【Linux Programming】Day3

Tags	
Date	@May 15, 2022
Summary	

【Ch2】Shell Programming

2.3 The Shell as a Programming Language

There are two ways of writing shell programs:

1. We can type a sequence of commands and allow the shell to execute them interactively
2. Or we can store those commands in a file that we can then invoke as a program.

2.3.1 Interactive Programs

Suppose we have a large number of C files and wish to examine the files that contain the string `POSIX`.

Rather than search using the g`rep` command for the string in the files and then list the files individually, we could perform the whole operation in an interactive script like this:

```
$ for file in *
> do
> if grep -l POSIX $file
> then
> more $file
> fi
> done
```

Note how the normal `$` shell prompt changes to a `>` when the shell is expecting further input. We can type away, letting the shell decide when we're finished, and the script will execute immediately.

In this example, the `grep` command prints the files it finds containing POSIX and then more displays the contents of the file to the screen. Finally, the shell prompt returns.

Note also that we called the shell variable that deals with each of the files to self-document the script. We could have equally used `i`, but `file` is more meaningful.

The shell also performs wildcard expansion(often referred to as globbing).

The use of '*' as a wildcard to match a string of characters.

`?`: Match a single character

`[set]`: Allows any of a number of single characters to be checked.

`[^set]`: Negate the set

Brace expansion using `{ }`: Allows us to group arbitrary strings together in a set that the shell will expand. For example,

```
$ ls my_{finger,toe}s
```

will list the files `my_fingers` and `my_toes`.

In addition

```
$ more $(grep -l POSIX *)
```

will output the same result.