3.2 Git 브랜치 - 브랜치와 Merge 의 기초

**브랜치와 Merge 의 기초**

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

1. 웹사이트가 있고 뭔가 작업을 진행하고 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 master 브랜치에 커밋을 몇 번 했다고 가정한다.



그림 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



그림 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]'



그림 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](https://git-scm.com/book/ko/v2/ch00/_git_stashing) 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. hotfix 라는 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(+)



그림 21. master 브랜치에서 갈라져 나온 hotfix 브랜치

운영 환경에 적용하려면 문제를 제대로 고쳤는지 테스트하고 최종적으로 운영환경에 배포하기 위해 hotfix 브랜치를 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 hotfix 는 master 브랜치에 포함됐고 운영환경에 적용할 수 있는 상태가 되었다고 가정해보자.



그림 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(+)



그림 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.

index.html | 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.



그림 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.



그림 25. Merge 커밋

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

$ git branch -d iss53

**충돌의 기초**

가끔씩 3-way Merge가 실패할 때도 있다. Merge 하는 두 브랜치에서 같은 파일의 한 부분을 동시에 수정하고 Merge 하면 Git은 해당 부분을 Merge 하지 못한다. 예를 들어, 53번 이슈와 hotfix 가 같은 부분을 수정했다면 Git은 Merge 하지 못하고 아래와 같은 충돌(Conflict) 메시지를 출력한다.

$ git merge iss53

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

Git은 자동으로 Merge 하지 못해서 새 커밋이 생기지 않는다. 변경사항의 충돌을 개발자가 해결하지 않는 한 Merge 과정을 진행할 수 없다. Merge 충돌이 일어났을 때 Git이 어떤 파일을 Merge 할 수 없었는지 살펴보려면 git status 명령을 이용한다.

$ 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")

충돌이 일어난 파일은 unmerged 상태로 표시된다. Git은 충돌이 난 부분을 표준 형식에 따라 표시해준다. 그러면 개발자는 해당 부분을 수동으로 해결한다. 충돌 난 부분은 아래와 같이 표시된다.

<<<<<<< HEAD:index.html

<div id="footer">contact : email.support@github.com</div>

=======

<div id="footer">

please contact us at support@github.com

</div>

>>>>>>> iss53:index.html

======= 위쪽의 내용은 HEAD 버전(merge 명령을 실행할 때 작업하던 master 브랜치)의 내용이고 아래쪽은 iss53 브랜치의 내용이다. 충돌을 해결하려면 위쪽이나 아래쪽 내용 중에서 고르거나 새로 작성하여 Merge 한다. 아래는 아예 새로 작성하여 충돌을 해결하는 예제다.

<div id="footer">

please contact us at email.support@github.com

</div>

충돌한 양쪽에서 조금씩 가져와서 새로 수정했다. 그리고 <<<<<<<, =======, >>>>>>> 가 포함된 행을 삭제했다. 이렇게 충돌한 부분을 해결하고 git add 명령으로 다시 Git에 저장한다.

다른 Merge 도구도 충돌을 해결할 수 있다. git mergetool 명령으로 실행한다.

$ git mergetool

This message is displayed because 'merge.tool' is not configured.

See 'git mergetool --tool-help' or 'git help config' for more details.

'git mergetool' will now attempt to use one of the following tools:

opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge ecmerge p4merge araxis bc3 codecompare vimdiff emerge

Merging:

index.html

Normal merge conflict for 'index.html':

{local}: modified file

{remote}: modified file

Hit return to start merge resolution tool (opendiff):

기본 도구 말고 사용할 수 있는 다른 Merge 도구도 있는데(Mac에서는 opendiff 가 실행된다), “one of the following tools.” 부분에 보여준다. 여기에 표시된 도구 중 하나를 고를 수 있다.

|  |  |
| --- | --- |
| **노트** | Merge 시에 발생한 충돌을 다루는 더 어렵고 요상한 내용은 뒤에 [고급 Merge](https://git-scm.com/book/ko/v2/ch00/_advanced_merging) 에서 다루기로 한다. |

Merge 도구를 종료하면 Git은 잘 Merge 했는지 물어본다. 잘 마쳤다고 입력하면 자동으로 git add 가 수행되고 해당 파일이 Staging Area에 저장된다. git status 명령으로 충돌이 해결된 상태인지 다시 한번 확인해볼 수 있다.

$ 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

충돌을 해결하고 나서 해당 파일이 Staging Area에 저장됐는지 확인했으면 git commit 명령으로 Merge 한 것을 커밋한다. 충돌을 해결하고 Merge 할 때는 커밋 메시지가 아래와 같다.

Merge branch 'iss53'

Conflicts:

index.html

#

# 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.

# On branch master

# All conflicts fixed but you are still merging.

#

# Changes to be committed:

# modified: index.html

#

어떻게 충돌을 해결했고 좀 더 확인해야 하는 부분은 무엇이고 왜 그렇게 해결했는지에 대해서 자세하게 기록한다. 자세한 기록은 나중에 이 Merge 커밋을 이해하는데 도움을 준다.