Scripting

What exactly is scripting?

Good question isn’t it?

Technically, we are referring to non-compiled sequences of instructions that produce some effect. There are many examples, but most Linux users would recognize a bash shell script as an example of scripting.

Scripting Languages

To be a little more formal, scripting languages are a subset of programming languages that are written for a runtime environment. They are interpreted rather than compiled and you typically get immediate feedback on your steps rather than having to wait for the compiler to finish.

Advantages

Scripting languages are typically:

- Easy to learn
- Easy to edit
- Interactive
- Extensible

Scripting Use Cases

- Web applications can use javascript, perl, AJAX, etc.
- System administration uses the various shells, python, ruby, perl, etc.
- Game systems utilize lua (check out game hacking for more info)
- Extensions/Plugins for other systems - Nagios plugins are written in many scritping languages

What am I discussing?

Rather than go through a script and do a demo, I’m taking a slightly different approach this year - I’m going to look at some scripting resources specific to bash to keep the choices manageable.

Why do that?

The object is to provide a couple of resources that enhance the scripting experience. Things have come a long way since I started with the original *sh* and *csh* under BSD 4.1 and I'd expect that some of this will be new to at least some of you.

Colours in script output

If you have ever decided to use colour for messages in a script, you probably had to go and look up the ansi escape codes for them and either set a bunch of variables or set some function to do this for you.

Ansi

There is a bash script called **ansi** that acts as an echo statement with parameters. Depending on the type of terminal, your results will vary.

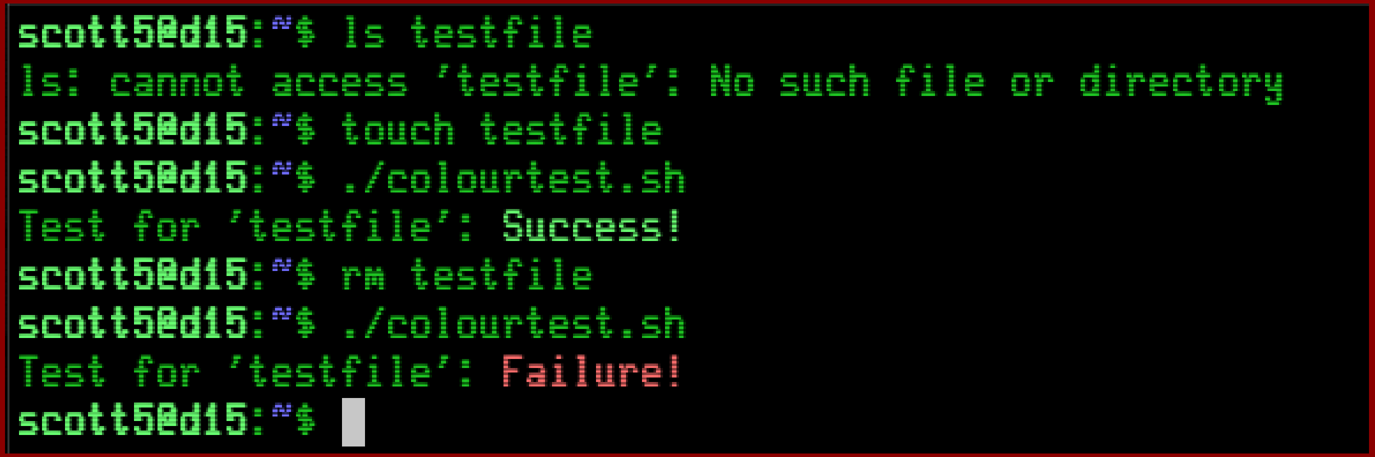
I like showing results with colour and if someone has done the work for me, I'll take it over looking up codes.

```
#!/usr/bin/env bash

ansi --no-newline "Test for 'testfile': "
if [ -f testfile ]; then
  ansi --green-intense "Success!"
else
  ansi --red-intense "Failure!"
fi
```

Sample output

This is a simple example, but you should get the idea.



```
scott5@d15:~$ ls testfile
ls: cannot access 'testfile': No such file or directory
scott5@d15:~$ touch testfile
scott5@d15:~$ ./colourtest.sh
Test for 'testfile': Success!
scott5@d15:~$ rm testfile
scott5@d15:~$ ./colourtest.sh
Test for 'testfile': Failure!
scott5@d15:~$ 
```

BATS

AKA: Bash Automated Testing System

A testing framework for verification that your script bahaves as expected. There is a lot of info available on the project page. This is a very simple example which is shown on the github page.

```
#!/usr/bin/env bats

@test "addition using bc" {
  result="$(echo 2+2 | bc)"
  [ "$result" -eq 4 ]
}

@test "addition using dc" {
  result="$(echo 2 2+p | dc)"
  [ "$result" -eq 4 ]
}
```

Test Results

BATS output for the previous example :

```
$ bats addition.bats
 ✓ addition using bc
 ✓ addition using dc

2 tests, 0 failures
```

Bash Powerline

What is it?

The short answer is it provides visual cues to your current status. That may not sound like much, but I have started to use it and I find it more useful than I would have thought.

Powerline Versions

There are a few variations of powerline out there, mostly in python.

This is a bash only version and works reasonably well. You really need a solarized terminal for it to be fully effective as it uses colours to indicate status and being able to easily distinguish them helps a lot.

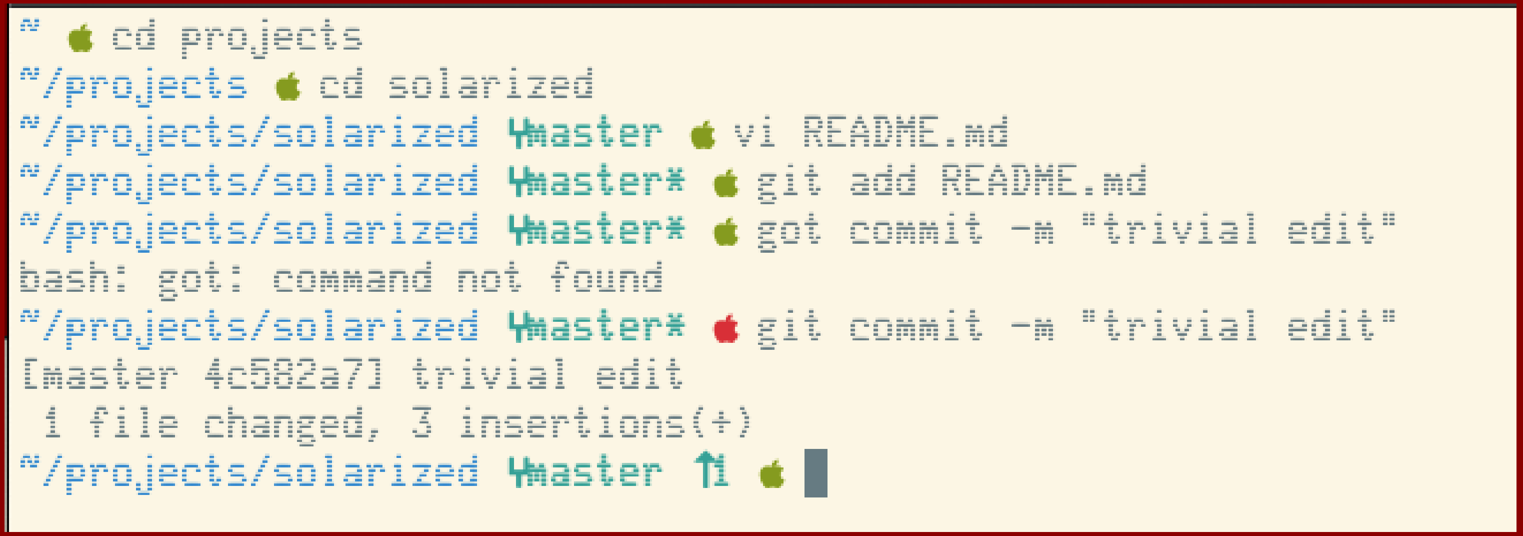
Features

- Platform-dependent prompt symbols.
- Color-coded prompt symbol according to previous command execution status.
- Use Bash builtin when possible to reduce delay.

Features

- Git: show branch name, tag name, or unique short hash.
- Git: show "*" symbol with uncommited modifications.
- Git: show “ ↑ ” symbol and number of commits ahead of remote.
- Git: show “ ↓ ” symbol and number of commits behind remote.

Powerline example output



```
~   cd projects
~/projects  cd solarized
~/projects/solarized master  vi README.md
~/projects/solarized master*  git add README.md
~/projects/solarized master*  got commit -m "trivial edit"
bash: got: command not found
~/projects/solarized master*  git commit -m "trivial edit"
[master 4c582a7] trivial edit
 1 file changed, 3 insertions(+)
~/projects/solarized master ⇡1  █
```

Summary

There is a wealth of information out there to aid you in creating a more feature rich (complicated) shell script or to just enhance your shell environment.

Unless you are looking for some serious speed or have other constraints, these resources are useful in all manner of situations.

I have included a list at the end here of a couple of places to look.

For more information see

- https://github.com/alebcay/awesome-shell
- https://awesomeopensource.com/project/awesome lists
- https://github.com/vinta/awesome-python
- https://opensource.com/article/19/2/testing-bash-bats
- https://www.dolthub.com/blog/2020-03-23-testing-dolt-bats
- https://www.oreilly.com/library/view/infrastructure as-code/9781786464910/ch10s02.html
- https://www.geeksforgeeks.org/