Shell scripting

Basic Example:

```
#!/usr/bin/env bash

NAME="Atul"
echo "Hello $NAME!"
```

String Quotes:

```
NAME="Atul"
echo "Hi $NAME"  #=> Hi Atul
echo 'Hi $NAME'  #=> Hi $NAME
```

Variables:

```
NAME="Atul"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```

Shell execution

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"
# Same
```

Conditionals:

```
if [[ -z "$string" ]]; then
echo "String is empty"
elif [[ -n "$string" ]]; then
echo "String is not empty"
fi
```

Functions:

```
get_name() {
echo "John"
}

echo "You are $(get_name)"
```

Brace expansion

```
echo {A,B}.js
{A,B}   Same as A B
{A,B}.js       Same as A.js B.js
{1..5}  Same as 1 2 3 4 5
```

```
Parameter expansions
Basics
name="John"
echo ${name}
echo ${name/J/j}    #=> "john" (substitution)
echo ${name:0:2}    #=> "Jo" (slicing)
echo ${name::2}     #=> "Jo" (slicing)
echo ${name::-1}    #=> "Joh" (slicing)
echo ${name:(-1)}   #=> "n" (slicing from right)
echo ${name:(-2):1} #=> "h" (slicing from right)
echo ${food:-Cake}  #=> $food or "Cake"
length=2
echo ${name:0:length}  #=> "Jo"

--------------------------
Substitution
${FOO%suffix}  Remove suffix
${FOO#prefix}  Remove prefix
${FOO%%suffix} Remove long suffix
${FOO##prefix} Remove long prefix
${FOO/from/to} Replace first match
${FOO//from/to}Replace all
${FOO/%from/to}Replace suffix
${FOO/#from/to}Replace prefix

--------------------------

STR="/path/to/foo.cpp"
echo ${STR%.cpp}    # /path/to/foo
echo ${STR%.cpp}.o  # /path/to/foo.o
echo ${STR%/*}      # /path/to

echo ${STR##*.}     # cpp (extension)
echo ${STR##*/}     # foo.cpp (basepath)

echo ${STR#*/}      # path/to/foo.cpp
echo ${STR##*/}     # foo.cpp

echo ${STR/foo/bar} # /path/to/bar.cpp
STR="Hello world"
echo ${STR:6:5}   # "world"
echo ${STR: -5:5}  # "world"
SRC="/path/to/foo.cpp"
BASE=${SRC##*/}   #=> "foo.cpp" (basepath)
DIR=${SRC%$BASE}  #=> "/path/to/" (dirpath)


----------------------------

# Single line comment
: '
This is a
multi line
comment
```

```
Substrings
${FOO:0:3}     Substring (position, length)
${FOO:(-3):3}  Substring from the right

------------------------------

Default values
${FOO:-val}    $FOO, or val if unset (or null)
${FOO:=val}    Set $FOO to val if unset (or null)
${FOO:+val}    val if $FOO is set (and not null)
${FOO:?message}Show error message and exit if $FOO is unset (or null)

---------------------------------
Manipulation
STR="HELLO WORLD!"
echo ${STR,}   #=> "hELLO WORLD!" (lowercase 1st letter)
echo ${STR,,}  #=> "hello world!" (all lowercase)

STR="hello world!"
echo ${STR^}   #=> "Hello world!" (uppercase 1st letter)
echo ${STR^^}  #=> "HELLO WORLD!" (all uppercase)

-----------------------------------

Loops:
Basic for loop:

for i in /etc/rc.*; do
echo $i
done


-------------------------------------

C-like for loop
for ((i = 0 ; i < 100 ; i++)); do
echo $i
done

-------------------------------------

Ranges
for i in {1..5}; do
echo "Welcome $i"
done


Reading lines
cat file.txt | while read line; do
echo $line
done

-------------------------------------
```

```
Forever
while true; do
...
done

---------------------------------------

Ranges:
for i in {1..5}; do
echo "Welcome $i"
done
With step size
for i in {5..50..5}; do
echo "Welcome $i"
done

---------------------------------------

myfunc() {
echo "hello $1"
}
# Same as above (alternate syntax)
function myfunc() {
echo "hello $1"
}
myfunc "John"

---------------------------------------

myfunc() {
local myresult='some value'
echo $myresult
}
result="$(myfunc)"

---------------------------------------

Raising errors
myfunc() {
return 1
}
if myfunc; then
echo "success"
else
echo "failure"
fi



Conditionals
Conditions
Note that [[ is actually a command/program
that returns either 0 (true) or 1 (false).
Any program that obeys the same logic
```

```
(like all base utils, such as grep(1) or ping(1))
can be used as condition, see examples.

[[ -z STRING ]]Empty string
[[ -n STRING ]]Not empty string
[[ STRING == STRING ]] Equal
[[ STRING != STRING ]] Not Equal
[[ NUM -eq NUM ]]      Equal
[[ NUM -ne NUM ]]      Not equal
[[ NUM -lt NUM ]]      Less than
[[ NUM -le NUM ]]      Less than or equal
[[ NUM -gt NUM ]]      Greater than
[[ NUM -ge NUM ]]      Greater than or equal
[[ STRING =~ STRING ]] Regexp
(( NUM < NUM ))Numeric conditions
More conditions
[[ -o noclobber ]]     If OPTIONNAME is enabled
[[ ! EXPR ]]   Not
[[ X && Y ]]   And
[[ X || Y ]]   Or


---------------------------------------

File conditions
[[ -e FILE ]]  Exists
[[ -r FILE ]]  Readable
[[ -h FILE ]]  Symlink
[[ -d FILE ]]  Directory
[[ -w FILE ]]  Writable
[[ -s FILE ]]  Size is > 0 bytes
[[ -f FILE ]]  File
[[ -x FILE ]]  Executable
[[ FILE1 -nt FILE2 ]]  1 is more recent than 2
[[ FILE1 -ot FILE2 ]]  2 is more recent than 1
[[ FILE1 -ef FILE2 ]]  Same files

---------------------------------------

# String
if [[ -z "$string" ]]; then
echo "String is empty"
elif [[ -n "$string" ]]; then
echo "String is not empty"
else
echo "This never happens"
fi
# Combinations
if [[ X && Y ]]; then
...
fi
# Equal
if [[ "$A" == "$B" ]]
# Regex
```

```
if [[ "A" =~ . ]]
if (( $a < $b )); then
echo "$a is smaller than $b"
fi
if [[ -e "file.txt" ]]; then
echo "file exists"
fi
```

```
Arrays
Defining arrays
Fruits=('Apple' 'Banana' 'Orange')
Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"

----------------------------------------
Working with arrays
echo ${Fruits[0]}           # Element #0
echo ${Fruits[-1]}          # Last element
echo ${Fruits[@]}           # All elements, space-separated
echo ${#Fruits[@]}          # Number of elements
echo ${#Fruits}             # String length of the 1st element
echo ${#Fruits[3]}          # String length of the Nth element
echo ${Fruits[@]:3:2}       # Range (from position 3, length 2)
echo ${!Fruits[@]}          # Keys of all elements, space-separated
----------------------------------------
Operations

Fruits=("${Fruits[@]}" "Watermelon")    # Push
Fruits+=('Watermelon')                  # Also Push
Fruits=( ${Fruits[@]/Ap*/} )            # Remove by regex match
unset Fruits[2]                         # Remove one item
Fruits=("${Fruits[@]}")                 # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=(`cat "logfile"`)                 # Read from file

----------------------------------------

Iteration
for i in "${arrayName[@]}"; do
echo $i
done


Dictionaries
Defining
declare -A sounds
sounds[dog]="bark"
sounds[cow]="moo"
sounds[bird]="tweet"
sounds[wolf]="howl"
----------------------------------------
```

```
Working with dictionaries
echo ${sounds[dog]} # Dog's sound
echo ${sounds[@]}   # All values
echo ${!sounds[@]}  # All keys
echo ${#sounds[@]}  # Number of elements
unset sounds[dog]   # Delete dog

----------------------------------------
Iteration
Iterate over values
for val in "${sounds[@]}"; do
echo $val
done
Iterate over keys
for key in "${!sounds[@]}"; do
echo $key
done
----------------------------------------

printf
printf "Hello %s, I'm %s" Sven Olga
#=> "Hello Sven, I'm Olga

printf "1 + 1 = %d" 2
#=> "1 + 1 = 2"

printf "This is how you print a float: %f" 2
#=> "This is how you print a float: 2.000000"
```