Linux Shell Scripting

A.Jegan

Table of Contents

What is Shell Scripting?

- A shell script is a computer program designed to be run by the Unix shell.
- It's a command-line interpreter.
- Allows for an easy way to perform routine tasks.

Basic Shell Script Structure

```
#!/bin/bash
# This is a comment
echo "Hello, World!"
```

Using Variables

```
#!/bin/bash
name="John"
echo "Hello, $name!"
```

Command Line Arguments

```
#!/bin/bash
echo "Script name: $0"
echo "First argument: $1"
echo "Second argument: $2"
```

If Statements

```
#!/bin/bash
if [ "$1" -gt "100" ]; then
    echo "That's a big number."
else
    echo "That's a small number."
fi
```

For Loops

```
#!/bin/bash
for i in {1..5}; do
    echo "Iteration number $i"
done
```

For Loops

```
#!/bin/bash
for i in {1..5}; do
    echo "Iteration number $i"
done
```

File Handling

- **Reading from a file:** Shell scripts can read the contents of a file line by line using loops. This is useful for processing data or configuration files.
- **Writing to a file:** Scripts can create new files or overwrite existing ones using the '>' redirection operator. This is useful for generating logs, reports, or output files.

File Handling

- **Appending to a file:** Instead of overwriting, scripts can add content to the end of an existing file using the '>>' redirection operator. This is commonly used for logging purposes.
- **File Descriptors:** Every open file is associated with a file descriptor. By default, ‘0‘ is for input (stdin), ‘1‘ for output (stdout), and ‘2‘ for errors (stderr).

Reading from a file:

```
#!/bin/bash
while read line; do
    echo $line
done < myfile.txt
```

Writing to a file:

```
#!/bin/bash
echo "This is some text" > outputfile.txt
```

Appending to a file:

```
#!/bin/bash
echo "This is more text" >> outputfile.txt
```

File Permissions in Linux

- **Types of Permissions:**
  - `r` - Read
  - `w` - Write
  - `x` - Execute
- **Permission Groups:**
  - `u` - User (owner)
  - `g` - Group
  - `o` - Others
  - `a` - All (user + group + others)
- **Changing Permissions:** Use the `chmod` command.
  - Example: `chmod u+x file.txt` (Give execute permission to the owner)
- **Viewing Permissions:** Use the `ls -l` command.
  - Output: `-rw-r--r--` (First character is file type, followed by user, group, and others permissions)
- **Special Permissions:**
  - `s` - Setuid/Setgid
  - `t` - Sticky bit

File Searching Tools in Linux

- **grep:** A powerful pattern searching tool. It searches for a pattern in a file (or input) and prints lines that match the pattern.
  - Example: `grep "pattern" file.txt`
- **find:** Searches for files and directories in a directory hierarchy based on different criteria like name, size, type, etc.
  - Example: `find /path/to/dir -name "*.txt"`
- **awk:** A text processing tool that scans for patterns and processes text based on the patterns. It's especially powerful for columnar data.
  - Example: `awk '/pattern/ print $1' file.txt`
- **sed:** A stream editor used to perform basic text transformations on an input stream or file.
  - Example: `sed 's/old/new/g' file.txt`

Using grep

Task: Search for the word "example" in a file named "sample.txt".

```
# Basic usage
grep "example" sample.txt

# Case-insensitive search
grep -i "example" sample.txt

# Display line numbers along with lines
grep -n "example" sample.txt
```

Using `find`

Task: Find all ‘.txt‘ files in the ‘/home/user/documents‘ directory.

```
# Basic usage
find /home/user -name "*.txt"

# Find files modified in the last 24 hours
find /home/user -name "*.txt" -mtime -1
```

Using awk

Task: Print the first column of a file named "data.csv" where the second column equals "100".

```
# Assuming data.csv is:
# A,100,450
# B,200,650
# C,100,750

awk -F, '$2 == "100" {print $1}' data.csv
# Output:
# A
# C
```

Using sed

Task: Replace all occurrences of "apple" with "orange" in a file named "fruits.txt".

```
# Basic replacement
sed 's/apple/orange/g' fruits.txt

# To save changes to the file
sed -i 's/apple/orange/g' fruits.txt
```

Links in Linux

- **Hard Links:**
  - Acts as a mirror of the target file. Both the link and the target share the same inode.
  - Changes to the link reflect in the target and vice-versa.
  - Cannot link directories or files across different file systems.
  - Example: `ln target_file link_name`
- **Symbolic (Soft) Links:**
  - Acts as a pointer or shortcut to the target file.
  - Has a different inode than the target.
  - Can link across different file systems and can link directories.
  - If the target is deleted, the symbolic link becomes a "dangling" link.
  - Example: `ln -s target_file link_name`
- **Viewing Links:**
  - Use `ls -l` to view links. Symbolic links will show as: `link_name -> target_file`
- **Link Count:**
  - Hard links increase the link count of a file. Use `ls -l` to view the link count (second field).

Inodes in Linux

- **Definition:** An inode (index node) is a data structure in a Unix-style file system that describes a file or directory.
- **Contents of an Inode:**
  - File type (regular, directory, symbolic link, etc.)
  - Permissions
  - Owner and group
  - File size
  - Timestamps (creation, modification, access)
  - Pointers to data blocks
- **Unique Inode Number:** Each inode is identified by a unique inode number within the file system.
- **Finding Inode Number:** Use the `ls -i` command.
  - Example: `ls -i file.txt`
- **Note:** Filenames are not stored in inodes. Instead, directories maintain a mapping of filenames to inode numbers.

Functions in Shell Scripting

Defining a Function: Functions help in organizing and reusing code. They can be defined in two ways:

```
# Method 1
function function_name {
    commands
}

# Method 2
function_name() {
    commands
}
```

Example:

```
function greet {
    echo "Hello, $1!"
}
```

Functions in Shell Scripting

Calling a Function: Once defined, a function can be called by its name.

```
greet "Alice"
```

Returning Values: Functions return an exit status (like commands). You can specify this with the ‘return‘ statement.

```
function check_number {
    if [ $1 -gt 10 ]; then
        return 1
    else
        return 0
    fi
}
check_number 15
echo $?   # This will print 1
```

Note: `$?` gives the exit status of the last command/function.

Conclusion

- Shell scripting is a powerful tool for automating tasks in Unix.
- With practice, you can write complex scripts to handle real-world problems.