Viewing Your Staged and Unstaged Changes

If the `git status` command is too vague for you — you want to know exactly what you changed, not just which files were changed — you can use the `git diff` command. We'll cover `git diff` in more detail later, but you'll probably use it most often to answer these two questions: What have you changed but not yet staged? And what have you staged that you are about to commit? Although `git status` answers those questions very generally by listing the file names, `git diff` shows you the exact lines added and removed — the patch, as it were.

Let's say you edit and stage the `README` file again and then edit the `CONTRIBUTING.md` file without staging it. If you run your `git status` command, you once again see something like this:

```
 git status
n branch master
our branch is up-to-date with 'origin/master'.
hanges to be committed:
 (use "git reset HEAD <file>..." to unstage)

   modified:   README

hanges 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:   CONTRIBUTING.md
```

To see what you've changed but not yet staged, type `git diff` with no other arguments:

```
 git diff
iff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
ndex 8ebb991..643e24f 100644
-- a/CONTRIBUTING.md
++ b/CONTRIBUTING.md
@ -65,7 +65,8 @@ branch directly, things can get messy.
Please include a nice description of your changes when you submit your PR;
if we have to read the whole diff to figure out why you're contributing
in the first place, you're less likely to get feedback and have your change
```

```
-merged in.
+merged in.  Also, split your changes into comprehensive chunks if your patch is
+longer than a dozen lines.

 If you are starting to work on a particular area, feel free to submit a PR
 that highlights your work in progress (and note in the PR title that it's
```

That command compares what is in your working directory with what is in your staging area. The result tells you the changes you've made that you haven't yet staged.

If you want to see what you've staged that will go into your next commit, you can use `git diff --staged`. This command compares your staged changes to your last commit:

```
 git diff --staged
iff --git a/README b/README
ew file mode 100644
ndex 0000000..03902a1
-- /dev/null
++ b/README
@ -0,0 +1 @@
My Project
```

It's important to note that `git diff` by itself doesn't show all changes made since your last commit — only changes that are still unstaged. If you've staged all of your changes, `git diff` will give you no output.

For another example, if you stage the `CONTRIBUTING.md` file and then edit it, you can use `git diff` to see the changes in the file that are staged and the changes that are unstaged. If our environment looks like this:

```
 git add CONTRIBUTING.md
 echo '# test line' >> CONTRIBUTING.md
 git status
n branch master
our branch is up-to-date with 'origin/master'.
hanges to be committed:
 (use "git reset HEAD <file>..." to unstage)

   modified:   CONTRIBUTING.md

hanges 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:   CONTRIBUTING.md
```

Now you can use `git diff` to see what is still unstaged:

```
 git diff
iff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
ndex 643e24f..87f08c8 100644
-- a/CONTRIBUTING.md
++ b/CONTRIBUTING.md
@ -119,3 +119,4 @@ at the
## Starter Projects

See our [projects list](https://github.com/libgit2/libgit2/blob/development/PROJECTS.md).
# test line
```

and `git diff --cached` to see what you've staged so far (`--staged` and `--cached` are synonyms):

```
 git diff --cached
iff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
ndex 8ebb991..643e24f 100644
-- a/CONTRIBUTING.md
++ b/CONTRIBUTING.md
@ -65,7 +65,8 @@ branch directly, things can get messy.
Please include a nice description of your changes when you submit your PR;
if we have to read the whole diff to figure out why you're contributing
in the first place, you're less likely to get feedback and have your change
merged in.
merged in.  Also, split your changes into comprehensive chunks if your patch is
longer than a dozen lines.

If you are starting to work on a particular area, feel free to submit a PR
that highlights your work in progress (and note in the PR title that it's
```

	Git Diff in an External Tool We will continue to use the `git diff` command in various ways throughout the rest of the book. There is another way to look at these diffs if you prefer a graphical or external diff viewing program instead. If you run `git difftool` instead of `git diff`, you can view any of these diffs in software like emerge, vimdiff and many more (including commercial products). Run `git difftool --tool-help` to see what is available on your system.

Revision #1
Created 16 April 2019 20:08:12 by ClassCloud
Updated 16 April 2019 20:33:57 by ClassCloud