Cheatography

Beginner Bash Cheat Sheet

by rebecca-burwei via cheatography.com/121026/cs/21996/

Builtin commands

builtin	Uses builtin version of the command
cut -f*number* -d*delimiter*	Displays the column specified by number
disown	Removes processes from the shell's list of jobs. Removes the job id.
eval	Re-run CL processing on arguments. Can be used to run commands passed as variables.
fg %*N*	Bring the job with shell ID *N* to the foreground
getopts	Parses positional parameters
grep -e	Regex searches for a pattern in lines in the argument files, or stdin if no files
jobs	List all jobs
printf	Prints a format string
read	Reads a line from stdin, spits it on `$IFS` characters, and assigns it to shell variables
trap	When specified signals are received, run specified command instead and resume normal execution
type	Displays paths of argument commands, aliases, functions, executables
wait	Waits for all background jobs to finish before finishing the script.

Emacs commands

CTRL-A	Move to beginning of line
CTRL-E	Move to end of line
CTRL-U	Kill backward to beginning of line
CTRL-K	Kill forward to end of line
CTRL-R	Search backward
CTRL-Y	Retrieve (yank) last killed item
ESC-B	Move one word backward
ESC-F	Move one word forward
ESC-DEL	Kill one word backward

Emacs commands (cont)

ESC-D	Kill one word forward
ESC-<	Move to first line of history list
ESC->	Move to last line of history list

Environment files

.bash_profile - Runs when a login shell starts.

.bashrc - Runs when a subshell starts.

.bash_logout - Runs when a login shell exits.

Special Characters

&	Background job
#	Comment
~	Home directory
!	Logical NOT
'	Quote (strong). Skips all CL processing.
"	Quote (weak). Skips all CL processing except variable expansion, command substitution, arithmetic substitution.
<	Redirect input
>	Redirect output
>>	Redirect output and append to file
|	Redirect (pipe) output to next command
/	Separator for pathname directories
;	Separator for shell commands. Use when EOL is missing.
[ ]	Start and end a character-set wildcard
{ }	Start and end a command block. Redirect I/O to a block of commands without starting a subprocess.
( )	Start and end a subshell
(( ))	Perform arithmetic
*	Wildcard
?	Wildcard - single character
$	Variable expression
\	Escape a special character (including RETURN)

More I/O Redirectors

n>&*m*	File descriptor *n* is made to be a copy of output file descriptor *m*
n<&*m*	File descriptor *n* is made to be a copy of input file descriptor *m*

Cheatography

Beginner Bash Cheat Sheet

by rebecca-burwei via cheatography.com/121026/cs/21996/

read builtin

`-a`	Read values into an array
`-d D`	Only read lines up to the character *D*
`-n N`	Only read the first *N* characters of each line
`-p`	Prints the string before reading input
`-r`	Usually backslash indicates a line continuation. This option interprets escaped characters like `\n`
`-s`	Do not echo the characters typed into the terminal
`-t T`	Wait *T* seconds for input, then finish

Signals

INT	Ctrl-C
TSTP	Ctrl-Z
TERM	`kill`
QUIT	`kill -QUIT`
KILL	`kill -KILL`

Variables

$0, $1, $2, ...	Positional parameters
$@	"$1" "$2" "$3" ...
$*	A string of positional params > 0
$#	Number of positional params - 1
$?	Exit status of last command run

Run a script

`source myscript`	Run in current shell
`./myscript`	Run script in a subshell
`myscript`	Run script in subshell. Must be in `$PATH`

Functions

Two ways to define:

`function myfunction { ... }`

`myfunction ( ) { ... }`

Call a function:

`myfunction arg1 arg2`

Keywords:

`local` - Limit variable scope. `$@, $*, $#, $0, $1` are automatically local.

String operators

`${varname:-word}`	Returns *word*
`${varname:=word}`	Sets and returns *word*
`${varname:?message}`	Prints *message* and exits
`${varname:offset:length}`	Returns substring (1-indexed)
`${varname:+word}`	If *varname* is defined, then returns *word*. Else returns null.

If *varname* does not exist or is null, then string operators follow the behavior above (except for the `:+`).

Pattern-matching operators

`${varname#pattern}`	Match shortest from the start and delete
`${varname##pattern}`	Match longest from the start and delete
`${varname%pattern}`	Match shortest from the end and delete
`${varname%%pattern}`	Match longest from the end and delete
`${varname/pattern/replace}`	Match longest and replace
`${varname//pattern/replace}`	Match all and replace

If / else conditions

statement1 `&&` *statement2*	If *statement1* runs, then run *statement2*
statement1 `||` *statement2*	If *statement1* fails, then run *statement2*
statement1 `-a` *statement2*	*statement1* AND *statement2*
statement1 `-o` *statement2*	*statement1* OR *statement2*
`-lt, -le, -eq, -gt, -ge, -ne`	Integer comparisons
`=, !=, <, >`	String comparisons
`-n str1`	*str1* has length > 0
`-z str1`	*str1* has length 0
`-d file`	*file* exists and is a directory
`-e file`	*file* exists

If / else conditions (cont)

`-f` *file*	*file* exists and is a regular file
`-r` *file*	User has read permission on *file*
`-s` *file*	*file* exists and is not empty
`-w` *file*	User has write permission on *file*
`-x` *file*	User has execute permission on *file*, or search permission if it's a directory
`-N` *file*	*file* was modified since it was last read
`-O` *file*	User owns *file*
`-G` *file*	*file*'s group ID matches one of the user's group IDs
file1 `-nt` *file2*	*file1* has a newer modification time than *file2*

All of the above conditions must go in square brackets (`[ ]`) because if/else test against *exit codes*. Parentheses indicting order of operations within square brackets must be escaped with a backslash.

Other flow control

`for` - Defaults to looping through `$@`. Set loop delimiter using `$IFS`.

```
case expression in
  pattern1 )
    statements ;;
  pattern2 | pattern3 )
    statements ;;
  ...
  * )
    last statements ;;
esac
```

```
while condition ; do statements ; done
until condition ; do statements ; done
```

There is also a `select` condition that operates like `case` on user input.

Subshell inheritance

These are inherited by subshells:
- the current directory
- environment variables
- standard input, output, error, and other open file descriptors
- signals that are ignored