http://gitref.org/remotes/

docs book git remote list, add and delete remote repository aliases
Unlike centralized version control systems that have a client that is very different from a server, Git repositories are all basically equal and you simply synchronize between them. This makes it easy to have more than one remote repository - you can have some that you have read-only access to and others that you can write to as well.

So that you don't have to use the full URL of a remote repository every time you want to synchronize with it, Git stores an alias or nickname for each remote repository URL you are interested in. You use the git remote command to manage this list of remote repos that you care about.

git remote list your remote aliases
Without any arguments, Git will simply show you the remote repository aliases that it has stored. By default, if you cloned the project (as opposed to creating a new one locally), Git will automatically add the URL of the repository that you cloned from under the name 'origin'. If you run the command with the -v option, you can see the actual URL for each alias.

```
$ git remote
origin
$ git remote -v
origin	git@github.com:github/git-reference.git (fetch)
origin	git@github.com:github/git-reference.git (push)
```

You see the URL there twice because Git allows you to have different push and fetch URLs for each remote in case you want to use different protocols for reads and writes.

git remote add add a new remote repository of your project
If you want to share a locally created repository, or you want to take contributions from someone else's repository - if you want to interact in any way with a new repository, it's generally easiest to add it as a remote. You do that by running git remote add [alias] [url]. That adds [url] under a local remote named [alias].

For example, if we want to share our Hello World program with the world, we can create a new repository on a server (Using GitHub as an example), which should give you a URL, in this case "git@github.com:schacon/hw.git". To add that to our project so we can push to it and fetch updates from it we would do this:

```
$ git remote
$ git remote add github git@github.com:schacon/hw.git
$ git remote -v
github	git@github.com:schacon/hw.git (fetch)
```

```
github	git@github.com:schacon/hw.git (push)
```

Like the branch naming, remote alias names are arbitrary - just as 'master' has no special meaning but is widely used because git init sets it up by default, 'origin' is often used as a remote name because git clone sets it up by default as the cloned-from URL. In this case we'll name the remote 'github', but you could name it just about anything.

git remote rm removing an existing remote alias

Git addeth and Git taketh away. If you need to remove a remote - you are not using it anymore, the project is gone, etc - you can remove it with git remote rm [alias].

```
$ git remote -v
github	git@github.com:schacon/hw.git (fetch)
github	git@github.com:schacon/hw.git (push)
$ git remote add origin git://github.com/pjhyett/hw.git
$ git remote -v
github	git@github.com:schacon/hw.git (fetch)
github	git@github.com:schacon/hw.git (push)
origin	git://github.com/pjhyett/hw.git (fetch)
origin	git://github.com/pjhyett/hw.git (push)
$ git remote rm origin
$ git remote -v
github	git@github.com:schacon/hw.git (fetch)
github	git@github.com:schacon/hw.git (push)
```

git remote rename [old-alias] [new-alias] rename remote aliases

If you want to rename remote aliases without having to delete them and add them again you can do that by running git remote rename [old-alias] [new-alias]. This will allow you to modify the current name of the remote.

```
$ git remote add github git@github.com:schacon/hw.git
$ git remote -v
github	git@github.com:schacon/hw.git (fetch)
github	git@github.com:schacon/hw.git (push)
$ git remote rename github origin
$ git remote -v
origin	git@github.com:schacon/hw.git (fetch)
origin	git@github.com:schacon/hw.git (push)
```

In a nutshell with git remote you can list our remote repositories and whatever URL that repository is using. You can use git remote add to add new remotes, git remote rm to delete existing ones or git remote rename [old-alias] [new-alias] to rename them.

git remote set-url update an existing remote URL

Should you ever need to update a remote's URL, you can do so with the git remote set-url command.

```
$ git remote -v
github      git@github.com:schacon/hw.git (fetch)
github      git@github.com:schacon/hw.git (push)
origin      git://github.com/pjhyett/hw.git (fetch)
origin      git://github.com/pjhyett/hw.git (push)
$ git remote set-url origin git://github.com/github/git-reference.git
$ git remote -v
github      git@github.com:schacon/hw.git (fetch)
github      git@github.com:schacon/hw.git (push)
origin      git://github.com/github/git-reference.git (fetch)
origin      git://github.com/github/git-reference.git (push)
```

In addition to this, you can set a different push URL when you include the --push flag. This allows you to fetch from one repo while pushing to another and yet both use the same remote alias.

```
$ git remote -v
github      git@github.com:schacon/hw.git (fetch)
github      git@github.com:schacon/hw.git (push)
origin      git://github.com/github/git-reference.git (fetch)
origin      git://github.com/github/git-reference.git (push)
$ git remote set-url --push origin git://github.com/pjhyett/hw.git
$ git remote -v
github      git@github.com:schacon/hw.git (fetch)
github      git@github.com:schacon/hw.git (push)
origin      git://github.com/github/git-reference.git (fetch)
origin      git://github.com/pjhyett/hw.git (push)
```

Internally, the git remote set-url command calls git config remote, but has the added benefit of reporting back any errors. git config remote on the other hand, will silently fail if you mistype an argument or option and not actually set anything.

For example, we'll update the github remote but instead reference it as guhflub in both invocations.

```
$ git remote -v
github      git@github.com:schacon/hw.git (fetch)
github      git@github.com:schacon/hw.git (push)
origin      git://github.com/github/git-reference.git (fetch)
origin      git://github.com/github/git-reference.git (push)
$ git config remote.guhflub git://github.com/mojombo/hw.git
$ git remote -v
github      git@github.com:schacon/hw.git (fetch)
github      git@github.com:schacon/hw.git (push)
origin      git://github.com/github/git-reference.git (fetch)
origin      git://github.com/github/git-reference.git (push)
$ git remote set-url guhflub git://github.com/mojombo/hw.git
```

```
fatal: No such remote 'guhflub'
```

In a nutshell, you can update the locations of your remotes with git remote set-url. You can also set different push and fetch URLs under the same remote alias.

docs book git fetch download new branches and data from a remote repository

docs book git pull fetch from a remote repo and try to merge into the current branch

Git has two commands to update itself from a remote repository. git fetch will synchronize you with another repo, pulling down any data that you do not have locally and giving you bookmarks to where each branch on that remote was when you synchronized. These are called "remote branches" and are identical to local branches except that Git will not allow you to check them out - however, you can merge from them, diff them to other branches, run history logs on them, etc. You do all of that stuff locally after you synchronize.

The second command that will fetch down new data from a remote server is git pull. This command will basically run a git fetch immediately followed by a git merge of the branch on that remote that is tracked by whatever branch you are currently in. Running the fetch and merge commands separately involves less magic and less problems, but if you like the idea of pull, you can read about it in more detail in the official docs.

Assuming you have a remote all set up and you want to pull in updates, you would first run git fetch [alias] to tell Git to fetch down all the data it has that you do not, then you would run git merge [alias]/[branch] to merge into your current branch anything new you see on the server (like if someone else has pushed in the meantime). So, if you were working on a Hello World project with several other people and wanted to bring in any changes that had been pushed since we last connected, we would do something like this:

```
$ git fetch github
remote: Counting objects: 4006, done.
remote: Compressing objects: 100% (1322/1322), done.
remote: Total 2783 (delta 1526), reused 2587 (delta 1387)
Receiving objects: 100% (2783/2783), 1.23 MiB | 10 KiB/s, done.
Resolving deltas: 100% (1526/1526), completed with 387 local objects.
From github.com:schacon/hw
   8e29b09..c7c5a10  master     -> github/master
   0709fdc..d4ccf73  c-langs    -> github/c-langs
   6684f82..ae06d2b  java       -> github/java
 * [new branch]      ada        -> github/ada
 * [new branch]      lisp       -> github/lisp
```

Here we can see that since we last synchronized with this remote, five branches have been added or updated. The 'ada' and 'lisp' branches are new, where the 'master', 'c-langs' and 'java' branches have been updated. In our example case, other developers are pushing proposed updates to remote branches for review before they're merged into 'master'.

You can see the mapping that Git makes. The 'master' branch on the remote repository becomes a branch named 'github/master' locally. That way you can merge the 'master' branch on that remote into the local 'master' branch by running git merge github/master. Or, you can see what new commits are on that branch by running git log github/master ^master. If your remote is named 'origin' it would be origin/master instead. Almost any command you would run using local branches you can use remote branches with too.

If you have more than one remote repository, you can either fetch from specific ones by running git fetch [alias] or you can tell Git to synchronize with all of your remotes by running git fetch --all.

In a nutshell you run git fetch [alias] to synchronize your repository with a remote repository, fetching all the data it has that you do not into branch references locally for merging and whatnot.

docs book git push push your new branches and data to a remote repository
To share the cool commits you've done with others, you need to push your changes to the remote repository. To do this, you run git push [alias] [branch] which will attempt to make your [branch] the new [branch] on the [alias] remote. Let's try it by initially pushing our 'master' branch to the new 'github' remote we created earlier.

```
$ git push github master
Counting objects: 25, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (25/25), done.
Writing objects: 100% (25/25), 2.43 KiB, done.
Total 25 (delta 4), reused 0 (delta 0)
To git@github.com:schacon/hw.git
  * [new branch]           master -> master
```

Pretty easy. Now if someone clones that repository they will get exactly what we have committed and all of its history.

What if you have a topic branch like the 'erlang' branch created earlier and want to share just that? You can just push that branch instead.

```
$ git push github erlang
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 652 bytes, done.
Total 6 (delta 1), reused 0 (delta 0)
To git@github.com:schacon/hw.git
  * [new branch]           erlang -> erlang
```

Now when people clone or fetch from that repository, they'll get an 'erlang' branch they can look at and merge from. You can push any branch to any remote repository that you have write access to in this way. If your branch is already on the server, it will try to update it, if it is not, Git will add it.

The last major issue you run into with pushing to remote branches is the case of someone pushing in the meantime. If you and another developer clone at the same time, you both do commits, then she pushes and then you try to push, Git will by default not allow you to overwrite her changes. Instead, it basically runs git log on the branch you're trying to push and makes sure it can see the current tip of the server's branch in your push's history. If it can't see what is on the server in your history, it concludes that you are out of date and will reject your push. You will rightly have to fetch, merge then push again - which makes sure you take her changes into account.

This is what happens when you try to push a branch to a remote branch that has been updated in the meantime:

```
$ git push github master
To git@github.com:schacon/hw.git
 ! [rejected]            master -> master (non-fast-forward)
error: failed to push some refs to 'git@github.com:schacon/hw.git'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes before pushing again.  See the 'Note about
fast-forwards' section of 'git push --help' for details.
```

You can fix this by running git fetch github; git merge github/master and then pushing again.