소프트웨어공학 git 과제\_60212568 이현정

1. VCS (Version Control System) 이란 무엇인가?

버전관리 시스템(VCS, Version Control System)이란 파일의 변화를 추적하고 관리하는 시스템이다. VSC는 파일 변화를 시간에 따라 기록해, 나중에 특정 시점의 버전을 다시 불러올 수 있게 만들어준다. VSC를 활용하면 선택한 파일을 이전 상태로 되돌릴 수 있고, 변경 사항을 비교하고, 변경한 사람 및 변경시기를 추적할 수 있다. 또한, 파일을 잃어버리거나 잘못 고쳤을 때도 쉽게 복구할 수 있다.

1. GIT 이 SVN 과 같은 중앙집중식 VCS 와 가장 큰 차이는 무엇인가?

Git은 저장소(repository)의 모든 버전 기록을 로컬에 복제(clone)해 작업하기 때문에 모든 작업을 로컬에서 수행할 수 있다. 하지만, SUV는 중앙 서버에만 저장소가 존재하므로 로컬에서는 해장 저장소의 파일을 체크아웃(checkout)해 작업해야 한다.

Git의 분산 모델은 협업 작업에서 매우 효과적이다. 모든 작업자가 로컬에서 독립적으로 작업할 수 있으며, 중앙 서버에 대한 의존도가 낮아지므로, 네트워크 연결이 끊어졌을 때에도 작업이 가능하기 때문이다.

또한 Git은 브랜치(branch)와 머지(merge)가 매우 쉽고 빠르게 이루어집니다. 브랜치는 독립적인 작업 영역을 생성하고, 머지는 여러 개발자가 동시에 작업한 내용을 통합하는 과정입니다. Git은 브랜치와 머지를 매우 쉽게 수행할 수 있으므로, 여러 개발자가 동시에 작업할 때에도 매우 효과적이다. 반면, SVN 같은 중앙집중식 VCS는 중앙 서버에 의존하므로, 네트워크 연결이 끊어지면 작업이 불가능하다.

1. 파일의 변경 사항을 저장소에 반영할 때 git add 와 git add –p 의 차이는 무엇인가?

git add 명령은 변경된 모든 파일을 새로운 스냅샷으로 저장소에 추가한다. 즉, 변경된 모든 파일의 모든 내용이 스테이징 영역으로 이동한다. 이후 git commit 명령을 사용하여 이 변경 내용을 저장소에 커밋할 수 있다. git add 명령은 모든 변경 내용을 한 번에 스테이징 영역으로 이동시키므로, 파일을 수정 후에는 모든 내용에 변경사항이 적용된다.

반면, git add -p 명령은 변경된 파일의 일부분만을 스테이징 영역으로 이동시킨다. -p 옵션이파일의 변경 내용을 세분화하고, 변경 내용을 수정하는 위치를 직접 선택할 수 있는 인터랙티브 모드를 제공하는 것이다. 즉, 이 명령을 사용하면 변경 내용의 일부만을 스테이징 영역으로 이동시킬 수 있으므로, 파일의 일부분만 커밋하고자 할 때 유용하다.

1. 다음 페이지의 Git lifecycle 을 참고해서 다음 케이스들에서 file1 이 어떤 state 인지 기술하라. (참고: git status 로 확인 가능하다.)
   1. 저장소에 없는 file1 이라는 파일을 메모장으로 만든 직후
      * 메인(main) branch 에 있는 걸로 나오고 commit기록은 아직 없다고 나온다-
      * On branch main

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

file1.rtf

* 1. git add 와 git commit 으로 file1 을 저장소에 추가한 직후
     + commit 할 항목이 없다고 나온다.
     + On branch main

nothing to commit, working tree clean

* 1. file의 내용을 수정한 직후
     + file1이 수정되었고 commit할 준비가 되지 않았다고 나온다.
     + 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: file1.rtf

* 1. 수정한 file1을 git add 또는 git add –p 한 직후
     + file1이 수정되었고 commit할 변경사항이 있다고 나온다
     + On branch main

Changes to be committed:

(use "git restore --staged <file>..." to unstage)

modified: file1.rtf

* 1. git commit 명령을 한 직후
     + commit 할 항목이 없다고 나온다
     + On branch main

nothing to commit, working tree clean.

1. Step #6 에서 메모장으로 확인한 file1 의 내용은 무엇인가?

-> “Hello World” 와 “What a wonderful world” 는 file1에 기록되어 있지만, “Great weather” 은 확인되지 않는다

1. Step #7 에서 file2 는 존재하는가? 그 내용은 무엇인가? 만일 존재하지 않는다면 왜 존재하지 않는지 기술하라.

-> 존재하지 않는다. branch2와 master 는 다른 branch로 각각의 개별된 다른 공간을 가지기 때문이다.

1. git show-branch 명령은 변경 사항에 대한 commit graph 를 보여준다.  
   Step #1-#7 까지 작업한 뒤의 commit graph 는 어떻게 되는가?

-> \* [branch2] branch2 frist commit

! [main] thrid commit

--

+ [main] thrid commit

\* [branch2] branch2 frist commit

\*+ [main^] second commit

1. Step #8 로 생성한 github 상의 저장소 주소는 무엇인가?
   * 참고: 교수가 이 URL 에서 작업한 git 내용과 보고서를 가지고 채점할 예정임

<https://github.com/mintchococ/softwareHomework.git>

1. git 에서 다른 branch 의 내용을 반영하는 방식에는 merge 와 rebase 가 있다.  
   두 방식의 차이에 대해서 조사해서 설명하라.

merge는 현재 브랜치와 다른 브랜치를 병합하여 새로운 커밋을 만드는 방법이다. 즉, 두 개의 독립적인 브랜치에서 작업한 내용을 합치는 것이다. merge는 새로운 병합 커밋을 만들어서 두 브랜치의 작업 내용을 합치고, 이 커밋을 현재 브랜치에 추가한다. 병합 커밋은 두 브랜치를 부모로 가진다고 볼 수 있다. Merge는 간단하게 사용할 수 있으며, 충돌을 해결하기가 쉽다는 장점이 있다. 하지만, 브랜치 히스토리가 깔끔하지 않을 수 있다는 단점이 존재한다. 여러 브랜치가 동시에 변경되었을 때, 병합 커밋이 많아지면서 브랜치 히스토리가 지저분해질 가능성이 높기 때문이다.

rebase는 현재 브랜치를 다른 브랜치의 끝으로 놓이게 하는 방법이다. 이때 다른 브랜치의 변경 내용을 현재 브랜치에 적용하고, 이후에 현재 브랜치의 커밋을 차례로 적용한다. 그렇기 때문에 . rebase는 커밋 히스토리가 깔끔하다는 장점이 있다. 하지만, 충돌이 발생할 가능성이 높다는 단점이 존재한다.

결론적으로, merge는 브랜치 히스토리를 보존하면서 브랜치를 합치고, rebase는 브랜치 히스토리를 깨끗하게 유지하면서 브랜치를 합친다는 차이가 있다.

1. git cherry-pick 이 어떤 일을 하는 명령인지 조사해서 설명하라.

git cherry-pick 명령은 다른 브랜치에 있는 커밋을 선택적으로, 현재 브랜치에 적용할 떼 사용되는 명령어다. 즉, 다른 브랜치에서 필요한 변경 사항이 있는데, 해당 커밋만 현 재 브랜치에 반영하고 싶을 때 사용한다.

1. git rebase -i 가 어떤 일을 하는 명령인지 조사하라. 직전 두 커밋의 순서를 바꾸고자 할 때 git rebase -i 를 써서 해결할 수 있다. 그 방법을 설명하라.

git rebase -i 명령은 대화형 모드로 rebase를 실행시킨다. git rebase -i 명령을 사용하여 커밋 의 순서를 변경하거나 기존의 커밋을 수정 및 삭제할 수 있다. git rebase -i 명령은 대화 형 에디터를 실행시킨다. 에디터에는 현재 브랜치의 커밋 목록이 나타나는데, 이 목록을 이용해 커밋을 수정하는 등의 작업을 수행한다. 변경된 커밋은 새로운 커밋으로 생성되 며, 기존의 커밋은 삭제처리 된다.

git rebase -I 명령을 이용해 직전 두 커밋의 순서를 바꾸는 방법은 “git rebase -i HEAD~2” 를 통해 구현할 수 있다. “git rebase -i HEAD~2”를 실행하면, 대화형 모드로 rebase를 실행 할 수 있는 에디터가 열린다. 이 에디터에서 순서를 변경하고자 하는 두 커밋의 위치를 변경한 뒤 저장하면 된다. 조금 더 자세히 설명하자면, ‘pick’ 명령 앞에 있는 두 커밋의 순서를 바꾸기 위해, 첫 번째 커밋의 pick 명령을 두 번째 커밋의 위로 옮기고 저장하는 것이다. 변경사항이 저장되면, 새로운 커밋이 생성되고 순서가 변경된 커밋 히스토리가 생성된다.