Version Control with Git

(index.html)

Discussion

Frequently Asked Questions

People often have questions about Git beyond the scope of the core material. Students who have completed the rest of the lessons might find value in looking through the following topics.

Note that since this material isn't essential for basic Git usage, it won't be covered by the instructor.

More Advanced Git Configuration

In Setting Up Git (02-setup.html), we used `git config --global` to set some default options for Git. It turns out that these configuration options get stored in your home directory in a plain text file called `.gitconfig`.

```
$ cat ~/.gitconfig
```

```
[user]
    name = Vlad Dracula
    email = vlad@tran.sylvan.ia
[color]
    ui = true
[core]
    editor = nano
```

This file can be opened in your preferred text editor. (Note that it is recommended to continue using the `git config` command, as this helps avoid introducing syntax errors.)

Eventually, you will want to start customizing Git's behaviour. This can be done by adding more entries to your `.gitconfig`. The available options are described in the manual:

```
$ git config --help
```

In particular, you might find it useful to add aliases. These are like shortcuts for longer git commands. For example, if you get sick of typing `git checkout` all the time, you could run the command:

```
$ git config --global alias.co checkout
```

Now if we return to the example from Exploring History (05-history.html) where we ran:

```
$ git checkout f22b25e mars.txt
```

we could now instead type:

```
$ git co f22b25e mars.txt
```

Styling Git's Log

A good target for customization is output from the log. The default log is quite verbose but gives no graphical hints such as information about which commits were done locally and which were pulled from remotes.

You can use `git log --help` and `git config --help` to look for different ways to change the log output. Try the following commands and see what effect they have:

```
$ git config --global alias.lg "log --graph"
$ git config --global log.abbrevCommit true
$ git config --global format.pretty oneline
$ git lg
```

If you don't like the effects, you can undo them with:

```
$ git config --global --unset alias.lg
$ git config --global --unset log.abbrevCommit
$ git config --global --unset format.pretty
```

📌 Version Controlling Your Git Configuration

You can use the `--unset` flag to delete unwanted options from `.gitconfig`. Another way to roll back changes is to store your `.gitconfig` using Git.

For hints on what you might want to configure, go to GitHub and search for "gitconfig". You will find hundreds of repositories in which people have stored their own Git configuration files. Sort them by the number of stars and have a look at the top few. If you find some you like, please check that they're covered by an open source license before you clone them.

Non-text Files

Recall when we discussed Conflicts (09-conflict.html) there was a challenge that asked:

> What does Git do when there is a conflict in an image or some other non-textual file that is stored in version control?

We will now revisit this in more detail.

Many people want to version control non-text files, such as images, PDFs and Microsoft Office or LibreOffice documents. It is true that Git can handle these filetypes (which fall under the banner of "binary" file types). However, just because it can be done doesn't mean it should be done.

Much of Git's magic comes from being able to do line-by-line comparisons ("diffs") between files. This is generally easy for programming source code and marked up text. For non-text files, a diff can usually only detect that the files have changed but can't say how or where.

This has various impacts on Git's performance and will make it difficult to compare different versions of your project.

For a basic example to show the difference it makes, we're going to go see what would have happened if Dracula had tried using outputs from a word processor instead of plain text.

Create a new directory and go into it:

```
$ mkdir planets-nontext
$ cd planets-nontext
```

Use a program such as Microsoft Word or LibreOffice Writer to create a new document. Enter the same text that we began with before:

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

Save the document into the `planets-nontext` directory with the name of `mars.doc`. Back in the terminal, run the usual commands for setting up a new Git repository:

```
$ git init
$ git add mars.doc
$ git commit -m "Starting to think about Mars"
```

Then make the same changes to `mars.doc` that we (or Vlad) previously made to `mars.txt`.

```
Cold and dry, but everything is my favorite color
The two moons may be a problem for Wolfman
```

Save and close the word processor. Now see what Git thinks of your changes:

```
$ git diff
```

```
diff --git a/mars.doc b/mars.doc
index 53a66fd..6e988e9 100644
Binary files a/mars.doc and b/mars.doc differ
```

Compare this to the earlier `git diff` obtained when using text files:

```
diff --git a/mars.txt b/mars.txt
index df0654a..315bf3a 100644
--- a/mars.txt
+++ b/mars.txt
@@ -1 +1,2 @@
 Cold and dry, but everything is my favorite color
+The two moons may be a problem for Wolfman
```

Notice how plain text files give a much more informative diff. You can see exactly which lines changed and what the changes were.

An uninformative `git diff` is not the only consequence of using Git on binary files. However, most of the other problems boil down to whether or not a good diff is possible.

This isn't to say you should never use Git on binary files. A rule of thumb is that it's okay if the binary file won't change very often, and if it does change, you don't care about merging in small differences between versions.

We've already seen how a word processed report will fail this test. An example that passes the test is a logo for your organization or project. Even though a logo will be stored in a binary format such as `jpg` or `png`, you can expect it will remain fairly static through the lifetime of your repository. On the rare occasion that branding does change, you will probably just want to replace the logo completely rather than merge little differences in.
