Github 를 이용하는 전체 흐름 이해하기

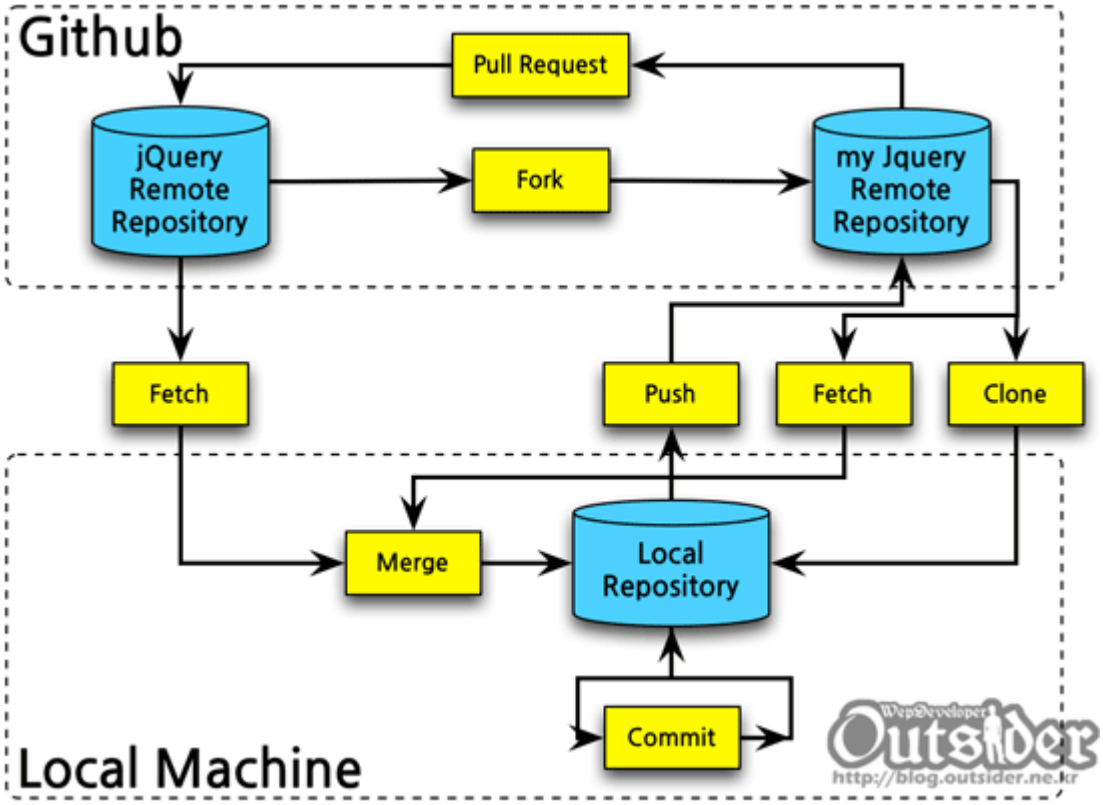
출처: Outsider's Dev Story
http://blog.outsider.ne.kr/865
http://blog.outsider.ne.kr/865

좀 어렵기는 하지만 Git 은 정말 좋은 도구이다. Git 을 쓰기 시작하면서 SVN 이 얼마나 불편하고 구린지를 이해할 수 있다. 어쨌든 Github 의 엄청난 성장아래 이제는 대부분의 오프소스 프로젝트들이 Github 로 이전하면서 회사에서 Git 을 사용하지 않더라도 Git 을 사용하지 않으면 안되는 때가 왔다. 하지만 Git 은 상당히 어려운 도구이고(git 홈페이지에 나온 easy to learn 은 홍보성 문구로 거짓말이다. 어렵다!!) 그룹스터디를 할 때 Github 을 사용하는 경우가 꽤 많았는데 사람들이 숙제를 해도 제출을 못하는 사태가 자주 발생하면서 각각의 명령어에 대한 사용법에 대한 설명도 중요하지만 전체적인 사용방식을 좀 설명할 필요가 있다고 느껴졌다.

내가 생각하기에 사람들이 Git 을 이해하는데 어려워하는 부분이 몇가지 있는데

- Github 를 설명할 때 Git 의 기능과 Github 의 기능을 명확하게 구분해 주지 않는다.
- 대부분의 개발자들이 가진 Subversion 에 대한 이해가 오히려 Git 을 이해하는데 방해가 된다. Git 은 DVCS 이고 SVN 은 VCS 이다. 앞에 Distribute 가 붙은건 마케팅적으로 괜히 붙힌게 아니라 분산 저장소이기 때문이다. Git 에서 이 분산저장소라는 것은 무척 중요하다.

그래서 Github 을 예시로 fork 해서 소스를 수정하고 적용까지 하는 전체적인 흐름 과정을 이해하면 각각의 명령어를 이해하는데 좀더 도움이 되지 않을까 한다. 예제의 설명을 위해서 jQuery 에서 어떤 버그를 발견해서 소스를 수정한뒤에 다시 jQuery 에 적용요청을 하는 일련의 과정을 예시로 들어보자. git 의 각 명령어를 일일이 다 설명하는 것은 내용이 너무 많기 때문에(모르는 명령어도 많고.. ——;;) 흐름을 이해할 수 있는 수준에서만 명령어를 설명할 것이다. 필요하다면 각 명령어는 따로 찾아보고 익혀야 한다.

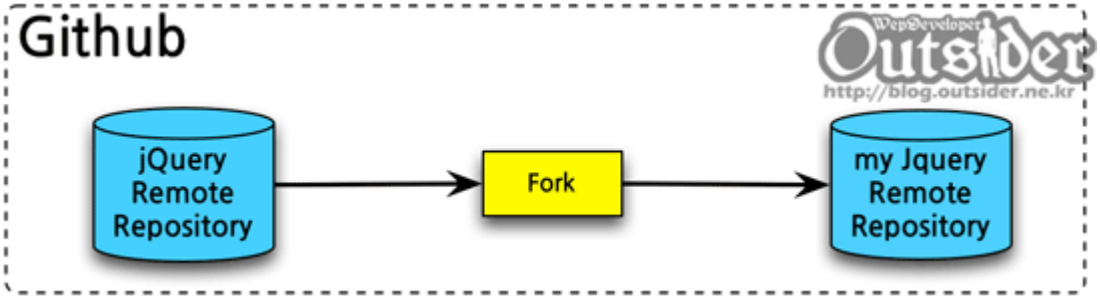


이 과정은 도식화 하면 위처럼 된다. 좀 복잡해 보일 수 있는데 하나씩 살펴보자.

Fork

Github 의 저장소를 보면 다음 화면처럼 3 가지의 저장소 주소를 제공하고 있다.(Git 에서 여러가지 프로토콜을 사용해서 저장소를 열어줄 수 있지만 Github 가 제공하는 저장소가 이렇다.)

jQuery 저장소에는 쓰기 권한을 가지고 있지 않으므로(아무나 쓰기를 할 수 있다면 저장소는 당연히 엉망일 될 것이다.) 우측에 보듯이 Read-Only 권한만 있음을 알 수 있다. 만약에 지금 시나리오처럼 소스를 수정해서 적용할 것이 아니고 그냥 혼자 소스를 살펴보기만 할 것이라면 이 주소를 사용해서 git clone 를 하면 로컬에 저장소를 내려받을 수 있다. 하지만 우리는 소스를 수정할 수 있는 저장소를 관리해야 하므로 Fork 를 해야 한다. 저장소 관리자와 친분이 있다면 요청을 해서 해당 저장소에 Push 할 수 있는 권한을 받을 수도 있겠지만 일반적으로는 Fork 한 뒤에 나중에 Pull Request 하는 방식으로 진행이 된다.



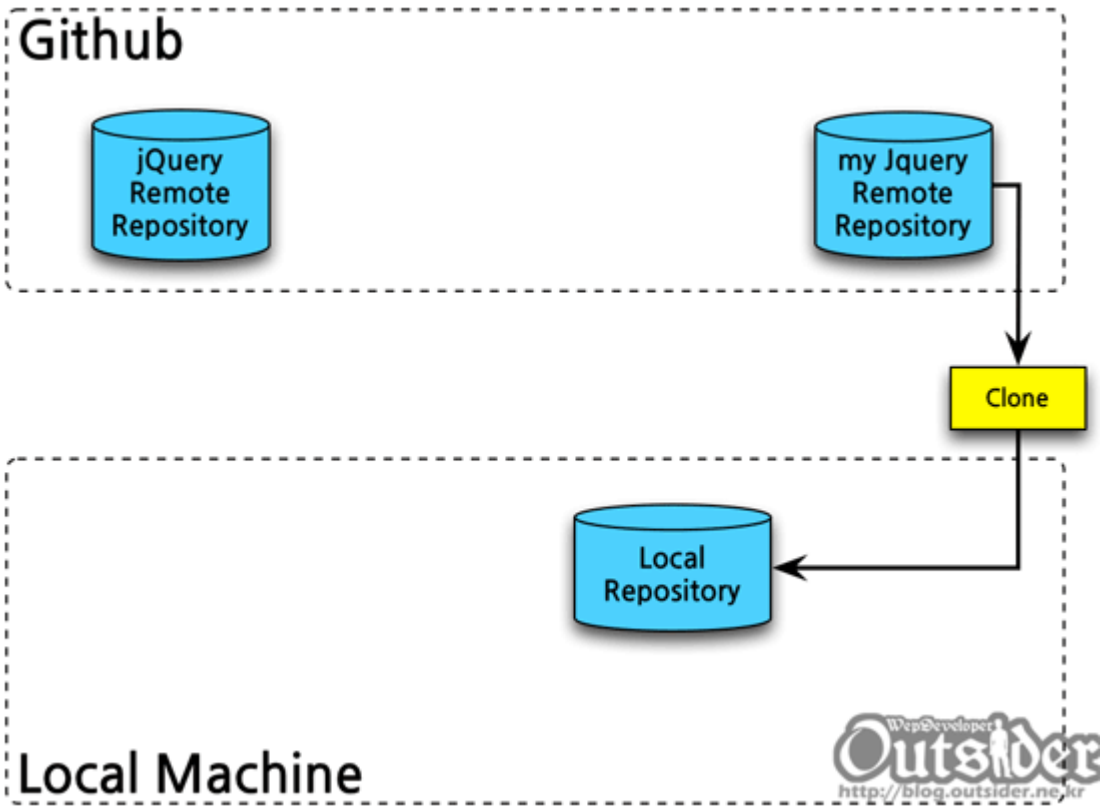
앞에서 본 다이어그램에서 위의 부분이다. jQuery 의 저장소를 Fork 받아서 내 저장소로 복사본을 만든다. 저장소의 우측상단을 보면 다음처런 Fork 버튼을 볼 수 있다.

이 Fork 버튼을 누르면 jQuery 저장소의 지금 상태 그대로를 복사해서 자신의 Github 계정에 jQuery 저장소를 생성한다. 이 Fork 는 git 의 기능이 아니라 Github 가 git 의 기능을 추상화해서 제공하는 기능이므로 git fork 같은 명령어는 없다. 간단히 생각하면 git clone 을 github 내에서 진행했다고 생각하면 된다.(내부 구현은 알 수 없지만....)

이제 자신의 계정에 jQuery 저장소가 생겼고 저장소 주소에 Read+Write 권한이 있는 것을 볼 수 있다. 이제 이 저장소에는 소스를 수정해서 푸시를 할 수 있다. 이 저장소는 jQuery 의 원래 저장소와 완전히 동일한(주소만 다른) 저장소이다. 분산저장소이기 때문에 저장소가 또 하나 생긴 것이고 다른 사람이 jQuery 원래 저장소 대신 내 저장소를 똑같이 Fork 받아서 수정하는 것도 당연히 가능하다 각 저장소는 모두 권한외에는 모두 동일한 기능을 가진다.

Clone

원격에서는 소스를 수정할 수 없으므로 이 저장소를 작업할 로컬 머신에 내려받아야 하는데 이 과정이 Clone 이다.



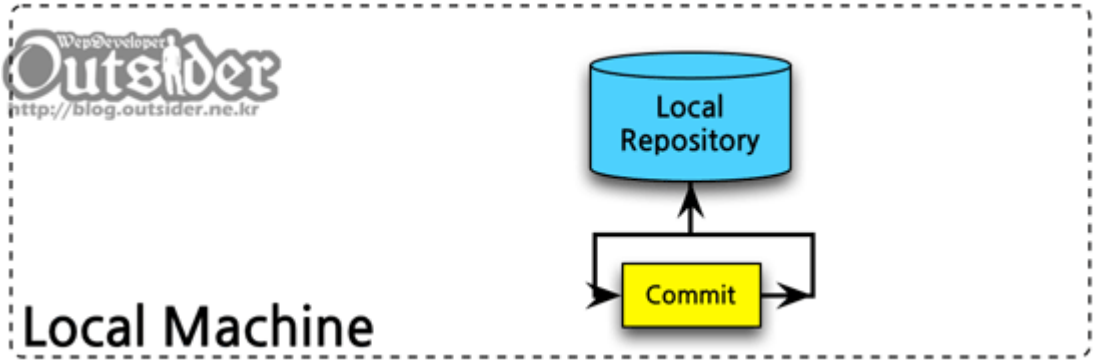
Clone 는 git 을 잘 몰라도 대부분 알고 있는 git clone git@github.com:outsideris/jquery.git 명령어로 수행한다. SSH 주소를 사용한 것은 git 프로토콜이 HTTPS 보다 훨씬 빠르고 Github 에 SSH 키를 등록해 놓으면 푸시할때 암호를 입력하지 않아도 되기 때문이다.

```
outsider@MacBookProRetina:~/projects/etc
⚡ git clone git@github.com:outsideris/jquery.git
Cloning into 'jquery'...
remote: Counting objects: 25147, done.
remote: Compressing objects: 100% (6570/6570), done.
remote: Total 25147 (delta 18388), reused 24620 (delta 17992)
Receiving objects: 100% (25147/25147), 13.77 MiB | 207 KiB/s, done.
Resolving deltas: 100% (18388/18388), done.
outsider@MacBookProRetina:~/projects/etc
⚡ cd jquery/
outsider@MacBookProRetina:~/projects/etc/jquery (git:4fed8eb@master)
⚡
```

Clone 을 받아오면 로컬에 jquery 폴더가 생기고 jquery 폴더안에 git 저장소를 내려받은 것을 볼 수 있다. 저장소 이름이 jquery 이기 때문에 jquery 폴더가 생겼고 다름 폴더명을 사용하려면 git clone git@github.com:outsideris/jquery.git NEW_NAME 과 같이 이름을 지정하면 된다. Fork 가 Github 내에서 저장소를 복사한 것이라면 clone 은 원격 저장소를 로컬로 복사한 것이다.

Commit

이제 저장소를 로컬에 가져왔으므로 소스를 수정하는 작업을 해야한다. git 에서는 관례적으로 메인 브랜치로 master 브랜치를 사용하지만 필요하다면 다른 브랜치를 메인으로 사용해도 아무런 문제가 없다. master 는 그냥 관례적으로 약속한 브랜치일 뿐이다.

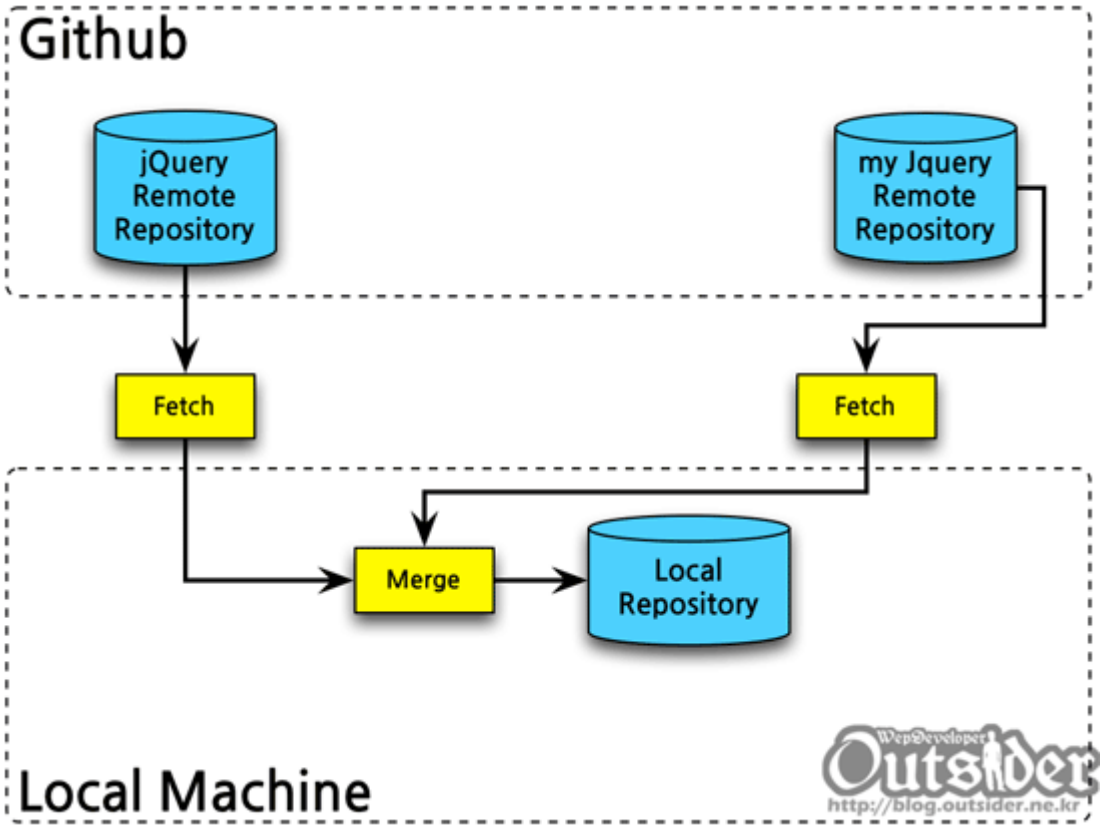


jQuery 의 소스를 수정하기 위해서 master 브랜치를 수정해야 하는데 Subversion 과는 다르게 Git 에서는 브랜치를 생성하는 것을 권장하고 있고 일반적으로 master 브랜치에서 다른 브랜치를 생성해서 작업하는 것이 일반적이다. 여기서는 patched 라는 브랜치를 생성한다.

```
outsider@MacBookProRetina:~/projects/etc/jquery (git:4fed8eb@master)
⚡ git branch hotfixes/patched
outsider@MacBookProRetina:~/projects/etc/jquery (git:4fed8eb@master)
⚡ git checkout hotfixes/patched
Switched to branch 'hotfixes/patched'
outsider@MacBookProRetina:~/projects/etc/jquery (git:4fed8eb@hotfixes/patched)
⚡
```

git branch 명령어를 사용해서 현 브랜치에서 새로운 브랜치를 생성하고 해당 브랜치로 사용하는 브랜치를 바꾸었다. 여기서 hotfixes/patched 라고 브랜치를 생성한 이유는 git-flow 의 관례이고 그냥 브랜치명이 hotfixes/patched 라고 지은것이고 계층적으로 구분하기 위해서 생성했다고 보면 된다. 위의 두 과정대신 git checkout -b hotfixes/patched 를 사용하면 명령어 하나로 브랜치를 생성해서 바로 이동까지 할 수 있다. 사실 git 을 사용하는 대부분의 과정은 이 단계에서 이루어지는데 기본적으로 소스를 수정하고 커밋할 소스를 git add 한 후에 git commit 을 한다. commit 은 현재의 git 저장소에 소스를 적용해서 히스토리를 남긴 것으로 SVN 의 commit 과 다르게 원격저장소에 적용되는 것은 아니다.

Fetch & Merge



로컬에서 작업을 하다보면 원격저장소에 변경사항이 생긴다. 클론받은 이후에 원격저장소에 누군가 소스를 푸시하면 이 변경사항을 다시 로컬로 가져와야 하는데 이 과정을 fetch 로 원격저장소의 변경사항을 로컬로 가져온 뒤에 로컬의 브랜치에 merge 하는 과정으로 이루어진다. 이 과정은 일반적으로는 git pull 이라는 명령어를 통해서 한방으로 이루어지는데 git pull 보다는 git fetch 후에 git merge 로 나누어서 작업하는 것을 보통 더 권장한다.(권장하는 이유는 여러가지가 있지만 conflict 가 생겼을 때 대처가 훨씬 쉽다던지 merge 를 훨씬 자유롭게 할 수 있다.)

?

```
1    $ git remote -v
2    origin    git@github.com:outsideris/jquery.git (fetch)
3    origin    git@github.com:outsideris/jquery.git (push)
```

위 와 같이 로컬 저장소에 등록된 원격저장소 목록을 보면 clone 을 받아온 대상인 원격 저장소가 origin 이라는 이름으로 등록되어 있다. github 에서 저장소를 새로 생성하면(fork 로 생성하는 경우외에) git remote add origin git@github.com:outsideris/jquery.git 와 같이 저장소를 추가하도록 안내를 하는데 이는 관례에 따라 메인 원격저장소를 origin 으로 등록한 것이다. 그리고 이어서 소스를 git push -u origin master 로 푸시하도록 안내하고 있는데 여기서 -u 옵션은 설정파일에 현재의 master 브랜치를 origin 의(여기서는 fork 받은 자신의 원격저장소) master 브랜치로 연결해 주어 다음부터는 자동으로 master 브랜치에서 git push 를 하면 origin 의 master 브랜치로 푸시가 되고 git pull 을 하면 origin 의 master 를 fetch 해서 로컬의 master 로 merge 하도록 설정하는 것이다. 그래서 .git/config 파일의 내용을 보면 다음과 같이 master 브랜치의 원격저장소와 머지할 브랜치가 설정되어 있는 것을 볼 수 있다.(clone 을 받아오면 이 설정이 자동으로 추가된다.)

?

```
1     [branch "master"]
2         remote = origin
3         merge = refs/heads/master
```

그래서 master 가 아닌 다른 브랜치를 원격저장소에 푸시하려면 대상 저장소와 브랜치를 직접 지정해 주어야한다.(매번하기 싫으면 위처럼 설정파일에 지정하면 된다.) 여기서는 fetch 와 merge 에 대해서 설명하는 단계이므로 이 부분에 대해서만 얘기하면 git pull 하면 자동으로 master 에 머지되도록 설정이 되어 있는 것이고 이를 다시 말하면 반드시 master 에만 머지해야 하는 것은 당연히 아니다.

지금까지 origin 만 살펴보았는데 이 시나리오에서는 우리에게 필요한 원격저장소가 하나 더 있다. 즉, fork 받은 자신의 저장소 외에 원본인 jQuery 의 원격저장소가 있다. 이 원격저장소에도 누군가 계속 소스를 수정할 것이므로 jQuery 를 기반으로 계속 작업을 할 것이라면 원본 jQuery 저장소의 변경사항도 로컬로 가져와야 한다. 처음 git 을 배울 때(정확히는 hg 로 DVCS 를 배웠지만...) 가장 궁금했던 것이 fork 한 저장소는 fork 한 시점으로 멈춰있으므로 변경사항을 가져오려면 어떻게 하는가?였다. 그래서 매번 새로운 내용이 필요할 때마다 github 의 저장소를 삭제하고 다시 fork 로 새 저장소를 만들어서 사용했다.(——;;) 이 당시에는 git 을 제대로 이해못했으므로(지금도 ㅠㅠ) 동기화기능을 github 에서 제공해야 하는 것이 아닌가 하고 생각했지만 이러한 부분은 git 이 자유롭게 다룰 수 있으므로 github 에서 제공할 이유가 없는 기능이다.

이 문제를 해결하려면 그냥 jQuery 의 원본저장소도 원격저장소로 추가해주면 된다.

```
?
1     $ git remote add jquery git@github.com:jquery/jquery.git
2     $ git remote
3     jquery
4     origin
```

앞에서 origin 이라는 이름으로 원격저장소를 추가한것처럼 이번에는 jquery 라는 이름으로(원하는 대로 지정하면 된다.) jquery 의 원본 원격저장소의 주소를 원격저장소로 등록하고 git remote 명령어로 확인해 보면 원격저장소가 2 개 등록되어 있는 것을 볼 수 있다. jquery 원본 저장소에는 push 할 권한은 없지만 읽기권한은 있으므로 변경사항을 가져오는데는 아무런 문제가 없다. 이러한 점이 git 이 DVCS 이기 때문에 갖는 강점인데 원격저장소는 필요한대로 추가할 수 있고 어디서나 가져와서 머지할 수 있다.

```
?
1     $ git fetch jquery
2     From github.com:jquery/jquery
3      * [new branch]      1.8-stable -> jquery/1.8-stable
4      * [new branch]      master     -> jquery/master
5      * [new branch]      promisea   -> jquery/promisea
6     $ git merge jquery/master
7     Already up-to-date.
```

이제 jquery 저장소(이름을 그렇게 주었으므로)의 내용을 fetch 받아오면 위처럼 새로운 브랜치들이 생기고 이 내용은 jquery/master 라는 이름으로 로컬에서 접근할 수 있다.(다른 브랜치도 몇 개 가져온 것을 볼 수 있다.) git 은 SVN 과는 다르게 히스토리를 검색하기 위해서 원격 저장소에 접근하지 않아도 되고 로컬에 전체 히스토리를 모두 내려온다. 그래서 실제로 origin 과 jquery 아래 내려받은 저장소의 브랜치가 존재하고 있다.

?

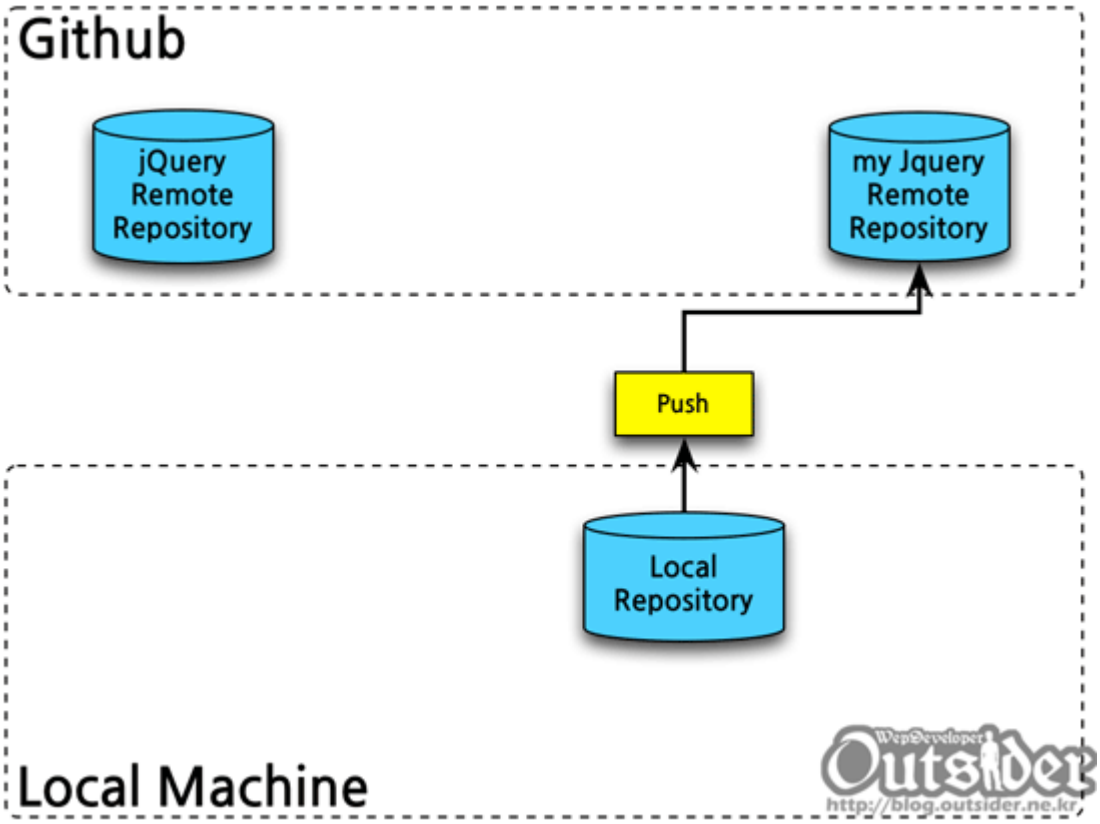
```
$ git branch -r
  jquery/1.8-stable
  jquery/master
  jquery/promisea
  origin/1.8-stable
  origin/HEAD -> origin/master
  origin/master
  origin/promisea
```

git branch -r 은 원격저장소의 브랜치를 보는 명령어인데 위처럼 현재의 원격 브랜치를 모두 볼 수 있다. 이 원격 브랜치는 원격에서 받아온 브랜치를 의미하지만 실제로 그 내용은 모두 로컬에 내려받은 것이다. 하지만 이 브랜치에는 커밋을 하는 등의 수정은 할 수 없고 오로지 읽기 전용이다. 그래서 이 원격브랜치들에서 새로운 브랜치를 생성하거나 기존의 브랜치에 머지를 하는 것이 가능하다. 앞에서 본 것처럼 이 원격 저장소의 변경사항을 업데이트하려면 git fetch jquery 처럼 원격저장소별로 업데이트를 하고 브랜치별로 업데이트를 하진 않는다. 앞에서 fetch 로 업데이트를 한 후에 현재 브랜치(master)에 머지를 했는데 꼭 master 에만 머지해야 하는 것은 아니다. 즉, 필요에 따라서는 작업을 master 에다 하고 원격 저장소의 내용을 추적하는 다른 브랜치를 만들어서 머지할 수도 있다.

앞에서 commit 과정을 설명할 때 일반적으로 작업을 master 에서 하지 않고 별도의 브랜치를 만들어서 한다고 설명했는데 여기서 설명하고 있는 시나리오처럼 jQuery 소스를 수정하는 경우 특별한 이유가 없다면 자신이 작업하는 master 브랜치를 jquery/master 브랜치와 동일하게 유지하는 것이 좋다. 그래서 merge 할 때 fast-forward 되도록 master 를 유지하는 것이 히스토리를 좀 더 이쁘게 할 수 있는 방법이다.(fast-forward 가 되지 않으면 merge 했다는 새로운 커밋이 하나 생기게 된다.)

Push

이제 로컬에서 수정작업을 어느정도 완료했다면 자신의 원격저장소에 수정한 커밋들을 푸시해야 한다.



이 과정은 로컬저장소에 커밋한 내역을 원격저장소에 적용하는 것으로 git push origin master 처럼 푸시할 원격저장소명과 브랜치를 지정해야 한다.(앞에서 처럼 -u 옵션으로 설정했다면 master 에선 git push 만 해도 된다.)지정해야 한다는 것은 권한만 있다면 어떤 원격저장소에도 또는 다른 브랜치에도 푸시할 수 있다는 것을 의미한다. 대부분의 git 저장소는 fast-forward 할 수 있는 경우에만 푸시할 수 있도록 하고 있고 이는 Github 도 마찬가지이다.

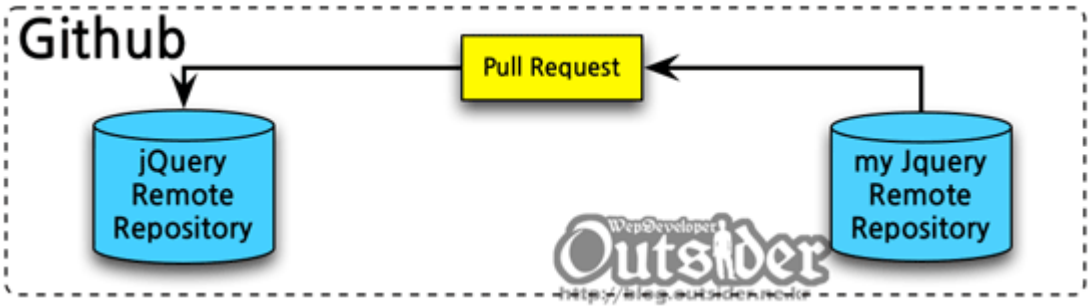
앞에서 봤듯이 jQuery 의 원본저장소와 자신의 원격저장소의 master 브랜치를 계속 싱크하려면 jquery 원본저장소를 fetch 받아서 master 에 머지한 뒤에 자신의 원격저장소의 master 로 푸시하면 된다. 이 과정을 계속 반복하면 항상 두 master 브랜치의 동기화를 할 수 있다. commit 과정에서 수정사항을 보통 별도의 브랜치를 만들어서 작업한다고 했는데 git 에서 원격저장소에 branch 와 tag 를 push 하기에서 설명했듯이 git push 원격저장소명 로컬브랜치명:원격브랜치명으로 원격저장소에 새로운 브랜치를 생성하면서 푸시해야 한다.

?

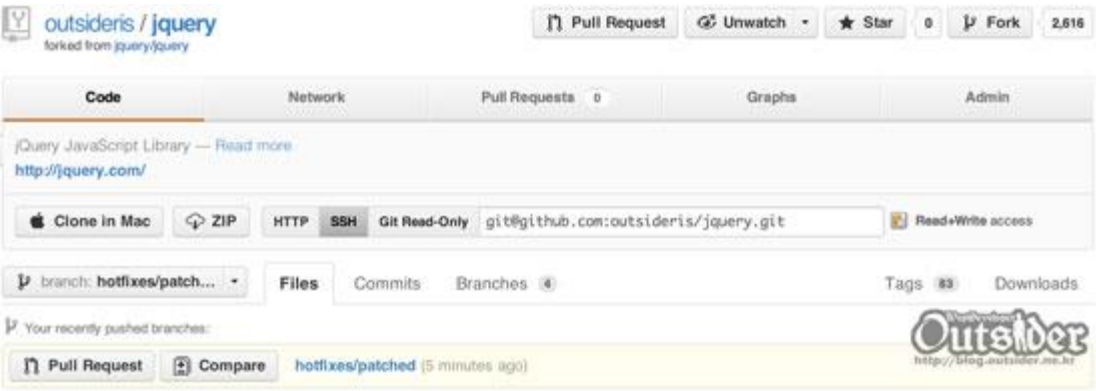
```
1    $ git push origin hotfixes/patched
2    Counting objects: 5, done.
3    Delta compression using up to 8 threads.
4    Compressing objects: 100% (3/3), done.
5    Writing objects: 100% (3/3), 309 bytes, done.
6    Total 3 (delta 2), reused 0 (delta 0)
7    To git@github.com:outsideris/jquery.git
8     * [new branch]      hotfixes/patched -> hotfixes/patched
```

이처럼 푸시를 하면 원격에 새로운 브랜치가 생성된 것을 볼 수 있다.

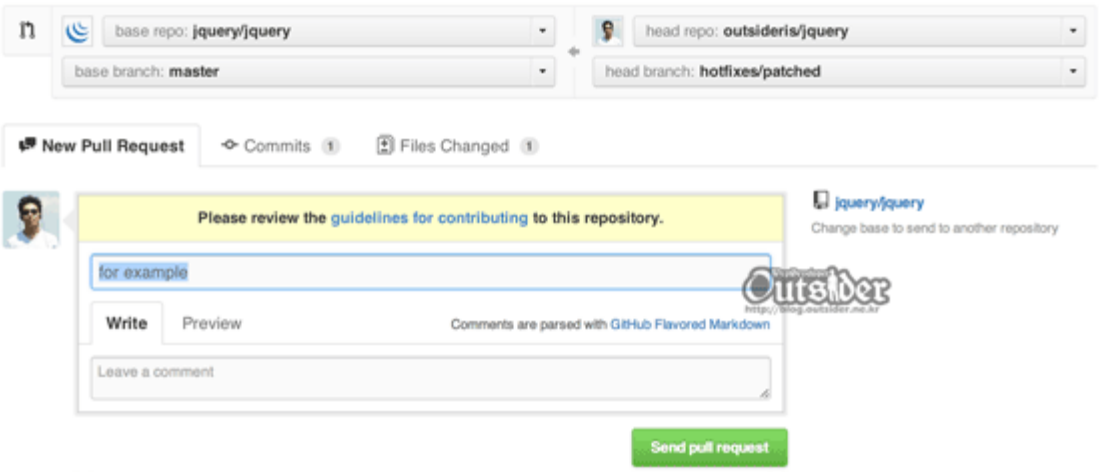
Pull Request



이제 원격에 올린 변경사항을 원본 jQuery 저장소에 적용하기 위해서 Pull Request 를 보내야 한다. Pull Request 도 git 자체에 있는 기능은 아니고 Github 에서 제공하는 기능이라고 할 수 있다.(써본 적은 없지만 Github 를 쓰지 않는다면 패치내용을 메일로 보내면 적용하는 방식을 취했었다.) Pull Request 를 사용하는 이유는 jQuery 원본저장소에는 쓰기권한이 없기 때문에 수정한 내용을 Pull Request 로 보내면 jQuery 저장소의 커미터들이 내용을 확인한뒤에 승인을 하면 Pull Request 의 내용이 jQuery 원본저장소에 merge 가 된다.(물론 승인하지 않고 거절할 수도 있다.)



이제 Github 의 자신의 저장소에서 변경사항을 적용한 브랜치페이지로 이동해서 상단의 Pull Request 버튼을 누르면 된다. 최근에 변경한 내용은 중간에도 Pull Request 버튼이 나온다. 이 버튼을 클릭하면 다음과 같이 Pull Request 를 보내는 화면을 볼 수 있다.



상단에 Pull Request 를 보내는 자신의 브랜치와 대상이 되는 브랜치를 지정할 수 있고 그 아래 Pull Request 에 대한 내용을 볼 수 있다. jQuery 의 경우 CONTRIBUTING.md 파일이 존재하기 때문에 위에 해당 부분에 대한 안내가 나온다. 오픈소스 프로젝트마다 Pull Request 를 받아주는 약속이 다르므로 보내기 전에 이를 먼저 확인해야 한다. 소스 수정이 잘 되었더라도 이 약속을 제대로 지키지 않으면 받아주지 않는다. 탭에서 Commits 나 Files Changed 를 클릭하면 Pull Request 를 보내는 커밋과 변경사항이 제대로 되었는지 확인할 수 있다.

물론 다른 사람의 소스를 Fork 해서 나중에 Pull Request 까지 보내는 경우가 많진 않지만(이정도까지 하는 사람은 대부분 이미 Git 에 익숙해져있다고 생각한다.) 직접 저장소를 생성해서 혼자 사용한다고 하더라도 이 전체 흐름의 일부로 포함되기 때문에 이 흐름에서 해당 부분만 참고해서 사용하면 된다.