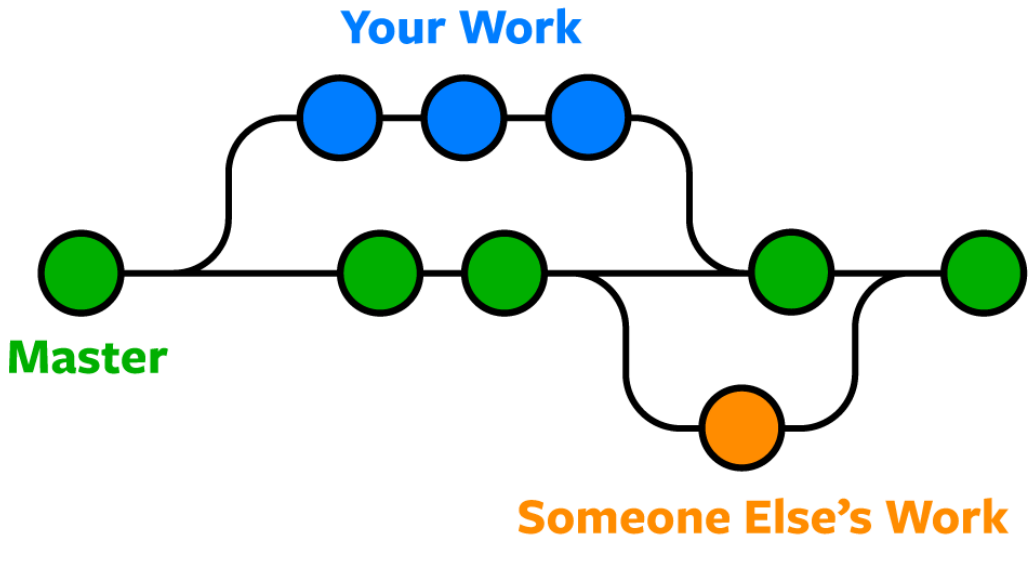
Git Branch(브랜치) 란?

SW를 개발할 때, 깃의 브랜치기능을 활용한다면 **같은 팀끼리 작업 프로젝트를 공유하고 같이 작업할 수 있도록** 해준다.

또 혼자서 작업하더라도 여러 버전을 만들어 놓을 때, 또는 본 작업에서는 시도하기 힘든 테스트를 할 때 바로 이 "**브랜치**"라는 것이 굉장히 유용하게 된다.

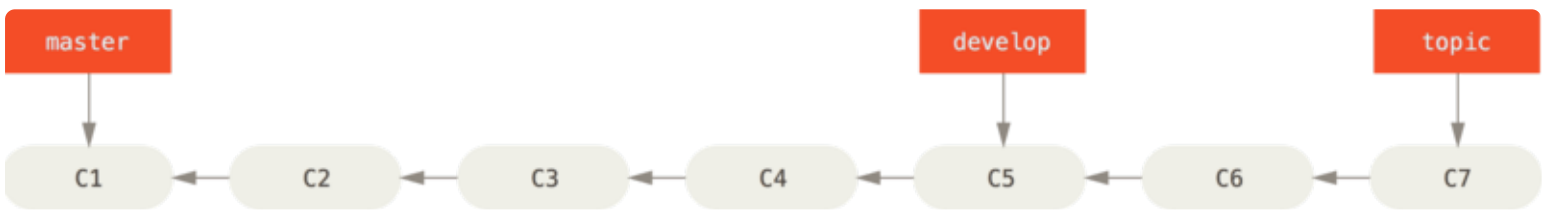


동시에 다양한 작업을 할 수 있게 만들어 주는 기능 "브랜치(Branch)"를 이용하면 **각자 독립적인 작업 영역(저장소) 안에**서 마음대로 소스 코드를 변경할 수 있다.

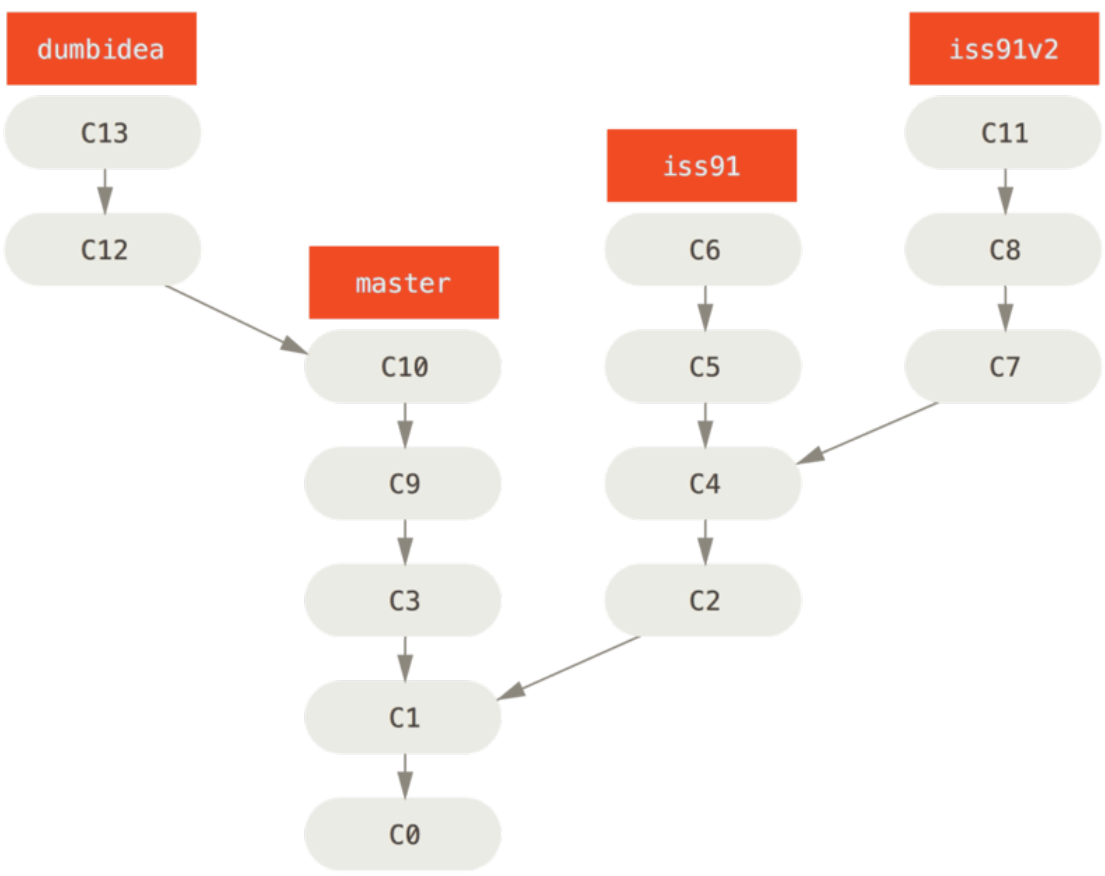
브래치는 독립적으로 어떤 작업을 진행하기 위한 개념으로,

필요에 의해 만들어지는 **각각의 브랜치는 다른 브랜치의 영향을 받지 않기 때문에** 여러 작업을 동시에 진행할 수 있게 된다.

만일 브랜치가 없다면 작업을 모두 수행 후 다른 작업자에게 넘겨줘야 한다.



하지만 브랜치 기능을 활용한다면 하나의 작업을 여러 시점에서 개발할 수 있게된다..



Git 브랜치 용어 정리

📌 마스터 브랜치(Master Branch)

저장소를 처음 생성하게 되면 깃은 '**Master**'라는 이름의 브랜치를 자동으로 생성하게 된다.

따로 새로운 브랜치를 생성하지 않으면 저장소에 새로운 파일을 추가하는 것이나 소스코드를 수정하여 **커밋하는 작업 모두 마스터 브랜치를 통해 이루어지게** 된다.

📌 통합 브랜치(Integration Branch)

언제든 **배포할 수 있는 버전**을 만들 수 있어야 하는 브랜치가 통합 브랜치 이다.

따라서 안정적인 상태, 즉 모든 기능이 정상적으로 동작하는 상태가 되어있어야 한다.

만약 **버그를 수정하거나 새로운 기능을 추가해야 한다면** 통합 브랜치가 아닌 토픽 브랜치를 만들어 사용한다.

일반적으로 마스터 브랜치를 통합 브랜치로 사용하는 편이다.

📌 토픽 브랜치(Topic Branch)

기능 추가나 **버그 수정과 같은 단위 작업을 위한 브랜치**를 토픽 브랜치라고 한다.

토픽 브랜치는 보통 통합 브랜치로부터 파생해서 생성하고, 특정 작업이 완료되면 통합 브랜치에 병합하는 방식으로 작업하게 된다.

피처 브랜치(Feature Branch)라고 부르기도 한다.

📌 체크 아웃(Checkout)

깃에서는 항상 작업할 브랜치를 미리 선택해야 한다.

맨 처음에는 마스터 브랜치가 선택되어 있는데, **다른 브랜치로 전환**하여 작업할 때 사용하는 명령어가 체크 아웃(Checkout) 이다.

독립된 작업 공간인 브랜치를 자유롭게 이동할 수 있다.

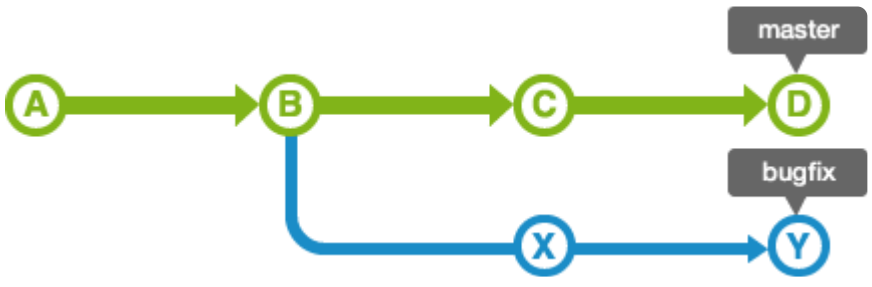
📌 헤드(Head)

헤드는 **현재 사용 중인 브랜치의 선두 부분**을 나타내는 이름이다.

📌 머지(Merge)

merge 를 사용하면, **여러 개의 브랜치를 하나로 모을 수** 있다.

브랜치에서 작업을 끝내고, 모든 협업자가 볼 수 있는 **master 브랜치로 병합할 수 있다.**



📌 리베이스(Rebase)

통합 브랜치에 토픽 브랜치를 통합한다는 점에서 머지와 비슷하지만 특징이 약간 다르다.

Merge는 변경 내용의 이력이 모두 남아 있어서 이력이 복잡해지지만,

Rebase는 이력이 단순해지지만, 원래 커밋 이력이 변경된다.

정확한 이력을 남겨둬야 할 필요가 없을 경우 사용한다.

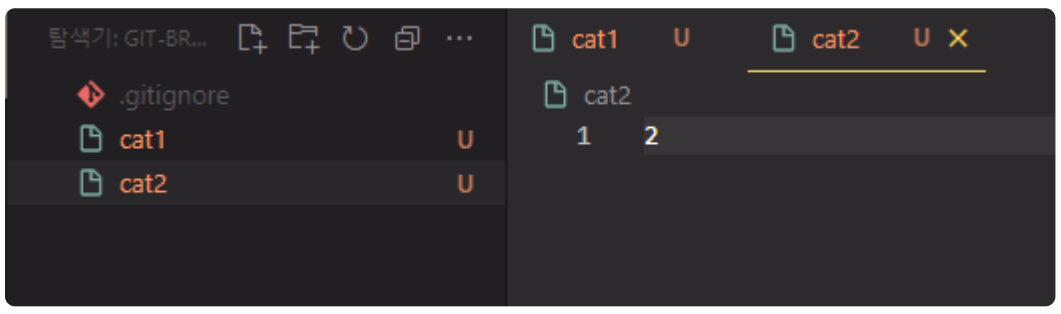
Branch 사용하기

git branch <브랜치이름>

1. 초기화 후 파일 생성

먼저 이전에 작업했던 내용을 모두 지운 깨끗한 상태에서 진행 해보자.

먼저 cat1파일과 cat2파일을 폴더 안에 생성한다.



그리고 git status로 확인해보면 아직 커밋하지 않은 파일이 Untracked하다고 뜨게 된다.

해당 내용을 add, commit 하자

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch
$ git init
Initialized empty Git repository in C:/Users/MS/Desktop/공부/git-branch/.git/

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        cat1
        cat2

nothing added to commit but untracked files present (use "git add" to track)
```

2. 새로운 브랜치 생성

BASH

```
$ git branch <브랜치이름>
$ git branch mouse
```

cat1, cat2라는 파일 두개를 추가 후,

mouse1, mouse2 파일을 추가해 줄지, dog1, dog2 파일을 추가해 줄지 고민이 되는 상황이라고 가정해보자.

일단 두 버전 다 만들어 보자.

여기서 중요한 점은, **아무 브랜치도 생성하지 않으면 처음에 생성되는 master브랜치에서 작업을 수행**하고 있는 것이다.
이 상태에서 브랜치를 생성해주면, **master 브랜치의 작업 내용 그대로 복사된 브랜치가 생성**된다.

브랜치를 생성 후, **반드시 checkout을 해줘야 해당 브랜치로 전환**이되며
전환된 브랜치에서 작업이 수행된다.

BASH

```
$ git checkout 브랜치이름
$ git checkout mouse # mouse브랜치로 이동하라
```

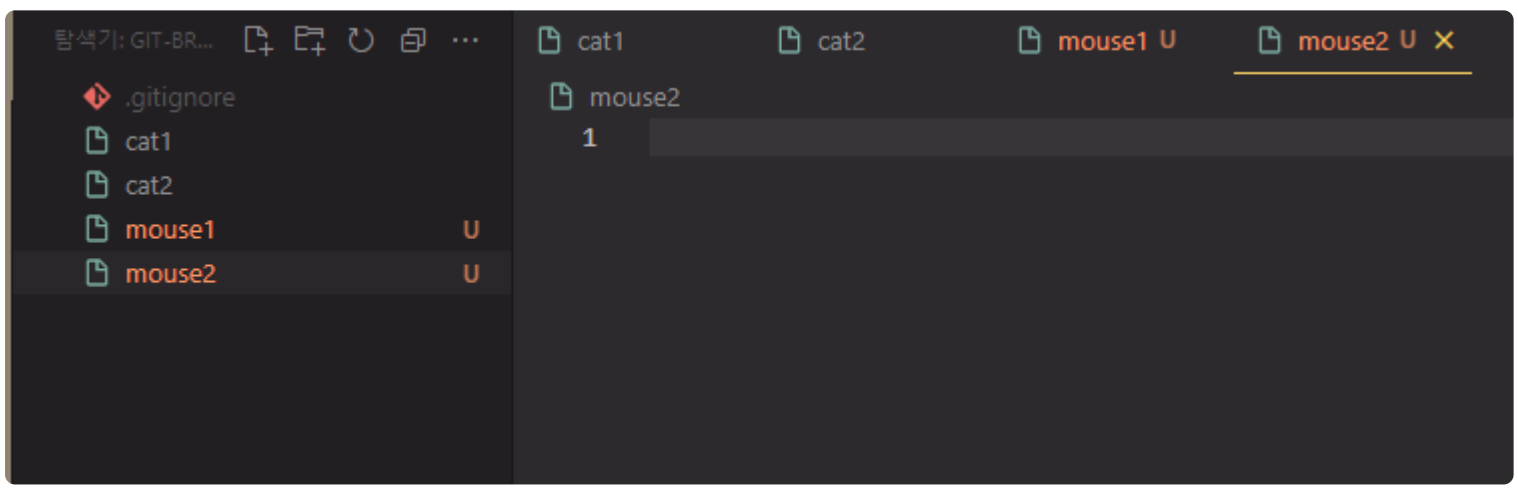
먼저 mouse1, mouse2파일을 추가해볼 mouse라는 이름의 브랜치를 생성하고 mouse브랜치로 이동하였다.
마스터브랜치에서 생성한 브랜치이므로 처음 작업 내용 그대로 복사된다.

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git branch mouse

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git checkout mouse
Switched to branch 'mouse'

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (mouse)
$ git shortlog
inpa (1):
      add cat1, cat2
```

그리고 mouse브랜치로 전환된 상태로 mouse1, mouse2라는 파일을 생성하고 add commit 한다.




```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (mouse)
$ git add mouse1

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (mouse)
$ git add mouse2

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (mouse)
$ git commit -m "add mouse1, mouse2"
[mouse 116b997] add mouse1, mouse2
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 mouse1
 create mode 100644 mouse2

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (mouse)
$ git shortlog
inpa (2):
      add cat1, cat2
      add mouse1, mouse2
```

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (mouse)
$ git shortlog
inpa (2):
      add cat1, cat2
      add mouse1, mouse2

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (mouse)
$ git checkout master
Switched to branch 'master'

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git shortlog
inpa (1):
      add cat1, cat2
```

브랜치 마다 작업내용이 다르다

만약 이 상태에서 마스터 브랜치로 체크아웃하지 않고,

또 다른 브랜치를 생성하면 cat1, cat2, mouse1, mouse2 파일 모두 포함된 브랜치가 생성되게 된다.

> **Tip**
>
> 여기서 주의할 점은
>
> git add -A와 같은 옵션을 사용하게 되면 mouse 브랜치의 작업 내용이 모든 브랜치에 적용되게 된다.

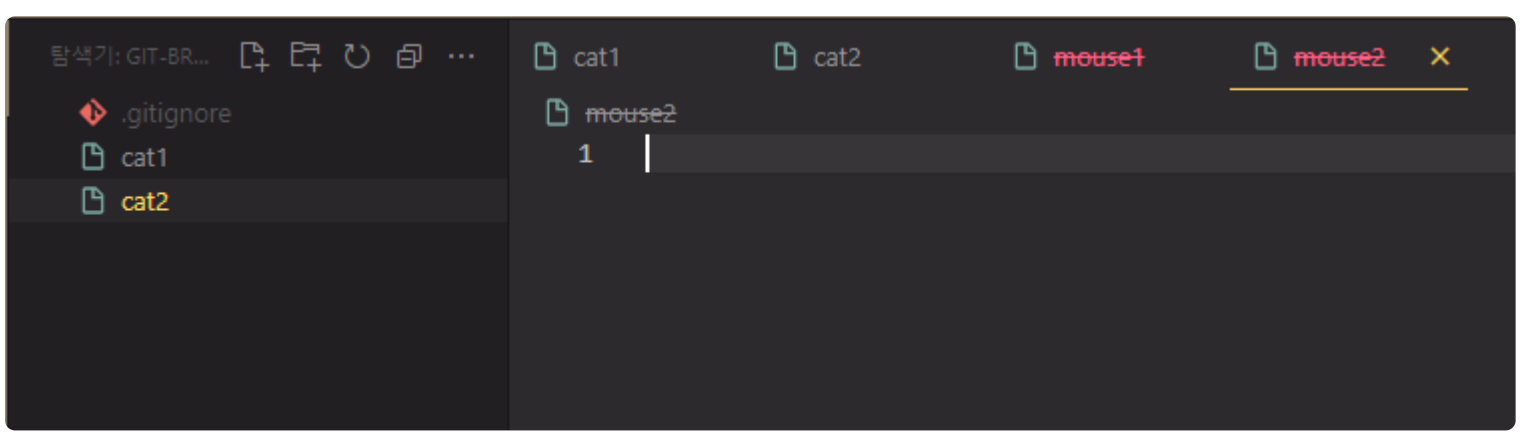
우리는

cat1, cat2, mouse1, mouse2를 가진 브랜치와

cat1, cat2, dog1, dog2를 가진 브랜치를 생성하고 싶으니,

다시 마스터 브랜치로 이동 후 dog 브랜치를 생성하자

git checkout master를 하게 되면 이렇게 mouse브랜치에서 작업한 내용은 보이지 않게 된다.



이 상태에서 dog branch를 생성해 준다.

BASH

```
# 브랜치 확인
$ git branch
```

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (dog)
$ git branch
* dog
  master
  mouse
```

dog 브랜치로 이동 후, dog1, dog2 파일을 생성하고 이후 add, commit 을 해준다.

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (dog)
$ git add .

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (dog)
$ git commit -m "add dog1, dog2"
```

> **Tip**
>
> ※ 터미널 내용 모두 지우는 명령어
>
> **$ clear**

git checkout -b <브랜치명>

하나의 명령어로 브랜치를 만들고 바로 이동할수있다.

BASH

```
# another이라는 브랜치를 만들고 checkout해서 another로 이동해라
$ git checkout -b another
```

git switch <브랜치이름>

이전에는 git checkout <BRANCH_NAME>으로 브랜치를 변경했지만 이젠 switch로 변경한다.

명령어만 달라졌고 사용 방법은 같다.

BASH

```
# 'develop' 브랜치로 전환
$ git switch develop
# 'new-branch'를 새로 만들고 바로 브랜치로 전환
$ git switch -c new-branch
```

브랜치를 어디서 만들지는 지정하지 않았으므로 HEAD가 사용되었는데,

특정 브랜치나 커밋에서 새로운 브랜치를 만들고 싶으면 브랜치 이름 뒤에 커밋을 지정해 주면 된다.

BASH

```
# 커밋515c633a 위치에서 새로운 브랜치 'new-branch2'를 만든다.
$ git switch -c new-branch2 515c633a
```

git merge <브랜치이름>

3. 브랜치 합치기 - merge

우리 팀은 고민하다가 cat과 mouse파일을 남기기로 최종 결정하였다고 한다.

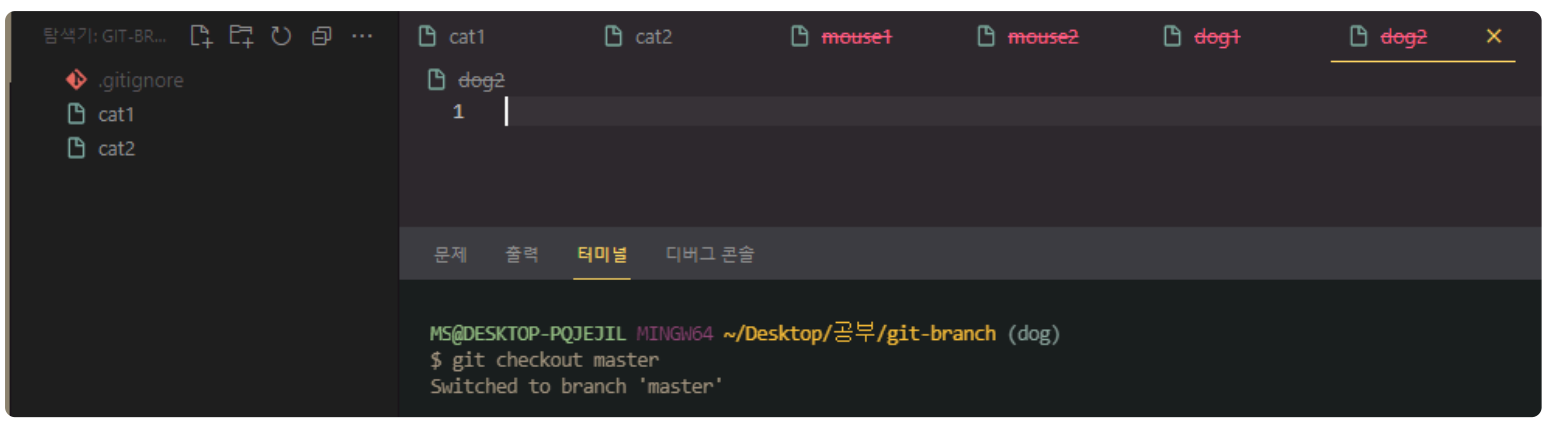
그러면 master 브랜치에 mouse브랜치를 이제 병합해야 한다.

BASH

```
# 지정한 branch의 commit들을 -> 현재 branch 및 워킹 트리에 반영
# <브랜치이름>에 merge하는게 아닌, 현재 브랜치 이곳에 <브랜치이름>을 merge하는 것이다.
```

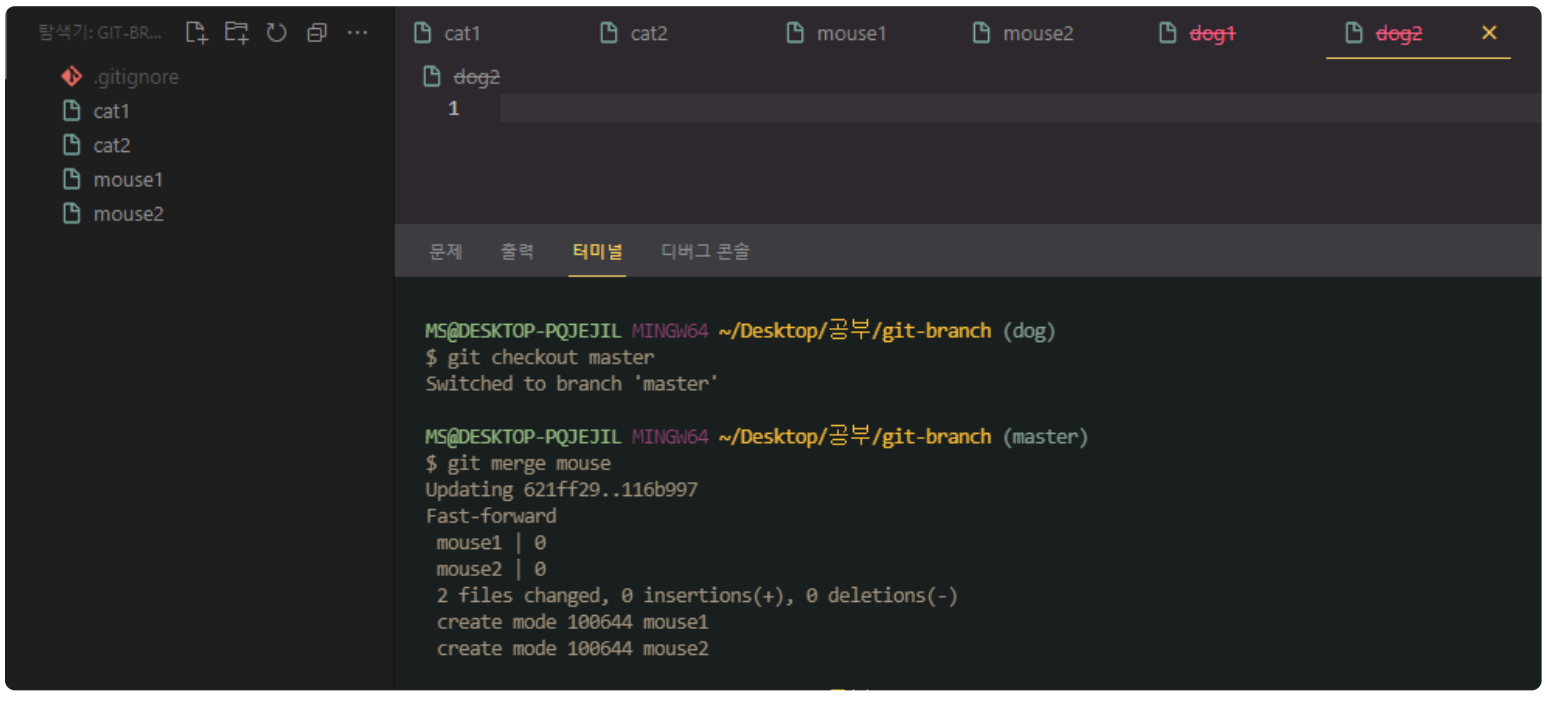
```
$ git merge <브랜치이름>
```

일단 다시 마스터 브랜치로 돌아온 후,



merge하게 되면, mouse브랜치에 있던 작업내용이 master 브랜치로 합쳐지게 된다.

mouse폴더들이 생긴걸 알 수 있다.



이후, 헷갈리지 않도록 되도록 브랜치를 삭제해주는 것이 좋다.

BASH

```
$ git branch -D <제거할 브랜치이름>
```

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git branch -D mouse
Deleted branch mouse (was 116b997).

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git branch
  dog
* master
```

※ 여러 분기에서 작업한 내용을 확인하는 명령어

BASH

```
$ git log --graph --all --decorate
```

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$  git log --graph --all --decorate
* commit c4e1920d1997373a9d0445beaf73f4452f61afd4 (dog)
| Author: inpa <jeffbk0170@gmail.com>
| Date:   Fri Nov 19 17:10:07 2021 +0900
|
|     add dog1, dog2
|
| * commit 116b9975929d8ba6d863c838a75feb2cf62a452e (HEAD -> master)
|/  Author: inpa <jeffbk0170@gmail.com>
|   Date:   Fri Nov 19 16:31:40 2021 +0900
|
|       add mouse1, mouse2
|
* commit 621ff292dd96b5c1ff57a1f66779883dffd7ba42
  Author: inpa <jeffbk0170@gmail.com>
  Date:   Fri Nov 19 16:26:52 2021 +0900

      add cat1, cat2
```

Tip

⚠ Merge시 주의할점

Merge시, **같은 파일 같은 라인에 수정**을 한 두 브랜치를 병합하려면 **Conflict**가 발생하게 된다.
파일 둘 중 1개를 선택해서 다시 merge를 수행해주어야 하니,
되도록 같은 파일을 다른 브랜치가 작업하게 만드는 것은 지양해야 한다.

git rebase <브랜치이름>

3-1. 브랜치 합치기 - rebase

BASH

```
# 내 branch의 commit들을 대상 branch에 재배치
$ git rebase <브랜치이름>
```

Tip

rebase합치기는 히스토리 **그래프가 일자로 깔끔해져서** 자주 사용하지만
원격 저장소에서는 사용 안 하는 것을 권장

예를 들자면,

a 커밋을 원격 저장소에 push 하고 로컬 저장소에서 rebase를 하게 되면,
원격에는 a가 존재하고 로컬에는 다른 커밋인 a`가 생성된다.

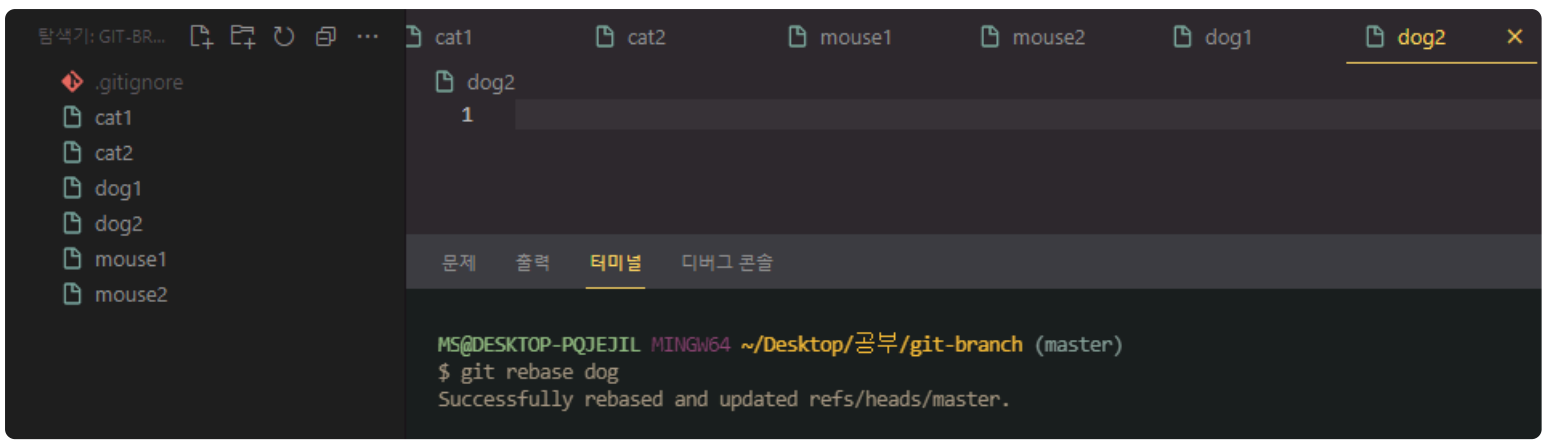
이때 내가 아닌 다른 개발자가 원격 저장소에 있던 a를 merge할 수 있고 이 때 변경된 a`도 언젠가는 원격 저장소에 push가 될 텐데 그럼 원격 저장소에는 실상 같은 commit이었던 a와 a`가 동시에 존재하게 된다.

이 때문에 원격 저장소에서와 같이 작업할때는 rebase를 사용하지 않는 것을 권장한다.

마스터브랜치에 mouse브랜치의 작업 내용을 merge를 통해 병합해보았다면
이번엔, rebase를 통해 dog브랜치에서 작업한 내용을 합쳐보자.

위에서 설명했듯이 rebase는 브랜치의 작업을 병합한다는 점에서는 merge와 같지만,

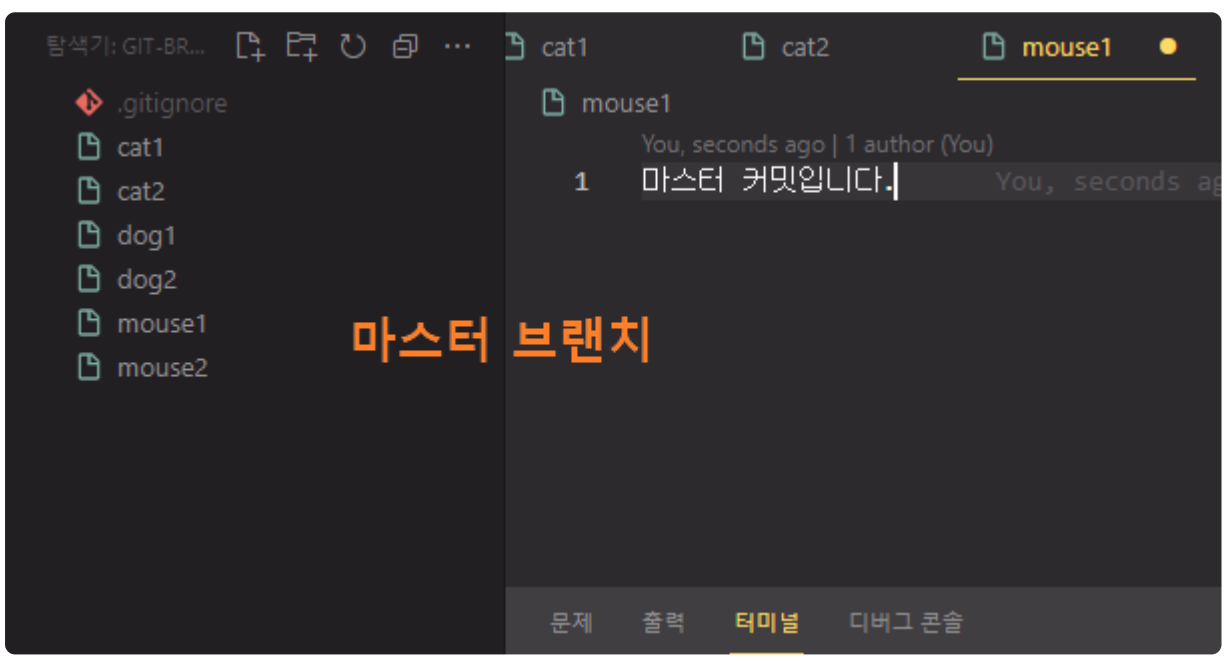
rebase는 원래 커밋 이력이 변경된다.



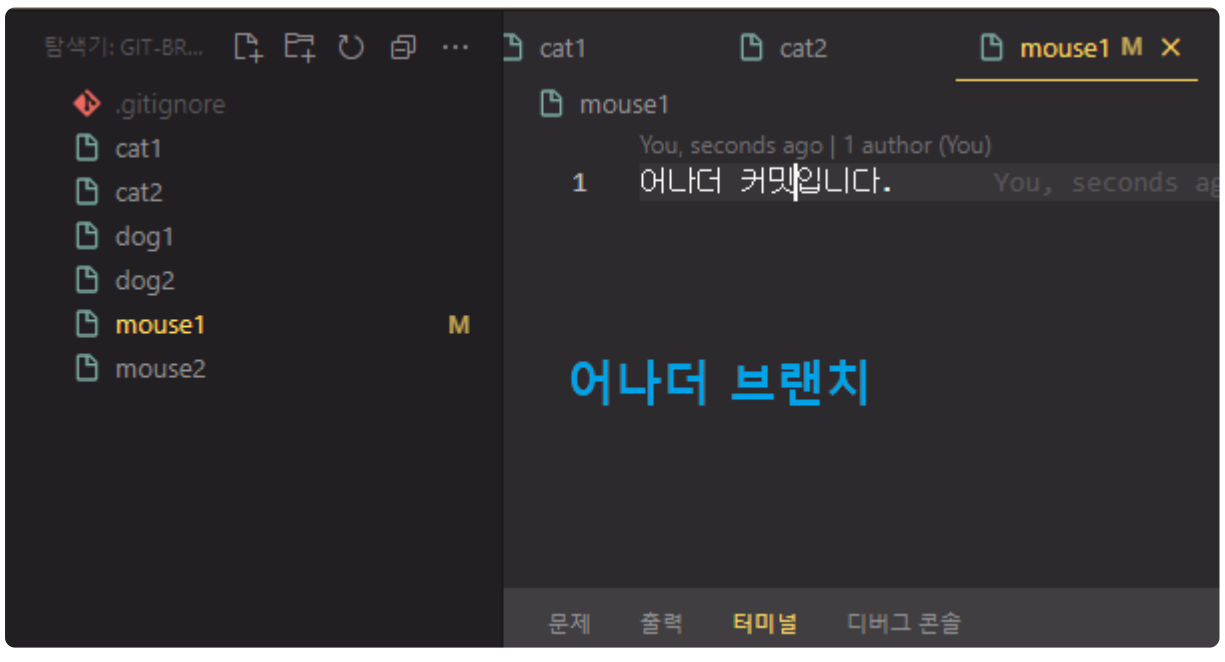
브랜치 충돌 해결 (merge vs rebase)

git merge 일 경우

master브랜치에서 mouse1 내용을 써주고 commit 해준다.



그리고 another이라는 브랜치를 만들고($ git branch another), mouse1 파일 내용을 써주고 commit 해준다..



그러면 다음과 같이 브랜치가 나뉘어지게 된다.

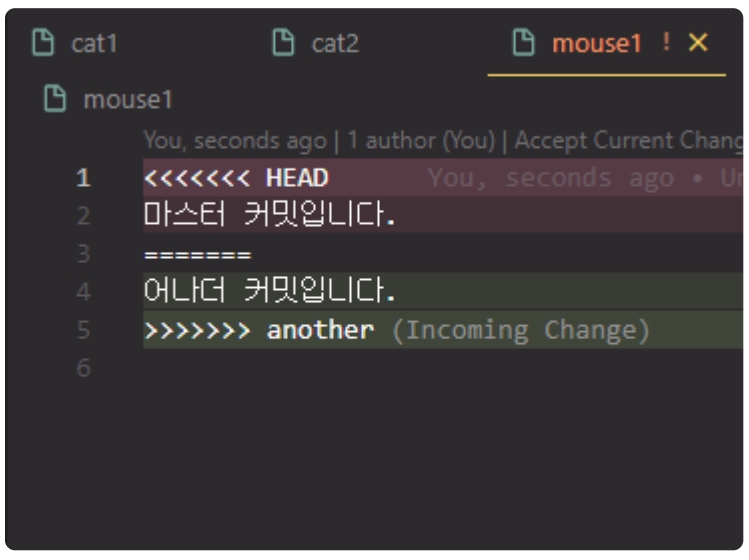
Graph	Description	Date	Author	Commit
	another 어나더 커밋	19 Nov 2021 18:29	inpa	5ea123a0
	master 마스터 커밋	19 Nov 2021 18:27	inpa	f3e969c0
	add mouse1, mouse2	19 Nov 2021 16:31	inpa	50cf9157
	dog add dog1, dog2	19 Nov 2021 17:10	inpa	c4e1920d
	add cat1, cat2	19 Nov 2021 16:26	inpa	621ff292

master브랜치에서 git merge another를 해주자.

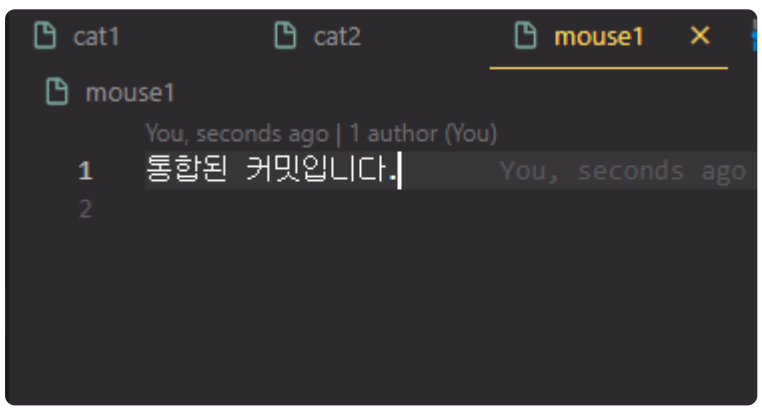
그러면 에러가 뜰 것이다.

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git merge another
Auto-merging mouse1
CONFLICT (content): Merge conflict in mouse1
Automatic merge failed; fix conflicts and then commit the result.
```

각각의 브랜치에서 같은 mouse1이라는 파일을 똑같은 라인에서 수정했으니, 당연한 결과이다.

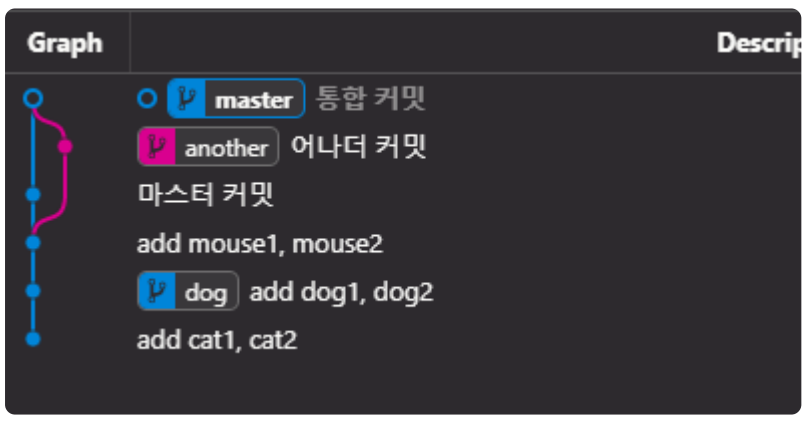


mouse1폴더에서 내용을 수정하자.



그리고 커밋을 하면,

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master|MERGING)
$ git commit  -am "통합 커밋"
[master 6c7e0da] 통합 커밋
```



다음과 같이 그래프가 하나로 merge된 모습을 볼수 있다.

git rebase 일 경우

그럼 같은 기능을 하는 rebase는 어떻게 될까?

위의 작업상황을 재현해보자.

BASH

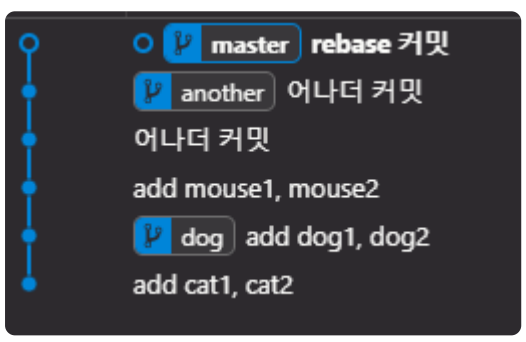
```
$ git rebase another # another브랜치를 합친다
$ git add . # 합치는 과정에서 충돌이 일어나면 내용 수정해주고 스테이징 해준다.
$ git rebase --continue # add하고 commit할 필요없다. 잠시 멈춘 rebase를 이어 나가면 알아서 된다.
```

```
MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master)
$ git rebase another
Auto-merging mouse1
CONFLICT (content): Merge conflict in mouse1
error: could not apply 56e1757... 마스터 커밋
hint: Resolve all conflicts manually, mark them as resolved with
hint: "git add/rm <conflicted_files>", then run "git rebase --continue".
hint: You can instead skip this commit: run "git rebase --skip".
hint: To abort and get back to the state before "git rebase", run "git rebase --abort".
Could not apply 56e1757... 마스터 커밋

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master|REBASE 1/1)
$ git rebase  --continue
mouse1: needs merge
You must edit all merge conflicts and then
mark them as resolved using git add

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master|REBASE 1/1)
$ git add mouse1

MS@DESKTOP-PQJEJIL MINGW64 ~/Desktop/공부/git-branch (master|REBASE 1/1)
$ git rebase  --continue
[detached HEAD ac39890] rebase 커밋
 1 file changed, 1 insertion(+), 1 deletion(-)
Successfully rebased and updated refs/heads/master.
```



merge와는 달리 아예 분기된 핑크색 그래프가 사라지고 **한줄로 쭉** 되는걸 볼수 있다.

어떤 사람들은 이 그래프 모양이 깔끔하다고 해서 rebase를 선호하기도 하지만, 사실 편한걸 쓰고싶은대로 쓰면 된다.