

branch merge 연습하기

≡ 구분 DB

Branch

나무가지 처럼 작업 공간을 나누어 독립적으로 작업할 때 사용
왜 나누는가?

가장 큰 이유는 원본(master)에 대한 안정성 확보라고 할 수 있다.

- master banch (제품으로 출시할 수 있는 프랜치 -> 상용화가 가능한 상태)
- develop branch (다음 버전 출시를 위한 준비 (예를들자면> 기존 버전에 작업속도 업그레이드))
- feature branch (다음 버전에 추가될 새로운 기능을 개발하는 branch)

```
m-v1  -----> m-v2      m-v3
          |
          |   → hotfix → dev → dev
          |   → dev -> dev
          |
          |   → feature   →
```

=====

branch를 생성하고 그리고 병합하는 것을 먼저 연습할 것입니다.

그리고 협업시 branch를 어떻게 이용할 수 있는지도 연습해 보겠습니다.

먼저 사전셋팅 부터 해보자.

1. 홈 디렉토리에 test 폴더를 생성하고 vscode를 실행합니다. 그리고 Git 저장소를 생성합니다.

```
$ git init
```

```
Initialized empty Git repository in C:/Users/SSAFY/Desktop/test/.git/
```

2. test.txt 를 생성하고 각각 master-1, master-2 그리고 master-3 이라는 내용을 순서대로 입력하면서 커밋을 총 3개를 작성합니다.

```
$ touch test.txt
```

```
# 그 다음에는 test.txt 파일 안에 "master-1" 라고 작성합니다. 그 다음
```

```
$ git add .
```

```
$ git commit -m "master-1"

# test.txt 파일 안에 "master-2" 라고 작성합니다. 그 다음

$ git add .
$ git commit -m "master-2"

# test.txt 파일 안에 "master-3" 라고 작성합니다. 그 다음

$ git add .
$ git commit -m "master-3"
```

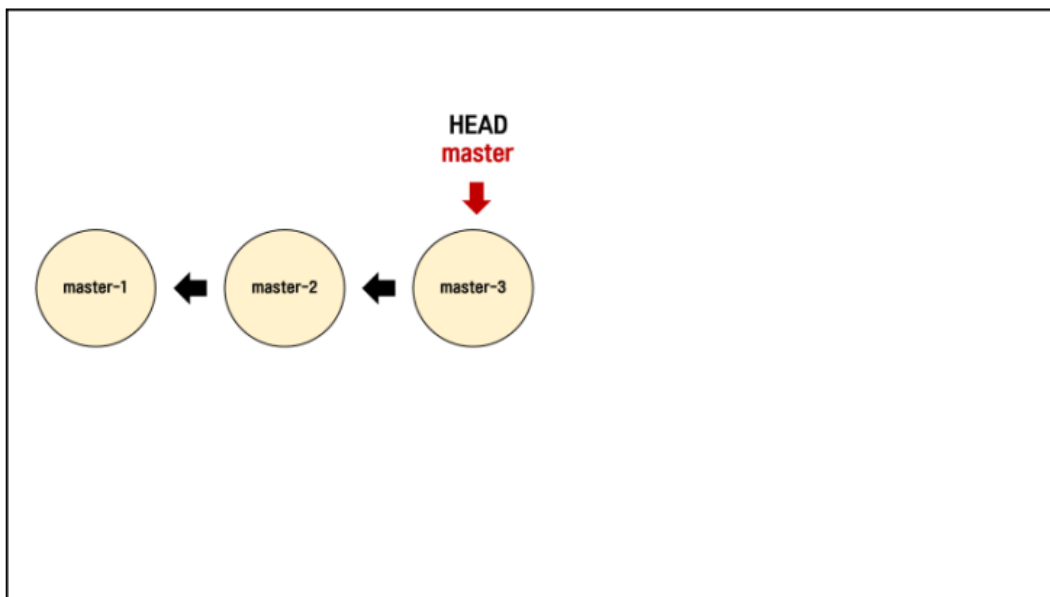
3. git log --oneline 을 입력했을 때 아래와 같이 나와야 정상입니다.
총 3개의 버전이 master 브랜치에 만들어졌습니다.

```
$ git log --oneline

0604dcd (HEAD -> master) master-3
9c22c89 master-2
3d71510 master-1
```

현재까지의 결과를 그림으로 나타 내자면 다음과 같습니다.,

현재까지의 결과



여기서 HEAD 가 의미하는 것은 지금 작업을 하고 있는 대상을 의미 합니다. HEAD가 가리키는 브랜치는 master 라는 뜻 입니다.

위 그림을 보면 커밋 진행 방향과 화살표가 방향이 다르다는 것을 확인 할 수 있습니다.

이는 커밋을 하면 이전 커밋 이후에 변경사항만 기록한 것으로 새로 생성된 커밋은 이전 커밋에 종속되어 생성된 것이라는 것을 표기하기 위해서 보통 그림으로 표현할 시에 화살표 방향이 거꾸로 표기 합니다.

4. login 이라는 이름의 브랜치를 새로 생성 해 보겠습니다.

```
$ git branch login
```

그리고 branch가 잘 생성 되었는지 확인 합니다.

```
$ git branch
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (master)
$ git branch
  login
* master
```

브랜치를 확인 하기 위해 git branch 라고 작성했고

현재 *이 가르키는 곳은 master 입니다. 이때 *은 head를 뜻합니다. head는 무엇인가?

head는 현재 작업을 하고 있는 대상을 의미 합니다.

login 이라는 branch가 생성됨을 확인 할 수 있습니다.

이번에는 \$ git log --oneline 이라고 작성 해 보겠습니다.

```
$ git log --oneline
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (master)
$ git log --oneline
91ca616 (HEAD -> master, login) master-3
8b69c94 master-2
f9a0c7b master-1
```

마찬가지로

91ca616 커밋 기록을 보면 head 가 가르키는 곳은 master 이고 login 이라는 branch 가 있다는 것을 확인 할 수 있습니다.

master 브랜치에서 1개의 커밋을 더 작성합니다.

```
# test.txt에 master-4 를 추가로 작성 후 add commit 합니다.
```

```
$ git add .  
$ git commit -m "master-4"
```

그리고 현재 커밋 상태를 한번 더 확인해 보겠습니다.

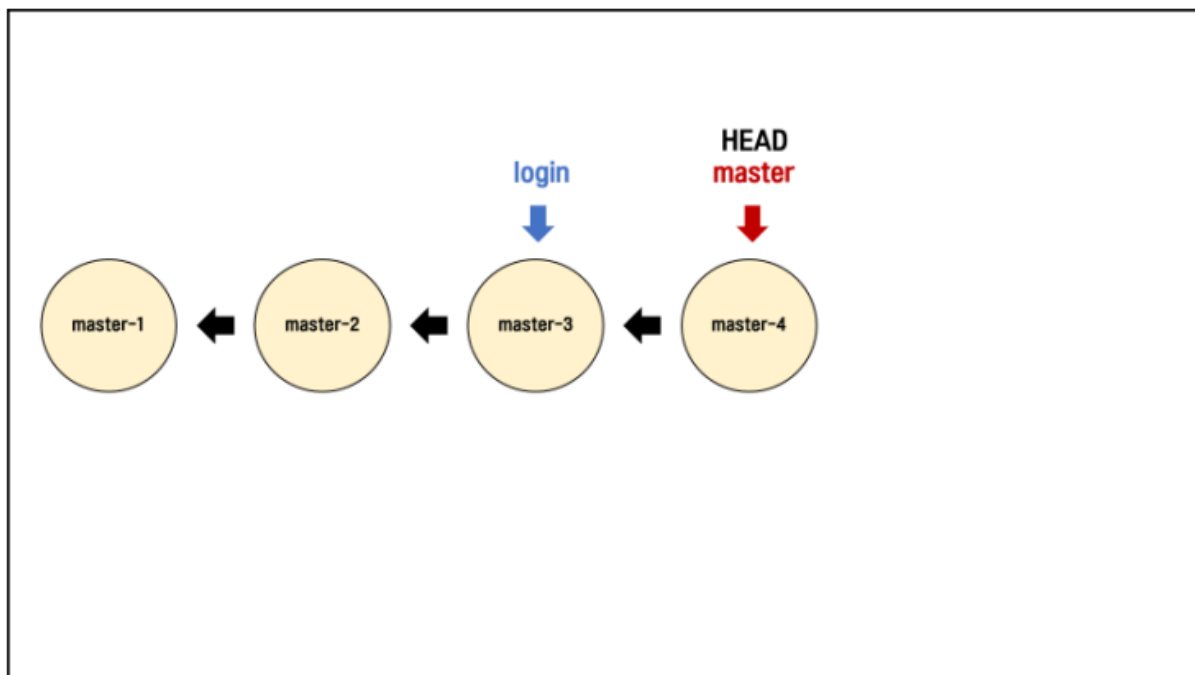
```
$ git log --oneline
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (master)  
$ git log --oneline  
0913847 (HEAD -> master) master-4  
91ca616 (login) master-3  
8b69c94 master-2  
f9a0c7b master-1
```

지금까지

login 브랜치는 master-3 커밋 후에 생성을 했고 그 이후에는
master 브랜치로 master-4 작업 했습니다. 그리고
add, commit 까지 완성했습니다.

지금 상황을 그림으로도 한번 살펴 보자면 아래와 같을 것입니다.



자 이제 브랜치 이동을 해 보겠습니다. 지금 작업대상 Head가 가르키고 있는곳은 master 브랜치 입니다. 이제 Master 브랜치에서 → login 브랜치로 이동하겠습니다.

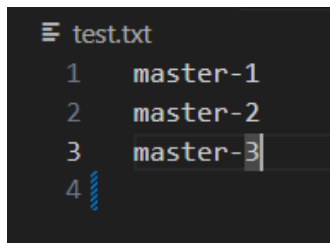
브랜치 이동시 switch 명령어를 사용하시면 됩니다.

```
$ git switch login
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (master)
$ git switch login
Switched to branch 'login'
```

이동을 하고 vscode 에 test.txt 파일을 확인해 보면

방금 만든 master-4 는 보이지 않습니다.



```
test.txt
1 master-1
2 master-2
3 master-3
4
```

\$ git log --oneline 으로 지금 log 상황을 살펴보겠습니다.

```
git log --oneline
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (login)
$ git log --oneline
91ca616 (HEAD -> login) master-3
8b69c94 master-2
f9a0c7b master-1
```

커밋 이력을 보니까 master 브랜치가 보이지 않습니다. 그 이유는 지금 현재 head가 가르키는 대상이 login 브랜치이기 때문 입니다.

master 브랜치가 사라질 리가 없으므로 git branch를 통해서 지금 branch 상황을 확인해 보겠습니다.

```
$ git branch
```

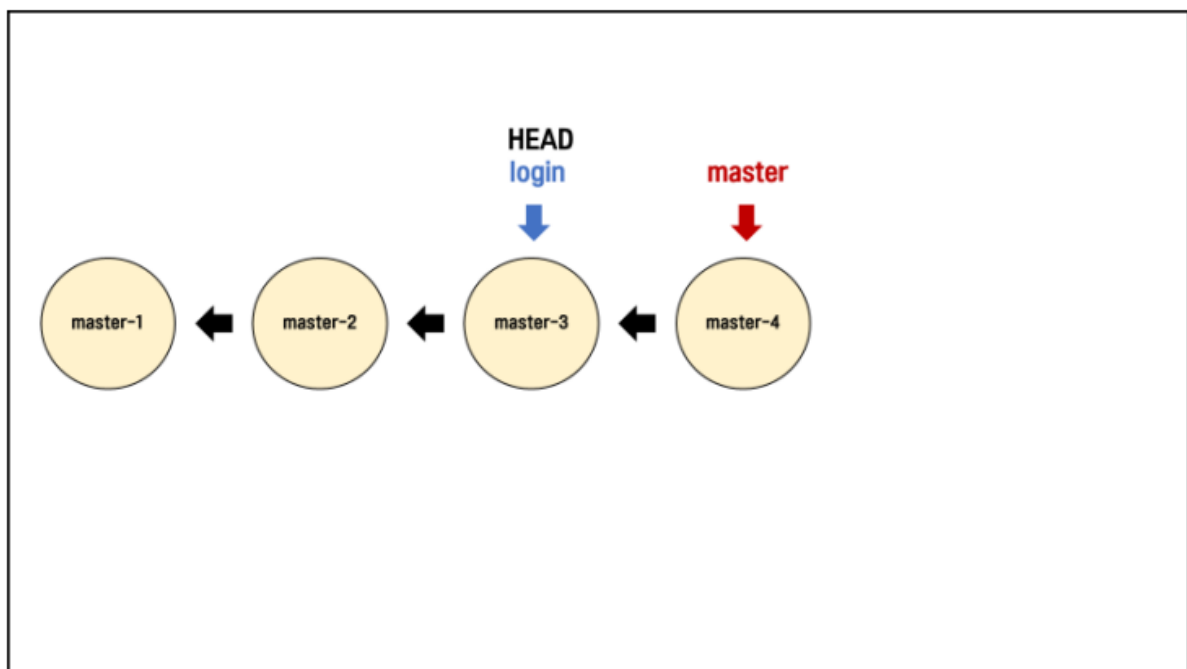
```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (login)
$ git branch
* login
  master
```

*은 head 이며, head는 현재 작업 대상을 의미한다고 했습니다.

현재 head가 가르키는 곳은 login 이며,

master 브랜치도 잘 있음을 눈으로 확인 할 수 있습니다.

지금 상황을 그림으로 보자면 다음과 같을 것입니다.



이번에는 \$ git log --oneline 에 --all 옵션을 더하여

모든 브랜치의 커밋이력을 확인해 보겠습니다.

```
$ git log --oneline --all
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (login)
$ git log --oneline --all
0913847 (master) master-4
91ca616 (HEAD -> login) master-3
8b69c94 master-2
f9a0c7b master-1
```

자 지금은 head가 login을 가르키고 있습니다.

이제 login 브랜치에서 커밋을 생성 하도록 하겠습니다.

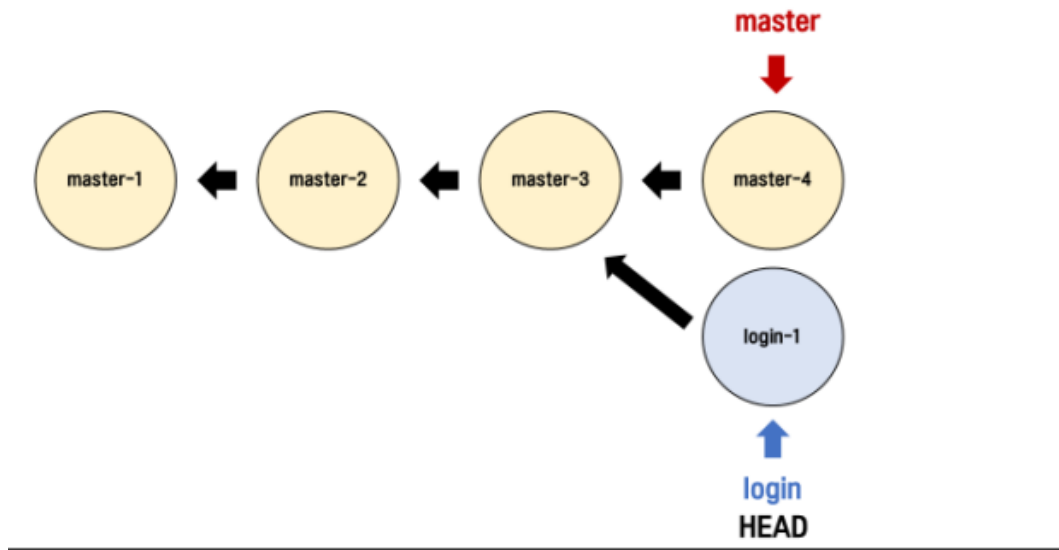
test_login.txt 파일을 생성하고 login-1이라고 작성해 보겠습니다.

```
$ echo login-1 > test_login.txt  
  
$ git add .  
$ git commit -m "login-1"
```

이번에는 모든 브랜치의 커밋을 그래프로 확인해 보겠습니다.

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

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (login)  
$ git log --oneline --graph --all  
* 7ca1d34 (HEAD -> login) login-1  
| * 0913847 (master) master-4  
|/  
* 91ca616 master-3  
* 8b69c94 master-2  
* f9a0c7b master-1
```



이번에는 두 브랜치를 합치는 merge를 해보도록 하겠습니다.

login 브랜치의 내용을 master 브랜치로 병합 하도록 하겠습니다.

그렇다면 login 브랜치에서 생성한 test_login.txt 파일이 master 브랜치에서도 확인이 가능 할 것입니다.

우리는 login 브랜치를 master 브랜치로 병합 할 것이기 때문에

HEAD (가르키는 대상)을 master 브랜치로 먼저 이동 후,

git merge <합칠 브랜치 이름> 을 통해서 병합을 할 수 있습니다.

(항상 Merge하기 전에 일단 다른 브랜치를 합치려고 하는, 즉 메인 브랜치로 switch 해야합니다.)

```
$ git switch master
```

```
$ git merge login
```

방금 login 브랜치를 master 브랜치로 병합을 했습니다.

vscode에 탐색기(explorer)를 확인해 보면 test_login.txt 파일이 추가 되어 있는 것을 확인 할 수 있습니다.

병합이 잘 되었는지 커밋 이력을 확인해 보겠습니다.

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



```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (master)
$ git log --oneline --all --graph
* 5d1a31a (HEAD -> master) Merge branch 'login'
| \
| * 7ca1d34 (login) login-1
* | 0913847 master-4
|/
* 91ca616 master-3
* 8b69c94 master-2
* f9a0c7b master-1
```

병합 후에는 5d1a31a 라는 해시코드와 함께 병합 후 새로운 커밋 이력이 생성 되었습니다.
(해시코드는 각자 다릅니다!!)

[참고]

여기서 [주의사항]을 하나 말씀 드리겠습니다.

병합을 위해서 login 브랜치에서 master 브랜치로 switch를 했었습니다.

그런데 항상 git switch 하기 전에 확인해야 할 사항이 있습니다.

switch를 하기 전에 Working Directory 에 작업하던 파일이 있다면 반드시 버전관리가 되고 있는 상태 인지 반드시 확인해야 합니다.

만약에 버전관리가 되고 있지 않는 파일이 있는 상태에서 switch를 진행 하였다면

working directory에서 버전관리가 되고 있지 않는 파일은 switch가 된 브랜치 에서도 똑같이 생성이 될 것이기 때문입니다.

따라서 switch 하기 전에는 버전관리가 안되고 있는 파일이 있는지 반드시 확인 해 주세요.

이제 login 브랜치는 필요가 없으므로 삭제를 하겠습니다.

```
$ git branch -d login
```

삭제 후 확인까지 해보겠습니다.

```
$ git branch
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (master)
$ git branch
* master
```

login 브랜치는 사라지고 master 브랜치만 존재하고 있는것을 확인 했습니다.

그리고 head는 master 브랜치를 가르키고 있습니다.

자 이렇게 병합을 간단하게 해 보았습니다. 그런데 한번 생각해 봐야 할 점이 있습니다.

방금 실습은 master에서 login 브랜치 분기 후에 master-4 라고 내용을 추가했고

login 브랜치에서는 test_login.txt 라는 파일만 생성을 했습니다.

즉, 두 브랜치를 병합하기 전에 각각의 브랜치에서 수정했던 부분이 달라서 아무 문제 없이 병합이 잘 되었습니다.

그런데 말입니다, 만약에 login 브랜치에서 수정한 파일의 내용과

master 브랜치에서 수정한 파일을 내용이 같아서 수정 내용이 겹친다면 어떻게 될까요?

만약에 같은 파일을 각각 수정했더라도 수정한 부분이 달라서 서로 영향을 끼치지 않는다면

병합하는데 아무 문제가 발생하지 않지만,

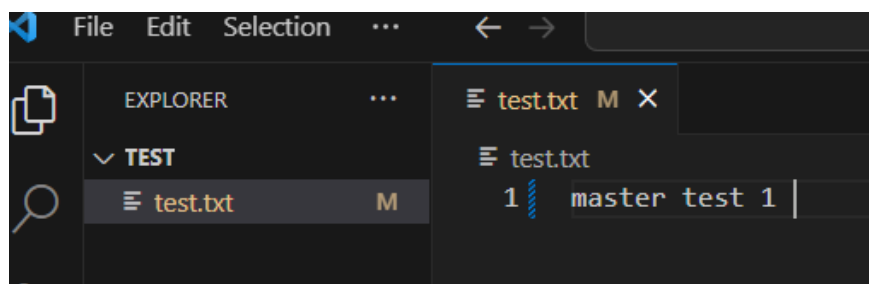
수정한 부분이 겹치거나 서로의 파일에 영향을 준다면 merge conflict 라고 해서 충돌이 일어 납니다.

만약에 수정한 부분이 겹쳤다면 git merge 후에 충돌 내용을 확인하고 수동으로 수정 후 다시 commit을 해야 합니다. (곧 실습 합니다.)

따라서 branch를 나눌 때에는, 병합 후 충돌이 나지 않도록 역할과 기능을 잘 분배하여 branch를 나눠야 하며 최대한 수정 부분이 겹치지 않도록 해야 할 것입니다.

지금부터는 몇 가지 상황을 그림을 통해서 설정을 하고 merge 연습을 더 해보겠습니다.

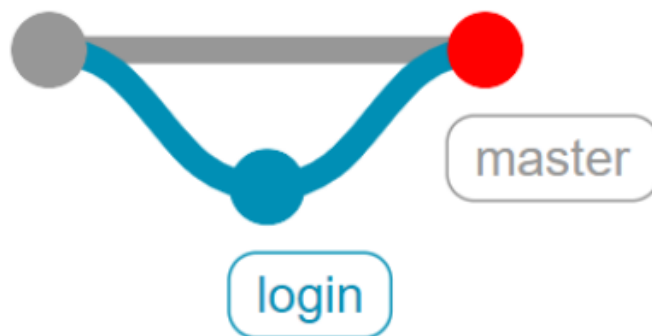
사전셋팅으로



- test_login.txt 파일을 삭제하고
- test.txt 에 master test 1 을 입력 후 저장 하도록 하겠습니다.

```
$ git status
$ git add .
$ git commit -m "master test 1"
```

첫번째 상황 셋팅은 아래 그림과 같습니다.



1. login branch 다시 생성 및 이동

git switch 이후에 -c 옵션까지 붙이면

login 브랜치를 생성과 동시에 head가 login으로 이동을 합니다.

```
$ git switch -c login
```

2. login 브랜치에서 login.txt 파일을 생성 합니다.

```
$ touch login.txt
$ git add .
$ git commit -m "login test 1"
```

3. master에 login을 병합 후 결과를 확인해 보겠습니다.

```
$ git switch master
$ git merge login
```

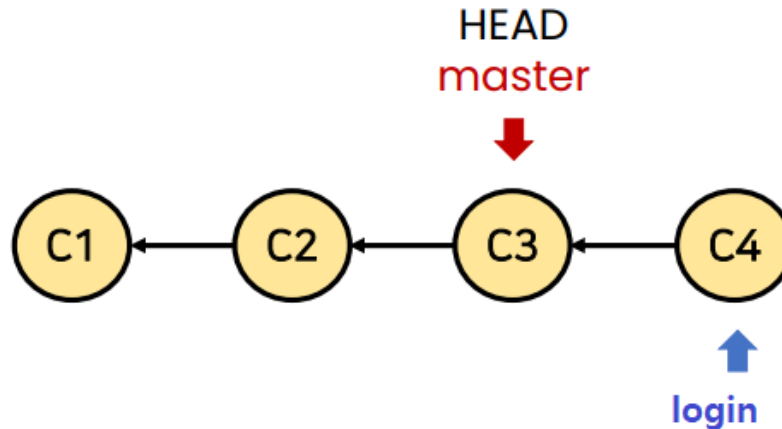
```
$ git log --oneline --all --graph
```

```
SSAFY@DESKTOP-MDJG1NJ MINGW64 ~/Desktop/test (master)
$ git log --oneline --all --graph
* 444ceb6 (HEAD -> master, login) login test 1
* 39a0e9f master test 1
* 5d1a31a Merge branch 'login'
| \
|  * 7ca1d34 login-1
* | 0913847 master-4
|/
* 91ca616 master-3
* 8b69c94 master-2
* f9a0c7b master-1
```

위 사진에서 빨간 박스 부분이 방금 실습한 내용입니다.

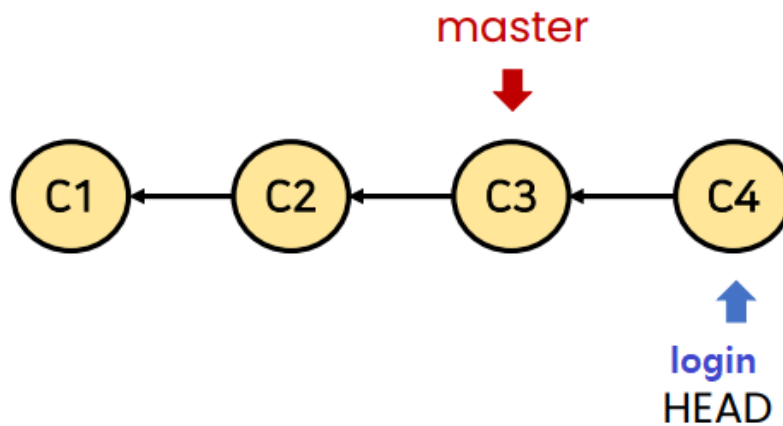
아래 그림으로 도 살펴 보겠습니다.

master에서 login 브랜치를 생성한 이후에 master 브랜치에서는 변경사항이 없었습니다.

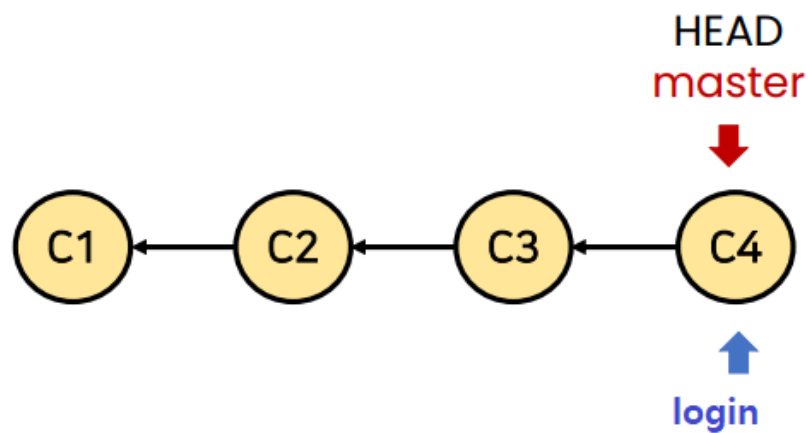


생성과 동시에 login 브랜치로 head가 옮겨 갔습니다.

그리고 login 브랜치에서만 변경사항이 있습니다. (login test 1 파일생성)



그리고 바로 master로 switch 를 한 이후에
login 브랜치를 master 브랜치로 merge 한 상황 입니다.



이럴 경우에는 그저 master 브랜치가 login 브랜치와 동일한 커밋을 가르키도록 이동만 합니다.
이를 앞으로 빨리감기와 같다는 의미로 Fast-forward 라고 합니다.

실습이 끝났으니까 login 브랜치 삭제 하겠습니다.

```
$ git branch -d login
```

자. 다음 상황을 보겠습니다. 합병 중 수정 부분이 같아서 충돌이 일어나는 경우를 살펴 볼 것입니다.

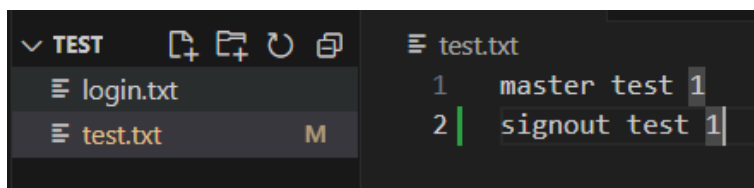


이번에는 signout 이라는 브랜치를 생성 후 이동하겠습니다.

```
$ git switch -c signout
```

현재 head는 signout을 가리키고 있을 것입니다.

그리고



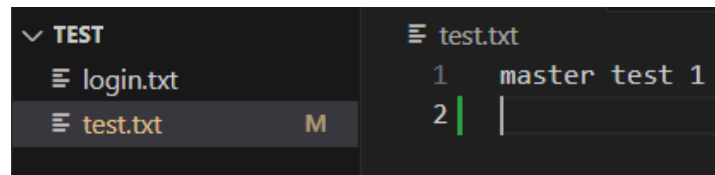
test.txt 파일 안에 signout test 1 라는 글을 추가 후 저장합니다.

```
$ git add .
$ git commit -m "signout test 1"
```

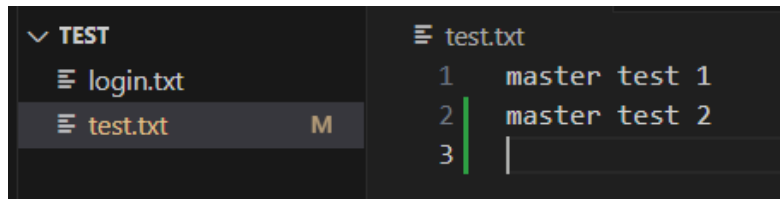
그리고 master 브랜치로 이동합니다.

```
$ git switch master
```

그러면 test.txt 파일을 확인해 보면 아래 사진과 같이
방금 작성한 signout test 1 글이 보이지 않아야 정상입니다.



이제 master test 2 내용을 추가 하도록 하겠습니다.



그리고 커밋을 진행 합니다.

```
$ git add .
$ git commit -m 'master test 2'
```

지금 상황을 종합 해 보자면,

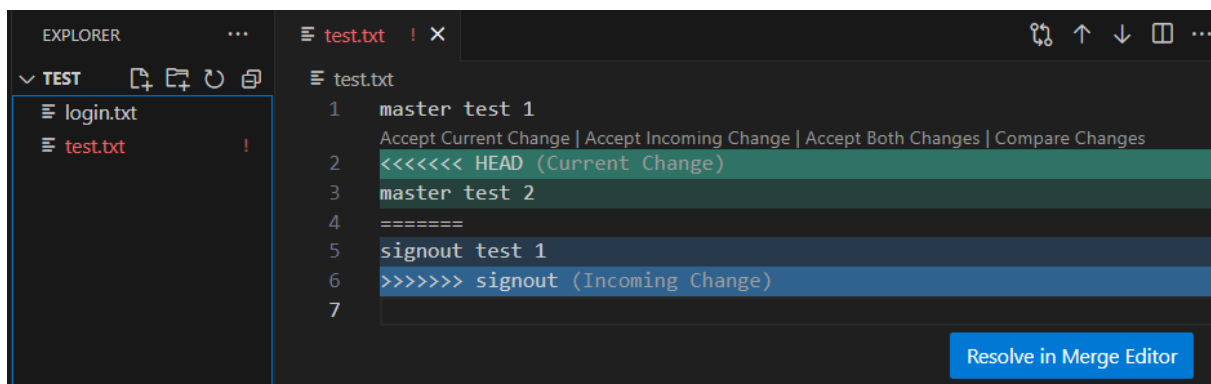
아까는 signout 브랜치에서 master test1 밑에 signout test 1 으로 수정 후 커밋 하였고

지금 master 브랜치로 돌아와서 master test1 밑에 signout test 1 으로 수정 후 커밋 하였습니다.

즉, 같은 파일 안에서 같은 부분을 수정 후 merge를 진행 해 보도록 하겠습니다.

```
$ git merge signout
```

그랬더니 다음과 같은 충돌 메시지가 뜹니다.



충돌 옵션을 살펴 보도록 하겠습니다.

accept current change 그리고 accept incoming change 등이 있습니다.

master test 1 아래에 수정 사항을 선택할 수가 있습니다.

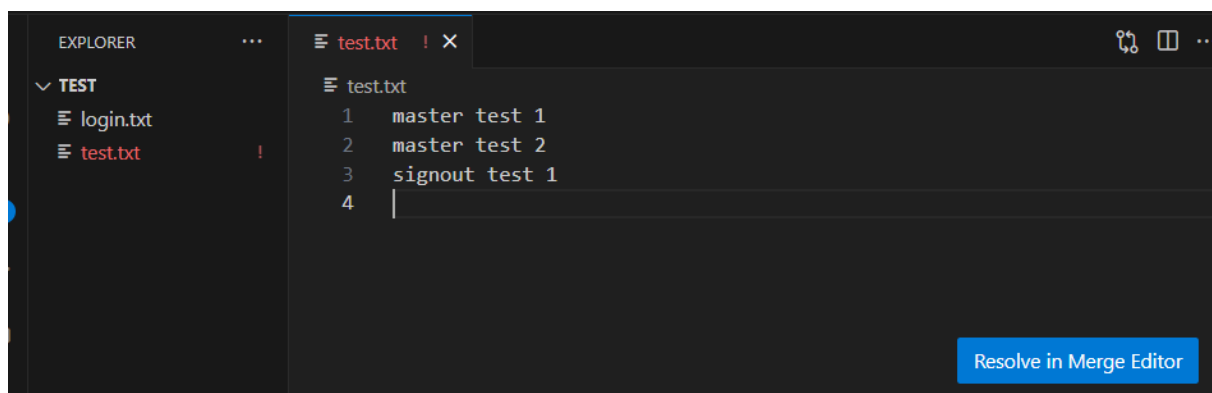
- 만약에 accept current change를 클릭하면

master test 1 아래에는 master test 2를 붙일 것이며

- 만약에 accept incoming change 를 클릭 한다면

master test 1 아래에는 signout test 1이 붙을 것입니다.

accept both changes를 클릭시 두 변화가 모두 적용이 되고 수동으로 변경사항을 수정 할 수 있습니다.



수정이 완료 되었다면 resolve in merge Editor (파란박스) 클릭 후
complete merge를 클릭 하시면 완료 됩니다. 클릭 클릭 클릭
그리고 마지막으로 commit을 하면 병합 완료 입니다.

```
$ git commit -m '병합완료'
```

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



```

* 06db61d (HEAD -> master) 병합완료
| \
| * 2d28606 (signout) signout test 1
* | 988697f master test 2
|/
* 444ceb6 login test 1
* 39a0e9f master test 1
* 5d1a31a Merge branch 'login'
| \
| * 7ca1d34 login-1
* | 0913847 master-4
|/
* 91ca616 master-3
* 8b69c94 master-2
* f9a0c7b master-1

```

자 이렇게 branch와 병합을 실습 해 보았습니다.

필요없는 signout 브랜치를 삭제 하도록 하겠습니다.

```
$ git branch -d signout
```

이제 다른 팀원과 git hub와 같은 원격 저장소를 이용하여 협업을 할 준비가 완료되었습니다.

<끝>