Updated: 02/03/2008 00:46:31

Bash Programming Cheat Sheet

Written By: ph34r

A quick cheat sheet for programmers who want to do shell scripting. This is not intended to teach programming, etc. but it is intended for a someone who knows one programming language to begin learning about bash scripting.

Basics

All bash scripts must tell the o/s what to use as the interpreter. The first line of any script should be:

#!/bin/bash

You must make bash scripts executable.

chmod +x filename

Variables

Create a variable - just assign value. Variables are non-datatyped (a variable can hold strings, numbers, etc. with out being defined as such).

varname=value

Access a variable by putting $ on the front of the name

echo $varname

Values passed in from the command line as arguments are accessed as $# where #= the index of the variable in the array of values being passed in. This array is base 1 not base 0.
command var1 var2 var3 .... varX
$1 contains whatever var1 was, $2 contains whatever var2 was, etc.

Built in variables:

$1-$N	Stores the arguments (variables) that were passed to the shell program from the command line.
$?	Stores the exit value of the last command that was executed.
$0	Stores the first word of the entered command (the name of the shell program).
$*	Stores all the arguments that were entered on the command line ($1 $2 ...).
"$@"	Stores all the arguments that were entered on the command line, individually quoted ("$1" "$2" ...).

Quote Marks

Regular double quotes ("like these") make the shell ignore whitespace and count it all as one argument being passed or string to use. Special characters inside are still noticed/obeyed.

Single quotes 'like this' make the interpreting shell ignore all special characters in whatever string is being passed.

The back single quote marks (`command`) perform a different function. They are used when you want to use the results of a command in another command. For example, if you wanted to set the value of the variable contents equal to the list of files in the current directory, you would type the following command: contents=`ls`, the results of the ls program are put in the variable contents.

Logic and comparisons

A command called test is used to evaluate conditional expressions, such as a if-then statement that checks the entrance/exit criteria for a loop.

test expression
Or
[expression]

Numeric Comparisons

int1 -eq int2	Returns True if int1 is equal to int2.
int1 -ge int2	Returns True if int1 is greater than or equal to int2.
int1 -gt int2	Returns True if int1 is greater than int2.
int1 -le int2	Returns True if int1 is less than or equal to int2
int1 -lt int2	Returns True if int1 is less than int2
int1 -ne int2	Returns True if int1 is not equal to int2

String Comparisons

str1 = str2	Returns True if str1 is identical to str2.
str1 != str2	Returns True if str1 is not identical to str2.
str	Returns True if str is not null.
-n str	Returns True if the length of str is greater than zero.
-z str	Returns True if the length of str is equal to zero. (zero is different than null)

File Comparisons

-d filename	Returns True if file, filename is a directory.
-f filename	Returns True if file, filename is an ordinary file.
-r filename	Returns True if file, filename can be read by the process.
-s filename	Returns True if file, filename has a nonzero length.
-w filename	Returns True if file, filename can be written by the process.
-x filename	Returns True if file, filename is executable.

Expression Comparisons

!expression

Returns true if expression is not true
expr1 -a expr2

Returns True if expr1 and expr2 are true. (&& , and)
expr1 -o expr2

Returns True if expr1 or expr2 is true. (||, or)

Logic Con't.

If...then

if [expression]
then
commands
fi

If..then...else

if [expression]
then
commands
else
commands
fi

If..then...else If...else

if [expression]
then
commands
elif [expression2]
then
commands
else
commands
fi

Case select

case string1 in
str1)
commands;;
str2)
commands;;
*)
commands;;
esac

string1 is compared to str1 and str2. If one of these strings matches string1, the commands up until the double semicolon (; ;) are executed. If neither str1 nor str2 matches string1, the commands associated with the asterisk are executed. This is the default case condition because the asterisk matches all strings.

Iteration (Loops)

for var1 in list
do
commands
done

This executes once for each item in the list. This list can be a variable that contains several words separated by spaces (such as output from ls or cat), or it can be a list of values that is typed directly into the statement. Each time through the loop, the variable var1 is assigned the current item in the list, until the last one is reached.

while [expression]

do
commands
done

until [expression]
do
commands
done

Functions

Create a function:

fname(){
commands
}

Call it by using the following syntax: fname

Or, create a function that accepts arguments:

fname2 (arg1,arg2...argN){
commands
}

And call it with: fname2 arg1 arg2 ... argN