ShellSubstitution

April 15, 2024

1 What is Substitution?

In shell scripting, substitution refers to the process of replacing a pattern, variable, or expression with its corresponding value. There are several types of substitution in shell scripting:

1. **Command Substitution**: Command substitution allows you to use the output of a command as an argument to another command or as a part of a variable assignment. It is performed by enclosing the command in `$(command)` or `` `command` ``.

   ```
   date_today=$(date)
   echo "Today's date is: $date_today"
   ```

2. **Variable Substitution**: Variable substitution allows you to replace a variable with its value. This is done by prefixing the variable name with a `$`.

   ```
   greeting="Hello, world!"
   echo $greeting  # Outputs: Hello, world!
   ```

3. **Arithmetic Substitution**: Arithmetic substitution allows you to perform arithmetic operations and replace an arithmetic expression with its result. This is done by enclosing the expression in `$((expression))`.

   ```
   result=$((2 + 2))
   echo $result  # Outputs: 4
   ```

4. **Wildcard Substitution (Globbing)**: Wildcard substitution allows you to use wildcard characters (`*`, `?`, and `[]`) to match filenames and directories.

   ```
   echo *.txt  # Outputs all .txt files in the current directory
   ```

5. **Brace Expansion**: Brace expansion allows you to generate arbitrary strings. It is performed by enclosing the comma-separated strings in `{}`.

   ```
   echo {A,B,C}.txt  # Outputs: A.txt B.txt C.txt
   ```

6. **Tilde Substitution**: Tilde substitution allows you to replace the tilde (`~`) with the path to the current user's home directory.

   ```
   echo ~  # Outputs the path to the current user's home directory
   ```

7. **Process Substitution**: Process substitution allows you to use the output of a command as an input to another command. It is performed by enclosing the command in `<()` or `>()`.

   ```
   diff <(command1) <(command2)  # Compares the output of command1 and command2
   ```

8. **Parameter Substitution**: Parameter substitution allows you to substitute the value of a variable under certain conditions.

```
echo ${var:-"default"}  # If var is unset or null, the expansion of "default" is substitut
```

2 escape sequences

Escape sequences in shell scripting are used to represent special characters which cannot be typed directly. They are typically used with the `echo -e` command to enable interpretation of these sequences. Here are some common escape sequences:

1. \n: Newline. Moves the cursor to the next line.

```
echo -e "Hello\nWorld"  # Outputs: Hello
                         #          World
```

2. \t: Horizontal tab. Moves the cursor to the next tab stop.

```
echo -e "Hello\tWorld"  # Outputs: Hello   World
```

3. \r: Carriage return. Moves the cursor to the beginning of the line.

```
echo -e "World\rHello"  # Outputs: Hello
```

4. \b: Backspace. Moves the cursor one space to the left.

```
echo -e "Helloo\b World"  # Outputs: Hello World
```

5. \a: Alert. Produces a system alert sound.

```
echo -e "\a"  # Produces a system alert sound
```

6. **\\": Backslash. Prints a literal backslash.

```
echo -e "\\"  # Outputs: \
```

7. \**'**: Single quote. Prints a literal single quote.

```
echo -e "\'"  # Outputs: '
```

8. \**"**: Double quote. Prints a literal double quote.

```
echo -e "\""  # Outputs: "
```

9. \0NNN: Octal value. Prints the character represented by the octal value NNN.

```
echo -e "\042"  # Outputs: "
```

10. \xHH: Hex value. Prints the character represented by the hex value HH.

```
echo -e "\x22"  # Outputs: "
```

Note: The `-e` option of the `echo` command enables interpretation of these escape sequences. If you don't use `-e`, the escape sequences will be printed as plain text.

3 Thank You!