3-5. 브랜치 관리하기

**깃과 branch를 더 편하게 관리하기 위한 몇 가지 명령을 살펴보자.**

**\* branch에서 checkout과 reset의 작동원리**

2장에서 checkout과 reset 명령에 대해 공부했다. 하지만 **branch와 함께 사용하면 더 다양하게 사용할 수 있다.** 먼저 HEAD와 branch의 개념부터 간단하게 알아보자.

1. 홈 디렉터리로 이동한 후 **test라는 깃 저장소를 만들고 test 디렉터리로 이동**한다.

2. 빔에서 **c1.txt 파일을 만들고 숫자 ‘1’을 입력하고 저장**한다. 그리고 방금 만든 c1.txt를 stage에 올리고 **commit (commit message: c1)**한다.

3. **git log 명령**을 실행해 보자.

텍스트, 시계, 시계겸용라디오이(가) 표시된 사진

자동 생성된 설명

commit log 첫 번째 줄에 **(HEAD -> master) 표시**가 있다. 여기에서 **HEAD**는 **현재 Working Tree가 어떤 버전을 기반으로 작업 중인지를 가리키는 포인터**이다. HEAD는 **기본적으로 master branch를 가리킨다.** 그리고 **branch는 기본적으로 branch에 담긴 commit 중에서 가장 최근의 commit을 가리킨다.** 예를 들어 repository에 c1 commit을 만들면 **HEAD는 master branch를 가리키고 master branch는 c1 commit을 가리킨다.**

4. **sub라는 branch를 만들어보자.** **sub branch 역시 c1 commit을 가리킨다.**



5. master branch에서 c2.txt 파일을 만들고 숫자 ‘2’를 입력한 후 저장하자. 그리고 파일을 stage에 올린 후 **commit(commit message: c2)을 수행**하자. 이제 **master branch는 c2 commit을 가리키고 HEAD는 그대로 master branch를 가리킨다.**

6. 이제 sub branch에서 commit을 만들어보자. **git checkout 명령**을 이용하면 sub branch로 이동할 수 있다. 해당 명령을 실행하면 **HEAD는 sub branch를 가리키게 된다.**

텍스트이(가) 표시된 사진

자동 생성된 설명

7. s1.txt 문서를 만들고 ‘s1’이라고 입력하고 저장하자. 그리고 stage에 올린 후 **commit(commit message: s1)을 수행**하자. 이제 **sub는 s1 commit을 가리킨다.**

8. branch가 여러 개 일 때 reset 명령은 어떻게 사용할 수 있을까? 2장에서는 reset 명령으로 master branch에 있던 여러 commit 중 하나를 골라서 되돌아갔다. **branch가 여러 개일때는 현재 branch가 아닌 다른 branch에 있는 commit을 골라서 최신 commit으로 지정할 수 있다.** 예를 들어 sub branch에 있는 상태에서 master branch에 있는 c2 commit을 sub branch의 최신 commit으로 지정할 수 있다. 먼저 **git log 명령을 사용해서 c2 commit의 commit hash를 확인**하자.

텍스트이(가) 표시된 사진

자동 생성된 설명

9. **git reset 명령 다음에 c2 commit의 commit hash를 입력**한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

10. **git log를 사용해 결과를 확인**해보자.

텍스트이(가) 표시된 사진

자동 생성된 설명

**sub branch의 최신 commit이 master branch의 최신 commit인 c2로 바뀌었다. 그리고 HEAD는 그대로 sub branch를 가리킨다.** 이렇게 **git reset 명령을 사용하면 현재 branch가 가리키는 commit을 여러 branch 사이를 넘나들면서 제어할 수 있다.** 단, sub branch는 c2 commit을 가리키고 있기 때문에 **원래 가리키던 s1 commit은 연결이 끊기면서 삭제된다.**

정리해보자. **git checkout 명령**을 사용하면 **HEAD를 제어해서 branch를 이동할 수 있다.** **git reset 명령**을 사용하면 **HEAD가 가리키고 있는 branch의 최신 commit을 원하는 commit으로 지정할 수 있다.** **이때 어떤 branch에 있는 commit이든 지정할 수 있고, 명령을 수행한 뒤 branch와 연결이 끊긴 commit은 삭제된다.**

**\* 수정 중인 파일 감추기 및 되돌리기 - git stash(stash: 넣어놓다, 숨기다)**

**branch에서 파일을 수정하고 commit하지 않은 상태에서 급하게 다른 파일을 commit해야할 경**우가 있다. 아직 commit하지 않은 파일들을 그냥 두어도 상관없지만 계속 commit하라는 메시지가 나타나기 때문에 번거롭다. 실수로 다른 파일과 함께 commit이 될 수도 있다. **이럴 때 아직 commit하지 않고 작업 중인 파일들을 잠시 감춰둘 수 있다.** 그리고 당장 필요한 작업들을 끝낸 후 다시 감춰둔 파일들을 꺼내면 된다.

1. 홈 디렉터리로 이동한 후 **st라는 저장소를 만들고 st 디렉터리로 이동**한다.

2. **git stash 명령을 사용하려면 파일이 tracked 상태여야 한다.** 먼저 빔을 사용해 **f1.txt 파일을 작성하고 commit(commit message: f1)하자.**

3. **f2.txt 파일을 만들고 commit(commit message: f2)하자.**

4. **f1.txt, f2.txt 파일 모두 아무렇게나 수정하고 저장**하자.

텍스트, 시계이(가) 표시된 사진

자동 생성된 설명

5. **git status 명령**을 실행해 보면 branch에서 f1.txt와 f2.txt가 수정되었다고 나타난다.

텍스트이(가) 표시된 사진

자동 생성된 설명

6. 이제 **f1.txt 파일과 f2.txt 파일을 commit하기 전에 다른 파일을 수정해야 한다고 가정해 보자.** **commit하지 않은 수정 내용을 어디에 보관하려면 git stash 명령을 사용한다.** git stash save 또는 간단히 git stash라고만 해도 된다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**git status**를 사용해 깃 상태를 확인해보면 이전에 나오던 **modified 메시지가 사라진 것을 확인**할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

같은 방법으로 여러 파일을 수정한 후 따로 보관할 수 있으며, 이렇게 감춘 파일들은 stash 목록에서 확인할 수 있다. **stash 목록은 git stash list 명령으로 확인할 수 있다.**

텍스트, 시계, 장치이(가) 표시된 사진

자동 생성된 설명

가장 먼저 감춘 것은 stash@{0}에 들어있다. 앞으로 다른 파일이 추가되면 기존 파일은 stash@{1}로 옮겨지고 새로 추가된 파일은 stash@{0}에 담긴다. 즉, 가장 최근에 보관된 것이 stash@{0}에 담긴다.

7. 급한 작업을 모두 마쳤다면 감춰둔 파일을 꺼내 계속 수정하거나 commit할 수 있다. **git stash 명령 뒤에 pop을 추가**하면 **stash 목록에서 가장 최근 항목을 출력**해준다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**(참고) stash apply와 stash drop**

stash 목록에 저장된 수정 내용을 나중에 또 사용할지도 모른다면 **git stash apply 명령**을 사용한다. **stash 목록에서 가장 최근 항목을 되돌리지만 저장했던 내용은 그대로 남겨둔다.**

**git stash drop 명령**은 **stash 목록에서 가장 최근 항목을 삭제**한다.