1. VCS (Version Control System) 이란 무엇인가?

VCS(버전 관리 시스템)은 소스코드 및 파일의 추가 및 수정과 같은 변경 내용을 추적하고 관리하는 소프트웨어 도구이다. 개발자는 프로젝트 협업 과정에서 코드의 변화를 추적하고 관리하기 위해 많이 사용하고 있다.

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

SVN(Subversion)은 이전 버전 관리 도구 CVS의 단점을 보완하고자 나온 2000년대에 자주 사용하던 프로그램이다. SVN은 소스의 저장소를 서버로 두고 작업하는 PC를 클라이언트로 연결하여 소스코드의 변경사항을 관리한다. 보통 대부분의 기능을 완성한뒤 소스코드를 중앙 저장소에 커밋하여 사용하는 중앙집중식 버전관리도구였다. SVN은 중앙집중형식의 버전관리 도구였기 때문에 개발자마다 자신의 버전관리를 할 수 없다. 또한 자신의 소스코드에 오류가 있을시 전체 코드에 영향이 미치게 된다는 문제가 있다.

Git은 SVN의 중앙집중식 버전 관리와 달리 분산형 버전 관리 시스템이다. 분산형 버전 관리가 가능해졌기에 개발가는 자신만의 버전 관리가 가능해졌고, 개발자가 서버 저장소와 독립적으로 관리 할 수가 있다. 또한 한명의 소스코드에 오류가 있더라도 코드전체에 바로 영향을 미치지 않는다. 이에 더해 소스코드 사본을 로컬 저장소에 저장되어 관리되기 때문에 서버에 데이터를 받아와야했던 SVN보다 속도가 훨씬 빠른 속도로 작업이 가능하다.

앞선 내용을 정리하자면, SVN은 중앙집중식 VCS이며 GIT은 분산형 VCS이며, SVN은 중앙 서버에 소스코드와 히스토리를 저장하지만, GIT은 소스코드와 히스토리를 개인 로컬 저장소에 분산하여 저장한다.

3. 파일의 변경 사항을 저장소에 반영할 때 git add 와 git add -p 의 차이는 무엇인가?

일반적으로 git add는 변경된 파일 전체를 stage에 저장한다. 하지만 git add -p는 변경된 파일에 일부분만 선택하여 stage에 추가할 수 있는 명령어이다. git add -p를 사용시 변경사항들이 Hunk라는 작은 단위로 나누어지며 이 Hunk를 stage에 추가할지 수정할지 선택한다. git add -p를 사용하면 변경사항을 세분화하여 staging 할 수 있기에 깔끔하고 세밀한 버전 관리가 가능하다.

4. 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
새롭게 파일을 생성하면 Untracked 상태로 git이 아직 관리하지 않는 "비 추적 상태"이다.

(2) git add 와 git commit 으로 file1 을 저장소에 추가한 직후
git add과 동시에 file1은 Traked(추적 상태)로 전환된다. Tracked된 file1은 git add시 file1은 Staged 상태로 전환되며 git commit시 Unmodifed 상태로 전환된다.

(3) file의 내용을 수정한 직후
file1 수정시 Unmodifed 상태였던 file1은 Modified 상태로 전환된다.

(4) 수정한 file1을 git add 또는 git add -p 한 직후
Modified 상태였던 file1은 Staged 상태로 전환된다.

(5) git commit 명령을 한 직후
Staged상태에서 Unmodified로 전환된다.

5.Step #6 에서 메모장으로 확인한 file1 의 내용은 무엇인가?
Hello World
What a wonderful world

6. Step #7 에서 file2 는 존재하는가? 그 내용은 무엇인가? 만일 존재하지 않는다면 왜 존재하지 않는지 기술하라.
존재하지 않는다. branch간에 작업내용은 공유되지 않는다.
따라서 file2 는 master branch 에서 작업한 내용이기 때문에 branch2에서는 file2 작성한 내용이 없기에 file2가 존재하지 않는다.

7.git show-branch 명령은 변경 사항에 대한 commit graph 를 보여준다.
Step#1-#7 까지 작업한 뒤의 commit graph 는 어떻게 되는가?
```
! [branch2] branch commit
 * [master] master commit
--
 * [master] master commit
+  [branch2] branch commit
+* [master^] second commit
```

8. Step #8 로 생성한 github 상의 저장소 주소는 무엇인가?
참고: 교수가 이 URL 에서 작업한 git 내용과 보고서를 가지고 채점할 예정임
https://github.com/ImDevelo/MJ_SE_Homework

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

merge는 다른 브랜치에서 현재 브랜치로 작업을 가져와 "병합"하는 작업을 말한다.
병합을 수행할 때 브랜치의 변경 사항이 서로 다른 경우 충돌이 발생하며
이때 충돌이 발생한 파일을 수정하고 저장하고 커밋하여 다시 병합을 시도하면된다.
merge는 히스토리가 남기때문에 git log등으로 병합한 내역을 히스토리로 확인할 수 있다.
rebase현재 브랜치의 변경 사항을 다른 브랜치의 변경 사항 위에 다시 적용하는 작업을 말한다.
다른 개발자들이 올린 수정사항을 내가 작업하고 있는 branch에 즉각 반영할 수 있어 branch를 항상 최신 내용으로 유지할 수 있다.
그리고 rebase 사용하면 커밋 히스토리가 남지 않아 관리가 더 깔끔해지며 병합 작업에서 발생하는 충돌이 적어진다.
대신 여러 사람이 함께 실시간으로 작업하는 경우 다른 사람의 작업 내용을 변경하는 위험이 있을 수 있다.

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

다른 브랜치에서 커밋을 가져와 현재 브랜치에 적용하는 명령어로
git cherry-pick (커밋 hash값)
으로 다른 브랜치에 있는 커밋 내역을 가져올수 있다.
이때 커밋의 변경 내용을 패치 형태로 현재 브랜치에 적용한다.

11. 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 -i에서 i는 인터렉트를 뜻하는 말로 rebase에 "인터랙티브" 모드를 사용하겠다는 말이다.
rebase -i를 사용하면 rebase 작업을 하기 전에 git은 편집기를 열어서 사용자가 선택한 커밋을 수정할 수 있다.
두 커밋에 내용을 바꾸려면 다음과 같은 순서로 해결할 수 있다.
1. git rebase -i HEAD~2 를 하면 직전 커밋 2개를 reabase한다.
2. 이때 rebase하기 전 git은 편집기를 제공하며 최근 커밋 2개 대한 간략한 내용을 보여준다.
(아래 내용은 예시로 step 과제를 하던 저장소에서 git rebase -i HEAD~2를 했을때 보여진 내용이다)
pick 7644bef second commit
pick 66e20d8 master commit
3. 편집기에서 이 두 커밋의 라인 위치를 바꾸고 저장하면 두 커밋의 순서가 바뀌어 커밋된다.