오픈소스 소프트웨어 개발 - 13

(Online) Git - branch 1

광운대학교 이윤상
2017년 2학기

이번 시간에 할 것

- Branch 개념 리뷰
- Branch 생성, 전환, 삭제 - git branch, git checkout
- [Tip] git-prompt 설치
- Branch 전환 시 주의 사항
- Branch history 보기 - git log
- Git 내부 동작 방식
- Branch 간의 비교 - git diff

Branch

- 원래 코드에 영향을 주지 않고 독립적으로 개발을 진행할 수 있는 공간
  - 여러 개발자들이 동시에 다양한 작업을 하는 것이 가능
  - 원래 코드의 일종의 '복사본'에서 개발을 진행 - 실제로 복사를 하지 않을 수도 있다.

- 아래와 같이, 공통의 코드로부터 각각 따로 진행되어야 하는 작업들을 위해 사용
  - 실험적인 기능의 구현
  - 급한 버그의 해결
  - 다음 버전 릴리즈 준비 등

Branching

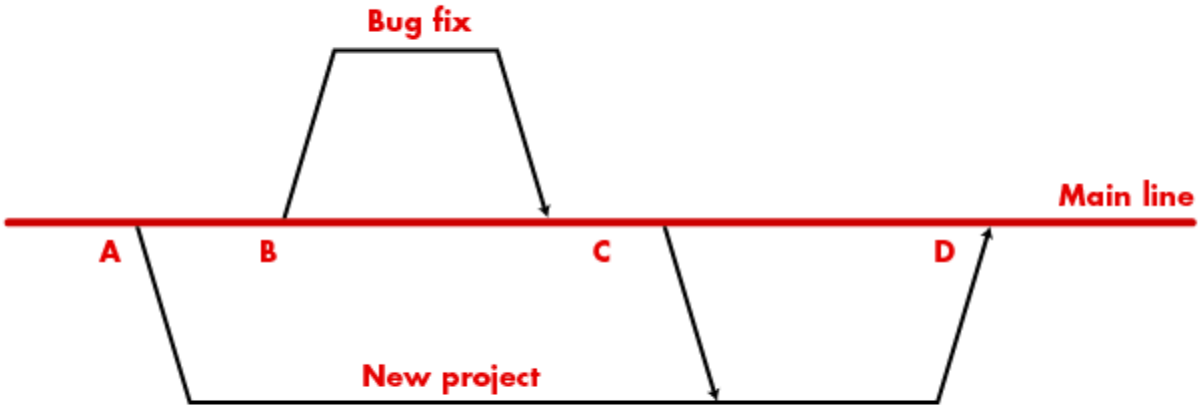


Figure 2

[실습] Branch 생성

- 12-(Online)Git-basic2 강의 마지막 부분에서 .gitignore를 만들어 commit 했던 VectorTest 프로젝트를 계속 이용하여 실습을 진행해보자.

[실습] Branch 생성

- **git branch <branchname>** : 새로운 branch를 생성
- **git branch** : 모든 branch 목록 출력
  - * 표시가 붙은 것이 현재 branch

- **master**라는 기본 branch가 자동으로 생성되어 있음.

(Shell)

```
# 아무 파일이나 수정해서 한 번 더 commit을 하자.
vi main.cpp
git add .
git commit

# 실험적인 기능의 개발을 위한 'experimental' branch 생성
git branch experimental
git branch    # 전체 branch 목록 확인
```

[실습] Branch 전환, 삭제

- **git checkout** <branchname> : 다른 branch로 전환
- **git checkout -b** <branchname> : 새로운 branch를 생성하고 바로 전환
- **git branch -d** <branchname> : branch 삭제

(Shell)

```
git checkout experimental
git branch    # 현재 branch가 experimental로 바뀐 것 확인

git checkout -b testing
git branch  # testing branch가 생성되고 현재 branch인 것 확인

git checkout master  # testing branch 삭제 위해 master로 전환
# (현재 branch는 삭제하지 못함)
git branch -d testing
git branch    # testing branch가 삭제된 것 확인
```

[Tip] git-aware-prompt 설치

- Shell prompt에 현재 git branch와 변경사항 발생여부를 표시해 주는 프로그램들이 있다

```
jim@torment ~$ cd .bash
jim@torment ~/.bash (master)$ git checkout develop
Switched to branch 'develop'
jim@torment ~/.bash (develop)$
```

- git-aware-prompt
  - https://github.com/jimeh/git-aware-prompt
  - 위 링크에 나와 있는 설치 방법을 다음 페이지에 요약함.

[Tip] git-aware-prompt 설치

(Shell)

```
mkdir ~/.bash
cd ~/.bash
git clone git://github.com/jimeh/git-aware-prompt.git

vi ~/.bashrc    # bash 설정파일 수정

(아래 내용을 ~/.bashrc 파일 마지막에 추가)
export GITAWAREPROMPT=~/.bash/git-aware-prompt
source "${GITAWAREPROMPT}/main.sh"
export
PS1="\${debian_chroot:+(\$debian_chroot)}\[\033[01;32m\]\u@\h
\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]
\[$txtcyn\]\$git_branch\[$txtred\]\$git_dirty\[$txtrst\]\$ "

# ~/.bashrc를 reload
source ~/.bashrc
```

[실습] 새 branch 변경 사항 발생 후 확인

(Shell)

```
git checkout experimental
vi file1.txt    # Add 'hello'
vi main.cpp    # 기존 파일 변경
git add .
git commit

# 두 branch의 working directory와 commit history 비교
# 탐색기 띄워놓은 채로 살펴봐도 좋음
git checkout master
ls -al
git log
git checkout experimental
ls -al
git log
```

- 이와 같이 각각의 branch에 서로 다른 commit을 쌓아갈 수 있다.

Branch 전환 시 주의사항

- 아직 commit하지 않은 변경 내용이 **전환할 branch와 충돌이 나는 경우,** branch를 전환할 수 없다.

- 이 경우 변경 내용을 commit한 후 전환해야 한다.

- (혹은 나중에 배울 git stash 사용 후 전환)

[실습] Branch 전환 시 주의사항

(Shell)

```
git checkout master
vi file1.txt  # master에도 file1.txt 추가. Add 'world'.

git checkout experimental
# experimental branch에 file1.txt가 이미 존재하기 때문에 아래와 같은 에러메시지가 출력
error: The following untracked working tree files would be overwritten by checkout:
      file1.txt
Please move or remove them before you can switch branches.
Aborting

git add .
git commit
git checkout experimental
# file1.txt를 master에 commit 하고 나면 experimental로 checkout 가능
```

[실습] Branch 전환 시 주의사항

- 아직 commit하지 않은 변경 내용이 **전환할 branch와 충돌이 나지 않는 경우,** 전환된 branch에서 해당 변경 작업이 진행되고 있는 상태가 된다.

(Shell)

```
git checkout master
vi file2.txt    # file2.txt라는 파일은 experimental에 없다
git status

git checkout experimental
git status    # file2.txt가 experimental에서 untracked로 표시
```

[실습] Branch History 보기

- git log --branches --decorate --graph --oneline
  - --branches : 모든 branch의 commit 보여줌
  - --decorate : 각 branch의 history상 현재 위치 표시
  - --graph : branch merge history를 보여주는 graph 표시
  - --oneline : commit을 한 줄로 보여줌

(Shell)

```
git checkout experimental
git log --branches --decorate --graph --oneline

git checkout master
git log --branches --decorate --graph --oneline
```

[실습] Branch History 보기

```
yoonsang@yoonsang-VirtualBox:~/test/git-2 (master)$ git checkout experimental
Switched to branch 'experimental'
yoonsang@yoonsang-VirtualBox:~/test/git-2 (experimental)$ git log --branches --d
ecorate --graph --oneline
* 286a06c (HEAD -> experimental) Add file1.txt
* 1a6ad09 (master) Add v3
* 3851686 Initial commit
yoonsang@yoonsang-VirtualBox:~/test/git-2 (experimental)$ git checkout master
Switched to branch 'master'
yoonsang@yoonsang-VirtualBox:~/test/git-2 (master)$ git log --branches --decorat
e --graph --oneline
* 286a06c (experimental) Add file1.txt
* 1a6ad09 (HEAD -> master) Add v3
* 3851686 Initial commit
yoonsang@yoonsang-VirtualBox:~/test/git-2 (master)$
```

- HEAD는 현재 branch를 가리키는 "포인터"라고 생각하자.

[실습] Branch History 보기

- master branch에 commit을 추가한 후 history가 어떻게 달라졌는지 확인해보자.

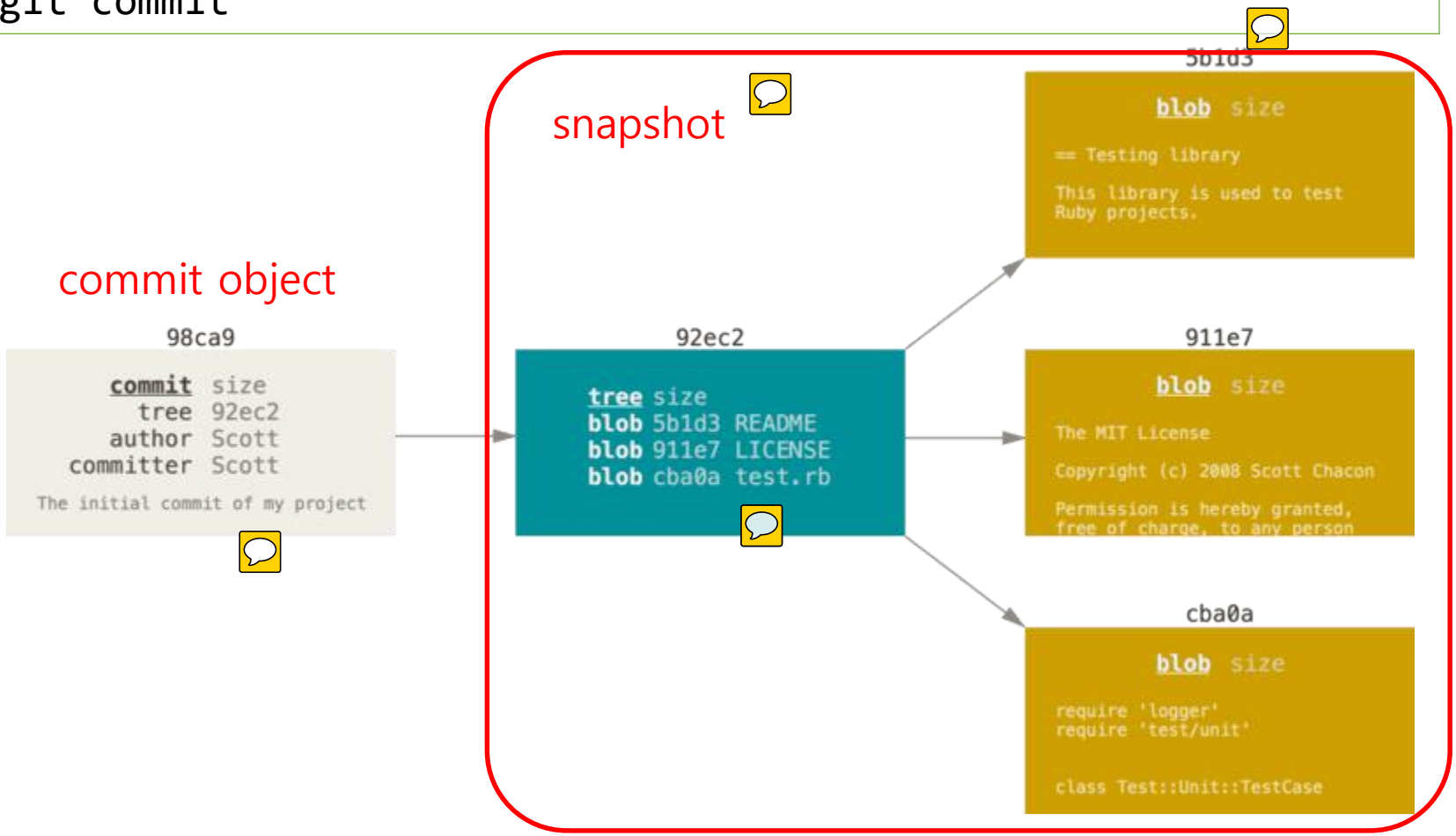
(Shell)

```
git checkout master
vi vector.h
git add .
git commit
git log --branches --decorate --graph --oneline
```

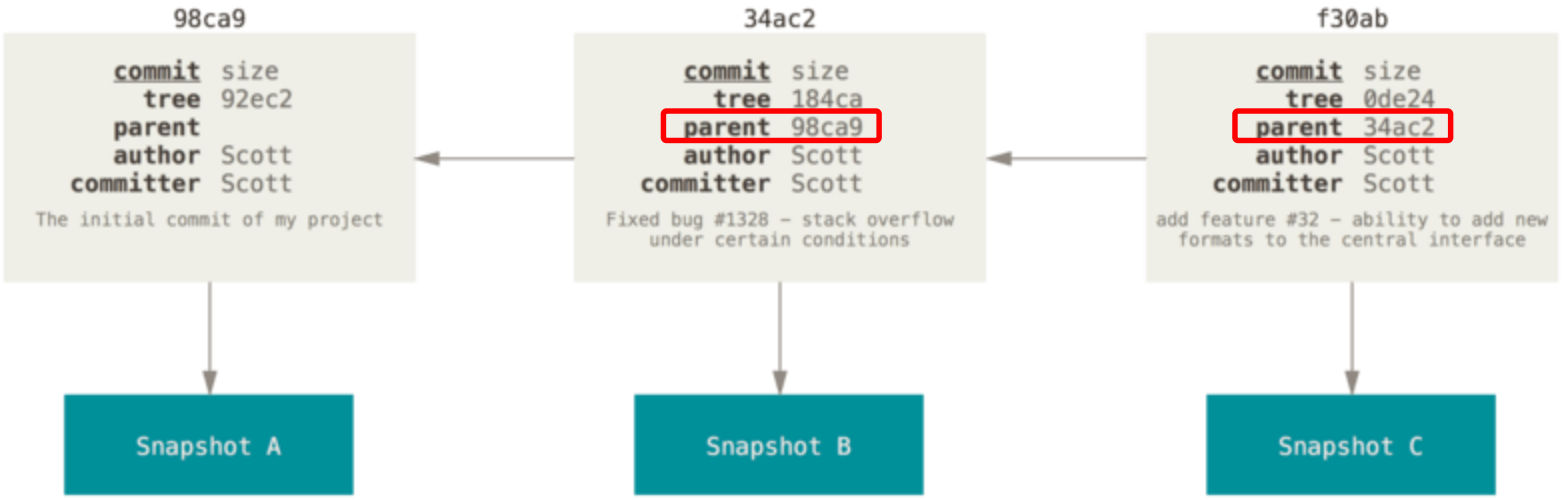
Git 내부 동작 방식

(Shell)

```
git add README test.rb LICENSE
git commit
```

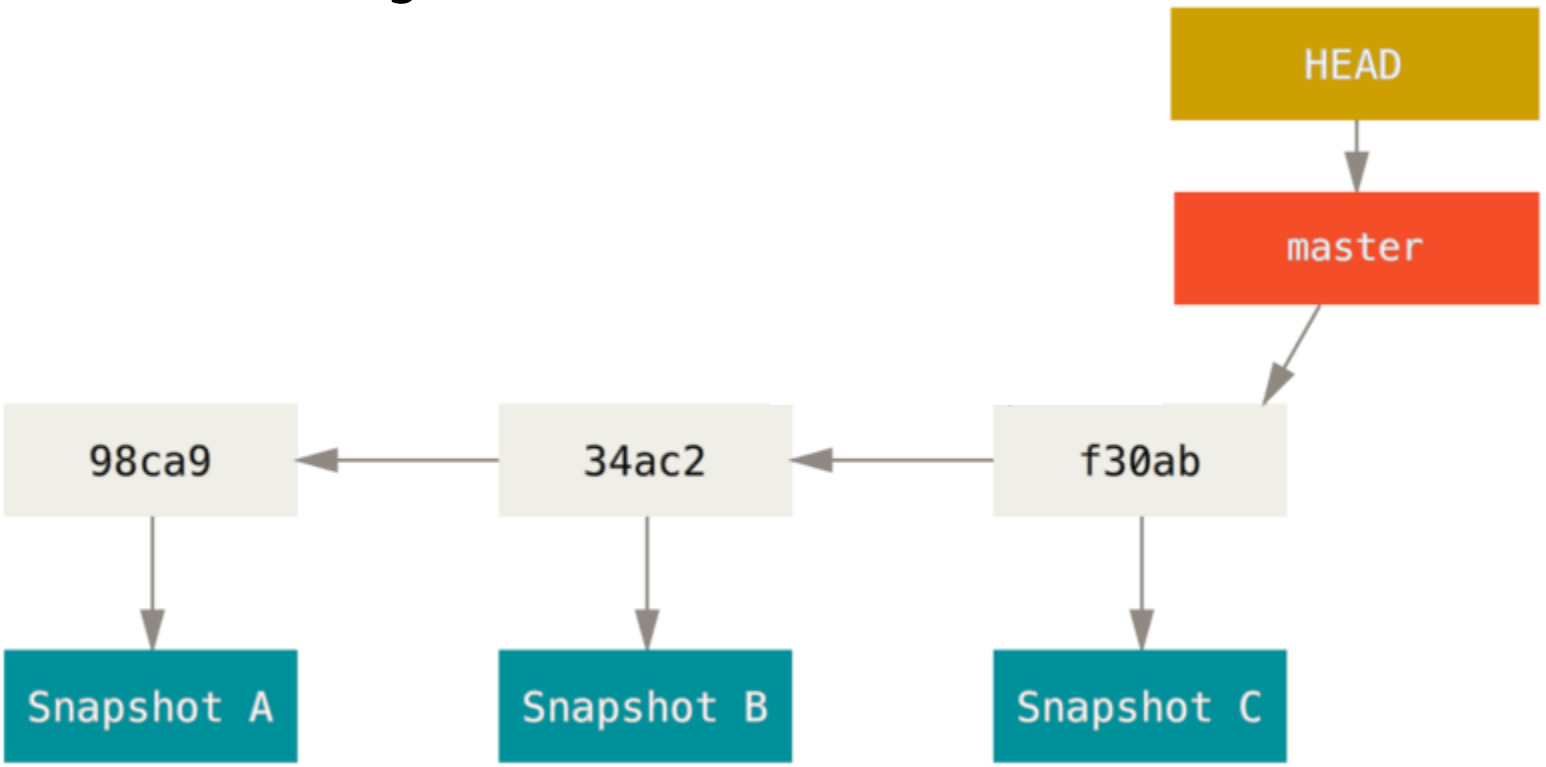


- Commit들은 이전 commit (부모 commit)을 가리키며 저장됨
- (.git 디렉터리 내부에 저장)



- **Branch**는 branch에 담긴 commit들 중 가장 마지막 commit을 가리키는 포인터이다.
- **HEAD**는 현재 branch를 가리키는 포인터이다.

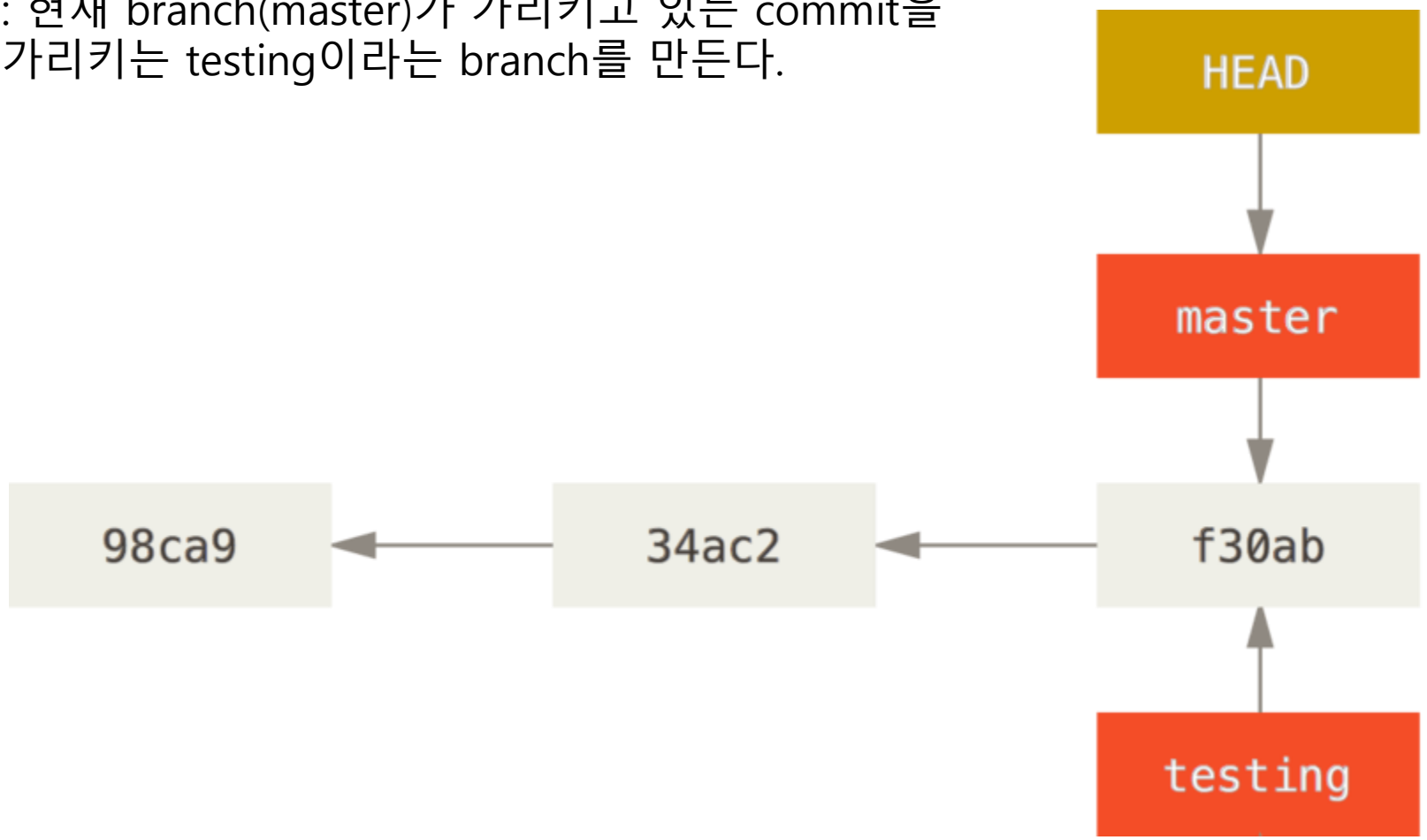
- (여기서 말하는 "포인터"는 실제로는 다른 object의 sha1 id를 저장하고 있는 .git 디렉터리 내부의 파일이다)



(Shell)

```
git branch testing
```

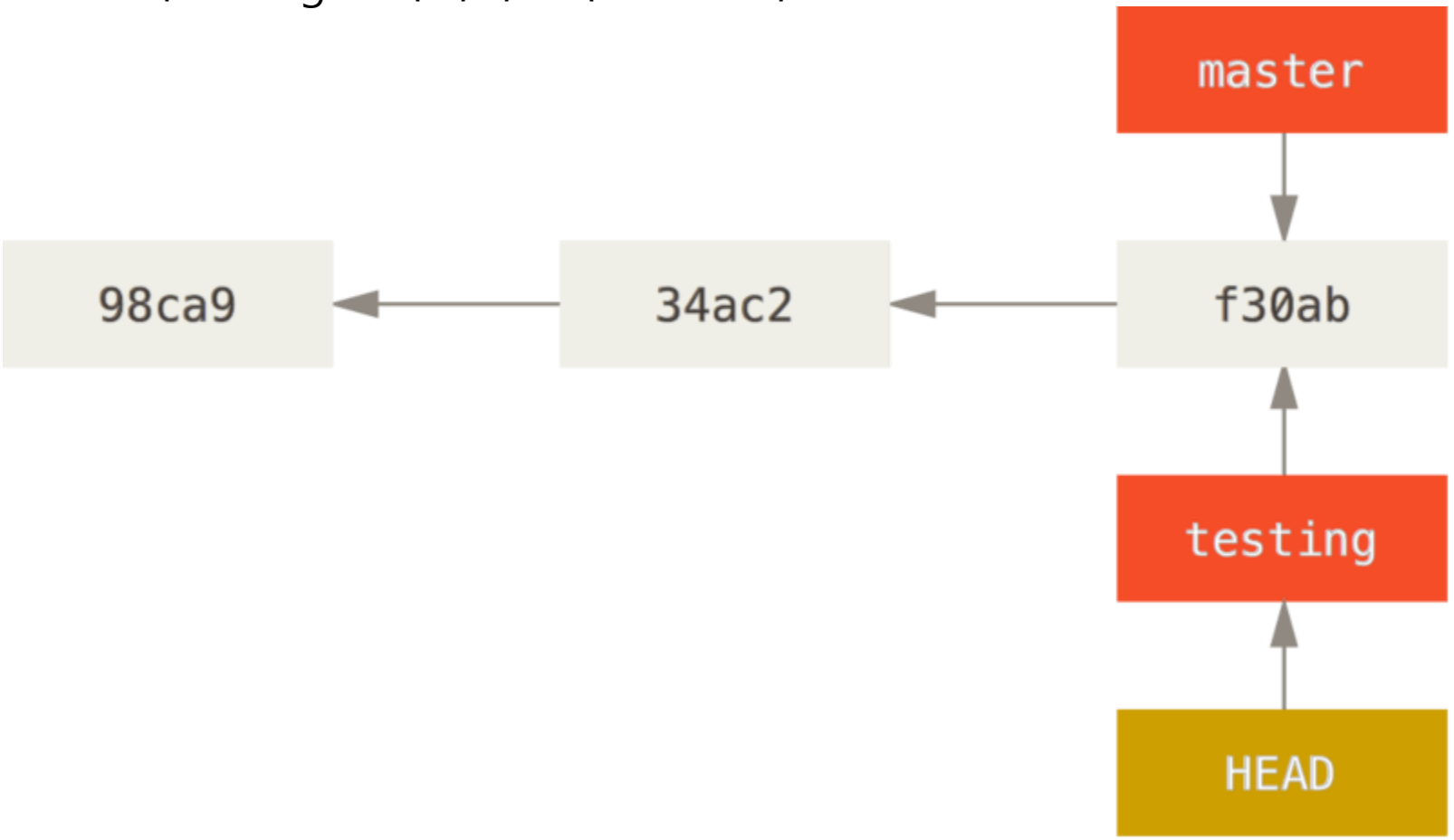
: 현재 branch(master)가 가리키고 있는 commit을 가리키는 testing이라는 branch를 만든다.



(Shell)

```
git checkout testing
```

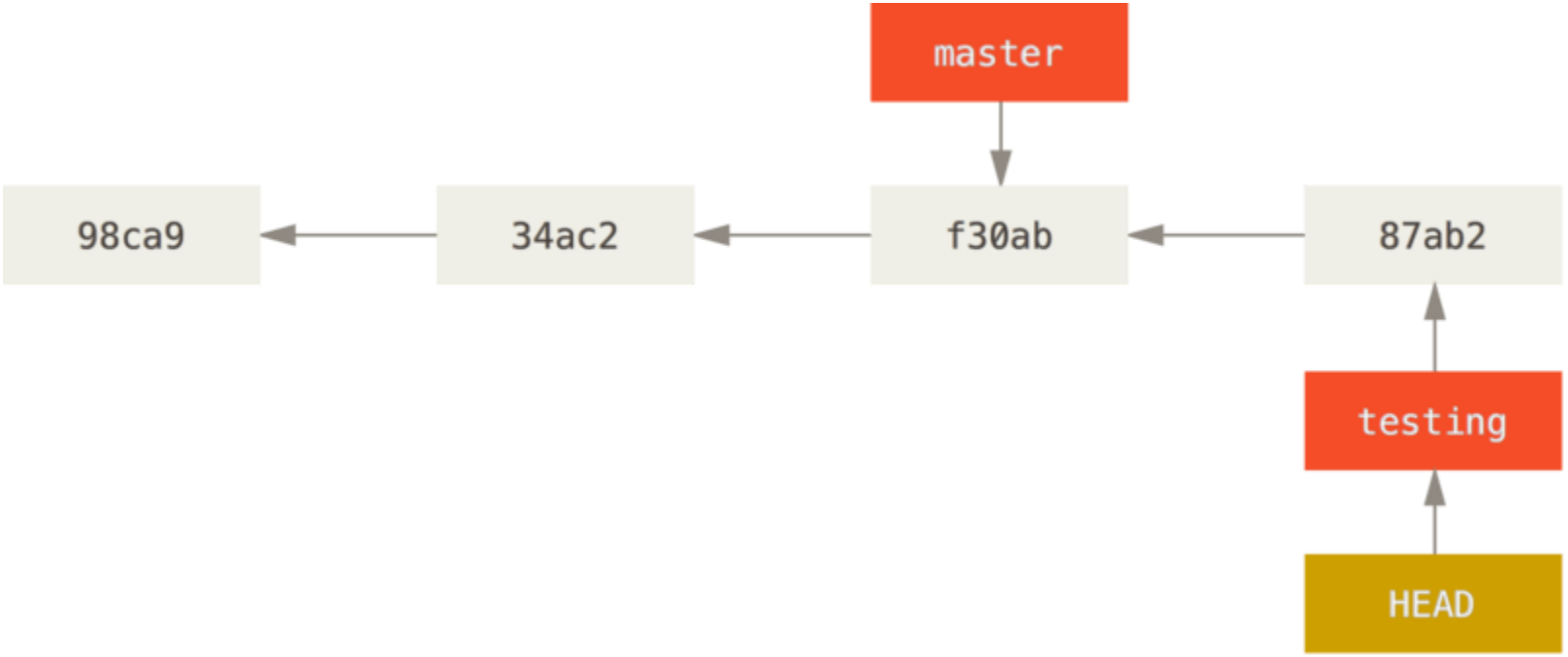
: HEAD가 testing을 가리키도록 변경한다.



(Shell)

```
vim test.rb
git commit
```

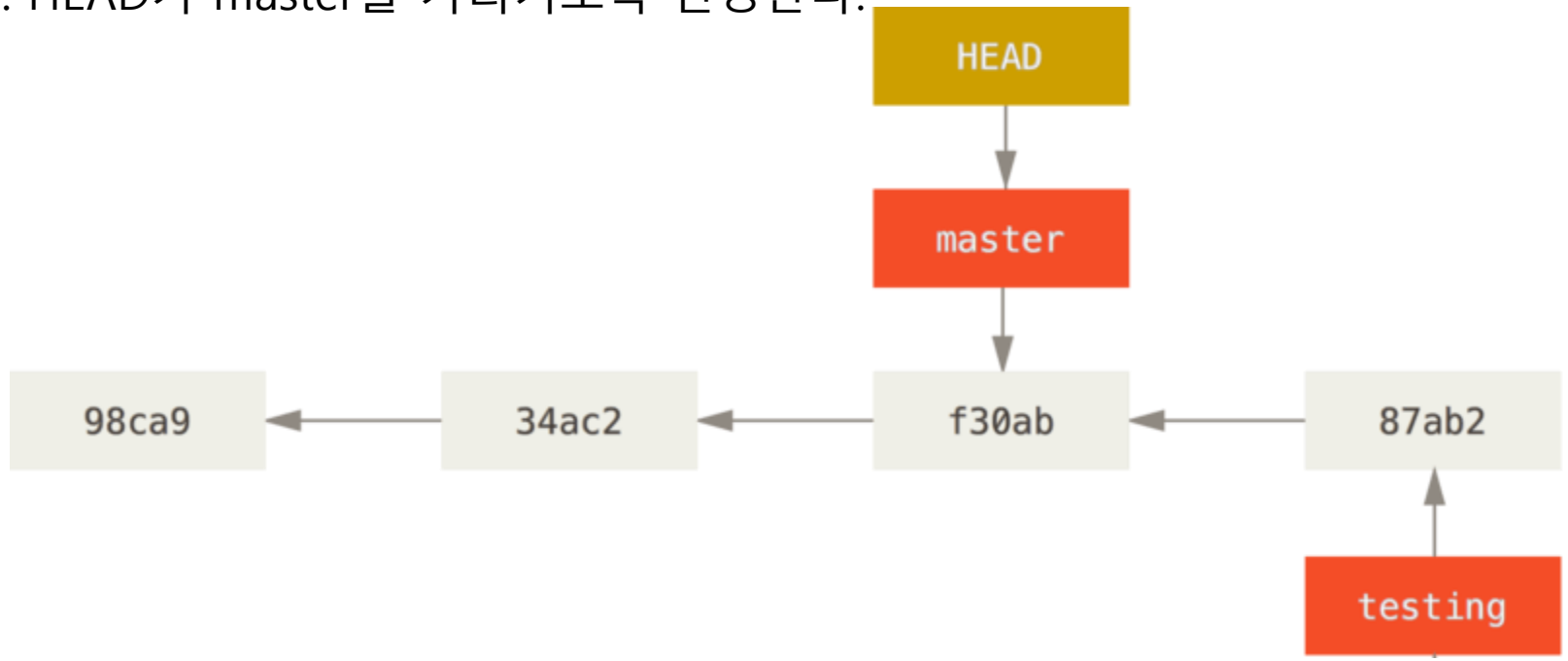
: 새로운 commit을 추가하고 HEAD가 가리키고 있는 branch (testing)가 새 commit을 가리키도록 변경한다. 바로 이전까지 testing이 가리키고 있던 commit을 새 commit이 부모로써 가리키게 된다.



(Shell)

```
git checkout master
```

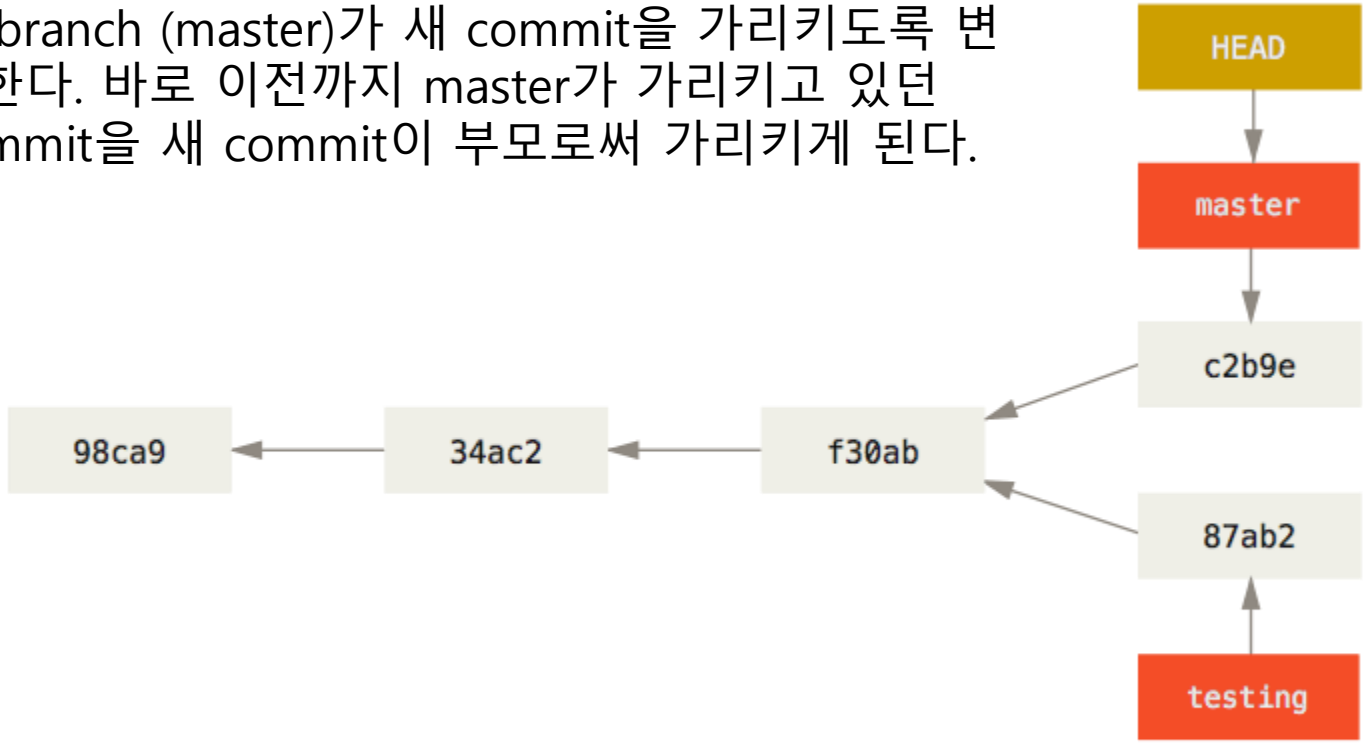
: HEAD가 master를 가리키도록 변경한다.



(Shell)

```
vim test.rb
git commit
```

: 새로운 commit을 추가하고 HEAD가 가리키고 있는 branch (master)가 새 commit을 가리키도록 변경한다. 바로 이전까지 master가 가리키고 있던 commit을 새 commit이 부모로써 가리키게 된다.



[실습] Branch 간의 비교

- **git log** branch1**..**branch2 : branch1에는 없고 branch2에는 있는 commit을 보여줌

- **git diff** branch1**..**branch2 : branch1과 branch2의 차이점을 보여줌

(Shell)

```
git log master..experimental
git log experimental..master
git diff master..experimental
```