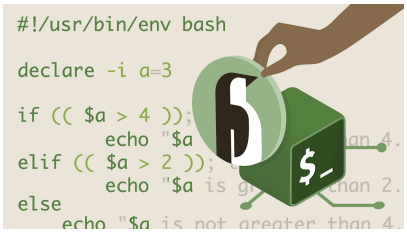
Learning Bash Scripting



Bash, or the Bourne Again Shell, is a widely popular command-line interpreter for administration and programming tasks. But Bash is different than most scripting languages. That's why Scott Simpson spends some time in this course running you through the syntax—introducing variables, numbers, and control structures—so you can start writing scripts right away. He shows you how to wrap up multiline operations in one file, implement flow control, and interact with users to get input. Plus, he offers challenges along the way that allow you to put what you've learned to the test.

Chapter: 1. Using Bash

Video: What's Bash? (1 Notes)

Bash stands for Bourne Again Shell (revision of previous Bourne shell)	0:07

Video: Pipes and redirections (2 Notes)

A variation of input redirection is called a here document and it lets us specify input freely up to a specified limits string, which can be useful for displaying long passages of text in a script or for providing input to an interactive command. A here document uses two less than symbols and a limit string to say, "Take all of this and feed it into the given command."	3:27
| Pipe; > Output redirection (truncate); >> Output redirection (append); < Input redirection; << Here document	4:08

Video: Bash builtins and other commands (9 Notes)

Bash has builtin commands that may be the same as OS commands.	0:17
A list of Bash builtin commands can be found at the man pages. https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#Shell-Builtin-Commands	0:27
The echo command includes a new line at the end (printf does not).	0:47
To run the command version, use "command [command]" https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#index-command	1:04
To run the builtin version, use "builtin [command]" https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#index-builtin	1:18
To view a description of a command, use command -V [command]. This will show whether something is a builtin shell command or a binary. Compare "command -V df" and "command -V echo". https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#index-command	1:40
But specific built-ins can be disabled in a session, with enable -n and the name of the built-in. Doing this will allow the command version to run instead.	2:15
Built-ins use a different documentation system than the regular man pages. There's a built-in called	

help, that shows supporting information about built-ins.	2:47
Typing help by itself displays the names of all the other builtins. Video: Brackets and braces in Bash (1 Notes)	3:05
Parentheses, braces and brackets. Video: Bash expansions and substitutions (4 Notes)	0:03
Bash Expansions and Substitutions: ~ Tilde expansion; {...} Brace expansion; ${...} Parameter expansion; $(...) Command substitution; $((...)) Arithmetic expansion	0:15
In bash, the tilde character represents the value of the user's home variable, and it's used in paths to represent the current user's home directory.	0:38
Another useful tilde expansion trick is that the tilde followed by a dash or a minus represents a Bash variable called OLDPWD, which is the directory that you were just in, if you've recently changed directories.	1:12
Reference: https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#index-tilde-expansion Video: Brace expansion (7 Notes)	1:33
Brace expansion. Typing echo {1..10} gives the numbers 1 through 10. https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#index-brace-expansion 	0:38
Typing echo {10..1} gives the numbers 10 through 1 (same numbers in reverse order).	0:48
Adding a zero in front of the 1 tells bash to pad the numbers. Ref: echo {01..10}	1:08
Sequences can include letters (lower or upper case). Ref: echo {a..z}; echo {Z..A}	1:17
Add an interval by including a second set of dots and an interval number. Ref: echo {1..30..3}; echo {a..z..2}	1:27
Brace expansions can be combined. Ref: touch file_{01..12}{a..d}	1:45
Brace expansion works with set lists of things. Ref: echo {cat,dog,fox}; echo {cat,dog,fox}_{1..5} Video: Parameter expansion (6 Notes)	2:21
Parameter expansion, lets us recall stored values and transform them in various ways. https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html#index-parameter-expansion	0:00
Parameter expansion also often features braces and those are used to make it clear what parameter is being used and to keep the shell from getting confused about nearby words or characters.	0:24
echo ${parameter:6} = expand the parameter starting with the character at position 6	1:05

echo ${parameter:6:3} = expand the parameter starting with the character at position 6 and continuing for 3 characters	1:16
echo ${parameter/oldstring/newstring} = replace part of the string in a parameter	1:54
echo ${parameter//oldstring/newstring} = replace all instances (two slashes after the parameter); echo ${parameter/oldstring/newstring} = replace first instance	2:30

Video: Command substitution (2 Notes)

Command substitution, is a kind of special case expansion which allows us to use the output of a command within another command.	0:01
Command substitution is often used with tools like grep, awk and cut in order to extract specific pieces of text from output.	1:44

Video: Arithmetic expansion (2 Notes)

Arithmetic expansion does math. $((...)) or old style $[...] (deprecated)	0:01
Bash can only do math with integers.	0:53

Chapter: 2. Programming with Bash

Video: Choosing a text editor for Bash scripting (1 Notes)

Text editor: Nano. See other courses: Learning Vim, Learning Emacs, Learning Nano.	0:25

Video: Understanding Bash script syntax (5 Notes)

bash myscript.sh (common file ending for bash scripts)	1:14
Executable bash script includes a shebang as the first line (#!/usr/bin/env bash)	1:32
Make script executable with chmod +x myscript and run using ./myscript (in the folder where the script is located).	3:12
Bash scripts run in a non-interactive shell (clean environment).	3:29
Change options for the subshell with set or shopt in the script.	3:49

Video: Displaying text with echo (2 Notes)

Single quotes are called strong quotes (everything is treated as literal text).	2:17
echo -n "No newline"	3:56