### 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 |
|---|---|
| `${#str}` | Length of $str |
| `${str/#sub/rep}` | If $sub matches front end of $str, substitute $rep for $sub |
| `${str/##sub/rep}` | If $sub matches front end of $str, substitute $rep for $sub - greedy |
| `${str/%sub/rep}` | If $sub matches back end of $str, substitute $rep for $sub |
| `${str/%sub/rep}` | If $sub matches back end of $str, substitute $rep for $sub - greedy |

### Relational Operators

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

### File Operators

| | True if file exists and... |
|---|---|
| -f file | ...is a regular file |
| -r file | ...is readable |
| -w file | ...is writable |
| -x file | ...is executable |
| -d file | ...is a directory |
| -s file | ...has a size greater than zero. |
| -nt file | ...is newer than |

### 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 ;;
  *) execute default commands
esac

while [ true ]
do
#  execute commands
done

until [ true ]
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


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 Definition

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

Functions have access to script variables,
positional variables and may have local variables.

```
$ local var=value
```

### Function Usage

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

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

### User Interaction

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

*- a var must be specified for read*

*- bash select not available*
*- dash is more sensitive to extra spaces*
*- don't use [[*
*- don't use ==*
*- limited string operations: use awk, sed, peip*
*- for control iterates over list only*
*- no arrays*

```
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

`$PWD` Current directory

### References


*dash(1) - Linux man page*
http://linux.die.net/man/1/dash

*Dash as /bin/sh*
https://wiki.ubuntu.com/DashAsBinSh

*BASH Quick Reference Card*
https://www.fejf.de/tips/bash.quickref.pdf


### Copyright & Licence



# DASH Quick Reference Card

*"All the useful stuff on a single card"*

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

$ dash [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` | Command entered |
| `$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
```