

## Arithmetic Operators

```
$ var=$(( 20 + 5 ))
$ expr 1 + 3 # 4
$ expr 2 - 1 # 1
$ expr 10 / 3 # 3
$ expr 20 % 3 # 2 (remainder)
$ expr 10 \* 3 # 30 (multiply)
```

## String Operators

Expression	Meaning
<code>\${#str}</code>	Length of <code>\$str</code>
<code>\${str:pos}</code>	Extract substring from <code>\$str</code> at <code>\$pos</code>
<code>\${str:pos:len}</code>	Extract <code>\$len</code> chars from <code>\$str</code> at <code>\$pos</code>
<code>\${str/sub/rep}</code>	Replace first match of <code>\$sub</code> with <code>\$rep</code>
<code>\${str//sub/rep}</code>	Replace all matches of <code>\$sub</code> with <code>\$rep</code>
<code>\${str/#sub/rep}</code>	If <code>\$sub</code> matches front end of <code>\$str</code> , substitute <code>\$rep</code> for <code>\$sub</code>
<code>\${str/%sub/rep}</code>	If <code>\$sub</code> matches back end of <code>\$str</code> , substitute <code>\$rep</code> for <code>\$sub</code>

## Relational Operators

Num	String	Test
<code>-eq</code>	<code>=</code>	Equal to
	<code>==</code>	Equal to
<code>-ne</code>	<code>!=</code>	Not equal to
<code>-lt</code>	<code>\&lt;</code>	Less than
<code>-le</code>		Less than or equal to
<code>-gt</code>	<code>\&gt;</code>	Greater than
<code>-ge</code>		Greater than or equal to
	<code>-z</code>	is empty
	<code>-n</code>	is not empty

## File Operators

	True if file exists and...
<code>-f file</code>	...is a regular file
<code>-r file</code>	...is readable
<code>-w file</code>	...is writable
<code>-x file</code>	...is executable
<code>-d file</code>	...is a directory
<code>-s file</code>	...has a size greater than zero.

## Control Structures

```
if [ condition ] # true = 0
then
# condition is true
elif [ condition1 ]
then
# condition1 is true
elif condition2
then
# condition2 is true
else
# None of the conditions is true
fi
```

```
case expression in
pattern1) execute commands ;;
pattern2) execute commands ;;
esac
```

```
while [ true ]
do
# execute commands
done
```

```
until [ false ]
do
# execute commands
done
```

```
for x in 1 2 3 4 5 # or for x in {1..5}
do
echo "The value of x is $x";
done
```

```
LIMIT=10
for ((x=1; x <= LIMIT ; x++))
do
echo -n "$x "
done
```

```
for file in *~
do
echo "$file"
done
```

```
break [n] # exit n levels of loop
continue [n] # go to next iteration of loop n up
```

## Function Usage

```
function-name arg1 arg2 arg3 argN
```

n.b. functions must be defined before use...

## Function Definition

```
function function-name ()
{
# statement1
# statement2
# statementN
return [integer] # optional
}
```

Functions have access to script variables, and may have local variables:

```
$ local var=value
```

## Arrays

```
$ vars[2]="two" # declare an array
$ echo ${vars[2]} # access an element
$ fruits=(apples oranges pears) # populate array
$ echo ${fruits[0]} # apples - index from 0
$ declare -a fruits # creates an array
```

```
echo "Enter your favourite fruits: "
read -a fruits
echo You entered ${#fruits[@]} fruits
for f in "${fruits[@]}"
do
echo "$f"
done
```

```
$ copy=( "${fruits[@]}" ) # copy an array
$ array=( "${fruits[@]}" "grapes" ) # add to end
$ unset fruits[1] # delete one element
$ unset fruits # delete array
```

Array elements do not have to be sequential - indices are listed in `{!fruits[@]}`:

```
for i in ${!fruits[@]}
do
echo fruits[$i]=${fruits[i]}
done
```

All variables are single element arrays:

```
$ var="The quick brown fox"
$ echo ${var[0]} # The quick brown fox
```

String operators can be applied to all the string elements in an array using `${name[@] ... }` notation, e.g.:

```
$ echo ${arrayZ[@]//abc/xyz} # Replace all occurrences of abc with xyz
```

## User Interaction

```
echo -n "Prompt: "
read
echo "You typed $REPLY."
```

```
echo -n "Prompt: "
read response
echo "You typed $response."
```

```
PS3="Choose a fruit: "
select fruit in "apples" "oranges" "pears"
do
  if [ -n "$fruit" ]
  then
    break
  fi
  echo "Invalid choice"
done
```

```
$ dialog --menu "Choose" 10 20 4 1 apples 2 \
oranges 3 pears 4 bananas 2>/tmp/ans
$ fruit=`cat /tmp/ans`
$ echo $fruit
```

```
$ zenity --list --radiolist --column "Choose" \
--column "Fruit" 0 Apples 0 Oranges 0 Pears 0 \
Bananas > /tmp/ans
$ fruit=`cat /tmp/ans`
$ echo $fruit
```

## Reading Input from a File

```
exec 6<&0 # 'Park' stdin on #6
exec < temp.txt # stdin=file "temp.txt"
read # from stdin
until [ -z "$REPLY" ]
do
  echo "$REPLY" # lists temp.txt
  read
done
exec 0<&6 6<&- # restore stdin
echo -n "Press any key to continue"
read
```

## Trapping Exceptions

```
TMPFILE=`mktemp`
on_break()
{
  rm -f $TMPFILE
  exit 1
}
trap on_break 2 # catches Ctrl+C
```

## Data and Time

```
$ start=`date +%s`
$ end=`date +%s`
$ echo That took=$((end-start)) seconds
$ date +%c" -d19540409
Fri 09 Apr 1954 12:00:00 AM GMT
```

## Case Conversion

```
$ in="The quick brown fox"
$ out=`echo $in | tr [:lower:] [:upper:]`
$ echo "$out"
THE QUICK BROWN FOX
```

## Preset Variables

\$HOME	User's home directory
\$HOSTNAME	Name of host
\$HOSTTYPE	Type of host (e.g. i486)
\$PWD	Current directory
\$REPLY	default variable for READ and SELECT
\$SECONDS	Elapsed time of script
\$TMOUT	Max. script elapsed time or wait time for read

## References

*Linux Shell Scripting Tutorial - A Beginner's handbook*  
<http://www.cyberciti.biz/nixcraft/linux/docs/uniqlinuxfeat/ures/lst/>  
*BASH Programming Introduction, Mike G*  
<http://www.tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>  
*Advanced BASH Scripting Guide, Mendel Cooper*  
<http://tldp.org/LDP/abs/html/>

## Copyright & Licence

This Reference Card is Copyright (c)2007 John McCreesh jpmcc@users.sf.net and is licensed under the Creative Commons Attribution-NonCommercial-Share Alike 2.5 UK: Scotland License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.5/scotland/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.

This version dated:20/02/07

## BASH Quick Reference Card

"All the useful stuff on a single card"

```
#!/bin/bash
$ chmod ugo+x shell_script.sh
```

```
$ bash [options] [file]
Options
-x show execution of [file]
-v echo lines as they are read
```

## Variables

```
$ var="some value" # declare a variable
$ echo $var # access contents of variable
$ echo ${var} # access contents of variable
$ echo ${var:-"default value"} # with default
$ var= # delete a variable
$ unset var # delete a variable
```

Quoting - "\$variable" - preserves whitespace

## Positional Variables

\$0	Name of script
\$1-\$9	Positional parameters #1 - #9
\${10}	to access positional parameter #10 onwards
\$#	Number of positional parameters
"\$*"	All the positional parameters (as a single word) *
"\$@"	All the positional parameters (as separate strings)
\$?	Return value

set [values] - sets positional params to [values]  
set -- - deletes all positional parameters  
shift [n] - move positional params n places to the left

## Command Substitution

```
$ var=`ls *.txt` # Variable contains output
$ var=$(ls *.txt) # Alternative form
$ cat myfile >/dev/null # suppress stdout
$ rm nofile 2>/dev/null # suppress stderr
$ cat nofile 2>/dev/null >/dev/null # suppress both
```