0. 세가지 상태 Git은 파일을 Committed, Modified, Staged 이렇게 세 가지 상태로 관리한다. Committed란 데이터가 로컬 데이터베이스에 안전하게 저장됐다는 것을 의미한다. Modified는 수정한 파일을 아직 로컬 데이터베이스에 커밋하지 않은 것을 말한다. Staged란 현재 수정한 파일을 곧 커밋할 것이라고 표시한 상태를 의미한다.

1. git 설치

yum install git-all

2.git config 설정

root@ubuntu:/var/www/html# git config --global user.name "choga88"

root@ubuntu:/var/www/html# git config --global user.email [choga88@sk.com](mailto:choga88@sk.com)

root@ubuntu:/var/www/html# git config --list

user.name=choga88

user.email=choga88@sk.com

core.repositoryformatversion=0

core.filemode=true

core.bare=false

core.logallrefupdates=true

3. 명령어 도움말 사용법

$ git help <verb>

$ git <verb> --help

$ man git-<verb>

config 도움말 사용법

$ git help config

4. git 저장소 만들기

cd /home/user/your\_repository

$ git init

$ git add –A

$ git commit -m 'initial project version'

5. 기존 저장소를 git clone 하기

$ git clone <https://github.com/libgit2/libgit2>

‘`libgit2’이라는 디렉토리를 만들고 그 안에 *.git* 디렉토리를 만든다. 그리고 저장소의 데이터를 모두 가져와서 자동으로 가장 최신 버전을 Checkout 해 놓는다

$ git clone https://github.com/libgit2/libgit2 mylibgit

디렉토리 이름이 mylibgit 이라는 것만 빼면 이 명령의 결과와 앞선 명령의 결과는 같다

5. git 라이프싸이클

워킹 디렉토리의 모든 파일은 크게 Tracked(관리대상임)와 Untracked(관리대상이 아님)로 나눈다. Tracked 파일은 이미 스냅샷에 포함돼 있던 파일이다. Tracked 파일은 또 Unmodified(수정하지 않음)와 Modified(수정함) 그리고 Staged(커밋으로 저장소에 기록할) 상태 중 하나이다. 그리고 나머지 파일은 모두 Untracked 파일이다. Untracked 파일은 워킹 디렉토리에 있는 파일 중 스냅샷에도 Staging Area에도 포함되지 않은 파일이다. 처음 저장소를 Clone 하면 모든 파일은 Tracked이면서 Unmodified 상태이다. 파일을 Checkout 하고 나서 아무것도 수정하지 않았기 때문에 그렇다.

마지막 커밋 이후 아직 아무것도 수정하지 않은 상태에서 어떤 파일을 수정하면 Git은 그 파일을 **Modified** 상태로 인식한다. 실제로 커밋을 하기 위해서는 이 수정한 파일을 Staged 상태로 만들고, Staged 상태의 파일을 커밋한다. 이런 라이프사이클을 계속 반복한다.

**Unstage**



6. git 상태확인

root@ubuntu:/var/www/html# git status

On branch master

nothing to commit, working directory clean

위의 내용은 파일을 하나도 수정하지 않았다는 것을 말해준다. Tracked 파일은 하나도 수정되지 않았다는 의미다. Untracked 파일은 아직 없어서 목록에 나타나지 않는다

root@ubuntu:/var/www/html# echo 'codeignigher' > README

root@ubuntu:/var/www/html# git status

On branch master

Untracked files:

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

README

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

root@ubuntu:/var/www/html# git add README

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

Changes to be committed” 에 들어 있는 파일은 Staged 상태라는 것을 의미한다. 커밋하면 git add`를 실행한 시점의 파일이 커밋되어 저장소 히스토리에 남는다. 앞에서 `git init 명령을 실행한 후, git add (files) 명령을 실행했던 걸 기억할 것이다. 이 명령을 통해 디렉토리에 있는 파일을 추적하고 관리하도록 한다. git add 명령은 파일 또는 디렉토리의 경로를 아규먼트로 받는다. 디렉토리면 아래에 있는 모든 파일들까지 재귀적으로 추가한다

7. 수정된 파일 stage하기

root@ubuntu:/var/www/html# vi application/controllers/topic.php

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

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: application/controllers/topic.php

이 topic.php파일은 ‘`Changes not staged for commit’'에 있다. 이것은 수정한 파일이 Tracked 상태이지만 아직 Staged 상태는 아니라는 것이다. Staged 상태로 만들려면 git add 명령을 실행해야 한다. git add 명령은 파일을 새로 추적할 때도 사용하고 수정한 파일을 Staged 상태로 만들 때도 사용한다.

root@ubuntu:/var/www/html# git add application/controllers/topic.php

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: application/controllers/topic.php

root@ubuntu:/var/www/html# vi application/controllers/topic.php

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: application/controllers/topic.php

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: application/controllers/topic.php

topic.php 가 Staged 상태이면서 \_동시에\_ Unstaged 상태로 나온다. 어떻게 이런 일이 가능할까? `git add 명령을 실행하면 Git은 파일을 바로 Staged 상태로 만든다. 지금 이 시점에서 커밋을 하면 git commit 명령을 실행하는 시점의 버전이 커밋되는 것이 아니라 마지막으로 git add 명령을 실행했을 때의 버전이 커밋된다. 그러니까 git add 명령을 실행한 후에 또 파일을 수정하면 git add 명령을 다시 실행해서 최신 버전을 Staged 상태로 만들어야 한다

root@ubuntu:/var/www/html# git add application/controllers/topic.php

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: application/controllers/topic.php

8. 파일상태 간단하게 확인

root@ubuntu:/var/www/html# git status -s

A README

M application/controllers/topic.php

$ git status -s

M README

MM Rakefile

A lib/git.rb

M lib/simplegit.rb

?? LICENSE.txt

아직 추적하지 않는 새 파일 앞에는 ??`표시가 붙는다. Staged 상태로 추가한 파일 중 새로 생성한 파일 앞에는 `A 표시가, 수정한 파일 앞에는 M 표시가 붙는다. 위 명령의 결과에서 상태정보 컬럼에는 두 가지 정보를 보여준다. 왼쪽에는 Staging Area에서의 상태를, 오른쪽에는 Working Tree에서의 상태를 표시한다. README 파일 같은 경우 내용을 변경했지만 아직 Staged 상태로 추가하지는 않았다. lib/simplegit.rb 파일은 내용을 변경하고 Staged 상태로 추가까지 한 상태이다. 위 결과에서 차이점을 비교해보자. `Rakefile`은 변경하고 Staged 상태로 추가한 후 또 내용을 변경해서 Staged 이면서 Unstaged 상태인 파일이다.

9. git에서 무시하기

.gitignore 파일을 만들고 그 안에 무시할 파일 패턴을 적는다  
$ cat .gitignore

\*.[oa]

\*~

10. git에서 변경내용 보기

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

modified: application/controllers/topic.php

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: application/controllers/topic.php

root@ubuntu:/var/www/html# git diff

diff --git a/application/controllers/topic.php b/application/controllers/topic.php

index 43a903d..14a4406 100644

--- a/application/controllers/topic.php

+++ b/application/controllers/topic.php

@@ -5,6 +5,6 @@ class Topic extends CI\_Controller {

public function index()

{

- echo 'topic test12';

+ echo 'topic test123';

}

}

이 명령은 워킹 디렉토리에 있는 것과 Staging Area에 있는 것을 비교한다. 그래서 수정하고 아직 Stage 하지 않은 것을 보여준다.

수정한 파일을 모두 Staging Area에 넣었다면 `git diff 명령은 아무것도 출력하지 않는다.

root@ubuntu:/var/www/html# git diff --staged

diff --git a/README b/README

new file mode 100644

index 0000000..079e2b3

--- /dev/null

+++ b/README

@@ -0,0 +1 @@

+codeignigher

diff --git a/application/controllers/topic.php b/application/controllers/topic.php

index bc20397..43a903d 100644

--- a/application/controllers/topic.php

+++ b/application/controllers/topic.php

@@ -5,6 +5,6 @@ class Topic extends CI\_Controller {

public function index()

{

- echo 'topic test';

+ echo 'topic test12';

}

}

만약 커밋하려고 Staging Area에 넣은 파일의 변경 부분을 보고 싶으면 git diff --staged 옵션을 사용한다. 이 명령은 저장소에 커밋한 것과 Staging Area에 있는 것을 비교한다.

11. 변경사항 commit하기

root@ubuntu:/var/www/html# git commit -m '2'

[master c8ff6c3] 2

2 files changed, 2 insertions(+), 1 deletion(-)

create mode 100644 README

root@ubuntu:/var/www/html# 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: application/controllers/topic.php

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

수정한 것을 커밋하기 위해 Staging Area에 파일을 정리했다. Unstaged 상태의 파일은 커밋되지 않는다는 것을 기억해야 한다. Git은 생성하거나 수정하고 나서 git add 명령으로 추가하지 않은 파일은 커밋하지 않는다. 그 파일은 여전히 Modified 상태로 남아 있다.

commit 명령은 몇 가지 정보를 출력하는데 위 예제는 (master) 브랜치에 커밋했고 체크섬은 (c8ff6c3)이라고 알려준다. 그리고 수정한 파일이 몇 개이고 삭제됐거나 추가된 라인이 몇 라인인지 알려준다.

root@ubuntu:/var/www/html# git commit -a -m '3'

[master c1e6942] 3

1 file changed, 1 insertion(+), 1 deletion(-)

git commit 명령을 실행할 때 -a 옵션을 추가하면 Git은 Tracked 상태의 파일을 자동으로 Staging Area에 넣는다. 그래서 git add 명령을 실행하는 수고를 덜 수 있다.

12. 파일 삭제하기

root@ubuntu:/var/www/html# rm README

root@ubuntu:/var/www/html# git status

On branch master

Changes not staged for commit:

(use "git add/rm <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

deleted: README

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

root@ubuntu:/var/www/html# git rm README

rm 'README'

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

deleted: README

root@ubuntu:/var/www/html# git commit -m '4'

[master 0313271] 4

1 file changed, 1 deletion(-)

delete mode 100644 README

root@ubuntu:/var/www/html# git status

On branch master

nothing to commit, working directory clean

root@ubuntu:/var/www/html# git rm --cached README

--cached옵션은 Staging Area에서만 제거하고 워킹 디렉토리에 있는 파일은 지우지 않고 남겨둘 수 있다. 다시 말해서 하드디스크에 있는 파일은 그대로 두고 Git만 추적하지 않게 한다

13. 파일 이름 변경하기

root@ubuntu:/var/www/html# git mv README README.mv

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README.mv

git mv는 아래 명령어 3개 수행과 동일한 효과

$ mv README.md README

$ git rm README.md

$ git add README

14. commit 히스토리 조회하기

root@ubuntu:/var/www/html# git log

commit 031327182e5dfaedfbf39e6d59f51609e5ab4e23

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 19:32:51 2017 -0700

4

commit c1e6942a85137da025fcf1597c37faa41df64423

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 18:48:56 2017 -0700

3

commit c8ff6c30fb31668cad1bb8f21d16e3ddea9a73b3

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 07:44:04 2017 -0700

2

commit 8c14b5b05f0e3df86cc33c207b963a70d346ec5b

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 07:03:47 2017 -0700

1

commit d4d58fd7e66a8bfdf5c75b4d50d911b77ab6a474

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 07:02:43 2017 -0700

1

특별한 아규먼트 없이 git log 명령을 실행하면 저장소의 커밋 히스토리를 시간순으로 보여준다. 즉, 가장 최근의 커밋이 가장 먼저 나온다. 그리고 이어서 각 커밋의 SHA-1 체크섬, 저자 이름, 저자 이메일, 커밋한 날짜, 커밋 메시지를 보여준다

root@ubuntu:/var/www/html# git log -p -2

commit 031327182e5dfaedfbf39e6d59f51609e5ab4e23

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 19:32:51 2017 -0700

4

diff --git a/README b/README

deleted file mode 100644

index 079e2b3..0000000

--- a/README

+++ /dev/null

@@ -1 +0,0 @@

-codeignigher

commit c1e6942a85137da025fcf1597c37faa41df64423

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 18:48:56 2017 -0700

3

diff --git a/application/controllers/topic.php b/application/controllers/topic.php

index 43a903d..14a4406 100644

--- a/application/controllers/topic.php

+++ b/application/controllers/topic.php

@@ -5,6 +5,6 @@ class Topic extends CI\_Controller {

public function index()

{

- echo 'topic test12';

+ echo 'topic test123';

}

}

git log옵션 중 `-p`는 굉장히 유용한 옵션이다. `-p`는 각 커밋의 diff 결과를 보여준다. 다른 유용한 옵션으로 `-2`가 있는데 최근 두 개의 결과만 보여주는 옵션이다

root@ubuntu:/var/www/html# git log --stat

commit 031327182e5dfaedfbf39e6d59f51609e5ab4e23

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 19:32:51 2017 -0700

4

README | 1 -

1 file changed, 1 deletion(-)

commit c1e6942a85137da025fcf1597c37faa41df64423

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 18:48:56 2017 -0700

3

application/controllers/topic.php | 2 +-

1 file changed, 1 insertion(+), 1 deletion(-)

commit c8ff6c30fb31668cad1bb8f21d16e3ddea9a73b3

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 07:44:04 2017 -0700

2

README | 1 +

application/controllers/topic.php | 2 +-

2 files changed, 2 insertions(+), 1 deletion(-)

commit 8c14b5b05f0e3df86cc33c207b963a70d346ec5b

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 07:03:47 2017 -0700

1

.editorconfig | 15 +++++++++++++++

.gitignore | 29 +++++++++++++++++++++++++++++

CodeIgniter-3.1.5/.editorconfig | 15 ---------------

CodeIgniter-3.1.5/.gitignore | 29 -----------------------------

4 files changed, 44 insertions(+), 44 deletions(-)

commit d4d58fd7e66a8bfdf5c75b4d50d911b77ab6a474

Author: choga88 <choga88@sk.com>

Date: Sun Jul 9 07:02:43 2017 -0700

1

CodeIgniter-3.1.5/.editorconfig | 15 +

CodeIgniter-3.1.5/.gitignore | 29 +

application/.htaccess | 6 +

application/cache/.htaccess | 6 +

application/cache/index.html | 11 +

application/config/autoload.php | 135 +

application/config/config.php | 523 ++++

application/config/constants.php | 85 +

application/config/database.php | 96 +

application/config/doctypes.php | 24 +

application/config/foreign\_chars.php | 103 +

application/config/hooks.php | 13 +

--stat 옵션은 어떤 파일이 수정됐는지, 얼마나 많은 파일이 변경됐는지, 또 얼마나 많은 라인을 추가하거나 삭제했는지 보여준다. 요약정보는 가장 뒤쪽에 보여준다

root@ubuntu:/var/www/html# git log --pretty=oneline

031327182e5dfaedfbf39e6d59f51609e5ab4e23 4

c1e6942a85137da025fcf1597c37faa41df64423 3

c8ff6c30fb31668cad1bb8f21d16e3ddea9a73b3 2

8c14b5b05f0e3df86cc33c207b963a70d346ec5b 1

d4d58fd7e66a8bfdf5c75b4d50d911b77ab6a474 1

추가로 short, full, fuller 옵션도 있는데 이것은 정보를 조금씩 가감해서 보여준다.

root@ubuntu:/var/www/html# git log --pretty=format:"%h %s" --graph

\* 0313271 4

\* c1e6942 3

\* c8ff6c3 2

\* 8c14b5b 1

\* d4d58fd 1

oneline 옵션과 format 옵션은 --graph 옵션과 함께 사용할 때 더 빛난다. 이 명령은 브랜치와 머지 히스토리를 보여주는 아스키 그래프를 출력한다

| Table 2. git log 주요 옵션 | | | |
| --- | --- | --- | --- |
| **옵션** | **설명** | | |
| -p | 각 커밋에 적용된 패치를 보여준다. | | |
| --stat | 각 커밋에서 수정된 파일의 통계정보를 보여준다. | | |
| --shortstat | --stat 명령의 결과 중에서 수정한 파일, 추가된 라인, 삭제된 라인만 보여준다. | | |
| --name-only | 커밋 정보중에서 수정된 파일의 목록만 보여준다. | | |
| --name-status | 수정된 파일의 목록을 보여줄 뿐만 아니라 파일을 추가한 것인지, 수정한 것인지, 삭제한 것인지도 보여준다. | | |
| --abbrev-commit | 40자 짜리 SHA-1 체크섬을 전부 보여주는 것이 아니라 처음 몇 자만 보여준다. | | |
| --relative-date | 정확한 시간을 보여주는 것이 아니라 ‘`2 weeks ago’'처럼 상대적인 형식으로 보여준다. | | |
| --graph | 브랜치와 머지 히스토리 정보까지 아스키 그래프로 보여준다. | | |
| --pretty | 지정한 형식으로 보여준다. 이 옵션에는 oneline, short, full, fuller, format이 있다. format은 원하는 형식으로 출력하고자 할 때 사용한다. | | |
| Table 3. git log 조회 범위를 제한하는 옵션 | | | |
| **옵션** | | | **설명** |
| -(n) | | | 최근 n 개의 커밋만 조회한다. |
| --since, --after | | | 명시한 날짜 이후의 커밋만 검색한다. |
| --until, --before | | | 명시한 날짜 이전의 커밋만 조회한다. |
| --author | | | 입력한 저자의 커밋만 보여준다. |
| --committer | | | 입력한 커미터의 커밋만 보여준다. |
| --grep | | | 커밋 메시지 안의 텍스트를 검색한다. |
| -S | | | 커밋 변경(추가/삭제) 내용 안의 텍스트를 검색한다 |

15. 추가하여 commit하기

root@ubuntu:/var/www/html# git commit --amend

[master b17393c] 4

Date: Sun Jul 9 19:32:51 2017 -0700

2 files changed, 1 insertion(+), 1 deletion(-)

delete mode 100644 README

create mode 100644 README.mv

root@ubuntu:/var/www/html#

root@ubuntu:/var/www/html#

root@ubuntu:/var/www/html# git status

On branch master

nothing to commit, working directory clean

너무 일찍 커밋했거나 어떤 파일을 빼먹었을 때 그리고 커밋 메시지를 잘못 적었을 때 한다. 다시 커밋하고 싶으면 --amend 옵션을 사용한다

16. 파일을 unstage 상태로 변경하기

root@ubuntu:/var/www/html# git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: README

root@ubuntu:/var/www/html# git reset HEAD README

root@ubuntu:/var/www/html# 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: README

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

root@ubuntu:/var/www/html# git checkout -- README

root@ubuntu:/var/www/html# git status

On branch master

nothing to commit, working directory clean

root@ubuntu:/var/www/html# cat README

aaa <-- bbb를 추가하기전 commit 상태로 되돌아감

git checkout -- [file] 명령은 꽤 위험한 명령이라는 것을 알아야 한다. 원래 파일로 덮어썼기 때문에 수정한 내용은 전부 사라진다. 수정한 내용이 진짜 마음에 들지 않을 때만 사용하자.

17. 리모트 저장소 사용하기

root@ubuntu:~# git clone https://github.com/schacon/ticgit

Cloning into 'ticgit'...

remote: Counting objects: 1857, done.

remote: Total 1857 (delta 0), reused 0 (delta 0), pack-reused 1857

Receiving objects: 100% (1857/1857), 331.41 KiB | 346.00 KiB/s, done.

Resolving deltas: 100% (837/837), done.

Checking connectivity... done.

root@ubuntu:~# cd ticgit/

root@ubuntu:~/ticgit# git remote

origin

저장소를 Clone 하면 origin이라는 리모트 저장소가 자동으로 등록되기 때문에 origin이라는 이름을 볼 수 있다.

root@ubuntu:~/ticgit# git remote -v

origin https://github.com/schacon/ticgit (fetch)

origin https://github.com/schacon/ticgit (push)

v`옵션을 주어 단축이름과 URL을 함께 볼 수 있다

git remote add [단축이름] [url]로 origin이외의 리모트 저장소를 추가

[root@centos-master ticgit]# git remote add pb https://github.com/paulboone/ticgit

[root@centos-master ticgit]# git remote -v

origin https://github.com/schacon/ticgit (fetch)

origin https://github.com/schacon/ticgit (push)

pb https://github.com/paulboone/ticgit (fetch)

pb https://github.com/paulboone/ticgit (push)

리모트 저장소 pb에서 로컬에는 없지만 리모트 저장소에 있는 데이터를 로컬로 가져오지만 merge하지는 않는다.

[root@centos-master ticgit]# git fetch pb

remote: Counting objects: 43, done.

remote: Total 43 (delta 22), reused 22 (delta 22), pack-reused 21

Unpacking objects: 100% (43/43), done.

From https://github.com/paulboone/ticgit

\* [new branch] master -> pb/master

\* [new branch] ticgit -> pb/ticgit

일반적으로 git clone 명령은 자동으로 로컬의 master 브랜치가 리모트 저장소의 master 브랜치를 추적하도록 한다(물론 리모트 저장소에 master 브랜치가 있다는 가정에서). 그리고 git pull 명령은 Clone 한 서버에서 데이터를 가져오고 그 데이터를 자동으로 현재 작업하는 코드와 Merge 시킨다

프로젝트를 공유하고 싶을 때 Upstream 저장소에 Push 할 수 있다. 이 명령은 git push [리모트 저장소 이름] [브랜치 이름]` 사용하며 로컬의 master 브랜치를 origin서버에 push한다.

[root@centos-master ticgit]# git push origin master

Username for 'https://github.com': choga88@sk.com

Password for 'https://choga88@sk.com@github.com':

root@ubuntu:~/ticgit# git remote show origin

\* remote origin

Fetch URL: https://github.com/schacon/ticgit

Push URL: https://github.com/schacon/ticgit

HEAD branch: master

Remote branches:

master tracked

ticgit tracked

Local branch configured for 'git pull':

master merges with remote master

Local ref configured for 'git push':

master pushes to master (up to date)

git remote rename 명령으로 리모트 저장소의 이름을 변경할 수 있다

[root@centos-master ticgit]# git remote rename pb paul

[root@centos-master ticgit]# git remote

origin

paul

리모트 저장소를 삭제해야 한다면 git remote remove`나 `git remote rm 명령을 사용한다

[root@centos-master ticgit]# git remote remove paul

[root@centos-master ticgit]# git remote

origin