**브랜치 (Branch)**

**브랜치 만들기**

Git 에서는 작업에 따라 자유롭게 브랜치를 만들 수 있습니다. 그러나 이것을 효과적으로 관리하려면, 먼저 함께 작업할 팀원들과 어떠한 방식으로 브랜치를 만들고 통합할 것인지 미리 정해두는 것이 좋습니다. 예를 들어 새로운 브랜치를 만들 때에 브랜치 이름은 어떤 규칙으로 지을 것인지 또는 어떤 상황에서 브랜치를 만들 지, 어느 시점에 통합할 것인지 등등 규칙은 정하기 나름입니다.

자, 그럼 이제부터 우리가 만들 수 있는 브랜치에는 어떤 종류가 있는지 살펴 보도록 하겠습니다.

**통합 브랜치(Integration Branch)**

통합 브랜치란 언제든지 배포할 수 있는 버전을 만들 수 있어야 하는 브랜치 입니다. 그렇기 때문에 늘 안정적인 상태를 유지하는 것이 중요합니다. 여기서 '안정적인 상태'란 현재 작업 중인 소스코드가 모바일에서 동작하는 어플리케이션을 개발하기 위한 것이라면, '그 어플리케이션의 모든 기능이 정상적으로 동작하는 상태'를 의미합니다.

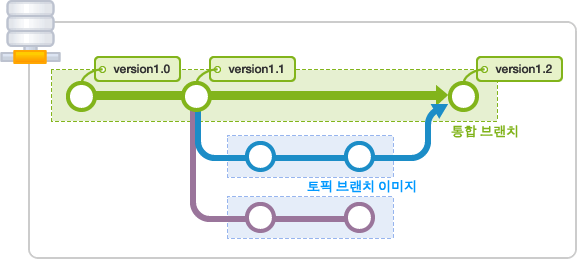
만약 이 어플리케이션에 어떤 문제가 발견되어 그 문제(버그)를 수정한다던지 새로운 기능을 추가해야 한다던지 해야할 때, 바로 '토픽 브랜치(Topic branch)'를 만들 수 있습니다. 처음에는 보통 통합 브랜치에서 토픽 브랜치를 만들어 냅니다.

일반적으로 저장소를 처음 만들었을 때에 생기는 'master' 브랜치를 통합 브랜치로 사용합니다.

**토픽 브랜치(Topic Branch)**

토픽 브랜치란, 기능 추가나 버그 수정과 같은 단위 작업을 위한 브랜치 입니다. 여러 개의 작업을 동시에 진행할 때에는, 그 수만큼 토픽 브랜치를 생성할 수 있습니다.

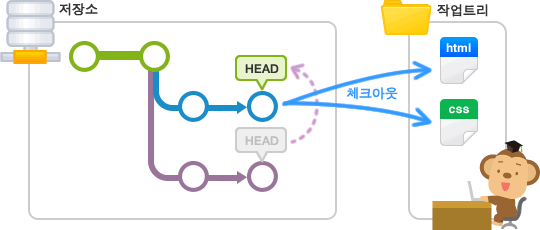
토픽 브랜치는 보통 통합 브랜치로부터 만들어 내며, 토픽 브랜치에서 특정 작업이 완료되면 다시 통합 브랜치에 병합하는 방식으로 진행됩니다. 이러한 토픽 브랜치는 '피처 브랜치(Feature branch)' 라고 부르기도 합니다.



**브랜치 (Branch)**

# 브랜치 전환하기

Git 에서는 항상 작업할 브랜치를 미리 선택해야 합니다. 처음에 Git을 설치하게 되면 'master' 브랜치가 선택되어 있죠. 현재 선택된 브랜치가 아닌 다른 브랜치에서 작업하고 싶을 때에는, '체크아웃(checkout)' 명령어를 실행하여 원하는 브랜치로 전환할 수 있습니다. 체크아웃을 실행하면, 우선 브랜치 안에 있는 마지막 커밋 내용이 작업 트리에 펼쳐집니다. 브랜치가 전환 되었으므로 이 후에 실행한 커밋은 전환한 브랜치에 추가됩니다.

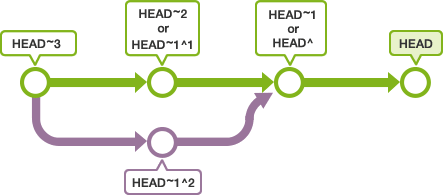


## HEAD

'HEAD' 란 현재 사용 중인 브랜치의 선두 부분을 나타내는 이름입니다. 기본적으로는 'master'의 선두 부분을 나타냅니다. 'HEAD' 를 이동하면, 사용하는 브랜치가 변경됩니다.

Note

커밋을 지정할 때, '~(틸드, 물결기호)'와 '^(캐럿, 삽입기호)'을 사용하여 현재 커밋으로부터 특정 커밋의 위치를 가리킬 수 있습니다. 이 때 자주 사용하는 것이 'HEAD' 로서, '~(틸드)'와 숫자를 'HEAD' 뒤에 붙여 몇 세대 앞의 커밋을 가리킬 수 있습니다. '^(캐럿)'은, 브랜치 병합에서 원본이 여럿 있는 경우 몇 번째 원본인지를 지정할 수 있습니다.

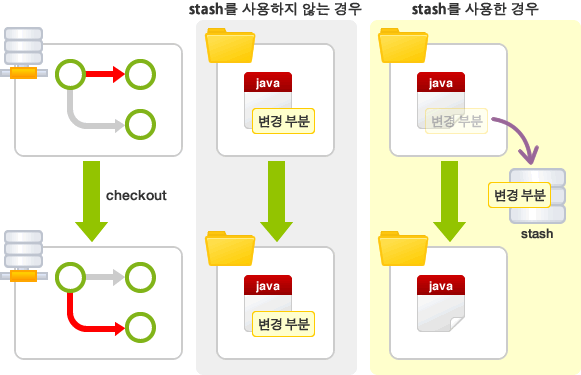


## stash

커밋하지 않은 변경 내용이나 새롭게 추가한 파일이 인덱스와 작업 트리에 남아 있는 채로 다른 브랜치로 전환(checkout)하면, 그 변경 내용은 기존 브랜치가 아닌 전환된 브랜치에서 커밋할 수 있습니다.

단, 커밋 가능한 변경 내용 중에 전환된 브랜치에서도 한 차례 변경이 되어 있는 경우에는 체크아웃에 실패할 수 있습니다. 이 경우 이전 브랜치에서 커밋하지 않은 변경 내용을 커밋하거나, stash 를 이용해 일시적으로 변경 내용을 다른 곳에 저장하여 충돌을 피하게 한 뒤 체크아웃을 해야 합니다.

stash 란, 파일의 변경 내용을 일시적으로 기록해두는 영역입니다. stash 를 사용하여 작업 트리와 인덱스 내에서 아직 커밋하지 않은 변경을 일시적으로 저장해 둘 수 있습니다. 이 stash 에 저장된 변경 내용은 나중에 다시 불러와 원래의 브랜치나 다른 브랜치에 커밋할 수 있습니다.



**브랜치 (Branch)**

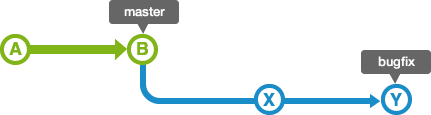
# 브랜치 통합하기

작업이 완료된 토픽 브랜치는 최종적으로 통합 브랜치에 병합됩니다. 브랜치 통합에는merge 를 사용하는 방법과 'rebase'를 사용하는 방법의 2가지 종류가 있습니다. 어느 쪽을 사용하느냐에 따라 통합 후의 브랜치의 이력이 크게 달라집니다.

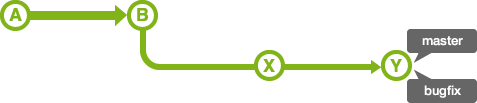
## merge

merge 를 사용하면, 여러 개의 브랜치를 하나로 모을 수 있습니다.

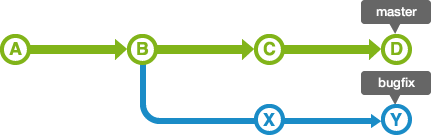
예를 들어, 아래 그림과 같이 'master' 브랜치에서 분기하는 'bugfix'라는 브랜치가 있다고 가정해 봅시다.



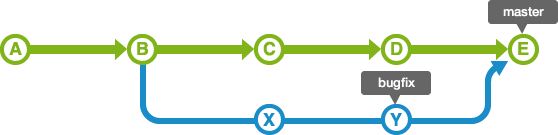
이 'bugfix' 브랜치를 'master' 브랜치로 병합할 때, 'master' 브랜치의 상태가 이전부터 변경되어 있지만 않으면 매우 쉽게 병합할 수 있습니다. 'bugfix' 브랜치의 이력은 'master' 브랜치의 이력을 모두 포함하고 있기 때문에, 'master' 브랜치는 단순히 이동하기만 해도 'bugfix' 브랜치의 내용을 적용할 수 있습니다. 또한 이 같은 병합은 'fast-forward(빨리 감기) 병합'이라고 부릅니다.



하지만 'bugfix' 브랜치를 분기한 이후에 'master' 브랜치에 여러 가지 변경 사항이 적용되는 경우도 있습니다. 이 경우에는 'master' 브랜치 내의 변경 내용과 'bugfix' 브랜치 내의 변경 내용을 하나로 통합할 필요가 있습니다.

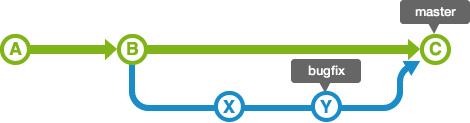


따라서 양쪽의 변경을 가져온 'merge commit(병합 커밋)'을 실행하게 됩니다. 병합 완료 후, 통합 브랜치인 'master' 브랜치로 통합된 이력이 아래 그림과 같이 생기게 됩니다.



Note

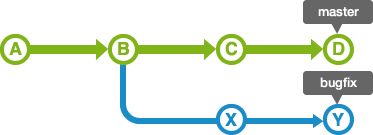
병합 실행 시에 'fast-forward 병합'이 가능한 경우라도 'non fast-forward 병합' 옵션을 지정하여 아래 그림과 같이 만들어 낼 수도 있습니다.



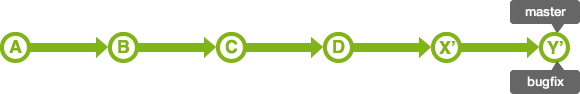
'non fast-forward 병합'을 실행하면, 브랜치가 그대로 남기 때문에 그 브랜치로 실행한 작업 확인 및 브랜치 관리 면에서 더 유용할 수 있습니다.

## rebase

위와 마찬가지로, 'master' 브랜치에서 분기하는 'bugfix' 브랜치가 있다고 가정합니다.

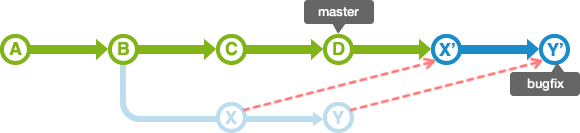


이제 rebase 를 이용해 어떻게 브랜치를 통합할 수 있는지 알아볼 차례 입니다. 아래 그림과 같이 'non fast-forward 병합' 방식으로 진행되는 시나리오를 만들어 봅시다.



우선 'bugfix' 브랜치를 'master' 브랜치에 rebase 하면, 'bugfix' 브랜치의 이력이 'master' 브랜치 뒤로 이동하게 됩니다. 그 때문에 그림과 같이 이력이 하나의 줄기로 이어지게 됩니다.

이 때 이동하는 커밋 X와 Y 내에 포함되는 내용이 'master'의 커밋된 버전들과 충돌하는 부분이 생길 수 있습니다. 그 때는 각각의 커밋에서 발생한 충돌 내용을 수정할 필요가 있습니다.



'rebase'만 하면 아래 그림에서와 같이, 'master'의 위치는 그대로 유지됩니다. 'master' 브랜치의 위치를 변경하기 위해서는 'master' 브랜치에서 'bugfix' 브랜치를 fast-foward(빨리감기) 병합 하면 됩니다.



Note

merge 와 rebase 는 통합 브랜치에 토픽 브랜치를 통합하고자 하는 목적은 같으나, 그 특징은 약간 다릅니다.

* **merge**  
  변경 내용의 이력이 모두 그대로 남아 있기 때문에 이력이 복잡해짐.
* **rebase**  
  이력은 단순해지지만, 원래의 커밋 이력이 변경됨. 정확한 이력을 남겨야 할 필요가 있을 경우에는 사용하면 안됨.

merge 와 rebase 는 팀 운용 방침에 따라 구별해 쓸 수 있습니다.   
예를 들어 이력을 하나로 모두 모아서 처리하도록 운용한다고 치면 아래와 같이 구별해 사용할 수 있습니다.

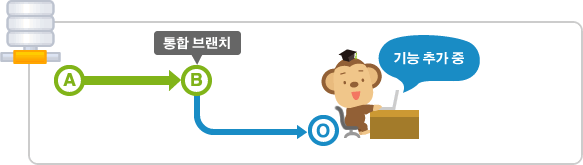
* 토픽 브랜치에 통합 브랜치의 최신 코드를 적용할 경우에는 rebase 를 사용,
* 통합 브랜치에 토픽 브랜치를 불러올 경우에는 우선 rebase 를 한 후 merge

**브랜치 (Branch)**

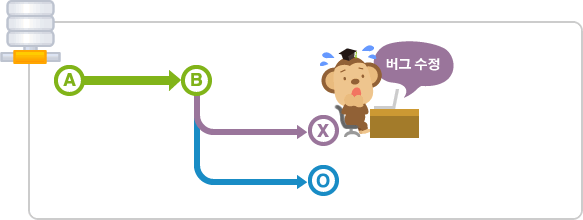
**토픽 브랜치와 통합 브랜치에서의 작업 흐름 파악하기**

토픽 브랜치와 통합 브랜치를 사용한 작업은 어떠한 순서로 진행되는 지, 간단한 예를 통해 알아보도록 하겠습니다.

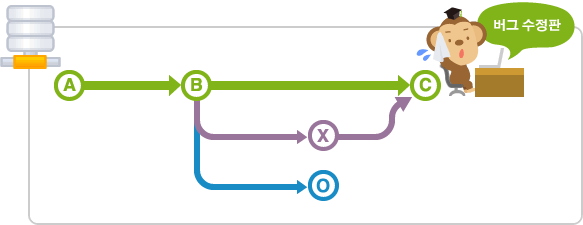
다음과 같이 토픽 브랜치에서 새로운 기능을 추가하는 작업과 버그 수정 작업을 동시에 진행 하는 경우를 생각해 봅시다.



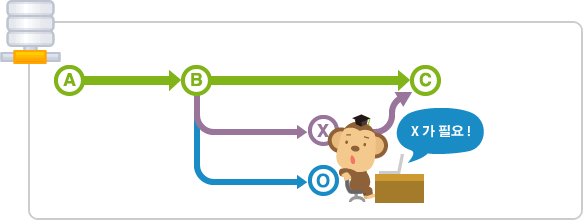
일단 통합 브랜치로부터 새롭게 버그 수정용 토픽 브랜치를 만들어, 새로운 기능을 추가하는 작업과는 별개로 버그 수정 작업을 진행할 수 있습니다.



버그 수정을 완료한 후, 통합 브랜치와 버그 수정용 토픽 브랜치를 병합하여 수정된 버전을 만들어 낼 수 있습니다.

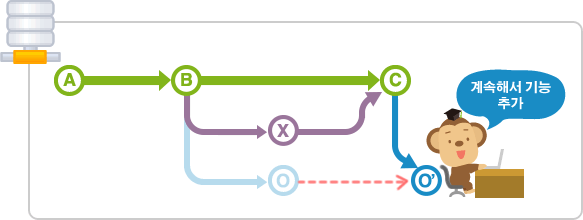


버그도 수정 했으니, 다시 원래 브랜치로 돌아와서 새로운 기능 추가 작업을 계속 진행하려 합니다.



그러나, 작업을 진행하려고 봤더니 앞서 적용한 커밋 X 의 버그가 수정된 버전의 소스코드를 지금의 커밋 O 에도 적용해야만 한다는 사실을 알게 되었습니다. 여기서 커밋 X 의 내용을 적용하려면, 직접 merge 하는 방법과 커밋 X 를 적용한 통합 브랜치에 rebase 하는 방법이 있습니다.

여기서는 통합 브랜치에 rebase 하는 방법을 이용해 보겠습니다.



이런 상황에서는, rebase 를 이용하여 커밋 X 의 내용을 적용한 상태로 새로운 기능을 추가하기 위해 아래 그림과 같이 O' 버전으로 만들어 내는 방법을 이용하면 됩니다.

브랜치를 능숙하게 잘~ 사용하면 상황에 맞게 여러 작업을 동시에 진행할 수 있다구!

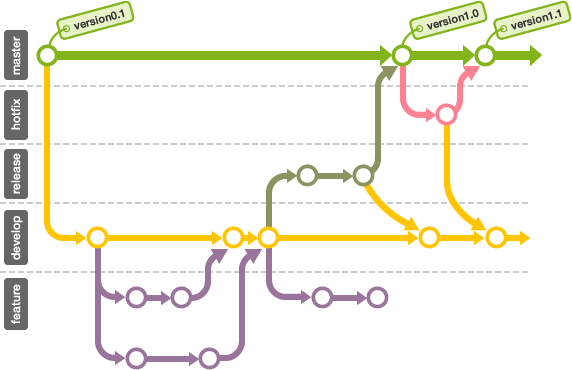
**컬럼 「A successful Git branching model」- 성공적인 Git 브랜칭 모델**

Git의 브랜치 운용 모델로서, "A successful Git branching model" 이란 컬럼을 소개합니다.

원문:  
<http://nvie.com/posts/a-successful-git-branching-model/>

이 운용 모델에서는 크게 나눠 4가지 종류의 브랜치를 이용하여 개발을 진행합니다.

* **메인 브랜치(Main branch)**
* **피처 브랜치(Feature branch) 또는 토픽 브랜치(Topic branch)**
* **릴리스 브랜치(Release branch)**
* **핫픽스 브랜치(Hotfix branch)**



**메인 브랜치(Main branch)**

'master' 브랜치와 'develop' 브랜치, 이 두 종류의 브랜치를 보통 메인 브랜치로 사용합니다.

* **master**  
  'master' 브랜치에서는, 배포 가능한 상태만을 관리합니다. 커밋할 때에는 태그를 사용하여 배포 번호를 기록합니다.
* **develop**  
  'develop' 브랜치는 앞서 설명한 통합 브랜치의 역할을 하며, 평소에는 이 브랜치를 기반으로 개발을 진행합니다.

**피처 브랜치(Feature branch)**

피처 브랜치는, 앞서 설명한 토픽 브랜치 역할을 담당합니다.

이 브랜치는 새로운 기능 개발 및 버그 수정이 필요할 때에 'develop' 브랜치로부터 분기합니다. 피처 브랜치에서의 작업은 기본적으로 공유할 필요가 없기 때문에, 원격으로는 관리하지 않습니다. 개발이 완료되면 'develop' 브랜치로 병합하여 다른 사람들과 공유합니다.

**릴리즈 브랜치(Release branch)**

릴리즈 브랜치에서는 버그를 수정하거나 새로운 기능을 포함한 상태로 모든 기능이 정상적으로 동작하는지 확인합니다. 릴리즈 브랜치의 이름은 관례적으로 브랜치 이름 앞에 'release-' 를 붙입니다. 이 때, 다음 번 릴리즈를 위한 개발 작업은 'develop' 브랜치 에서 계속 진행해 나가면 됩니다.

릴리즈 브랜치에서는 릴리즈를 위한 최종적인 버그 수정 등의 개발을 수행합니다. 모든 준비를 마치고 배포 가능한 상태가 되면 'master' 브랜치로 병합시키고, 병합한 커밋에 릴리즈 번호 태그를 추가합니다.

릴리즈 브랜치에서 기능을 점검하며 발견한 버그 수정 사항은 'develop' 브랜치에도 적용해 주어야 합니다. 그러므로 배포 완료 후 'develop' 브랜치에 대해서도 병합 작업을 수행합니다.

**핫픽스 브랜치(Hotfix branch)**

배포한 버전에 긴급하게 수정을 해야 할 필요가 있을 경우, 'master' 브랜치에서 분기하는 브랜치입니다. 관례적으로 브랜치 이름 앞에 'hotfix-'를 붙입니다.

예를 들어 'develop' 브랜치에서 개발을 한창 진행하고 있는 도중에 이전에 배포한 소스코드에 아주 큰 버그가 발견되는 경우를 생각해 보세요. 문제가 되는 부분을 빠르게 수정해서 안정적으로 다시 배포해야 하는 상황입니다. 'develop' 브랜치에서 문제가 되는 부분을 수정하여 배포 가능한 버전을 만들기에는 시간도 많이 소요되고 안정성을 보장하기도 어렵습니다. 그렇기 때문에 바로 배포가 가능한 'master' 브랜치에서 직접 브랜치를 만들어 필요한 부분 만을 수정한 후 다시 'master'브랜치에 병합하여 이를 배포하려고 하는 것이죠.

이 때 만든 핫픽스 브랜치에서의 변경 사항은 'develop' 브랜치에도 병합하여 문제가 되는 부분을 처리해 주어야 합니다.