ShellBasicOperators

April 15, 2024

1 Shell Basic Operators

In Shell scripting, there are several types of operators:

1. **Arithmetic Operators**: These are used to perform mathematical operations. Examples include `+` (addition), `-` (subtraction), `*` (multiplication), `/` (division), `%` (modulus), `=` (assignment), `==` (equality), and `!=` (not equal).
2. **Relational Operators**: These are used to compare two values. Examples include `-eq` (equal to), `-ne` (not equal to), `-gt` (greater than), `-lt` (less than), `-ge` (greater than or equal to), and `-le` (less than or equal to).
3. **Boolean Operators**: These are used to perform logical operations. Examples include `!` (NOT), `-o` (OR), and `-a` (AND).
4. **String Operators**: These are used to manipulate strings. Examples include `=` (equal to), `!=` (not equal to), `-z` (string is null), `-n` (string is not null), and `$` (string variable).
5. **File Test Operators**: These are used to test properties of a file. Examples include `-b` (file is block special), `-c` (file is character special), `-d` (file is a directory), `-f` (file is an ordinary file), `-r` (file is readable), `-w` (file is writable), `-x` (file is executable), and `-s` (file has size greater than zero).

Remember that the usage and behavior of these operators can vary depending on the context and the specific shell you are using. Always refer to the documentation of your shell for accurate information.

2 Arithmetic Operators?

Sure, here are the arithmetic operators used in Shell Scripting:

1. **Addition (+)**: Adds two numbers.

   ```
   a=10
   b=20
   echo $((a + b))  # Outputs: 30
   ```

2. **Subtraction (-)**: Subtracts the second number from the first.

   ```
   a=10
   b=20
   echo $((a - b))  # Outputs: -10
   ```

3. **Multiplication (`*`)**: Multiplies two numbers.

```
a=10
b=20
echo $((a * b))  # Outputs: 200
```

4. **Division (`/`)**: Divides the first number by the second. Note that this performs integer division.

```
a=10
b=20
echo $((a / b))  # Outputs: 0
```

5. **Modulus (`%`)**: Finds the remainder of the division of the first number by the second.

```
a=10
b=20
echo $((a % b))  # Outputs: 10
```

6. **Assignment (`=`)**: Assigns the value on the right to the variable on the left.

```
a=10
echo $a  # Outputs: 10
```

7. **Equality (`==`)**: Checks if two numbers are equal.

```
a=10
b=10
if ((a == b)); then
    echo "Equal"
else
    echo "Not Equal"
fi  # Outputs: Equal
```

8. **Not Equal (`!=`)**: Checks if two numbers are not equal.

```
a=10
b=20
if ((a != b)); then
    echo "Not Equal"
else
    echo "Equal"
fi  # Outputs: Not Equal
```

Note: Arithmetic operations in shell scripting are performed in a `$(( ))` construct.

3 Relational Operators?

Relational operators in shell scripting are used to compare two values. Here are the relational operators:

1. **Equal (`-eq`)**: Returns true if two numbers are equal.

```
a=10
b=10
if [ $a -eq $b ]; then
    echo "Equal"
else
    echo "Not Equal"
fi  # Outputs: Equal
```

2. **Not Equal (`-ne`)**: Returns true if two numbers are not equal.

```
a=10
b=20
if [ $a -ne $b ]; then
    echo "Not Equal"
else
    echo "Equal"
fi  # Outputs: Not Equal
```

3. **Greater Than (`-gt`)**: Returns true if the first number is greater than the second.

```
a=10
b=20
if [ $a -gt $b ]; then
    echo "a is greater than b"
else
    echo "a is not greater than b"
fi  # Outputs: a is not greater than b
```

4. **Less Than (`-lt`)**: Returns true if the first number is less than the second.

```
a=10
b=20
if [ $a -lt $b ]; then
    echo "a is less than b"
else
    echo "a is not less than b"
fi  # Outputs: a is less than b
```

5. **Greater Than or Equal (`-ge`)**: Returns true if the first number is greater than or equal to the second.

```
a=10
b=10
if [ $a -ge $b ]; then
    echo "a is greater than or equal to b"
else
    echo "a is less than b"
fi  # Outputs: a is greater than or equal to b
```

6. **Less Than or Equal (`-le`)**: Returns true if the first number is less than or equal to the second.

```
a=10
b=20
if [ $a -le $b ]; then
    echo "a is less than or equal to b"
else
    echo "a is greater than b"
fi  # Outputs: a is less than or equal to b
```

Note: Always use spaces around the brackets in the test conditions. For example, use `[ $a -eq $b ]` instead of `[$a -eq $b]`.

4 What are the logical operators used in shell scripting?

Logical operators in shell scripting are used to combine multiple conditions. Here are the logical operators:

1. **Logical AND (`-a` or `&&`)**: Returns true if both conditions are true.

   ```
   a=10
   b=20
   if [ $a -lt $b -a $b -gt $a ]; then
       echo "Both conditions are true"
   else
       echo "At least one condition is false"
   fi  # Outputs: Both conditions are true
   ```

2. **Logical OR (`-o` or `||`)**: Returns true if at least one of the conditions is true.

   ```
   a=10
   b=20
   if [ $a -gt $b -o $b -gt $a ]; then
       echo "At least one condition is true"
   else
       echo "Both conditions are false"
   fi  # Outputs: At least one condition is true
   ```

3. **Logical NOT (`!`)**: Returns true if the condition is false.

   ```
   a=10
   b=20
   if [ ! $a -gt $b ]; then
       echo "a is not greater than b"
   else
       echo "a is greater than b"
   fi  # Outputs: a is not greater than b
   ```

Note: Always use spaces around the brackets in the test conditions. For example, use `[ $a -lt $b -a $b -gt $a ]` instead of `[$a -lt $b -a $b -gt $a]`.

5 String Operators?

Determining workspace structure

Deciding which workspace information to collect

Gathering workspace info

In your workspace, string operations are performed in the `String/Strings_Operation.sh` file. Here are some of the operations:

- Concatenation: Joining two strings together.

[ ]:
```
greeting=$str1$str2
```

- Length: Getting the number of characters in a string.

[ ]:
```
echo ${#greeting}
```

- Substring: Extracting a part of the string.

[ ]:
```
echo ${greeting:0:5}
```

- Replace: Replacing a substring within a string.

[ ]:
```
echo ${greeting/World/Earth}
```

- Replace all: Replacing all occurrences of a substring within a string.

[ ]:
```
echo ${greeting//o/O}
```

- Uppercase: Converting all characters in a string to uppercase.

[ ]:
```
echo ${greeting^^}
```

- Lowercase: Converting all characters in a string to lowercase.

[ ]:
```
echo ${greeting,,}
```

- Capitalize: Converting the first character of a string to uppercase.

[ ]:
```
echo ${greeting^}
```

- Reverse: Reversing a string.

[ ]:
```
echo $(echo $greeting | rev)
```

- Split: Splitting a string into an array.

[ ]:
```
IFS=", " read -r -a array <<< $greeting
```

- Join: Joining an array into a string.

[ ]:
```
echo $(IFS=", "; echo "${array[*]}")
```

- Check if substring exists: Checking if a string contains a specific substring.

[ ]:
```
if [[ $greeting == *"Hello"* ]]; then
  echo "Substring found."
fi
```

- Check if string starts with: Checking if a string starts with a specific substring.

[ ]:
```
if [[ $greeting == "Hello"* ]]; then
  echo "String starts with 'Hello'."
fi
```

- Check if string ends with: Checking if a string ends with a specific substring.

[ ]:
```
if [[ $greeting == *"World!" ]]; then
  echo "String ends with 'World!'."
fi
```

- Check if string is empty: Checking if a string is empty.

[ ]:
```
if [[ -z $greeting ]]; then
  echo "String is empty."
fi
```

You can find these operations in the `String/Strings_Operation.sh` file.

6 File Test Operators?

File test operators in shell scripting are used to test various properties of a file. Here are the file test operators:

1. **Check if file exists (`-e` or `-f`)**:

```
file="path_to_file"
if [ -e $file ]; then
    echo "File exists"
else
    echo "File does not exist"
fi
```

2. **Check if directory exists (`-d`)**:

```
dir="path_to_dir"
if [ -d $dir ]; then
    echo "Directory exists"
else
    echo "Directory does not exist"
fi
```

3. **Check if file is readable (`-r`):**

   ```
   file="path_to_file"
   if [ -r $file ]; then
       echo "File is readable"
   else
       echo "File is not readable"
   fi
   ```

4. **Check if file is writable (`-w`):**

   ```
   file="path_to_file"
   if [ -w $file ]; then
       echo "File is writable"
   else
       echo "File is not writable"
   fi
   ```

5. **Check if file is executable (`-x`):**

   ```
   file="path_to_file"
   if [ -x $file ]; then
       echo "File is executable"
   else
       echo "File is not executable"
   fi
   ```

6. **Check if file is empty (`-s`):**

   ```
   file="path_to_file"
   if [ -s $file ]; then
       echo "File is not empty"
   else
       echo "File is empty"
   fi
   ```

7. **Check if file is a regular file (as opposed to a directory, a device special file, etc.) (`-f`):**

   ```
   file="path_to_file"
   if [ -f $file ]; then
       echo "File is a regular file"
   else
       echo "File is not a regular file"
   fi
   ```

Note: Always use spaces around the brackets in the test conditions. For example, use `[ -e $file ]` instead of `[-e $file]`.

7 Thank You!