

Bash

Notes for Professionals

Chapter 12: Arrays

Section 12.1: Array Assignments

List Assignment
If you are familiar with Perl, C, or Java, you might think that Bash would use commas to separate elements, however this is not the case; instead, Bash uses spaces.

```
# Array in Perl
my @array = (1, 2, 3, 4);
# Array in Bash
array=(1 2 3 4)
```

Create an array with new elements:

```
array=( "first element" "second element" "third element" )
```

Subscript Assignment
Create an array with explicit element indices:

```
array[[1]]= "fourth element" [[4]]= "fifth element"
```

Assignment by Index

```
array[0]= "first element"
array[1]= "second element"
```

Assignment by Name (associative array)

```
version = 4.0
declare -A array
array[first]= "first element"
array[second]= "second element"
```

Dynamic Assignment
Create an array from the output of other command, for example use seq:

```
array=( $(seq 1 10) )
```

Assign from script's input arguments:

```
array=( "$@" )
```

Assignment within loops:

```
while read -r; do
  array+=("$REPLY") # Array append
  array+=( $(seq 1 10) ) # Assignment by index
  array[[1]]= "$REPLY" # Increment index
  let i++
done < $(seq 1 10) # Command substitution
```

Chapter 21: Quoting

Section 21.1: Double quotes for variable and command substitution

Variable substitutions should only be used inside double quotes.

```
scalation=$((2 + 3))
echo "$scalation" # prints 5
echo $(scalation) # prints 2, the list of files in the current directory, and 3
echo "${scalation}" # prints 5
```

Outside of double quotes, `var` takes the value of `var`, splits it into whitespace-delimited parts, and interprets each part as a glob (wildcard) pattern. Unless you want this behavior, always put `var` inside double quotes: `"$var"`. The same applies to command substitutions: `$(mycommand)` is the output of `mycommand`, `$(mycommand)` is the result of `split-glob` on the output.

```
echo "$var" # good
echo "$(mycommand)" # good
another="$var" # also works, assignment is implicitly double-quoted
make -D "$(uname -s)" # bad! This is not a bash assignment.
make -D "$(uname -s)" # good
make -D "$(uname -s)" # also good
```

Command substitutions get their own quoting contexts. Writing a literally nested substitution is easy because the parser will keep track of nesting depth instead of greedily searching for the first `"` character. The StackOverflow syntax highlighter parses this wrong, however. For example:

```
echo "formatted text: $(printf "a + b = %04d" "$((1))") # formatted text: a + b = 0001"
```

Variable arguments to a command substitution should be double-quoted inside the expansions as well:

```
echo "$(mycommand "$arg1" "$arg2")"
```

Section 21.2: Difference between double quote and single quote

Double quote	Single quote
Allows variable expansion	Prevents variable expansion
Allows history expansion if enabled	Prevents history expansion
Allows command substitution	Prevents command substitution
<code>+</code> and <code>@</code> can have special meaning	<code>+</code> and <code>@</code> are always literals
Can contain both single quote or double quote	Single quote is not allowed inside single quote
<code>\"</code> , <code>'</code> , <code>`</code> , <code>~</code> can be escaped with <code>\"</code> to prevent their special meaning. All of them are literals.	

Properties that are common to both:

- Prevents globbing
- Prevents word splitting

Examples:

```
Bash Notes for Professionals
```

Chapter 36: Internal variables

Section 36.1: Bash internal variables at a glance

Variable	Details
<code>\$+</code> / <code>\$@</code>	Function/script positional parameters (arguments). Expand as follows: <ul style="list-style-type: none"><code>\$+</code> is the same as <code>"\$1 \$2 ..."</code> (note that it generally makes no sense to leave those unquoted)<code>\$@</code> is the same as <code>"\$1" "\$2" ...</code>
<code>\$#</code>	Number of positional parameters passed to the script or function
<code>\$1</code>	Process ID of the last (right-most) for pipeline's command in the most recently job put into the background (note that it's not necessarily the same as the job's process group ID when job control is enabled)
<code>\$!</code>	ID of the process that executed <code>bash</code>
<code>\$n</code>	Exit status of the last command
<code>\$(n)</code>	Positional parameters, where <code>n=1, 2, 3, ... 9</code>
<code>\$*</code>	Positional parameters (same as above), but it can be <code>> 9</code>
<code>\$*</code>	In scripts, path with which the script was invoked with <code>bash -e "printf %s\n" "\$@"</code> name
<code>\$*</code>	Last field of the last argument after the inline script, otherwise the <code>arg[0]</code> that <code>bash</code> received
<code>\$PATH</code>	Internal field separator
<code>\$PWD</code>	PATH environment variable used to look-up executables
<code>\$PDIR</code>	Previous working directory
<code>\$PWD</code>	Present working directory
<code>\$FUNCTION</code>	Array of function names in the execution call stack
<code>\$BASH_SOURCE</code>	Array containing source paths for elements in <code>\$FUNCTION</code> array. Can be used to get the script path.
<code>\$BASH_ALIASES</code>	Associative array containing all currently defined aliases
<code>\$BASH_REMATCH</code>	Array of matches from the last regex match
<code>\$BASH_VERSION</code>	Bash version string
<code>\$BASH_VERSINFO</code>	An array of 6 elements with Bash version information
<code>\$BASH</code>	Absolute path to the currently executing Bash shell itself (theoretically determined by <code>bash --help</code> on <code>arg[0]</code> and the value of <code>\$PDIR</code> may be wrong in corner cases)
<code>\$BASH_SHELL</code>	Both <code>bash</code> level
<code>\$UID</code>	Real (not effective if different) User ID of the process running <code>bash</code>
<code>\$PS1</code>	Primary command line prompt; see Using the <code>PS*</code> variables
<code>\$PS2</code>	Secondary command line prompt; see Using the <code>PS*</code> variables
<code>\$PS3</code>	Tertiary command line prompt (used for additional input)
<code>\$PS4</code>	Quaternary command line prompt (used in select loop)
<code>\$RANDOM</code>	A pseudo-random integer between 0 and 32767
<code>\$RANDOM</code>	Variable used by <code>read</code> by default when no variable is specified (also with verbose output)
<code>\$REPLY</code>	Array variable that holds the exit status values of each command in the most recently executed foreground pipeline.
<code>\$_</code>	Array variable that holds the exit status values of each command in the most recently executed foreground pipeline.

100+ pages
of professional hints and tricks

Contents

About	1
Chapter 1: Getting started with Bash	2
Section 1.1: Hello World	2
Section 1.2: Hello World Using Variables	4
Section 1.3: Hello World with User Input	4
Section 1.4: Importance of Quoting in Strings	5
Section 1.5: Viewing information for Bash built-ins	6
Section 1.6: Hello World in "Debug" mode	6
Section 1.7: Handling Named Arguments	7
Chapter 2: Script shebang	8
Section 2.1: Env shebang	8
Section 2.2: Direct shebang	8
Section 2.3: Other shebangs	8
Chapter 3: Navigating directories	10
Section 3.1: Absolute vs relative directories	10
Section 3.2: Change to the last directory	10
Section 3.3: Change to the home directory	10
Section 3.4: Change to the Directory of the Script	10
Chapter 4: Listing Files	12
Section 4.1: List Files in a Long Listing Format	12
Section 4.2: List the Ten Most Recently Modified Files	13
Section 4.3: List All Files Including Dotfiles	13
Section 4.4: List Files Without Using `ls`	13
Section 4.5: List Files	14
Section 4.6: List Files in a Tree-Like Format	14
Section 4.7: List Files Sorted by Size	14
Chapter 5: Using cat	16
Section 5.1: Concatenate files	16
Section 5.2: Printing the Contents of a File	16
Section 5.3: Write to a file	17
Section 5.4: Show non printable characters	17
Section 5.5: Read from standard input	18
Section 5.6: Display line numbers with output	18
Section 5.7: Concatenate gzipped files	18
Chapter 6: Grep	20
Section 6.1: How to search a file for a pattern	20
Chapter 7: Aliasing	21
Section 7.1: Bypass an alias	21
Section 7.2: Create an Alias	21
Section 7.3: Remove an alias	21
Section 7.4: The BASH_ALIASES is an internal bash assoc array	22
Section 7.5: Expand alias	22
Section 7.6: List all Aliases	22
Chapter 8: Jobs and Processes	23
Section 8.1: Job handling	23
Section 8.2: Check which process running on specific port	25

Section 8.3: Disowning background job	25
Section 8.4: List Current Jobs	25
Section 8.5: Finding information about a running process	25
Section 8.6: List all processes	26
Chapter 9: Redirection	27
Section 9.1: Redirecting standard output	27
Section 9.2: Append vs Truncate	27
Section 9.3: Redirecting both STDOUT and STDERR	28
Section 9.4: Using named pipes	28
Section 9.5: Redirection to network addresses	30
Section 9.6: Print error messages to stderr	30
Section 9.7: Redirecting multiple commands to the same file	31
Section 9.8: Redirecting STDIN	31
Section 9.9: Redirecting STDERR	32
Section 9.10: STDIN, STDOUT and STDERR explained	32
Chapter 10: Control Structures	34
Section 10.1: Conditional execution of command lists	34
Section 10.2: If statement	35
Section 10.3: Looping over an array	36
Section 10.4: Using For Loop to List Iterate Over Numbers	37
Section 10.5: continue and break	37
Section 10.6: Loop break	37
Section 10.7: While Loop	38
Section 10.8: For Loop with C-style syntax	39
Section 10.9: Until Loop	39
Section 10.10: Switch statement with case	39
Section 10.11: For Loop without a list-of-words parameter	40
Chapter 11: true, false and : commands	41
Section 11.1: Infinite Loop	41
Section 11.2: Function Return	41
Section 11.3: Code that will always/never be executed	41
Chapter 12: Arrays	42
Section 12.1: Array Assignments	42
Section 12.2: Accessing Array Elements	43
Section 12.3: Array Modification	43
Section 12.4: Array Iteration	44
Section 12.5: Array Length	45
Section 12.6: Associative Arrays	45
Section 12.7: Looping through an array	46
Section 12.8: Destroy, Delete, or Unset an Array	47
Section 12.9: Array from string	47
Section 12.10: List of initialized indexes	47
Section 12.11: Reading an entire file into an array	48
Section 12.12: Array insert function	48
Chapter 13: Associative arrays	50
Section 13.1: Examining assoc arrays	50
Chapter 14: Functions	52
Section 14.1: Functions with arguments	52
Section 14.2: Simple Function	53
Section 14.3: Handling flags and optional parameters	53

Section 14.4: Print the function definition	54
Section 14.5: A function that accepts named parameters	54
Section 14.6: Return value from a function	55
Section 14.7: The exit code of a function is the exit code of its last command	55
Chapter 15: Bash Parameter Expansion	57
Section 15.1: Modifying the case of alphabetic characters	57
Section 15.2: Length of parameter	57
Section 15.3: Replace pattern in string	58
Section 15.4: Substrings and subarrays	59
Section 15.5: Delete a pattern from the beginning of a string	60
Section 15.6: Parameter indirection	61
Section 15.7: Parameter expansion and filenames	61
Section 15.8: Default value substitution	62
Section 15.9: Delete a pattern from the end of a string	62
Section 15.10: Munging during expansion	63
Section 15.11: Error if variable is empty or unset	64
Chapter 16: Copying (cp)	65
Section 16.1: Copy a single file	65
Section 16.2: Copy folders	65
Chapter 17: Find	66
Section 17.1: Searching for a file by name or extension	66
Section 17.2: Executing commands against a found file	66
Section 17.3: Finding file by access / modification time	67
Section 17.4: Finding files according to size	68
Section 17.5: Filter the path	69
Section 17.6: Finding files by type	70
Section 17.7: Finding files by specific extension	70
Chapter 18: Using sort	71
Section 18.1: Sort command output	71
Section 18.2: Make output unique	71
Section 18.3: Numeric sort	71
Section 18.4: Sort by keys	72
Chapter 19: Sourcing	74
Section 19.1: Sourcing a file	74
Section 19.2: Sourcing a virtual environment	74
Chapter 20: Here documents and here strings	76
Section 20.1: Execute command with here document	76
Section 20.2: Indenting here documents	76
Section 20.3: Create a file	77
Section 20.4: Here strings	77
Section 20.5: Run several commands with sudo	78
Section 20.6: Limit Strings	78
Chapter 21: Quoting	80
Section 21.1: Double quotes for variable and command substitution	80
Section 21.2: Difference between double quote and single quote	80
Section 21.3: Newlines and control characters	81
Section 21.4: Quoting literal text	81
Chapter 22: Conditional Expressions	83
Section 22.1: File type tests	83

Section 22.2: String comparison and matching	83
Section 22.3: Test on exit status of a command	85
Section 22.4: One liner test	85
Section 22.5: File comparison	85
Section 22.6: File access tests	86
Section 22.7: Numerical comparisons	86
Chapter 23: Scripting with Parameters	88
Section 23.1: Multiple Parameter Parsing	88
Section 23.2: Argument parsing using a for loop	89
Section 23.3: Wrapper script	89
Section 23.4: Accessing Parameters	90
Section 23.5: Split string into an array in Bash	91
Chapter 24: Bash history substitutions	92
Section 24.1: Quick Reference	92
Section 24.2: Repeat previous command with sudo	93
Section 24.3: Search in the command history by pattern	93
Section 24.4: Switch to newly created directory with !#:N	93
Section 24.5: Using !\$	94
Section 24.6: Repeat the previous command with a substitution	94
Chapter 25: Math	95
Section 25.1: Math using dc	95
Section 25.2: Math using bash capabilities	96
Section 25.3: Math using bc	96
Section 25.4: Math using expr	97
Chapter 26: Bash Arithmetic	98
Section 26.1: Simple arithmetic with (())	98
Section 26.2: Arithmetic command	98
Section 26.3: Simple arithmetic with expr	99
Chapter 27: Scoping	100
Section 27.1: Dynamic scoping in action	100
Chapter 28: Process substitution	101
Section 28.1: Compare two files from the web	101
Section 28.2: Feed a while loop with the output of a command	101
Section 28.3: Concatenating files	101
Section 28.4: Stream a file through multiple programs at once	101
Section 28.5: With paste command	102
Section 28.6: To avoid usage of a sub-shell	102
Chapter 29: Programmable completion	103
Section 29.1: Simple completion using function	103
Section 29.2: Simple completion for options and filenames	103
Chapter 30: Customizing PS1	104
Section 30.1: Colorize and customize terminal prompt	104
Section 30.2: Show git branch name in terminal prompt	105
Section 30.3: Show time in terminal prompt	105
Section 30.4: Show a git branch using PROMPT_COMMAND	106
Section 30.5: Change PS1 prompt	106
Section 30.6: Show previous command return status and time	107
Chapter 31: Brace Expansion	109
Section 31.1: Modifying filename extension	109

[Click here to download full PDF material](#)