Bash Scripting Cheat Sheet

-Tests (for ifs and loops) are done with [ ] or with the test command.

Checking files:

-r file Check if file is readable -w file Check if file is writable -x file Check if have execute access to file
-f file Check if file is an ordinary file (as opposed to a directory, a device special file, etc.)
-s file Check if file has size greater than 0. -d file Check if file is a directory.
-e file Check if file exists. Is true even if file is a directory.

Example:

```
if [ -s file ]
then
      #such and such
fi
```

Checking strings:

s1 = s2 Check if s1 equals s2 s1 != s2 Check if s1 is not equal to s2 -z s1 Check if s1 has size 0
-n s1 Check if s2 has nonzero size s1 Check if s1 is not the empty string.

Example:

```
if [ $myvar = "hello" ] ; then
      echo "We have a match"
fi
```

Checking numbers: Note, variables could be strings representing numbers. Use these to check the numerical value.

n1 -eq n2 Check to see if n1 equals n2 n1 -ne n2 Check to see if n1 is not equal to n2
n1 -lt n2 Check to see if n1 < n2 n1 -le n2 Check to see if n1 <= n2
n1 -gt n2 Check to see if n1 > n2 n1 -ge n2 Check to see if n1 >= n2.

Example:

```
if [ $# -gt 1 ] then
      echo "ERROR: should have 0 or 1 command-line parameters"
fi
```

Boolean operators: ! not -a and -o or

Example:

```
if [ $num -lt 10 -o $num -gt 100 ]
then
      echo "Number $num is out of range"
elif [ ! -w $filename ]
then
      echo "Cannot write to $filename"
fi
```

Note that ifs can be nested. For example:

```
if [ $myvar = "y" ]
then
      echo "Enter count of number of items"
      read num
      if [ $num -le 0 ]
      then
            echo "Invalid count of $num was given"
      else
            #... do whatever ...
      fi
fi
```

The above example illustrates reading a string from the keyboard to a shell variable. Note, most UNIX commands return true (!0) or false (0) to indicate success. At the command line, echo $status. In a shell script use something like this:

```
if grep -q shell bshellref
then
      echo "true"
else
      echo "false"
fi
```

-q is the quiet version of grep. It checks if string ‘shell’ occurs in file ‘bshellref’. It does not print the matching lines.

I/O Redirection:

pgm > file Output of pgm is redirected to file pgm < file Program pgm reads its input from file
pgm >> file Output of pgm is appended to file pgm1 | pgm2 Output of pgm1 is piped into pgm2 as the input to pgm2

n > file Output from stream with descriptor n redirected to file
n >> file Output from stream with descriptor n appended to file
n >& m Merge output from stream n with stream m
n <& m Merge input from stream n with stream m
<< tag Standard input comes from here through next tag at start of line.

Note that file descriptor 0 is normally standard input, 1 is standard output, and 2 is standard error output.

Shell Built-in Variables:

$0 Name of this shell script itself
$1 Value of first command line parameter (similarly $2, $3, etc)
$# In a shell script, the number of command line parameters
$$ Process id of script (really id of the shell running the script)
$* All of the command line parameters
$- Options given to the shell
$? Return the exit status of the last command.

Pattern Matching:

* Matches 0 or more characters
? Matches 1 character
[AaBbCc] Example: matches any 1 char from the list
[^RGB] Example: matches any 1 char not in the list
[a-g] Example: matches any 1 char from this range.

Quoting:

\c Take character c literally
`cmd` Run cmd and replace it in the line of code with its output
"whatever" Take whatever literally, after first interpreting $, `...`, \
'whatever' Take whatever absolutely literally.

```
match=`ls *.bak`        #Puts names of .bak files into shell variable match.
      echo \*           #Echos * to screen, not all filename as in: echo *
      echo '$1$2hello'  #Writes literally $1$2hello on screen.
      echo "$1$2hello"  #Writes value of parameters 1 and 2 and string hello.
```

Grouping: Parentheses may be used for grouping, but must be preceded by backslashes since parentheses normally have a different meaning to the shell (namely to run a command or commands in a subshell). For example:

```
if test \( -r $file1 -a -r $file2 \) -o \( -r $1 -a -r $2 \) then
      #do whatever
fi
```

Case statement: Example looks for a match with a, b, or c. Else, $1 always matches the * case.

```
case "$1" in
      a) cmd1 ;;
      b) cmd2 ;;
      c) cmd3 ;;
      *) cmd4 ;;
esac
```

Loops: Bash supports loops written in a number of forms,

```
for arg in [list]
do
      echo $arg
done
for arg in [list] ; do
      echo $arg
done
NUMBERS="1 2 3"   # You can supply [list] directly
for number in `echo $NUMBERS`
do
      echo $number
done
for number in $NUMBERS
do
      echo -n $number
done
for number in 1 2 3
do
      echo -n $number
done
```

If [list] is a glob pattern then bash can expand it directly, for example:

```
for file in *.tar.gz
do
      tar -xzf $file
done
```

You can also execute statements for [list], for example:

```
for x in `ls -tr *.log`
do
      cat $x &gt;&gt; biglog
done
```

Shell Arithmetic: In the original Bourne shell arithmetic is done using the expr command as in:

result=`expr $1 + 2` result2=`expr $2 + $1 / 2` result=`expr $2 \* 5` #note the \ on the * symbol

With bash, an expression is normally enclosed using [ ] and can use the following operators, in order of precedence:

* / % (times, divide, remainder) + - (add, subtract) < > <= >= (comparison operators)

== != (equal to, not equal to) && (logical and) || (logical or) = (assignment)

Arithmetic is done using long integers.

Example:

result=$[$1 + 3] # Take the value of the first parameter, add 3, and place the sum into result.

Order of Interpretation: The bash shell interprets for each line in the following order:

brace expansion (see a reference book) ~ expansion (for login ids) parameters (such as $1)

variables (such as $var) command substitution (Example: match=`grep DNS *`) arithmetic (from left to right)

word splitting pathname expansion (using *, ?, and [abc])

Other Shell Features:

$var Value of shell variable var ${var}abc Example: value of shell variable var with string abc appended

At start of line, indicates a comment var=value Assign the string value to shell variable var

cmd1; cmd2 Do cmd1 and then cmd2 cmd1 & cmd2 Do cmd1, start cmd2 without waiting for cmd1 to finish

cmd1 && cmd2 Run cmd1, then if cmd1 successful run cmd2, otherwise skip

cmd1 || cmd2 Run cmd1, then if cmd1 not successful run cmd2, otherwise skip

(cmds) Run cmds (commands) in a subshell.

Sed Cheat Sheet

Sed command line options

Sed syntax: sed [options] sed-command [input-file]

-n	Suppress default pattern space printing	sed -n '3 p' employee.txt
-i	Backup and modify input file directly	sed -ibak 's/John/Johnny/' employee.txt
-f	Execute sed script file	sed -f script.sed employee.txt
-e	Execute multiple sed commands	sed -e 'command1' -e 'command2' input-file

Sed substitute command and flags

Syntax: sed 's/original-string/replacement-string/[flags]' [input-file]

g flag	Global substitution	sed 's/Windows/Linux/g' world.txt
1,2.. flag	Substitute the nth occurrence	sed 's/locate/find/2' locate.txt
p flag	Print only the substituted line	sed -n 's/John/Johnny/p' employee.txt
w flag	Write only the substituted line to a file	sed -n 's/John/Johnny/w output.txt' employee.txt
i flag	Ignore case while searching	sed 's/john/Johnny/i' employee.txt
e flag	Substitute and execute in the command line	sed 's/^/ls -l /' files.txt
/ | ^ @ !	Substitution delimiter can be any character	sed 's@/usr/local/bin@/usr/bin@' path.txt
&	Gets matched pattern. Use in replacement string.	sed 's/^.*/<&>/' employee.txt #Encloses whole line between < and >
\(\)	Group using \(and \). Use \1, \2 in	sed 's/\([^,]*\),\([^,]*\),\([^,]*\).*/\1,\3/g' employee.txt
\1 \2 \3	replacement string to refer the group.	#Get only 1st and 3rd column

Sed commands

p	Print pattern space	sed -n '1,4 p' employee.txt
d	Delete lines	sed -n '1,4 d' employee.txt
w	Write pattern space to file	sed -n '1,4 w output.txt' employee.txt
a	Append line after	sed '2 a new-line' employee.txt
i	Insert line before	sed '2 i new-line' employee.txt
c	Change line	sed '2 c new-line' employee.txt
l	Print hidden characters	sed -n l employee.txt
=	Print line numbers	sed = employee.txt | sed '{N;s/\n/ /}'
y	Change case	sed 'y/abcde/ABCDE/' employee.txt
q	Quit sed	sed '3 q' employee.txt
r	Read from file	sed '$ r log.txt' employee.txt
#	Comment inside sed script	

Sed hold and pattern space commands

n	Print pattern space, empty pattern space, and read next line.
x	Swap pattern space with hold space
h	Copy pattern space to hold space
H	Append pattern space to hold space
g	Copy hold space to pattern space
G	Append hold space to pattern space

Grep Cheat Sheet

Wildcards

\d	Any Digit
[\l\u]	Any Letter
.	Any Character
\s	Any White Space
\w	Any Word Character
\l	Any Lowercase Letter
\u	Any Uppercase Letter

Any Digit finds each single digit:
Mary had **3** little lambs. Her flock was made up of these three, and **15** fully-grown sheep.
Any Letter finds each single letter (uppercase or lowercase):
Mary had 3 little lambs. Her flock was made up of these three, and 15 fully-grown sheep.
Any Character finds each single character (except line break):
Mary had 3 little lambs. Her flock was made up of these three, and 15 fully-grown sheep.

Locations

\<	Beginning of Word
\>	End of Word
\b	Word Boundary
^	Beginning of Paragraph
$	End of Paragraph
\	Beginning of Story
\Z	End of Story

Beginning of Paragraph:
Mary had 3 little lambs. Her flock was made up of these three, and 15 fully-grown sheep.
End of Paragraph:
Mary had 3 little lambs. Her flock was made up of these three, and 15 fully-grown shee**p.**
End of Story:
Mary had 3 little lambs. Her flock was made up of these three, and 15 fully-grown sheep.
Her sister, Shari, didn't have any sheep at al**l.**

Repeat

?	Zero or One Time
*	Zero or More Times
+	One or More Times
*?	Zero or One Time (Shortest Match)
+?	One+ Times (Shortest Match)

Shortest Match:
Keeps the search to the first complete sequence. Without "shortest match," InDesign looks to the whole paragraph for the sequence.

Match

()	Marking Subexpression
(?:)	Non-Marking Subexpression
[]	Character Set
|	Or
(?<=)	Positive Lookbehind
(?<!)	Negative Lookbehind
(?=)	Positive Lookahead
(?!)	Negative Lookahead

Look for something OR something else:

Look for: gr(e|a)y	Will find: grey AND gray
Look for: (Red|Green|Blue)	Will find: Red, Green, AND Blue

Look before or after to find a string:

Look for: (?<=\.)\d+	To find this: 123.**45**

Symbols

\\	Backslash Character
\^	Caret Character
\(	Open Parenthesis
\{	Open Brace
\[	Open Bracket

Markers

~#	Any Page Number
~N	Current Page Number
~X	Next Page Number
~V	Previous Page Number
~x	Section Marker
~a	Anchored Object Marker
~F	Footnote Reference Marker
~I	Index Marker

Break Character

~b	Standard Carriage Return ~b
~M	Column Break ~M
~R	Frame Break ~R
~P	Page Break ~P
~L	Odd Page Break ~L
~E	Even Page Break ~E
~k	Discretionary Line Break ~k

White Space

~m	Em Space
~>	En Space
~f	Flush Space
~|	Hair Space
~S	Nonbreaking Space
~s	Nonbreaking Space (Fixed Width)
~<	Thin Space
~/	Figure Space
~.	Punctuation Space
~3	Third Space
~4	Quarter Space
~%	Sixth Space

Quotation Marks

"	Any Double Quotation Marks
'	Any Single Quotation Mark
~"	Straight Dbl Quotation Marks
~{	Double Left Quotation Mark
~}	Double Right Quotation Mark
~'	Straight Single Quotation Mark
~[	Single Left Quotation Mark
~]	Single Right Quotation Mark

POSIX

[[:alnum:]]
[[:alpha:]]
[[:digit:]]
[[:lower:]]
[[:punct:]]
[[:space:]]
[[:upper:]]
[[:word:]]
[[:xdigit:]]
[[:=a=:]]

Hyphens and Dashes

~_	Em Dash
~=	En Dash
~-	Discretionary Hyphen
~~	Nonbreaking Hyphen

Other

~y	Right Indent Tab
~i	Indent to Here
~h	End Nested Style Here
~j	Non-joiner