Review Chapter 6 and 7: Shell Scripting

The point of server-side scripting is to write out tasks that a system administrator must perform over and over again so that the script can be run without the need of a person. Basically, a good script automates away an IT administrator's job so that they can spend there time doing better things. The script makes you more efficient, capable, and clear-headed. As a bonus, it also serves as an excellent way to self-document administration procedures so that you are not relying on the memory of a single person to accomplish a task.

There are many shells available:

Name:	**Command:**	**Description:**
Bourne Shell	sh	The predecessor of Bash
Korn Shell	ksh	A contemporary of Bash
Bourne-Again Shell	bash	The standard shell
C-shell	csh	A shell using C-like syntax
Powershell	powershell	A shell used on Windows that is now open-sourced and available on Linux, it separates itself from the pack by having a heavy reliance on processing using the .Net object-oriented data model. Especially good to know if you are in a Microsoft-based shop or a shop where both systems run in tandem

You can use these shells (if they are installed) to write scripts by typing:

#!/path/to/shell

at the top of your file. You would of course replace path to shell with the actual path to the shell. For instance,

#!/bin/bash

You can also write scripts that parse other programming languages in the same manner. The parser need not be an entire shell.

Some examples of parsers you could use would be

Name	Command	Description
Perl	#!/bin/perl	Sets the Perl parser as the script interpreter
Python	#!/bin/python	Sets Python to parse the script
PHP	#!/bin/php	Sets PHP to parse the script
Scala	#!/bin/scala	Sets Scala to parse the script
Node.js	#!/bin/node	Set Node.js to parse the script

These parsers are based around general-purpose programming languages that shine in their particular domains for which they were specifically built.

Perl – A language invented in 1987 to replace sed and awk so that server-side scripting could be done under a single unified language

Python – An object-oriented language invented in 1989 for fun and hobbyist programming designed to be aesthetically pleasing and easily readable by using white space to delimit blocks of code, has since overtaken Perl as the dominant server-side scripting language

PHP – A language specifically developed for server-side scripting the processing of requests on the internet and the web. Currently has the largest code base amongst web servers

Scala – A langauge that can be interpreted or compiled that was designed to replace Java. It is used heavily server side in the Big Data market alongside Python

Node.js – A parser built of the C library that allows JavaScript to run as a server-side scripting language. JavaScript was originally designed to run in browsers on the client side. The Node.js parser uses Google Chrome's V8 engine to parse and speaks to the server via C libraries. It has gained a foothold in the market because it is fast, achieving this by not blocking on I/O-bound processes

Also see: Rust, Julia, and Go

For chapters 6 and 7, we are going to focus on straight Bash scripting because it will likely be on any Unix-based system you encounter. If you know Bash, you will be capable of running the internet during a nuclear winter, the purpose for which it was designed.

Types of tokens in scripts:

Variables $HOME, $d

operators - =, *, |,+, (), {}, $

control structures - if, case, loop

Types of Variable:

Configuration Variables - store the configurations of the operating system

Environment Variables - store the configurations of your login shell

Shell Variables – store values for the duration of your login session (or the duration of a script which create a sub-shell when they run)

Useful Environment Variables:

HOME	- location of current user's HOME directory
LOGNAME	- holds the account name of the user currently logged in
PPID	- holds the id of the currently running processes
TZ	- holds timezone information
IFS	- specifies a delimiter for working in files
PATH	- holds all of the directories when searching for a command to run
PS1	- holds what is printed at the shell prompt
BASH	- contains the absolute path to the bash shell
BASH_VERSION	- holds information about the version of bash
ENV	- contains the filename containing the script that initializes the shell
EXINIT	- contains the initialization commands for the Vi editor
FUNCNAME	- contains the name of the function that is running
GROUPS	- contains the groups to which the current user belongs
HISTFILE	- identifies the file in which history commands are stored
HOSTFILE	- identifies the file in which networking information is stored
OSTYPE	- contains information about what OS the shell is running on
PWD	- contains the path of the present working directory
RANDOM	- yields a random integer
TERM	- contains the name of the terminal type in use by bash
UID	- contains the user identification number of the currently logged in user

Types of Operators

Defining operators – assigns a value to a variable (=)

Evaluating operators – expands a variable to its value ($)

Arithmetic and relational operators – for performing mathematical operations (-,+,!,~,*,/,%,>,<,=,!=)

Redirection operators – for directing streams to certain programs or files (<,>,2>,>>,|)

Extras

Read a value into a variable from the command line:
read varName

Refer to arguments passed to your script via the command line, with zero being your program name
$0, $1, $2, $3 …

Debugging

Trap

Exporting variables:

EXPORT VARNAME

Assigning and outputing variables:

Expand a variable to display its contents: $varName	 $PATH
Print the contents of a variable to stdout: echo $varName	 echo $HOME
Assign text to a variable Assign to a variable with spaces: Assign to a variable with variables: Assign to a variable supressing expansion:	d=hello d=”hello there friend” username=thor d=”hello there $username” username=loki d=’hello there $username’
Assign the results of standard output to a variable: varName=`command` OR varName=$(command)	 d=`date` d=$(date)

Create an empty array
 	
Initialize array`${arr[2]}`Retrieve third element
 	
Retrieve all elements
 	
Retrieve array indices
 	
Calculate array size
 	
Overwrite 1st element
 	
Append value(s)
 	
Save `ls` output as a string
 	
Save **`ls`** output as an array of files
 	
Retrieve n elements **`starting at index s`**
	`Arr=()`
	
	`arr=(1 2 3)`
	
	`${arr[@]}`
	
	`${!arr[@]}`
	
	`${#arr[@]}`
	
	`arr[0]=3`
	
	`arr+=(4)`
	
	`str=$(ls)`
	
	`arr=( $(ls) )`
	
	`${arr[@]:s:n}`

Store the result of the arithmetic operation in n	let n=a * 2 * 3 + 1

Redirecting

Send contents of file to stdin	grep < file.txt “bin/bash”
Send stdout of a command to a file	cat file.txt > newfile.txt
Append stdout to a file	cat indgredients2.txt >> ingredients.txt
Redirect stdout of command to stdin of next command	cat ingredients.txt | grep “bin/bash”
Direct stderr to a file	rmdir filledDir 2> errors.txt
Convert stdout stream to a temporary pipe (to send streams to a program that usually accepts files)	paste <(pwd) <(pwd)
execute the contents of a file or any stdout stream (convert any file to a stream for a program that usually accepts streams)	cat file.txt | bash
Store contents of file in a variable	f=$(<op.txt)
Execute contents of a variable	echo $myVar | bash
Store contents of variable to a file	echo $myVar > file.txt

Above are various ways of converting between files, variables, and streams. Anything that is a file can become a variable or stream. Anything that is a variable can become a stream or a file. And anything that is a stream can become file or a variable.

Program Control Flow

If Statements:

```
if [  $1 -eq $3 ]; then
echo "argument 1 is equal to argument 3"
else
echo "argument 1 is not equal argument 3"
fi
```

Loops:

```
 for i in "${arr[@]}"
 do
 echo "$i"
 done
```

Define a function:

```
f()
{
pwd
ls
echo “we’re finished!”
}
```

To call the function:

```
f
```

Case statement:

```
case “$1” in
1) echo “We made choice 1!”
;;
2) echo “We made choice 2!”
;;
3) echo “We made choice 3!”
;;
*)echo “We didn’t understand your choice.”
;;
esac
```