
- 혼자만의 실험 브랜치 진행
- 다른 사람과 공유 전에 커밋을 정리 정돈

# Git 프로젝트의 세 영역



# Git 파일의 상태



# Outline

- 버전 관리 시스템
- Git
- 저장소와 커밋
- 브랜치
- 병합
- Git 고급 기능

# 준비

- Git 설치
- 다양한 GUI 툴이 있으나 CLI로 작업
- CLI 툴은 여러 OS 공통이자 오리지널
- Git의 모든 기능을 쓸 수 있음
- 기본 CLI 툴로 익히면, 여러 다른 GUI 툴도 쉽게 적응
- 그 반대는 잘 안됨

# Git 설치

- Goto [www.git-scm.com](http://www.git-scm.com) to download
- git bash 버전



# 기본 설정

```
$ git config --global user.name "홍길동"  
$ git config --global user.email "hong@gil.dong"  
$ git config --global color.ui true
```

각자 이름과 이메일 주소를 적습니다

--global : 지금 로그인한 계정에 전체 설정

--local : 현재 저장소 로컬 설정

# 지역 저장소 초기화

```
$ mkdir hello
$ cd hello
$ git init
Initialized empty Git repository in D:/Git/hello/.git/
$ git status
On branch master

No commits yet

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

- 현재 디렉토리 기준, Git 저장소 초기화
- .gitignore를 이용하여 관리 대상 파일 선택 적용 가능



# 새 파일 추가 (hello.html)

```
<> hello.html ●
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Git 실습</title>
6      </head>
7      <body>
8          <main>
9              <h1>안녕하세요</h1>
10         </main>
11     </body>
12 </html>
13
```

# 현재 상태 확인

작업 디렉토리에 조금 전 hello.html을 추가하고,

```
$ git status
On branch master

No commits yet

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

    hello.html

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

현재 작업영역 상태 다시 확인

# 스테이지 영역에 추가

스테이지 영역에 hello.html 추가하고

```
$ git add hello.html
$ git status
On branch master

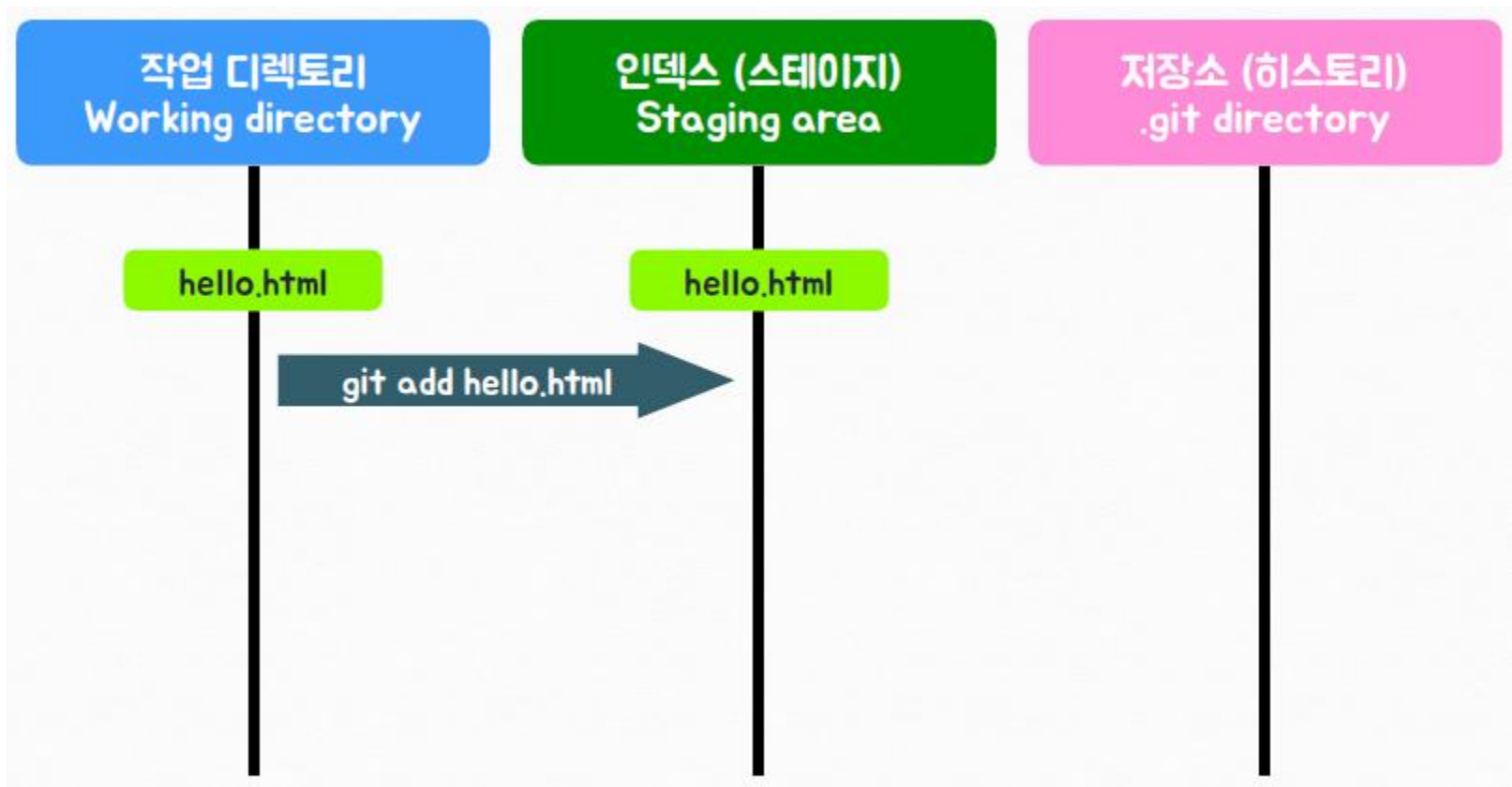
No commits yet

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

    new file:   hello.html
$
```

현재 작업영역 상태 다시 확인

# 스테이지 영역에 올라간 hello.html



# 첫 커밋!

커밋하면서 설명을 남긴다

```
$ git commit -m 'add hello.html'
[master (root-commit) 25120b7] add hello.html
 1 file changed, 12 insertions(+)
 create mode 100644 hello.html
$ git status
On branch master
nothing to commit, working tree clean
$
```

현재 작업영역 상태 다시 확인

# 커밋 로그 확인

```
$ git log
commit 25120b7677322301366245812fb413a83d0ca20c (HEAD -> master)
Author: Inshik Song <inshiksong@gmail.com>
Date:   Sun Mar 10 12:58:09 2019 +0900

    add hello.html
$
```

## 커밋 로그 보기

# 25120b767...?

- 고유 SHA1 값
- 160bits = 20bytes = 40 hexadecimals
- Git 전반에 ID로 쓰임
- 앞 몇 자만 따서 쓰는 편 (e.g. 앞 7자리)

# 안전하게 커밋된 hello.html





# main.css 추가

```
<> hello.html ● # main.css ●  
1  main {  
2      background-color: #ddd;  
3      margin: 1rem;  
4      padding: 1rem;  
5      border-radius: 3px;  
6  }
```

# hello.html에 main.css 추가

```
<> hello.html ● # main.css ●
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Git 실습</title>
6          <link rel="stylesheet" type="text/css" href="main.css">
7      </head>
8      <body>
9          <main>
10         <h1>안녕하세요</h1>
11     </main>
12 </body>
13 </html>
14
```

# 상태 확인 후 main.css 스테이징 영역으로

```
$ git status
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:   hello.html

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

        main.css

no changes added to commit (use "git add" and/or "git commit -a")
$ git add '*.css'
```

확장자 .css인 파일 모두 인덱스

# 다시 상태 확인

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

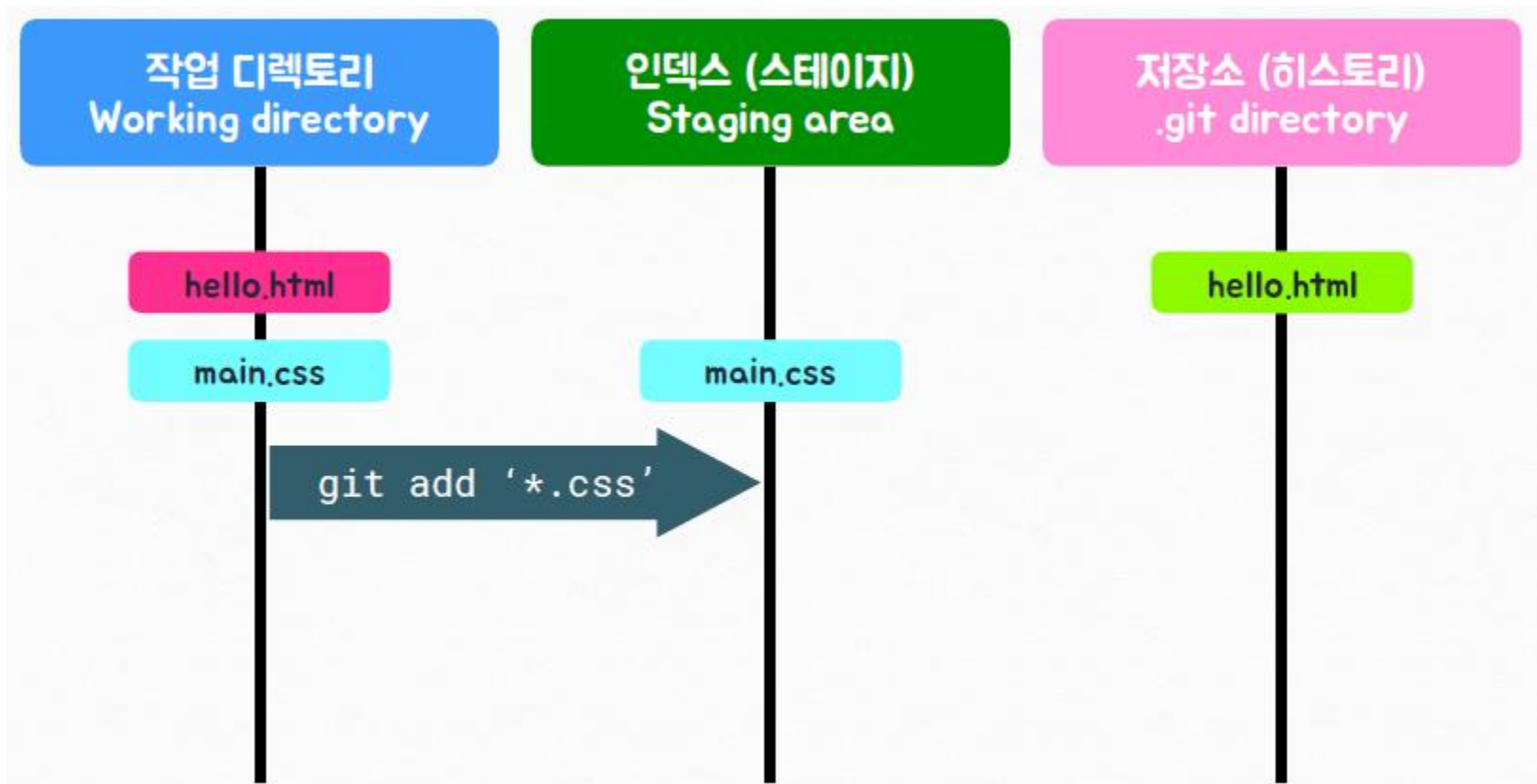
    new file:   main.css

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:   hello.html

$
```

## 실습 3c – main.css를 스테이징 영역으로



# 이 상황에 커밋하면?

- main.css는 저장소에 잘 기록됨
- 그러나 hello.html의 변경 분이 저장소에 반영되지 않음

# hello.html도 인덱스

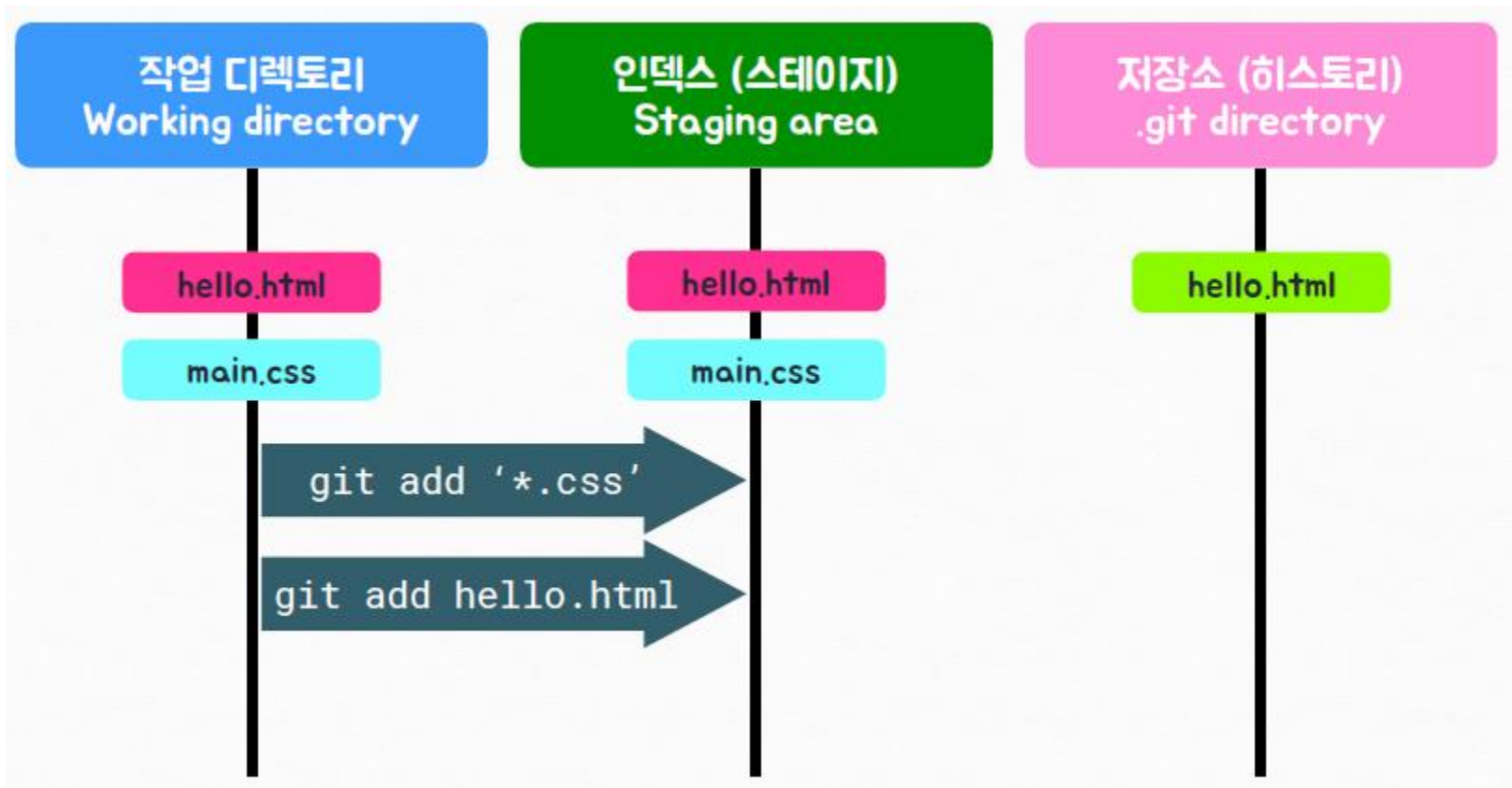
스테이지 영역에 hello.html 또 추가

```
$ git add hello.html
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   hello.html
        new file:   main.css

$
```

# 커밋 준비 완료





# 두번째 커밋

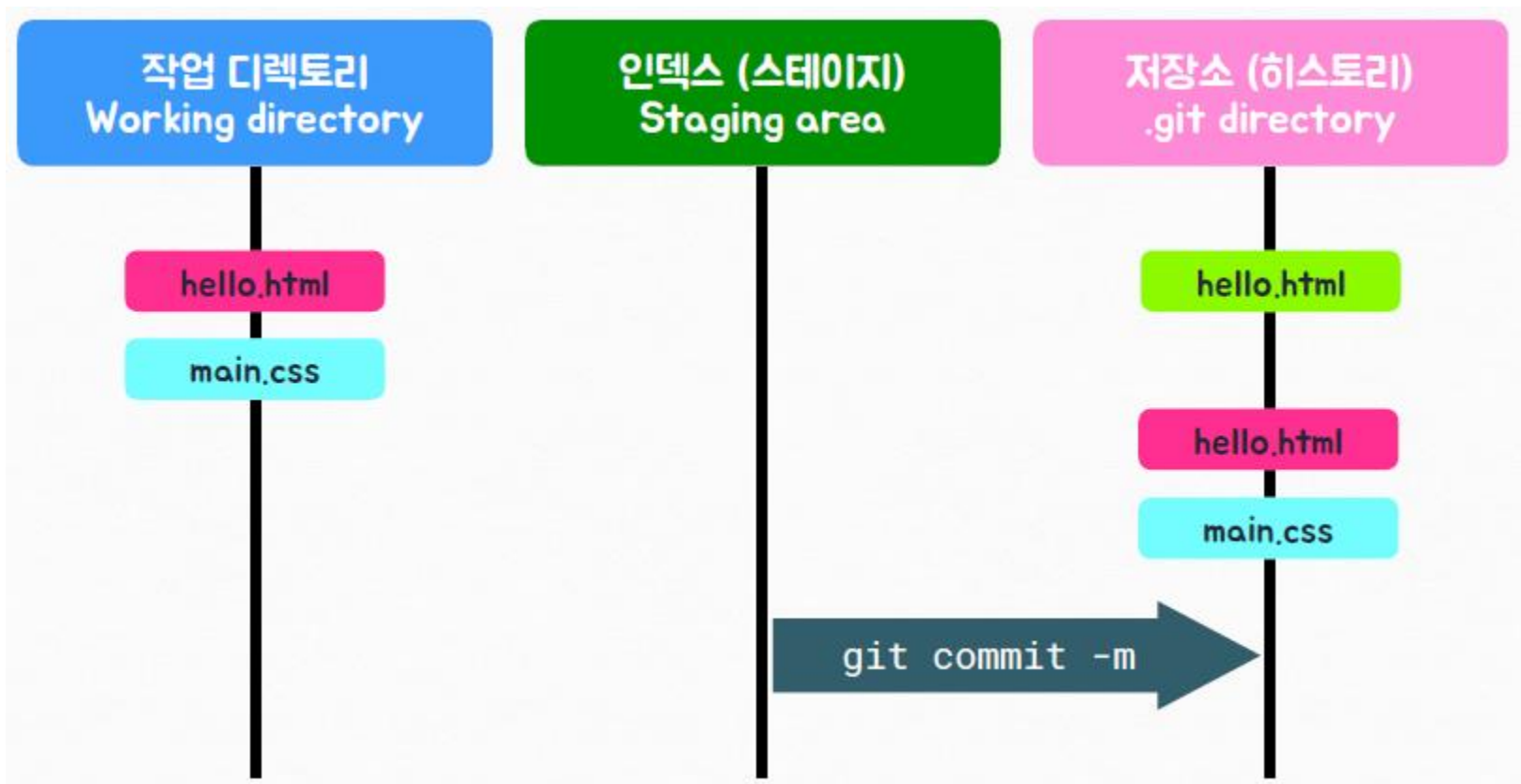
```
$ git commit -m 'add styling'
[master dbb3a05] add styling
 2 files changed, 7 insertions(+)
 create mode 100644 main.css
$ git log
commit dbb3a053efb42f6493c581c034e3c3a0ffb80196 (HEAD -> master)
Author: Inshik Song <inshiksong@gmail.com>
Date:   Sun Mar 10 13:40:28 2019 +0900

    add styling

commit 25120b7677322301366245812fb413a83d0ca20c
Author: Inshik Song <inshiksong@gmail.com>
Date:   Sun Mar 10 12:58:09 2019 +0900

    add hello.html
$
```

# 두번째 커밋



# 연습 #1 – sub.css 추가

```
$ git log
commit 80174a9aa0dbe4ba3c72b90f4561f0073648cfd7 (HEAD -> master)
Author: Inshik Song <inshiksong@gmail.com>
Date:   Sun Mar 10 13:53:57 2019 +0900

    add sub.css

commit dbb3a053efb42f6493c581c034e3c3a0ffb80196
Author: Inshik Song <inshiksong@gmail.com>
Date:   Sun Mar 10 13:40:28 2019 +0900

    add styling

commit 25120b7677322301366245812fb413a83d0ca20c
Author: Inshik Song <inshiksong@gmail.com>
Date:   Sun Mar 10 12:58:09 2019 +0900

    add hello.html

$
```

# 결과

```
<> hello.html  # sub.css  ●  # main.css
1  main h1 {
2      text-decoration: underline;
3  }
```

```
<> hello.html x  # sub.css  # main.css
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Git 실습</title>
6          <link rel="stylesheet" type="text/css" href="main.css">
7          <link rel="stylesheet" type="text/css" href="sub.css">
8      </head>
9      <body>
10         <main>
11             <h1>안녕하세요</h1>
12         </main>
13     </body>
14 </html>
15
```

1. sub.css 파일 추가
2. hello.html에 sub.css 링크 추가
3. sub.css 인덱스에 추가
4. hello.html도 인덱스에 추가
5. 세번째 커밋!
6. 로그 확인

# Outline

- 버전 관리 시스템
- Git
- 저장소와 커밋
- 브랜치
- 병합
- Git 고급 기능

# 브랜치(Branch)

- 작업 흐름 가지, 커밋 그래프, 커밋 줄기
- 각각의 작업 흐름 병행
- 때때로 병합(merge)하거나 버리거나 삭제

# 브랜치의 장점

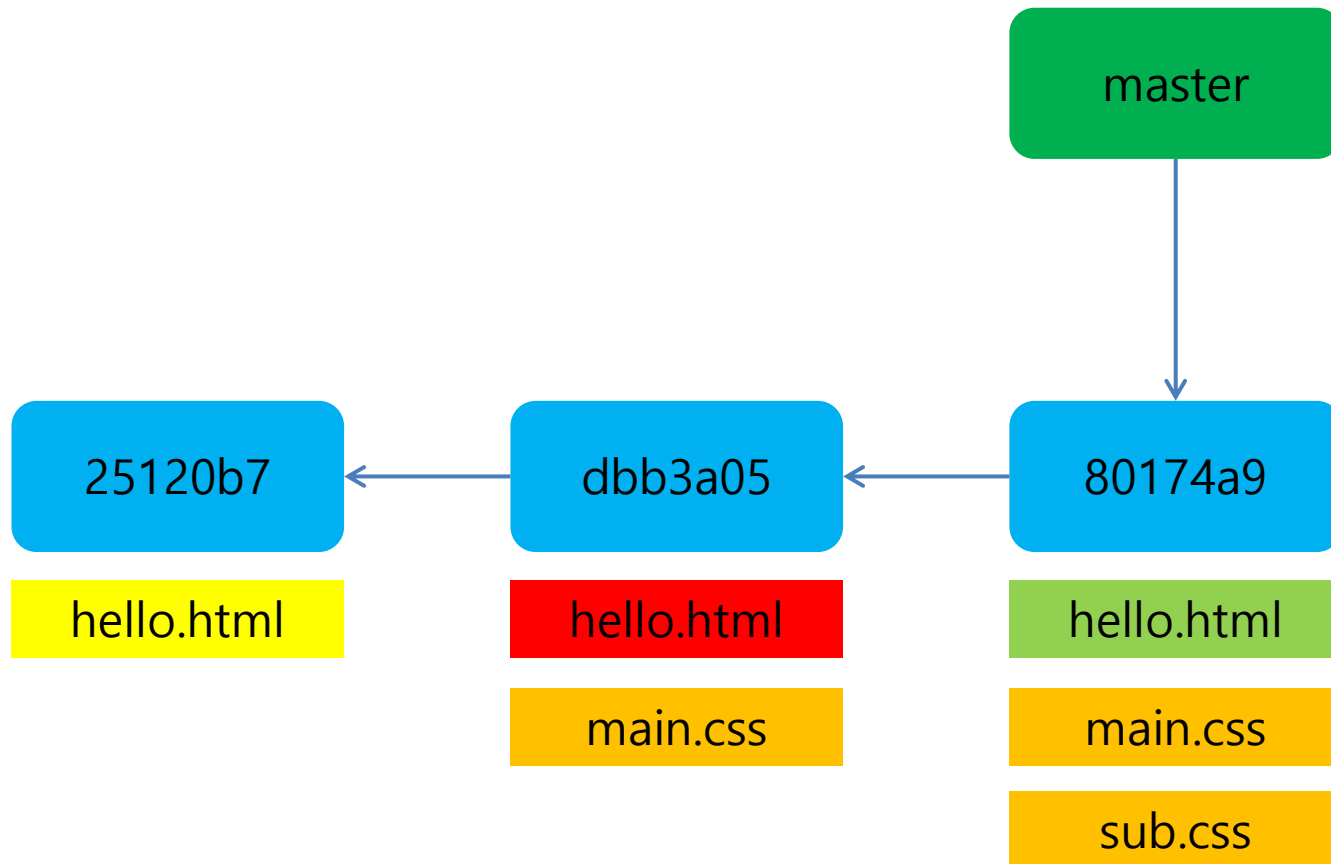
- 상호 독립된 다수의 로컬 브랜치
- 아주 쉽게 문맥 전환
- 역할 구분에 활용하기 좋음(배포/개발/테스트)
- 구현하려는 기능 단위 브랜치도 Good!

# master 브랜치

- 지금까지 작업한 단일 브랜치
- 주(main) 작업 영역



# 지금껏 master 브랜치 상황



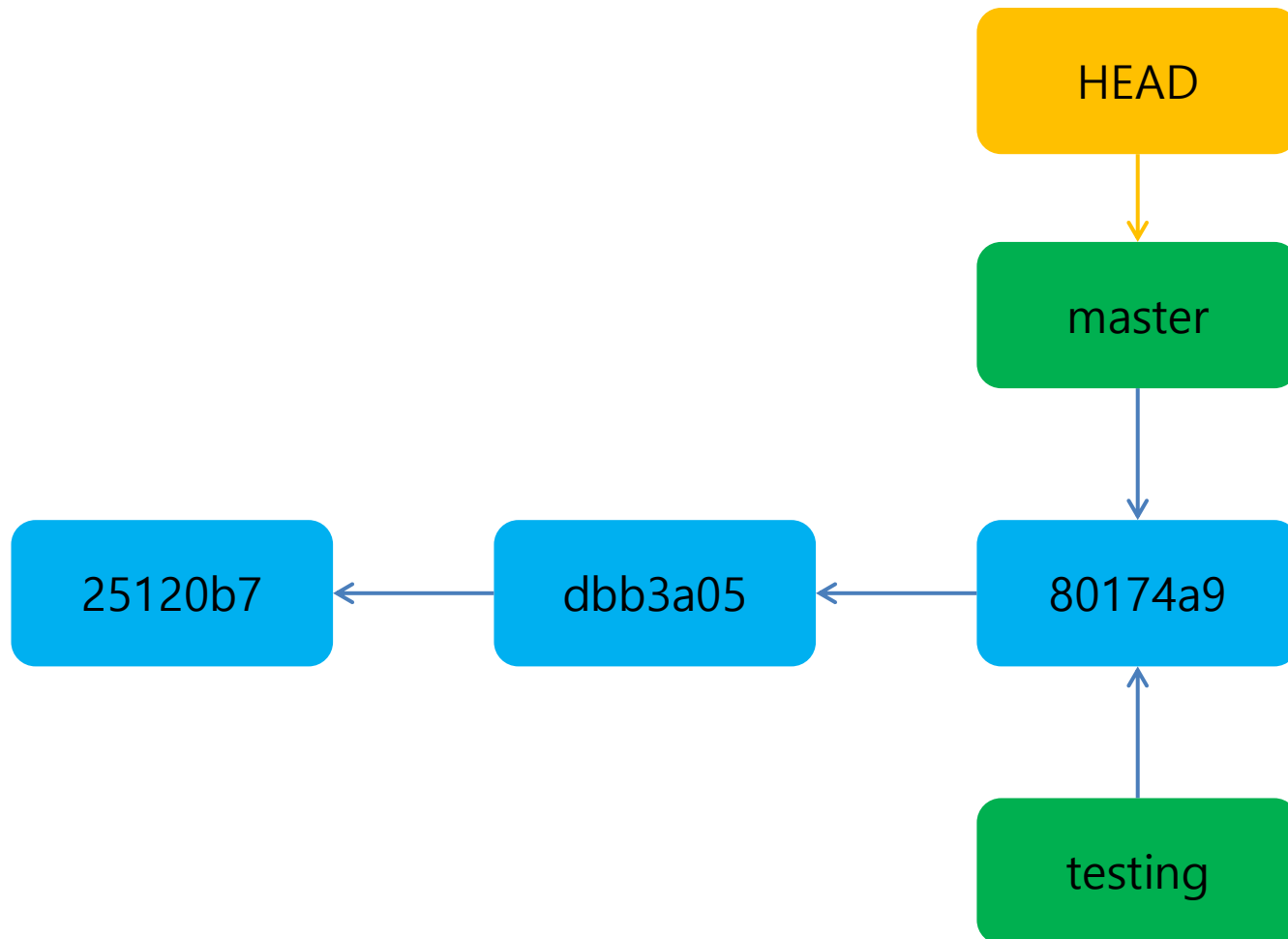
# 현재 브랜치 확인

```
$ git branch
* master
$ git branch -v
* master 80174a9 add sub.css
$
```

# 새 브랜치 만들기

```
$ git branch testing
$ git branch
* master
  testing
$ git branch -v
* master 80174a9 add sub.css
  testing 80174a9 add sub.css
$
```

# master/testing 브랜치 상황



# HEAD

- 각 브랜치는 특정 (최종) 커밋을 가리킴
- HEAD는 현재 브랜치를 가리킴
- 결국, HEAD는 현재 브랜치의 최종 커밋을 가리킴

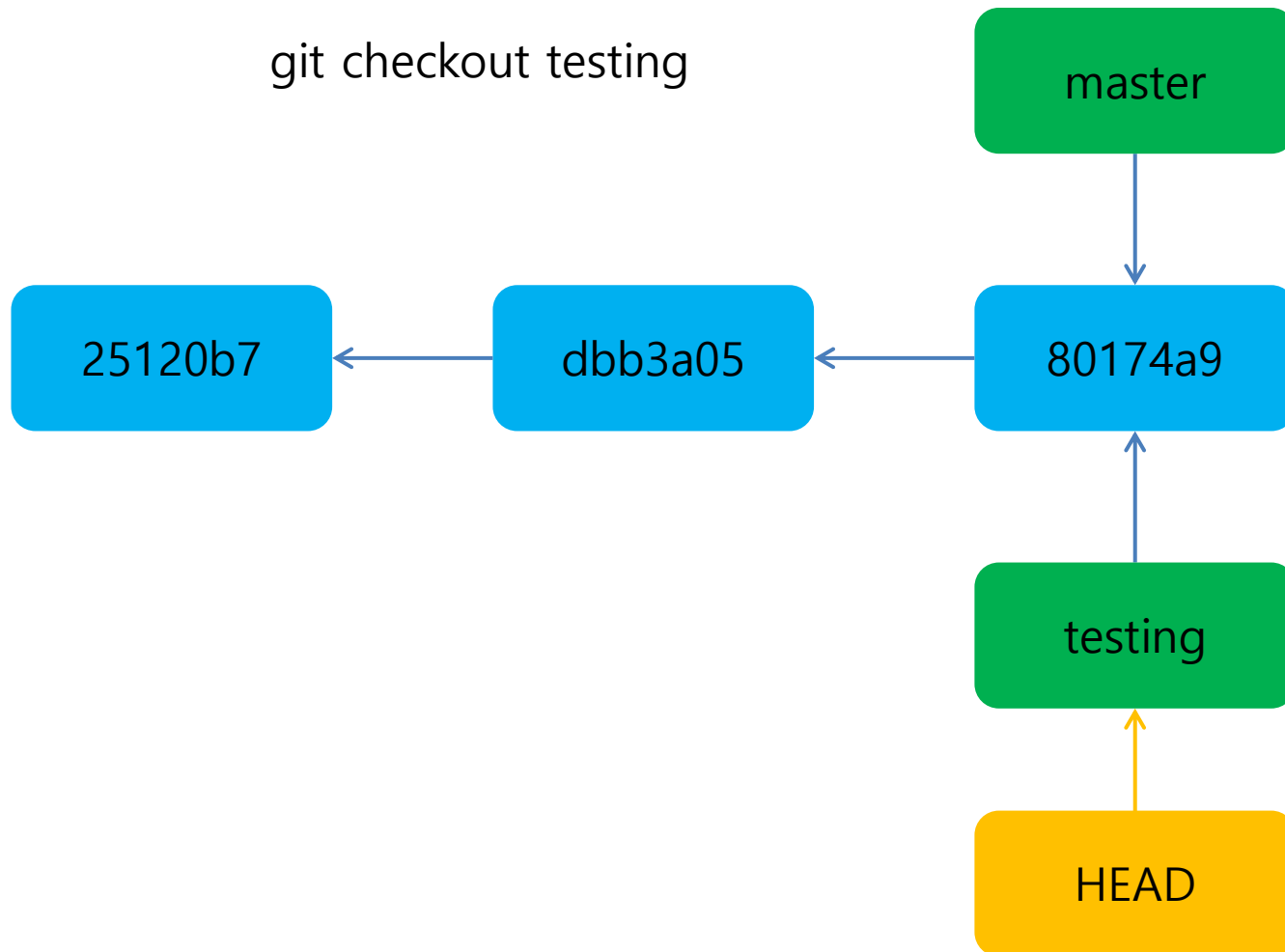
# 새 브랜치로 전환

```
$ git checkout testing
Switched to branch 'testing'

$ git branch -v
  master 80174a9 add sub.css
* testing 80174a9 add sub.css

$
```

# master/testing 브랜치 상황



# testing 브랜치에 추가 작업

```
hello.html ● # main.css # sub.css JS test.js  
1 console.log("test.js 로드")  
2 alert("테스트")
```

```
<> hello.html ● # main.css # sub.css JS test.js ●  
1 <!DOCTYPE html>  
2 <html>  
3   <head>  
4     <meta charset="UTF-8">  
5     <title>Git 실습</title>  
6     <link rel="stylesheet" type="text/css" href="main.css">  
7     <link rel="stylesheet" type="text/css" href="sub.css">  
8   </head>  
9   <body>  
10    <main>  
11    <h1>안녕하세요</h1>  
12    </main>  
13    <script src="test.js"></script>  
14  </body>  
15 </html>  
16 |
```



# testing 브랜치 상황

```
$ git status
On branch testing
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:   hello.html

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

        test.js

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

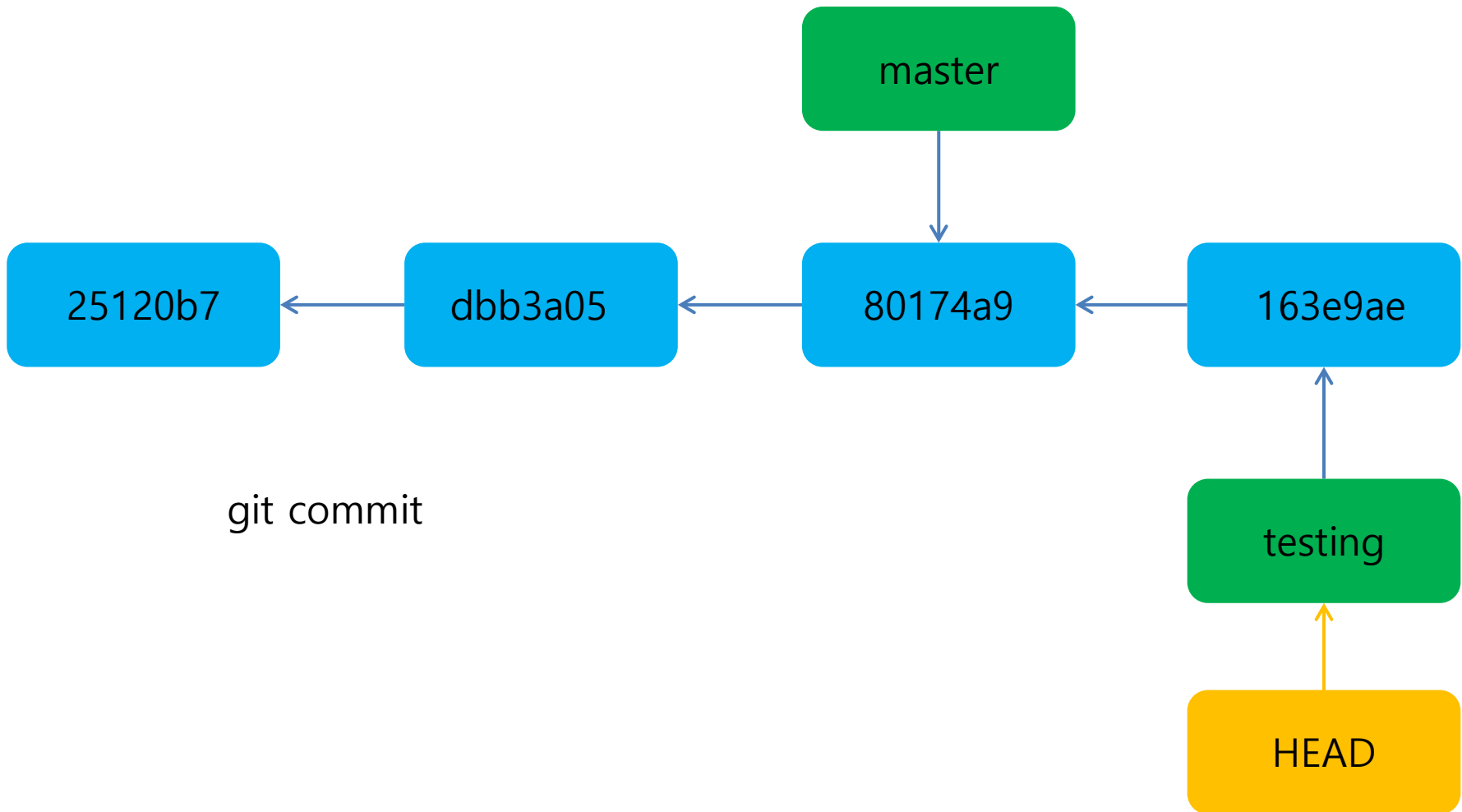
# testing 브랜치에 커밋

```
$ git add test.js

$ git commit -a -m "add test.js"
[testing 163e9ae] add test.js
 2 files changed, 3 insertions(+)
 create mode 100644 test.js

$
```

# master/testing 브랜치 상황



# master 브랜치로 복귀

```
$ git branch -v
  master 80174a9 add sub.css
* testing 163e9ae add test.js
```

```
$ git checkout master
Switched to branch 'master'
```

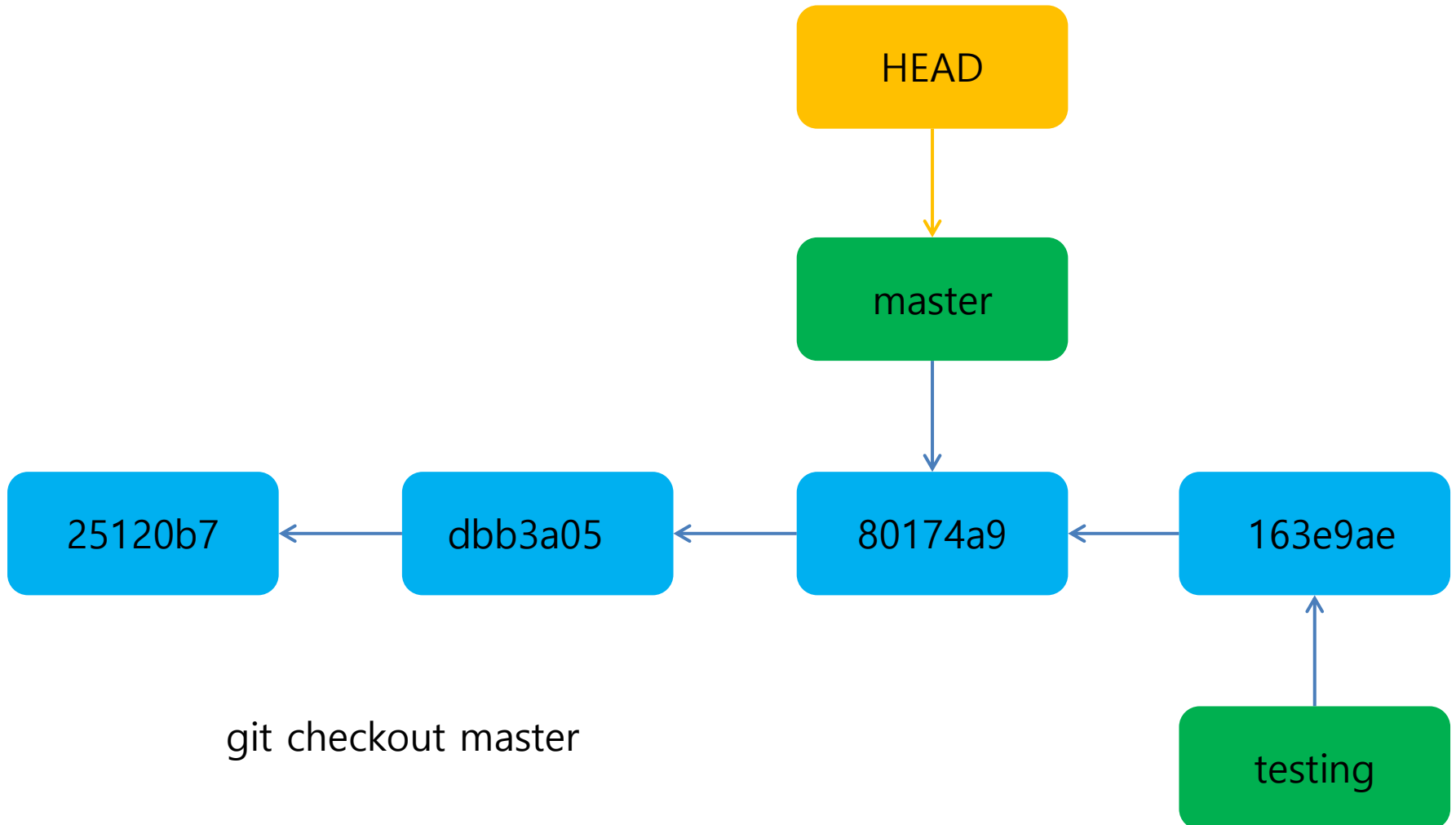
```
$ git branch -v
* master 80174a9 add sub.css
  testing 163e9ae add test.js
```

```
$ ls -l
total 3
-rw-r--r-- 1 Inshik 197609 349 3월 17 13:34 hello.html
-rw-r--r-- 1 Inshik 197609 102 3월 10 13:21 main.css
-rw-r--r-- 1 Inshik 197609 45 3월 10 13:47 sub.css
```

```
$
```

test.js 없음

# master/testing 브랜치 상황



# 브랜치 변경 유의 사항

- 브랜치를 전환하면 작업 디렉토리 내용도 함께 바뀜

# Outline

- 버전 관리 시스템
- Git
- 저장소와 커밋
- 브랜치
- 병합
- Git 고급 기능

# 테스트 성공, 마스터 반영

- testing 브랜치 작업이 성공적
- master에도 변경 내역을 반영합시다
- master와 testing을 합치기(merge)



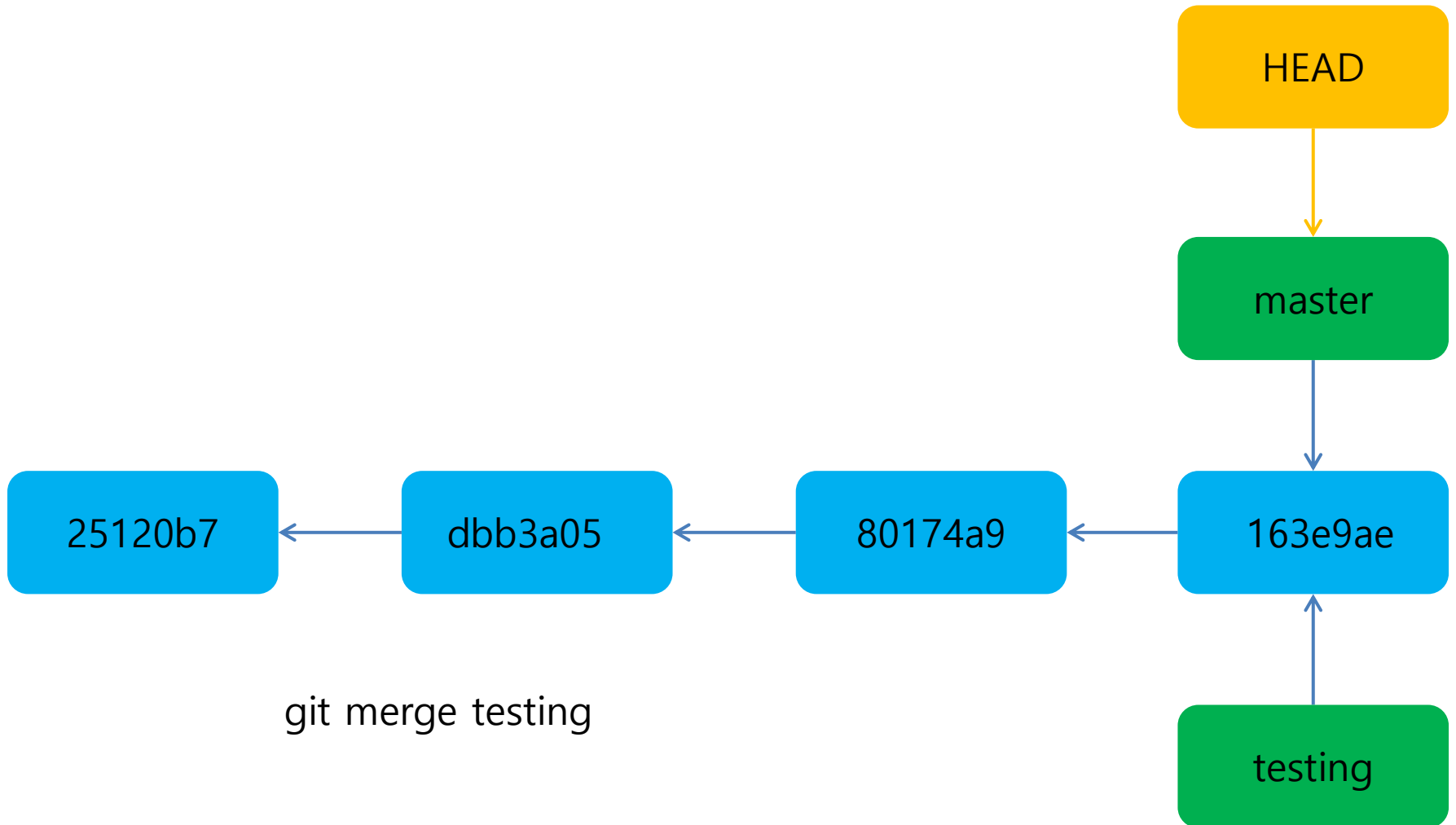
# merge

```
$ git merge testing
Updating 80174a9..163e9ae
Fast-forward
 hello.html | 1 +
 test.js    | 2 ++
2 files changed, 3 insertions(+)
create mode 100644 test.js

$ git branch -v
* master 163e9ae add test.js
  testing 163e9ae add test.js

$
```

# master/testing 브랜치 상황



# 앗차! 실수! 취소!

- 아직 master 브랜치에 testing 브랜치 합치면 안됨
- 뒤늦게 실수임을 인지
- 간단하게 되돌립니다

# reset --hard

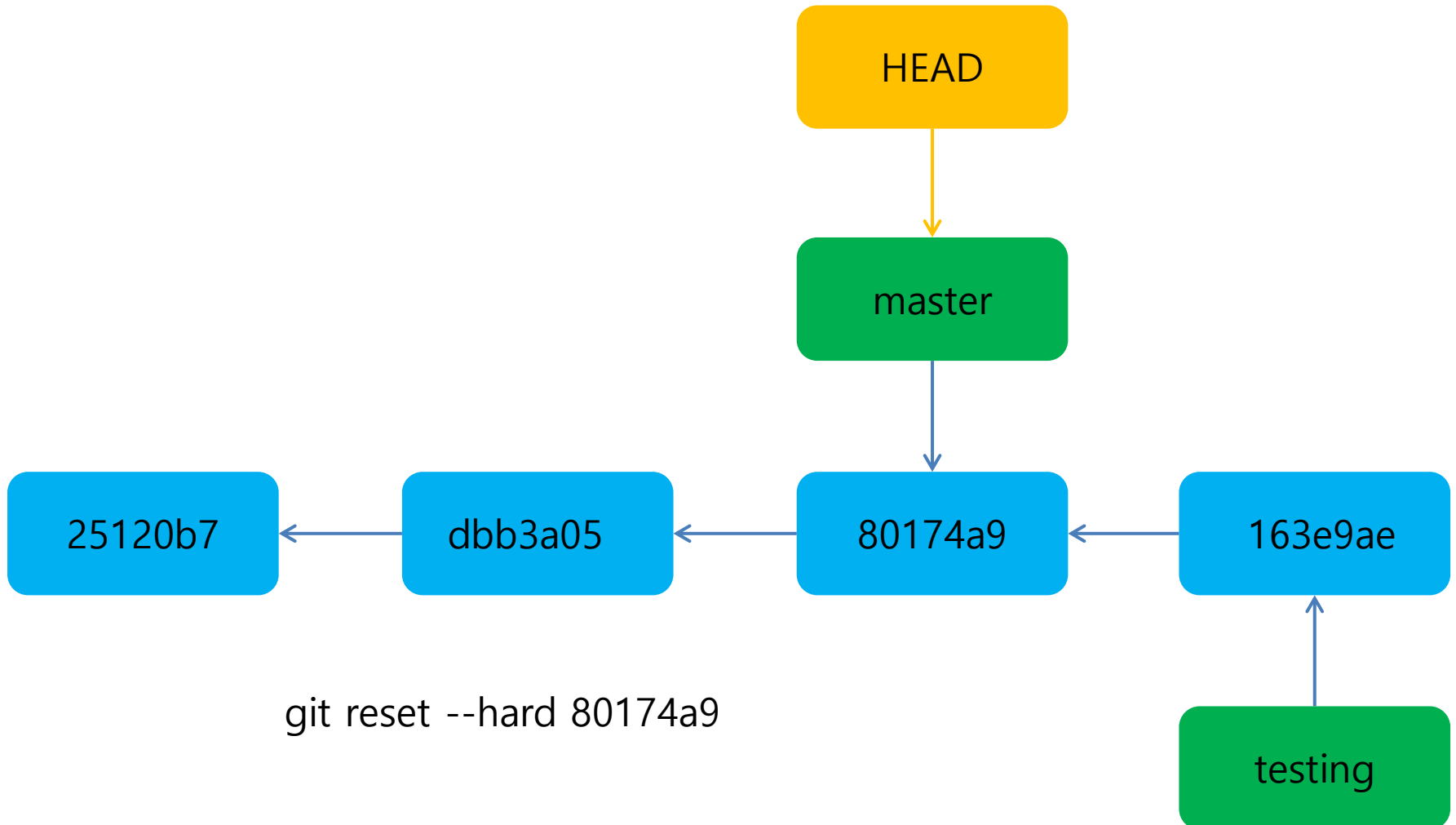
```
$ git branch -v
* master 163e9ae add test.js
  testing 163e9ae add test.js

$ git reset --hard 80174a9
HEAD is now at 80174a9 add sub.css

$
```

git reset --hard 커밋ID

# master/testing 브랜치 상황

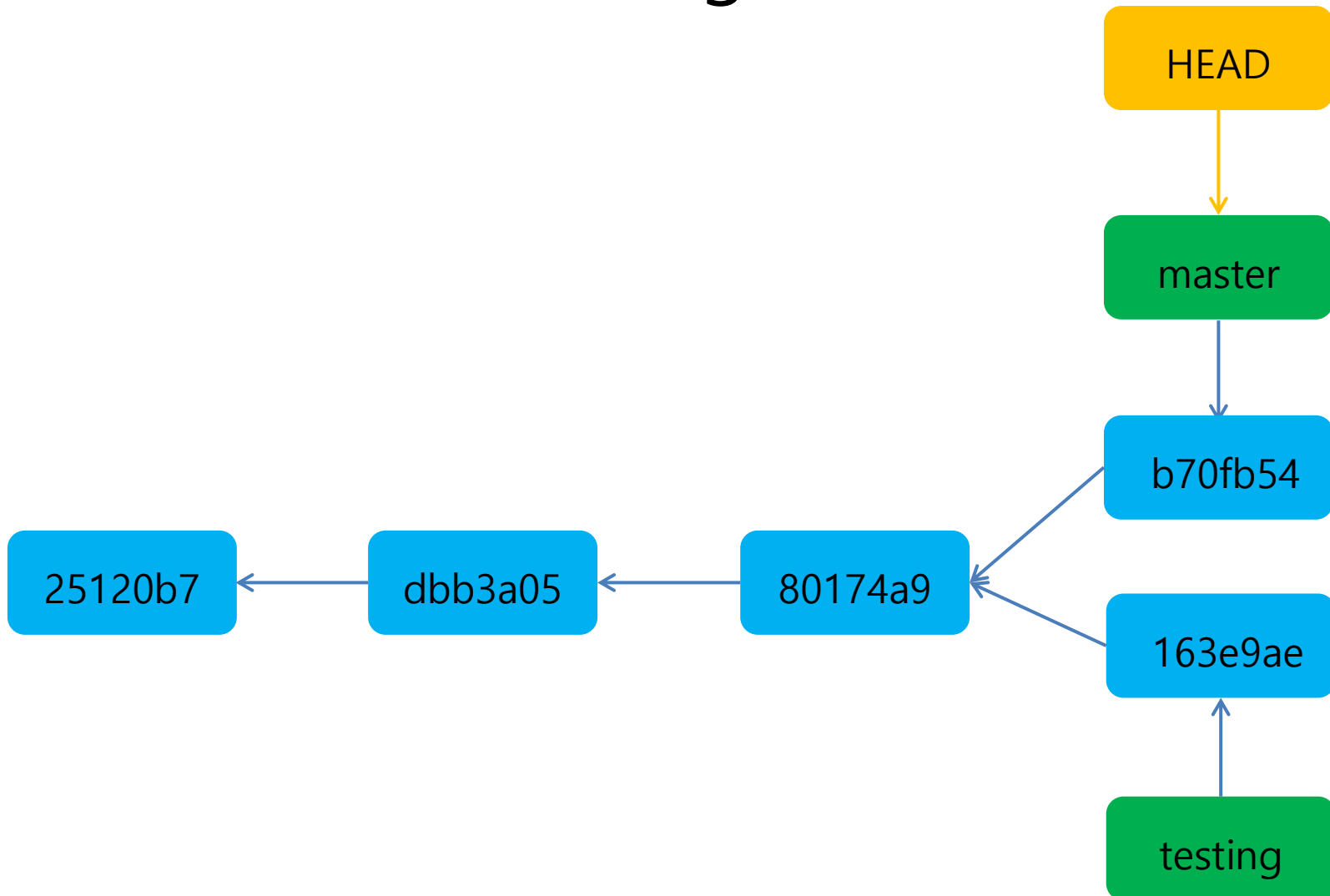


## 연습 #2 – master 브랜치에 커밋 후 merge

```
> hello.html ● # main.css # sub.css JS test.js (deleted from disk)
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <meta charset="UTF-8">
5          <title>Git 실습</title>
6          <link rel="stylesheet" type="text/css" href="main.css">
7          <link rel="stylesheet" type="text/css" href="sub.css">
8      </head>
9      <body>
10         <main>
11             <h1>안녕하세요</h1>
12             <div>대학생을 위한 Git 실습</div>
13         </main>
14     </body>
15 </html>
16
```

1. hello.html에 아래 한 줄 추가 git add
2. 새로 커밋 추가 git commit -m " "
3. testing브랜치 합치기 git merge testing
4. 종이에 현재 상황 그림 그리기

# master/testing 브랜치 상황



# 나뉜 브랜치를 합치기

- master 브랜치와 testing 브랜치가 나뉘어 각자의 길을 간 상황
- Git이 합치려는 브랜치들의 공통 조상을 찾아서 처리



# merge

```
$ git branch -v
* master b70fb54 added div
testing 163e9ae add test.js
```

```
$ git merge testing
```

Merge commit 메시지를 요구

```
Auto-merging hello.html
```

```
Merge made by the 'recursive' strategy.
```

```
hello.html | 1 +
```

```
test.js    | 2 ++
```

```
2 files changed, 3 insertions(+)
```

```
create mode 100644 test.js
```

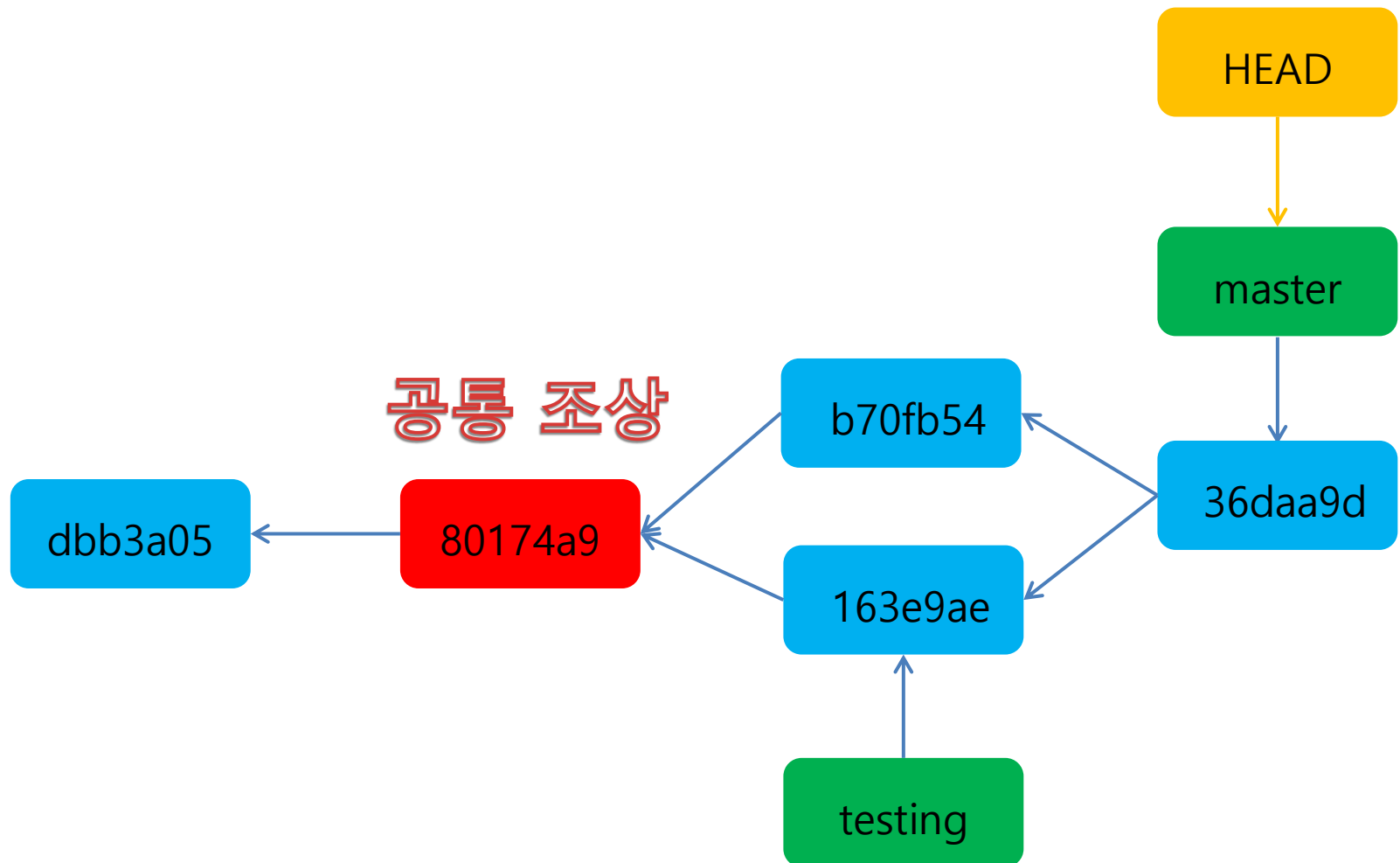
```
$ git branch -v
```

```
* master 36daa9d Merge branch 'testing'
```

```
testing 163e9ae add test.js
```

```
$
```

# master/testing 브랜치 상황



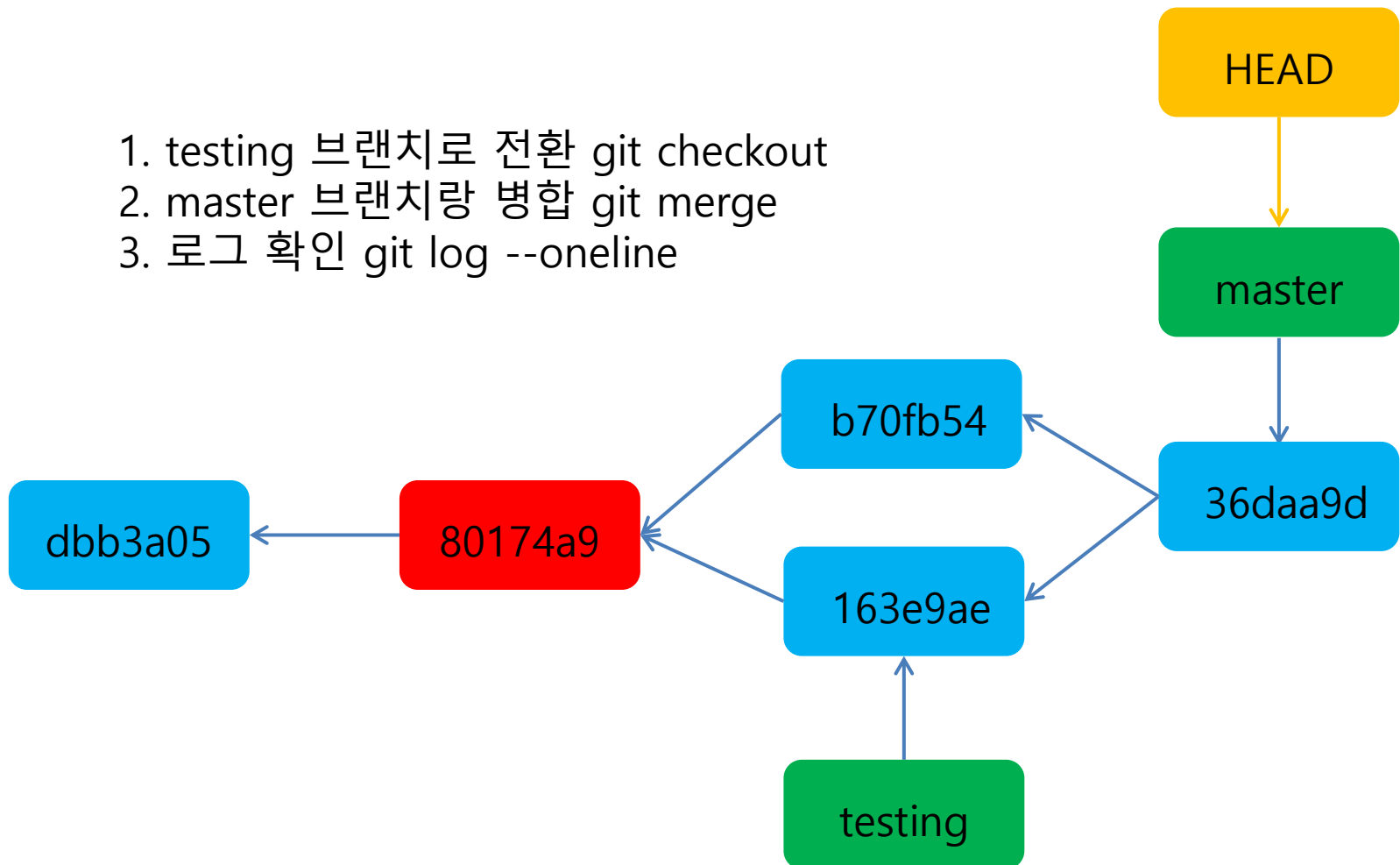
# 현 상황 로그 간단히 보기

```
$ git log --oneline
36daa9d (HEAD -> master) Merge branch 'testing'
b70fb54 added div
163e9ae (testing) add test.js
80174a9 add sub.css
dbb3a05 add styling
25120b7 add hello.html

$
```

# 연습 #3 – testing 브랜치도 master로 병합

1. testing 브랜치로 전환 git checkout
2. master 브랜치랑 병합 git merge
3. 로그 확인 git log --oneline



# Outline

- 버전 관리 시스템
- Git
- 저장소와 커밋
- 브랜치
- 병합
- Git 고급 기능

# git tag : 특정 커밋을 참조하는 이름 붙이기

- `git tag tag_name`: 가장 최근 커밋에 이름 붙이기
- `git tag tag_name commit_checksum`: 특정 커밋에 이름 붙이기
- `git tag -a tag_name` : 간단한 메시지를 입력할 수 있도록 편집기가 열림
- `git show tag` : 누가 언제 어떤 메시지를 입력했는지 확인 가능

# git commit --amend : 마지막 커밋 수정하기

- 최종 커밋과 커밋하지 않은 상태에 있는 변경 내역이 합쳐진 새 커밋을 생성
- 실제로는 마지막 커밋을 수정하는 것이 아니라 마지막 커밋을 대체하는 새로운 커밋을 생성

# git revert : 공개된 커밋의 변경 내역을 되돌리기

- 이미 공개된 커밋을 임의로 수정하는 것은 매우 위험!
- 안전하게 변경 내역을 되돌리는 방법
- 커밋으로 발생한 변경 내역의 반대 커밋을 수행
- 추가한 코드는 빼고, 지운 코드는 다시 추가



# git reset : 이전 작업 결과를 저장한 상태로 되돌리기

- git revert 명령은 이전 커밋을 남겨 두지만 git reset 명령은 이전 커밋을 남기지 않고 새로운 커밋을 남김
- HEAD의 위치, 인덱스, 작업 저장소 디렉토리 등도 함께 되돌리지 여부에 따라 다양한 모드 지정 가능 (매뉴얼 참조)

# git checkout HEAD -- filename : 특정 파일을 최종 커밋 시점으로 되돌리기

- 예를 들어 최종 커밋 후 README.md 파일을 변경 후 git add README.md 명령을 수행한 후 git status로 확인하면 커밋할 내용이 있다고 알려 줌
- 이 때 git checkout HEAD -- README.md를 실행하면 README.md 파일의 내용이 복원되고 git status로 확인하면 커밋할 내용이 없다고 나옴

# git rebase : 브랜치 이력을 확인하면서 병합하기

- 여러 명의 작업자가 여러 기능을 동시에 개발한다면 여러 개의 브랜치가 생기고 이를 git merge로 병합한다면 병합 이후 그래프가 복잡해지게 됨
- 이 경우 git rebase 명령을 이용하면 충돌을 해결하거나, 강제 병합, 또는 rebase 취소 옵션을 통해 각각의 브랜치의 변경 내역을 master 브랜치에 순차적으로 작용하여 그래프의 모양을 단순하게 변경할 수 있음
- 자세한 내용은 매뉴얼 참조

# Git GUI

- Git은 좋은 GUI를 지원하는 IDE (Eclipse, Visual Studio, IntelliJ, Xcode 등)과 결합하여 사용할 수도 있고 Git의 주요 명령을 지원하는 GUI 도구를 이용하여 사용할 수 있음 (<https://git-scm.com/downloads> 참고)
- 대표적인 Git GUI 도구는 SourceTree (<https://www.sourcetreeapp.com> 참고)

# Questions?