



# Mastering UNIX<sup>®</sup> Shell Scripting

---

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

**Second Edition**

Randal K. Michael



WILEY

Wiley Publishing, Inc.



**Mastering UNIX<sup>®</sup> Shell  
Scripting  
Second Edition**





# **Mastering UNIX<sup>®</sup> Shell Scripting**

---

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

**Second Edition**

Randal K. Michael



WILEY

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](http://www.wiley.com)

Copyright © 2008 by Randal K. Michael

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

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, or online at <http://www.wiley.com/go/permissions>.

**Limit of Liability/Disclaimer of Warranty:** The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Website is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Website may provide or recommendations it may make. Further, readers should be aware that Internet Websites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (800) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Library of Congress Cataloging-in-Publication Data is available from publisher.

**Trademarks:** Wiley, the Wiley logo, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. UNIX is a registered trademark of The Open Group. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

*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

<b>Acknowledgments</b>	<b>xxv</b>
<b>Introduction</b>	<b>xxvii</b>
<b>Part One The Basics of Shell Scripting</b>	
<b>Chapter 1 Scripting Quick Start and Review</b>	<b>3</b>
Case Sensitivity	3
UNIX Special Characters	3
Shells	4
Shell Scripts	4
Functions	4
Running a Shell Script	5
Declare the Shell in the Shell Script	6
Comments and Style in Shell Scripts	6
Control Structures	8
if ... then statement	8
if ... then ... else statement	8
if ... then ... elif ... (else) statement	9
for ... in statement	9
while statement	9
until statement	9
case statement	10
Using break, continue, exit, and return	10
Here Document	11
Shell Script Commands	12
Symbol Commands	14
Variables	15
Command-Line Arguments	15
shift Command	16
Special Parameters \$* and \$@	17
Special Parameter Definitions	17
Double Quotes, Forward Tics, and Back Tics	18

Using awk on Solaris	19
Using the echo Command Correctly	19
Math in a Shell Script	20
Operators	20
Built-In Mathematical Functions	21
File Permissions, suid and sgid Programs	21
chmod Command Syntax for Each Purpose	22
To Make a Script Executable	22
To Set a Program to Always Execute as the Owner	23
To Set a Program to Always Execute as a Member of the File Owner's Group	23
To Set a Program to Always Execute as Both the File Owner and the File Owner's Group	23
Running Commands on a Remote Host	23
Setting Traps	25
User-Information Commands	25
who Command	26
w Command	26
last Command	26
ps Command	27
Communicating with Users	27
Uppercase or Lowercase Text for Easy Testing	28
Check the Return Code	29
Time-Based Script Execution	30
Cron Tables	30
Cron Table Entry Syntax	31
at Command	31
Output Control	32
Silent Running	32
Using getopts to Parse Command-Line Arguments	33
Making a Co-Process with Background Function	34
Catching a Delayed Command Output	36
Fastest Ways to Process a File Line-by-Line	37
Using Command Output in a Loop	40
Mail Notification Techniques	41
Using the mail and mailx Commands	41
Using the sendmail Command to Send Outbound Mail	41
Creating a Progress Indicator	43
A Series of Dots	43
A Rotating Line	43
Elapsed Time	44
Working with Record Files	45
Working with Strings	46
Creating a Pseudo-Random Number	47
Using /dev/random and /dev/urandom	48
Checking for Stale Disk Partitions in AIX	48

Automated Host Pinging	49	
Highlighting Specific Text in a File	49	
Keeping the Printers Printing	50	
AIX “Classic” Printer Subsystem	50	
System V and CUPS Printing	50	
Automated FTP File Transfer	51	
Using rsync to Replicate Data	51	
Simple Generic rsync Shell Script	52	
Capturing a List of Files Larger than \$MEG	53	
Capturing a User’s Keystrokes	53	
Using the bc Utility for Floating-Point Math	54	
Number Base Conversions	55	
Using the typeset Command	55	
Using the printf Command	55	
Create a Menu with the select Command	56	
Removing Repeated Lines in a File	58	
Removing Blank Lines from a File	58	
Testing for a Null Variable	58	
Directly Access the Value of the Last Positional Parameter, \$#	59	
Remove the Column Headings in a Command Output	59	
Arrays	60	
Loading an Array	60	
Testing a String	61	
Summary	65	
<b>Chapter 2</b>	<b>24 Ways to Process a File Line-by-Line</b>	<b>67</b>
Command Syntax		67
Using File Descriptors		68
Creating a Large File to Use in the Timing Test		68
24 Methods to Parse a File Line-by-Line		73
Method 1: cat_while_read_LINE		74
Method 2: while_read_LINE_bottom		75
Method 3: cat_while_LINE_line		76
Method 4: while_LINE_line_bottom		77
Method 5: cat_while_LINE_line_cmdsub2		78
Method 6: while_LINE_line_bottom_cmdsub2		79
Method 7: for_LINE_cat_FILE		79
Method 8: for_LINE_cat_FILE_cmdsub2		80
Method 9: while_line_outfile		81
Method 10: while_read_LINE_FD_IN		81
Method 11: cat_while_read_LINE_FD_OUT		83
Method 12: while_read_LINE_bottom_FD_OUT		85
Method 13: while_LINE_line_bottom_FD_OUT		86
Method 14: while_LINE_line_bottom_cmdsub2_FD_OUT		87
Method 15: for_LINE_cat_FILE_FD_OUT		87
Method 16: for_LINE_cat_FILE_cmdsub2_FD_OUT		88
Method 17: while_line_outfile_FD_IN		89

	Method 18: while_line_outfile_FD_OUT	90
	Method 19: while_line_outfile_FD_IN_AND_OUT	91
	Method 20: while_LINE_line_FD_IN	92
	Method 21: while_LINE_line_cmdsub2_FD_IN	93
	Method 22: while_read_LINE_FD_IN_AND_OUT	94
	Method 23: while_LINE_line_FD_IN_AND_OUT	96
	Method 24: while_LINE_line_cmdsub2_FD_IN_AND_OUT	97
	Timing Each Method	98
	Timing Script	99
	Timing Data for Each Method	117
	Timing Command-Substitution Methods	127
	What about Using Command Input Instead of File Input?	128
	Summary	129
	Lab Assignments	129
<b>Chapter 3</b>	<b>Automated Event Notification</b>	<b>131</b>
	Basics of Automating Event Notification	131
	Using the mail and mailx Commands	132
	Setting Up a sendmail Alias	134
	Problems with Outbound Mail	134
	Creating a “Bounce” Account with a .forward File	136
	Using the sendmail Command to Send Outbound Mail	137
	Dial-Out Modem Software	139
	SNMP Traps	139
	Summary	140
	Lab Assignments	141
<b>Chapter 4</b>	<b>Progress Indicators Using a Series of Dots, a Rotating Line, or Elapsed Time</b>	<b>143</b>
	Indicating Progress with a Series of Dots	143
	Indicating Progress with a Rotating Line	145
	Indicating Progress with Elapsed Time	148
	Combining Feedback Methods	151
	Other Options to Consider	153
	Summary	153
	Lab Assignments	154
<b>Part Two</b>	<b>Scripts for Programmers, Testers, and Analysts</b>	
<b>Chapter 5</b>	<b>Working with Record Files</b>	<b>157</b>
	What Is a Record File?	157
	Fixed-Length Record Files	158
	Variable-Length Record Files	159
	Processing the Record Files	160
	Tasks for Records and Record Files	164
	Tasks on Fixed-Length Record Files	164
	Tasks on Variable-Length Record Files	166
	The Merge Process	169



Working with Strings	171
Putting It All Together	173
Other Things to Consider	183
Summary	184
Lab Assignments	184
<b>Chapter 6 Automated FTP Stuff</b>	<b>187</b>
Syntax	187
Automating File Transfers and Remote Directory Listings	190
Using FTP for Directory Listings on a Remote Machine	190
Getting One or More Files from a Remote System	192
Pre and Post Events	195
Script in Action	196
Uploading One or More Files to a Remote System	196
Replacing Hard-Coded Passwords with Variables	199
Example of Detecting Variables in a Script's Environment	200
Modifying Our FTP Scripts to Use Password Variables	203
What about Encryption?	209
Creating Encryption Keys	210
Setting Up No-Password Secure Shell Access	210
Secure FTP and Secure Copy Syntax	211
Automating FTP with autoexpect and expect Scripts	212
Other Things to Consider	217
Use Command-Line Switches to Control Execution	217
Keep a Log of Activity	217
Add a Debug Mode to the Scripts	217
Reading a Password into a Shell Script	217
Summary	218
Lab Assignments	218
<b>Chapter 7 Using rsync to Efficiently Replicate Data</b>	<b>219</b>
Syntax	219
Generic rsync Shell Script	220
Replicating Multiple Directories with rsync	222
Replicating Multiple Filesystems with rsync	237
Replicating an Oracle Database with rsync	251
Filesystem Structures	252
rsync Copy Shell Script	254
Summary	289
Lab Assignments	289
<b>Chapter 8 Automating Interactive Programs with Expect and Autoexpect</b>	<b>291</b>
Downloading and Installing Expect	291
The Basics of Talking to an Interactive Script or Program	293
Using autoexpect to Automatically Create an Expect Script	296
Working with Variables	304
What about Conditional Tests?	306

	Expect's Version of a case Statement	306
	Expect's Version of an if...then...else Loop	313
	Expect's Version of a while Loop	314
	Expect's Version of a for Loop	315
	Expect's Version of a Function	317
	Using Expect Scripts with Sun Blade Chassis and JumpStart	318
	Summary	323
	Lab Assignments	324
<b>Chapter 9</b>	<b>Finding Large Files and Files of a Specific Type</b>	<b>325</b>
	Syntax	326
	Remember That File and Directory Permissions Thing	327
	Don't Be Shocked by the Size of the Files	327
	Creating the Script	327
	Narrowing Down the Search	333
	Other Options to Consider	333
	Summary	334
	Lab Assignments	334
<b>Chapter 10</b>	<b>Process Monitoring and Enabling Pre-Processing, Startup, and Post-Processing Events</b>	<b>335</b>
	Syntax	336
	Monitoring for a Process to Start	336
	Monitoring for a Process to End	338
	Monitor and Log as a Process Starts and Stops	342
	Timed Execution for Process Monitoring, Showing Each PID, and Timestamp with Event and Timing Capability	347
	Other Options to Consider	367
	Common Uses	367
	Modifications to Consider	367
	Summary	367
	Lab Assignments	368
<b>Chapter 11</b>	<b>Pseudo-Random Number and Data Generation</b>	<b>369</b>
	What Makes a Random Number?	369
	The Methods	370
	Method 1: Creating a Pseudo-Random Number Utilizing the PID and the RANDOM Shell Variable	371
	Method 2: Creating Numbers between 0 and 32,767	371
	Method 3: Creating Numbers between 1 and a User-Defined Maximum	372
	Method 4: Creating Fixed-Length Numbers between 1 and a User-Defined Maximum	373
	Why Pad the Number with Zeros the Hard Way?	375
	Method 5: Using the /dev/random and /dev/urandom Character Special Files	376
	Shell Script to Create Pseudo-Random Numbers	379
	Creating Unique Filenames	384

Creating a File Filled with Random Characters	392
Other Things to Consider	399
Summary	399
Lab Assignments	400
<b>Chapter 12 Creating Pseudo-Random Passwords</b>	<b>401</b>
Randomness	401
Creating Pseudo-Random Passwords	402
Syntax	403
Arrays	403
Loading an Array	403
Building the Password-Creation Script	405
Order of Appearance	405
Define Functions	406
Testing and Parsing Command-Line Arguments	414
Beginning of Main	418
Setting a Trap	418
Checking for the Keyboard File	419
Loading the KEYS Array	419
Building a New Pseudo-Random Password	420
Printing the Manager’s Password Report for Safekeeping	421
Other Options to Consider	431
Password Reports?	432
Which Password?	432
Other Uses?	432
Summary	432
Lab Assignments	432
<b>Chapter 13 Floating-Point Math and the bc Utility</b>	<b>433</b>
Syntax	433
Creating Some Shell Scripts Using bc	434
Creating the float_add.ksh Shell Script	434
Testing for Integers and Floating-Point Numbers	440
Building a Math Statement for the bc Command	441
Using a Here Document	442
Creating the float_subtract.ksh Shell Script	443
Using getopt to Parse the Command Line	449
Building a Math Statement String for bc	450
Here Document and Presenting the Result	451
Creating the float_multiply.ksh Shell Script	452
Parsing the Command Line for Valid Numbers	458
Creating the float_divide.ksh Shell Script	460
Creating the float_average.ksh Shell Script	467
Other Options to Consider	472
Creating More Functions	472
Summary	473
Lab Assignments	473

<b>Chapter 14</b>	<b>Number Base Conversions</b>	<b>475</b>
	Syntax	475
	Example 1: Converting from Base 10 to Base 16	476
	Example 2: Converting from Base 8 to Base 16	476
	Example 3: Converting Base 10 to Octal	477
	Example 4: Converting Base 10 to Hexadecimal	477
	Scripting the Solution	477
	Base 2 (Binary) to Base 16 (Hexadecimal) Shell Script	478
	Base 10 (Decimal) to Base 16 (Hexadecimal) Shell Script	481
	Script to Create a Software Key Based on the Hexadecimal Representation of an IP Address	485
	Script to Translate between Any Number Base	490
	Using getopt to Parse the Command Line	495
	Example 5: Correct Usage of the <code>equate_any_base.ksh</code> Shell Script	495
	Example 6: Incorrect Usage of the <code>equate_any_base.ksh</code> Shell Script	495
	Continuing with the Script	497
	Beginning of Main	498
	An Easy, Interactive Script to Convert Between Bases	500
	Using the <code>bc</code> Utility for Number Base Conversions	506
	Other Options to Consider	512
	Software Key Shell Script	512
	Summary	512
	Lab Assignments	513
<b>Chapter 15</b>	<b>hgrep: Highlighted grep Script</b>	<b>515</b>
	Reverse Video Control	516
	Building the <code>hgrep.Bash</code> Shell Script	517
	Other Options to Consider	524
	Other Options for the <code>tput</code> Command	524
	Summary	525
	Lab Assignments	525
<b>Chapter 16</b>	<b>Monitoring Processes and Applications</b>	<b>527</b>
	Monitoring Local Processes	527
	Remote Monitoring with Secure Shell and Remote Shell	530
	Checking for Active Oracle Databases	536
	Using <code>autoexpect</code> to Create an <code>expect</code> Script	539
	Checking if the HTTP Server/Application Is Working	545
	What about Waiting for Something to Complete Executing?	546
	Other Things to Consider	547
	Proper <code>echo</code> Usage	548
	Application APIs and SNMP Traps	548
	Summary	548
	Lab Assignments	549

**Part Three    Scripts for Systems Administrators**

<b>Chapter 17</b>	<b>Filesystem Monitoring</b>	<b>553</b>
	Syntax	553
	Adding Exceptions Capability to Monitoring	559
	The Exceptions File	559
	Using the MB-of-Free-Space Method	565
	Using MB of Free Space with Exceptions	568
	Percentage Used — MB Free and Large Filesystems	573
	Running Filesystem Scripts on AIX, Linux, HP-UX, OpenBSD, and Solaris	583
	Command Syntax and Output Varies between Operating Systems	585
	Programming a Shell-Neutral Script	590
	Other Options to Consider	600
	Event Notification	600
	Automated Execution	600
	Modify the egrep Statement	601
	Summary	601
	Lab Assignments	602
 <b>Chapter 18</b>	 <b>Monitoring Paging and Swap Space</b>	 <b>603</b>
	Syntax	604
	AIX lsps Command	604
	HP-UX swapinfo Command	605
	Linux free Command	606
	OpenBSD swapctl Command	606
	Solaris swap Command	607
	Creating the Shell Scripts	607
	AIX Paging Monitor	607
	HP-UX Swap-Space Monitor	613
	Linux Swap-Space Monitor	618
	OpenBSD Swap-Space Monitor	622
	Solaris Swap-Space Monitor	625
	All-in-One Paging- and Swap-Space Monitor	630
	Other Options to Consider	638
	Event Notification	638
	Log File	638
	Scheduled Monitoring	638
	Summary	638
	Lab Assignments	639
 <b>Chapter 19</b>	 <b>Monitoring System Load</b>	 <b>641</b>
	Installing the System-Statistics Programs in Linux	642
	Syntax	644
	Syntax for uptime	644

Linux	645
What's the Common Denominator?	645
Syntax for iostat	645
AIX	646
HP-UX	646
Linux	647
OpenBSD	647
Solaris	647
What Is the Common Denominator?	648
Syntax for sar	649
AIX	649
HP-UX	649
Linux	650
Solaris	650
What Is the Common Denominator?	650
Syntax for vmstat	651
AIX	652
HP-UX	652
Linux	652
OpenBSD	652
Solaris	653
What Is the Common Denominator?	653
Scripting the Solutions	654
Using uptime to Measure the System Load	655
Scripting with the uptime Command	655
Using sar to Measure the System Load	659
Scripting with the sar Command	660
Using iostat to Measure the System Load	665
Scripting with the iostat Command	665
Using vmstat to Measure the System Load	670
Scripting with the vmstat Command	670
Other Options to Consider	674
Try to Detect Any Possible Problems for the User	674
Show the User the Top CPU Hogs	675
Gathering a Large Amount of Data for Plotting	675
Summary	675
Lab Assignments	675
<b>Chapter 20 Monitoring for Stale Disk Partitions (AIX-Specific)</b>	<b>677</b>
AIX Logical Volume Manager (LVM)	677
The Commands and Methods	678
Disk Subsystem Commands	678
Method 1: Monitoring for Stale PPs at the LV Level	679
Method 2: Monitoring for Stale PPs at the PV Level	684
Method 3: VG, LV, and PV Monitoring with a resync	687
Other Options to Consider	694
SSA Disks	694

Log Files	695
Automated Execution	695
Event Notification	695
Summary	696
Lab Assignment	696
<b>Chapter 21   Turning On/Off SSA Identification Lights</b>	<b>697</b>
Syntax	698
Translating an hdisk to a pdisk	698
Identifying an SSA Disk	698
The Scripting Process	698
Usage and User Feedback Functions	699
Control Functions	703
The Full Shell Script	709
Other Things to Consider	721
Error Log	721
Cross-Reference	721
Root Access and sudo	721
Summary	721
Lab Assignment	722
<b>Chapter 22   Automated Hosts Pinging with Notification of Failure</b>	<b>723</b>
Syntax	723
Creating the Shell Script	725
Define the Variables	725
Creating a Trap	728
The Whole Shell Script	728
Other Options to Consider	736
\$PINGLIST Variable-Length-Limit Problem	736
Ping the /etc/hosts File Instead of a List File	737
Logging	737
Notification of “Unknown Host”	738
Notification Method	738
Automated Execution Using a Cron Table Entry	739
Summary	739
Lab Assignments	739
<b>Chapter 23   Creating a System-Configuration Snapshot</b>	<b>741</b>
Syntax	742
Creating the Shell Script	744
Other Options to Consider	774
Summary	774
Lab Assignment	775
<b>Chapter 24   Compiling, Installing, Configuring, and Using sudo</b>	<b>777</b>
The Need for sudo	777
Configuring sudo on Solaris	778
Downloading and Compiling sudo	778

Compiling sudo	779
Configuring sudo	790
Using sudo	797
Using sudo in a Shell Script	798
Logging to the syslog with sudo	801
The sudo Log File	806
Summary	806
Lab Assignments	807
<b>Chapter 25 Print-Queue Hell: Keeping the Printers Printing</b>	<b>809</b>
System V versus BSD versus CUPS Printer Systems	809
AIX Print-Control Commands	810
Classic AIX Printer Subsystem	810
System V Printing on AIX	814
More System V Printer Commands	818
CUPS — Common UNIX Printing System	820
HP-UX Print-Control Commands	823
Linux Print-Control Commands	825
Controlling Queuing and Printing Individually	831
Solaris Print-Control Commands	833
More System V Printer Commands	837
Putting It All Together	839
Other Options to Consider	849
Logging	849
Exceptions Capability	849
Maintenance	849
Scheduling	849
Summary	850
Lab Assignments	850
<b>Chapter 26 Those Pesky Sarbanes-Oxley (SOX) Audits</b>	<b>851</b>
What to Expect	852
How to Work with the Auditors	852
What the Auditors Want to See	853
Some Handy Commands	854
Using the id Command	854
Using the find Command	855
Using the awk and cut Commands	856
Using the sed Command	862
Using the dirname and basename Commands	863
Other Things to Consider	864
Summary	864
Lab Assignments	865
<b>Chapter 27 Using Dirvish with rsync to Create Snapshot-Type Backups</b>	<b>867</b>
How Does Dirvish Work?	868
How Much Disk Storage Will I Need?	868
Configuring Dirvish	868



Installing Dirvish	869
Modifying the master.conf Dirvish Configuration File	872
Creating the default.conf File for Each Filesystem Backup	873
Performing a Full System Backup	874
Using Dirvish on the Command Line	875
A Menu-Interface Shell Script to Control Dirvish	876
Running All Backups	878
Running a Particular Backup	879
Locating and Restoring Images	880
Expiring and Deleting Backup Images	881
Using sed to Modify the summary File	883
Adding a New Backup	884
Removing a Backup	889
Managing the Dirvish Backup Banks	890
Adding a New Dirvish Backup Bank	891
Deleting a Dirvish Backup Bank	892
Putting It All Together	893
Using the dirvish_ctrl Shell Script	918
Running All Backups Defined in the Runall: Stanza	918
Running One Particular Backup	919
Locating and Restoring Files	919
Deleting Expired Backups and Expiring Backups	921
Adding a New Dirvish Backup Vault	925
Removing a Dirvish Vault	930
Managing Dirvish Backup Banks	930
Adding a New Dirvish Backup Bank	931
Removing a Dirvish Backup Bank	932
Other Things to Consider	932
Summary	933
Lab Assignments	933
<b>Chapter 28 Monitoring and Auditing User Keystrokes</b>	<b>935</b>
Syntax	936
Scripting the Solution	937
Logging User Activity	937
Starting the Monitoring Session	939
Where Is the Repository?	939
The Scripts	940
Logging root Activity	942
Some sudo Stuff	946
Monitoring Other Administration Users	948
Other Options to Consider	951
Emailing the Audit Logs	951
Compression	952
Need Better Security?	953
Inform the Users	953
Sudoers File	953

Summary	953
Lab Assignments	954
A Closing Note from the Author	954
<b>Appendix A What's on the Web Site</b>	<b>955</b>
Shell Scripts	955
Functions	966
<b>Index</b>	<b>977</b>



# 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 Sleva 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](http://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 ANSI 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](http://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**

# **The Basics of Shell Scripting**

Chapter 1: Scripting Quick Start and Review

Chapter 2: 24 Ways to Process a File Line-by-Line

Chapter 3: Automated Event Notification

Chapter 4: Progress Indicators Using a Series of Dots,  
a Rotating Line, or Elapsed Time



# 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	<code>/bin/sh</code> <b>OR</b> <code>/usr/bin/sh</code>
Bash	<code>/bin/Bash</code> <b>OR</b> <code>/usr/bin/Bash</code>
Korn	<code>/bin/ksh</code> <b>OR</b> <code>/usr/bin/ksh</code>

---

## 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
<code>#!/bin/sh</code> or <code>#!/usr/bin/sh</code>	Declares a Bourne shell
<code>#!/bin/ksh</code> or <code>#!/usr/bin/ksh</code>	Declares a Korn shell
<code>#!/bin/csh</code> or <code>#!/usr/bin/csh</code>	Declares a C shell
<code>#!/bin/Bash</code> or <code>#!/usr/bin/Bash</code>	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](http://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
```