

Grunt & Git 활용

—
작성자: 조성민

소속팀 : UIT 개발실

작성년월일: 2014. 8. 10

대외비

목차

3. Git 저장소

3.4. Git 되돌리기(Undo)

4. Git Branch

4.1. Git Branch란?

4.2. Git Branch 생성

4.3. Git Branch 머지

4.4. Git Branch 삭제

4.5 Git Branch 머지 충돌

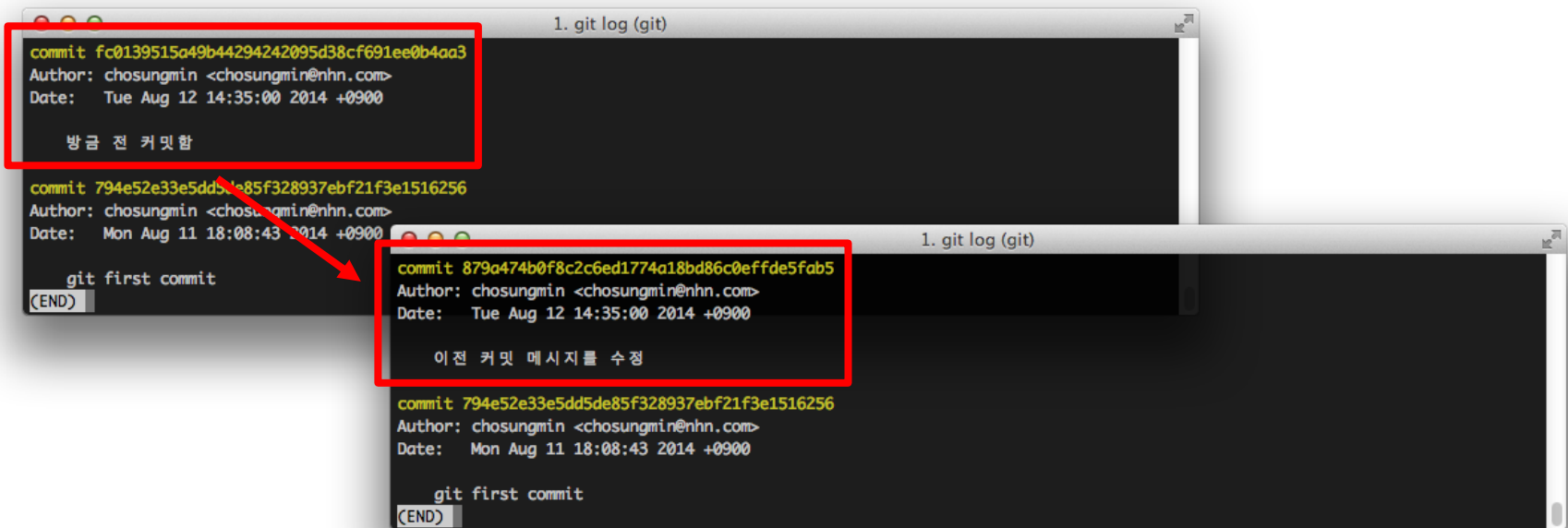
--- 3. Git 저장소

커밋 정보 수정하기

커밋 완료 후 바로 이전 커밋 정보를 수정해야 할 때(어떤 파일을 빼먹었을 때, 커밋 메시지를 잘못 적었을 때 등)는 이전 커밋을 수정하고 싶으면 **--amend** 옵션을 사용하면 된다.

```
$ git commit --amend -m '이전 커밋 메시지 수정'
```

```
$ git commit --amend -am '수정한 파일 정보를 이전 커밋에 포함시키고 메시지 수정'
```



파일 상태를 Unstage로 변경하기

실수로 add 한 파일(Staging Area)을 Unstage 상태로 되돌리는 방법은 아래와 같다.

```
$ git reset HEAD <file>
```

The image displays three overlapping terminal windows from a macOS environment, demonstrating the steps to unstage a file in Git. The top-left window shows the output of `git status`, indicating that `index.html` is modified and `test.html` is a new file. The middle window shows the command `git reset HEAD index.html` being entered, with a red box highlighting the command and a red arrow pointing to the next window. The bottom-right window shows the result of the command: `index.html` is now listed under 'Changes not staged for commit', while `test.html` remains under 'Changes to be committed'.

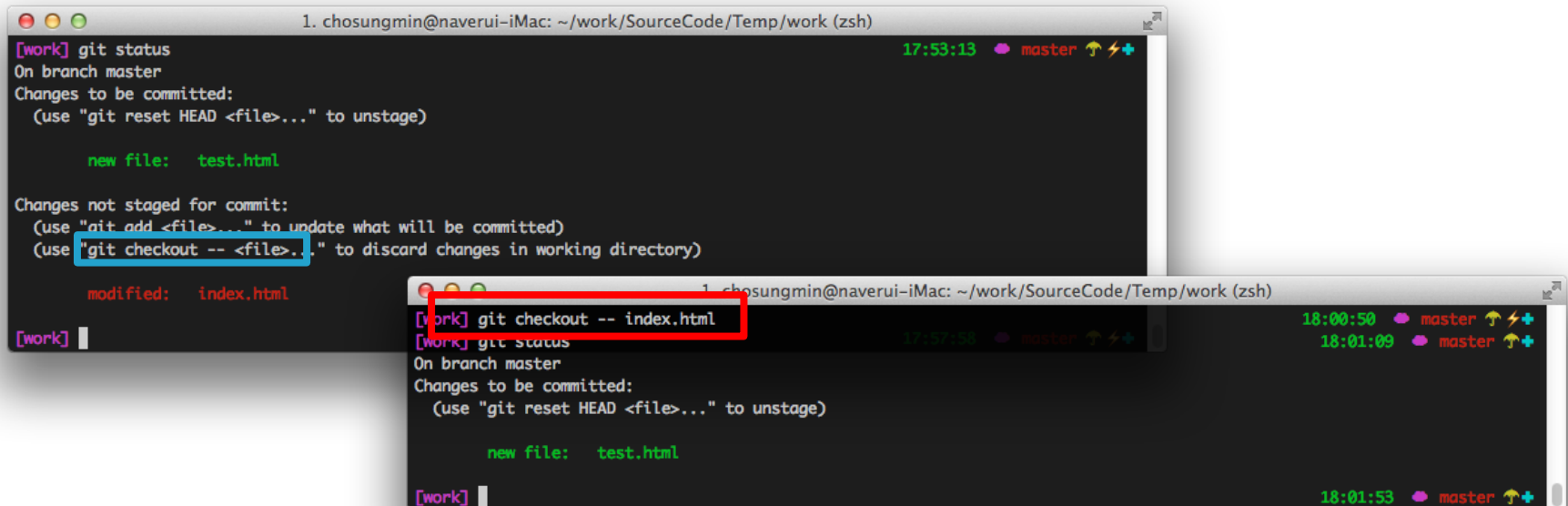
```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>.." to unstage changes)
    modified:   index.html
    new file:   test.html
[work]

1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git reset HEAD index.html
Unstaged changes after reset:
  M   index.html
[work] git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    new file:   test.html
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:   index.html
[work]
```

Modified 파일 되돌리기

실수로 add 한 파일(Staging Area)을 Unstage 상태로 되돌리는 방법은 아래와 같다.

```
$ git checkout -- <file>
```



The image shows two terminal windows from a macOS environment. The top window shows the output of `git status` on the master branch. It indicates that `test.html` is a new file staged for commit, while `index.html` is modified but not staged. The bottom window shows the command `git checkout -- index.html` being executed, which successfully unstages the modified `index.html` file. The terminal output after the command shows that `index.html` is no longer listed in the staged changes.

```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   test.html

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:   index.html

[work]

1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git checkout -- index.html
[work] git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   test.html

[work]
```

4. Git Branch

Git 브랜치

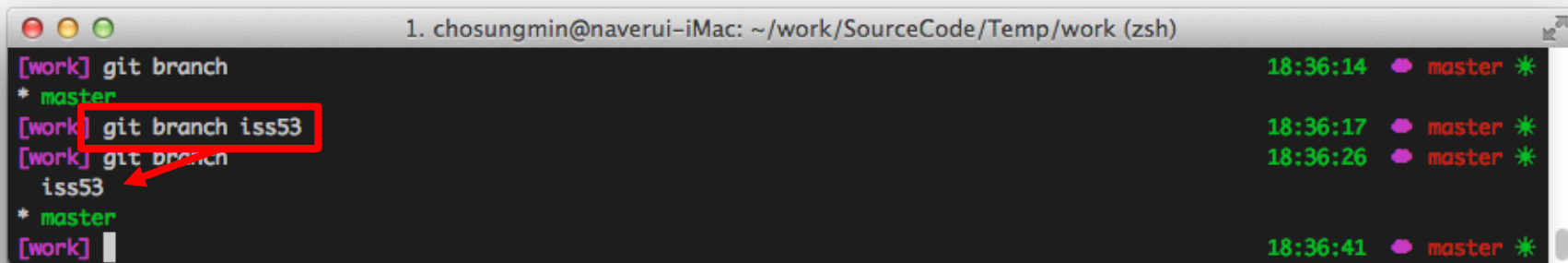
개발을 하다 보면 코드를 여러 개로 복사해야 하는 일이 자주 생긴다.

코드를 통째로 복사하고 나서 원래 코드와는 상관없이 독립적으로 개발을 진행할 수 있는데, 이렇게 독립적으로 개발하는 것이 브랜치다.

1. Git의 브랜치는 매우 가볍다.
2. 순식간에 브랜치를 새로 만들고 브랜치 사이를 이동할 수 있다.
3. 다른 버전 관리 시스템과는 달리 Git은 브랜치를 만들어 작업하고 나중에 Merge하는 방법을 권장한다.
4. 심지어 하루에 수십번씩해도 괜찮다.

Git Branch 생성

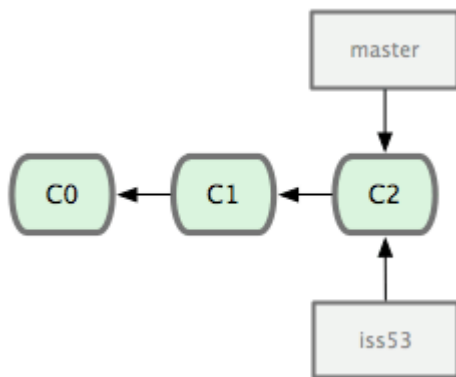
\$ git branch <브랜치명>



```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git branch
* master
[work] git branch iss53
[work] git branch
iss53
* master
[work]
```

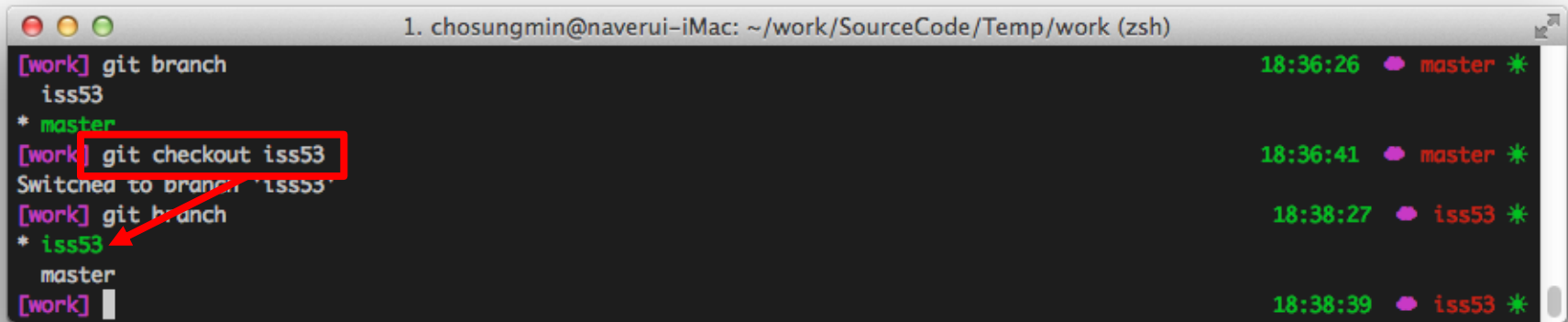
The terminal screenshot shows a sequence of commands to create a new branch. The first command 'git branch' lists the current branch 'master'. The second command 'git branch iss53' is highlighted with a red box and an arrow, indicating the creation of a new branch. The third command 'git branch' shows both 'master' and 'iss53' as available branches, with 'master' being the active one.

※ “git branch”만 입력하면 내 저장소에 있는 branch를 볼 수 있다.



Git Branch 이동

\$ git checkout <브랜치명>



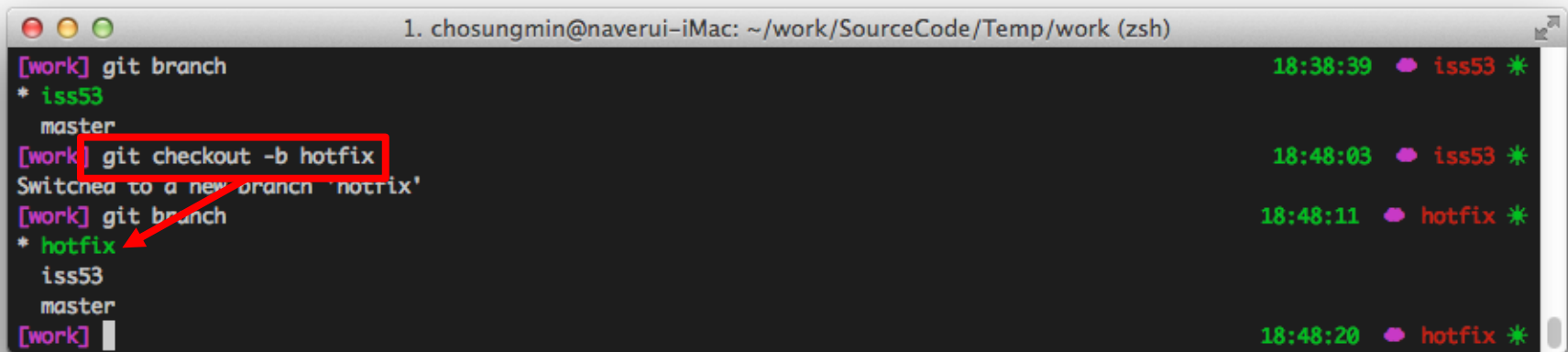
A terminal window titled "1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)" showing a sequence of Git commands and their outputs. The commands are: `git branch`, `git checkout iss53` (highlighted with a red box), and `git branch` again. The outputs show the current branch switching from `master` to `iss53`. A red arrow points from the `iss53` argument in the `git checkout` command to the `iss53` branch name in the subsequent `git branch` output.

```
[work] git branch
iss53
* master
[work] git checkout iss53
Switched to branch 'iss53'
[work] git branch
* iss53
master
[work]
```

18:36:26 🍌 master ✨
18:36:41 🍌 master ✨
18:38:27 🍌 iss53 ✨
18:38:39 🍌 iss53 ✨

Git Branch 생성 후 바로 이동

```
$ git checkout -b <브랜치명>
```



A terminal window titled "1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)" showing the following commands and output:

```
[work] git branch
* iss53
master
[work] git checkout -b hotfix
Switched to a new branch 'hotfix'
[work] git branch
* hotfix
iss53
master
[work]
```

On the right side of the terminal, there are timestamps and branch names: "18:38:39 iss53", "18:48:03 iss53", "18:48:11 hotfix", and "18:48:20 hotfix". A red box highlights the command "git checkout -b hotfix", and a red arrow points from it to the "hotfix" branch name in the subsequent "git branch" output.

Git Branch 머지

\$ git checkout <수정된 내용이 합쳐질 브랜치명>

\$ git merge <머지할 브랜치명>

```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)

[work] git status
On branch hotfix
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.html
        new file:   new_file.html

[work] git commit -m '커밋'
[hotfix 5b5614e] 커밋
2 files changed, 14 insertions(+)
create mode 100644 new_file.html

[work] git checkout master
Switched to branch 'master'

[work] git merge hotfix
Updating 5b5614e..5b5614e
Fast-forward
 index.html | 1 +
new_file.html | 13 ++++++++
2 files changed, 14 insertions(+)
create mode 100644 new_file.html

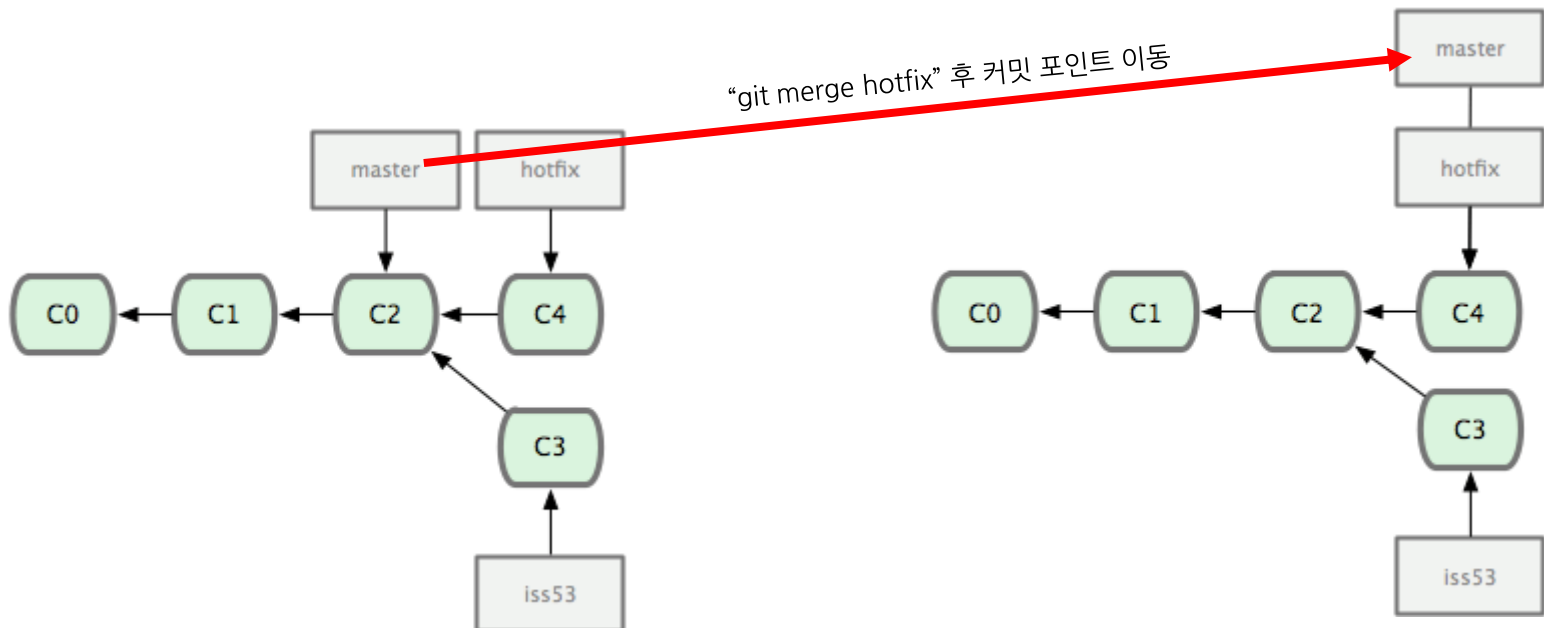
[work] git status
On branch master
nothing to commit, working directory clean

[work]
```

Git Branch 머지 > Fast forward 머지

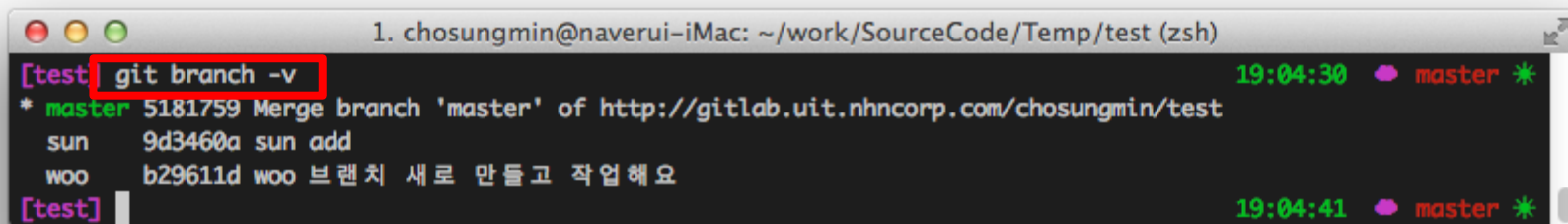
Merge할 브랜치가 가리키고 있던 커밋이 현 브랜치가 가리키는 것보다 '앞으로 진행한' 커밋이기 때문에 master 브랜치 포인터는 최신 커밋으로 이동한다.

이런 Merge 방식을 'Fast forward'라고 부른다.



각 Branch 마지막 커밋 메시지 보기

```
$ git branch -v
```

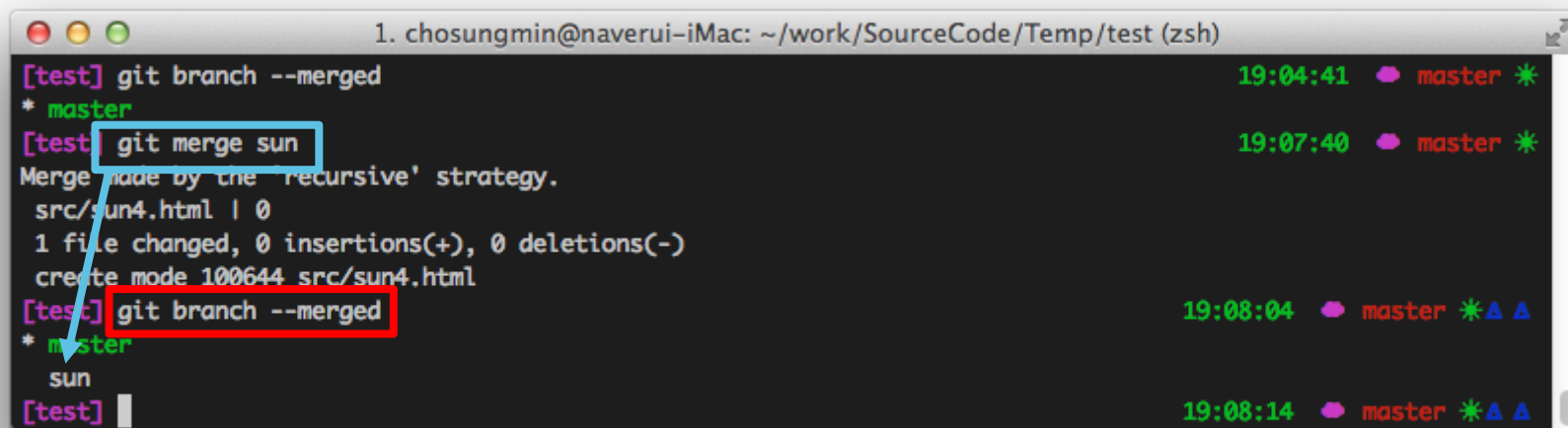


```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/test (zsh)
[test] git branch -v 19:04:30 master *
* master 5181759 Merge branch 'master' of http://gitlab.uit.nhncorp.com/chosungmin/test
  sun    9d3460a sun add
  woo    b29611d woo 브랜치 새로 만들고 작업해요
[test] 19:04:41 master *
```

현 Branch에 머지한 브랜치 목록 보기

```
$ git branch --merged
```

현재 Checkout한 브랜치를 기준으로 Merge된 브랜치인지 그렇지 않은지 필터링해 볼 수 있다.



A terminal window titled "1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/test (zsh)" showing a sequence of Git commands and their outputs. The commands and outputs are as follows:

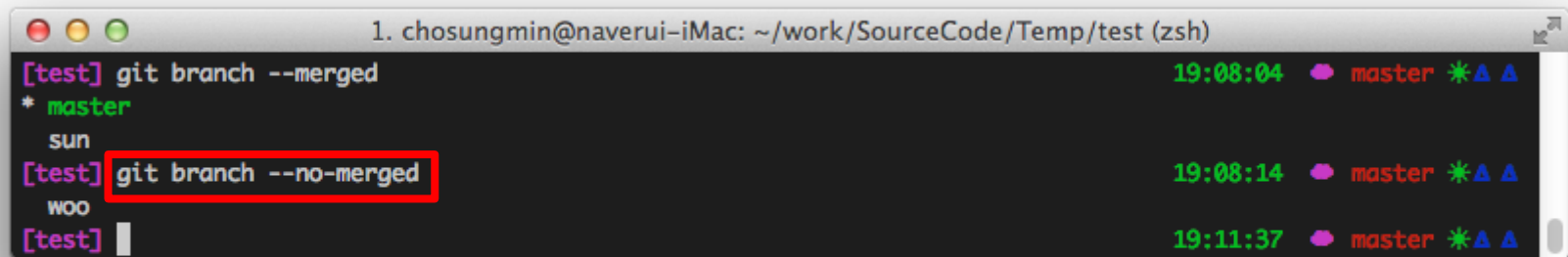
- `[test] git branch --merged` at 19:04:41, output: `* master`
- `[test] git merge sun` at 19:07:40, output: `Merge made by the 'recursive' strategy.
src/sun4.html | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 src/sun4.html`
- `[test] git branch --merged` at 19:08:04, output: `* master`
- `[test]` at 19:08:14, output: `sun`

Annotations in the image: A blue box highlights the `git merge sun` command, and a red box highlights the `git branch --merged` command. A blue arrow points from the `git merge sun` command to the `git branch --merged` command.

현 Branch에 머지하지 않은 브랜치 목록 보기

```
$ git branch --no-merged
```

현재 Checkout한 브랜치에 Merge하지 않은 브랜치를 살펴볼 수 있다.

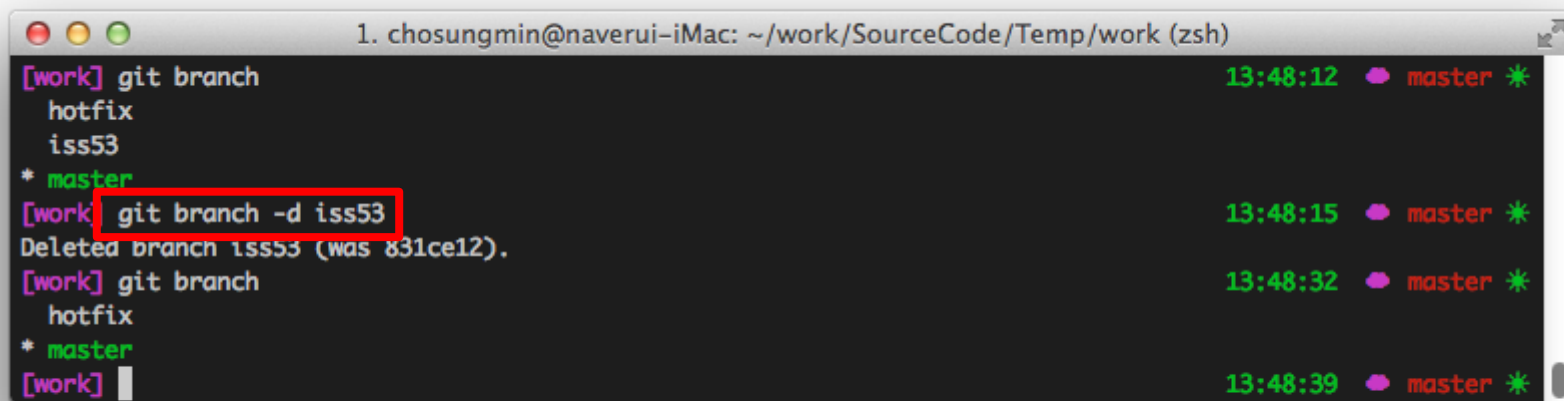


A terminal window titled "1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/test (zsh)" showing the execution of Git commands. The first command is `[test] git branch --merged`, which outputs `* master` followed by a list of branches: `sun`, `[test]`, `woo`, and `[test]`. The second command is `[test] git branch --no-merged`, which is highlighted with a red box and outputs `[test]`. The terminal also shows timestamps and status indicators for each command.

```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/test (zsh)
[test] git branch --merged                                     19:08:04  master *▲▲
* master
sun
[test] git branch --no-merged                                  19:08:14  master *▲▲
woo
[test]                                                         19:11:37  master *▲▲
```


Git Branch 삭제

```
$ git branch -d <삭제할 브랜치명>
```



```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git branch                                     13:48:12  master *
      hotfix
      iss53
* master
[work] git branch -d iss53                             13:48:15  master *
Deleted branch iss53 (was 831ce12).
[work] git branch                                     13:48:32  master *
      hotfix
* master
[work]                                                13:48:39  master *
```

Git Branch 강제 삭제

```
$ git branch -D <삭제할 브랜치명>
```

삭제하려는 브랜치에 머지하지 않는 정보가 있다면 “-d” 옵션으로 브랜치를 삭제할 수 없다.

이럴 때는 **강제 옵션인 “-D”(대문자)**를 사용하여 브랜치를 삭제하면 된다.

```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git branch
* hotfix
  master
13:52:30 hotfix ⬆⚡

[work] git add .
13:52:36 hotfix ⬆⚡
[work] git commit -m '브랜치 삭제 안될걸'
13:52:41 hotfix ⬆+
[hotfix ac203f5] 브랜치 삭제 안될걸
1 file changed, 2 insertions(+), 2 deletions(-)

[work] git checkout master
Switched to branch 'master'
13:53:11 hotfix ✨

[work] git branch -d hotfix
13:53:30 master ✨
error: The branch 'hotfix' is not fully merged.
If you are sure you want to delete it, run 'git branch -D hotfix'.

[work] git branch -D hotfix
13:53:35 master ✨
Deleted branch hotfix (was ac203f5).

[work] git branch
13:53:45 master ✨
* master

[work]
13:54:37 master ✨
```

Git Branch 머지 충돌 해결 (계속...)

```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)

[work] git status                                     14:25:16  🍆 iss53 ⚡
On branch iss53
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:   index.html

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

[work] git add .                                     14:25:29  🍆 iss53 ⚡
[work] git commit -m 'test'                          14:26:20  🍆 iss53 ⚡
[iss53 407e625] test
1 file changed, 6 insertions(+)

[work] git co master                                14:26:32  🍆 iss53 ✨
Switched to branch 'master'
[work] git commit -am 'commit test'                  14:26:35  🍆 master ✨
[master 2af557d] commit test
1 file changed, 4 insertions(+)

[work] git merge iss53                               14:27:46  🍆 master ✨
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.

[work]                                               14:28:25  🍆 master ⚡
```

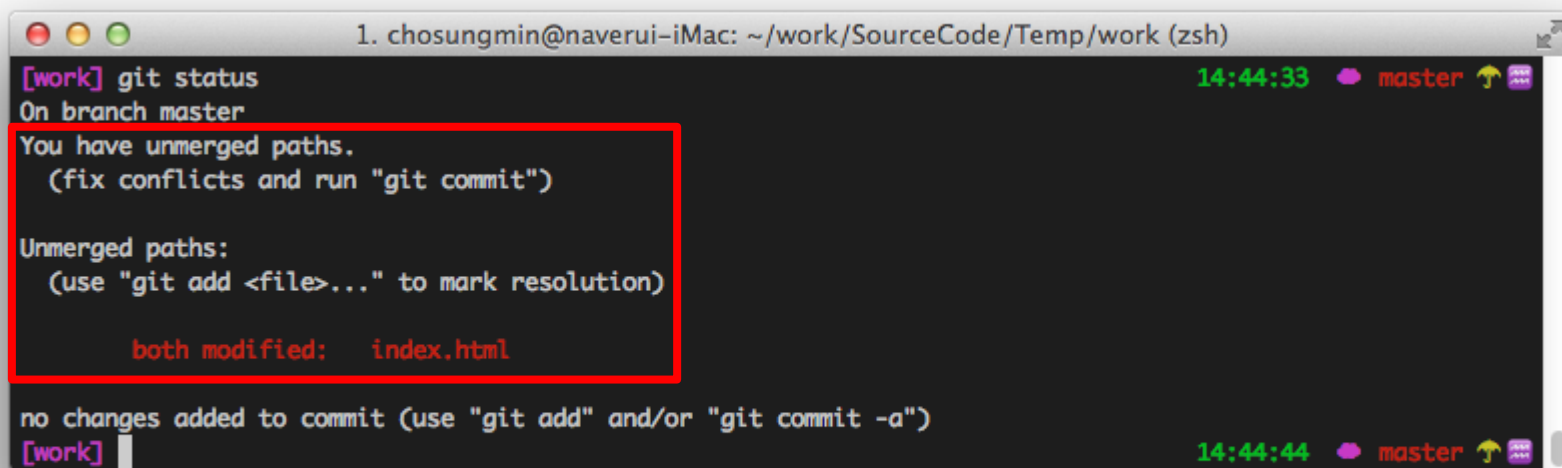
iss53 브랜치에서 작업 후 커밋

master 브랜치에서 비슷한 영역 작업 후 커밋

머지 했더니 충돌 발생

Git Branch 머지 충돌 해결 (계속...)

Merge 충돌이 일어났을 때 Git이 어떤 파일을 Merge할 수 없었는지 살펴보려면 "git status" 명령을 이용한다.



```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)
[work] git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

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

        both modified:   index.html

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

Git Branch 머지 충돌 해결 (계속...)

The diagram illustrates the resolution of a Git merge conflict between the HEAD (master) branch and the iss53 branch. It shows two versions of the `index.html` file.

Left Editor (Conflicting State):

- Lines 1-10: Standard HTML boilerplate.
- Line 11: `<<<<<<<< HEAD` (Annotated with a blue box and arrow: "HEAD(master) 버전에서 추가된 내용")
- Line 12: `이렇게 하면 충돌이 나겠지...`
- Line 13: `목감기 걸렸더니 몸이....ㅠ_ㅠ`
- Line 14: `>>>>>>> iss53` (Annotated with a red box and arrow: "iss53 브랜치에서 추가된 내용")
- Line 15: ``
- Line 16: `1`
- Line 17: `2`
- Line 18: `3`
- Line 19: ``
- Line 20: `충돌나게 처리해야지^^`
- Line 21: `>>>>>>> iss53`
- Line 22: `</body>`
- Line 23: `</html>`

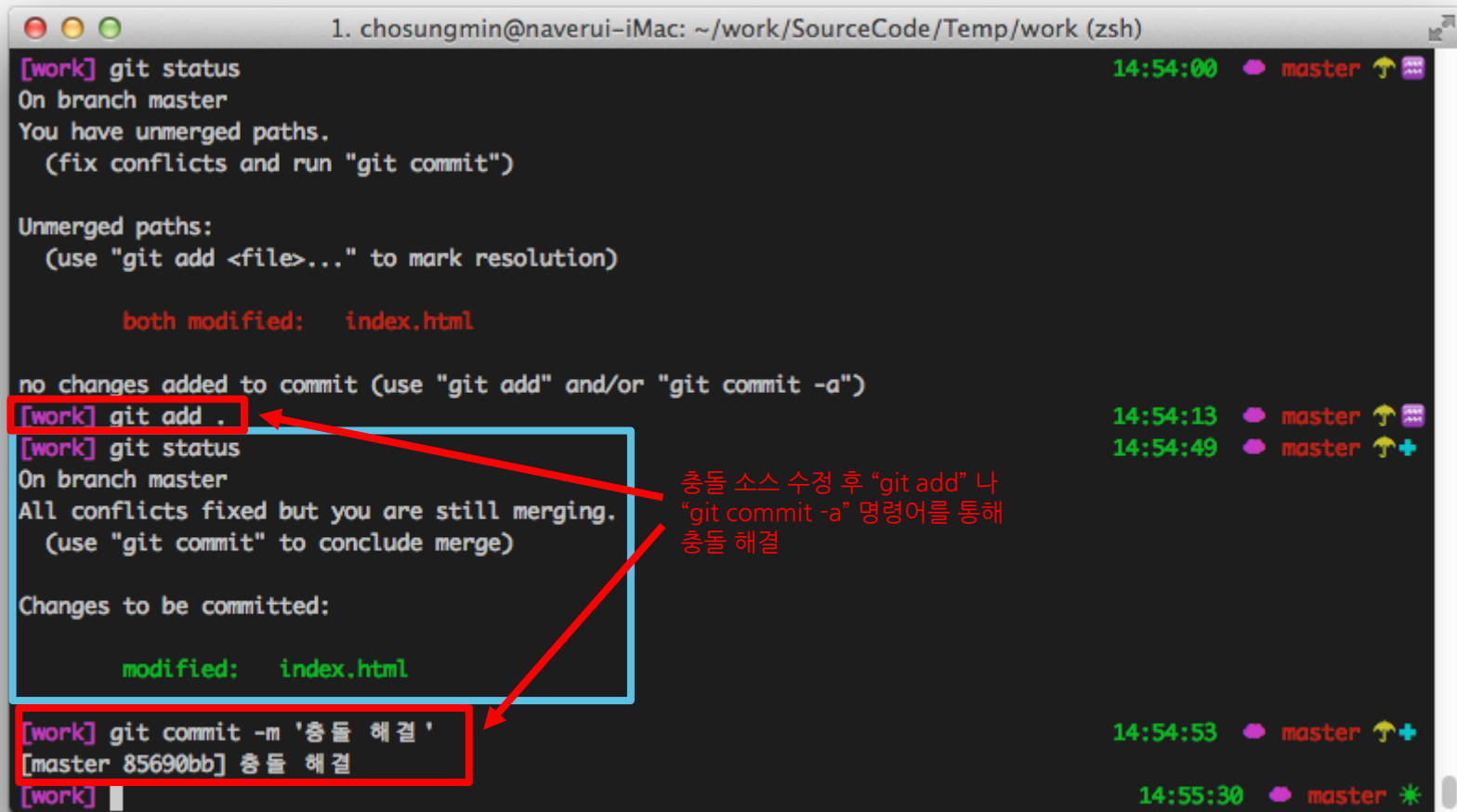
Right Editor (Resolved State):

- Lines 1-10: Standard HTML boilerplate.
- Line 11: `이렇게 하면 충돌이 나겠지... 역시 충돌 났어 ^^`
- Line 12: `목감기 걸렸더니 몸이....ㅠ_ㅠ`
- Line 13: ``
- Line 14: `1`
- Line 15: `2`
- Line 16: `3`
- Line 17: ``
- Line 18: `충돌나게 했으니 충돌 해결하고 저장!!!`
- Line 19: `</body>`
- Line 20: `</html>`

Resolution Process:

- A large orange arrow labeled "충돌 해결" (Conflict Resolution) points from the left editor to the right editor.
- Red arrows indicate the source of the conflicting changes: one from the HEAD branch to line 12, and another from the iss53 branch to line 18.

Git Branch 머지 충돌 해결 (계속...)



The terminal window shows the process of resolving a merge conflict on the master branch. The user runs 'git status' and sees 'both modified: index.html'. They then run 'git add .' and 'git commit -m \'충돌 해결\''. The terminal output shows the conflict is resolved and the commit is successful.

```
1. chosungmin@naverui-iMac: ~/work/SourceCode/Temp/work (zsh)

[work] git status
On branch master
You have unmerged paths.
(fix conflicts and run "git commit")

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

      both modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
[work] git add .
[work] git status
On branch master
All conflicts fixed but you are still merging.
(use "git commit" to conclude merge)

Changes to be committed:
      modified:   index.html

[work] git commit -m '충돌 해결'
[master 85690bb] 충돌 해결
[work]
```

충돌 소스 수정 후 "git add" 나
"git commit -a" 명령어를 통해
충돌 해결

Thank you.