branch scenario

branch 사용법

실습 준비

```
$ mkdir git_branch
$ cd git_branch

$ git init
$ touch a.txt

# a.txt 에 text 1 을 입력 후 저장

$ git status
$ git add .

$ git status
$ git add .

$ git commit -m "text 1"
$ git log

# 위와 같은 흐름으로 2번 더 반복하여 3번째 commit "text 3" 까지 작성해보기

$ git log --oneline
```

- 다른 브랜치에서 로그인 기능을 만든다고 생각해보자.

```
(master) $ git branch login
```

```
(master) $ git branch
  login
* master
```

```
(master) $ git log --oneline
770f719 (HEAD -> master, login) text 3
90c233c text 2
568d9c9 text 1
```

    - `HEAD` : 현재 우리가 속해있는 위치
    - login 브랜치는 현재 마스터가 머물고 있는 text 3 라는 커밋의 위치에서 만들었기 때문
- a.txt 에 다음과 같이 문장을 추가하고 버전을 만들어보자.

```
# a.txt

text 1
text 2
text 3
master text 4
```

```
(master) $ git add .
(master) $ git commit -m "master text 4"
```

```
(master) $ git log --oneline

ce14329 (HEAD -> master) master text 4
770f719 (login) text 3
90c233c text 2
568d9c9 text 1
```

    - 우리는 master text 4 라는 버전으로 이동했고 login 브랜치는 여전히 text 3 에 남겨진 상태
    - **이 상태에서 a.txt 를 vscode로 한쪽 화면에 띄운 뒤 어떻게 변화하는지 확인해보자.**

```
(master) $ git checkout login
```

        - master text 4 문장이 사라진다.

```
(login) $ git log --all --oneline

0eff56d (master) master text 4
8df0e56 (HEAD -> login) text 3
55b9fe1 text 2
726a391 text 1
```

- login 브랜치에서 a.txt 에 다음과 같이 작업하고 login.txt 파일도 만들어서 다음과 같이 입력해보자.

```
# a.txt

text 1
text 2
text 3
login text 4
```

```
(login) $ touch login.txt
```

```
# login.txt

login text 4
```

- 버전을 생성한 후 log 를 확인해보자.

```
(login) $ git status
On branch login
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:   a.txt

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

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

```
(login) $ git add .
(login) $ git commit -m "login text 4"
```

```
(login) $ git log --all --graph --oneline
* 6beda95 (HEAD -> login) login text 4
| * 0eff56d (master) master text 4
|/
* 8df0e56 text 3
* 55b9fe1 text 2
* 726a391 text 1
```

  - 이제야 그래프 처럼 보인다. 우리의 현재 상태는 login 브랜치에 있다.
  - `login text 4` 와 `master text 4`의 부모 커밋은 `text 3` 다.

---

branch 병합

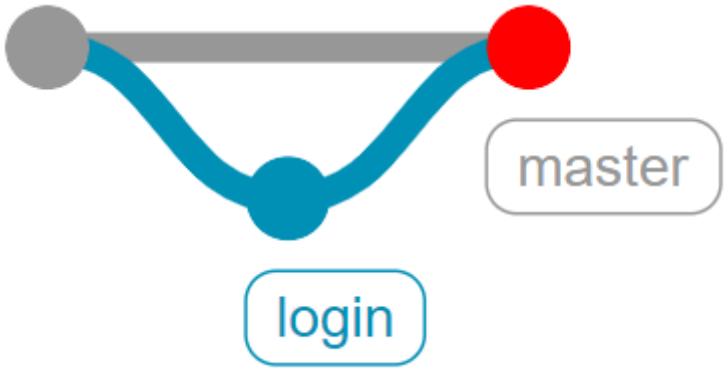
실습준비

```
$ mkdir git_merge
$ cd git_merge

$ git init
$ touch test.txt

# test.txt 에 master test 1 을 입력 후 저장

$ git status
$ git add .

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

3가지 병합 상황

1. fast-forward

> "다른 브랜치가 생성된 이후 master 브랜치에 변경 사항이 없는 상황"
>
> 즉, master 브랜치에서 login 브랜치를 Merge 할 때
> login 브랜치가 master 브랜치 이후의 커밋을 가리키고 있으면
> 그저 master 브랜치가 login 브랜치와 동일한 커밋을 가리키도록 이동시킬 뿐



1. login branch 생성 및 이동

```
$ git checkout -b login
```

2. 특정 작업 완료 후 commit

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

3. master 브랜치로 이동

```
$ git checkout master

$ git log --oneline --all

* 7a79de3 (login) login test 1
* 5910361 (HEAD -> master) master test 1
```

4. master 에 병합

```
$ git merge login

Updating 5910361..7a79de3
Fast-forward
 login.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 login.txt
```

5. 결과 확인 (fast-foward, 단순히 HEAD를 이동)

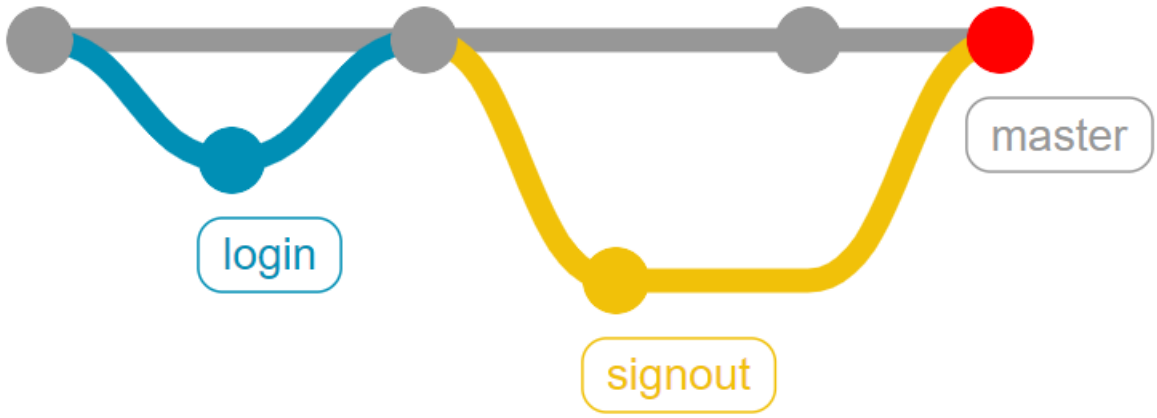
```
$ git log --oneline

7a79de3 (HEAD -> master, login) login test 1
5910361 master test 1
```

6. branch 삭제

```
$ git branch -d login

$ git log --oneline

7a79de3 (HEAD -> master) login test 1
5910361 master test 1
```

2. 3-way Merge (Merge commit)

> 현재 브랜치(master)가 가리키는 커밋이 Merge 할 브랜치의 조상이 아니면,
> git 은 각 브랜치가 가리키는 커밋 2 개와 공통조상 하나를 사용하며 3-way Merge 한다.
>
> 단순히 브랜치 포인터를 최신 커밋으로 옮기는 게 아니라 3-way Merge 의 결과를
> 별도의 커밋으로 만들고 나서 해당 브랜치가 그 커밋을 가리키도록 이동시킨다.
> 그래서 이런 커밋은 부모가 여러 개고 Merge commit 이라고 부른다.



1. signout 브랜치 생성 및 이동

```
$ git checkout -b signout
```

2. 특정 작업 완료 후 commit

```
$ touch signout.txt

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

$ git log --oneline --all
37c8937 (HEAD -> signout) signout test 1
7a79de3 (master) login test 1
5910361 master test 1
```

3. master 브랜치로 이동

```
$ git checkout master
```

4. master 에 추가 작업 후 commit (단 **signout 브랜치와 다른 파일**을 생성 혹은 수정)

```
$ touch master.txt

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

$ git log --oneline --all --graph
* 432f927 (HEAD -> master) master test 2
| * f7404a9 (signout) signout test 1
|/
* 9f69ebb login test 1
* e7512d1 master test 1
```

5. master 에 병합

```
$ git merge signout
```

6. 자동 merge commit 발생
   - 커밋 편집기 화면 등장
   - 자동으로 작성된 커밋 메세지를 확인하고 저장 및 종료

```
$ git merge signout
Merge made by the 'recursive' strategy.
 signout.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 signout.txt
```

7. log 확인

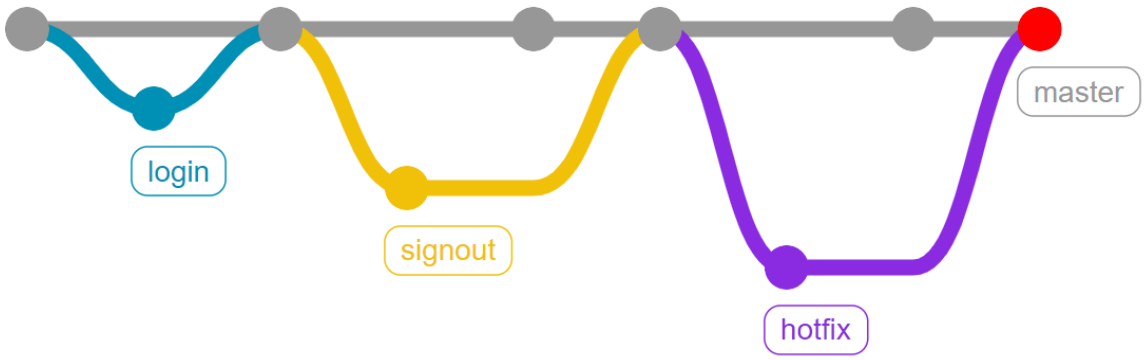
```
$ git log --all --graph --oneline
*   3ae8333 (HEAD -> master) Merge branch 'signout'
|\
| * 37c8937 (signout) signout test 1
* | 5ae736d master test 2
|/
* 7a79de3 login test 1
* 5910361 master test 1
```

8. branch 삭제

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

3. Merge Conflict

> Merge 하는 두 브랜치에서 **같은 파일의 같은 부분을 동시에 수정**하고 Merge 하면
> Git은 해당 부분을 자동으로 Merge 하지 못한다.
> (반면 동일 파일이더라도 서로 다른 부분을 수정했다면, Conflict 없이 자동으로 Merge Commit 된다.)



1. hotfix 브랜치 생성 및 이동

```
$ git checkout -b hotfix
```

2. 특정 작업 완료 후 commit

```
# test.txt 수정

master test 1
이건 hotfix 에서 작성한
문장이에요!!
```

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

$ git log --graph --oneline
* 1a12012 (HEAD -> hotfix) hotfix test 1
*   3ae8333 (master) Merge branch 'signout'
|\
| * 37c8937 signout test 1
* | 5ae736d master test 2
|/
* 7a79de3 login test 1
* 5910361 master test 1
```

3. master 브랜치로 이동

```
$ git checkout master
```

4. 특정 작업(hotfix 와 동일 파일의 동일 부분 수정) 완료 후 commit

```
# text.txt 수정

master test 1
이건 master 에서 작성한
코드에용ㅎㅎ!!
```

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

$ git log --graph --oneline --all
* ac05762 (HEAD -> master) master test 3
| * 1a12012 (hotfix) hotfix test 1
|/
*   3ae8333 Merge branch 'signout'
|\
| * 37c8937 signout test 1
* | 5ae736d master test 2
|/
* 7a79de3 login test 1
* 5910361 master test 1
```

5. master 에 병합

```
$ git merge hotfix
```

6. 결과 → `merge conflict` 발생
7. 충돌 확인 및 해결
    - Merge 충돌이 일어났을 때 Git이 어떤 파일을 Merge 할 수 없었는지 살펴보려면 git status 명령을 이용한다.

```
$ git status
On branch master
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:   test.txt

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

```
master test 1
<<<<<<< HEAD
이건 master 에서 작성한
코드에용ㅎㅎ!!
=======
이건 hotfix 에서 작성한
문장이에요!!
>>>>>>> hotfix
```

    - `=======` 위쪽의 내용은 HEAD 버전(merge 명령을 실행할 때 작업하던 master 브랜치)의 내용이고 아래쪽은 `hotfix` 브랜치의 내용이다. 충돌을 해결하려면 위쪽이나 아래쪽 내용 중에서 고르거나 새로 작성하여 Merge 한다. (`<<<<<<<`, `=======`, `>>>>>>>` 가 포함된 행을 삭제)

```
# test.txt 최종본

master test 1
충돌을
해결해보자!!
```

8. merge commit 진행

```
$ git add .
$ git commit
```

- 이전에 진행했던 커밋 편집기 재등장

```
Merge branch 'hotfix'

# Conflicts:
#       test.txt
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   test.txt
#
```

  - 자동으로 작성된 커밋 메세지를 확인하고 저장 및 종료

```
$ git commit
[master 2aa2b1e] Merge branch 'hotfix'
```

9. log 확인

```
$ git log --all --graph --oneline
*   2aa2b1e (HEAD -> master) Merge branch 'hotfix'
|\
| * 1a12012 (hotfix) hotfix test 1
* | ac05762 master test 3
|/
*   3ae8333 Merge branch 'signout'
|\
| * 37c8937 signout test 1
* | 5ae736d master test 2
|/
* 7a79de3 login test 1
* 5910361 master test 1
```

10. 브랜치 삭제

```
$ git branch -d hotfix
```