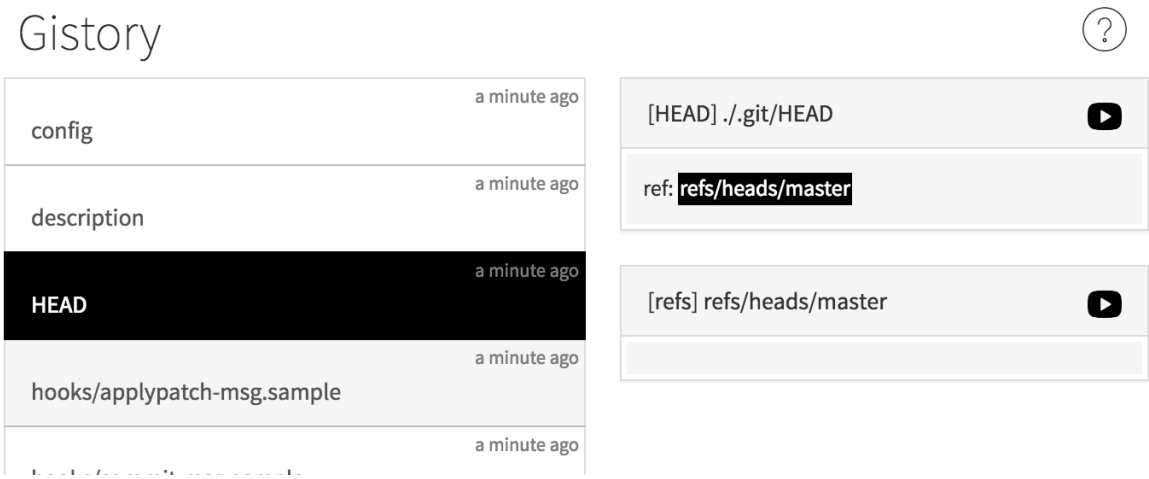
Git의 원리
1)branch의 원리

>git init 을 실행할경우

Gistory ?

config	a minute ago
description	a minute ago
HEAD	a minute ago
hooks/applypatch-msg.sample	a minute ago
hooks/commit-msg.sample	a minute ago
hooks/post-update.sample	a minute ago
hooks/pre-applypatch.sample	a minute ago
hooks/pre-commit.sample	a minute ago
hooks/pre-push.sample	a minute ago
hooks/pre-rebase.sample	a minute ago
hooks/prepare-commit-msg.sample	a minute ago
hooks/update.sample	a minute ago
info/exclude	a minute ago

여기서 HEAD는 git을 처음 생성했을때 만들어진다.



refs/heads/master 를 기억해두자.

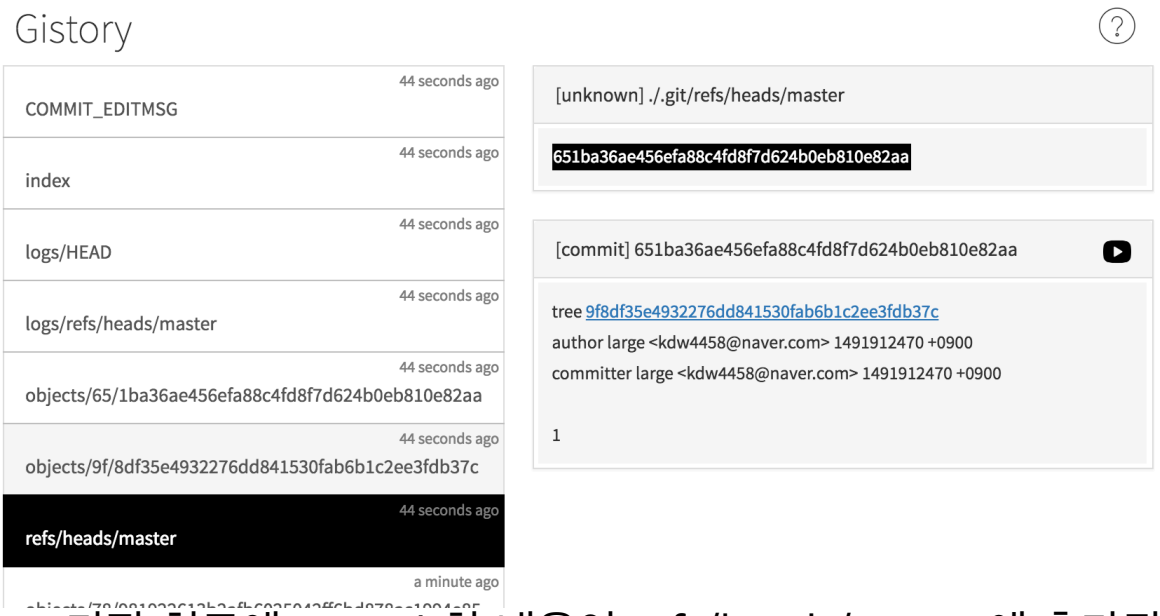
여기서 git을 commit 했을때 변화를 보면

>vim f1.txt //파일생성

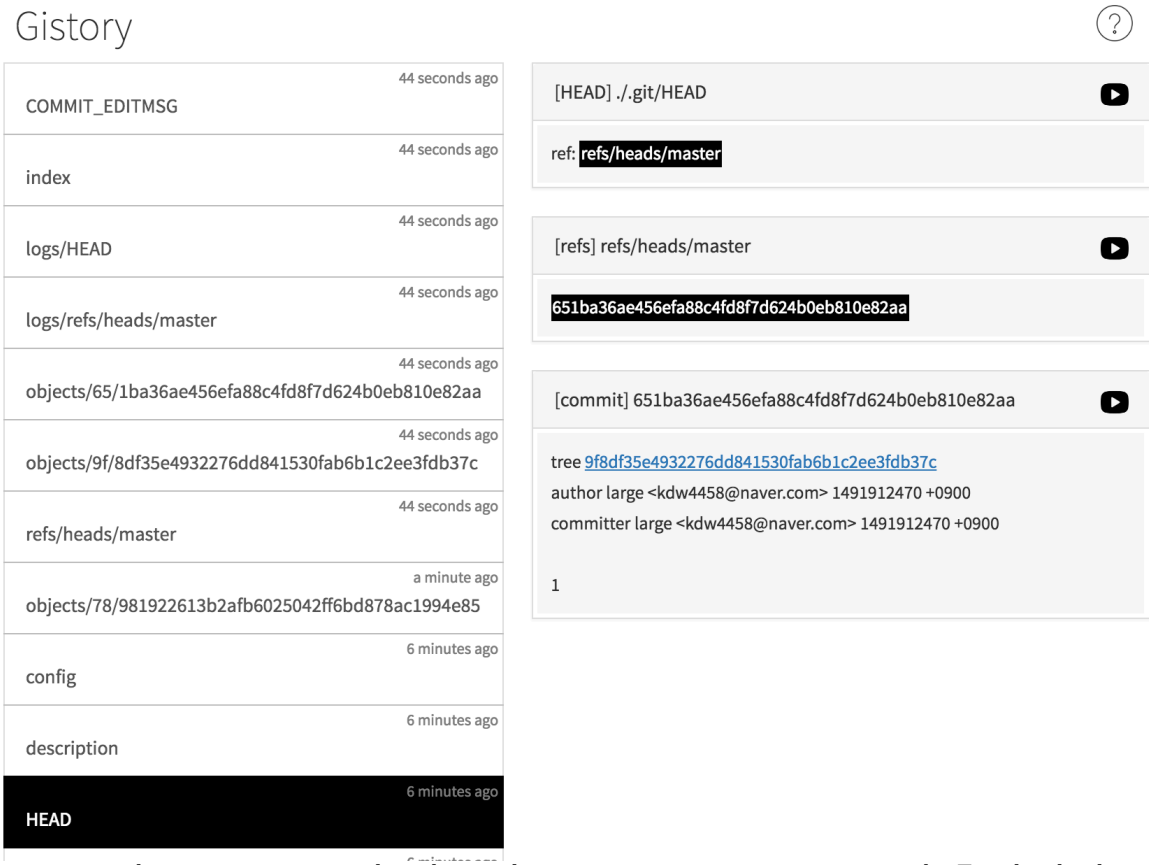
```
a
~
```

>git add f1.txt
>git commit -m "1"

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth5% git commit -m "1"
[master (root-commit) 651ba36] 1
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt
```



가장 최근에 commit한 내용이 refs/heads/master에 추가된 것을 볼 수 있다.



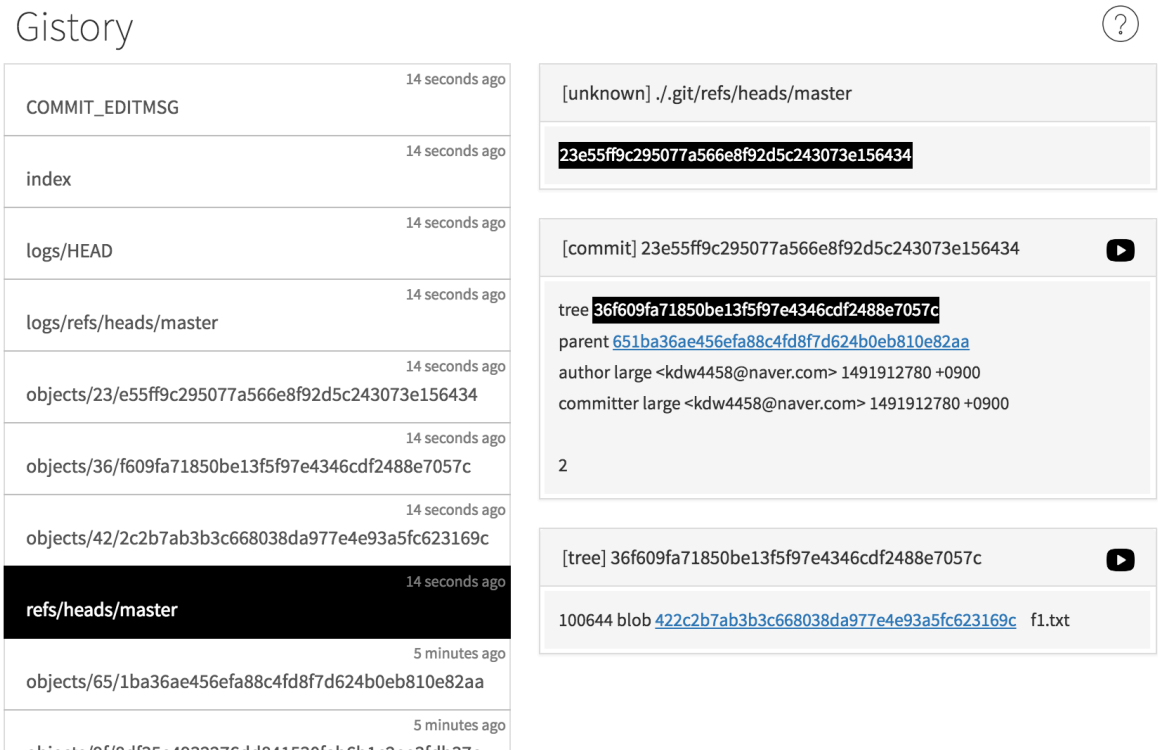
HEAD에도 commit한 내용이 refs/heads/master에 추가되었다!!

>vim f1.txt　　//파일내용 수정

```
a
b
~
```

>git commit -am "2"

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth5% git commit -am "2"
[master 23e55ff] 2
 1 file changed, 1 insertion(+)
```



방금 commit한 2번을 가리킨다.

Gistory

COMMIT_EDITMSG	14 seconds ago
index	14 seconds ago
logs/HEAD	14 seconds ago
logs/refs/heads/master	14 seconds ago
objects/23/e55ff9c295077a566e8f92d5c243073e156434	14 seconds ago
objects/36/f609fa71850be13f5f97e4346cdf2488e7057c	14 seconds ago
objects/42/2c2b7ab3b3c668038da977e4e93a5fc623169c	14 seconds ago
refs/heads/master	14 seconds ago
objects/65/1ba36ae456efa88c4fd8f7d624b0eb810e82aa	5 minutes ago
objects/9f/8df35e4932276dd841530fab6b1c2ee3fdb37c	5 minutes ago
objects/78/981922613b2afb6025042ff6bd878ac1994e85	5 minutes ago
config	11 minutes ago
description	11 minutes ago
HEAD	11 minutes ago
	11 minutes ago

[HEAD] ./.git/HEAD

ref: refs/heads/master

[refs] refs/heads/master

23e55ff9c295077a566e8f92d5c243073e156434

[commit] 23e55ff9c295077a566e8f92d5c243073e156434

tree 36f609fa71850be13f5f97e4346cdf2488e7057c
parent 651ba36ae456efa88c4fd8f7d624b0eb810e82aa
author large <kdw4458@naver.com> 1491912780 +0900
committer large <kdw4458@naver.com> 1491912780 +0900

2

HEAD의 master역시 가장 최근의 commit 2번을 가리킨다.

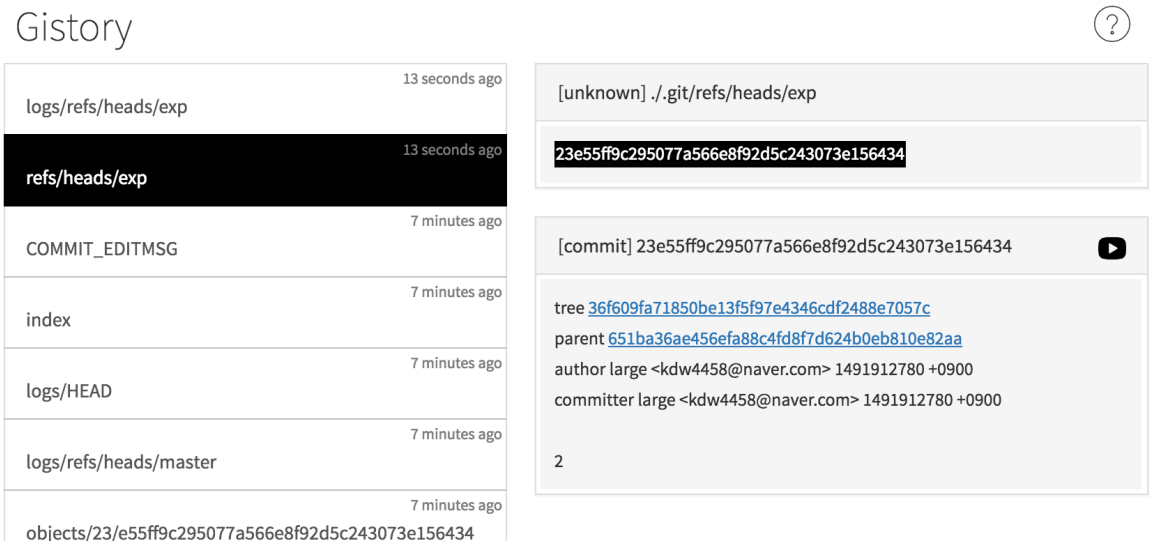
우리가
>git log

```
commit 23e55ff9c295077a566e8f92d5c243073e156434
Author: large <kdw4458@naver.com>
Date:   Tue Apr 11 21:13:00 2017 +0900

    2

commit 651ba36ae456efa88c4fd8f7d624b0eb810e82aa
Author: large <kdw4458@naver.com>
Date:   Tue Apr 11 21:07:50 2017 +0900

    1
```

로 2번이 가장 최신의 commit임을 알려주기위해서 위와같은 원리를 갖고 있는 것이다.

즉 git의 HEAD라는 파일을 갖고 있고, 그 HEAD는 refs/heads/master 라는 파일을 가리키고 있고,
그 refs/heads/master라는 파일은 가장 최신의 commit을 가리키고 있다.
git log 명령어는 이러한 순차적인 과정으로 바로위의 결과를 출력하는 것이다. 그리고 그이전의 것들은 parent로
탐색해 나갈 수 있다.
여기서 branch는 refs/heads/master 아래의 파일에 불과하다.

이제 branch를 생성해보자
>git branch exp



refs/heads/exp 라는 파일이 생성되었고, refs/heads/master와 같은 최신의 commit정보(2번)을 갖고있다.

>git branch

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth5% git branch
  exp
* master
```

여기서
>rm .git/refs/heads/exp //파일을 지우면
>git branch

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth5% rm .git/refs/heads/exp
```

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth5% git branch
* master
```

exp branch가 사라진 것을 알 수 있다.

>git log

```
commit 23e55ff9c295077a566e8f92d5c243073e156434
Author: large <kdw4458@naver.com>
Date:   Tue Apr 11 21:13:00 2017 +0900

    2

commit 651ba36ae456efa88c4fd8f7d624b0eb810e82aa
Author: large <kdw4458@naver.com>
Date:   Tue Apr 11 21:07:50 2017 +0900

    1
(END)
```

최신 commit 2번을 바탕으로

>vim .git/refs/heads/exp //생성

```
23e55ff9c295077a566e8f92d5c243073e156434
~
~
```

[esc] + :wq + [enter]

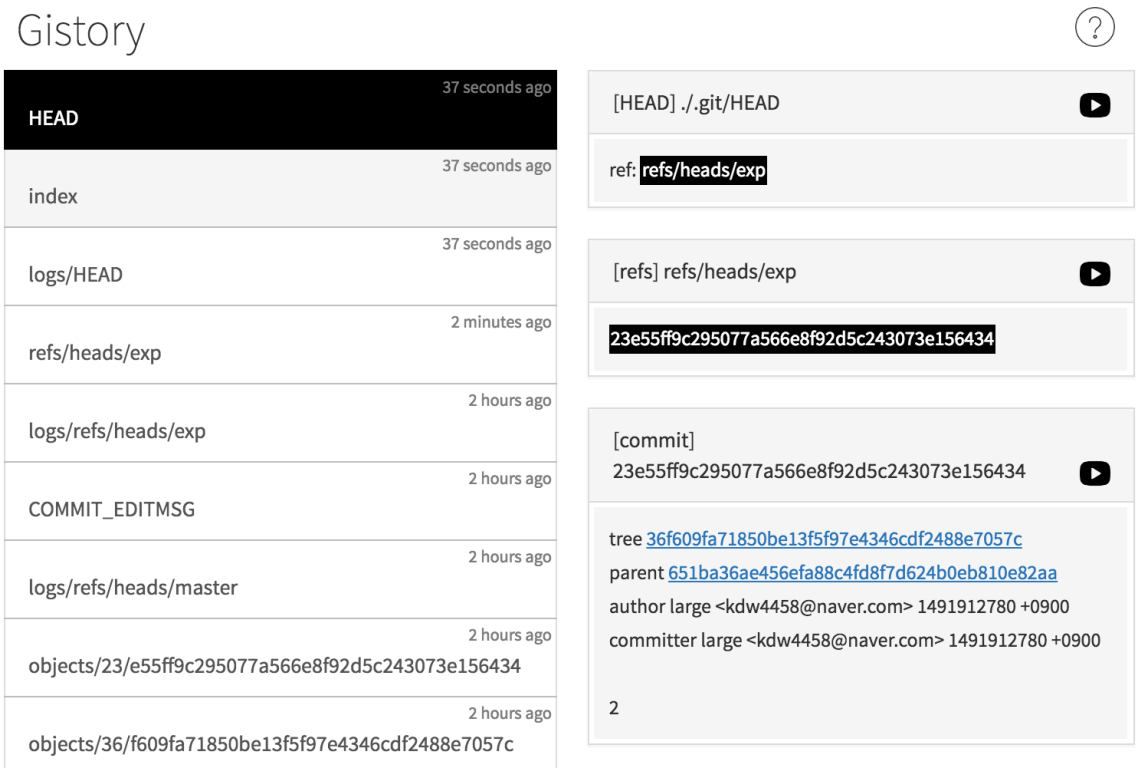
>git branch

```
gimdaewon@iMac:~/documents/gitfth5% git branch
  exp
* master
```

branch가 매우 중요한 역할을 하지만 일반 파일에 불과하다. 그것도 일반 텍스트파일..

>git checkout exp

```
gimdaewon@iMac:~/documents/gitfth5% git checkout exp
Switched to branch 'exp'
```



exp는 다시 2번 commit을 가리킨다

>vim f2.txt //생성

```
a
~
```

>git add f2.txt

>git commit -m '3'

```
gimdaewon@iMac:~/documents/gitfth5% git commit -m '3'
[exp b1176f9] 3
 Committer: 김대원 <gimdaewon@iMac.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
 create mode 100644 f2.txt
```

Gistory

COMMIT_EDITMSG	a minute ago
index	a minute ago
logs/HEAD	a minute ago
logs/refs/heads/exp	a minute ago
objects/76/9a8040f07a85bb8d791df566043fbb698ddfb1	a minute ago
objects/b1/176f94abf094a6ffdb061f4c7a1bc9386d8c65	a minute ago
refs/heads/exp	a minute ago
objects/78/981922613b2afb6025042ff6bd878ac1994e85	a minute ago
HEAD	4 minutes ago
	2 hours ago

[HEAD] ./.git/HEAD

ref: refs/heads/exp

[refs] refs/heads/exp

b1176f94abf094a6ffdb061f4c7a1bc9386d8c65

[commit] b1176f94abf094a6ffdb061f4c7a1bc9386d8c65

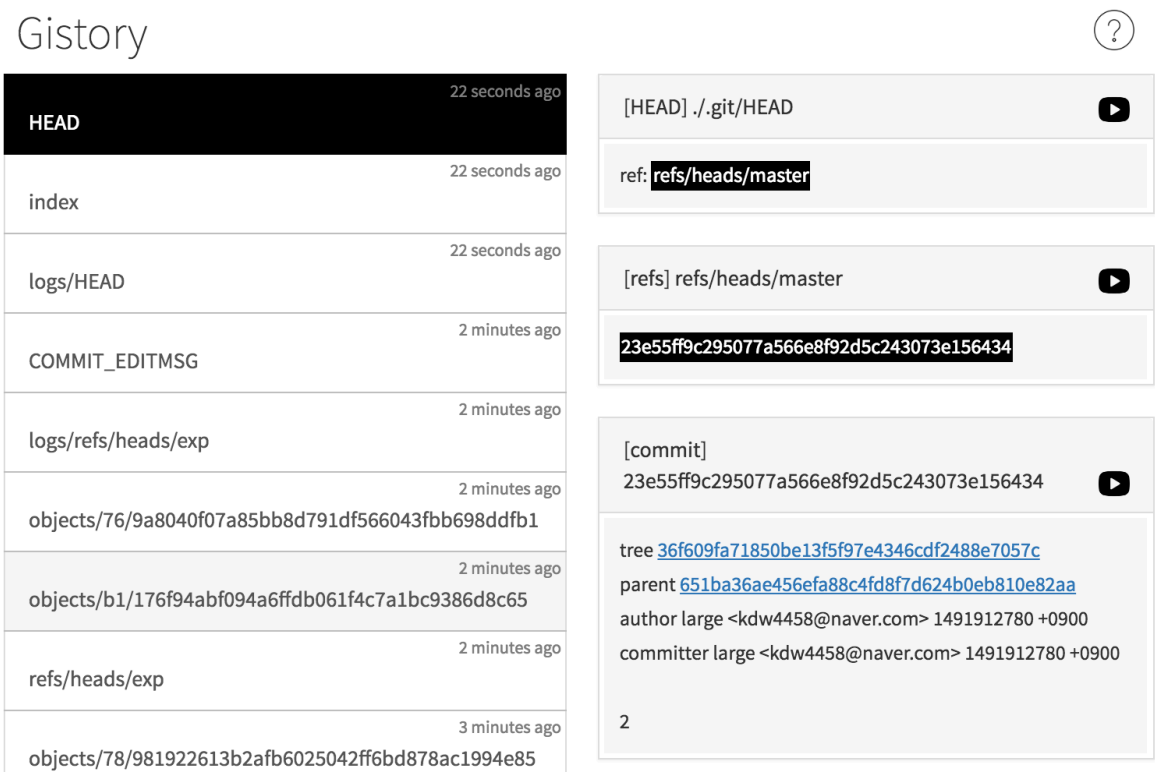
tree 769a8040f07a85bb8d791df566043fbb698ddfb1
parent 23e55ff9c295077a566e8f92d5c243073e156434
author 김대원 <gimdaewon@iMac.local> 1491921281 +0900
committer 김대원 <gimdaewon@iMac.local> 1491921281 +0900

3

exp는 방금 commit 했던 3번을 가리킨다.

>git checkout master

```
gimdaewon@iMac:~/documents/gitfth5% git checkout master
Switched to branch 'master'
```



HEAD 파일은 master를 가리키고 이 master의 최신 commit 은 2번이다

>ls -al

```
gimdaewon@iMac:~/documents/gitfth5% ls -al
total 8
drwxr-xr-x@  4 gimdaewon  staff   136  4 11 23:37 .
drwx------+ 50 gimdaewon  staff  1700  4 11 23:21 ..
drwxr-xr-x@ 13 gimdaewon  staff   442  4 11 23:37 .git
-rw-r--r--@  1 gimdaewon  staff     4  4 11 21:12 f1.txt
```

git을 만드는 순간부터 HEAD라는 파일이 만들어지고 refs/heads/master 파일이 생성되고 저장소(git)이 만들어지는 순간, branch라는 요소는 중요한 요소로 작동된다.

>git log --branches --decorate --graph

```
* commit b1176f94abf094a6ffdb061f4c7a1bc9386d8c65 (exp)
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 23:34:41 2017 +0900
|
|     3
|
* commit 23e55ff9c295077a566e8f92d5c243073e156434 (HEAD -> master)
```

```
| Author: large <kdw4458@naver.com>
| Date:   Tue Apr 11 21:13:00 2017 +0900
|
|     2
|
* commit 651ba36ae456efa88c4fd8f7d624b0eb810e82aa
  Author: large <kdw4458@naver.com>
  Date:   Tue Apr 11 21:07:50 2017 +0900

      1
```

현재 우리가 사용하고있는(checkout한) branch가 무엇인가 알려주는 중요한 요소이고, 내부적으로 특정한 파일이다.

2)reset과 checkout의 원리

>git init

```
gimdaewon@iMac:~/documents/gitfth6% git init
Initialized empty Git repository in /Users/gimdaewon/Documents/
gitfth6/.git/
```

>vim f1.txt //파일생성

```
a
~
```

[esc] + :wq + [enter]

>git add f1.txt

>git commit -m '1'

```
gimdaewon@iMac:~/documents/gitfth6% git commit -m '1'
[master (root-commit) 732680e] 1
 Committer: 김대원 <gimdaewon@iMac.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
```

```
 create mode 100644 f1.txt
```

>vim f1.txt //수정

```
a
b
~
```

[esc] + :wq + [enter]

>git commit -am '2'

```
gimdaewon@iMac:~/documents/gitfth6% git commit -am '2'
[master ca245ab] 2
 Committer: 김대원 <gimdaewon@iMac.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
```

>vim f1.txt //수정

```
a
b
c
~
```

[esc] + :wq + [enter]

>git commit -am '3'

```
gimdaewon@iMac:~/documents/gitfth6% git commit -am '3'
[master d64af6c] 3
 Committer: 김대원 <gimdaewon@iMac.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit
```

```
After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
```

>vim f1.txt //수정

```
a
b
c
d
~
```

[esc] + :wq + [enter]

>git commit -am '4'

```
gimdaewon@iMac:~/documents/gitfth6% git commit -am '4'
[master 7b884cf] 4
 Committer: 김대원 <gimdaewon@iMac.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

 1 file changed, 1 insertion(+)
```

>git log

```
commit 7b884cf66ffe4ffc10a341855b6d4fe9069a2e8c
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:56:27 2017 +0900

    4

commit d64af6cfca002c02b63ca8ca61771bc41d920909
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:55:07 2017 +0900

    3

commit ca245ab28c3537807c20e8ce33837df4af529f30
```

```
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:53:43 2017 +0900

    2

commit 732680ee9a808d88289d98e62d44d7110653f3d0
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:52:17 2017 +0900

    1
```

이때 commit id가 `d64af6cfca002c02b63ca8ca61771bc41d920909`인(3번) 곳으로 돌아가고 싶고 그위의 commit(`7b884cf66ffe4ffc10a341855b6d4fe9069a2e8c`) 4번을 삭제하고 싶을경우

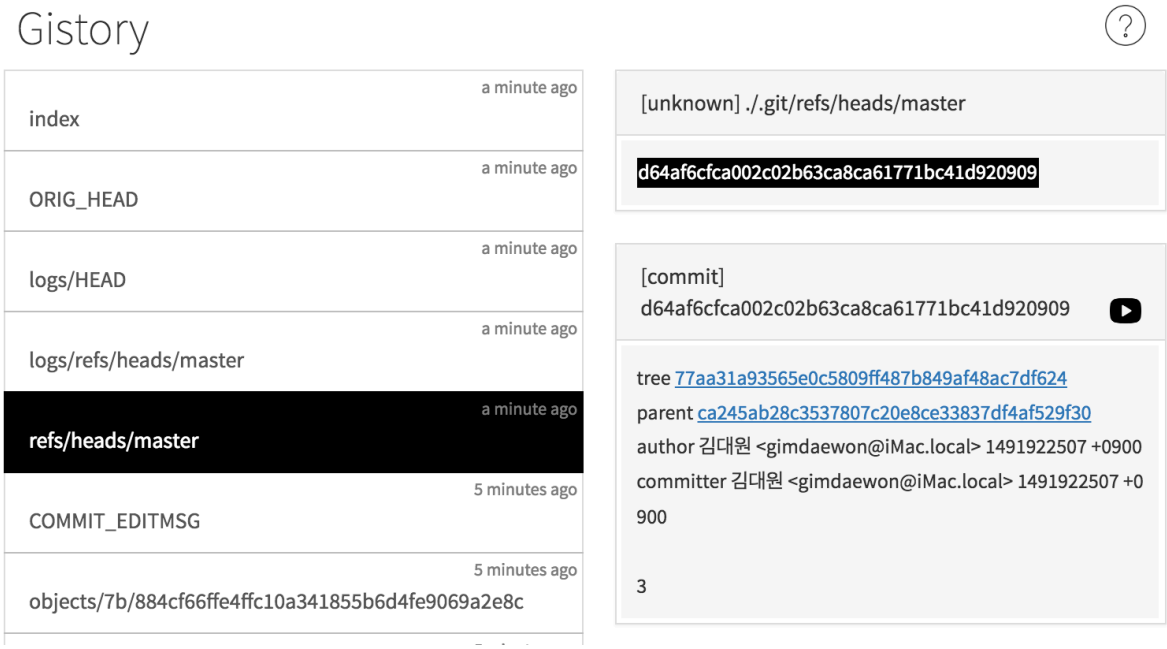
>git reset --hard `d64af6cfca002c02b63ca8ca61771bc41d920909` //3번의 commit id

```
gimdaewon@iMac:~/documents/gitfth6% git reset --hard
d64af6cfca002c02b63ca8ca61771bc41d920909
HEAD is now at d64af6c 3
```

>git log

```
commit d64af6cfca002c02b63ca8ca61771bc41d920909
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:55:07 2017 +0900

    3

commit ca245ab28c3537807c20e8ce33837df4af529f30
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:53:43 2017 +0900

    2

commit 732680ee9a808d88289d98e62d44d7110653f3d0
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:52:17 2017 +0900

    1
```

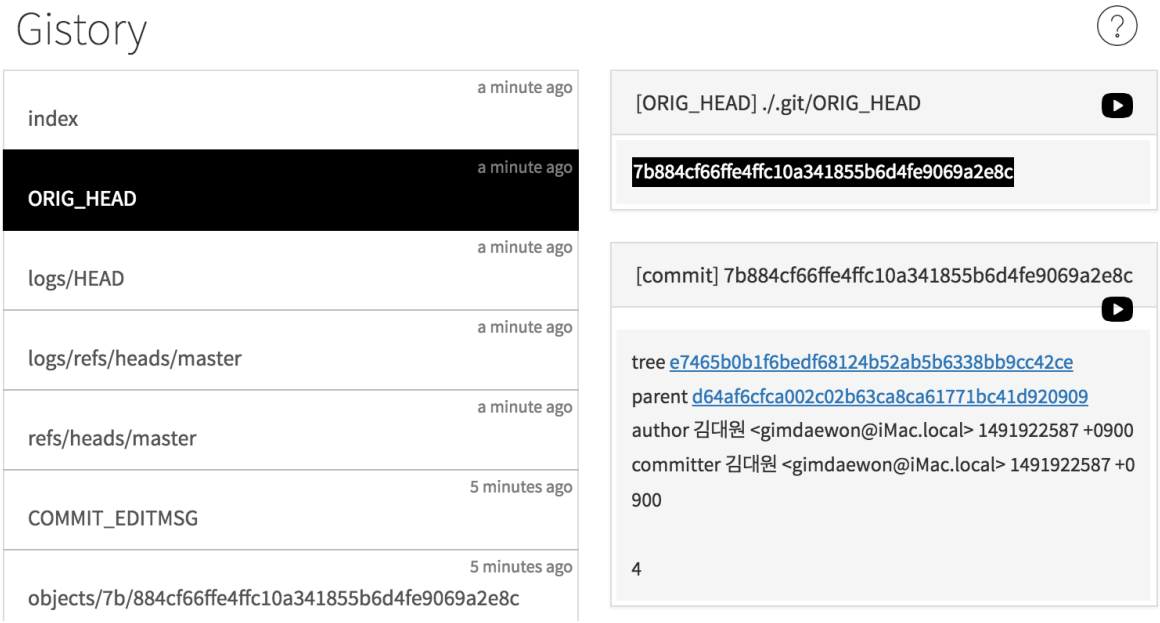
현재 나의 branch의 최신 commit 은 3번이 되었다.



현재 나의 branch(master)는 3번 (d64af6cfca002c02b63ca8ca61771bc41d920909)을 가리킴을 알 수 있다.

그럼 우리가 지웠던 4번 commit은 지워진 것일까 ? git은 왠만해서는 파일을 지우지않는다

(크기가 커질경우 가비지콜렉터가 지우는 경우가 있지만 드물다)



우리가 삭제한 4번이 요기있네 ?

Gistory

index	a minute ago
ORIG_HEAD	a minute ago
logs/HEAD	a minute ago
logs/refs/heads/master	a minute ago
refs/heads/master	a minute ago
COMMIT_EDITMSG	5 minutes ago
objects/7b/884cf66ffe4ffc10a341855b6d4fe9069a2e8c	5 minutes ago
objects/d6/8dd4031d2ad5b7a3829ad7df6635e27a7daa22	5 minutes ago
objects/e7/465b0b1f6bedf68124b52ab5b6338bb9cc42ce	5 minutes ago
objects/77/aa31a93565e0c5809ff487b849af48ac7df624	7 minutes ago
objects/d6/4af6cfca002c02b63ca8ca61771bc41d920909	7 minutes ago
objects/de/980441c3ab03a8c07dda1ad27b8a11f39deb1e	7 minutes ago

요기두 있네?

[logs] ./.git/logs/refs/heads/master

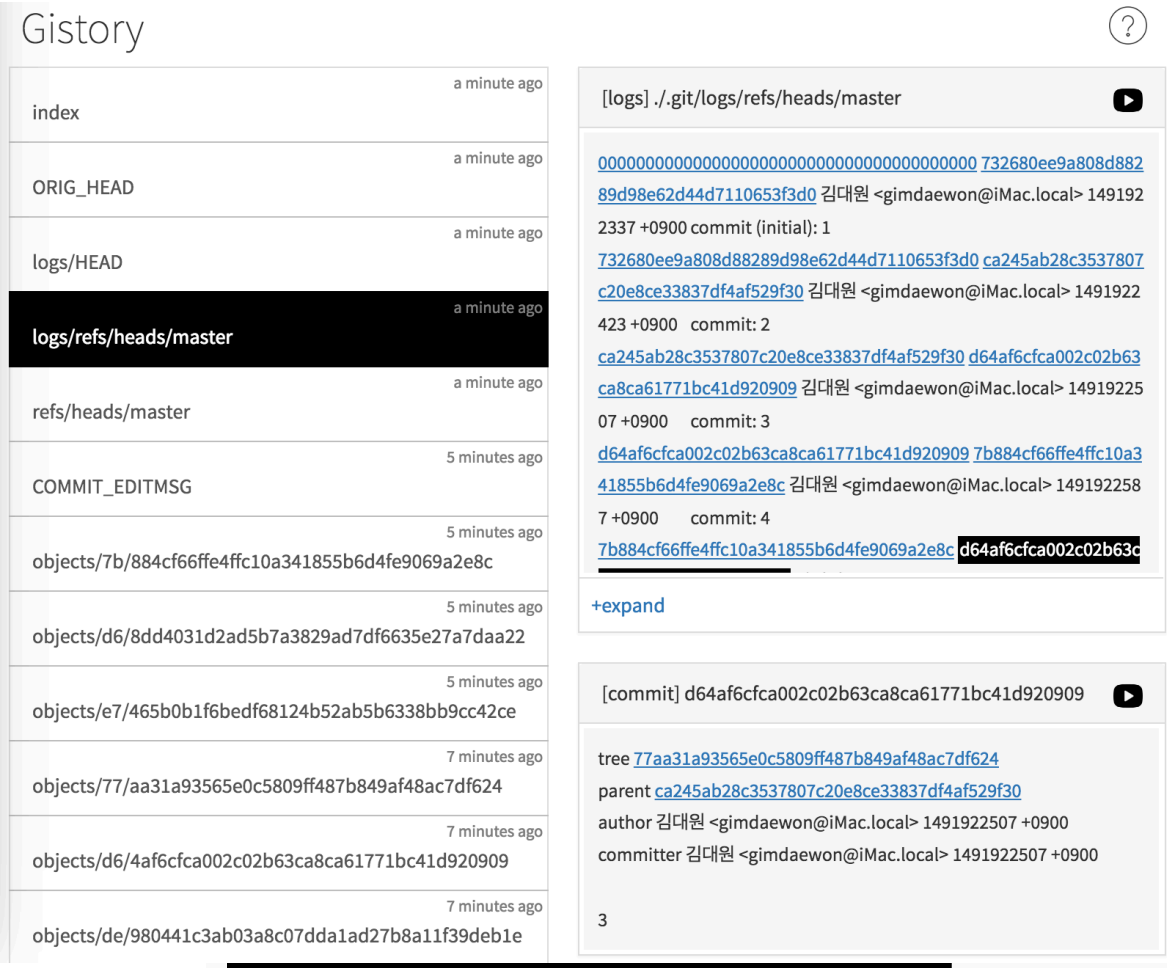
0000000000000000000000000000000000000000 732680ee9a808d88289d98e62d44d7110653f3d0 김대원 <gimdaewon@iMac.local> 1491922337 +0900 commit (initial): 1
732680ee9a808d88289d98e62d44d7110653f3d0 ca245ab28c3537807c20e8ce33837df4af529f30 김대원 <gimdaewon@iMac.local> 1491922423 +0900 commit: 2
ca245ab28c3537807c20e8ce33837df4af529f30 d64af6cfca002c02b63ca8ca61771bc41d920909 김대원 <gimdaewon@iMac.local> 1491922507 +0900 commit: 3
d64af6cfca002c02b63ca8ca61771bc41d920909 7b884cf66ffe4ffc10a341855b6d4fe9069a2e8c 김대원 <gimdaewon@iMac.local> 1491922587 +0900 commit: 4
7b884cf66ffe4ffc10a341855b6d4fe9069a2e8c d64af6cfca002c02b63c

+expand

[commit] 7b884cf66ffe4ffc10a341855b6d4fe9069a2e8c

tree e7465b0b1f6bedf68124b52ab5b6338bb9cc42ce
parent d64af6cfca002c02b63ca8ca61771bc41d920909
author 김대원 <gimdaewon@iMac.local> 1491922587 +0900
committer 김대원 <gimdaewon@iMac.local> 1491922587 +0900

4



그옆에 d64af6cfca002c02b63ca8ca61771bc41d920909은 변경한 현재의 commit 이다

HEAD는 위험한 명령 reset을 실행하기전에 지우려는 commit을 ORIG_HEAD파일에 저장하고,
reset명령을 실행한다.
이로인해 ORIG_HEAD 명령으로 다시 되돌릴 수 있다.

>git reset --hard ORIG_HEAD

```
gimdaewon@iMac:~/documents/gitfth6% git reset --hard ORIG_HEAD
HEAD is now at 7b884cf 4
```

>git log

```
commit 7b884cf66ffe4ffc10a341855b6d4fe9069a2e8c
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:56:27 2017 +0900

    4

commit d64af6cfca002c02b63ca8ca61771bc41d920909
```

```
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:55:07 2017 +0900

    3

commit ca245ab28c3537807c20e8ce33837df4af529f30
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:53:43 2017 +0900

    2

commit 732680ee9a808d88289d98e62d44d7110653f3d0
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 23:52:17 2017 +0900

    1
```

reset이 취소되었다!!!!

>git reflog

```
7b884cf HEAD@{0}: reset: moving to ORIG_HEAD
d64af6c HEAD@{1}: reset: moving to d64af6cfca002c02b63ca8ca61771bc41d920909
7b884cf HEAD@{2}: commit: 4
d64af6c HEAD@{3}: commit: 3
ca245ab HEAD@{4}: commit: 2
732680e HEAD@{5}: commit (initial): 1
```

내가했던 명령들이 나열된다. 여기서 추가적으로 >git reset `7b884cf` 또는 >git reset `HEAD@{0}` 을 입력했을때 그 명령자체가 취소된다.

git checkout exp

>git checkout `d64af6cfca002c02b63ca8ca61771bc41d920909` // `d64af6c` `HEAD@{1}` 명령에 branch 생성

```
gimdaewon@iMac:~/documents/gitfth6% git checkout
d64af6cfca002c02b63ca8ca61771bc41d920909
Note: checking out 'd64af6cfca002c02b63ca8ca61771bc41d920909'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>
```

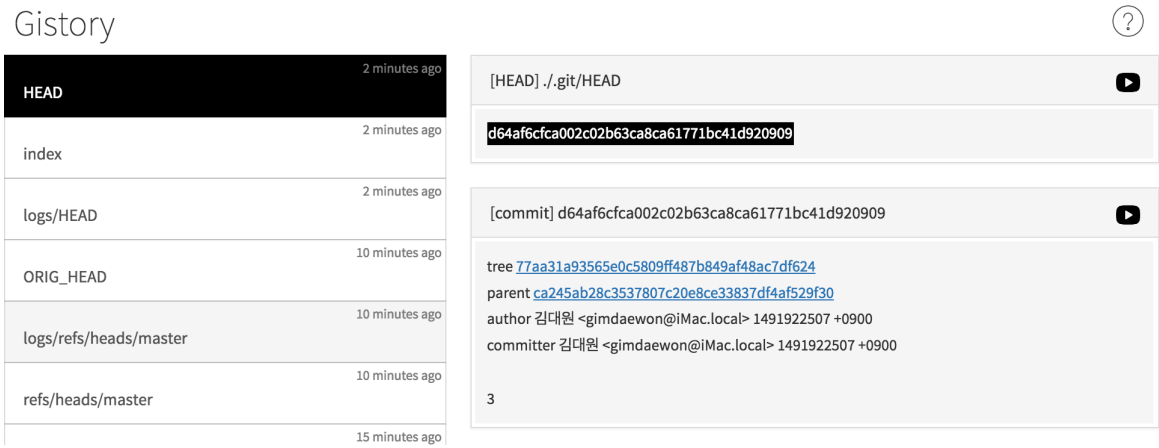
```
HEAD is now at d64af6c... 3
```

commit id로 직접 branch를 생성하는 것이 가능하다.

>git branch

```
gimdaewon@iMac:~/documents/gitfth6% git branch
* (HEAD detached at d64af6c)
  master
```

branch가 생성되고 switch된다.

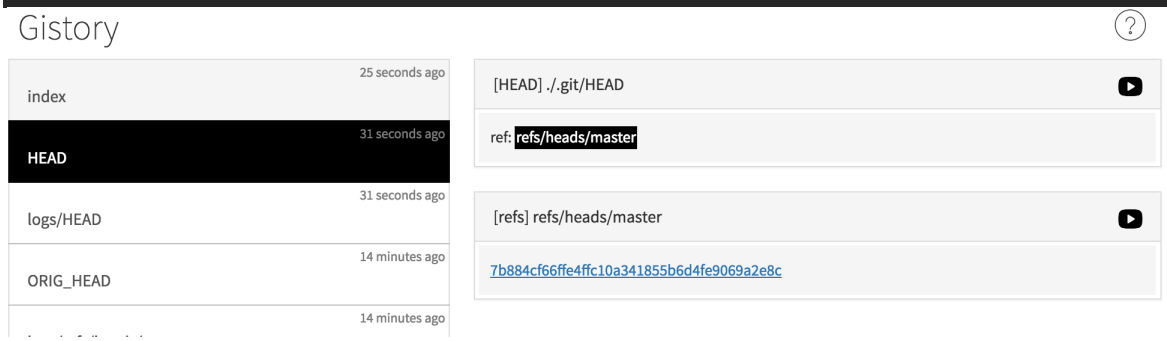


지금의 HEAD는 어떤 특정한 commit의 id를 직접 가리킨다. (detached 되어있는상태)

지금 맥락에서는 쓸모가 없음...ㅜㅠ..(그럼 왜 지금 알려주는 것이여...)

>git checkout master

```
gimdaewon@iMac:~/documents/gitfth6% git checkout master
Previous HEAD position was d64af6c... 3
Switched to branch 'master'
```



3)reset으로 알아보는 working copy, index,

repository

전부 이해하지 않아도 좋다!!

	working directory working tree working copy	index staging area cache	repository history tree
			git reset --soft
		git reset --mixed	
	git reset --hard		

>git reset --help

```
           --soft
               Does not touch the index file or the working tree at all
(but resets the head
               to <commit>, just like all modes do). This leaves all your
changed files
               "Changes to be committed", as git status would put it.

           --mixed
               Resets the index but not the working tree (i.e., the changed
files are
               preserved but not marked for commit) and reports what has
not been updated.
               This is the default action.

               If -N is specified, removed paths are marked as intent-to-
add (see git-add(1)).

           --hard
               Resets the index and working tree. Any changes to tracked
files in the working
               tree since <commit> are discarded.
```

>git init

```
gimdaewon@iMac:~/documents/gitfth7% git init
Initialized empty Git repository in /Users/gimdaewon/Documents/
gitfth7/.git/
```

>vim f1.txt

```
init
~
~
```

[esc] + :wq + [enter]

>git add f1.txt
>git commit -m '1'

>vim f1.txt //수정

```
repository
~
~
```

>git add f1.txt
>git commit -m '2'

	working directory working tree working copy	index staging area cache	repository history tree
			git reset --soft
			git reset --mixed
			git reset --hard
f1.txt	repository	repository	repository

현재상태

>cat f1.txt

```
gimdaewon@iMac:~/documents/gitfth7% cat f1.txt
repository
```

Gistory

3 minutes ago COMMIT_EDITMSG	[index] index
3 minutes ago **index**	100644 09d06c8f2087539716887077f2c0f8df5ab0ee24 0 f1.txt
3 minutes ago logs/HEAD	[blob] 09d06c8f2087539716887077f2c0f8df5ab0ee24
3 minutes ago logs/refs/heads/master	repository
3 minutes ago objects/0f/a066e062858f51889b194e4819294ff6a00eb2	f1.txt (This information was not included the original message)

Gistory

3 minutes ago COMMIT_EDITMSG	[unknown] ./.git/refs/heads/master
3 minutes ago index	0fa066e062858f51889b194e4819294ff6a00eb2
3 minutes ago logs/HEAD	[commit] 0fa066e062858f51889b194e4819294ff6a00eb2
3 minutes ago logs/refs/heads/master	tree 7b60d3bfdfa15b16c74154cd889a52fd089c5117 parent aba8d9d2d26e69ab220d42fa9290abd5440a9066 author 김대원 <gimdaewon@iMac.local> 1491925604 +0900 committer 김대원 <gimdaewon@iMac.local> 1491925604 +0900 2
3 minutes ago objects/0f/a066e062858f51889b194e4819294ff6a00eb2	
3 minutes ago objects/7b/60d3bfdfa15b16c74154cd889a52fd089c5117	
3 minutes ago **refs/heads/master**	[tree] 7b60d3bfdfa15b16c74154cd889a52fd089c5117
3 minutes ago objects/09/d06c8f2087539716887077f2c0f8df5ab0ee24	100644 blob 09d06c8f2087539716887077f2c0f8df5ab0ee24 f 1.txt
4 minutes ago objects/5e/1d0348228d61e954384a2fba74c4c3698970f5	
4 minutes ago objects/ab/a8d9d2d26e69ab220d42fa9290abd5440a9066	[blob] 09d06c8f2087539716887077f2c0f8df5ab0ee24
5 minutes ago objects/b1/b716105590454bfc4c0247f193a04088f39c7f	repository
6 minutes ago config	f1.txt (This information was not included the original message)

위 3가지의 f1.txt는 모두같은 repository 이다

>vim f1.txt //수정

```
index
~
~
```

[esc] + :wq + [enter]

>git add f1.txt

>git status

```
gimdaewon@iMac:~/documents/gitfth7% git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

     modified:   f1.txt
```

아직 commit 하지 않은 상태

	working directory working tree working copy	index staging area cache	repository history tree
			git reset --soft
			git reset --mixed
			git reset --hard
f1.txt	index	index	repository

>vim f1.txt

```
working copy
~
~
```

	working directory working tree working copy	index staging area cache	repository history tree
			git reset --soft
			git reset --mixed
			git reset --hard
f1.txt	working copy	index	repository

자이제 reset을 해보자

>git log

```
commit 0fa066e062858f51889b194e4819294ff6a00eb2
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 00:46:44 2017 +0900

    2

commit aba8d9d2d26e69ab220d42fa9290abd5440a9066
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 00:45:09 2017 +0900

    1
```

>git reset --soft `aba8d9d2d26e69ab220d42fa9290abd5440a9066`

>git log

```
commit aba8d9d2d26e69ab220d42fa9290abd5440a9066
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 00:45:09 2017 +0900

    1
```

>git log -p

```
commit aba8d9d2d26e69ab220d42fa9290abd5440a9066
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 00:45:09 2017 +0900

    1
```

```
diff --git a/f1.txt b/f1.txt
new file mode 100644
index 0000000..b1b7161
--- /dev/null
+++ b/f1.txt
@@ -0,0 +1 @@
+init
```

f1.txt가 우리가 처음 commit 했던 init(+init)이라고 하는 text를 갖고있다.

	working directory working tree working copy	index staging area cache	repository history tree
			git reset --soft
			git reset --mixed
			git reset --hard
f1.txt	working copy	index	init

repository의 내용만 초기화가 된다.

>git diff

```
diff --git a/f1.txt b/f1.txt
index 9015a7a..1116dba 100644
--- a/f1.txt
+++ b/f1.txt
@@ -1 +1 @@
-index
+working copy
```

git diff만 하게되면 working copy(working directory,working tree)의 내용과 index(staging area, cache)의 내용을 비교하는 명령이 된다. 바로 위를 결과를 통해 알 수 있는것은. working copy와 index는 유지됨을 알 수 있따.

>git reset --soft ORIG_HEAD //repository(history, tree)의 내용을 다시 돌리기

//그냥 reset을 하게되면 mixed로 된다.

// (index, repository영역 모두 repository 된다.)

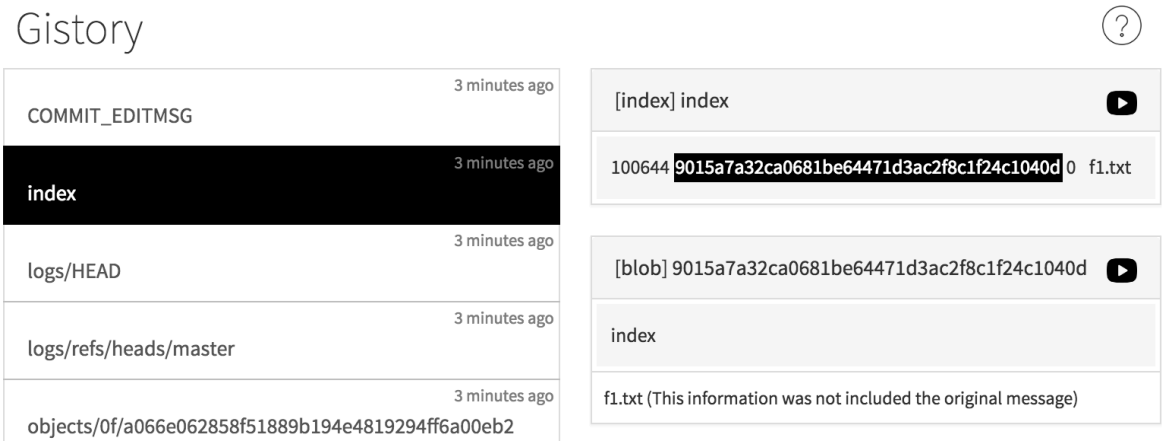
>git status

```
gimdaewon@iMac:~/documents/gitfth7% git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

     modified:   f1.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

     modified:   f1.txt
```

>cat f1.txt

```
gimdaewon@iMac:~/documents/gitfth7% cat f1.txt
working copy
```

working directory의 내용(변경)



index 영역의 내용(add)

	working directory working tree working copy	index staging area cache	repository history tree
			git reset --soft
			git reset --mixed
			git reset --hard
f1.txt	working copy	index	repository

현재상태

>git log

```
commit 0fa066e062858f51889b194e4819294ff6a00eb2
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 00:46:44 2017 +0900

    2

commit aba8d9d2d26e69ab220d42fa9290abd5440a9066
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 00:45:09 2017 +0900

    1
```

>git reset --mixed aba8d9d2d26e69ab220d42fa9290abd5440a9066

```
gimdaewon@iMac:~/documents/gitfth7% git reset --mixed
aba8d9d2d26e69ab220d42fa9290abd5440a9066
Unstaged changes after reset:
M    f1.txt
```

>git diff

```
diff --git a/f1.txt b/f1.txt
index b1b7161..1116dba 100644
--- a/f1.txt
+++ b/f1.txt
@@ -1 +1 @@
-init
+working copy
(END)
```

	working directory working tree working copy	index staging area cache	repository history tree
			git reset --soft
			git reset --mixed
			git reset --hard
f1.txt	working copy	init	init

>git reset --hard

```
gimdaewon@iMac:~/documents/gitfth7% git reset --hard
HEAD is now at aba8d9d 1
```

>git status

```
gimdaewon@iMac:~/documents/gitfth7% git status
On branch master
nothing to commit, working tree clean
```

>cat f1.txt

```
gimdaewon@iMac:~/documents/gitfth7% cat f1.txt
init
```

세영역 모두 f1.txt의 내용이 init이 된다.

4)merge & conflict의 원리

>git init

>vim f1.txt //생성

```
function(){
        return "common";
}
~
~
```

>git add f1.txt
>git commit -m '1'

>git checkout -b exp //exp라는 branch를 만들며 exp branch로 switch

```
gimdaewon@iMac:~/documents/gitfth8% git checkout -b exp
Switched to a new branch 'exp'
```

>vim f1.txt

```
function(){
        return "exp";
}
~
~
```

>git commit -am 'common->exp'

>git checkout master

```
gimdaewon@iMac:~/documents/gitfth8% git checkout master
Switched to branch 'master'
```

>vim f1.txt

```
function(){
        return "master";
}
~
~
```

>git commit -am 'common->master'

자이제 갈등이 시작된다....

>git merge exp

```
gimdaewon@iMac:~/documents/gitfth8% git merge exp
Auto-merging f1.txt
CONFLICT (content): Merge conflict in f1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

자동 merge 실패..

이런 충돌상황에서 git의 내부에는 어떤일이 벌어지는가 ?

Gistory

index	a minute ago
MERGE_HEAD	a minute ago
MERGE_MODE	a minute ago
MERGE_MSG	a minute ago
ORIG_HEAD	a minute ago
objects/6a/87af3c40c1b86f1ce5604305bc7b5849af6321	a minute ago

몇개의 파일이 생성되었고

Gistory

index	a minute ago
MERGE_HEAD	a minute ago
MERGE_MODE	a minute ago
MERGE_MSG	a minute ago
ORIG_HEAD	a minute ago
objects/6a/87af3c40c1b86f1ce5604305bc7b5849af6321	a minute ago
	3 minutes ago

[index] index

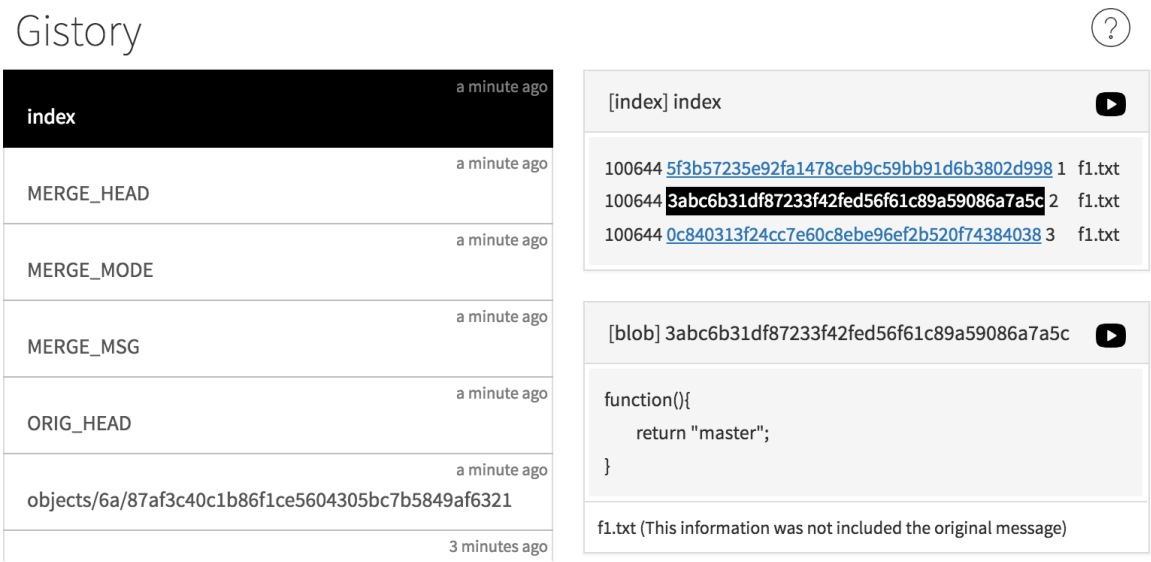
```
100644 5f3b57235e92fa1478ceb9c59bb91d6b3802d998 1 f1.txt
100644 3abc6b31df87233f42fed56f61c89a59086a7a5c 2 f1.txt
100644 0c840313f24cc7e60c8ebe96ef2b520f74384038 3 f1.txt
```

[blob] 5f3b57235e92fa1478ceb9c59bb91d6b3802d998

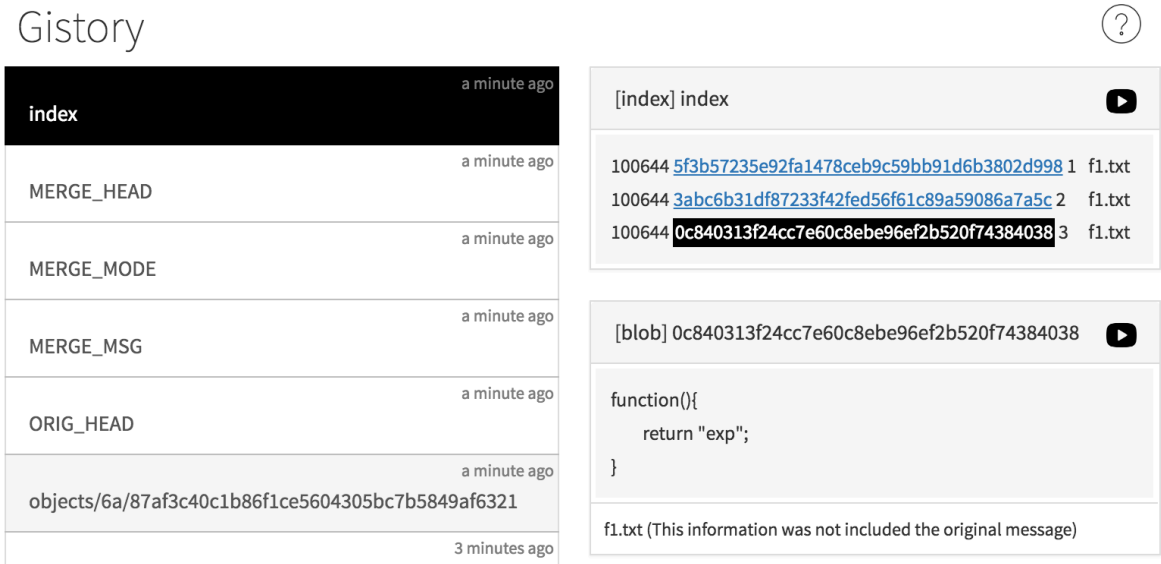
```
function(){
    return "common";
}
```

f1.txt (This information was not included the original message)

master와 exp가 갖고있었던 공통의 f1.txt



master의 변경된 f1.txt



exp의 변경된 f1.txt

이 상황에서 3way merge라는 병합이 수행된다.
공통이 되는 파일과 각자의 파일들 총 3개가 있어서 3개를 근거로
자동으로 git은 병합작업을 시도한다.

충돌이 일어난 파일들의 내용에 관한 것이다.

>vim f1.txt

```
function(){
<<<<<<< HEAD
        return "master";
=======
        return "exp";
>>>>>>> exp
}
~
~
```

병합작업을 수행하기 위해 무료 오픈소스인 kdiff3 설치하기
>git config --global merge.tool kdiff3

>git mergetool

```
gimdaewon@iMac:~/documents/gitfth8% git mergetool
Merging:
f1.txt

Normal merge conflict for 'f1.txt':
  {local}: modified file
  {remote}: modified file
The merge tool kdiff3 is not available as 'kdiff3'
```

나는 오류가 생겨서 실행이 안된다... 구글링해봐도 방법이 없는듯...이부분은 그냥 넘어갑시다..

나는 그냥 텍스트로 수정해야겠다...

>vim f1.txt //수정

```
function(){
        return "master, exp";
}
~
~
```

[esc] + :wq + [enter]

>git add f1.txt

>git commit

```
Merge branch 'exp'

# Conflicts:
#       f1.txt
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#       .git/MERGE_HEAD
# and try again.


# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Committer: 김대원 <gimdaewon@iMac.local>
#
# On branch master
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#       modified:   f1.txt
#
# Untracked files:
"~/Documents/gitfth8/.git/COMMIT_EDITMSG" 33L, 687C
```

[esc] + :wq + [enter]

>git log

```
commit fbea883830104c1004a2c0edb3b97804c6af4782
Merge: 6641bf8 58aeaba
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 02:01:17 2017 +0900

    Merge branch 'exp'

commit 6641bf8c1edf8a3dd8b57f7cad84408950ef3f0f
```

```
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 01:38:35 2017 +0900

    common->master

commit 58aeaba16041634c730a8b69b63875994d5b8178
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 01:34:42 2017 +0900

    common->exp

commit 86c07aa5b92012e045edc5315c17ade2f4238085
Author: 김대원 <gimdaewon@iMac.local>
Date:   Wed Apr 12 01:32:01 2017 +0900

:
```

5)3 way merge

Me	Base	Other	2 way merge	3 way merge
A	A		?	
B	B	B	B	B
1	C	2	?	?
	D	D	?	

꼭 몰라도 git을 사용하는데 큰 문제 없음

(1)4행의 코드가 있다고 하자.

	A			
	B			
	C			
	D			

(2)Me라는 현재의 branch와 Other라는 또다른 branch가 있다.

Me		Other		
	A			
	B			
	C			
	D			

(3)Me(현재의 branch)는 A와 B는 그대로 C는1로 바꾸고 D는 지웠고,
Other은 A는 지우고 B는 그대로 C는2로 바꾸고 D는 그대로 두었다.
가운데 코드는 Base코드가 된다.

Me	Base	Other		
A	A			
B	B	B		
1	C	2		
	D	D		

(4)Other라는 branch를 Me로 Merge 하려고 한다면(>git merge Other),,,,,,

(4)-1. 2 way merge

Base를 보지않고 Me와 Other과의 차이점만으로 병합을 하는 방식이다.

따라서 B이외에 모든부분들(?표가 있는부분들)에서 Conflict가 나게 된다.

Me		Other	2 way merge
A			?
B		B	B
1		2	?
		D	?

(4)-2. 3 way merge

Base를 참고로해서 Me와 Other을 병합하는 방식이다.

B만 Merge가 되고 ?표부분 1개만 conflict가 나게된다.

Me	Base	Other	2 way merge	3 way merge
A	A		?	
B	B	B	B	B
1	C	2	?	?
	D	D	?	

설명이 완벽하게 이해가 안가서 구글링해본 자료..

3-way merge 알고리즘 보충자료

Git에서 두 브랜치를 머지할 때, 기본적으로 3-way merge 알고리즘을 사용하게 된다.

어떤 sequence A, B와 그 둘의 base인 sequence O가 있다고 하자. Git으로 치면 A, B는 머지할 브랜치, O는 base commit이 된다. sequence의 각 item은 소스코드의 한 줄이라고 생각하면 된다.

이렇게 A, O, B가 있을 때,

1. A, O, B에 대해 [Longest common subsequence](이하 LCS) ((이름 그대로 가장 긴 공통 sequence이다. 알고리즘은 http://rosettacode.org/wiki/Longest_common_subsequence 의 예제코드들을 보면 쉽게 이해할 수 있을 것이다.)) 를 찾는다. A, O에 대해 LCS를 먼저 찾고, O, B에 대해서도 찾은 다음 이 LCS들을 통해 얻는다.
2. 위의 LCS를 이용해 모든 chunk에 대해 stable, unstable 여부를 검사한다. A, O, B가 모두 같다면 stable이고 아니면 unstable이다.
3. unstable chunk에서 A와 O가 같지만 B가 다르다면 B를 택하고, B와 O가 같지만 A가 다르다면 A를 택한다. 셋이 모두 다르다면 충돌로 처리한다.

A Formal Investigation of Diff3에 소개된 예를 통해 위의 알고리즘을 설명하겠다.

다음과 같은 세 sequence가 있다면,

```
A = [1, 4, 5, 2, 3, 6]
O = [1, 2, 3, 4, 5, 6]
B = [1, 2, 4, 5, 3, 6]
```

A, O의 LCS는 다음과 같고

```
A = [[1], [4, 5], [2, 3], [],      [6]]
O = [[1], []      [2, 3], [4, 5], [6]]
```

O, b의 LCS는 다음과 같다.

```
O = [[1, 2], [3], [4, 5], [],  [6]]
B = [[1, 2], [],  [4, 5], [3], [6]]
```

셋의 LCS를 찾으면 다음과 같다.

```
A = [[1], [4, 5], [2], [3],        [6]]
O = [[1], [],     [2], [3, 4, 5], [6]]
B = [[1], [],     [2], [4, 5, 3], [6]]
```

위에서 `[1], [2], [6]`은 셋이 모두 같으므로, stable chunk라고 하며 그 외에는 unstable chunk라고 한다. unstable chunk에 대해서 conflict가 발생했는지 찾는다. A,

O, B가 모두 다른 chunk가 있다면 conflict다.

```
A' = [[1], [4, 5], [2], [3],        [6]]
O' = [[1], [4, 5], [2], [3, 4, 5], [6]]
B' = [[1], [4, 5], [2], [4, 5, 3], [6]]
```

2번째 chunk는 A만 `[4, 5]`이고 O와 B는 같으므로(`[]`) A에서만 변경이 있었음을 의미한다. merge하면 따라서 `[4, 5]`가 선택된다. 4번째 chunk인 `[3], [3, 4, 5], [4, 5, 3]`은 A', O', B'가 모두 다르므로 conflict다.

위 결과를 출력하면 다음과 같다.

```
1
4
5
2
<<<<<<< A
3
||||||| O
3
4
5
=======
4
5
```

```
3
>>>>>>> B
6
```

그러나 실제 patch, diff3, Git등으로 merge를 해 보면 이보다 더 쉽게 충돌한다. 주위 문맥을 고려하기 때문이다. 대개 전후 3줄 이내에 변경이 있으면 충돌로 간주한다.

3-way merge에 대해 더 자세히 알고 싶다면 A Formal Investigation of Diff3을 보라. Longest common subsequence를 포함해 3-way merge의 전 과정에 대해 자세히 설명하고 있다.