Professional Skills
Agile Fundamentals
Jira
Git
Databases Introduction
Java Beginner
Maven
Testing (Foundation)
Java Intermediate
HTML
CSS
Javascript
Spring Boot
Selenium
Sonarqube
Advanced Testing (Theory)
Cucumber
MongoDB
Express
NodeJS
React
Express-Testing
Networking
Security
Cloud Fundamentals
AWS Foundations
AWS Intermediate
Linux
- Linux Introduction

Aliases, Functions and Variables

Contents

- Overview
- Shell variables
  - Setting variables
  - Some standard Shell Variables
    - The Path Variable
- Aliases
  - Setting Aliases
- Functions
  - Defining functions
- Persistence
- Tutorial
- Exercises

Overview

The `bash environment` is an area for the user to work in.
In Linux, this would either be the shell or the GUI.
A shell environment is made up of *variables*, *aliases* and *functions*.

Shell variables

Variables are just content stored in a callable keyword.
They are useful in situations where you need to use sensitive data, but want to keep it hidden.
They are also sometimes requried in order for a process to run.

Setting variables

Variable names must begin with a letter.
They are defined by setting the *variable* name on the left, and the variable value on the right.

```
$ user=steve
```

In order to be seen and used by a child process, *variables* must be exported.

```
$ export user
$ export greeting=hello
```

We can then echo these varibales to display them on the terminal. For example:

```
$ echo ${greeting} ${user}
```

would return:

```
hello steve
```

There are three commands which are useful when dealing with *variables*:

- `set` displays all shell variables
- `env` displays all exported variables
- `unset <variable name>` deletes a variable

Some standard Shell Variables

Linux Distributions
Bash Interpreter
Sessions in Linux
Lists and Directories
Editing Text
Aliases, Functions and Variables
User Administration
Ownership
Data Streams
Pipes and Filters
Scripting in Linux
Sudoers
Managing systemd Services
Systemd Service Configuration
OpenSSH
Screens in Linux
DevOps
Jenkins Introduction
Jenkins Pipeline
Markdown
IDE Cheatsheet

```
HOME        # the home directory for the user
PATH        # the search path
PS1, PS2    # primary, and secondary prompt strings
```

The Path Variable

The `PATH` *variable* sets the directories which the shell will search when trying to run a program.
Each directory is given by the full path, and is separated by a colon (`:`).

```
$ echo $PATH
/bin:/usr/bin:/usr/local/bin:/usr/bin/X11
```

You can add directories to the `PATH` when you install new programs.
Usually there is an option to do this, but we can do it manually with the following command:

```
$ export PATH=[name of directory]:$PATH
```

replacing the [name of directory] with the name of the directory you wish to add to the PATH.

However, you should never remove any directories from the `PATH`, as the shell might not be able to find some programs if you do.

Aliases

Aliases are usually used to substitute commands for a callable keyword.
They are useful for when you are having to type out long commands repeatedly, and you want to save yourself some time.
We can also use *aliases* to override existing commands.

Setting Aliases

```
$ alias rm=echo
$ rm /tmp/junk
/tmp/junk
$ unalias rm
```

Here, we are overriding what the `rm` command does and replacing it with the functionality of the `echo` command.
You can see that instead of removing the file `/tmp/junk`, we simply print it to the terminal.
This is a pretty trivial example to illustrate how *aliases* work, but this can be applied to much more useful commands.

e.g. if you use the command `ls -al` a lot then you can *alias* that to simply `ll`

```
$ alias ll="ls -al"
```

Functions

Functions consist of repeatable content held in a callable keyword.
Similarly to *aliases* they are mainly used to save time, but *functions* consist of a set of multiple commands, intsead of the single command stored in an *alias*.

Defining functions

A bash *function* is defined by a callable keyword, and a set of commands for the shell to execute.

```
$ helloworld() {
> echo "Hello Everyone!"
> pwd
> ls
> }
```

This *function* displays on the terminal "Hello Everyone!", followed by the current working directory and the contents of that directory.

The callable keyword here is `helloworld`.

Note: The `>` is defined by the `PS2` variable which was discussed in the variables section.

Functions are a useful way to package up several commands to be called by a single keyword, but the example above is fairly inflexible. To make functions more flexible we can pass them **arguments**.

Arguments in bash are positional, so in a function definition, `$1` represents the first argument the function is called with, `$2` the second, and so forth. Consider the following example of a function which can be used to initialise a git repository with a given name:

```
$ initrepo() {
>    mkdir $1 && cd $_
>    git init
>    echo "# $1" > README.md
>    git add README.md
>    git commit -m 'Initial commit'
> }
```

To initialise a git repository we can then call this function with a name of our choosing, e.g.:

```
$ initrepo example-repo
```

to create a repository called example-repo, initialised with a README file. To create another repo we could use the same function with a different name, giving greater flexibility in the use of our functions.

Persistence

All *variables*, *aliases* and *functions* which are defined on the command line will be lost on logout.
So we need a way to keep these after the user session ends.

There are four files which execute when the user starts a session, these are:

- `/etc/profile` - This is where *variables* are stored if they need to be set for all users.
- `/etc/bashrc` - This is where *aliases* and *functions* are stored if they need to be set for all users.
  For `ubuntu` machines, the file is called `/etc/bash.bashrc` instead.
- `~/.bashrc` - This is where *aliases* and *functions* are stored for one user, this file lives in that user's home directory.
- `~/.bash_profile` - This is where *variables* are stored for one user, this file lives in that user's home directory.

Tutorial

▶ A Useful Alias
▶ An Example Function

Exercises

How would you display all exported variables?

▶ Show Solution

What is stored in the PS1 variable?

▶ Show Solution

Write a function which makes a new directory called 'exercises' and changes directory to it

▶ Show Solution