Branch Checkout

The checkout process lets you switch branches. If you do:

```
$ git checkout devel
```

you have now switched to the development branch in the preceding example, and any files that have been changed will have their contents changed to reflect it. **HEAD** is set to the top commit of the branch.

Note that you have not lost the old branch; all the information to go back to it is still in the repository. All you would have to do is:

```
$ git checkout master
```

and the active branch will be reset and the file contents will revert. It is all blindingly fast too (changes induced by **git checkout**):

Command	Source Files	Index	Commit Chain	References
git checkout	Modified to match commit tree specified by branch or commit ID; un-tracked files not deleted	Unchanged	Unchanged	Current branch reset to that checked out; **HEAD** (in **.git/HEAD**) now refers to last commit in branch

If you have made changes to your working directory that have not yet been committed, switching branches would be a bad move. So, git will refuse to do it and will spit out an error message.

Suppose you do:

```
$ git branch devel

$ echo hello > hello

$ git add hello

$ git commit -a

$ git checkout devel
```

you will see the file **hello** does not exist in the devel branch.

It is also possible to combine the operations of creating a new branch and checking it out, by use of the **-b** option to the **checkout** operation. Doing:

```
$ git checkout -b newbranch startpoint
```

is entirely equivalent to:

```
$ git branch newbranch startpoint

$ git checkout newbranch
```