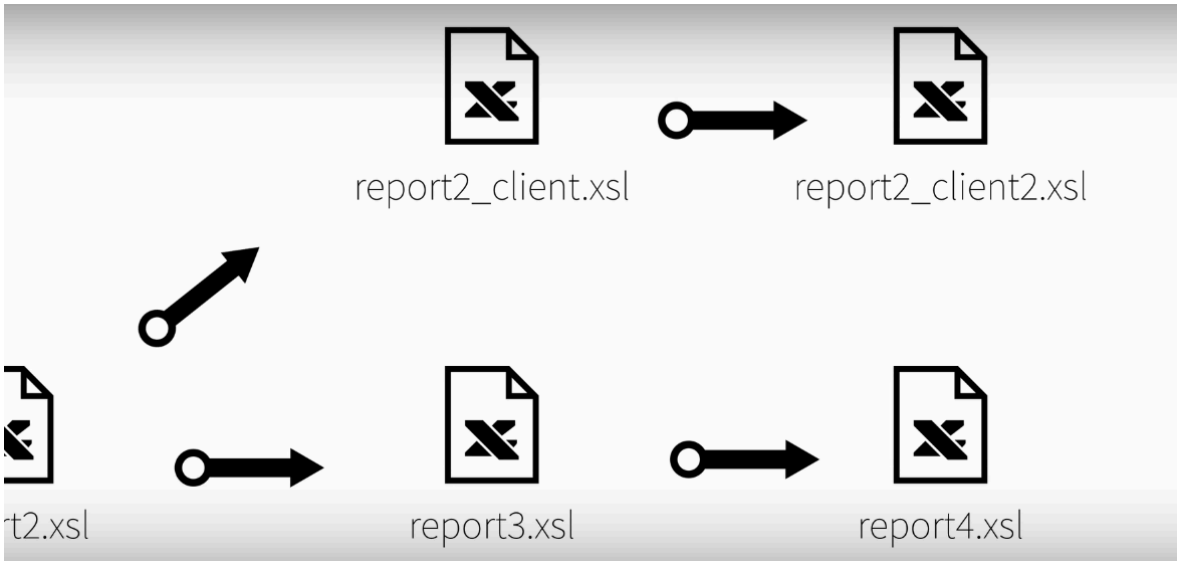
- git의 혁신 branch

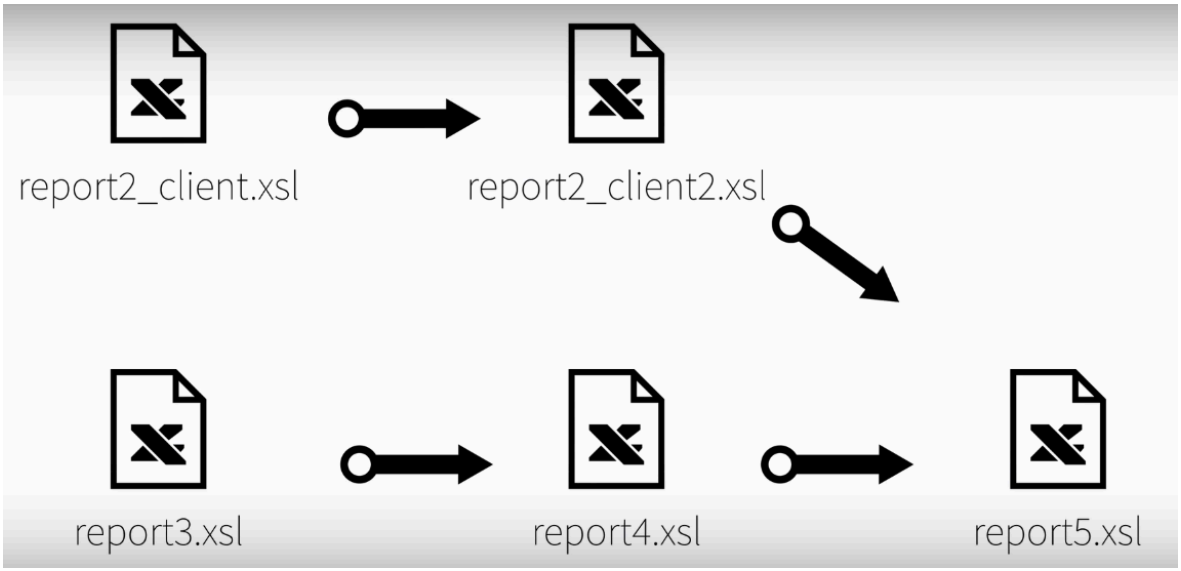
어떤 파일을 수정한다
report.xsl 이라는 파일이 수정되어
report1.xsl
report2.xsl
report3.xsl 등등으로 생성된다.

고객용으로 바꿀경우 report2_client.xsl 로 수정되어 배포된다.

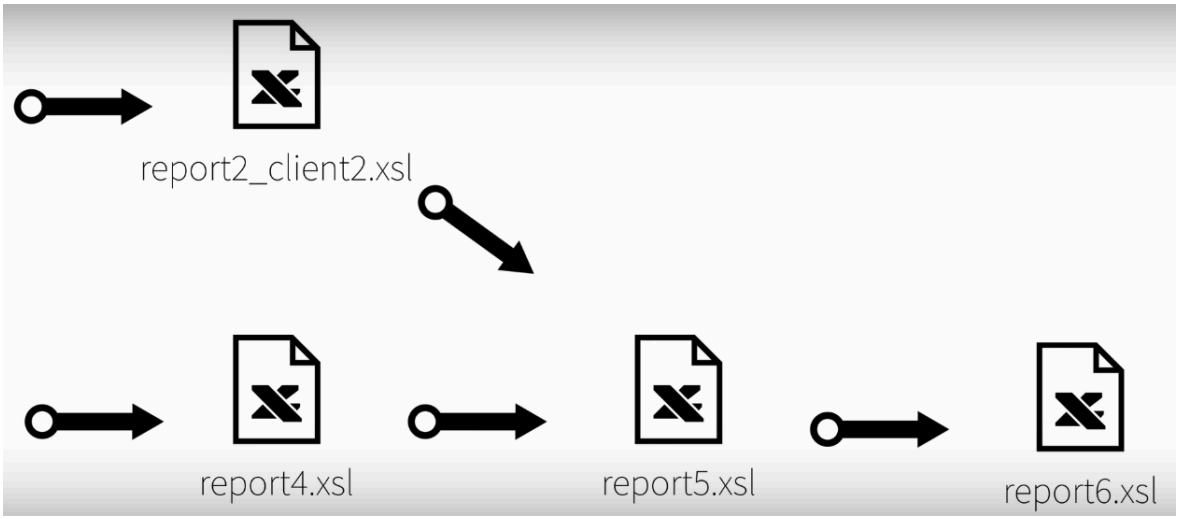
이는 지속적으로 파일명이 변경되어가며 배포



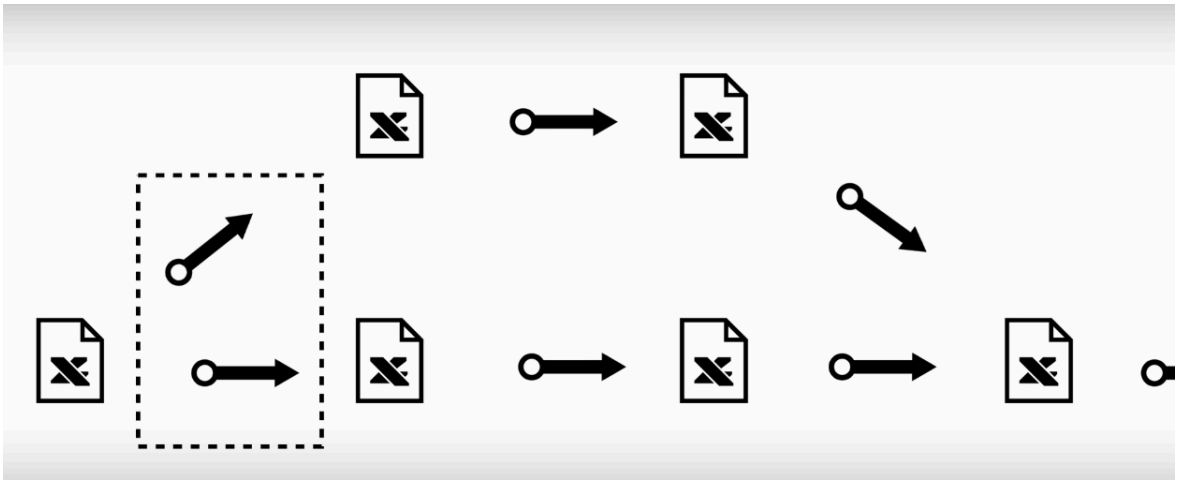
그런데 클라이언트에게 제공한 문서(report2_client2.xsl)가 우리의 오리지널 문서에 적용되어야 할때,



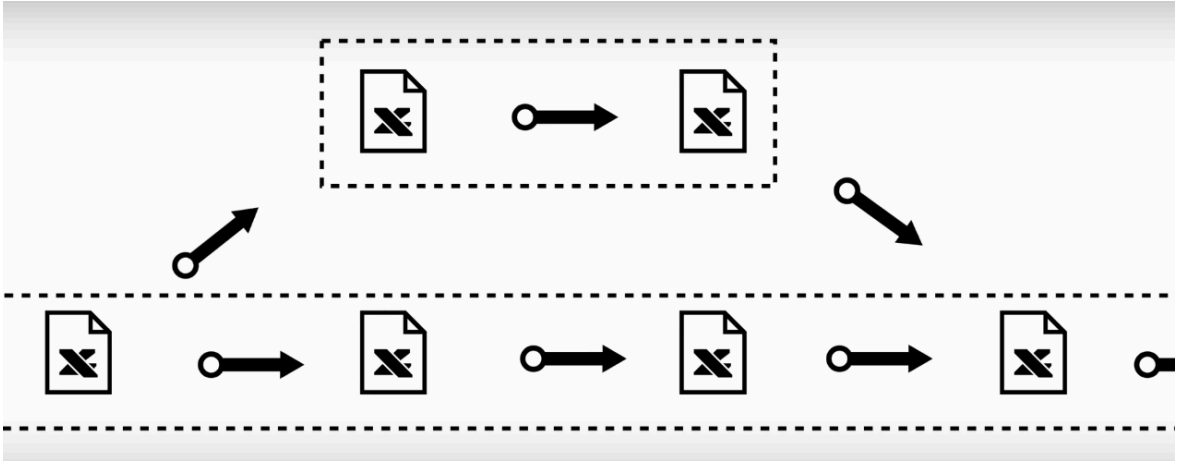
나뉘어져있는 파일을 합치는 것은 힘들고 파일이름이 더러워진다.



branch를 만든다.



이렇게 해서 2개의 branch를 갖게 된다.



1)branch 만들기

> git init
> vim f1.txt 생성
> git commit -m “1”
> vim f1.txt 수정
> git commit -am “2”

여기서 -a는 commit시 자동으로 add해주지만

버전관리가 아직 시작되지 않은 파일들은 자동으로 add되지 않음

> git log

```
commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:55 2017 +0900

    2

commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:01 2017 +0900

    1
```

혹시 필요 없을거라 생각되는 부분을 분기를 만들어 작업을 진행하면 추후에 필요없는것을 지우기에 유용하다.

> git branch

```
gimdaewon@iMac:~/documents/gitfth3% git branch
* master
```

브랜치의 목록을 볼 때

git branch

브랜치를 생성할 때

git branch "새로운 브랜치 이름"

브랜치를 삭제할 때

git branch -d

병합하지 않은 브랜치를 강제 삭제할 때

git branch -D

브랜치를 전환(체크아웃)할 때

git checkout "전환하려는 브랜치 이름"

브랜치를 생성하고 전환까지 할 때

git checkout -b "생성하고 전환할 브랜치 이름"

>git branch exp
>git branch

```
gimdaewon@iMac:~/documents/gitfth3% git branch
  exp
* master
```

>git checkout exp

```
gimdaewon@iMac:~/documents/gitfth3% git checkout exp
Switched to branch 'exp'
```

>git branch

```
gimdaewon@iMac:~/documents/gitfth3% git branch
* exp
  master
```

>ls -al

```
gimdaewon@iMac:~/documents/gitfth3% ls -al
total 8
drwxr-xr-x   4 gimdaewon  staff   136  4 11 01:20 .
drwx------+ 48 gimdaewon  staff  1632  4 11 01:31 ..
drwxr-xr-x  14 gimdaewon  staff   476  4 11 01:31 .git
-rw-r--r--   1 gimdaewon  staff     4  4 11 01:20 f1.txt
```

>git log

```
commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:55 2017 +0900
```

```
    2

commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:01 2017 +0900

    1
```

branch를 생성하면 현재 상태의 branch(master의) 그대로를 갖는다.

>vim f1.txt f1.txt 수정
>git add f1.txt
>git commit -m “3”
>git log

```
commit c071055551b2dca6f1820a1674b7077a842a96ea
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:34:08 2017 +0900

    3

commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:55 2017 +0900

    2

commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:01 2017 +0900

    1
```

master branch로 바꿔보자

>git checkout master

```
gimdaewon@iMac:~/documents/gitfth3% git checkout master
Switched to branch 'master'
```

>git branch

```
gimdaewon@iMac:~/documents/gitfth3% git branch
  exp
* master
```

>git log

```
commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:55 2017 +0900

    2

commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:01 2017 +0900

    1
```

>cat f1.txt

```
gimdaewon@iMac:~/documents/gitfth3% cat f1.txt
a
b
```

다시 exp branch로 돌아와서

>git checkout exp

>git log

```
commit c071055551b2dca6f1820a1674b7077a842a96ea
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:34:08 2017 +0900

    3

commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:55 2017 +0900

    2

commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:01 2017 +0900

    1
```

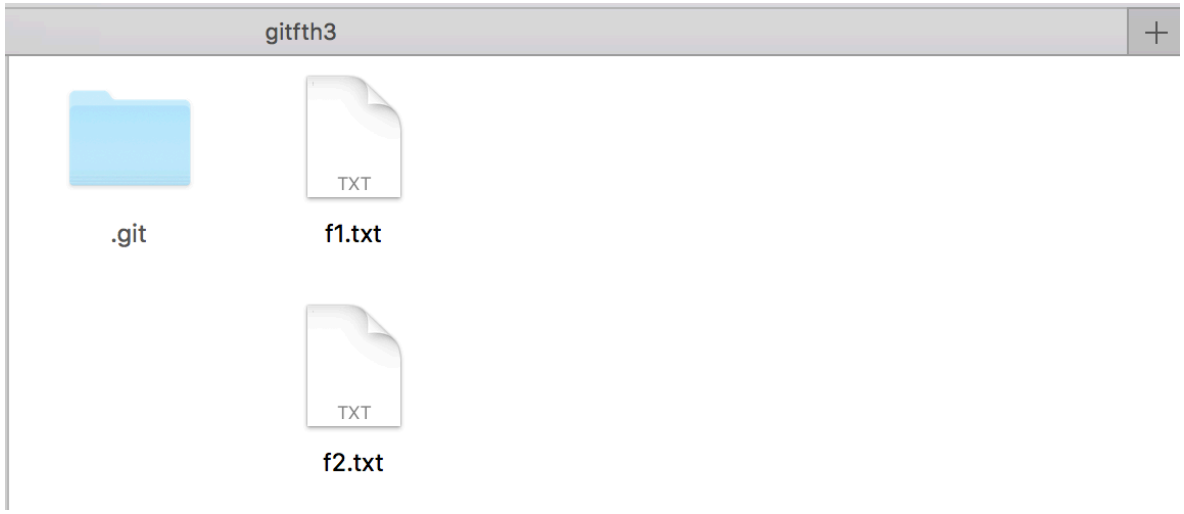
>cat f1.txt

```
gimdaewon@iMac:~/documents/gitfth3% cat f1.txt
a
```

```
b
c
```

또다른 실습

>vim f2.txt f2.txt파일 생성



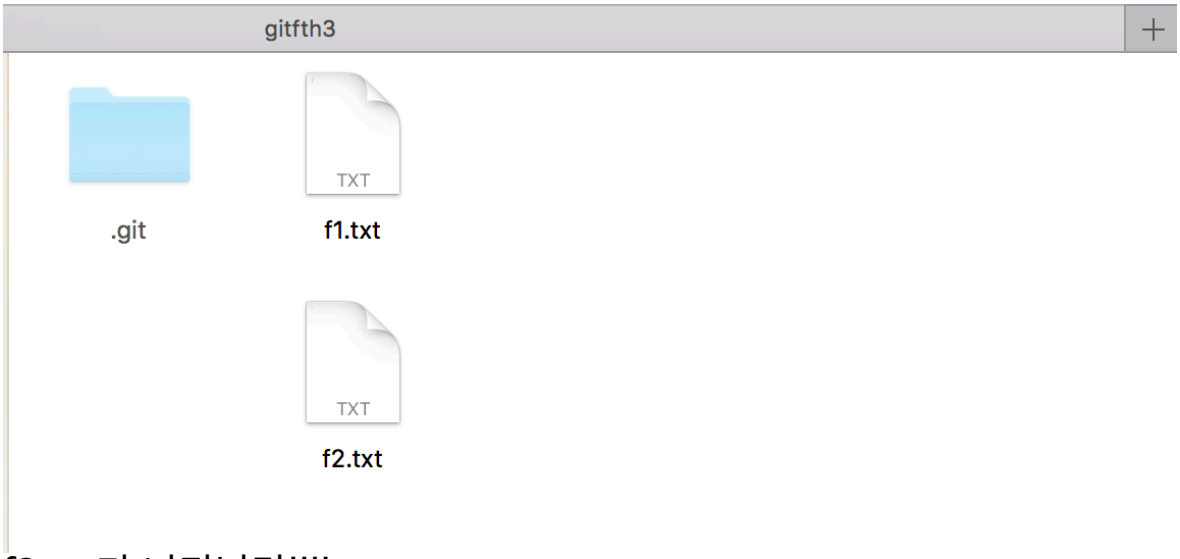
>git add f2.txt
>git commit -m “4”

master로 branch 바꾸기

>git checkout master

f2.txt 가 사라진닷!!!!!

다시 exp로 branch를 바꾸면



f2.txt가 나타난닷!!!!

이러한 편리함으로인해 우리가 익혀야 될것이 있다.

1)branch 정보확인

branch로 인해 상당한 복잡성을 갖게 된다…

>git branch

```
gimdaewon@iMac:~/documents/gitfth3% git branch
* exp
```

```
  master
```

>git log --branches --decorate

```
commit 9cf9ab845fb022a7f7041e26971cd71c41fec158 (HEAD -> exp)
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:41:52 2017 +0900

    4

commit c071055551b2dca6f1820a1674b7077a842a96ea
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:34:08 2017 +0900

    3

commit 796492d8b9c3c68fff670eb10c32f31aa5426c12 (master)
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:55 2017 +0900

    2

commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:01 2017 +0900

    1
```

`master` branch의 가장 최신 commit이 `2` 라는 것
`exp` branch에 checkout 되었다가 `HEAD`라는 파일의 의미(더 깊은 의미는 추후에..)

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

```
* commit 9cf9ab845fb022a7f7041e26971cd71c41fec158 (HEAD -> exp)
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 01:41:52 2017 +0900
|
|     4
|
* commit c071055551b2dca6f1820a1674b7077a842a96ea
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 01:34:08 2017 +0900
|
|     3
|
* commit 796492d8b9c3c68fff670eb10c32f31aa5426c12 (master)
```

```
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 01:20:55 2017 +0900
|
|     2
|
* commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
  Author: 김대원 <gimdaewon@iMac.local>
  Date:   Tue Apr 11 01:20:01 2017 +0900

      1
```

여기서는 크게 티가 나지 않지만 exp와 mater가 각자의 길을 걷는 commit 상태가 되었을때 더
도드라진다.
그럼 더 도드라지게 해보자! 어떻게 ? master에 새로운 commit 을 만들어보자

>git checkout master
>vim f3.txt
>git add f3.txt
>git commit -m "5"
>git log

```
commit 8b3fee12b8e1f80c310fe9b236c6cb1cb5c95089
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:56:10 2017 +0900

    5

commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:55 2017 +0900

    2

commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:20:01 2017 +0900

    1
```

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

```
* commit 8b3fee12b8e1f80c310fe9b236c6cb1cb5c95089 (HEAD -> master)
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 01:56:10 2017 +0900
|
|     5
|
```

```
| * commit 9cf9ab845fb022a7f7041e26971cd71c41fec158 (exp)
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 01:41:52 2017 +0900
| |
| |     4
| |
| * commit c071055551b2dca6f1820a1674b7077a842a96ea
|/  Author: 김대원 <gimdaewon@iMac.local>
|   Date:   Tue Apr 11 01:34:08 2017 +0900
|
|       3
|
* commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 01:20:55 2017 +0900
|
|     2
|
* commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
  Author: 김대원 <gimdaewon@iMac.local>
  Date:   Tue Apr 11 01:20:01 2017 +0900

      1
```

`exp`의 최신commit 은 4이고 그 이전의 commit 은 3이다.
그리고 2라고하는 commit에서 비롯된 것이다.
`master`는 5이전의 2라고 하는 commit이었고 `master`와 `exp`의 공통의 조상은 2라고하는 commit 이다.

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

```
* 8b3fee1 (HEAD -> master) 5
| * 9cf9ab8 (exp) 4
| * c071055 3
|/
* 796492d 2
* 6b4b3fc 1
```

(END)

한눈에 보여준다. *ㅅ*!!!

>git log master..exp

```
commit 9cf9ab845fb022a7f7041e26971cd71c41fec158
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:41:52 2017 +0900
```

```
    4

commit c071055551b2dca6f1820a1674b7077a842a96ea
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:34:08 2017 +0900

    3
```

master에는 없고 exp에는 있는것들을 보여준다.

>git log exp..master

```
commit 8b3fee12b8e1f80c310fe9b236c6cb1cb5c95089
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:56:10 2017 +0900

    5
```

exp에는 없고 master에는 있는 것들을

>git log -p exp..master

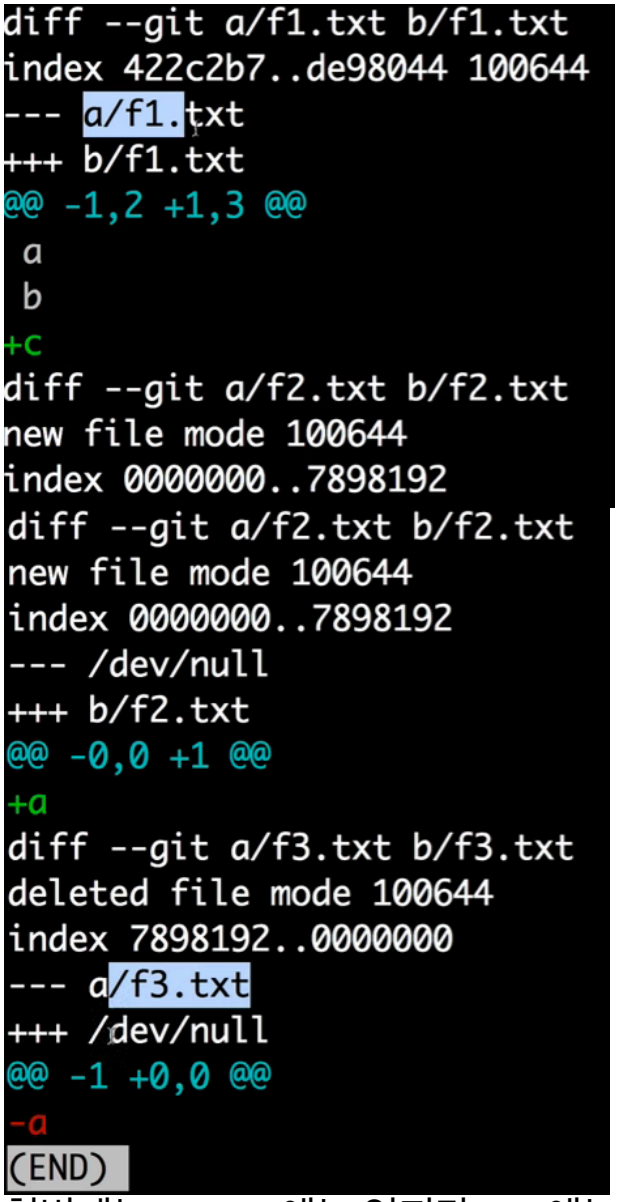
```
commit 8b3fee12b8e1f80c310fe9b236c6cb1cb5c95089
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 01:56:10 2017 +0900

    5

diff --git a/f3.txt b/f3.txt
new file mode 100644
index 0000000..7898192
--- /dev/null
+++ b/f3.txt
@@ -0,0 +1 @@
+a
```

exp에는 없고(`--- /dev/null`) master에는 있는(`+++ b/f3.txt`) 것들을 내용까지 상세하게 보여준다.

>git diff master..exp



첫번째는 master에는 없지만 exp에는 있는 내용(`+c`)을 나타낸다.
두번째는 master에는 파일이 없지만(`--- /dev/null`) exp에는 있는 파일(`f2.txt`)과 그내용(`+a`)
세번째는 master에는 파일(`f3.txt`)이 있고 exp에는 파일이 없고(`--- /dev/null`) 없는파일의 내용(`-a`)

브랜치 간에 비교할 때

git log "비교할 브랜치 명 1".."비교할 브랜치 명 2"

브랜치 간의 코드를 비교 할 때

git diff "비교할 브랜치 명 1".."비교할 브랜치 명 2"

로그에 모든 브랜치를 표시하고, 그래프로 표현하고, 브랜치 명을 표시하고, 한줄로 표시할 때

3)branch 병합

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

```
* 8b3fee1 (HEAD -> master) 5
| * 9cf9ab8 (exp) 4
| * c071055 3
|/
* 796492d 2
* 6b4b3fc 1
```

exp의 내용(3,4라는 commit)을 master로 옮기는 방법

master로 checkout 한다음 merge하는 방법

>git checkout master

>git merge exp

```
Merge branch 'exp'

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"~/Documents/gitfth3/.git/MERGE_MSG" 7L, 246C
```

[esc] + :wq + [Enter]

```
gimdaewon@iMac:~/documents/gitfth3% git merge exp
Merge made by the 'recursive' strategy.
 f1.txt | 1 +
 f2.txt | 1 +
 2 files changed, 2 insertions(+)
 create mode 100644 f2.txt
```

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

```
*   d821919 (HEAD -> master) Merge branch 'exp'
|\
| * 9cf9ab8 (exp) 4
| * c071055 3
* | 8b3fee1 5
|/
* 796492d 2
* 6b4b3fc 1
```

(END)

이 commit(`d821919`)은 5(`8b3fee1`)번(부모)와 `exp`가 작업했던 3번(`c071055`)과 4번(`9cf9ab8`)을 부모로 갖는다.

>ls -al

```
gimdaewon@iMac:~/documents/gitfth3% ls -al
total 40
drwxr-xr-x   7 gimdaewon  staff   238  4 11 02:30 .
drwx------+ 48 gimdaewon  staff  1632  4 11 02:34 ..
-rw-r--r--@  1 gimdaewon  staff  6148  4 11 02:30 .DS_Store
drwxr-xr-x  15 gimdaewon  staff   510  4 11 02:30 .git
-rw-r--r--   1 gimdaewon  staff     6  4 11 02:29 f1.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 02:29 f2.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 02:12 f3.txt
```

이제 master는 f1.txt, f2.txt(exp에만 있던), f3.txt 모두를 갖게 되었다.

master가 작업했던 5번(`8b3fee1`)을 exp에도 병합하는 방법

>git checkout exp

```
gimdaewon@iMac:~/documents/gitfth3% git checkout exp
Switched to branch 'exp'
```

>git merge master

```
gimdaewon@iMac:~/documents/gitfth3% git merge master
Updating 9cf9ab8..d821919
Fast-forward
 f3.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 f3.txt
```

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

```
*   d821919 (HEAD -> exp, master) Merge branch 'exp'
|\
| * 9cf9ab8 4
| * c071055 3
* | 8b3fee1 5
|/
* 796492d 2
* 6b4b3fc 1
```

(END)

이제는 필요없어진 exp branch지우기

>git checkout master

```
gimdaewon@iMac:~/documents/gitfth3% git checkout master
Switched to branch 'master'
```

>git branch -d exp

```
gimdaewon@iMac:~/documents/gitfth3% git branch -d exp
Deleted branch exp (was d821919).
```

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

```
*   d821919 (HEAD -> master) Merge branch 'exp'
|\
| * 9cf9ab8 4
| * c071055 3
* | 8b3fee1 5
|/
* 796492d 2
* 6b4b3fc 1
```

(END)

A 브랜치로 B 브랜치를 병합할 때 (A ← B)

```
git checkout A
git merge B
```

4)branch 수련

브랜치와 Merge 의 기초

실제 개발과정에서 겪을 만한 예제를 하나 살펴보자. 브랜치와 Merge는 보통 이런 식으로 진행한다.

1. 작업 중인 웹사이트가 있다.
2. 새로운 이슈를 처리할 새 Branch를 하나 생성한다.
3. 새로 만든 Branch에서 작업을 진행한다.

이때 중요한 문제가 생겨서 그것을 해결하는 Hotfix를 먼저 만들어야 한다. 그러면 아래와 같이 할 수 있다.

1. 새로운 이슈를 처리하기 이전의 운영(Production) 브랜치로 이동한다.
2. Hotfix 브랜치를 새로 하나 생성한다.
3. 수정한 Hotfix 테스트를 마치고 운영 브랜치로 Merge 한다.
4. 다시 작업하던 브랜치로 옮겨가서 하던 일 진행한다.

브랜치의 기초

먼저 지금 작업하는 프로젝트에서 이전에 커밋을 몇 번 했다고 가정한다.

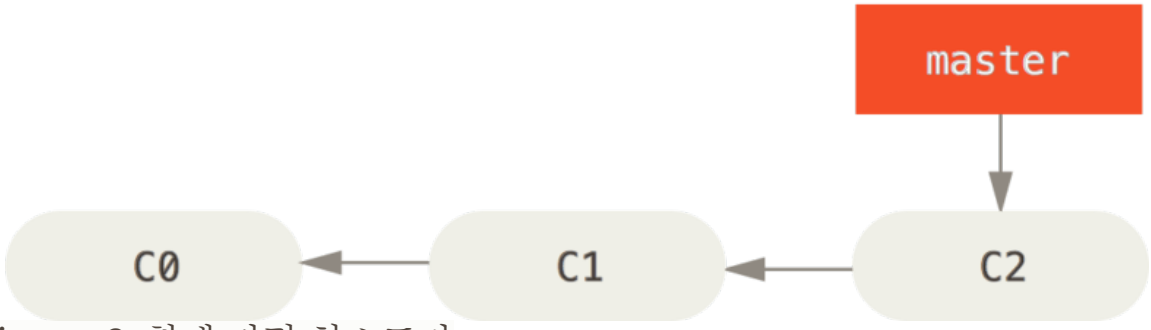


Figure 18. 현재 커밋 히스토리

이슈 관리 시스템에 등록된 53번 이슈를 처리한다고 하면 이 이슈에 집중할 수 있는 브랜치를 새로 하나 만든다. 브랜치를 만들면서 Checkout까지 한 번에 하려면 `git checkout` 명령에 `-b`라는 옵션을 추가한다.

```
$ git checkout -b iss53
Switched to a new branch "iss53"
```

위 명령은 아래 명령을 줄여놓은 것이다.

```
$ git branch iss53
$ git checkout iss53
```

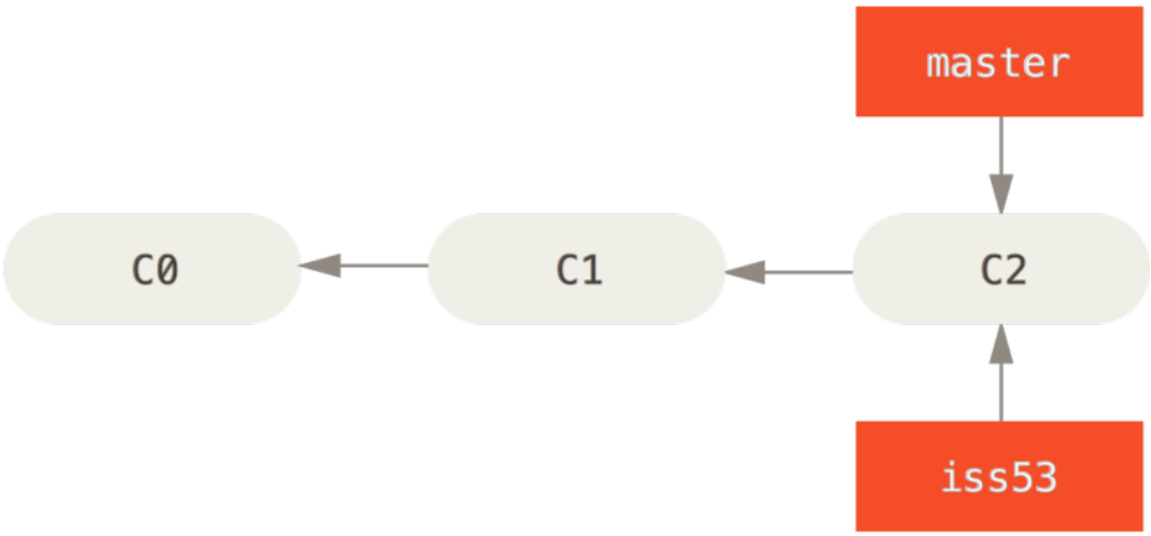


Figure 19. 브랜치 포인터를 새로 만듦

`iss53` 브랜치를 Checkout 했기 때문에(즉, `HEAD`는 `iss53` 브랜치를 가리킨다) 뭔가 일을 하고 커밋하면 `iss53`브랜치가 앞으로 나아간다.

```
$ vim index.html
$ git commit -a -m 'added a new footer [issue 53]'
```

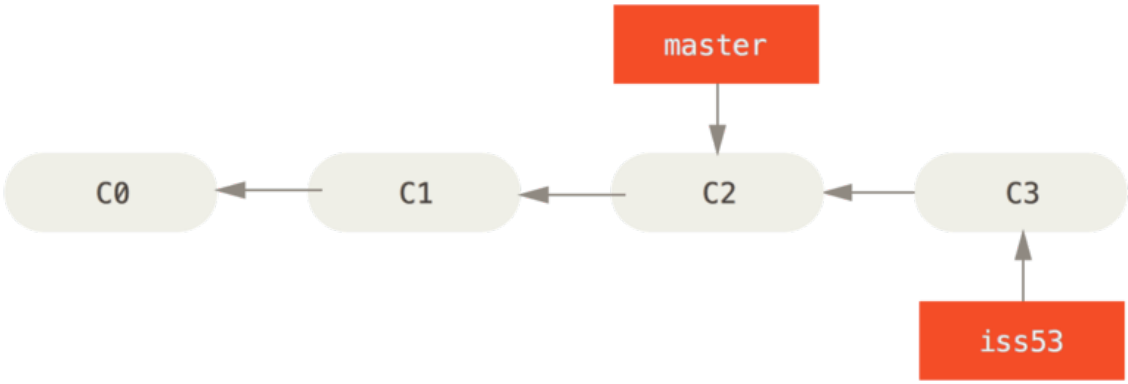


Figure 20. 진행 중인 iss53 브랜치

다른 상황을 가정해보자. 만드는 사이트에 문제가 생겨서 즉시 고쳐야 한다. 버그를 해결한 Hotfix에 `iss53`이 섞이는 것을 방지하기 위해 `iss53`과 관련된 코드를 어딘가에 저장해두고 원래 운영 환경의 소스로 복구해야 한다. Git을 사용하면 이런 노력을 들일 필요 없이 그냥 `master` 브랜치로 돌아가면 된다.

그렇지만, 브랜치를 이동하려면 해야 할 일이 있다. 아직 커밋하지 않은 파일이 Checkout 할 브랜치와 충돌 나면 브랜치를 변경할 수 없다. 브랜치를 변경할 때는 워킹 디렉토리를 정리하는 것이 좋다. 이런 문제를 다루는 방법은(주로, Stash이나 커밋 Amend에 대해) 나중에 Stashing과 Cleaning 에서 다룰 것이다. 지금은 작업하던 것을 모두 커밋하고 `master` 브랜치로 옮긴다:

```
$ git checkout master
Switched to branch 'master'
```

이때 워킹 디렉토리는 53번 이슈를 시작하기 이전 모습으로 되돌려지기 때문에 새로운 문제에 집중할 수 있는 환경이 만들어진다. Git은 자동으로 워킹 디렉토리에 파일들을 추가하고, 지우고, 수정해서 Checkout 한 브랜치의 마지막 스냅샷으로 되돌려 놓는다는 것을 기억해야 한다.

이젠 해결해야 할 핫픽스가 생겼을 때를 살펴보자. hotfix라는 브랜치를 만들고 새로운 이슈를 해결할 때까지 사용한다.

```
$ git checkout -b hotfix
Switched to a new branch 'hotfix'
$ vim index.html
$ git commit -a -m 'fixed the broken email address'
[hotfix 1fb7853] fixed the broken email address
```

```
 1 file changed, 2 insertions(+)
```

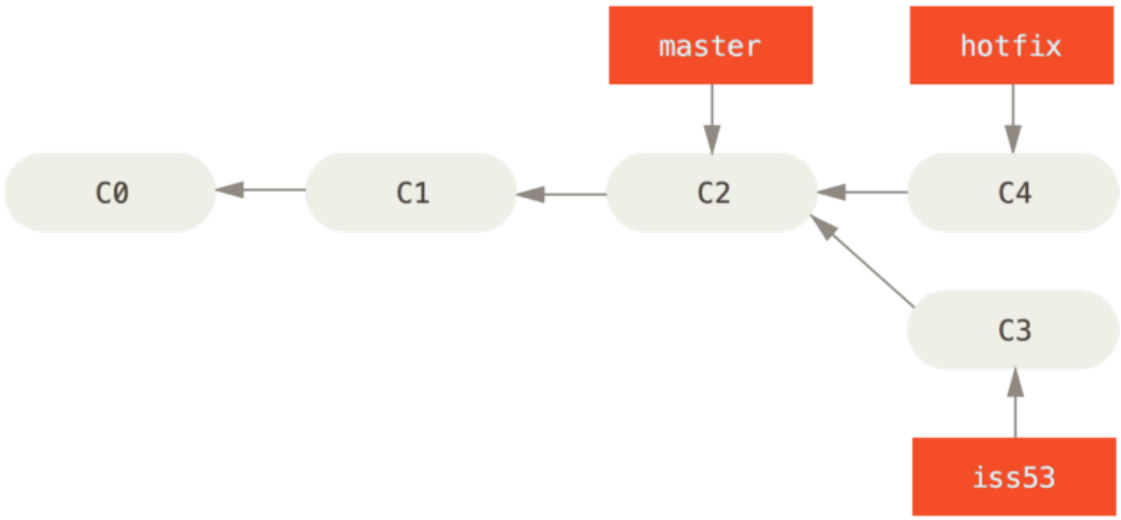


Figure 21. `master` 브랜치에서 갈라져 나온 hotfix 브랜치
운영 환경에 적용하려면 문제를 제대로 고쳤는지 테스트하고 `master` 브랜치에 합쳐야 한다. `git merge` 명령으로 아래와 같이 한다.

```
$ git checkout master
$ git merge hotfix
Updating f42c576..3a0874c
Fast-forward
 index.html | 2 ++
 1 file changed, 2 insertions(+)
```

Merge 메시지에서 ‘`fast-forward”가 보이는가. `hotfix` 브랜치가 가리키는 `C4` 커밋이 `C2` 커밋에 기반한 브랜치이기 때문에 브랜치 포인터는 Merge 과정 없이 그저 최신 커밋으로 이동한다. 이런 Merge 방식을 'Fast forward’라고 부른다. 다시 말해 A 브랜치에서 다른 B 브랜치를 Merge 할 때 B 브랜치가 A 브랜치 이후의 커밋을 가리키고 있으면 그저 A 브랜치가 B 브랜치와 동일한 커밋을 가리키도록 이동시킬 뿐이다.

이제 hotfix는 master 브랜치에 포함됐고 운영환경에 적용할 수 있는 상태가 되었다고 가정해보자.

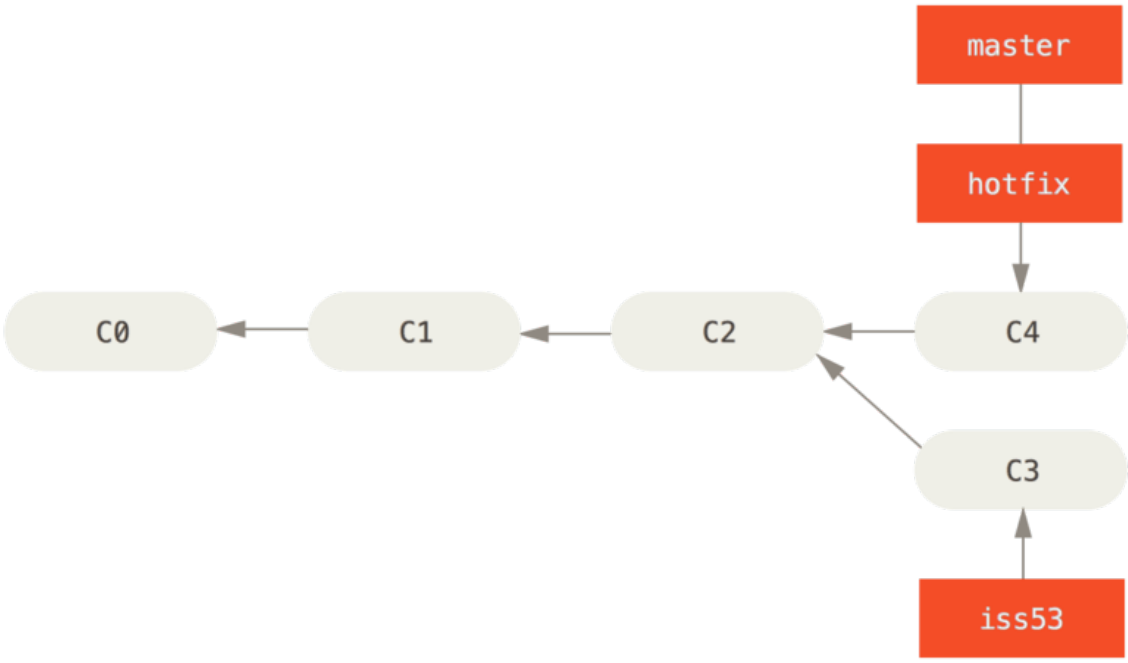


Figure 22. Merge 후 `hotfix` 같은 것을 가리키는 `master` 브랜치

급한 문제를 해결하고 master 브랜치에 적용하고 나면 다시 일하던 브랜치로 돌아가야 한다. 이제 더 이상 필요없는 hotfix 브랜치는 삭제한다. `git branch` 명령에 `-d` 옵션을 주고 브랜치를 삭제한다.

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

자 이제 이슈 53번을 처리하던 환경으로 되돌아가서 하던 일을 계속 하자.

```
$ git checkout iss53
Switched to branch "iss53"
$ vim index.html
$ git commit -a -m 'finished the new footer [issue 53]'
[iss53 ad82d7a] finished the new footer [issue 53]
1 file changed, 1 insertion(+)
```

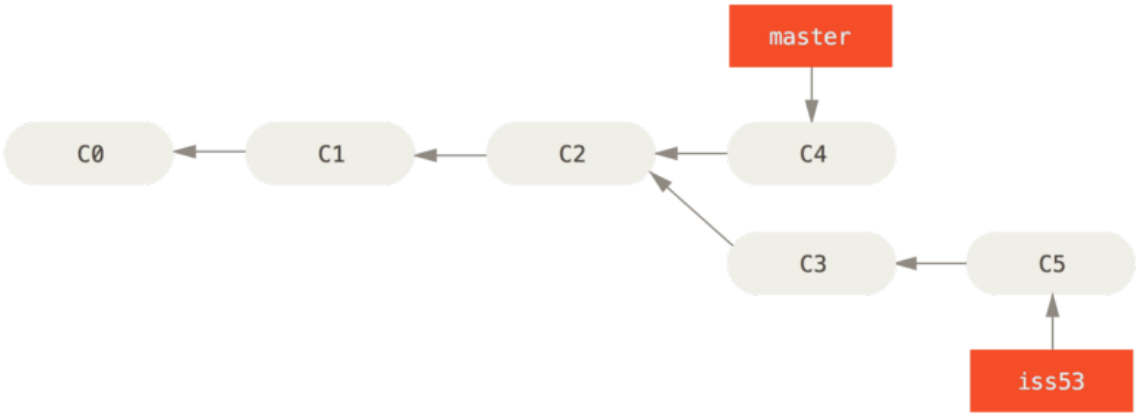


Figure 23. master와 별개로 진행하는 iss53 브랜치

위에서 작업한 `hotfix`가 `iss53` 브랜치에 영향을 끼치지 않는다는 점을 이해하는 것이 중요하다. `git merge master` 명령으로 `master` 브랜치를 `iss53` 브랜치에 Merge 하면 `iss53` 브랜치에 `hotfix`가 적용된다. 아니면 `iss53` 브랜치가 `master`에 Merge 할 수 있는 수준이 될 때까지 기다렸다가 `Merge` 하면 `hotfix`와 `iss53` 브랜치가 합쳐진다.

Merge 의 기초

53번 이슈를 다 구현하고 master 브랜치에 Merge 하는 과정을 살펴보자. `iss53` 브랜치를 `master` 브랜치에 Merge 하는 것은 앞서 살펴본 `hotfix` 브랜치를 Merge 하는 것과 비슷하다. `git merge` 명령으로 합칠 브랜치에서 합쳐질 브랜치를 Merge 하면 된다.

```
$ git checkout master
Switched to branch 'master'
$ git merge iss53
Merge made by the 'recursive' strategy.
README |    1 +
1 file changed, 1 insertion(+)
```

`hotfix`를 Merge 했을 때와 메시지가 다르다. 현재 브랜치가 가리키는 커밋이 Merge 할 브랜치의 조상이 아니므로 Git은 'Fast-forward'로 Merge 하지 않는다. 이 경우에는 Git은 각 브랜치가 가리키는 커밋 두 개와 공통 조상 하나를 사용하여

3-way Merge를 한다.

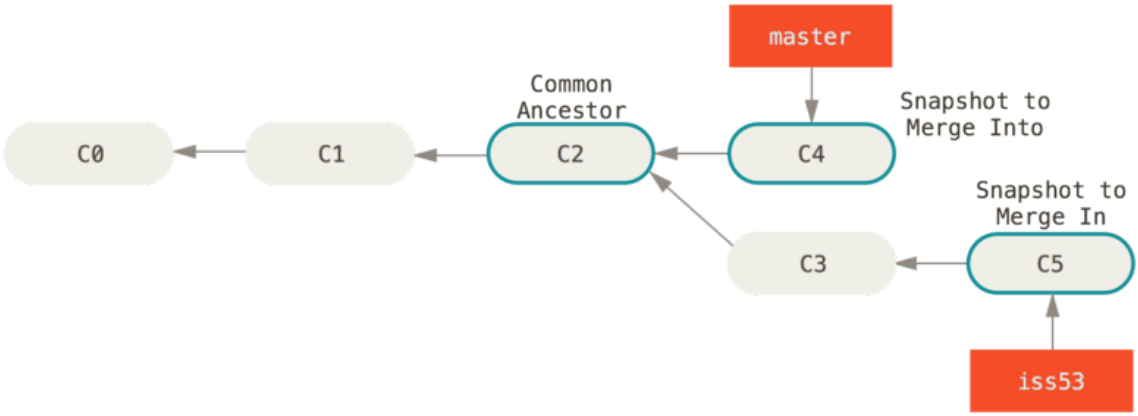


Figure 24. 커밋 3개를 Merge

단순히 브랜치 포인터를 최신 커밋으로 옮기는 게 아니라 3-way Merge 의 결과를 별도의 커밋으로 만들고 나서 해당 브랜치가 그 커밋을 가리키도록 이동시킨다. 그래서 이런 커밋은 부모가 여러 개고 Merge 커밋이라고 부른다.

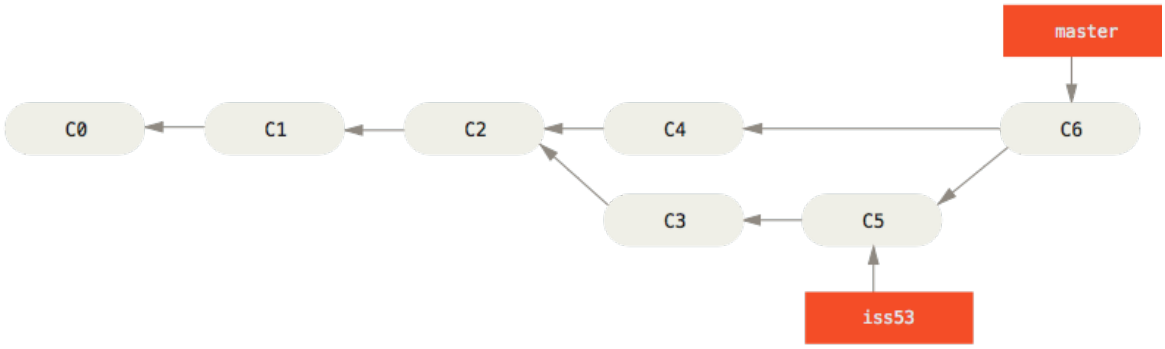


Figure 25. Merge 커밋

Git은 Merge 하는데 필요한 최적의 공통 조상을 자동으로 찾는다. 이런 기능도 Git이 다른 버전 관리 시스템보다 나은 점이다. CVS나 Subversion 같은 버전 관리 시스템은 개발자가 직접 공통 조상을 찾아서 Merge 해야 한다. Git은 다른 시스템보다 Merge가 대단히 쉽다.

iss53 브랜치를 master에 Merge 하고 나면 더는 iss53 브랜치가 필요 없다. 다음 명령으로 브랜치를 삭제하고 이슈의 상태를 처리 완료로 표시한다.

```
$ git branch -d iss53
```

4)branch 병합 시 충돌해결

master branch 상태에서
>git branch exp
>vim master.txt　　//master.txt 생성
>git add 'mater.txt'
>git commit -m '6'

>git checkout exp
>vim exp.txt　　//exp.txt 생성
>git add 'exp.txt'
>git commit -m '7'

>git checkout master
>git log --branches --decorate --graph

```
* commit d6f8c81b14647c335198ea488b72aa787f719783 (exp)
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 03:15:17 2017 +0900
|
|     7
|
| * commit f0c26d0b585cfd1d6551eb513a6c8c20216c93ae (HEAD -> master)
|/  Author: 김대원 <gimdaewon@iMac.local>
|   Date:   Tue Apr 11 03:14:52 2017 +0900
|
|       6
|
*   commit d8219199a07d182ee966ac4d8235a35036f67739
|\  Merge: 8b3fee1 9cf9ab8
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 02:29:54 2017 +0900
| |
| |     Merge branch 'exp'
| |
| * commit 9cf9ab845fb022a7f7041e26971cd71c41fec158
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 01:41:52 2017 +0900
| |
| |     4
| |
| * commit c071055551b2dca6f1820a1674b7077a842a96ea
```

```
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 01:34:08 2017 +0900
| |
| |     3
| |
* | commit 8b3fee12b8e1f80c310fe9b236c6cb1cb5c95089
|/  Author: 김대원 <gimdaewon@iMac.local>
|   Date:   Tue Apr 11 01:56:10 2017 +0900
|
|       5
|
* commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 01:20:55 2017 +0900
|
|     2
|
* commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
  Author: 김대원 <gimdaewon@iMac.local>
  Date:   Tue Apr 11 01:20:01 2017 +0900

      1
```

>git merge exp

```
Merge branch 'exp'

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"~/Documents/gitfth3/.git/MERGE_MSG" 7L, 246C
```

[esc] + :wq + [enter]

```
Merge made by the 'recursive' strategy.
```

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

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

```
*   commit 110050320d31dfd492faffdcf0538419836c699b (HEAD -> master)
|\  Merge: f0c26d0 d6f8c81
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 03:20:14 2017 +0900
| |
| |     Merge branch 'exp'
| |
| * commit d6f8c81b14647c335198ea488b72aa787f719783 (exp)
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 03:15:17 2017 +0900
| |
| |     7
| |
* | commit f0c26d0b585cfd1d6551eb513a6c8c20216c93ae
|/  Author: 김대원 <gimdaewon@iMac.local>
|   Date:   Tue Apr 11 03:14:52 2017 +0900
|
|       6
|
*   commit d8219199a07d182ee966ac4d8235a35036f67739
|\  Merge: 8b3fee1 9cf9ab8
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 02:29:54 2017 +0900
| |
| |     Merge branch 'exp'
| |
| * commit 9cf9ab845fb022a7f7041e26971cd71c41fec158
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 01:41:52 2017 +0900
| |
| |     4
| |
| * commit c071055551b2dca6f1820a1674b7077a842a96ea
| | Author: 김대원 <gimdaewon@iMac.local>
| | Date:   Tue Apr 11 01:34:08 2017 +0900
| |
| |     3
| |
* | commit 8b3fee12b8e1f80c310fe9b236c6cb1cb5c95089
|/  Author: 김대원 <gimdaewon@iMac.local>
|   Date:   Tue Apr 11 01:56:10 2017 +0900
|
|       5
|
* commit 796492d8b9c3c68fff670eb10c32f31aa5426c12
```

```
| Author: 김대원 <gimdaewon@iMac.local>
| Date:   Tue Apr 11 01:20:55 2017 +0900
|
|     2
|
* commit 6b4b3fc2e7c41f3766409bfcbe4a48ce1dce8cab
  Author: 김대원 <gimdaewon@iMac.local>
  Date:   Tue Apr 11 01:20:01 2017 +0900

      1
```

>ls -al

```
gimdaewon@iMac:~/documents/gitfth3% ls -al
total 56
drwxr-xr-x   9 gimdaewon  staff   306  4 11 03:20 .
drwx------+ 48 gimdaewon  staff  1632  4 11 03:22 ..
-rw-r--r--@  1 gimdaewon  staff  6148  4 11 03:15 .DS_Store
drwxr-xr-x  15 gimdaewon  staff   510  4 11 03:20 .git
-rw-r--r--   1 gimdaewon  staff     2  4 11 03:20 exp.txt
-rw-r--r--   1 gimdaewon  staff     6  4 11 02:29 f1.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 02:29 f2.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 02:49 f3.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 03:15 master.txt
```

exp branch에서 만들어진 exp.txt가 아주 자연스럽게 병합되었다!

>git checkout exp
>vim common.txt //생성

```
function b()
~
~
```

>git add common.txt
>git commit -m "8"

>git checkout master
>git merge exp
>ls -al

```
gimdaewon@iMac:~/documents/gitfth3% ls -al
total 64
drwxr-xr-x  10 gimdaewon  staff   340  4 11 03:27 .
drwx------+ 48 gimdaewon  staff  1632  4 11 03:28 ..
-rw-r--r--@  1 gimdaewon  staff  6148  4 11 03:25 .DS_Store
drwxr-xr-x  15 gimdaewon  staff   510  4 11 03:27 .git
-rw-r--r--   1 gimdaewon  staff    13  4 11 03:27 common.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 03:20 exp.txt
```

```
-rw-r--r--   1 gimdaewon  staff     6  4 11 02:29 f1.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 02:29 f2.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 02:49 f3.txt
-rw-r--r--   1 gimdaewon  staff     2  4 11 03:26 master.txt
```

exp branch에 있던 common.txt가 병합된것이 보인다.

>vim common.txt //수정

```
function b()

function a()
~
~
```

>git commit -am '9'

```
gimdaewon@iMac:~/documents/gitfth3% git commit -am '9'
[master 79e3855] 9
 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, 2 insertions(+)
```

>git checkout exp

>vim common.txt //master와 다르게 수정

```
function a()

function c()
~
~
```

>git commit -am '10'

```
gimdaewon@iMac:~/documents/gitfth3% git commit -am '10'
[exp adfc5ce] 10
 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, 2 insertions(+)
```

>git checkout master
>git merge exp

```
Merge branch 'exp'

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
"~/Documents/gitfth3/.git/MERGE_MSG" 7L, 246C
```

[esc] + :wq + [enter]

>vim common.txt

```
function b()

function a()

function c()
~
~
~
```

와!!! 깔끔하게 합쳐졌닷!!!!!!

BUT!!...언제나 이렇게 행복하지는 않다.....

>git checkout exp
>cat common.txt

```
gimdaewon@iMac:~/documents/gitfth3% cat common.txt
function a()

function c()
```

여기서 master의 내용을 exp에 병합해보자

>git merge master

```
gimdaewon@iMac:~/documents/gitfth3% git merge master
Updating adfc5ce..bdc64c7
Fast-forward
 common.txt | 2 ++
 master.txt | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 master.txt
```

>cat common.txt

```
gimdaewon@iMac:~/documents/gitfth3% cat common.txt
function b()

function a()

function c()
```

>git checkout master

>vim common.txt //수정

```
function b()

function a(master)

function c()
~
```

[esc] + :wq + [enter]

>git commit -am '11'

```
gimdaewon@iMac:~/documents/gitfth3% git commit -am '11'
[master 5abf538] 11
 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(+), 1 deletion(-)
```

>git checkout exp
>vim common.txt //수정

```
function b()

function a(exp)

function c()
~
```

[esc] + :wq + [enter]

>git commit -am '12'

```
gimdaewon@iMac:~/documents/gitfth3% git commit -am '12'
[exp e1c294a] 12
 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(+), 1 deletion(-)
```

>git checkout master
>git merge exp

```
gimdaewon@iMac:~/documents/gitfth3% git merge exp
Auto-merging common.txt
CONFLICT (content): Merge conflict in common.txt
Automatic merge failed; fix conflicts and then commit the result.
```

충돌발견!!!!! ㅜ_ㅠ...

>git status

```
gimdaewon@iMac:~/documents/gitfth3% git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
```

```
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:   common.txt

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

    .DS_Store

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

>vim common.txt

```
function b()

<<<<<<< HEAD
function a(master)
=======
function a(exp)
>>>>>>> exp

function c()
~
~
```

구분자(`=======`)를 중심으로
현재 checkout한 branch의 수정사항(`<<<<<<< HEAD`)
exp branch의 내용(`>>>>>>> exp`)으로 충돌이난 부분을 표시해준다.

예를들어 이런식으로 수정한후

```
function b()

function a(master, exp)

function c()
```

저장한다.
그후에
>git add common.txt
>git commit

```
Merge branch 'exp'

# Conflicts:
```

```
#       common.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:   common.txt
#
# Untracked files:
#       .DS_Store
#
```

저장종료

>git log

```
commit e0e44385f4eb03e754ee46f5db1a87bedd39500d
Merge: 5abf538 e1c294a
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 04:08:17 2017 +0900

    exp

commit e1c294ab744d0816f0a997244b9db321443ce402
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 03:50:09 2017 +0900

    12

commit 5abf53869a9ec3e56967241953ac5cfe65e4fefb
Author: 김대원 <gimdaewon@iMac.local>
Date:   Tue Apr 11 03:45:20 2017 +0900

    11
```

4)stash

branch를 활발하게 사용하지 않는경우 자주 쓰이지 않는다.

다른 브랜치로 checkout을 해야 하는데 아직 현재 브랜치에서 작업이 끝나지 않은 경우는 커밋을 하기가 애매합니다. 이런 경우 stash를 이용하면 작업중이던 파일을 임시로 저장해두고 현재 브랜치의 상태를 마지막 커밋의 상태로 초기화 할 수 있습니다. 그 후에 다른 브랜치로 이동하고 작업을 끝낸 후에 작업 중이던 브랜치로 복귀한 후에 이전에 작업하던 내용을 복원할 수 있습니다. 여기서는 이 기능에 대해서 알아봅니다.

>git init

>vim f1.txt //생성

```
a
~
~
```

>git add f1.txt

>git commit -m 1

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git commit -m 1
[master (root-commit) 1e6c395] 1
 1 file changed, 1 insertion(+)
 create mode 100644 f1.txt
```

>git checout -b exp //exp라는 branch를 생성하면서 exp branch로 switch 한다

>vim f1.txt

```
a
b
~
```

>git checkout master

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git checkout master
M    f1.txt
Switched to branch 'master'
```

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch master
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
```

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

>git checkout exp

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
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

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

>git stash --help //>git stash --help 참조하기

```
GIT-STASH(1)                      Git Manual                      GIT-STASH
(1)



NAME
       git-stash - Stash the changes in a dirty working directory away

SYNOPSIS
       git stash list [<options>]
       git stash show [<stash>]
       git stash drop [-q|--quiet] [<stash>]
       git stash ( pop | apply ) [--index] [-q|--quiet] [<stash>]
       git stash branch <branchname> [<stash>]
       git stash [save [-p|--patch] [-k|--[no-]keep-index] [-q|--quiet]
                    [-u|--include-untracked] [-a|--all] [<message>]]
       git stash clear
       git stash create [<message>]
       git stash store [-m|--message <message>] [-q|--quiet] <commit>
```

>git stash

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash
Saved working directory and index state WIP on exp: 1e6c395 1
HEAD is now at 1e6c395 1
```

working driectory와 index가 저장되었다. 어디에 ? exp에 저장되었다.

WIP(`WIP`)의 뜻은 작업중이라는 뜻이다.

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
```

```
nothing to commit, working tree clean
```

>vim f1.txt

```
a
~
~
```

더이상 f1.txt의 변경사항을 저장하고 있지 않다.

>git checkout master

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git checkout master
Switched to branch 'master'
```

>git checkout exp

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git checkout exp
Switched to branch 'exp'
```

더이상 checkout 하는데 commit되지 않은 부분이 있다는 메시지가 뜨지 않는다.

>git stash apply //>git stash --help 참조하기

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash apply
On branch exp
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

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

>vim f1.txt

```
a
b
~
```

변경사항이 다시 되살아 났다~!!!

>git stash list

```
stash@{0}: WIP on exp: 1e6c395 1
(END)
```

>git reset --hard HEAD //가장 최신상태의 commit 으로 바꾼다.

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git reset --hard
HEAD
HEAD is now at 1e6c395 1
```

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
nothing to commit, working tree clean
```

아무것도 commit 할것이 없다고 뜬다.

>git stash list

```
stash@{0}: WIP on exp: 1e6c395 1
(END)
```

아직 남아있는것을 볼 수 있다.

>git stash apply

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash apply
On branch exp
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

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

다시 복원된다 !!

>git reset --hard　　//사라지고~...
>git stash apply　　//다시복원되고!!!

결론 : 우리가 >git stash list 로 나타나는 것을 명시적으로 삭제하지 않을 경우 항상 살아있다.

>git reset --hard

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git reset --hard
HEAD is now at 1e6c395 1
```

>vim f2.txt

```
a
~
```

>git add f2.txt

>git stash

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash
Saved working directory and index state WIP on exp: 1e6c395 1
HEAD is now at 1e6c395 1
```

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
nothing to commit, working tree clean
```

>git stash list

```
stash@{0}: WIP on exp: 1e6c395 1
stash@{1}: WIP on exp: 1e6c395 1
(END)
```

2개의 stash가 나오고 위에것(`stash@{0}:`)이 최신이고 아래것(`stash@{1}:`)이 그이전의 것이다.

>git stash apply

제일위에있는 stash(`stash@{0}: WIP on exp: 1e6c395 1`)를 적용하게 된다.

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash apply
On branch exp
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   f2.txt
```

>git stash list

```
stash@{0}: WIP on exp: 1e6c395 1
stash@{1}: WIP on exp: 1e6c395 1
(END)
```

그런데 삭제가 되지 않는다!!...

이럴때는

>git stash drop

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash drop
Dropped refs/stash@{0} (1f1dcefcb5f330bbb29cdab8f7922009e4212436
```

으로 가장최신의 stash를 삭제한다.

>git stash list

```
stash@{0}: WIP on exp: 1e6c395 1
(END)
```

가장최신 stash가 삭제된것을 볼 수 있다.

>git stash apply; git stash drop; //적용하고 삭제해준다.

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash apply;
git stash drop;
On branch exp
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   f2.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

Dropped refs/stash@{0} (01e47a77b60667a572fcbd19895d20a91293d8a6)
```

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   f2.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
```

>git stash list

아무것도 나오지 않는다.

이것을 한번에 하는방법

>git reset --hard //초기화 시키고

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git reset --hard
HEAD is now at 1e6c395 1
```

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
nothing to commit, working tree clean
```

>vim f1.txt //수정

```
a
b
~
```

>git stash

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash
Saved working directory and index state WIP on exp: 1e6c395 1
HEAD is now at 1e6c395 1
```

숨기고

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
nothing to commit, working tree clean
```

숨겨진후 변경사항이 없어짐을 확인

>git stash pop //>git stash apply; git stash drop; 과 같은 명령어이다.

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash pop
On branch exp
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

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (d80a653e3e4035ada544e8f7829bda03a59a754e)
```

>git stash list

아무것도 나오지 않는다.

추가적으로

>git stash --help

```
NAME
       git-stash - Stash the changes in a dirty working directory away
```

stash는 변경사항을 감춘다는 이야기이다.

>ls -al

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% ls -al
total 8
drwxr-xr-x   4 gimdaewon  staff   136  4 11 20:39 .
```

```
drwx------+ 62 gimdaewon  staff  2108  4 11 18:38 ..
drwxr-xr-x  14 gimdaewon  staff   476  4 11 20:39 .git
-rw-r--r--   1 gimdaewon  staff     4  4 11 20:39 f1.txt
```

>git reset --hard

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git reset --hard
HEAD is now at 1e6c395 1
```

>vim f1.txt //파일수정

```
a
b
~
```

vimf2.txt //파일생성

```
a
~
~
```

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
On branch exp
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

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

    f2.txt

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

//f1.txt는 tracked(추적)되고 있는 파일이고 f2.txt는 Untracked(`Untracked`)되고있는 파일이다.

이상태에서

>git stash

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git stash
Saved working directory and index state WIP on exp: 1e6c395 1
HEAD is now at 1e6c395 1
```

>git status

```
gimdaewon@gimdaewon-ui-MacBook-Pro:~/documents/gitfth4% git status
```

```
On branch exp
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    f2.txt

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

f2.txt는 Untracked(`Untracked`)이기 때문에 stash 되지 않는다.