

Git 사용 방법 - 중급

1. git reset

1) 먼저 실습을 위해 현재 작업 폴더의 모든 파일을 삭제하고, 다시 Git 을 초기화한다.

```
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$
```

2) Working Directory, Staging Area, Repository 에 올린 내용은 어떻게 취소할 수 있는가?

3) **git reset** 설명

4) Option : 어디까지 되돌릴 수 있는가?

- **--hard HEAD[^]** : 수정한 것까지 통째로 되돌리자
- **--mixed HEAD[^]** : **add** 한 것까지(default)
- **--soft HEAD[^]** : **commit** 한 것만

5) **HEAD** : 현재 작업 중인 branch/commit 중 가장 최근 commit 을 가리킨다.

6) **HEAD[^]** : 가장 최근 버전에서(HEAD)에서 ^(하나 되돌리자)

- **git reset --hard HEAD[^]** : 가장 최근 **commit** 으로부터 **한 개 전**으로 되돌리자.
- **git reset --hard HEAD^{^^}** : 가장 최근 **commit** 으로부터 **두 개 전**으로 되돌리자.
- **git reset --hard HEAD^{^^^}** : 가장 최근 **commit** 으로부터 **세 개 전**으로 되돌리자.

7) 실습을 위해 test.txt 파일을 만들고 "This is first commit"의 문자열을 저장한다.

```
henry-MacBook-Pro:GitWork henry$ touch test.txt
henry-MacBook-Pro:GitWork henry$ echo "This is first commit" > test.txt
henry-MacBook-Pro:GitWork henry$ cat test.txt
This is first commit
henry-MacBook-Pro:GitWork henry$
```

8) 지금의 Git 상태를 확인한다.

```
[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)
    test.txt

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

9) 현재 Working Directory에 있는 test.txt 를 Staging Area에 올린다.

```
[henry-MacBook-Pro:GitWork henry$ git add test.txt
[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:   test.txt

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

10) Repository에 Commit 을 하고 Log 를 확인해 본다.

```
[henry-MacBook-Pro:GitWork henry$ git commit -m "First Commit"
[main (root-commit) bead1c9] First Commit
  1 file changed, 1 insertion(+)
  create mode 100644 test.txt
[henry-MacBook-Pro:GitWork henry$ git log
commit bead1c94de09bfd574d3c40e6562205b10002c04 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:10:29 2023 +0900

  First Commit
henry-MacBook-Pro:GitWork henry$ ]
```

11) 상태도 확인해 본다. 성공적으로 Commit 됐음을 확인할 수 있다.

```
[henry-MacBook-Pro:GitWork henry$ git status
On branch main
nothing to commit, working tree clean
henry-MacBook-Pro:GitWork henry$ ]
```

12) test.txt 파일의 내용을 수정하고 **add** 한다.

```
GitWork -- bash -- 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Repository" > test.txt
henry-MacBook-Pro:GitWork henry$ git add test.txt
henry-MacBook-Pro:GitWork henry$
```

13) 다음과 같이 **commit** 도 수행하고 **log** 로 확인한다.

```
GitWork -- bash -- 80x24
henry-MacBook-Pro:GitWork henry$ git commit -m "Repository Commit"
[main d7d69af] Repository Commit
 1 file changed, 1 insertion(+), 1 deletion(-)
henry-MacBook-Pro:GitWork henry$ git log
commit d7d69afb4f60cac2762c5c051924b0bf7cfb1c0b (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:12:15 2023 +0900

    Repository Commit

commit bead1c94de09bfd574d3c40e6562205b10002c04
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:10:29 2023 +0900

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

14) 다시 test.txt 를 다음과 같이 수정하고 **add** 했다.

```
GitWork -- bash -- 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Staging Area" > test.txt
henry-MacBook-Pro:GitWork henry$ git add test.txt
henry-MacBook-Pro:GitWork henry$
```

15) 이제까지 한 것을 정리하면, I'm here Staging Area 문자열을 갖고 있는 test.txt 는 Staging Area에 있고, I'm here Repository 문자열이 저장되어 있는 test.txt 는 Repository에 있다.

16) 마지막으로 다음과 같이 This is fourth commit 문자열을 test.txt 에 수정했다.

```
GitWork -- bash -- 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Working Directory" > test.txt
henry-MacBook-Pro:GitWork henry$
```

17) 지금까지의 내용을 확인해보자.

```
[henry-MacBook-Pro:GitWork henry$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   test.txt

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:   test.txt

[henry-MacBook-Pro:GitWork henry$ git log
commit d7d69afb4f60cac2762c5c051924b0bf7cfb1c0b (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:12:15 2023 +0900

  Repository Commit

commit bead1c94de09bfd574d3c40e6562205b10002c04
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:10:29 2023 +0900

  First Commit
henry-MacBook-Pro:GitWork henry$ ]
```

18) 먼저 Status 를 보면, test.txt 파일이 수정되었고, 아직 Staging Area에 올리지 않은 내용의 파일이 있으며, Log 를 보면, 2 번의 Commit 이 있었음을 알 수 있다.

19) 다시 말하면, I'm here Respository 의 문자열을 갖고 있는 test.txt 는 이미 Commit 되어

Repository에 위치하고 있고, I'm here Staging Area 문자열을 갖고 있는 test.txt 는 Staging Area에 있으며, I'm here Working Directory 문자열을 갖고 있는 test.txt 는 Working Area에 위치하고 있다.

20) 현재의 HEAD 는 Repository Commit 을 한 곳에 맞춰져 있다.

21) 다음의 명령을 수행해 보자.

\$ git reset --hard HEAD^

```
[henry-MacBook-Pro:GitWork henry$ git reset --hard HEAD^
HEAD is now at bead1c9 First Commit
[henry-MacBook-Pro:GitWork henry$ git status
On branch main
nothing to commit, working tree clean
henry-MacBook-Pro:GitWork henry$ ]
```

22) 현재의 **HEAD** 가 First Commit 을 가리키고 있다. 그리고 test.txt 파일의 내용을 확인해 보자.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ cat test.txt
This is first commit
henry-MacBook-Pro:GitWork henry$
```

23) 즉 모든 Commit 이 취소되고, Add 한 것도 취소된 것을 알 수 있다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ git status
On branch main
nothing to commit, working tree clean
henry-MacBook-Pro:GitWork henry$ git log
commit bead1c94de09bfd574d3c40e6562205b10002c04 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:10:29 2023 +0900

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

24) 현재 **HEAD** 가 Repository commit 한 것을 가리키고 있었기 때문에 Repository Commit 을 취소한 것을 알 수 있다.

25) 연쇄적으로 Staging Area에 **add** 한 것도 취소하고, 마지막으로 test.txt 를 수정하고 아직 **add** 하지 않은 내용도 취소한 것을 알 수 있다.

26) 시간을 되돌려서 **reset** 되기 전으로 이동해 보자.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Repository" > test.txt
henry-MacBook-Pro:GitWork henry$ git add test.txt
henry-MacBook-Pro:GitWork henry$ git commit -m "Repository Commit Again"
[main 520848f] Repository Commit Again
 1 file changed, 1 insertion(+), 1 deletion(-)
henry-MacBook-Pro:GitWork henry$
```



```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Staging Area" > test.txt
henry-MacBook-Pro:GitWork henry$ git add test.txt
henry-MacBook-Pro:GitWork henry$
```



```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Working Directory" > test.txt
henry-MacBook-Pro:GitWork henry$
```

27) 현재 상태를 확인해 보자.

```
henry-MacBook-Pro:GitWork henry$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   test.txt

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:   test.txt

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

28) 아직 Staging Area에 **add** 하지 않은 파일(I'm here Working Directory)이 있고, Staging Area에 있는 파일(I'm here Staging Area)도 있다.

29) Mixed mode로 **reset** 해보자.

\$ **git reset --mixed HEAD^**

```
henry-MacBook-Pro:GitWork henry$ git reset --mixed HEAD^
Unstaged changes after reset:
 M      test.txt
henry-MacBook-Pro:GitWork henry$
```

30) Log를 확인해보자.

```
henry-MacBook-Pro:GitWork henry$ git log
commit bead1c94de09bfd574d3c40e6562205b10002c04 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:10:29 2023 +0900

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

31) 당연히 첫번째 Commit만 남겨져 있고, 두번째 Commit은 사라졌다. 상태를 확인해보자.

```
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:   test.txt

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

32) Staging Area에 있던 내용도 사라졌음을 확인할 수 있다. 그럼, 파일의 내용을 확인해보자.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ cat test.txt
I'm here Working Directory
henry-MacBook-Pro:GitWork henry$
```

33) 아직 마지막으로 수정한 파일의 내용은 변경되지 않았음을 알 수 있다. 즉, hard mode로 reset하는 것과 mixed mode로 reset하는 것의 차이가 여기에 있다. [Mixed mode로 reset하는 것은 commit과 add는 취소하고 Working Area에 있는 파일은 변경하지 않은 것이다.](#)

34) 마지막으로 Soft mode로 **reset**을 연습하기 위해 다시 **reset**되기 전으로 되돌린다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Repository" > test.txt
henry-MacBook-Pro:GitWork henry$ git add test.txt
henry-MacBook-Pro:GitWork henry$ git commit -m "Repository Commit Again Again"
[main ee6f973] Repository Commit Again Again
 1 file changed, 1 insertion(+), 1 deletion(-)
henry-MacBook-Pro:GitWork henry$
```



```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Staging Area" > test.txt
henry-MacBook-Pro:GitWork henry$ git add test.txt
henry-MacBook-Pro:GitWork henry$
```



```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ echo "I'm here Working Directory" > test.txt
henry-MacBook-Pro:GitWork henry$
```

35) Soft reset을 해보자.

\$ **git reset --soft HEAD^**

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ git reset --soft HEAD^
henry-MacBook-Pro:GitWork henry$
```

36) Log를 보면 **commit**이 취소된 것을 알 수 있다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ git log
commit bead1c94de09bfd574d3c40e6562205b10002c04 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 11:10:29 2023 +0900

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

37) 상태를 확인해보면, Staging Area 는 취소되지 않았다.

```
[henry-MacBook-Pro:GitWork henry$ git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   test.txt

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:   test.txt

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

38) 파일의 내용도 확인해보면 변경되지 않았다.

```
[henry-MacBook-Pro:GitWork henry$ cat test.txt
I'm here Working Directory
henry-MacBook-Pro:GitWork henry$ ]
```

39) 즉 **Soft reset** 은 **commit** 만 취소됨을 알 수 있다.

2. git diff

- 1) 다른 Commit 과 Working Directory 를 비교하는 명령어
- 2) 현재 Working Directory 와 Staging Area 사이의 차이를 확인할 수 있다.
- 3) **git diff**
 - 현재 Branch 의 마지막 Commit 과의 차이점 비교
- 4) **git diff [Commit ID]**
 - 특정 Commit 과의 차이점 비교
- 5) **git diff [Commit ID] -- [file path]**
 - 특정 Commit 과 특정 File 과의 차이점 비교
- 6) 일단 현재 index.html 를 생성하고 간단한 Message 와 함께 Commit 할 것이다. 먼저 GitWork 폴더를 Git 으로 초기화하고, README.md 를 Commit 한 다음, index.html 파일을 생성하고 Commit 한다.

```
GitWork — -bash — 80x24
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$ 
```



```
GitWork — -bash — 80x24
henry-MacBook-Pro:GitWork henry$ touch README.md
henry-MacBook-Pro:GitWork henry$ git add README.md
henry-MacBook-Pro:GitWork henry$ git commit -m "Add README.md File"
[main (root-commit) f8a793e] Add README.md File
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 README.md
henry-MacBook-Pro:GitWork henry$ 
```



```
GitWork — -bash — 80x24
henry-MacBook-Pro:GitWork henry$ touch index.html
henry-MacBook-Pro:GitWork henry$ git add index.html
henry-MacBook-Pro:GitWork henry$ git commit -m "Add index.html File"
[main c15f53b] Add index.html File
 1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 index.html
henry-MacBook-Pro:GitWork henry$ 
```

- 7) 현재 Git 저장소는 index.html 과 README.md 2 개의 File 을 간단한 Commit Message 와 함께 Commit 했다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ git log
commit c15f53b250c9a44e054dc6edff01040d85d7be19 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:21:15 2023 +0900

    Add index.html File

commit f8a793ec34832366386535cb19abc8a767883d5c
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:20:42 2023 +0900

    Add README.md File
henry-MacBook-Pro:GitWork henry$
```

- 8) 추가로 index.html File 에 아래와 같이 Source 를 변경하고 다시 Commit 했다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ echo "<html></html>" > index.html
henry-MacBook-Pro:GitWork henry$ git commit -am "Add html Tag"
[main f7b088f] Add html Tag
 1 file changed, 1 insertion(+)
henry-MacBook-Pro:GitWork henry$
```

- 9) 변경 Log 는 아래와 같다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ git log index.html
commit f7b088f7887a7965e980b57580aacb9f78f41736 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:22:34 2023 +0900

    Add html Tag

commit c15f53b250c9a44e054dc6edff01040d85d7be19
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:21:15 2023 +0900

    Add index.html File
henry-MacBook-Pro:GitWork henry$
```

10) Commit ID 를 알기 위해 **oneline** option 을 사용해 보자.

```
[henry-MacBook-Pro:GitWork henry$ git log --oneline  
f7b088f (HEAD -> main) Add html Tag  
c15f53b Add index.html File  
f8a793e Add README.md File  
henry-MacBook-Pro:GitWork henry$ ]
```

11) 원래의 Commit ID 는 40 자가 넘는데, **--oneline option** 을 사용하면 앞자리 7 자리만 보여준다.

12) 7 자리로도 충분히 구별이 되기 때문에 **git diff <Commit ID>**를 넣을 때에는 7 자리만 사용하기로 한다.

13) Index.html 을 처음 Commit 할 때의 ID 를 이용해서 **git diff** 를 해보자.

```
[henry-MacBook-Pro:GitWork henry$ git diff c15f53b  
diff --git a/index.html b/index.html  
index e69de29..18ecdc 100644  
--- a/index.html  
+++ b/index.html  
@@ -0,0 +1 @@  
+<html></html>  
henry-MacBook-Pro:GitWork henry$ ]
```

14) 위의 결과를 보면 index.html 은 초록색의 글자를 통해, **<html></html>** 부분이 추가된 것을 알 수 있다.

15) '+' 기호가 붙은 부분은 추가된 줄을 의미한다.

16) 반면 제거된 줄이 있다면 '-' 기호가 붙는다.

17) 이번에는 이미 Commit 됐던 README.md file 을 수정해보자.

18) 그리고 아래와 같이 어떤 option 없이 **git diff** 를 실행해 보자.

```
[henry-MacBook-Pro:GitWork henry$ echo "<html><head></head></html>" > README.md  
[henry-MacBook-Pro:GitWork henry$ git diff  
diff --git a/README.md b/README.md  
index e69de29..ec5ca16 100644  
--- a/README.md  
+++ b/README.md  
@@ -0,0 +1 @@  
+<html><head></head></html>  
henry-MacBook-Pro:GitWork henry$ ]
```

- 19) Git diff 를 실행하면 현재 Working Directory 와 Staging Area 영역 사이의 차이를 확인할 수 있다.
- 20) 현재의 상태를 보면 README.md 는 File 의 내용이 수정됐기 때문에 Working Directory 에 있고 Staging Area 에 올라가 있지 않은 것을 알 수 있다.
- 21) 역시 '+'기호는 추가를, '-'기호는 제거를 의미한다.

22) 현재의 상태는 아래와 같다.

```
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:   README.md

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

- 23) 이제 README.md 를 Staging Area 에 추가해 보자.

```
henry-MacBook-Pro:GitWork henry$ git add README.md
henry-MacBook-Pro:GitWork henry$ git diff
henry-MacBook-Pro:GitWork henry$
```

- 24) 현재 상태에서 **git diff** 명령어를 사용하면 Working Directory 와 Staging Area 사이의 차이가 없으므로 아무것도 표시되지 않는다.

- 25) 대신, 최신 Commit 과의 차이를 보고 싶다면 다음과 같은 명령어를 실행하면 된다.

\$ git diff HEAD

```
henry-MacBook-Pro:GitWork henry$ git diff HEAD
diff --git a/README.md b/README.md
index e69de29..ec5ca16 100644
--- a/README.md
+++ b/README.md
@@ -0,0 +1 @@
+<html><head></head></html>
henry-MacBook-Pro:GitWork henry$
```

- 26) 이렇게 하면 현재 Commit 과 이전 Commit 과의 차이를 한 눈에 확인할 수 있다.

- 27) 현재 명령어를 입력한 HEAD 는 현재 작업하고 있는 Branch 의 최신 Commit 을 참조하는 Pointer 이다.

28) Commit 을 확인했으므로 **git commit** 명령으로 Commit 하자.

```
[henry-MacBook-Pro:GitWork henry]$ git commit -m "Add head tag in README.md"
[main f420f27] Add head tag in README.md
 1 file changed, 1 insertion(+)
henry-MacBook-Pro:GitWork henry$
```

29) Commit 확인도 하자.

```
[henry-MacBook-Pro:GitWork henry]$ git log
commit f420f272b22ad73f05aec593328de1c19ee98811 (HEAD -> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:28:30 2023 +0900

    Add head tag in README.md

commit f7b088f7887a7965e980b57580aacb9f78f41736
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:22:34 2023 +0900

    Add html Tag

commit c15f53b250c9a44e054dc6edff01040d85d7be19
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:21:15 2023 +0900

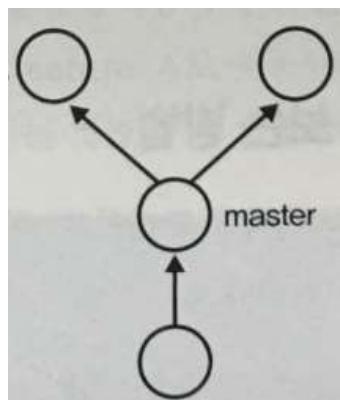
    Add index.html File

commit f8a793ec34832366386535cb19abc8a767883d5c
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 12:20:42 2023 +0900

    Add README.md File
    ...skipping...
```

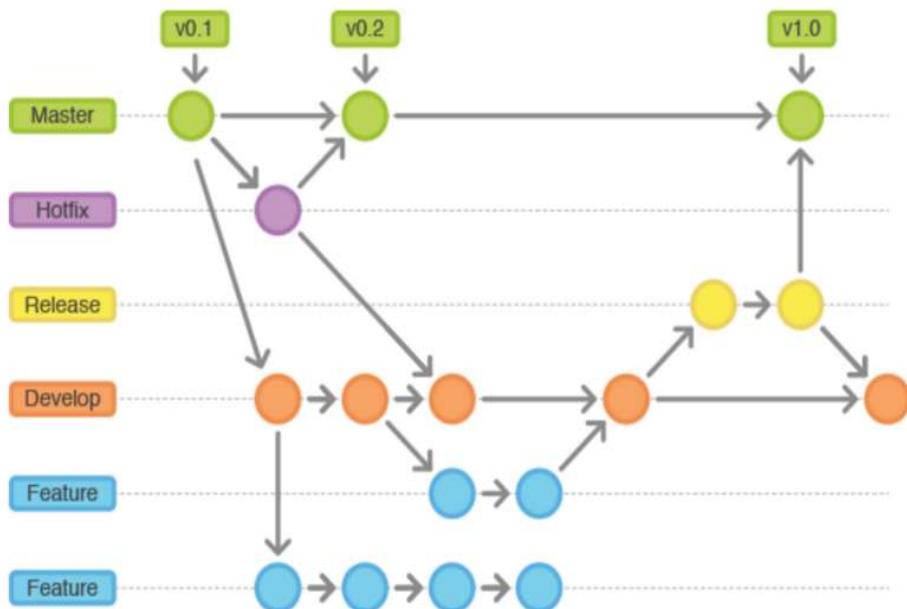
3. git branch

- 1) Branch 를 생성, 수정, 삭제 등을 하는 명령어
 - 2) 여러 Branch 목록을 표시하고, 현재 어떤 Branch 를 사용하고 있는지 확인할 수 있는 명령어.
 - 3) 작업 단위로 나누고, 각자 작업하고, 그리고 합친다라는 개념.
 - 4) **git branch <branch_name>**
 - Branch 생성
 - 5) **git branch -d <branch_name>**
 - Branch 삭제
 - 6) **git branch -m <old_branch_name> <new_branch_name>**
 - Branch 이름 변경
-
- 7) Branch 는 각각의 작업을 병행할 때 사용한다.
 - 8) 보통 **git init** 의 명령어를 통해 Git 을 초기화하면 **main(구 master)** Branch 가 기본적으로 생성된다.
 - 9) 한 번에 여러가지 작업을 수행할 때는 각각의 작업에 따라 Branch 를 나눠서 사용하게 된다.
 - 10) Branch 를 나눠서 개발하는 주된 이유는 main(구 master)라는 Branch 는 항상 배포 가능한 상태여야 하기 때문이다.
 - 11) 따라서 각각의 목적에 따라 Branch 를 개발하고 향후 main(구 master)에 **merge** 하면 된다.
 - 12) 아래의 그림처럼 이렇게 Branch 를 나누면 완전히 다른 작업을 동시에 할 수 있다.



[오오츠카 히로키 저, 윤인성 역, <소셜 코딩으로 이끄는 GitHub 실천기술>, Jpub, 2015, p85]

13) 그리고, 각각의 작업이 모두 완료되면 아래의 그림처럼 Merge 하면 된다.



[<https://medium.com/devsondevs/gitflow-workflow-continuous-integration-continuous-delivery-7f4643abb64f>]

14) 따라서 Branch 를 잘 활용하면 동시에 여러 사람들과 함께 효율적으로 개발할 수 있다.

15) 현재 Branch 를 확인해 보자.

\$ git branch

```
GitWork — bash — 80x23
[henry-MacBook-Pro:GitWork henry$ git branch
* main
henry-MacBook-Pro:GitWork henry$ ]
```

16) [develop]이라는 Branch 를 생성해본다.

\$ git branch develop

```
GitWork — bash — 80x23
[henry-MacBook-Pro:GitWork henry$ git branch develop
henry-MacBook-Pro:GitWork henry$ ]
```

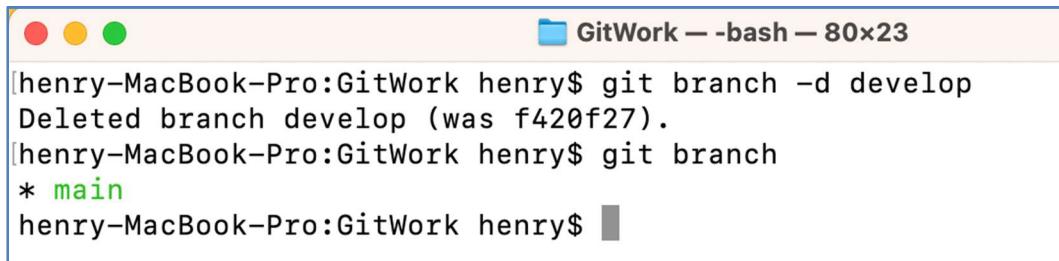
17) 전체 Branch 목록을 본다.

\$ git branch

```
GitWork — bash — 80x23
[henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
henry-MacBook-Pro:GitWork henry$ ]
```

18) 방금 생성한 Branch 인 [develop]을 삭제한다.

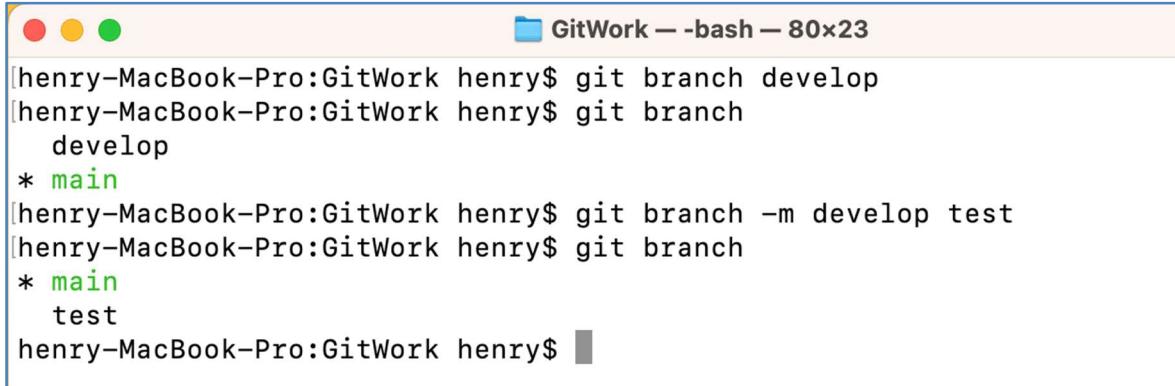
```
$ git branch -d develop
```



```
[henry-MacBook-Pro:GitWork henry$ git branch -d develop
Deleted branch develop (was f420f27).
[henry-MacBook-Pro:GitWork henry$ git branch
* main
henry-MacBook-Pro:GitWork henry$ ]
```

19) 다시 [develop] Branch 를 생성하고 이름을 [test]로 변경해 본다.

```
$ git branch -m develop test
```



```
[henry-MacBook-Pro:GitWork henry$ git branch develop
[henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
[henry-MacBook-Pro:GitWork henry$ git branch -m develop test
[henry-MacBook-Pro:GitWork henry$ git branch
* main
  test
henry-MacBook-Pro:GitWork henry$ ]
```

20) '*' 표시는 현재 Branch 가 어디인지를 표시하는 기호이다.

4. git checkout

- 먼저 실습을 위해 현재 작업 폴더의 모든 파일을 삭제하고, 다시 Git 을 초기화한다.

```
[henry-MacBook-Pro:GitWork henry$ rm -rf .git
[henry-MacBook-Pro:GitWork henry$ ls
README.md      index.html
[henry-MacBook-Pro:GitWork henry$ rm README.md index.html
[henry-MacBook-Pro:GitWork henry$ ls
[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$ ]
```

- Working Directory 의 Source 를 특정 Commit 으로 변경하는 명령어

- git checkout <branch_name>

- 특정 Branch 로 Working Directory 변경

- git checkout -b <branch_name>

- 새로운 Branch 를 생성하고 그 Branch 로 변경하는 명령어
- git branch <branch_name> 과 git checkout <branch_name> 를 순서대로 실행한 것과 동일.

- git checkout <commitID>

- 특정 CommitID 로 Working Directory 변경

- git checkout <file_name>

- 특정 File 을 해당 Branch 또는 Commit 상태로 변경

- 먼저 main(구 master) Branch 에서 README.md File 을 생성한다.

- 그리고 [develop] Branch 를 생성한다.

- 여기서 중요한 것은 어떤 File 도 Commit 한 적 없이 Branch 를 생성하면 아래의 그림과 같은 오류가 발생한다는 것이다.

```
[henry-MacBook-Pro:GitWork henry$ touch README.md
[henry-MacBook-Pro:GitWork henry$ git branch develop
fatal: not a valid object name: 'main'
henry-MacBook-Pro:GitWork henry$ ]
```

10) 그래서 index.html 을 생성한다음, Commit 한 뒤 다시 [develop] Branch 를 생성해 본다.

```
GitWork — bash — 80x23
[henry-MacBook-Pro:GitWork henry$ touch index.html
[henry-MacBook-Pro:GitWork henry$ git add index.html
[henry-MacBook-Pro:GitWork henry$ git commit -m "Add index.html File"
[main (root-commit) 6455314] Add index.html File
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 index.html
[henry-MacBook-Pro:GitWork henry$ git branch develop
[henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
henry-MacBook-Pro:GitWork henry$
```

11) 현재 [main] Branch 에 있다.

12) 여기서 README.md File 의 내용을 본다.

```
GitWork — bash — 80x23
[henry-MacBook-Pro:GitWork henry$ cat README.md
henry-MacBook-Pro:GitWork henry$
```

13) 현재 File 에 아무 내용도 없다.

14) 이제 [develop] Branch 로 이동한다.

\$ **git checkout develop**

15) 이동 후 README.md File 을 수정해 본다.

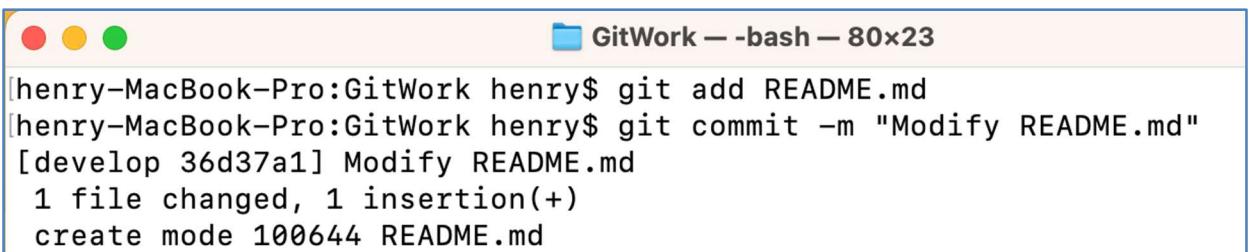
\$ **echo "# Git Tutorial" > README.md**

16) 현재 [develop] Branch 에서는 README.md 의 변경된 내용을 확인할 수 있다.

```
GitWork — bash — 80x23
[henry-MacBook-Pro:GitWork henry$ git checkout develop
Switched to branch 'develop'
[henry-MacBook-Pro:GitWork henry$ git branch
* develop
  main
[henry-MacBook-Pro:GitWork henry$ echo "# Git Tutorial" > README.md
[henry-MacBook-Pro:GitWork henry$ cat README.md
# Git Tutorial
henry-MacBook-Pro:GitWork henry$
```

17) 이제 README.md 를 Commit 한다.

```
$ git add README.md  
$ git commit -m "Modify README.md"
```



```
[henry-MacBook-Pro:GitWork henry]$ git add README.md  
[henry-MacBook-Pro:GitWork henry]$ git commit -m "Modify README.md"  
[develop 36d37a1] Modify README.md  
 1 file changed, 1 insertion(+)  
 create mode 100644 README.md
```

18) 다시 [main] Branch 로 변경한다.

19) README.md File 의 내용을 확인해 보자.

```
[henry-MacBook-Pro:GitWork henry]$ git checkout main  
Switched to branch 'main'  
[henry-MacBook-Pro:GitWork henry]$ cat README.md  
cat: README.md: No such file or directory  
[henry-MacBook-Pro:GitWork henry]$ ls  
index.html  
henry-MacBook-Pro:GitWork henry$
```

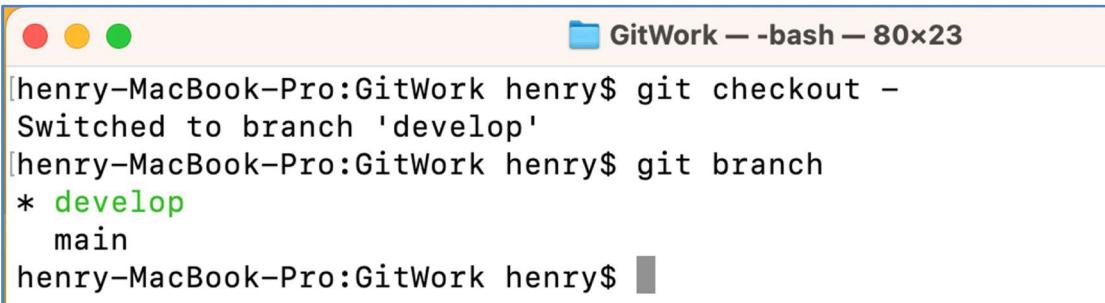
20) 그런데, README.md 파일이 보이지 않는다.

21) 다시 말하면, [develop] Branch 에서 변경했던 내용들이 [main] Branch 에는 영향을 주지 않는다는 것이다.

22) 이것이 Branch 를 이용해서 개발하는 장점이다.

23) Branch 를 여러 개로 나누어서 개발하면 서로의 Branch 에 영향을 주지 않기 때문에 동시에 병행 개발이 가능하다.

24) 방금전의 Branch 로 돌아가려면 아래와 같이 '-' 기호를 사용하면 된다.



```
[henry-MacBook-Pro:GitWork henry]$ git checkout -  
Switched to branch 'develop'  
[henry-MacBook-Pro:GitWork henry]$ git branch  
* develop  
  main  
henry-MacBook-Pro:GitWork henry$
```

25) 다른 예제를 다뤄보자. 모든 파일을 삭제하고, 다시 Git 을 초기화했다.

```
[henry-MacBook-Pro:GitWork henry$ rm -rf .git
[henry-MacBook-Pro:GitWork henry$ ls
 README.md      index.html
[henry-MacBook-Pro:GitWork henry$ rm README.md index.html
[henry-MacBook-Pro:GitWork henry$ ls
[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$
```

26) index.html 파일을 생성해서 Commit 한다.

```
[henry-MacBook-Pro:GitWork henry$ touch index.html
[henry-MacBook-Pro:GitWork henry$ git add index.html
[henry-MacBook-Pro:GitWork henry$ git commit -m "Add index.html"
[main (root-commit) cd1c150] Add index.html
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.html
henry-MacBook-Pro:GitWork henry$
```

27) [develop]라는 Branch 와 [release]라는 Branch 를 생성한다.

```
[henry-MacBook-Pro:GitWork henry$ git branch develop
[henry-MacBook-Pro:GitWork henry$ git branch release
[henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
  release
henry-MacBook-Pro:GitWork henry$
```

28) index.html File 에 간단하게 "<html></html>" Tag 를 입력하여 File 을 변경시킨다.

```
[henry-MacBook-Pro:GitWork henry$ ls
index.html
[henry-MacBook-Pro:GitWork henry$ echo "<html></html>" > index.html
[henry-MacBook-Pro:GitWork henry$ cat index.html
<html></html>
henry-MacBook-Pro:GitWork henry$
```

29) [release] Branch 로 이동한다.

```
[henry-MacBook-Pro:GitWork henry$ git checkout release
M      index.html
Switched to branch 'release'
[henry-MacBook-Pro:GitWork henry$ ls
index.html
[henry-MacBook-Pro:GitWork henry$ cat index.html
<html></html>
henry-MacBook-Pro:GitWork henry$ ]
```

30) [develop] Branch 에 있을 때 변경한 index.html File 을 Commit 하지 않은 상태에서, [release] Branch 로 이동하면 [release] Branch 에서도 변경된 index.html 이 함께 이동한 상태이기 때문에 변경된 내용이 보인다.

31) 이제 **git log** 를 보자.

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

```
[henry-MacBook-Pro:GitWork henry$ git log --oneline --graph --decorate --all
* cd1c150 (HEAD -> release, main, develop) Add index.html
henry-MacBook-Pro:GitWork henry$ ]
```

32) index.html 은 어느 Branch 에서도 확인이 가능하다.

33) 즉, Commit 하지 않고 Branch 를 이동하면 함께 Branch 로 File 도 이동하기 때문이다.

34) 위의 그림처럼 처음 index.html 을 Commit 할 때의 상태를 3 개의 Branch 가 같이 공유하고 있다.

35) [develop] branch 로 이동하여 index.html 을 Commit 한 다음, Log 를 본다.

```
[henry-MacBook-Pro:GitWork henry$ git checkout develop
M      index.html
Switched to branch 'develop'
[henry-MacBook-Pro:GitWork henry$ git commit -am "Add html Tag"
[develop 26457df] Add html Tag
 1 file changed, 1 insertion(+)
[henry-MacBook-Pro:GitWork henry$ git log --oneline --graph --decorate --all
* 26457df (HEAD -> develop) Add html Tag
* cd1c150 (release, main) Add index.html
henry-MacBook-Pro:GitWork henry$ ]
```

36) [release] Branch 로 이동하여 about.html 을 생성한다.

```
[henry-MacBook-Pro:GitWork henry$ git checkout release
Switched to branch 'release'
[henry-MacBook-Pro:GitWork henry$ touch about.html
[henry-MacBook-Pro:GitWork henry$ git status
On branch release
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    about.html

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

37) [main] Branch 로 가서 **git status** 로 상태를 확인해 본다.

```
[henry-MacBook-Pro:GitWork henry$ git checkout main
Switched to branch 'main'
[henry-MacBook-Pro:GitWork henry$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    about.html

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

38) 예상했던 대로 [main] Branch 에서도 [release] Branch 와 같은 상태임을 알 수 있다.

39) 다시 [release] Branch 로 돌아와서 about.html File 을 Commit 하자.

```
[henry-MacBook-Pro:GitWork henry$ git checkout release
Switched to branch 'release'
[henry-MacBook-Pro:GitWork henry$ git add about.html
[henry-MacBook-Pro:GitWork henry$ git commit -m "Add about.html"
[release 51f9a75] Add about.html
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 about.html
[henry-MacBook-Pro:GitWork henry$ git status
On branch release
nothing to commit, working tree clean
henry-MacBook-Pro:GitWork henry$ ]
```

40) [main] Branch 로 이동해서 File 을 확인해 보면 [release] Branch 에서 Commit 했던 about.html File 이 보이지 않는 것을 확인할 수 있다.

```
[henry-MacBook-Pro:GitWork henry$ git checkout main
Switched to branch 'main'
[henry-MacBook-Pro:GitWork henry$ ls
index.html
henry-MacBook-Pro:GitWork henry$ ]
```

41) Git log 를 확인해 보자.

```
[henry-MacBook-Pro:GitWork henry$ git log --oneline --graph --decorate --all
* 51f9a75 (release) Add about.html
| * 26457df (develop) Add html Tag
|/
* cd1c150 (HEAD -> main) Add index.html
henry-MacBook-Pro:GitWork henry$ ]
```

42) 보는 법은 아래에서 위로 가면서 확인하는 것이다.

43) 아래에서는 위쪽이 보이지 않는다고 생각하면 된다.

44) 그래서, 최초 index.html 을 생성했던 곳에서 즉 [main] Branch 에서는 [release] Branch 에서 Commit 했던 about.html File 이 보이지 않는 것이다.

45) 이 File 은 [develop] Branch 에서도 보이지 않는다.

46) [main] Branch 에서 index.html File 을 변경해 보자.

```
[henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
  release
[henry-MacBook-Pro:GitWork henry$ cat index.html
[henry-MacBook-Pro:GitWork henry$ echo "<html></html>" > index.html
henry-MacBook-Pro:GitWork henry$ ]
```

47) index.html 에 "<html></html>" 를 추가했다.

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

48) Commit 하고, Log 를 확인한다.

```
[henry-MacBook-Pro:GitWork henry$ git commit -am "Add html Tag"
[main 900c700] Add html Tag
 1 file changed, 1 insertion(+)
[henry-MacBook-Pro:GitWork henry$ git log --oneline
900c700 (HEAD -> main) Add html Tag
cd1c150 Add index.html
henry-MacBook-Pro:GitWork henry$
```

49) 위의 그림에 보면, 현재 **git log** 를 통해 방금 "html tag"를 추가한 Commit ID 가 900c700 이다.

50) 이 Commit ID 로 **checkout** 을 해 본다.

```
[henry-MacBook-Pro:GitWork henry$ git log --oneline
900c700 (HEAD -> main) Add html Tag
cd1c150 Add index.html
[henry-MacBook-Pro:GitWork henry$ git checkout 900c700
Note: switching to '900c700'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -
```

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 900c700 Add html Tag

51) 다시 말해서, 특정 Commit ID 로 Working Directory 를 변경할 수 있다.

```
[henry-MacBook-Pro:GitWork henry$ ls
index.html
[henry-MacBook-Pro:GitWork henry$ cat index.html
<html></html>
henry-MacBook-Pro:GitWork henry$
```

52) 이동한 Working Directory에서는 index.html File 의 내용이 방금 추가했던 내용으로 확인된다.

53) 다른 CommitID 로 이동해 본다. 아래 그림에서 보면 "Add index.html" Commit Message 의 CommitID 가 cd1c150 이다.

```
[henry-MacBook-Pro:GitWork henry]$ git log --oneline  
900c700 (HEAD, main) Add html Tag  
cd1c150 Add index.html  
[henry-MacBook-Pro:GitWork henry]$ git checkout cd1c150  
Previous HEAD position was 900c700 Add html Tag  
HEAD is now at cd1c150 Add index.html  
[henry-MacBook-Pro:GitWork henry]$ cat index.html  
henry-MacBook-Pro:GitWork henry$
```

54) 새로 이동한 Branch 는 처음 index.html 을 생성하고 Commit 했던 Working Directory 이기 때문에 index.html File 의 내용은 아무것도 없다.

55) 다시 [main] Branch 로 이동해서 Log 를 확인해 본다.

```
[henry-MacBook-Pro:GitWork henry]$ git checkout main  
Previous HEAD position was cd1c150 Add index.html  
Switched to branch 'main'  
[henry-MacBook-Pro:GitWork henry]$ git log --oneline --graph --decorate --all  
* 900c700 (HEAD -> main) Add html Tag  
| * 51f9a75 (release) Add about.html  
|/  
| * 26457df (develop) Add html Tag  
|/  
* cd1c150 Add index.html  
henry-MacBook-Pro:GitWork henry$
```

56) 위의 그림에서 about.html 을 추가했던 CommitID 인 51f9a75 로 Checkout 해보자.

```
GitWork — bash — 80x24
[henry-MacBook-Pro:GitWork henry]$ git checkout 51f9a75
Note: switching to '51f9a75'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 51f9a75 Add about.html
[henry-MacBook-Pro:GitWork henry]$ ls -l
total 0
-rw-r--r--  1 henry  staff  0 Apr  4 15:53 about.html
-rw-r--r--  1 henry  staff  0 Apr  4 15:53 index.html
henry-MacBook-Pro:GitWork henry$
```

57) 예상대로 about.html File 0| 있다.

58) 하지만 [main] Branch 에는 about.html 이 보이지 않는다.

```
GitWork — bash — 80x24
[henry-MacBook-Pro:GitWork henry]$ git branch
* (HEAD detached at 51f9a75)
  develop
  main
  release
[henry-MacBook-Pro:GitWork henry]$ git checkout main
Previous HEAD position was 51f9a75 Add about.html
Switched to branch 'main'
[henry-MacBook-Pro:GitWork henry]$ ls
index.html
henry-MacBook-Pro:GitWork henry$
```

59) 다음은, 특정 CommitID 에 특정 File 로 Working Directory 를 이동하는 것이다.

60) [main] Branch 에서 Log 를 확인한다.

```
GitWork — -bash — 80x24
[henry-MacBook-Pro:GitWork henry$ git log --oneline
900c700 (HEAD -> main) Add html Tag
cd1c150 Add index.html
henry-MacBook-Pro:GitWork henry$ ]
```

61) 위에서 보면, index.html 을 추가했던 Commit ID 가 cd1c150 으로 확인이 된다. 그래서 Commit ID 와 index.html 로 Checkout 해본다.

```
GitWork — -bash — 80x24
[henry-MacBook-Pro:GitWork henry$ git checkout cd1c150 index.html
Updated 1 path from 0a5568b
[henry-MacBook-Pro:GitWork henry$ cat index.html
henry-MacBook-Pro:GitWork henry$ ]
```

62) 특정 Commit ID 의 특정 File 로도 Working Directory 를 이동할 수 있다는 것을 알 수 있다.

63) 마지막으로 특정 CommitID 를 이용해서 Checkout 하면 원복할 수 있다는 것을 배워보자.

```
GitWork — -bash — 80x24
[henry-MacBook-Pro:GitWork henry$ git log --oneline
900c700 (HEAD -> main) Add html Tag
cd1c150 Add index.html
henry-MacBook-Pro:GitWork henry$ ]
```

64) 즉, 위의 log 를 보면 900c700 으로 Checkout 하면 index.html 의 내용이 "<html></html>" 을 추가한 시점으로 이동할 것이고, cd1c150 으로 Checkout 하면 비워져 있던 초기 index.html 의 시점으로 이동할 것이다.

```
GitWork — -bash — 80x24
[henry-MacBook-Pro:GitWork henry$ git checkout 900c700 index.html
Updated 1 path from 1fe2297
[henry-MacBook-Pro:GitWork henry$ cat index.html
<html></html>
henry-MacBook-Pro:GitWork henry$ ]
```

65) 결론은, Checkout 을 이용하면 File 의 원복시점도 정할 수 있다.

5. git merge

- 1) 다른 2 개의 Source 를 병합하는 명령어

```
$ git merge develop
```

- 2) 실습을 위해 GitWork 폴더를 Git 으로 초기화한다.

```
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$
```

- 3) Merge 가 잘되는 경우 vs Merge 가 실패하는 경우

- 4) 먼저 [main] Branch 에서 index.html 을 생성하고 "<html>" 를 넣고 Staging Area 에 올렸다.

```
henry-MacBook-Pro:GitWork henry$ touch index.html
henry-MacBook-Pro:GitWork henry$ echo "<html></html>" > index.html
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

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

- 5) 이후 Commit 하고 Log 를 확인했다.

```
henry-MacBook-Pro:GitWork henry$ git commit -m "Add index.html File"
[main (root-commit) 4e31deb] Add index.html File
 1 file changed, 1 insertion(+)
  create mode 100644 index.html
henry-MacBook-Pro:GitWork henry$ git log
commit 4e31debe80f4ce8a1017bfe2c150634cc6aade7e (HEAD --> main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 17:26:50 2023 +0900

  Add index.html File
henry-MacBook-Pro:GitWork henry$
```

- 6) 이제 [develop] Branch 를 만들고 이동한다.

```
[henry-MacBook-Pro:GitWork henry$ git branch develop
[henry-MacBook-Pro:GitWork henry$ git checkout develop
Switched to branch 'develop'
[henry-MacBook-Pro:GitWork henry$ git branch
* develop
  main
henry-MacBook-Pro:GitWork henry$ ]
```

- 7) 방금 생성한 [develop] Branch 에서 README.md File 을 생성한 후 Commit 까지 진행한다.

```
[henry-MacBook-Pro:GitWork henry$ touch README.md
[henry-MacBook-Pro:GitWork henry$ git add README.md
[henry-MacBook-Pro:GitWork henry$ git commit -m "Add README.md File"
[develop 9a4b892] Add README.md File
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
henry-MacBook-Pro:GitWork henry$ ]
```

```
[henry-MacBook-Pro:GitWork henry$ git log
commit 9a4b892f352e38fd05ea7f5916d504454cc91af2 (HEAD -> develop)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 17:28:02 2023 +0900

    Add README.md File

commit 4e31debe80f4ce8a1017bfe2c150634cc6aade7e (main)
Author: henry <javaexpert@nate.com>
Date:   Tue Apr 4 17:26:50 2023 +0900

    Add index.html File
henry-MacBook-Pro:GitWork henry$ ]
```

- 8) 이렇게 되면 [main] Branch 에서는 index.html 을 Commit 했고, [develop] Branch 에서는 README.md 를 Commit 했다.
9) 이제 2 개의 Source 를 Merge 해 보자.

10) [main] Branch 로 Checkout 했다.

```
GitWork — bash — 80x24
[henry-MacBook-Pro:GitWork henry$ git checkout main
Switched to branch 'main'
[henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
[henry-MacBook-Pro:GitWork henry$ git merge develop
Updating 4e31deb..9a4b892
Fast-forward
  README.md | 0
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 README.md
henry-MacBook-Pro:GitWork henry$
```

11) 위의 그림처럼 [main] Branch 에서 [develop] Branch 와 Merge 하면, [develop] Branch 에 있던 README.md File 이 [main] Branch 로 넘어오게 된다.

```
GitWork — bash — 80x24
[henry-MacBook-Pro:GitWork henry$ ls
README.md      index.html
henry-MacBook-Pro:GitWork henry$
```

12) 즉 README.md File 은 이제 [main] Branch 에서도 작업할 수 있게 됐다.

13) 다음은 Conflict 가 발생하는 Merge 를 해보자.

14) 먼저 Merge 실습 시나리오이다.

- [main] Branch 에서 File 생성 및 Commit
- [develop] Branch 로 변경
- [develop] Branch 에서 File 변경 및 Commit
- [main] Branch 로 변경
- [main] Branch 에서 File 변경
- Merge

```
GitWork — bash — 80x24
[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$
```

15) 먼저 Hello.java 의 코드는 아래와 같다.

```
[henry-MacBook-Pro:GitWork henry$ nano Hello.java
[henry-MacBook-Pro:GitWork henry$ cat Hello.java
package com.example;

public class Hello{
    public static void main(String [] args){
        System.out.println("message = %s\n", "Hello, World");
    }
}
henry-MacBook-Pro:GitWork henry$ ]
```

16) 이 File 을 Staging Area에 Add 하고 Commit 한다.

```
[henry-MacBook-Pro:GitWork henry$ git add Hello.java
[henry-MacBook-Pro:GitWork henry$ git commit -m "Add Hello.java File"
[main (root-commit) 3d6d417] Add Hello.java File
 1 file changed, 7 insertions(+)
 create mode 100644 Hello.java
henry-MacBook-Pro:GitWork henry$ ]
```

17) [develop] Branch 를 생성하고 [develop] Branch 로 Checkout 한다.

```
[henry-MacBook-Pro:GitWork henry$ git branch develop
[henry-MacBook-Pro:GitWork henry$ git checkout develop
Switched to branch 'develop'
henry-MacBook-Pro:GitWork henry$ ]
```

18) [develop] Branch 에서 방금 [main] Branch 에서 Commit 했던 Hello.java 의 Source 를 수정한다.

19) **System.out.print()** 부분이 잘못 Coding 되어 일부분을 수정하기로 했다.

```
[henry-MacBook-Pro:GitWork henry$ nano Hello.java
[henry-MacBook-Pro:GitWork henry$ cat Hello.java
package com.example;

public class Hello{
    public static void main(String [] args){
        System.out.printf("message = %s\n", "Hello, World");
    }
}
henry-MacBook-Pro:GitWork henry$ ]
```

20) [develop] Branch에서 Hello.java 를 수정했기 때문에 현재 이 File 은 Untracked 상태일 것이다.

```
henry-MacBook-Pro:GitWork henry$ git status
On branch develop
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.java

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

21) 다시 이 File 을 [develop] Branch에서 Commit 한다.

```
henry-MacBook-Pro:GitWork henry$ git commit -am "Modify Hello.java"
[develop 8e335ad] Modify Hello.java
 1 file changed, 1 insertion(+), 1 deletion(-)
henry-MacBook-Pro:GitWork henry$
```

22) 이제 [main] Branch로 Checkout 한 후, Hello.java 의 Source 를 확인해본다.

```
henry-MacBook-Pro:GitWork henry$ git checkout main
Switched to branch 'main'
henry-MacBook-Pro:GitWork henry$ cat Hello.java
package com.example;

public class Hello{
    public static void main(String [] args){
        System.out.println("message = %s\n", "Hello, World");
    }
}
henry-MacBook-Pro:GitWork henry$
```

23) 처음 Source 의 Logic 이 변경되어 출력부분을 다시 수정하기로 하였다.

24) 그래서 아래와 같이 Source 를 변경한 후, 다시 Commit 한다.

```
henry-MacBook-Pro:GitWork henry$ nano Hello.java
henry-MacBook-Pro:GitWork henry$ cat Hello.java
package com.example;

public class Hello{
    public static void main(String [] args){
        System.out.println("message = Hello, World");
    }
}
henry-MacBook-Pro:GitWork henry$
```

```
henry-MacBook-Pro:GitWork henry$ git commit -am "Modify Hello.java again"
[main 8b37728] Modify Hello.java again
 1 file changed, 1 insertion(+), 1 deletion(-)
henry-MacBook-Pro:GitWork henry$
```

25) 이 상태에서 [develop] Branch 와 Merge 를 시도한다.

```
henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
henry-MacBook-Pro:GitWork henry$ git merge develop
Auto merging Hello.java
CONFLICT (content): Merge conflict in Hello.java
Automatic merge failed; fix conflicts and then commit the result.
henry-MacBook-Pro:GitWork henry$
```

26) 결과는 충돌(conflict)가 발생하여 Merge 에 실패했다.

27) 이럴 경우에는 자동으로 Merge 가 되지 않기 때문에 결국 수동으로 하나씩 Conflict 되는 부분을 찾아서 수정한 후 다시 Commit 해야 한다.

28) 현재 상태는 아래와 같다.

```
henry-MacBook-Pro:GitWork henry$ git status
On branch main
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified:  Hello.java

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

29) 같은 File 의 같은 부분을 두 Branch 에서 수정했기 때문이다.

30) File 의 내용을 보자.

```
[henry-MacBook-Pro:GitWork henry$ cat Hello.java
package com.example;

public class Hello{
    public static void main(String [] args){
<<<<< HEAD
        System.out.println("message = Hello, World");
=====
        System.out.printf("message = %s\n", "Hello, World");
>>>>> develop
    }
}
henry-MacBook-Pro:GitWork henry$ ]
```

31) Conflict 되는 부분은 개발자가 직접 선택을 해야 한다.

32) 아래 [develop] Branch 에서 작성한 Source 를 채택하기로 했다.

```
UW PICO 5.09                               GitWork — nano Hello.java — 80x24
File: Hello.java

package com.example;

public class Hello{
    public static void main(String [] args){
<<<<< HEAD
        System.out.println("message = Hello, World");
=====
        System.out.printf("message = %s\n", "Hello, World");
>>>>> develop
    }
}

^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Pg   ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where is    ^V Next Pg   ^U UnCut Text ^T To Spell
```

33) 그러면, HEAD 부분에 있는 Code 를 지우고 저장한다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ nano Hello.java
henry-MacBook-Pro:GitWork henry$ cat Hello.java
package com.example;

public class Hello{
    public static void main(String [] args){
        System.out.printf("message = %s\n", "Hello, World");
    }
}
henry-MacBook-Pro:GitWork henry$
```

34) 수정된 File 을 다시 Commit 한다.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ git branch
  develop
* main
henry-MacBook-Pro:GitWork henry$ git commit -am "Modify Hello.java because merge failure"
[main f76ba97] Modify Hello.java because merge failure
henry-MacBook-Pro:GitWork henry$
```

35) 최종적으로 Log 를 확인해보자.

```
GitWork — bash — 80x24
henry-MacBook-Pro:GitWork henry$ git log --oneline --decorate --graph --all
*   f76ba97 (HEAD -> main) Modify Hello.java because merge failure
|\ 
| * 8e335ad (develop) Modify Hello.java
* | 8b37728 Modify Hello.java again
|/
* 3d6d417 Add Hello.java File
henry-MacBook-Pro:GitWork henry$
```

36) 처음 Hello.java 가 추가된 후부터 다른 Branch 에서 수정됐다가 최종적으로 [main] Branch 에서 Merge 된 것을 알 수 있다.