Shell Scripting 2

$$ Command (Special Variable):

Description:

The special variable $$ holds the process ID number (PID) of the current process. It's useful for various tasks like identifying running shell scripts or creating unique temporary files.

Example 1:

- **echo $$**

This command prints the PID of the current shell process.

Example 2:

touch coco.txt

mv coco.txt coco.$$_$$

In this example, a file named coco.txt is created, and then it's renamed with a unique name that includes the PID of the current process, making it coco.<PID>_<PID>. This ensures uniqueness, which is helpful when creating temporary files in a multiprocessing environment.

Command Title: test (or [ ])

Description:

The test command (or its equivalent [ ]) is used for evaluating conditions in shell scripts. It checks various conditions like string equality, numeric comparison, file properties, etc., and returns an exit status indicating whether the condition is true or false.

Example 1 - String Operators:

```
name="user"
test $name = "user"
echo $?
test $name = "userr"
echo $?
```

These commands test whether the variable name is equal to "user" and "userr", respectively, and print the exit status (0 for true, non-zero for false).

Operators used in test command:

Operator	Returns TRUE (exit status of 0) if
`string1 = string2`	`string1` is identical to `string2`
`string1 != string2`	`string1` is not identical to `string2`
`string`	`string` is not null
`-n string`	`string` is not null (and `string` must be seen by `test`)
`-z string`	`string` is null (and `string` must be seen by `test`)

Example 2 - An Alternative Format:

```
name=""
[ -n "$name" ]
echo $?
[ -z "$name" ]
echo $?
```

These commands demonstrate the use of [ ] with string tests, checking if name is non-empty and empty, respectively.

The following two tables are operators to be used in the expressions:

Operator	Returns TRUE (exit status of 0) if
int_1 -eq int_2	int_1 is equal to int_2
int_1 -ge int_2	int_1 is greater than or equal to int_2
int_1 -gt int_2	int_1 is greater than int_2
int_1 -le int_2	int_1 is less than or equal to int_2
int_1 -lt int_2	int_1 is less than int_2
int_1 -ne int_2	int_1 is not equal to int_2

Operator	Returns TRUE (exit status of 0) if
`-d file`	`file` is a directory
`-e file`	`file` exists
`-f file`	`file` is an ordinary file
`-r file`	`file` is readable by the process
`-s file`	`file` has nonzero length
`-w file`	`file` is writable by the process
`-x file`	`file` is executable
`-L file`	`file` is a symbolic link

Command Title: The Logical AND Operator (-a)

Description:

The -a operator performs a logical AND of two expressions and returns true only if both joined expressions are true.

Example:

count=3

["$count" -ge 0 -a "$count" -lt 10]

echo $?

The statement ["$count" -ge 0 -a "$count" -lt 10] will be true if the variable count contains an integer value greater than or equal to zero but less than 10. Thus, echo $? will display a 0 since the previous expression succeeds.

Note:

The -a operator has lower precedence than the integer comparison operators (and the string and file operators). The expression [("$count" -ge 0) -a ("$count" -lt 10)] demonstrates this precedence. When using parentheses, ensure they are quoted properly. The correct syntax is [\("$count" -ge 0 \) -a \("$count" -lt 10 \)].

Command Title: The Logical OR Operator (-o)

Description:

The -o operator is similar to the -a operator, forming a logical OR of two expressions. Evaluation of the expression will be true if either the first expression is true or the second expression is true.

Example:

```
count=3

[ "$count" -ge 0 -o "$count" -lt 10 ]

echo $?
```

echo $? will display a 0 since the previous expression succeeds.

Note:

The -o operator has lower precedence than the -a operator. The expression "$a" -eq 0 -o "$b" -eq 2 -a "$c" -eq 10 is evaluated by test as "$a" -eq 0 -o ("$b" -eq 2 -a "$c" -eq 10). When using parentheses, remember to quote them properly, as in \("$a" -eq 0 -o "$b" -eq 2 \) -a "$c" -eq 10.

Command Title: if...else...fi

Description:

The if...else...fi construct is used for branching in shell scripts. It executes different sets of commands based on whether a condition evaluates to true or false.

Example:

```
user="$1"
if who | grep "^$user " > /dev/null
then
   echo "$user is logged on"
else
   echo "$user is not logged on"
fi
```

This example checks if a user is logged on and prints a corresponding message.

Command Title: The exit Command

Description:

The exit command immediately terminates the execution of a shell program. It allows you to specify an exit status, which is returned to the calling process.

Example:

```
exit 1
```

This command exits the shell script with an exit status of 1.

Command Title: The elif Construct

Description:

The elif construct allows for avoiding nested if statements by providing an alternative condition to check if the initial if condition is false.

Example:

```
hour=$(date | cut -c12-13)
if [ "$hour" -ge 0 -a "$hour" -le 11 ]
then
   echo "Good morning"
elif [ "$hour" -ge 12 -a "$hour" -le 17 ]
then
   echo "Good afternoon"
else
   echo "Good evening"
fi
```

This example prints a greeting message based on the time of day.

Command Title: The case Command

Description:

The case command allows comparison of a single value against multiple values and executes commands based on the matched pattern.

Example:

```
case "$1"
    in
   0) echo zero;;
   1) echo one;;
   2) echo two;;
   # other cases...
   *) echo "Bad argument; please specify a single digit";;
esac
```

This example translates a digit into its English equivalent.

Title: Special Pattern Matching Characters

Description:

The shell allows the use of special characters for specifying patterns in a case statement, similar to filename substitution. The `?` matches any single character, `*` matches zero or more occurrences of any character, and [...] matches any single character enclosed between the brackets.

Example:

```
if [ $# -ne 1 ]
then
        echo Usage: fname char
        exit 1
fi
# Ensure that only one character was typed
char="$1"
numchars=$(echo -n "$char" | wc -c)
if [ "$numchars" -ne 1 ]
then
        echo Please type a single character
        exit 1
fi
# Now classify it
case "$char"
in
[0-9] ) echo digit;;
[a-z] ) echo lowercase letter;;
[A-Z] ) echo uppercase letter;;
* ) echo special character;;
esac
```

This script classifies a single character into different categories such as digit, lowercase letter, uppercase letter, or special character.

To test the above script, run commands such as ./ fname a, ./ fname Z, ./case_3 3, ./ fname @, and ./ fname 123 to ensure you get the expected output after each execution.

Command Title: The Null Command

Description:

The : (colon) command, also known as the null command, does nothing when executed. It's typically used as a placeholder in situations where a command is required syntactically but no action is necessary.

Example:

```
if who | grep "^$user " > /dev/null
then
   :
else
   echo "$user is not logged on"
fi
```

In this example, the null command is used as a placeholder when the if condition is true, indicating that no action needs to be taken.

Command Title: The && and || Constructs

Description:

The && and || constructs are used for executing commands based on the success or failure of preceding commands. && executes the second command only if the first one succeeds, while || executes the second command only if the first one fails.

Example:

sort bigdata > /tmp/sortout && mv /tmp/sortout bigdata

In this example, the mv command is executed only if the sort command succeeds.

Command Title: Debugging shell scripts with the -x option

Description:

The -x option enables tracing of shell script execution, showing each command as it's executed along with its arguments and expanded variables. This is useful for debugging and understanding the flow of execution.

Example:

sh -x ./fname a

This command runs the shell script case_3 with tracing enabled, providing detailed output of each step during execution.

اللَّهُمَّ صَلِّ عَلَى مُحَمَّدٍ وَعَلَى آلِ مُحَمَّد في الأَوَّلِينَ وَالآخِرِينَ، وَفِي الْمَلأِ الأَعْلَى إِلَى يَوْمِ الدِّينِ