Git – 06

목차

- Part 3. Git의 다양한 활용 방법
  - Chapter 10. Git 고급
  - ~~Chapter 11. Git GUI~~
  - Chapter 12. GitHub로 협업하기
- 생략 내용
  - Part 2. 네 가지 주요 IDE의 Git 활용
    - 이클립스, VS, IntelliJ IDEA, Xcode
  - MAC OS X 관련 내용
  - part 3. Git GUI
    - 소스트리에서 깃 사용으로, GUI 부분 제외하면 part 1과 같음.

Chapter 10. Git 고급

- git tag : 특정 commit을 참조하는 이름 붙이기
- git commit -amend : 마지막 commit 수정하기
- git revert : 공개된 commit의 변경 내역을 되돌리기
- git reset : 이전 작업 결과를 저장한 상태로 되돌리기
- git checkout HEAD –filename : 특정 파일을 최종 commit 시점으로 되돌리기
- git rebase : branch 이력을 확인하면서 병합하기
- git rebase -i : commit 내역 합치기

Git 고급

- 명령어

명령어	설명
git tag	커밋을 참조하기 쉽도록 알기 쉬운 이름을 붙임.
git commit --amend	같은 브랜치 상에 있는 최종 커밋을 취소하고 새로운 내용을 추가하거나 설명을 덧붙인 커밋을 할 수 있음.
git revert	이전에 작성한 커밋을 지움. 그런데 특정 커밋의 내용을 지우는 새로운 커밋을 만들어 지운 내역을 모든 사람이 알 수 있게 함.
git reset	어떤 커밋을 버리고 이전의 특정 버전으로 다시 되돌릴 때 사용함. git revert와 다른 점은 지운 커밋 내역을 남기지 않는다는 점.
git checkout HEAD --filename	아직 커밋하지 않은 변경 내역을 취소함.
git rebase	git merge처럼 병합할 때 사용함. 하지만 브랜치가 많을 경우 브랜치 이력을 확인하면서 병합함.
git rebase -i	서로 다른 두 개의 커밋 내역을 합침.

git tag

- 특정 commit을 참조하는 이름 붙이기
  - tag 명령은 저장소의 commit에 tag를 붙이는 명령어이다.
    - 간단하게 버전 이름같이 이름만을 붙이는 light weight tag와 태그 작성자와 간단한 메모를 함께 태그에 남기는 annotated tag가 있다.
  - git tag 1.0
    - 버전 이름으로 태그를 붙임
  - git log --decorate -1

```
root@netsys:~/LabSeminar/git_tutorial# git log --decorate -1
commit aa63053324d61468b43f7ac8c4a041e0a53bd330 (HEAD -> master, tag: 1.0, origin/master)
Author: yh0921k <35794280+yh0921k@users.noreply.github.com>
Date:   Mon Feb 5 21:24:20 2018 +0000

    git pull
```

    - 가장 최근 commit인 git pull 명령 실행에 태그가 붙은 것을 확인할 수 있다.

NSL

git tag

- 특정 commit을 참조하는 이름 붙이기
  - git tag -l
    - 현재 저장소에 있는 태그 리스트
  - git show-ref --tags
    - 태그와 commit checksum 값을 같이 확인

```
root@netsys:~/LabSeminar/git_tutorial# git tag -l
1.0
root@netsys:~/LabSeminar/git_tutorial# git show-ref --tags
aa63053324d61468b43f7ac8c4a041e0a53bd330 refs/tags/1.0
```

git tag

- 특정 commit을 참조하는 이름 붙이기
  - 가장 최근 commit이 아닌, 특정 commit에 tag 붙이기
    - commit의 SHA-1 checksum 값을 알아야 한다.
  - git log -2

```
root@netsys:~/LabSeminar/git_tutorial# git log -2
commit aa63053324d61468b43f7ac8c4a041e0a53bd330
Author: yh0921k <35794280+yh0921k@users.noreply.github.com>
Date:   Mon Feb 5 21:24:20 2018 +0000

    git pull

commit 241535de49d8d2743129613dbb3105414d601d0f
Merge: 023ca73 aa8dbf5
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 06:19:13 2018 +0900

    conflict resolved GitHub
```

  - 대략적인 checksum 값을 기억한다.
    - 241535....

NSL

git tag

- 특정 commit을 참조하는 이름 붙이기
  - git tag 0.9 241535
  - git show-ref --tags
    - 태그가 붙여진 것을 확인할 수 있다.

```
root@netsys:~/LabSeminar/git_tutorial# git tag 0.9 241535
root@netsys:~/LabSeminar/git_tutorial# git show-ref --tags
241535de49d8d2743129613dbb3105414d601d0f refs/tags/0.9
aa63053324d61468b43f7ac8c4a041e0a53bd330 refs/tags/1.0
```

  - git tag 명령은 light weight tag만을 생성한다.
    - 누가, 언제, 왜, 이 태그를 붙였는지 전혀 알 수 없다.
  - annotated tag는 누가 언제 태그를 붙였는지 기록하고, 추가 메시지까지 같이 저장한다.
    - 로그를 살펴보면서 태그들을 보다가 해당 태그 시점에 태그에 대한 의문이 생기면 누구에게 질문해야 하는지 한번에 알 수 있다.
    - 또한 커밋과 다른 시점에 붙은 버전 태그가 있다면, 언제 해당 버전이 배포되었는지 알 수 있다.

git tag

- 특정 commit을 참조하는 이름 붙이기
  - git log -3
    - hello.py modified on Local repository 라는 commit
    - 체크섬 값을 입력하지 않으면, 현재 작업 중인 브랜치의 최신 커밋에 태그를 붙인다.
  - git tag –a 0.8 023ca

```
ver 0.8 by kyh
#
# 다음 태그에 대한 메시지를 쓰십시오:
#   0.8
# '#' 문자로 시작하는 줄은 무시됩니다.
```

  - tag를 적절히 작성 후 종료
  - git show 0.8
    - tag 확인

git commit --amend

- 마지막 commit 수정하기
  - git commit –amend(입력 x)
    - 위 명령을 수행하면, 마지막 커밋과 커밋하지 않은 상태에 있는 변경 내역이 서로 합쳐진 새 커밋을 만든다.
    - 만약 아무런 변경 내역을 만들지 않고 명령어를 실행하면 커밋 메시지만 변경하게 되는 것과 같은 효과를 낼 수 있다.
  - hello.py에 변경내역 만들기
    - # Third : git commit --amend 내용 추가
  - git add hello.py
    - 변경 내역 추가
  - git commit --amend
    - 다시 commit messge를 입력하는 편집기가 등장한다.
    - git pull 뒤에 by kyh라는 commit message 작성
  - git status
    - 상태 확인

NSL

git commit --amend

- 마지막 commit 수정하기
  - 이번에는 변경 내역이 없는 상태에서 commit message만 바꿔보자.
  - git commit --amend
    - 커밋 메시지를 수정한다.
  - git log -1
    - 변경된 commit message 확인
  - 부가적으로 설명하면, git commit --amend는 최종 commit을 수정하는 것이 아니라, 최종 commit을 대체하는 새로운 commit을 만드는 작업이다.
    - 명령을 실행하기 전과 후의 commit SHA-1 checksum 값을 비교해보자.

git revert

- 공개된 commit의 변경 내역을 되돌리기
  - 이미 공개된 commit 내역을 수정하는 것은 매우 위험하다.
    - 할 수는 있지만, 해서는 안된다.
  - 하지만 안전하게 변경 내역을 되돌리는 방법이 있다.
    - commit으로 발생한 변경 내역의 반대 commit을 하면 된다.
    - 즉, 추가한 코드는 빼고, 지운 코드는 다시 추가하는 commit을 뜻한다.
    - 이를 수행하는 명령이 git revert이다.
    - 이 명령을 특정 지점의 commit SHA-1 checksum 값에 적용하면, 해당 지점까지 변경 내역을 취소하게 된다.
  - git log -1
  - git revert 78192
    - commit messge 수정
  - 파일 내용을 확인해보면, 해당 시점의 주석이 사라진 것을 확인할 수 있다.
  - git revert 명령의 원리는, 실제로 되돌리는 것이 아니라 되돌리는 것 같은 효과를 내는 것임을 인지하자.
    - revert 명령을 실행한 시점부터, 대상 commit까지 변경 내역을 거꾸로 적용하는 새 커밋을 만드는 것이다.

git revert

- 공개된 commit의 변경 내역을 되돌리기
  - git log -3
    - 이미 공개된 commit 내역은 이런 안전한 방법으로 되돌려야 한다.

```
root@netsys:~/LabSeminar/git_tutorial# git log -3
commit df68e00fe02d6df9e1e6ca74723b1edbe765af03
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 11:42:06 2018 +0900

    Revert "GitHub for git pull by kyh"

    This reverts commit 78192be0f677eca742369de5ebc91bda626d446c.

commit 78192be0f677eca742369de5ebc91bda626d446c
Author: yh0921k <35794280+yh0921k@users.noreply.github.com>
Date:   Mon Feb 5 21:24:20 2018 +0000

    GitHub for git pull by kyh

commit 241535de49d8d2743129613dbb3105414d601d0f
Merge: 023ca73 aa8dbf5
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 06:19:13 2018 +0900

    conflict resolved GitHub
```

git reset

- 이전 작업 결과를 저장한 상태로 되돌리기
  - git reset 명령은 어떤 특정 commit을 사용하지 않게 되어 다시 되돌릴 때 사용한다.
    - revert 명령은 이전 commit을 남겨두지만, reset 명령은 이전 commit을 남기지 않고 새로운 commit을 남긴다.
  - 해당 명령은 현재 commit인 HEAD의 위치, 인덱스, 작업하는 저장소 디렉터리 등도 함께 되돌릴지를 결정하기 위한 모드를 지정할 수 있다.
    - hard : 완전히 되돌림(HEAD, 인덱스, 저장소 디렉터리)
    - mixed : 인덱스의 상태를 되돌림, default option(HEAD, 인덱스)
    - soft : commit만 되돌림(HEAD)
  - 인덱스
    - 실제 commit 전 변경 내역을 담는 준비 영역
    - git add 명령을 실행했을 때 이 영역으로 이동한다.
  - 저장소 디렉터리
    - 실제 파일이 담겨있는 작업 영역

git reset

- 이전 작업 결과를 저장한 상태로 되돌리기
  - commit을 취소하기 위한 옵션
  - ^ or ~
    - ~는 commit 내역 하나를 의미한다.
    - 표시한 수 만큼 commit을 되돌린다.
    - 하나면 최종 commit 내역일 것이다.
  - ORIG_HEAD
    - git reset 명령을 실행했을 때 지운 commit 내역을 보관한다.
    - 해당 명령을 통해 git reset 명령으로 지운 commit을 되돌릴 수 있다.

git reset

- 이전 작업 결과를 저장한 상태로 되돌리기
  - git log -5
  - git reset --soft HEAD~~~

```
root@netsys:~/LabSeminar/git_tutorial# git log -5
commit df68e00fe02d6df9e1e6ca74723b1edbe765af03
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 11:42:06 2018 +0900

    Revert "GitHub for git pull by kyh"

    This reverts commit 78192be0f677eca742369de5ebc91bda626d446c.

commit 78192be0f677eca742369de5ebc91bda626d446c
Author: yh0921k <35794280+yh0921k@users.noreply.github.com>
Date:   Mon Feb 5 21:24:20 2018 +0000

    GitHub for git pull by kyh

commit 241535de49d8d2743129613dbb3105414d601d0f
Merge: 023ca73 aa8dbf5
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 06:19:13 2018 +0900

    conflict resolved GitHub

commit 023ca73d110d264c04004be0671ab0f02cf5c661
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 05:59:07 2018 +0900

    hello.py modified on Local repository

commit aa8dbf5ad00a533c0a37ae0edb124488d7a8b850
Author: yh0921k <35794280+yh0921k@users.noreply.github.com>
Date:   Mon Feb 5 20:55:49 2018 +0000

    hello.py modified on GitHub
```

```
root@netsys:~/LabSeminar/git_tutorial# git log -3
commit 023ca73d110d264c04004be0671ab0f02cf5c661
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 05:59:07 2018 +0900

    hello.py modified on Local repository

commit 02cb9f1f78f393577b433d64d94e959ca03e385b
Author: kyh <yh0921k@gmail.com>
Date:   Tue Feb 6 05:39:40 2018 +0900

    remote repository add a README.md

commit 647e17ad35ef28562724fad51ecbad26d941b02d
Merge: 0c08c5d d1515ea
Author: kyh <yh0921k@gmail.com>
Date:   Mon Feb 5 16:07:43 2018 +0900

    conflice resolved
```

NSL

git reset

- 이전 작업 결과를 저장한 상태로 되돌리기
  - 파일 내용을 확인해보면 바뀌지 않았다.
  - git reset --soft ORIG_HEAD
    - 원래 상태로 되돌린다.
  - git log -5
    - 확인
  - git reset --hard HEAD~~~
    - 현재 head의 위치를 알려준다.
    - git log -3 명령을 실행하면 이전과 최신 commit에 대한 결과는 같다.
    - 하지만 파일의 내용을 보면 파일 내용이 변경되었다.
  - 다음 실습을 위해 git reset --hard ORIG_HEAD 명령을 실행한다.

git checkout HEAD -- filename

- 특정 파일을 최종 commit 시점으로 되돌리기
  - 위의 명령을 실행하면 해당 파일이 최종 commit 시점으로 되돌아가게 된다.
    - HEAD 대신 SHA-1 checksum 값을 이용하여 해당 commit 시점으로도 되돌리기 가능
  - 또한, --는 반드시 붙여야 한다.
    - 파일 이름이 브랜치 이름과 같을 경우 해당 브랜치로 체크아웃 하거나 특정 커밋 시점으로 저장소 전체가 되돌아갈 수 있다.
    - --는 git checkout 명령에 뒤따라 오는 것이 파일이라는 것을 확실하게 해주는 것이다.

git checkout HEAD -- filename

- 특정 파일을 최종 commit 시점으로 되돌리기
  - README.md
    - git checkout HEAD -- filename test 추가
  - git add README.md
    - 인덱스에 변경 내역 저장
  - git status
    - commit할 내역이 있다고 알려준다.
  - git checkout HEAD -- README.md
    - 파일이 수정 전으로 돌아감
  - git status
    - commit할 내역 없음
  - git checkout HEAD -- filename 명령은 git add 명령을 실행한 후 추가 수정 사항이 있을 때 되돌리기 위한 명령이다.

git rebase

- branch 이력을 확인하면서 병합하기
  - 실제 git을 이용한 시스템에서 작업 흐름은, 여러 개의 브랜치와 커밋 내역을 만들고, 마지막에 작업 내역을 확인하고 올바른 작업물만 병합하는 것이다.
  - Git의 특징 중 하나는 commit 내역을 수정할 수 있다는 것인데, 이미 원격 저장소에 push가 끝난 commit을 수정하는 것은 정말 특별한 상황이 아닌 이상 절대로 권고되지 않는다.
  - push 전에 git merge 명령을 이용해서 병합하면, 충돌 해결 commit등을 남기게 되고, 이는 작업 흐름을 일관되게 파악하는데 깔끔하지 않다.
  - 따라서 로컬 저장소에 있던 커밋을 깔끔하게 정리해서 푸시하는 것이 좋다.

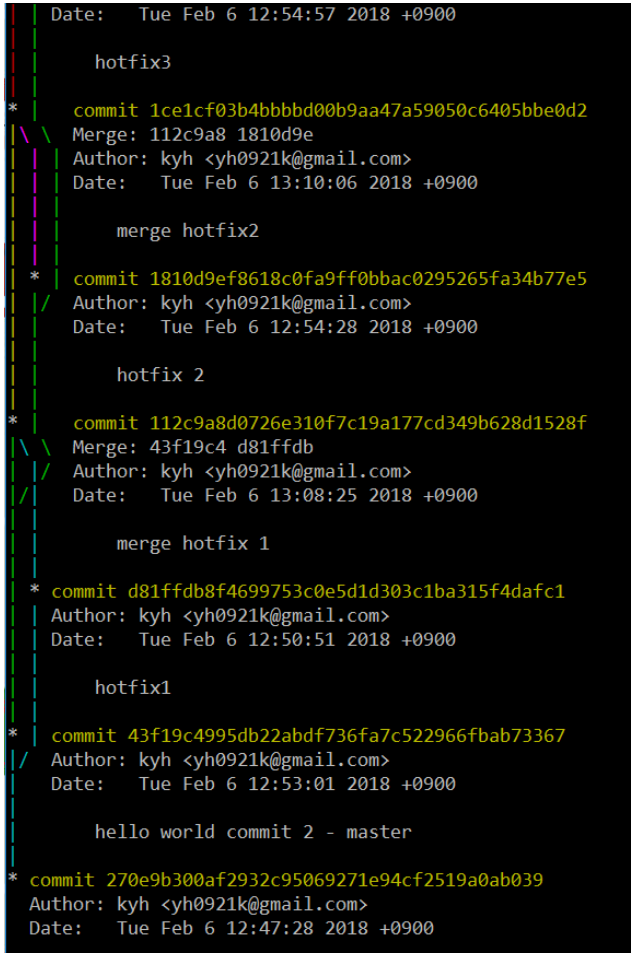
git rebase

- branch 이력을 확인하면서 병합하기
  - 저장소를 아래와 같이 만든다.

branch	file	code	commit message
master	hello.py	print(hello)	hello – master
		print(hello2)	hello2 – master
hotfix1	hello.py	print(hello)	
		print(hotfix1)	hotfix1
hotfix2	hello.py	print(hello)	
		print(hello2)	
		print(hotfix2)	hotfix2
hotfix3	hello.py	print(hello)	
		print(hello2)	
		print(hotfix3)	hotfix3

git rebase

- branch 이력을 확인하면서 병합하기
  - 전 장의 상태를 만들고 hotfix1, hotfix2, hotfix3를 차례로 merge하면 아래와 같은 그래프를 볼 수 있다.



git rebase

- branch 이력을 확인하면서 병합하기
  - 이전과 같은 상황을 git rebase 명령을 사용함으로써 해결할 수 있다.
  - hotfix1 branch 정리
  - git checkout hotfix1
  - git rebase master
    - 충돌이 발생하고, 이를 수정한다.

```
root@netsys:~/LabSeminar/git_re# git checkout hotfix1
'hotfix1' 브랜치로 전환합니다
root@netsys:~/LabSeminar/git_re# git rebase master
작업 사항을 다시 넣기 위해 먼저 헤드를 뒤로 돌립니다...
적용하는 중: hotfix1
인덱스 정보를 사용해 기본 트리를 다시 만듭니다...
M       hello.py
베이스 패치 적용 및 3-방향 병합으로 대신합니다...
자동 병합: hello.py
충돌! (내용): hello.py에 병합 충돌
error: 변경 사항에서 병합하는데 실패했습니다.
패치가 0001 hotfix1 위치에서 실패했습니다
실패한 패치의 복사본이 다음 위치에 있습니다: .git/rebase-apply/patch

이 문제를 해결하면, ""git rebase --continue"를 실행하십시오.
이 패치를 건너뛰려면, 대신에 "git rebase --skip"을 실행하십시오.
원래 브랜치를 체크아웃하고 리베이스를 중지하려면, "git rebase --abort"를 실행하십시오.
```

NSL

git rebase

- branch 이력을 확인하면서 병합하기
  - 충돌 상태를 해결하기 위한 세 가지 옵션
    - git rebase --continue : 충돌 상태를 해결한 후 계속 작업을 진행한다.
    - git rebase --skip : 병합 대상 브랜치의 내용으로 강제 병합을 실행한다. 즉, 이 상태에서 이 명령을 수행하면 master 브랜치를 강제로 병합한 상태가 된다. 또한 해당 브랜치에서는 다시 git rebase 명령을 실행할 수 없다.
    - git rebase --abort : rebase 명령을 취소한다. 다시 git rebase hotfix2 명령을 실행할 수 있다.
  - 수정 후 git add hello.py
  - git rebase --continue
  - git checkout master
  - git merge hotfix1 --no-ff

git rebase

- branch 이력을 확인하면서 병합하기
  - hotfix2와 hotfix3 branch도 차례로 정리해준다.

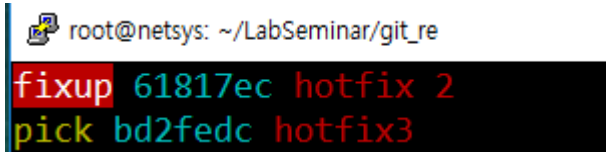
  - git checkout hotfix2
  - git rebase master
  - hello.py 수정
  - git add hello.py
  - git rebase --continue
  - git checkout master
  - git merge hotfix2 --no-ff

  - git checkout hotfix3
  - git rebase master
  - hello.py 수정
  - git add hello.py
  - git rebase --continue
  - git checkout master
  - git merge hotfix3 --no-ff

  - git log --graph
    - 충돌 commit 내역 없이, 어느 시점에서 commit 했고 어느 시점에서 merge 했는지 직관적으로 알 수 있다.

git rebase -i

- commit 내역 합치기
  - 앞에서 설명한 rebase 명령에는 활용도가 높은 옵션 -i가 있다.
    - interactive i로 상호 작용하면서 rebase할 commit을 고른다.
  - 전 장에서 완료한 commit graph도 완성도가 높은 그래프지만, 이러한 commit 내역과 작업 내역을 모두 합해서 더 깔끔하게 정리할 수 있다.
  - 가장 최근 commit 내역 두 개 합치기
  - git rebase -i HEAD~~
    - 남기는 커밋 메시지 앞에는 접두어로 pick을 붙인다.
    - 없애는 커밋 메시지 앞에는 접두어로 fixup을 붙인다.
    - 커밋 SHA-1 checksum 값은 꼭 남겨둔다
    - 기존의 커밋 메시지를 새롭게 수정할 수는 없다.



NSL