Shell script:

* Is a plain text file.
* Executes the commands listed in the script.
* Can automate tedious or repetitive tasks.
* Act as form of documentation.
* Are fair quick and easy to write.
* Anything you can do at the common line can be automated with a shell script.

Types:

* The Bourne Shell
* The C Shell
* The Korn Shell
* The GNU Bourne-Again Shell

Basic script:

#!/bin/sh

echo " enter your name"

read NAME ------------- **read** command which takes the input from the keyboard and assigns it as the value of the variable

echo "hello, $NAME"

variable Names

* The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (\_).
* By convention, Unix shell variables will have their names in UPPERCASE.

Defining Variables

\* Variable name = “variable value” --- Variables of this type are called **scalar variables**. A scalar variable can hold only one value at a time.

Accessing variables:

* To access the value stored in a variable, prefix its name with the dollar sign (**$**) –

Ex: #! bin/bash

NAME=” HARITHA”

echo $NAME

special variables:

$0: The filename of the current script.

$n ($1, $2): display first and second argument

$#: The number of arguments supplied to a script.

$\*: All the arguments are double quoted. If a script receives two arguments, $\* is equivalent to $1 $2.

$@: All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.

$?: The exit status of the last command executed.

$$: The process number of the current shell. For shell scripts, this is the process ID under which they are executing.

$!: The process number of the last background command.

Ex:

#!/bin/sh

echo "file name: $0"

echo "pid: $$"

echo "number of arguments: $#"

echo "first parameter: $1"

echo "second parameter: $2"

echo "double quotes: $\*"

echo "double quotes: $@"

o/p:

pid: 648

number of arguments: 2

first parameter: haritha

second parameter: ediga

double quotes: haritha ediga

double quotes: haritha ediga

arrays: This can hold multiple values at the same time.

Script:

#!/bin/sh

NAME[1]="ABC"

NAME[2]="DEF"

NAME[3]="GHI"

NAME[4]="JKL"

NAME[5]="MNO"

echo " names: ${NAME[\*]}"

operators:

* Arithmetic Operators
* Relational Operators
* Boolean Operators
* String Operators
* File Test Operators

Arithmetic Operators

Bourne shell didn't originally have any mechanism to perform simple arithmetic operations but it uses external programs, either **awk** or **expr**.

Addition:

#!/bin/sh

a=10

b=10

val=`expr $a + $b`

echo "total value: $val"

subtraction:

#!/bin/sh

a=5

b=3

val=`expr $a - $b`

echo "total value: $val"

multiplication:

#!/bin/sh

a=10

b=20

val=`expr $a \\* $b`

echo $val

division

#!/bin/sh

a=10

b=20

val=`expr $a / $b`

echo $val

modulus:

#!/bin/sh

a=3

b=9

val=`expr $a % $b`

echo $val

Arithmetic operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| + (Addition) | Adds values on either side of the operator | `expr $a + $b` will give 30 |
| - (Subtraction) | Subtracts right hand operand from left hand operand | `expr $a - $b` will give -10 |
| \* (Multiplication) | Multiplies values on either side of the operator | `expr $a \\* $b` will give 200 |
| / (Division) | Divides left hand operand by right hand operand | `expr $b / $a` will give 2 |
| % (Modulus) | Divides left hand operand by right hand operand and returns remainder | `expr $b % $a` will give 0 |
| = (Assignment) | Assigns right operand in left operand | a = $b would assign value of b into a |
| == (Equality) | Compares two numbers, if both are same then returns true. | [ $a == $b ] would return false. |
| != (Not Equality) | Compares two numbers, if both are different then returns true. | [ $a != $b ] would return true. |

Relational operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **-eq** | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a -eq $b ] is not true. |
| **-ne** | Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true. | [ $a -ne $b ] is true. |
| **-gt** | Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true. | [ $a -gt $b ] is not true. |
| **-lt** | Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true. | [ $a -lt $b ] is true. |
| **-ge** | Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -ge $b ] is not true. |
| **-le** | Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true. | [ $a -le $b ] is true |

Boolean operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **!** | This is logical negation. This inverts a true condition into false and vice versa. | [ ! false ] is true. |
| **-o** | This is logical **OR**. If one of the operands is true, then the condition becomes true. | [ $a -lt 20 -o $b -gt 100 ] is true. |
| **-a** | This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false. | [ $a -lt 20 -a $b -gt 100 ] is false. |

String Operators:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| **=** | Checks if the value of two operands are equal or not; if yes, then the condition becomes true. | [ $a = $b ] is not true. |
| **!=** | Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true. | [ $a != $b ] is true. |
| **-z** | Checks if the given string operand size is zero; if it is zero length, then it returns true. | [ -z $a ] is not true. |
| **-n** | Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true. | [ -n $a ] is not false. |
| **str** | Checks if **str** is not the empty string; if it is empty, then it returns false. | [ $a ] is not false. |

Decision making:

* if...else statement
* case...esac statement

Shell supports following forms of **if…else** statement −

* [if...fi statement](https://www.tutorialspoint.com/unix/if-fi-statement.htm)
* [if...else...fi statement](https://www.tutorialspoint.com/unix/if-else-statement.htm)
* [if...elif...else...fi statement](https://www.tutorialspoint.com/unix/if-elif-statement.htm)

EX:

#!/bin/bash

if [ $a -eq $b ]

then

echo equal

else

echo not equal

fi

loops:

#!/bin/sh

a=0

while [ "$a" -lt 10 ] # this is loop1

do

b="$a"

while [ "$b" -ge 0 ] # this is loop2

do

echo -n "$b "

b=`expr $b - 1`

done

echo

a=`expr $a + 1`

done

control loop:

\*break

\*continue

Script:

while [ $a -lt 10 ]

do

echo $a

if [ $a -eq 5 ]

then

break

fi

a=`expr $a + 1`

done

functions:

#!/bin/sh

# Define your function here

Hello () {

echo "Hello World"

}

# Invoke your function

Hello