7.2 Git Tools - Interactive Staging

Interactive Staging

In this section, you'll look at a few interactive Git commands that can help you craft your commits to include only certain combinations and parts of files. These tools are helpful if you modify a number of files extensively, then decide that you want those changes to be partitioned into several focused commits rather than one big messy commit. This way, you can make sure your commits are logically separate changesets and can be reviewed easily by the developers working with you.

If you run `git add` with the `-i` or `--interactive` option, Git enters an interactive shell mode, displaying something like this:

```
$ git add -i
           staged     unstaged path
  1:    unchanged        +0/-1 TODO
  2:    unchanged        +1/-1 index.html
  3:    unchanged        +5/-1 lib/simplegit.rb

*** Commands ***
  1: [s]tatus     2: [u]pdate      3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff        7: [q]uit       8: [h]elp
What now>
```

You can see that this command shows you a much different view of your staging area than you're probably used to — basically, the same information you get with `git status` but a bit more succinct and informative. It lists the changes you've staged on the left and unstaged changes on the right.

After this comes a "Commands" section, which allows you to do a number of things like staging and unstaging files, staging parts of files, adding untracked files, and displaying diffs of what has been staged.

Staging and Unstaging Files

If you type `u` or `2` (for update) at the `What now>` prompt, you're prompted for which files you want to stage:

```
What now> u
           staged     unstaged path
  1:    unchanged        +0/-1 TODO
  2:    unchanged        +1/-1 index.html
  3:    unchanged        +5/-1 lib/simplegit.rb
Update>>
```

To stage the `TODO` and `index.html` files, you can type the numbers:

```
Update>> 1,2
           staged     unstaged path
* 1:    unchanged        +0/-1 TODO
* 2:    unchanged        +1/-1 index.html
  3:    unchanged        +5/-1 lib/simplegit.rb
Update>>
```

The `*` next to each file means the file is selected to be staged. If you press Enter after typing nothing at the `Update>>` prompt, Git takes anything selected and stages it for you:

```
Update>>
updated 2 paths

*** Commands ***
  1: [s]tatus     2: [u]pdate      3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff        7: [q]uit       8: [h]elp
What now> s
```

```
           staged     unstaged path
  1:        +0/-1      nothing TODO
  2:        +1/-1      nothing index.html
  3:    unchanged        +5/-1 lib/simplegit.rb
```

Now you can see that the `TODO` and `index.html` files are staged and the `simplegit.rb` file is still unstaged. If you want to unstage the `TODO` file at this point, you use the `r` or `3` (for revert) option:

```
*** Commands ***
  1: [s]tatus     2: [u]pdate      3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff        7: [q]uit       8: [h]elp
What now> r
           staged     unstaged path
  1:        +0/-1      nothing TODO
  2:        +1/-1      nothing index.html
  3:    unchanged        +5/-1 lib/simplegit.rb
Revert>> 1
           staged     unstaged path
* 1:        +0/-1      nothing TODO
  2:        +1/-1      nothing index.html
  3:    unchanged        +5/-1 lib/simplegit.rb
Revert>> [enter]
reverted one path
```

Looking at your Git status again, you can see that you've unstaged the `TODO` file:

```
*** Commands ***
  1: [s]tatus     2: [u]pdate      3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff        7: [q]uit       8: [h]elp
What now> s
           staged     unstaged path
  1:    unchanged        +0/-1 TODO
  2:        +1/-1      nothing index.html
  3:    unchanged        +5/-1 lib/simplegit.rb
```

To see the diff of what you've staged, you can use the `d` or `6` (for diff) command. It shows you a list of your staged files, and you can select the ones for which you would like to see the staged diff. This is much like specifying `git diff --cached` on the command line:

```
*** Commands ***
  1: [s]tatus     2: [u]pdate      3: [r]evert     4: [a]dd untracked
  5: [p]atch      6: [d]iff        7: [q]uit       8: [h]elp
What now> d
           staged     unstaged path
  1:        +1/-1      nothing index.html
Review diff>> 1
diff --git a/index.html b/index.html
index 4d07108..4335f49 100644
--- a/index.html
+++ b/index.html
@@ -16,7 +16,7 @@ Date Finder

 <p id="out">...</p>

-<div id="footer">contact : support@github.com</div>
+<div id="footer">contact : email.support@github.com</div>

 <script type="text/javascript">
```

With these basic commands, you can use the interactive add mode to deal with your staging area a little more easily.

Staging Patches

It's also possible for Git to stage certain *parts* of files and not the rest. For example, if you make two changes to your `simplegit.rb` file and want to stage one of them and not the other, doing so is very easy in Git. From

the same interactive prompt explained in the previous section, type `p` or `5` (for patch). Git will ask you which files you would like to partially stage; then, for each section of the selected files, it will display hunks of the file diff and ask if you would like to stage them, one by one:

```
diff --git a/lib/simplegit.rb b/lib/simplegit.rb
index dd5ecc4..57399e0 100644
--- a/lib/simplegit.rb
+++ b/lib/simplegit.rb
@@ -22,7 +22,7 @@ class SimpleGit
   end

   def log(treeish = 'master')
-    command("git log -n 25 #{treeish}")
+    command("git log -n 30 #{treeish}")
   end

   def blame(path)
Stage this hunk [y,n,a,d,/,j,J,g,e,?]?
```

You have a lot of options at this point. Typing `?` shows a list of what you can do:

```
Stage this hunk [y,n,a,d,/,j,J,g,e,?]? ?
y - stage this hunk
n - do not stage this hunk
a - stage this and all the remaining hunks in the file
d - do not stage this hunk nor any of the remaining hunks in the file
g - select a hunk to go to
/ - search for a hunk matching the given regex
j - leave this hunk undecided, see next undecided hunk
J - leave this hunk undecided, see next hunk
k - leave this hunk undecided, see previous undecided hunk
K - leave this hunk undecided, see previous hunk
s - split the current hunk into smaller hunks
e - manually edit the current hunk
? - print help
```

Generally, you'll type `y` or `n` if you want to stage each hunk, but staging all of them in certain files or skipping a hunk decision until later can be helpful too. If you stage one part of the file and leave another part unstaged, your status output will look like this:

```
What now> 1
           staged     unstaged path
  1:    unchanged        +0/-1 TODO
  2:        +1/-1      nothing index.html
  3:        +1/-1        +4/-0 lib/simplegit.rb
```

The status of the `simplegit.rb` file is interesting. It shows you that a couple of lines are staged and a couple are unstaged. You've partially staged this file. At this point, you can exit the interactive adding script and run `git commit` to commit the partially staged files.

You also don't need to be in interactive add mode to do the partial-file staging — you can start the same script by using `git add -p` or `git add --patch` on the command line.

Furthermore, you can use patch mode for partially resetting files with the `git reset --patch` command, for checking out parts of files with the `git checkout --patch` command and for stashing parts of files with the `git stash save --patch` command. We'll go into more details on each of these as we get to more advanced usages of these commands.

prev | next