Session 2

Revision 조회

- SHA-1 해시 값의 앞 몇 글자만으로 어떤 커밋인지 확인 가능

```
git show 1c002d
```

- `git log –addrev-commit` 옵션 → 짧고 중복되지 않는 해시 값을 보여줌
- branch의 최근 커밋을 보고 싶으면 branch 이름으로 가리키는 것도 가능

```
git show topic1 #topic1의 가장 최근 커밋을 보여줌
```

- reflog를 이용하여 커밋을 가리키는 것도 가능 (reflog : branch, head가 몇 달 동안 가리켰었던 커밋을 기록하는 로그)

```
git reflog
```

- `@{n}` 규칙을 이용해 예전에 가리키던 것이 무엇인지 순서&시간으로 확인 가능

```
git show HEAD@{5}
git show master@{yesterday}
```

- 계통 관계로 커밋 표현 가능

```
git show HEAD^ #HEAD의 부모, 바로 이전 커밋
git show HEAD~ #HEAD의 부모, 바로 이전 커밋
git show HEAD^2 #HEAD의 두번째 부모 (Merge 커밋에만 사용 가능)
git show HEAD~2 #HEAD의 첫번째 부모의 첫번째 부모
```

- Double dot, Triple dot으로 커밋 표현 가능

```
git log master..experiment #master에는 없고 experiment에는 있는 ㅊ
git log origin/master..HEAD #origin 저장소의 master에는 없고 현재
git log master...experiment #master와 experiment의 공통부분을 빼고
```

대화형 명령(interactive staging)

- `git add -i` 명령을 사용해 대화형 모드로 들어갈 수 있음.
- 파일의 일부분만 Staging area에 추가하고 싶으면, 대화형 프롬프트에 `5, p` (patch)를 입력

Stashing & Cleaning

- stash : working directory에서 수정한 파일들만 저장
  - modified이면서 tracked 상태인 파일과 staging area에 있는 파일들을 보관해두는 곳

```
git stash #변경사항들을 stash에 저장
git stash list #저장되어있는 변경사항들 조회

git stash apply #가장 최근의 stash를 적용
git stash drop #stash를 적용하고 나서 바로 stack에서 제거
```

- stash를 만드는 방법

```
git stash --keep-index #staging area에 들어 있는 파일을 stash하지
git stash -u #untracked 파일도 같이 저장
git stash --patch #수정된 모든 사항을 저장하지 않음.
```

- stash를 적용한 branch 만드는 방법

```
git stash branch #stash 당시의 커밋을 checkout 한 후 새로운 브랜치를
```

- clean : 작업하고 있던 파일을 stash하지 않고 그냥 지움

```
git clean -f -d #untracked 정보를 워킹 디렉토리에서 지워버림(강제로)
git clean -d -n #가상으로 실행해보고 어떤 파일이 지워지는지 알려줌
glt clean -n -d -x #무시된 파일까지 함께 지움
git clean -x -i #대화형으로 실행
```

작업에 서명하기(Signing)

- GPG 이용

```
gpg --gen-key #키 생성
git config --global user.signingkey #이미 개인키가 있을 경우 git에
git tag -s v1.5 -m 'my sign' #새로 만드는 태그에 서명
git commit -a -S -m 'signed commit' #커밋에 서명
```

검색(Searching)

- `git grep` 명령어를 이용하여 커밋 트리의 내용이나 워킹 디렉토리의 내용을 쉽게 찾을 수 있음.

```
git grep -n gmtime_r #gmtime_r 문자열이 어느 라인에 있는지 번호까지
git grep --count gmtime_r #어떤 파일에 몇개가 있는지만 조회
```

히스토리 단장하기(Rewriting history)

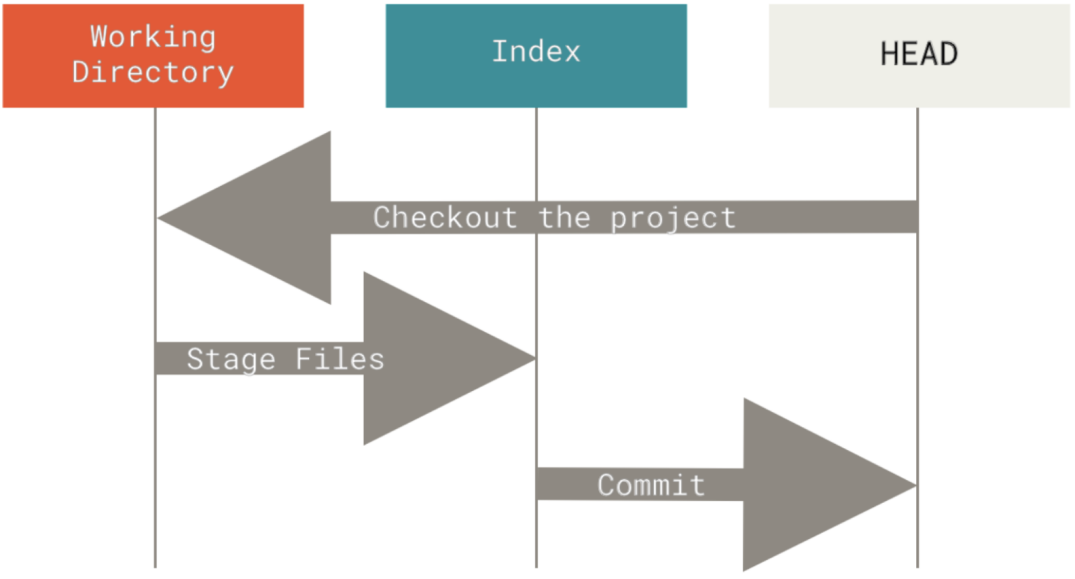
- `git commit --amend` 명령어를 이용해 가장 마지막 커밋을 수정할 수 있음
- `git rebase` 명령을 이용해 예전 커밋을 수정할 수 있음

```
git rebase -i HEAD~3 #마지마 커밋의 부모를 HEAD~3로 해서 넘기기
```

- `git rebase`를 이용하여 커밋 순서를 바꾸거나, 여러 개의 커밋을 하나로 합치거나, 하나의 커밋을 여러 개의 커밋으로 분리하는 것도 가능

Reset 명확히 알고 가기

- Git은 일반적으로 3가지 트리(파일의 묶음)를 관리하는 시스템
  - HEAD : 현재 브랜치를 가리키는 포인터, 즉 마지막 커밋의 스냅샷
  - INDEX : 바로 다음에 커밋할 것들.
  - Working directory : 샌드박스.



- 브랜치를 checkout 하면, HEAD가 새로운 브랜치를 가리키도록 바꾸고, 새로운 커밋의 스냅샷을 index에 놓고, index의 내용을 워킹 디렉토리로 복사

- reset 명령이 하는 역할
  1. HEAD 이동 `--soft`: HEAD 브랜치가 가리키는 커밋을 바꾼다. HEAD가 가리키는 브랜치는 이때 변하지 않는다
  2. INDEX 업데이트 `--mixed`: Index를 현재 HEAD가 가리키는 스냅샷으로 업데이트 한다
  3. Working directory 업데이트 `--hard` : 워킹 디렉토리까지 업데이트

- reset 과 checkout 의 차이점
  - `git checkout [branch]` 명령은 `git reset --hard [branch]` 명령과 유사, but
    - checkout 명령은 WD를 안전하게 지킨다. WD에서 merge를 시도해보고 변경하지 않은 파일만 업데이트한다.
    - checkout 명령은 HEAD 자체를 다른 branch로 옮긴다.

	HEAD	Index	Workdir	WD Safe?
Commit Level				
`reset --soft [commit]`	REF	NO	NO	YES
`reset [commit]`	REF	YES	NO	YES
`reset --hard [commit]`	REF	YES	YES	**NO**
`checkout [commit]`	HEAD	YES	YES	YES
File Level				
`reset (commit) [file]`	NO	YES	NO	YES
`checkout (commit) [file]`	NO	YES	YES	**NO**