An Introduction to Shell Scripting

Published in the United Kingdom by:
Slash Etcetera Ltd
4 Highwater View
St Leonards-on-Sea
TN38 8EL
01424 855022
info@slashetcbooks.co.uk

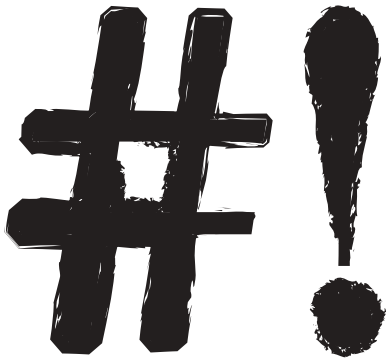
A catalogue record for this book is available from
the British Library.

ISBN 0 9544010 0 X

Printed by:
Antony Rowe Ltd
2 Whittle Drive
Highfield Industrial Estate
Eastbourne
BN23 6QH
www.antonyrowe.co.uk

An Introduction to Shell Scripting

A guide on how to write
Bourne and Korn shell scripts



Glen Smith

Contents

Introduction

Everybody has a different approach to learning a programming language. Many people, myself included, like to pick apart small, but whole example programs that can be run, changed and then rerun. The individual commands can be examined in detail by using command references like the Unix man pages.

Following this approach, this book describes the 57 commands and topics below. The scripts in the tutorial section show each command *in situ*, whilst the reference section covers the commands individually.

- &&
- ' '
- $0
- $1, $2 ... $9
- $#
- ${#VAR}
- $?
- $*
- $IFS
- $RANDOM
- $SECONDS
- $( )
- (( )), $(( ))
- >/dev/null
- #!
- Arrays
- backslash
- bc
- break
- cal
- case
- cat
- Comments
- continue
- cut
- date
- Double-quotes
- echo
- eval
- exit
- exit-code
- expr
- false
- for
- getopts
- grep
- if
- nl
- pipe
- read
- Run-quotes
- sed
- set
- Shell functions
- shift
- sleep
- stty
- test
- tput
- tr
- true
- typeset
- unset
- Variables
- wc
- while
- whitespace

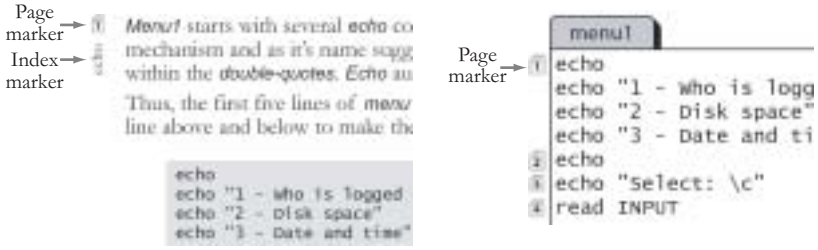
The scripts in the book are not geared towards any particular section of the Unix community. There are no examples specific to system administrators or web administrators; the scripts are designed to be as generic as possible.

The reason for this is that, in my experience, most people who want to learn how to shell script already know the commands to do their jobs. They are hoping to write some shell scripts in order to make their jobs easier.

Shell scripts are used to combine commands. More often than not, they are used to take the output from a command, extract some relevant piece of information and run a secondary command on this data.

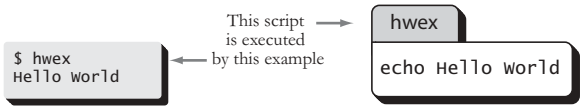
A good example of this is a backup script. First, the data requiring backing up is identified (e.g. df). The filesystems are then extracted from this list and a backup command ran against these filesystems. If the filesystem list changes, the script automatically adapts.

The tutorial section is divided into sixteen chapters, each containing a script of a page or less. In order to help the reader follow the descriptive text, the current script is displayed on every right-hand page and the lines of the script that are described on a given page are indicated by page markers. This is intended to help readers locate the text for a section of the script they are interested in. The index entries are also separated out, again to aid location.



The book makes extensive use of code illustrations. Any examples that start with a "$" can be entered at the command prompt (the initial "$" should not be entered). If the command extends over more than one line, the continuation prompt ">" is used. The "$" and ">" values are stored in the environment variables $PS1 & $PS2:

Small example scripts are shown as tabbed boxes:



Many of the commands used in shell scripting are also English words. In order to distinguish between the two, commands such as *if*, *then*, *while*, *for*, *do*, *done*, *test* etc are shown in *italics* in the text.

Getting Started

If you have never written a shell script before you may need the information on this page to get you started.

Shell scripts are only text files, but in order to run them they need to be flagged as executable. This is achieved by the *chmod* command. In the example below a script called “hwex” is created, is then make executable and is finally executed. The script prints the words “Hello World”.

Create hwex →
make it executable →
run hwex →

```
$ echo "echo Hello World" > hwex
$ chmod +x hwex
$ hwex
Hello World
```

In the example above, the text of the script was created using a *echo* statement. This is fine for one line scripts, but for anything else the only realistic option is to use a editor. The standard editor with Unix is vi.

To start a *vi* session simply enter *vi* and the filename you want to edit or create. If the file does not exist, there will be a “[New File]” message at the bottom of the screen. For an existing file, the contents will be displayed.

Vi starts in *command mode* which means that in order to enter any text, you first have to enable insert mode. This is achieved with the “a” command (append). To return to *command mode* use the “Esc” key.

To save the file use “:w”. When the colon is entered, it will appear at the bottom of the screen. To save and exit the file use “:wq” or “ZZ”. Remember to make the file executable before trying to run it.

The pages overleaf contain a list of useful *vi* commands.

One very useful, but underused, feature of *vi* is the ability to run a script without exiting. This is achieved via the “:!” construct which allows a shell command to be typed after the exclamation mark. Thus, to run the *hwex* command, enter “:!hwex” and press <Return>.

Basic Vi Commands

a	append text after the cursor	use ESC to terminate insert mode
i	insert text	use ESC to terminate insert mode
A	append to end of line	use ESC to terminate insert mode
o	open (insert) a line below the current one (insert mode is also turned on)	
O	open (insert) a line above the current one (insert mode is also turned on)	

x	delete	
dw	delete word	
dd	delete line	
D	delete to end of line	

r	replace one character	
cw	change a word	use ESC to terminate insert mode

w	move forward a word
b	move back a word
0	move to start of line
$	move to end of line
1G	move to first line
G	move to end of file

:w	save file
:wq	save and exit file
ZZ	save and exit file
:q!	exit without saving

Useful Vi Commands

u undo last action

~ change case of a character

Y Copy a line into the buffer
p Print contents of buffer after cursor
P Print contents of buffer before cursor

J Join lines

. Repeat last action

:!*cmd* Run *cmd*
:!% Run current file
:!chmod +x % Make current file executable

:r *file* read contents from *file* into current script
:*n* goto line *n*

Vi has extra ":set" modes that can make it easier to use.

:set ts=4 sw=4	set 4 space tabstops (default is 8)
:set showmode	shows what mode you are in (insert, append etc)
:set ic	case insensitive searches
:set ai	auto-indent (use ^D in insert mode to go back a tabstop)
:set noerrorbells	don't beep on error (e.g. pressing Esc when not needed)
:set noflash	don't flash the screen on error

Vi also reads the variable EXINIT and will apply any modes contained in it.

Commands by Chapter

Chapter One

- echo, echo "\c"
- read
- if-then-fi
- [x = y],[x != y]
- nested if statements
- else
- elif
- case-in-esac
- shell variables
- ;;
- case *)
- double-quotes

Chapter Two

- while loops (while true)
- tput clear
- echo "*"
- case [ ])
- case "")
- continue
- break
- exit
- sleep
- echo "\n"

Chapter Three

- $1, $2 ... $9
- if [-lt, -le, -eq, -ne, -gt, -ge, -ne]
- if [-o]
- case ¦)
- Comments - #
- $?
- cal
- pipe - ¦
- grep, grep "$"
- > /dev/null
- echo " \" "

Chapter Four

- run-quotes - ` `
- cut -d -f
- &&
- while read
- bc
- expr

Chapter Five

- Korn shell
- #!/bin/ksh
- Arrays
- tr "x" "y"
- ((*expression*)), $(( ))
- *cmd* ¦ read

Chapter Six

- set -A
- backslash - \
- typeset -Z*n*
- typeset -L*n*
- date "+*format*"
- sed -e "s/*x*/*y*/g"

Commands by Chapter

Chapter Seven

- case ?)

Chapter Eight

- [-z "$VAR"]
- tr -d "[0-9]"
- [-n "$VAR"]
- $( )

Chapter Nine

- Shell functions
- fatal( )
- $0
- $*

Chapter Ten

- sed -e "s/./& /g"
- whitespace compression

Chapter Eleven

- sed -e "s/\(\)/\1/"
- " `*cmd*` "

Chapter Twelve

- Processing passed parameters
- shift
- Single-quotes - ' '
- $IFS
- ${#VAR} (length)
- set -- *args*

Chapter Thirteen

- getopts :wcl VAR
- case \?)
- $OPTIND
- $OPTARG
- shift *n*

Chapter Fourteen

- tr '[a-z]' '[.*]'
- tput cup 0 0
- tr -c "[$VAR\n]" '[.*]'
- single-quotes
- eval
- echo "\b"
- echo "\r\c"
- tput el
- $SECONDS

Chapter Fifteen

- if [! -f "$WORDLIST"]
- grep -c
- grep "^"
- $RANDOM
- grep "^ *$NUM[ ]"

Chapter Sixteen

- trap
- tput smso, tput rmso
- stty -g
- stty -a
- stty -echo
- stty -icanon min 0 time 50

Scripts by Chapter

Chapter One

menu1	A simple menu
menu2	Like menu1 but uses *else* for efficiency
menu3	Like menu2 but using *elif*
menu4	Menu3 rewritten using a *case* statement

Chapter Two

fullmenu	A usable menu
yesno	Simple *case* statements used for requesting a Yes or No answer

Chapter Three

validdate	Checks the supplied date

Chapter Four

dow	Returns the Day of Week from a supplied date

Chapter Five

dow2	A Korn shell version of dow

Chapter Six

tomorrow	Returns tomorrow's date. Output format can be selected

Chapter Seven

numcvt	Converts any number under one thousand to English text

Chapter Eight

counter	Used to test numcvt with a series of values
counter2	Counter with limited validation for supplied values
counter3	Counter with all input validated

Chapter Nine

counter3f	Counter3 rewritten using shell functions
numcvt2	Numcvt with input validation

Scripts by Chapter

Chapter Ten

numcvt3 The final and best version of numcvt

Chapter Eleven

num2words Number to English converter that works over one thousand

Chapter Twelve

wc2 A simple shell version of the Unix command wc

Chapter Thirteen

wc3 A more featured version of wc2 using *getopts*

Chapter Fourteen

hangman Guess the word game

Chapter Fifteen

random_word Select a word from a file of words. Used by hangman

Chapter Sixteen

fancymenu Fullmenu from Chapter Two with enhancements
mkpwd Create an encrypted password for fancymenu

Glossary

ampersand
: The “&” character

asterisk
: The “*” character

backslash
: The “\” character. Forces a character to be treated as text, rather than shell code. Used to embed double-quotes within double-quotes.

builtin
: A sub-command of the shell, implemented within the shell rather than by calling an external program.

carriage-return
: ASCII character 11, moves the cursor to the left edge of the screen, but leaves the cursor on the same line.

command line
: The string used to invoke a program. The first word is the program name and all other words are arguments. Command line arguments translate to passed parameters within a program.

complement
: Exactly opposite; a mirror image (e.g. true and false)

double-quotes - “ ”

escape sequence
: A sequence of non-printing text that indicates a special condition to software or hardware; e.g. turn on underlining.

exit-code
: The final status of a program when it exited. Exit-code zero usually means success, anything else means failure.

indentation
: The use of left-hand spacing to layout code in a more readable format

iteration
: Once around a programming loop

Glossary

keyword
: A word that has special significance to the shell; e.g. *in*, *done*, *case*

metacharacters
: Special characters that might be replaced during command parsing; e.g. *.

nested
: One inside another.

newline
: ASCII character 10, moves the current position down one line.

parse
: Interpretation by a program.

passed parameters
: Information specified on a command line that modifies the action of a program. Referenced within a script by $1, $2 etc.

<Return>
: The Return key on the keyboard

single-quotes - ‘ ’

super shells
: High functionality shells like Korn and Bash.

syntax
: Strict rules relating to text layouts and interpretation

whitespace
: Space, tab and (often) newlines. Sometimes ignored, sometimes compressed - see the reference entry