Lab: Exploring Git Diff Changes

As we saw in the previous lab, we can refer to commits by their identifiers. You can refer to the *most recent commit* of the working directory by using the identifier `HEAD`.

We've been adding one line at a time to `mars.txt`, so it's easy to track our progress by looking, so let's do that using our `HEAD`s. Before we start, let's make a change to `mars.txt`, adding yet another line.

```
cd ~/Desktop/planets
nano mars.txt
cat mars.txt
```

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
An ill-considered change
```

Now, let's see what we get.

```
git diff HEAD mars.txt
```

```
diff --git a/mars.txt b/mars.txt
index b36abfd..0848c8d 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1,3 +1,4 @@
 Cold and dry, but everything is my favorite color
 The two moons may be a problem for Wolfman
 But the Mummy will appreciate the lack of humidity
+An ill-considered change.
```

which is the same as what you would get if you leave out `HEAD` (try it). The real goodness in all this is when you can refer to previous commits. We do that by adding `~1` (where "~" is "tilde", pronounced [**til**-d*uh*]) to refer to the commit one before `HEAD`.

```
git diff HEAD~1 mars.txt
```

If we want to see the differences between older commits we can use `git diff` again, but with the notation `HEAD~1`, `HEAD~2`, and so on, to refer to them:

```
git diff HEAD~3 mars.txt
```

```
diff --git a/mars.txt b/mars.txt
index df0654a..b36abfd 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1 +1,4 @@
 Cold and dry, but everything is my favorite color
+The two moons may be a problem for Wolfman
+But the Mummy will appreciate the lack of humidity
+An ill-considered change
```

We could also use `git show` which shows us what changes we made at an older commit as well as the commit message, rather than the *differences* between a commit and our working directory that we see by using `git diff`.

```
git show HEAD~3 mars.txt
```

```
commit f22b25e3233b4645dabd0d81e651fe074bd8e73b
Author: Vlad Dracula <fenago@example.com>
Date:   Thu Aug 22 09:51:46 2013 -0400

    Start notes on Mars as a base

diff --git a/mars.txt b/mars.txt
new file mode 100644
index 0000000..df0654a
--- /dev/null
+++ b/mars.txt
@@ -0,0 +1 @@
+Cold and dry, but everything is my favorite color
```

In this way, we can build up a chain of commits. The most recent end of the chain is referred to as `HEAD`; we can refer to previous commits using the `~` notation, so `HEAD~1` means "the previous commit", while `HEAD~123` goes back 123 commits from where we are now.

We can also refer to commits using those long strings of digits and letters that `git log` displays. These are unique IDs for the changes, and "unique" really does mean unique: every change to any set of files on any computer has a unique 40-character identifier. Our first commit was given the ID, so let's try this:

```
git diff UPDATE_ID_HERE mars.txt
```

```
diff --git a/mars.txt b/mars.txt
index df0654a..93a3e13 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1 +1,4 @@
 Cold and dry, but everything is my favorite color
+The two moons may be a problem for Wolfman
+But the Mummy will appreciate the lack of humidity
+An ill-considered change
```

That's the right answer, but typing out random 40-character strings is annoying, so Git lets us use just the first few characters (typically seven for normal size projects):

```
git diff UPDATE_ID_HERE mars.txt
```

```
diff --git a/mars.txt b/mars.txt
index df0654a..93a3e13 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1 +1,4 @@
 Cold and dry, but everything is my favorite color
+The two moons may be a problem for Wolfman
```

```
+But the Mummy will appreciate the lack of humidity
+An ill-considered change
```

All right! So we can save changes to files and see what we've changed. Now, how can we restore older versions of things? Let's suppose we change our mind about the last update to `mars.txt` (the "ill-considered change").

`git status` now tells us that the file has been changed, but those changes haven't been staged:

```
git status
```

```
On branch main
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:   mars.txt

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

We can put things back the way they were by using `git checkout`:

```
git checkout HEAD mars.txt
cat mars.txt
```

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
But the Mummy will appreciate the lack of humidity
```

As you might guess from its name, `git checkout` checks out (i.e., restores) an old version of a file. In this case, we're telling Git that we want to recover the version of the file recorded in `HEAD`, which is the last saved commit. If we want to go back even further, we can use a commit identifier instead:

```
git checkout UPDATE_ID_HERE mars.txt
```

```
cat mars.txt
```

```
Cold and dry, but everything is my favorite color
```

```
git status
```

```
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   mars.txt
```

Notice that the changes are currently in the staging area. Again, we can put things back the way they were by using `git checkout`:

```
git checkout HEAD mars.txt
```