Mastering UNIX® Shell Scripting

Bash, Bourne, and Korn Shell Scripting for Programmers, System Administrators, and UNIX Gurus

Second Edition

Randal K. Michael

Wiley Publishing, Inc.

Mastering UNIX® Shell Scripting

Second Edition

Mastering UNIX® Shell Scripting

Bash, Bourne, and Korn Shell Scripting for Programmers, System Administrators, and UNIX Gurus

Second Edition

Randal K. Michael

Wiley Publishing, Inc.

Mastering UNIX®Shell Scripting: Bash, Bourne, and Korn Shell Scripting for Programmers, System Administrators, and UNIX Gurus, Second Edition

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-18301-4

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

This book is dedicated to my wife Robin, the girls, Andrea and Ana, and the grandchildren, Gavin, Jocelyn, and Julia – my true inspiration.

About the Author

Randal K. Michael is a UNIX Systems Administrator working as a contract consultant. He teaches UNIX shell scripting in corporate settings, where he writes shell scripts to address a variety of complex problems and tasks, ranging from monitoring systems to replicating large databases. He has more than 30 years of experience in the industry and 15 years of experience as a UNIX Systems Administrator, working on AIX, HP-UX, Linux, OpenBSD, and Solaris.

Credits

Executive Editor
Carol Long

Development Editor
John Sleeva

Technical Editor
John Kennedy

Production Editor
Dassi Zeidel

Copy Editor
Kim Cofer

Editorial Manager
Mary Beth Wakefield

Production Manager
Tim Tate

Vice President and Executive Group Publisher
Richard Swadley

Vice President and Executive Publisher
Joseph B. Wikert

Project Coordinator, Cover
Lynsey Stanford

Proofreader
Candace English

Indexer
Robert Swanson

Contents

Acknowledgments

The information that I gathered together in this book is the result of working with some of the most talented UNIX professionals on the topic. I have enjoyed every minute of my association with these UNIX gurus and it has been my pleasure to have the opportunity to gain so much knowledge from the pros. I want to thank every one of these people for asking and answering questions over the past 20 years. If my brother Jim had not kept telling me, ''you should write a book,'' after querying me for UNIX details on almost a weekly basis, I doubt the first edition of this book would have ever been written.

I especially want to thank Jack Renfro at Chrysler Corporation for giving me my first shell scripting project so long ago. I had to start with the man pages, but that is how I learned to dig deep to get the answer. Since then I have been on a mission to automate, through shell scripting, support tasks on every system I come in contact with. I certainly value the years I was able to work with Jack.

I must also thank the talented people at Wiley Publishing. As executive editor, Carol Long helped keep things going smoothly. Development editor John Sleeva kept me on schedule and made the edits that make my writing flow with ease. Dassi Zeidel, my production editor, helped with the final edits and prepared the book for layout. John Kennedy, my technical editor, kept me honest, gave me some tips, and ensured the code did not have any errors. It has been a valuable experience for me to work with such a fine group of professionals at Wiley Publishing. I also want to thank my agent, Carole McClendon, at Waterside Productions for all her support on this project. Carole is the best agent that anyone could ever ask for. She is a true professional with the highest ethics.

Of course, my family had a lot to do with my success on this and every project. I want to thank Mom, Pop, Gene, Jim, Marcia, Rusty, Mallory, Anica, and Chad. I want to thank my beautiful bride forever, Robin, for her understanding, patience, and support for the long hours required to complete this project. The girls, Andrea and Ana, always keep a smile on my face, and Steve is always on my mind. The grandchildren, Gavin, Jocelyn, and Julia, are an inspiration for long life, play time, learning, and adventure. I am truly living the dream.

I could not have written this book without the support of all these people and the many others that remain unnamed. It has been an honor!

Introduction

In UNIX there are many ways to accomplish the same task. Given a problem to solve, we may be able to get to a solution in any number of ways. Of course, some techniques will be more efficient, use fewer system resources, and may or may not give the user feedback on what is going on or give more accurate details and more precision to the result. In this book we are going to step through every detail of creating shell scripts to solve real-world UNIX problems and tasks. The shell scripts range from using a pseudo-random number generator to creating passwords using arrays to replicating data with `rsync` to working with record files. The scope of solutions is broad and detailed. The details required to write a good shell script include commenting each step for future reference. Other details include combining many commands together into a single command statement when desirable, separating commands on several lines of code when readability and understanding the concept may be diminished, and making a script readable and easy to maintain through the life cycle. We will see the benefits of variables and files to store data, show methods to strip out unneeded data from command output, and format data for a particular purpose. Additionally, we are going to show how to write and use functions in our shell scripts and demonstrate the benefits of functions over a shell script written without functions.

This book is intended for any flavor of UNIX, but it emphasizes the AIX, HP-UX, Linux, OpenBSD, and Solaris operating systems. Almost every script in the book is also included on the book's companion web site (`www.wiley.com/go/michael2e`). Many of the shell scripts are rewritten for various UNIX flavors, when it is necessary. Other shell scripts are not platform-dependent. These script rewrites are necessary because command syntax and output vary, sometimes in a major way, between UNIX flavors. The variations are sometimes as small as extracting data out of a different column or using a different command switch to get the same result, or they can be as major as putting several commands together to accomplish the same task and get a similar output or result on different flavors of UNIX.

In each chapter we start with the very basic concepts to accomplish a task, and then work our way up to some very complex and difficult concepts. The primary purpose of a shell script is to automate repetitive and complex tasks. This alleviates keystroke errors and allows for time-scheduled execution of the shell scripts. It is always better to have the system tell us that it has a problem than to find out too late to be proactive. This book will help us to be more proactive and efficient in our dealing with the system. At every level you will gain more knowledge to allow you to move on to ever

increasingly complex ideas with ease. You are going to see different ways to solve real-world example tasks. There is not just one way to solve a challenge, and we are going to look at the pros and cons of attacking a problem in various ways. Our goal is to be confident and flexible problem solvers. Given a task, we can solve it in any number of ways, and the solution will be intuitively obvious when you complete this book.

Overview of the Book and Technology

This book is intended as a learning tool and study guide to learn how to write shell scripts to solve a multitude of problems by starting with a clear goal. We will cover most shell scripting techniques about seven times, each time hitting the topic from a different angle, solving a different problem. I have found this technique to work extremely well for retention of the material.

Each chapter ends with Lab Assignments that let you either write a new script or modify a shell script covered in the chapter. There is not a ''solutions'' book. The solution is to make it work! I urge everyone to read this book from cover to cover to get the maximum benefit. The shells covered in this book include Bash, Bourne, and Korn. C shell is not covered. Advanced topics include using `rsync` to replicate data, creating snapshot-style backups utilizing Dirvish, working with record files to parse data, and many others.

This book goes from some trivial task solutions to some rather advanced concepts that everyone from high school and college students to Systems Administrators will benefit from, and a lot in between. There are several chapters at each level of complexity scattered throughout the book. The shell scripts presented in this book are complete shell scripts, which is one of the things that sets this book apart from other shell-scripting books on the market. The solutions are explained thoroughly, with each part of the shell scripts explained in minute detail down to the philosophy and mindset of the author.

How This Book Is Organized

Each chapter starts with a typical UNIX challenge that occurs every day in the computer world. With each challenge we define a specific goal and start the shell script by defining the correct command syntax to solve the problem. After we present the goal and command syntax, we start by building the shell script around the commands. The next step is to filter the commands' output to strip out the unneeded data, or we may decide to just extract the data we need from the output. If the syntax varies between UNIX flavors, we show the correct syntax to get the same or a similar result. When we get to this point we go further to build options into the shell script to give the end user more flexibility on the command line.

When a shell script has to be rewritten for each operating system, a combined shell script is shown at the end of the chapter that will run on all the UNIX flavors studied in this book, except where noted. To do this last step, we query the system for the UNIX flavor using the `uname` command. By knowing the flavor of the operating system, we are able to execute the proper commands for each UNIX flavor by using a simple

`case` statement. If this is new to you, don't worry; everything is explained in detail throughout the book.

Each chapter targets a different real-world problem. Some challenges are very complex, whereas others are just interesting to play around with. Some chapters hit the problem from several different angles in a single chapter, and others leave you the challenge to solve on your own — of course, with a few hints to get you started. Each chapter solves the challenge presented and can be read as a single unit without referencing other chapters in the book, except where noted. Some of the material, though, is explained in great detail in one chapter and lightly covered in other chapters. Because of this variation, I recommend that you start at the beginning of the book and read and study every chapter, and solve each of the Lab Assignments through to the end of the book, because this *is* a learning experience!

Who Should Read this Book

This book is intended for anyone who works with UNIX from the command line on a daily basis. The topics covered in this book are mainly for UNIX professionals — computer science students, programmers, programmer-analysts, Systems Operators, application support personnel, Systems Administrators, and anyone who is interested in getting ahead in the support and development arenas. Beginners will get a lot out of this book, too, although some of the material may be a little high-level, so a basic UNIX book may be needed to answer some questions. Everyone should have a good working knowledge of common UNIX commands before starting this book; we do not explain basic UNIX commands in much detail.

I started my career in UNIX by learning on the job how to be a Systems Operator. I wish I had a book like this when I started. Having this history, I wanted others to get a jump-start on their careers. I wrote this book with the knowledge that I was in your shoes at one time, and I remember that I had to learn everything from the man pages, one command at a time. Use this book as a study guide, and you will have a jump-start to get ahead quickly in the UNIX world, which is getting bigger all the time.

Tools You Will Need

To get the most benefit from this book you need access to a UNIX machine, preferably with AIX, HP-UX, Linux, OpenBSD, or Solaris installed. You can run Linux, Solaris, and OpenBSD on standard PC hardware, and this is relatively inexpensive, if not free. Your default shell should be set to Bash or Korn shell. You can find your default shell by entering **echo $SHELL** on the command line. None of the shell scripts in this book requires a graphical terminal, but it does not hurt to have Gnome, CDE, KDE, or X-Windows running. This way you can work in multiple windows at the same time and cut and paste code between windows.

You also need a text editor that you are comfortable using. UNIX operating systems come with the **vi** editor, and many include **emacs**. You can also use the text editor that comes with KDE, CDE, and Gnome. Remember that the editor must be a text editor that stores files in a standard ANSII format. You will also need some time, patience, and an open, creative mind that is ready to learn.

Another thing to note is that all of the variables used in the shell scripts and functions in this book are in uppercase characters. I did this because it is much easier to follow along with the shell script if you know quickly where the variables are located in the code. When you write your own shell scripts, please use lowercase for all shell script and function variables. The reason this is important is that the operating system, and applications, use *environment variables* that are uppercase. If you are not careful, you can overwrite a critical system or application variable with your own value and hose the system; however, this is dependent on the scope of where the variable is in the code. Just a word of warning: be careful with uppercase variables!

What's on the Web Site

On the book's companion web site, `www.wiley.com/go/michael2e`, all the shell scripts and most of the functions that are studied in the book can be found. The functions are easy to cut and paste directly into your own shell scripts to make the scripting process a little easier. Additionally, there is a shell script *stub* that you can copy to another filename. This script stub has everything to get started writing quickly. The only thing you need to do is fill in the fields for the following: Script Name, Author, Date, Version, Platform, and Rev. List, when revisions are made. There is a place to define variables and functions, and then you have the ''BEGINNING OF MAIN'' section to start the main body of the shell script.

Summary

This book is for learning how to be creative, proactive, and professional problem solvers. Given a task, the solution will be *intuitively obvious* to you on completion of this book. This book will help you attack problems logically and present you with a technique of building on what you know. With each challenge presented you will see how to take basic syntax and turn it into the basis for a shell scripting solution. We always start with the basics and build more and more logic into the solution before we add additional options the end user can use for more flexibility.

Speaking of end users, we must always keep our users informed about how processing is proceeding. Giving the user a blank screen to look at is the worst thing that you can do, so for this we can create progress indicators. You will learn how to be proactive by building tools that monitor for specific system events and situations that indicate the beginning stages of an upcoming problem. This is where knowing how to query the system puts you ahead of the game.

With the techniques presented in this book, you will learn. You will learn about problem resolution. You will learn about starting with what you know about a situation and building a solution effectively. You will learn how to make a single shell script work on other platforms without further modifications. You will learn how to be proactive. You will learn how to use plenty of comments in a shell script. You will learn how to write a shell script that is easy to read and follow through the logic. Basically, you will learn to be an effective problem solver, and the solution to any challenge will be *intuitively obvious*!

PART I

The Basics of Shell Scripting

CHAPTER 1

Scripting Quick Start and Review

We are going to start out by giving a targeted refresher course. The topics that follow are short explanations of techniques that we always have to search the book to find; here they are all together in one place. The explanations range from showing the fastest way to process a file line-by-line to the simple matter of case sensitivity of UNIX and shell scripts. This should not be considered a full and complete list of scripting topics, but it is a very good starting point and it does point out a sample of the topics covered in the book. For each topic listed in this chapter there is a very detailed explanation later in the book.

We urge everyone to study this entire book. Every chapter hits a different topic using a different approach. The book is written this way to emphasize that there is never only one technique to solve a challenge in UNIX. All the shell scripts in this book are real-world examples of how to solve a problem. Thumb through the chapters, and you can see that we tried to hit most of the common (and some uncommon!) tasks in UNIX. All the shell scripts have a good explanation of the thinking process, and we always start out with the correct command syntax for the shell script targeting a specific goal. I hope you enjoy this book as much as I enjoyed writing it. Let's get started!

Case Sensitivity

UNIX is case sensitive. Because UNIX is case sensitive, our shell scripts are also case sensitive.

UNIX Special Characters

All of the following characters have a special meaning or function. If they are used in a way that their special meaning is not needed, they must be *escaped*. To escape,

or remove its special function, the character must be immediately preceded with a backslash, \, or enclosed within ' 'forward tic marks (single quotes).

```
\ / ; , . ~ # $ ? & * ( ) [ ] ' ' " + - ! ^ = | < >
```

Shells

A *shell* is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of shells, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions. This book works with the Bourne, Bash, and Korn shells. Shells are located in either the `/usr/bin/` directory or the `/bin/` directory, depending on the UNIX flavor and specific version.

Table 1-1

SHELL	DIRECTORY
Bourne	`/bin/sh` or `/usr/bin/sh`
Bash	`/bin/Bash` or `/usr/bin/Bash`
Korn	`/bin/ksh` or `/usr/bin/ksh`

Shell Scripts

The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by a pound sign or hash mark, #, describing the steps. There are conditional tests, such as value A is greater than value B, loops allowing us to go through massive amounts of data, files to read and store data, variables to read and store data, and the script may include *functions*.

We are going to write a lot of scripts in the next several hundred pages, and we should always start with a *clear goal* in mind. With a clear goal, we have a specific purpose for the script, and we have a set of expected results. We will also hit on some tips, tricks, and, of course, the gotchas in solving a challenge one way as opposed to another to get the same result. All techniques are not created equal.

Shell scripts and functions are both *interpreted*. This means they are not compiled. Both shell scripts and functions are ASCII text that is read by the shell command interpreter. When we execute a shell script, or function, a command interpreter goes through the ASCII text line-by-line, loop-by-loop, test-by-test, and executes each statement as each line is reached from the top to the bottom.

Functions

A function is written in much the same way as a shell script but is different in that it is defined, or written, within a shell script most of the time, and is called within the

script. This way we can write a piece of code, which is used over and over, just once and use it without having to rewrite the code every time. We just call the function instead.

We can also define functions at the system level that is always available in our environment, but this is a topic for later discussion.

A function has the following form:

```
function function_name
{

     commands to execute

}
```

or

```
function_name ()
{

     commands to execute

}
```

When we write functions into our scripts we must remember to declare, or write, the function *before* we use it. The function must appear above the command statement calling the function. We can't use something that does not yet exist.

Running a Shell Script

A shell script can be executed in the following ways:

```
ksh  shell_script_name
```

will create a Korn shell and execute the `shell_script_name` in the newly created Korn shell environment. The same is true for `sh` and `Bash` shells.

```
shell_script_name
```

will execute `shell_script_name` *if the execution bit is set on the file* (see the manual page on the `chmod` command, **man chmod**). The script will execute in the shell that is *declared* on the first line of the shell script. If no shell is declared on the first line of the shell script, it will execute in the default shell, which is the user's system-defined shell. Executing in an unintended shell may result in a failure and give unpredictable results.

Table 1-2 Different Types of Shells to Declare

COMMAND	DESCRIPTION
`#!/bin/sh` or `#!/usr/bin/sh`	Declares a Bourne shell
`#!/bin/ksh` or `#!/usr/bin/ksh`	Declares a Korn shell
`#!/bin/csh` or `#!/usr/bin/csh`	Declares a C shell
`#!/bin/Bash` or `#!/usr/bin/Bash`	Declares a Bourne-Again (Bash) shell

Declare the Shell in the Shell Script

Declare the shell! If we want to have complete control over how a shell script is going to run and in which shell it is to execute, we must *declare* the shell in *the first line of the script*. If no shell is declared, the script will execute in the default shell, defined by the system for the user executing the shell script. If the script was written, for example, to execute in Bash shell, `Bash`, and the default shell for the user executing the shell script is the C shell, `csh`, the script will most likely have a failure during execution. To declare a shell, one of the declaration statements in Table 1-2 must appear on the *first line* of the shell script.

Comments and Style in Shell Scripts

Making good comments in our scripts is stressed throughout this book. What is intuitively obvious to us may be total Greek to others who follow in our footsteps. We have to write code that is readable and has an easy flow. This involves writing a script that is easy to read and easy to maintain, which means that it must have plenty of comments describing the steps. For the most part, the person who writes the shell script is not the one who has to maintain it. There is nothing worse than having to hack through someone else's code that has no comments to find out what each step is supposed to do. It can be tough enough to modify the script in the first place, but having to figure out the mindset of the author of the script will sometimes make us think about rewriting the entire shell script from scratch. We can avoid this by writing a clearly readable script and inserting plenty of comments describing what our philosophy is and how we are using the input, output, variables, and files.

For good style in our command statements, we need it to be readable. For this reason it is sometimes better, for instance, to separate a command statement onto three separate lines instead of stringing, or *piping*, everything together on the same line of code; it may be just too difficult to follow the pipe and understand what the expected result should be for a new script writer. However, in some cases it is more desirable to create a long pipe. But, again, it should have comments describing our thinking step by step. This way someone later will look at our code and say, ''Hey, now that's a groovy way to do that.''

Command readability and step-by-step comments are just the very basics of a well-written script. Using a lot of comments will make our life much easier when we have to come back to the code after not looking at it for six months, and believe me; we will look at the code again. Comment everything! This includes, but is not limited to, describing what our variables and files are used for, describing what loops are doing, describing each test, maybe including expected results and how we are manipulating the data and the many data fields. A hash mark, `#`, precedes each line of a comment.

The *script stub* that follows is on this book's companion web site at `www.wiley.com/go/michael2e`. The name is `script.stub`. It has all the comments ready to get started writing a shell script. The `script.stub` file can be copied to a new filename. Edit the new filename, and start writing code. The `script.stub` file is shown in Listing 1-1.

```
#!/bin/Bash
#
# SCRIPT: NAME_of_SCRIPT
# AUTHOR: AUTHORS_NAME
# DATE:   DATE_of_CREATION
# REV:    1.1.A (Valid are A, B, D, T and P)
#               (For Alpha, Beta, Dev, Test and Production)
#
# PLATFORM: (SPECIFY: AIX, HP-UX, Linux, OpenBSD, Solaris
#                      or Not platform dependent)
#
# PURPOSE: Give a clear, and if necessary, long, description of the
#          purpose of the shell script. This will also help you stay
#          focused on the task at hand.
#
# REV LIST:
#        DATE: DATE_of_REVISION
#        BY:   AUTHOR_of_MODIFICATION
#        MODIFICATION: Describe what was modified, new features, etc--
#
#
# set -n   # Uncomment to check script syntax, without execution.
#          # NOTE: Do not forget to put the comment back in or
#          #       the shell script will not execute!
# set -x   # Uncomment to debug this shell script
#
##########################################################
#         DEFINE FILES AND VARIABLES HERE
##########################################################
```

Listing 1-1 script.stub shell script starter listing

```
############################################################
#               DEFINE FUNCTIONS HERE
############################################################



############################################################
#                BEGINNING OF MAIN
############################################################



# End of script
```

Listing 1-1 (*continued*)

The shell script starter shown in Listing 1-1 gives you the framework to start writing the shell script with sections to declare variables and files, create functions, and write the final section, `BEGINNING OF MAIN`, where the main body of the shell script is written.

Control Structures

The following control structures will be used extensively.

if . . . then statement

```
if [ test_command ]
then

        commands

fi
```

if . . . then . . . else statement

```
if [ test_command ]
then

     commands

else

     commands

fi
```