Leanpub Sample Technical Book

Shell Functions

how to – create, save, & re-use

Marty McGowan

This book is for sale at http://leanpub.com/shellfunctions

This version was published on 2013-10-19

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Tweet This Book!

Please help Marty McGowan by spreading the word about this book on Twitter!

The suggested hashtag for this book is #shell.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#shell

Contents

Acknowledgments

- my friends on the itToolbox, shell forum
- helpers from the StackOverflow
- Nitin Chitniss, for the incentive to capture small shell functions
- Bill Anderson, with whom I collaborated on “The Software Assembly Line” and has provided support over the decades
- Pat, for the first edit for “voice”
- my employers of the past decade:
  - Benedictine Academy, Elizabeth NJ, who taught real challenges
  - Fidessa, who re-invigorated my love for Unix® and the shell
  - and Union Co College, Cranford NJ, who has eased me into retirement

Preface

In this book you will learn how to write, save, and re-use shell functions.

The shell concepts are introductory. You may be familiar with them all. But if you aren't familiar with shell functions, you will learn how simple and powerful they are to use. These exercises are for you if you are the least bit curious about how to write and use shell functions.

Introduction

Each chapter is simple enough to require a quarter to a half an hour of your time. Each is meant to be worked at a terminal window on a Unix®, Linux, or other *nix server.

When you've completed these exercises, you will be comfortable with creating, using, saving and re-using shell functions.

As an introduction, this short book as is the first of similar books on the shell function. You can explore the planned topics in the what's next section.

To get started, here are the few assumptions we make. That you:

- have access to an open terminal window
- can open simultaneous mulitple terminal windows
- are running the **bash** shell
- have experience with one of the popular editors: **vi**, vim, … or **emacs** for command line editing.

If you need help getting started, you can contact me here[1] now, and at relevant places in the exercises.

[1] mailto:mcgowan@alum.mit.edu?subject=introduction

1 Write a shell function

The simplest shell functions may be written on a single line at the command prompt.

In this chapter, you will write and use two simple shell functions: **hello**, and **today**.

1.1 Hello world

In this book, you can assume your command prompt is the dollar sign:

```
1   $ ...
```

Here is the *Programmers Birth Announcement'* – **Hello World!**. Type it at your command prompt:

```
1 $ hello () { echo 'Hello World!'; }
```

So, on the above line, you type everything from **hello** thru the closing curly brace, followed by a carriage return.

You use the function by typing its *name* at the command line. Here is the function definition (on line 1), followed by using it (line 2), and the shell's response (line 3). The next shell prompt is line 4.

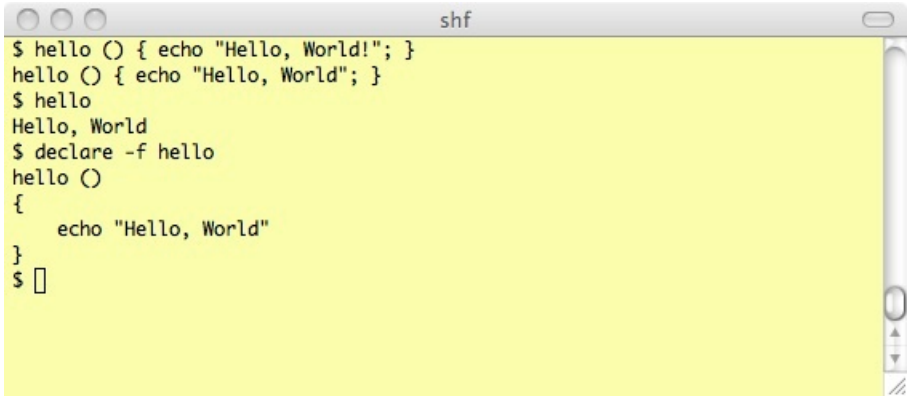
```
1 $ hello () { echo 'Hello World!' ; }
2 $ hello
3 Hello World!
4 $
```

You can see the definition of a function with the **declare** built-in:

```
1 $ declare -f hello
```

Type that command. Notice your function has been slightly reformatted. More on that later.

Here is a screenshot;



1.2 Getting it right

The function syntax:

name () { command ... ; }

has a **name** of your choosing, and a **command** or semi-colon-separated commands of your choosing. While there are other ways to define a function, I've found the parenthesis-pair simplest to identify the name. A pair of curly braces enclose the commands. And if the trailing curly brace is on the same line as a command, you need a semi-colon separator. You can separate commands on separate lines. The only mandatory space in the function definition is the space following the first curly brace.

questions:

- *What does the **declare** command tell you about the function syntax?*
- *how might you write a function to capture that idea?* a good answer requires you know how to use function arguments. feel free to experiment.
- *what would you name that function?*

1.3 More interesting

Arguments, like file names and options, make functions more useful. But before looking at how arguments are used, whet your appetite with this one, called **today**:

```
$ today () { date +%Y%m%d; }
$ today
20131008
$
```

Type the definition and invoke your new function **today**. Since **date** takes almost any upper- or lower-case letter, we'll deal with those later as arguments.

1.4 Activity

- investigate the options to the **date** command: search for *unix manual date*.

Mail me if you have questions[1]

[1] mailto:mcgowan@alum.mit.edu?subject=writeAshellFunction

2 Whats next?

Before taking a peek at what's next, it's time to assess where you are. First, you've used these commands and shell built-ins, in more or less the order encountered in the text:

```
echo
date
declare
for
do
done
set
eval
local
shift
history
grep
awk
source
tee
chmod
cat
printf
```

If you have any questions about them, it's quite simple to search for **bash shell** *command-name*.

You now possess the skills to begin crafting your own function library and make it available on later terminal sessions. You are now able to collect and re-use functions as you come to need them. You'll collect them, store and re-use them as multiple instances of the one case you have worked in the book.

As you do that, you'll recognize other challenges:

- how do I use these functions as part of another library?
- can I repair a function quickly as the need arises and easily restore it to it's proper library?
- it seems I'm getting a large collection of functions: how do I keep them straight?

2.1 Looking ahead

The very next subject I'm planning is that of library management. I've functions on hand to:

- capture a daily log of when functions were created.
- quickly update a function library with additions or changes
- deleting a function or moving a function from one library to another,

This all belongs to a practice I've established on the content and form of a function library:

- building a Quick-Reference, which may be part of
- more extensive documentation
- what you may,may not, and must do when *source*ing a library.

This is all supported by what I've called: *Then only backup system you'll ever need.* That said, in the last two years, I've been an avid user of github[1]. And not yet that experienced with **git**, so what you'll see in the backup system I offer might be called a crutch, but I use it as the routine check-point, preserving files in a more accessible state than git offers: i.e, using ordinary commands, such as **cp** to retrieve a backed-up version, which may be more than one edition old.

And to control versions, the subject of the *cloud* appears. At this moment, that includes git, and Dropbox[2], so a practice and a function library to make that usage as concise as you need. In the last month (Sept-Oct 2013) I've started using Dropbox as my virtual **HOME** directory.

At some point I will discuss the distinction between using shell function libraries and the more common parlance of the *shell script*. If you haven't noticed this yet, I hadn't use the term before. That's conscious, since much of my practice is devoted to the command line. We will need to make the connection between this view of the shell, and connecting to the business needs. A good place to do that is constructing an appliction suitable for a cron job[3]

[1] http://github.com/applemcg
[2] http://Dropbox.com
[3] http://en.wikipedia.org/wiki/Cron

2.2 Further ahead

I have shell functions for the major application areas of:

- make – the utility conventionally used to build programs, I've long held that **make** offers much wider use than in application development, particualrly documentation, testing, and management information.
- database – I've been carrying around a personal copy of the too-little-used RDB[4].
  Originally built on awk[5] there are now *perl*-based implementations. I've stayed close to the author's original idea that *the shell is the only 4GL you'll ever need.*
- documentation – I'm a more recent convert to **markdown**. This offers the prospect of sharpening your tools to produce a properly indexed and referenced document. This is one of my goals here.

Mail me if you have questions[6]

[4] http://www.amazon.com/Relational-Database-Management-Prentice-Hall-Software/dp/013938622X
[5] http://www.grymoire.com/Unix/Awk.html
[6] mailto:mcgowan@alum.mit.edu?subject=whatsNext