Git ? GitHub?

Git :지옥에서 온 관리자

- 리누스 토르발스 -> 리눅스 개발 위해 제작

GitHub : 협업을 위한 온라인 서비스

Git으로 뭘 할 수 있는가?

- 버전관리
- 백업
- 협업

버전관리(Version Control)

```
수정을 지점을 기록(commit) 하여 버전을 관리
```

백업(Backup)

```
GitHub을 통한 백업
```

협업(Collaboration)

```
백업된 GitHub repository공간을 공유하여 협업 진행
```

학습 순서

설치 → 버전관리 → 백업 → 협업

환경 설정

```
$ git config --global user.name "원하는 이름"
$ git config --globla user.email "원하는 메일"
```

- 현재 컴퓨터에 있는 모든 저장소에서 같은 사용자 정보를 사용하도록 설정
- 깃헙과 는 다른 느낌, 단순히 누가 기록을 남기고 있는지 구분하는 차원

리눅스 명령어 연습

예시

Git 저장소 만들기

Git 초기화 : git init

```
버전관리 시작을 위한 최초 준비
```

```
$ mkdir hello-git #hello-git이라는 폴더 생성
$ cd hello-git    #hello-git 폴더로 이동

$ git init
```

Git은 디렉토리에서 가려져 있음

버전 관리

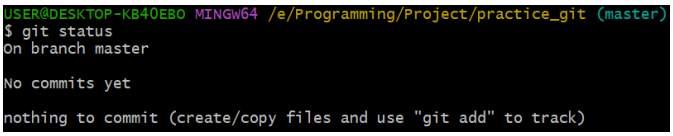
Git-seminar.excalidraw

버전 만들어지는 순서

> 1. **작업 트리에서 문서를 생성/수정**
> 2. 수정한 파일 중 **버전으로 만들고 싶은 것을 스테이지에 저장**
> 3. **스테이지에 있던 파일을 저장소로 커밋**

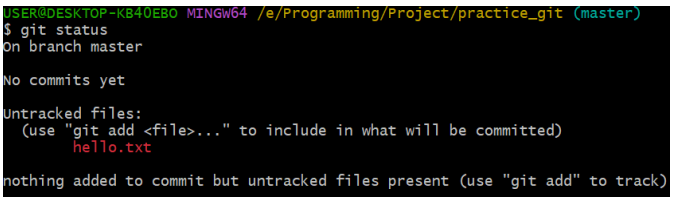
1. 작업 트리에서 파일 생성/수정

```
$ git init
$ git status   #깃의 상태 확인
```



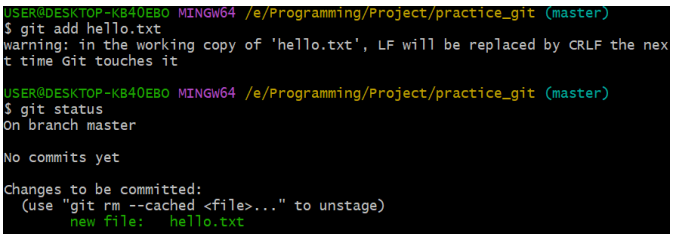
On branch master : 현재 master 브랜치에 있다는 뜻
No commits yet : 아직 커밋한 파일이 없다
nothing to commit : 현재 커밋할 파일이 없다.

```
$ vim hello.txt #vim 문서편집기 실행 후 수정
        a : 수정 시작
        esc : 수정 끝나면 수정화면에서 빠져나오기
        :wq : 작성 및 저장 명령어

$ git status
```



2. 수정한 파일을 스테이징 하기 - git add

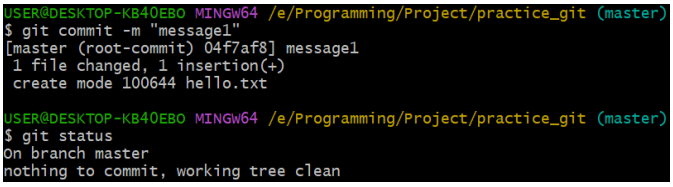
```
$ git add hello.txt
$ git status
```



Untracked files 에서 Changes to be committed로 바뀜
= 새 파일 : hello.txt를 (앞으로) 커밋할 것이다.

3. 스테이지에 올라온 파일 커밋하기 - git commit

```
$ git commit -m "message1"
$ git status
```

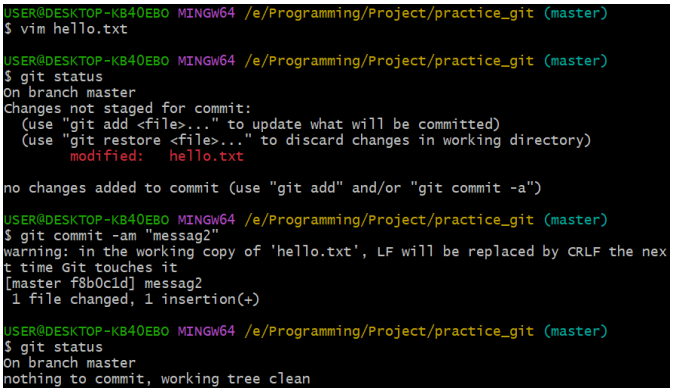


버전으로 만들 파일도 없고(nothing to commit) 작업 트리에서도 수정사항 없이 깨끗하다(working tree clean)

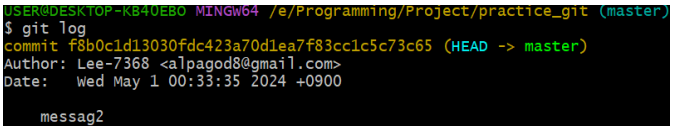
```
$ git log
```

4. 스테이징과 커밋 한꺼번에 처리하기 - git commit -am

```
$ vim hello.txt
$ git status
$ git commit -am "message2"
$ git status
```



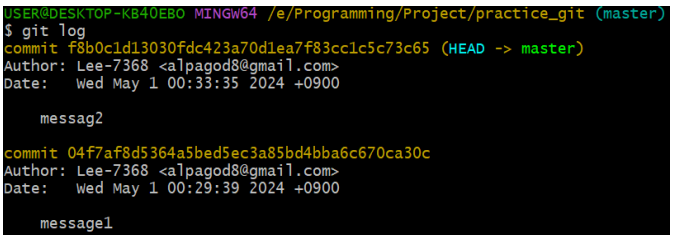

```
$ git log
```



커밋 내용 확인하기

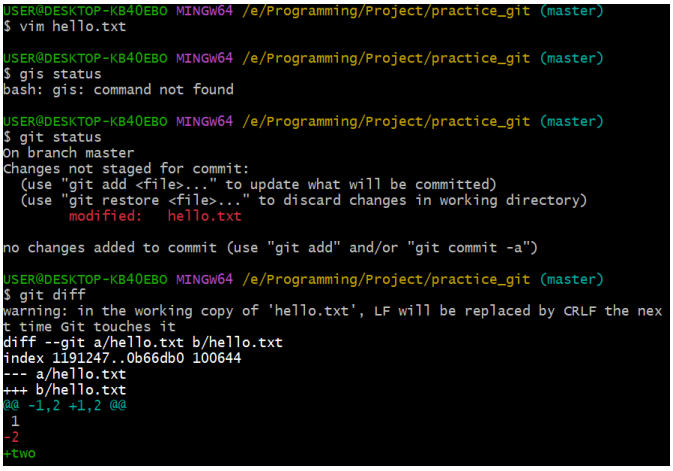
커밋 기록 살펴보기 - git t log

```
$ git log
```



변경사항 확인하기 - git diff

```
$ vim hello.txt
$ git status
$ git diff
```



버전 만드는 단계마다 파일 상태 알아보기

tracked 파일과 untracked 파일

```
$ vim hello.txt #hello.txt 파일에 '3'추가 후 저장
$ vim hello2.txt #hello.txt 파일에 a,b,c,d 개행하여 작성
```

```
$ git status
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git status
On branch master
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:   hello.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hello2.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

깃은 한 번이라도 커밋을 한 파일의 수정 여부를 계속 추적함

```
$ git add hello.txt
$ git add hello2.txt #git add. 하면 수정한 파일을 한꺼번에 스테이지에 올리 수 있음
$ git commit -m "message3"
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git add hello.txt
warning: in the working copy of 'hello.txt', LF will be replaced by CRLF the nex
t time Git touches it

USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git add hello2.txt
warning: in the working copy of 'hello2.txt', LF will be replaced by CRLF the ne
xt time Git touches it

USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git commit -m "message3"
[master 9b50340] message3
 2 files changed, 5 insertions(+)
 create mode 100644 hello2.txt
```

```
$ git log
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git log
commit 9b50340ebfd6a96887152a5fbd38c1f145ffc143 (HEAD -> master)
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 00:51:45 2024 +0900

    message3

commit f8b0c1d13030fdc423a70d1ea7f83cc1c5c73c65
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 00:33:35 2024 +0900

    messag2

commit 04f7af8d5364a5bed5ec3a85bd4bba6c670ca30c
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 00:29:39 2024 +0900
```

```
$ git log --stat  //더 자세한 정보. 정보 열람 후 'q'로 빠져나오기
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git log --stat
commit 9b50340ebfd6a96887152a5fbd38c1f145ffc143 (HEAD -> master)
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 00:51:45 2024 +0900

    message3

 hello.txt  | 1 +
 hello2.txt | 4 ++++
 2 files changed, 5 insertions(+)

commit f8b0c1d13030fdc423a70d1ea7f83cc1c5c73c65
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 00:33:35 2024 +0900

    messag2

 hello.txt | 1 +
 1 file changed, 1 insertion(+)

commit 04f7af8d5364a5bed5ec3a85bd4bba6c670ca30c
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 00:29:39 2024 +0900

    message1

 hello.txt | 1 +
 1 file changed, 1 insertion(+)
```

unmodified, modified, staged 상태

한 번이라도 commit을 한 파일은 tracted(추적) 되기 때문에 파일의 수정을 확인가능

```
$ git status
$ vim hello2.txt //b,c,d 삭제
$ git status
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git status
On branch master
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:   hello2.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

changes not staged for commit : 커밋을 위해 스테이지에 올라가지 않은 변경사항 있음

```
$ git add hello2.txt
$ git status
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello2.txt
```

changes to be committed : 커밋할 변경사항 존재 = 커밋 직전단계 즉, staged 상태

```
$ git commit -m "delete b,c,d"
$git status
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git commit -m "delete b,c,d"
[master d909311] delete b,c,d
 1 file changed, 3 deletions(-)

USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git status
On branch master
nothing to commit, working tree clean
```

작업 되돌리기

작업 트리에서 수정한 파일 되돌리기 - git restore

```
$ vim hello.txt //'3'을 'three'로 수정
$ git status
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ vim hello.txt

USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git status
On branch master
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:   hello.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

수정되었지만 스테이지에 아직 올라가지 않은 상태에서
git restore file to discard changes in working directory : 작업트리에서 수정 사항 폐기

```
$ git restore hello.txt
$ cat hello.txt
```

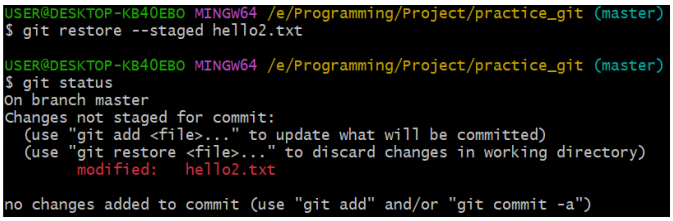
```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git restore hello.txt

USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ cat hello.txt
1
2
3
```

스테이징 되돌리기 - git reset HEAD 파일이름

```
$ vim hello2.txt  //기존 내용 삭제 후 대문자 'A, B, C, D' 개행하여 입력
$ git add hello2.txt
$ git status
```

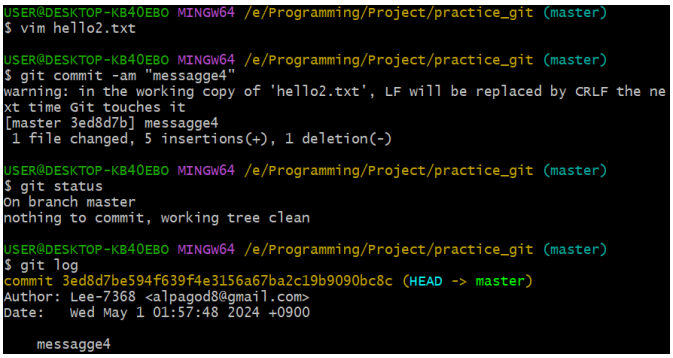
```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   hello2.txt
```

```
$ git restore --staged hello2.txt
$ git status
```



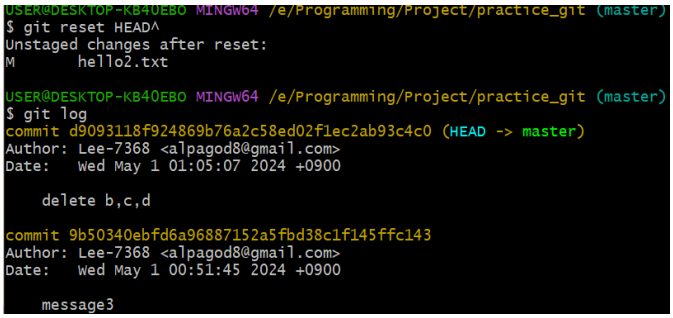
최신 커밋 되돌리기 - git reset HEAD^

```
$ vim hello2.txt //대문자 'E' 추가
$ git commit -am "message4"
$ git log
```



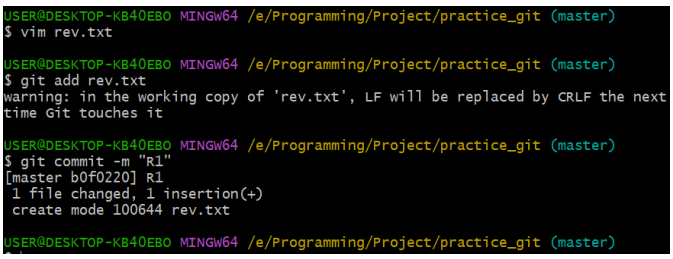
HEAD^ 는 현재 위치한 브랜치의 가장 최신 커밋을 가리킴
git reset HEAD^ 로 가장 최신 커밋을 삭제

```
$ git reset HEAD^
$ git log
```

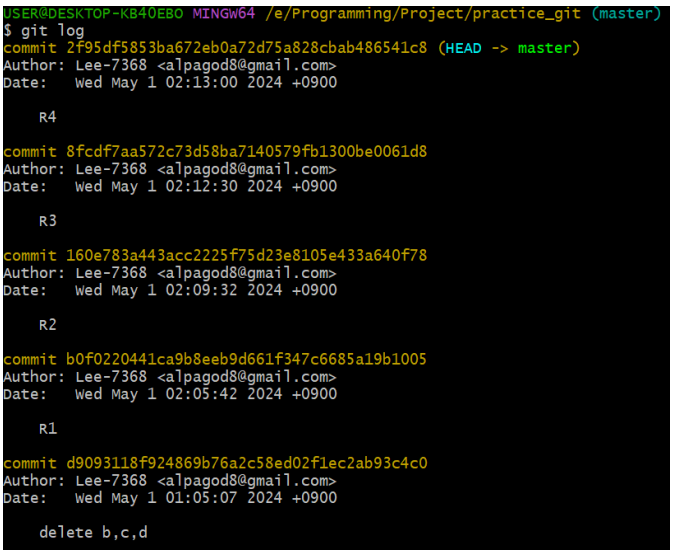


특정 커밋으로 되돌아가기 - git reset 커밋 해시

```
$ vim rev.txt //연습 위해 새 txt 파일 생성 후 'a' 작성, 저장
$ git add rev.txt
$ git commit -m "R1"
```




```
$ vim rev.txt //'b' 추가 작성 후 저장
$ git commit -am "R2"
$ vim rev.txt //'c' 추가 작성 후 저장
$ git commit -am "R3"
$ vim rev.txt //'d' 추가 작성 후 저장
$ git commit -am "R4"
$ git log //커밋들 확인
```



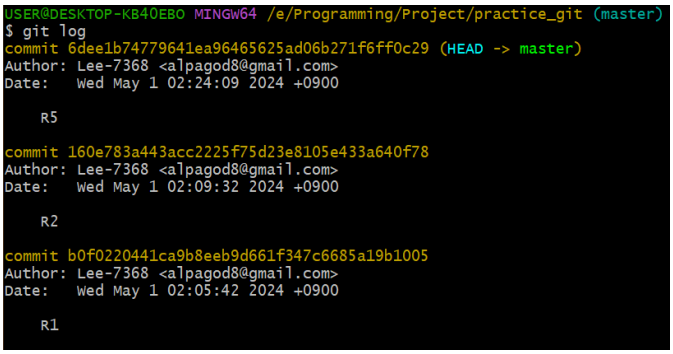
주의 사항 : 특정 해시 A로 리셋을 하면 A를 리셋 하는 것이 아닌 A로 리셋을 하는 것
즉, A 커밋 이후에 만들어진 모든 커밋을 삭제하고 A 커밋으로 이동하겠다는 뜻임

```
$ git reset --hard160e783a443acc2225f75d23e8105e433a640f78 //R2 커밋으로 이동
$ cat rev.txt //rev.txt 파일 내용 확인
```

```
--hard : 최근 커밋과 스테이징, 파일 수정을 하기 전 상태로 작업 트리를 되돌린다.
--mixed : 최근 커밋과 스테이징을 하기 전 상태로 작업 트리를 되돌린다. 기본 옵션
--soft : 최근 커밋을 하기 전 상태로 작업 트리를 되돌린다.
```

커밋 삭제하지 않고 되돌리기 - git revert

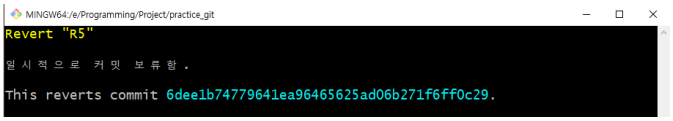
```
$ vim rev.txt //'e' 추가후 저장
$ git commit -am "R5"
$ git log
```



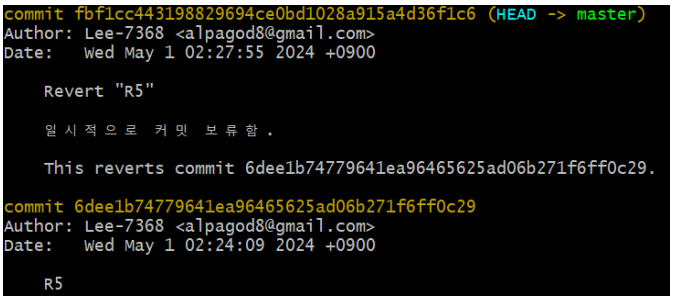
가장 최근 커밋 R5 버전을 취소하고 R2로 되돌아감
revert 의 경우 명령어 뒤에 취소하려고 하는 커밋 해시를 지정해야함 reset과 다름
`reset`: 이동하려는 커밋 해시를 지정
`revert`: 취소하려는 커밋해시를 지정

```
$ git revert 6dee1b74779641ea96465625ad06b271f6ff0c29
```

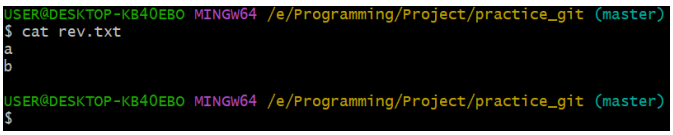
커밋 메시지를 입력 할 수 있는 기본 편집기 작동. 맨 윗줄은 어떤 버전을 revert 했는지 따라서 revert 하면서 추가로 남겨둘 내용을 입력하기 위한 장치. `i` 를 입력해 입력모드로 전환 후 작성 가능




```
$ git log
```




```
$ cat rev.txt
```



브랜치란 ?

나뭇가지라는 뜻
필요한 이유

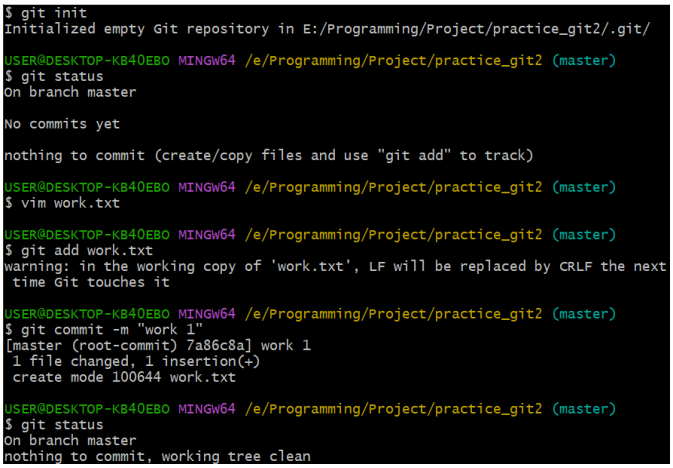
brach 설명

브랜치 만들기

환경설정

```
$ mkdir practice_git2
$ git init
```

```
$ vim work.txt // 'content 1' 작성 후 저장
$ git add work.txt
$ git commit -m "work 1"
```



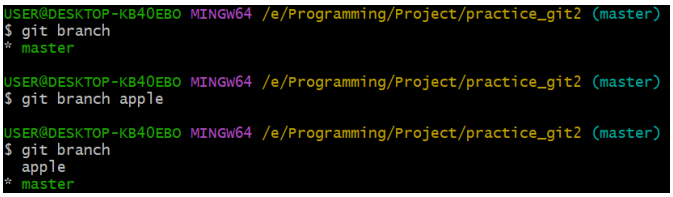

```
$ git log
```


```
$ vim work.txt // 'content 2' 추가후 저장
$ git commit -am "work 2"
$ vim work.txt // 'content 3' 추가후 저장
$ git commit -am "work 3"
$ git log
```

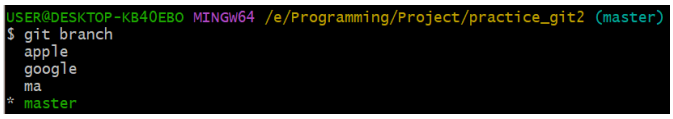
HEAD 가 master 브랜치를 가리키고 있음
HEAD 는 여러 브랜치 중에서 현재 작업 중인 브랜치를 가리킨다.
지금까지는 모두 공통을 사용 할 수 있는 내용이다. 이후 여러 고객사에 서로 다른 내용의 사용 설명서 제공해야하는 상황

새 브랜치 만들기 - git branch 브랜치 명

```
$ git branch //브랜치 현황을 알려줌
$ git branch apple // 'apple' 이라는 이름의 브랜치 생성
```

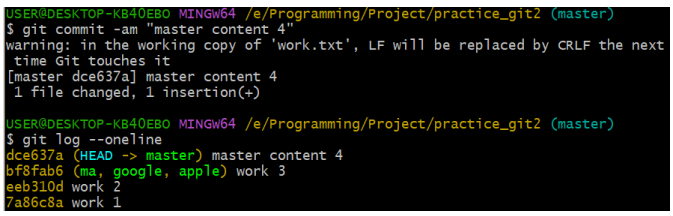



```
$ git branch google
$ git branch ms
$ git branch
```



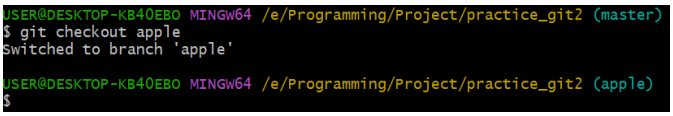
브랜치 사이 이동하기 - git checkout

```
$ vim work.txt // 'master content 4' 이력 후 저장
$ git commit -am "master conntent 4"
$ git log --oneline
```



최신 커밋인 'master work 4'는 master 브런치에만 적용된 모습
ms, google, apple 은 아직 work 3 커밋 상태

```
$ git checkout apple
```




```
$ git log --oneline
$ cat work.txt
```

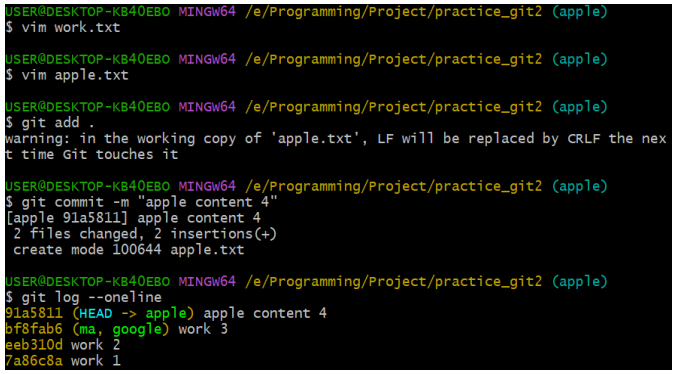
work 3에서 분기를 한 만큼 분기 이후에 master branch 에서 수정된 내용은 영향 주지 않음

새 브랜치에서 커밋하기

```
$ vim work.txt // work.txt에 'apple content 4' 추가 입력 후 저장
$ vim apple.txt // apple.txt를 만들고 'apple content 4' 입력 후 저장
```

```
apple 고객사만을 위한 추가 파일도 함께 작성
두 파일 각각 따로 스테이지에 올릴 수 있지만. 'sdd .' 을 통해 한꺼번에 스테이징
```

```
$ git add .
$ git commit -m "apple content 4"
$ git log --oneline
```




```
$ git log --oneline --branches // 각 브랜치의 커밋을 함께 볼 수 있음
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git2 (apple)
$ git log --oneline --branches
91a5811 (HEAD -> apple) apple content 4
dce637a (master) master content 4
bf8fab6 (ma, google) work 3
eeb310d work 2
7a86c8a work 1
```

```
$ git log --oneline --branches --graph // 위 옵션에 분기를 시각화 해줌
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git2 (apple)
$ git log --oneline --branches --graph
* 91a5811 (HEAD -> apple) apple content 4
| * dce637a (master) master content 4
|/
* bf8fab6 (ma, google) work 3
* eeb310d work 2
* 7a86c8a work 1
```

브랜치 사이 차이점 알아보기 - git log 비교브랜치1..비교브랜치2

```
git log master..apple // master branch 기준 다른 것 출력
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git2 (apple)
$ git log master..apple
commit 91a5811018215c8a7df92df4e6dd7207b6ca55be (HEAD -> apple)
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 03:44:37 2024 +0900

    apple content 4
```

```
$ git log apple..master // apple branch 기준 다른 것 출력
```

```
USER@DESKTOP-KB40EBO MINGW64 /e/Programming/Project/practice_git2 (apple)
$ git log apple..master
commit dce637abaa64108983aa377e75f0a174f70950f4 (master)
Author: Lee-7368 <alpagod8@gmail.com>
Date:   Wed May 1 03:34:14 2024 +0900

    master content 4
```

브랜치 병합하기

서로 다른 파일 병합하기 - git merge

```
병합간의 실수 잦은 탓에 새 dirc에서 진행
```

```
$ cd ..
$ git init practice_git3
$ cd practice_git3
```

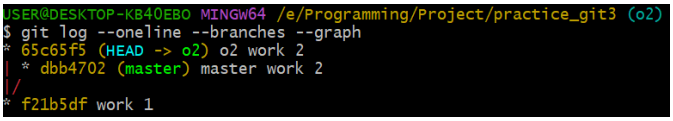
```
situation.1
$ vim work.txt // '1' 입력 후 저장
$ git add work.txt
```

```
$ git commit -m "work 1"
$ git branch o2
```

```
situation.2
$ vim master.txt // 'master 2' 추가 입력 후 저장
$ git add master.txt
$ git commit -m "master work 2"
```

```
situation.3
$ git checkout o2
$ vim o2.txt // o2.txt 를 만들어 'o2 2' 입력 후 저장
$ git commit -am "o2 work 2" // o2 work 2 커밋 안에는 o2.txt 파일과 work.txt 모두 있음
```

```
$ git log --oneline --branches --graph
```



이제 o2에서의 작업이 다 뜯났다고 가정하고 o2 내용을 master 브랜치로 병합
이를 위해서는 우선 master 브랜치로 돌아가야 함

```
situation.4
$ git checkout master
$ git merge o2
```

같은 문서의 다른 위치를 수정했을 때 병합하기

```
위 예제에서는 같은 work.txt는 건드리지 않고 서로다른 파일을 생성 후 병합하였다면
이번에는 양쪽 브랜치에서 work.txt를 수정하되 서로 다른 줄을 수정한 후 병합해봄
```

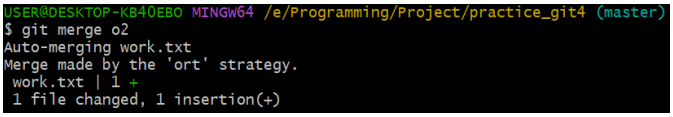
```
$ cd ..
$ git init parctice_git4
```

```
$ cd practice_git4
$ vim work.txt // 같은 파일이되 위치가 다른 두 파일을 합칠 만틈 두줄을 띄고 작성
```


```
$ git add work.txt
$ git commit -m "work 1"
$ git branch o2
$ vim work.txt // 개행된 첫줄에 'master content2' 작성후 저장
$ git commit -am "master work 2"
```

```
$ git checkout o2
$ vim work.txt // 마지막 content 밑에 'o2 content 2' 작성 후 저장
$ git commit -am "o2 work 2"
```

```
$ git checkout master
$ git merge o2
$ cat work.txt
```



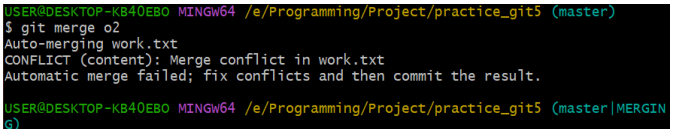
Auto-merging work.txt

> 깃에서는 줄단위로 변경 여부를 확인
> 각 브랜치에 같은 파일 이름을 가지고 있으면서 같은 줄을 수정 했을 때 브랜치를 병합하면 브랜치 충돌(conflict)가 발생
> 이후에는 다른 브랜치에서 같은 파일의 같은 위치르 수정한 후 병합해 보면서 어떤 경우에 브랜치 충돌이 일어나고 어떻게 충돌을 해결하는지 확인

같은 문서의 같은 위치를 수정하고 병합했을 때

```
$ cd ..
$ git init practice_git5
$ cd practice_git5
$ vim work.txt // 같은 파일 같은 위치 수정후 병합할 만큼 한줄 띄고 작성
```


```
$ git add work.txt
$ git commit -m "work 1"
$ git branch o2
$ vim work.txt // 가운데 빈 공간에 master content 2 작성 후 저장
$ git commit -am "master work 2"
$ git checkout o2
$ vim work.txt // 가운데 빈 공간에 o2 content 2 작성 후 저장
$ git commit -am "a2 work 2"
$ git checkout master
$ git merge o2
```

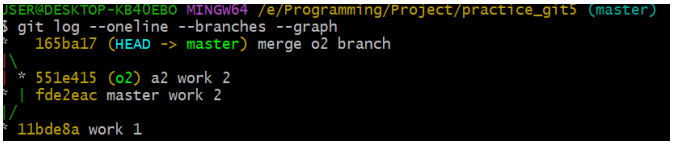


자동 병합간 충돌 발생, 충돌이 생긴 문서는 자동으로 병합 될 수 없음으로 사용자가 직접 충돌 부분을 해결한 후 다시 커밋해야 함

```
$ vim work.txt
```

원하는 모습으로 수정 후 저장 종료


```
$ git commit -am "merge o2 branch"
$ git --oneline --branches --graph
```



병합이 끝난 브랜치 삭제하기 : git branch -d

```
$ git branch
$ git checkout master
$ git branch -d o2
```

브랜치 관리하기

브랜치에서 `checkout` 과 `reset`의 동작원리

- 나중에

백업 - GitHub

원격 저장소와 Github

local repository

- 지금까지 우리가 작업한 환경

remote repository

- 원격 저장소
- 원격 저장소를 공유하여 서로 다른 로컬 저장소와 협업 가능

활용 순서

- 깃허브 가입
- 깃허브 원격 저장소 만들기
- local repo 생성
- local repo와 remote repo 연결
  - http vs ssh
- local repo에서 작업 후 remote repo에 올리기
- 필요시 ssh키 생성하기
- ssh키 깃헙에 등록
- ssh키 등록 확인

깃헙 가입

[GitHub]

GitHub 원격 저장소 만들기

[GitHub]

local repository 준비

```
$ git init loc-repo
$ cd loc-git
$ vim f1.txt   //아무 내용 입력 'a'
$ git add f1.txt
$ git commit -m "add a"
```

local repo와 remote repo 연결

```
$ git remote add origin 복사한 주소 붙여넣기 //로그인 요청시 로그인
$ git remote -v   //연결 여부 확인
```

remote repo에 파일 올리기 - git push

```
$ git push -u origin master   // 최초에 한해서만 실행
$ git push                    // 최초 push 후 수정사항을 업로드 할 때
```

Github 사이트에서 직접 커밋하기

GitHub

remote repo에서 파일 내려 받기 - git pull

```
$ git pull origin master
```

SSH키 생성 및 관리

```
secure shell : public key, private key
```

```
$ ssh-keygen
$ cd ~/.ssh
$ cat id_rsa.pub
```

협업

여러 컴퓨터에서 원격 저장소 함께 사용하기

원격 저장소 복제하기 - git clone

원격 브랜치 정보 가져오기

원격 브랜치 정보 가져오기 - git fetch

협업의 기본

1. 깃헙 repo 파기

2. 공동 작업자 추가하기

3. 작업 환경 구성하기

```
$ git init 저장소 이름
$ cd 저장소 이름
```

```
$ git config user.name "사용자 이름"
$ git config user.email 메일 주소
```

4. 원격 저장소에 첫 커밋 푸시하기

```
$ vim overview.txt
$ git add overview.txt
$ git commit -m "overview"

$ git remote add origin 복사한 저장소 주소
$ git puch -u origin master
```

5. 공동 작업자 컴퓨터에 원격 저장소 복제하기

```
$ git clone 원격 저장소 주소(https or ssh)
```

6. 첫 커밋자가 아니라면 풀 먼저 하기

```
모든 작업을 시작하기 전에는 무조거거건!!!! 풀 하여 최신 작업물 불러오기
그러지 않으면 rejected 됨
```

7. 이후 작업 후 첫 커밋을 할 때는 -u origin master

협업에서 브랜치 사용하기

필수 브랜치명

- master : 프로젝트의 대표격 브랜치, 자주 할 필요 없이, release 브랜치가 업데이트 될때 진행
- develop : 기능개발
- feature : 내가 기능을 만든 후에 모두와 합치는 브랜치
- release : feature 브랜치에서 테스트 돌리고 완벽해져서 배포를 생각하는 수준의 결과물이 나왔을 때 때리는 push