Linux Operating System and Applications

Bash Shell Programming

Contents

Introduction to Shell

- A **shell** is a command-line interface that allows users to interact with the Linux operating system.
- Common shell types: `bash`, `ash`, `bsh`, `csh`, `ksh`, `tcsh`, `nologin`
- To check which shell you are currently using:

```
echo $SHELL
```

- A user's **default shell** is specified in the `/etc/passwd` file.

Special Shell Configuration Files

- `/etc/profile` – Startup file for **all users**, executed by **login shells** (automatically runs when logging into the system)
- `~/.bash_profile` – User-specific startup file, executed by **login shells**
- `~/.bashrc` – User-specific startup file, executed by **interactive non-login shells**

  `~/.bash_logout` – Cleanup file, executed when **exiting a login shell**

Variable Declaration

- ❑ Used in shell scripting and to control the **execution environment**
- ❑ Assign a value to a variable: `var_name=value`
- ❑ Access the value of a variable: `$var_name`

```
$ foo="hello world"
$ echo $foo
```

- ❑ `set` – List all defined shell variables
- ❑ `unset` – Remove a shell variable
- ❑ `export` – Convert a shell variable into an **environment variable** (available to other shells and processes)

Environment Variables

- Control the behavior of the command execution environment
- Common environment variables:
  - `HOME` – User's home directory
  - `SHELL` – Current shell program
  - `PATH` – Directories to search for executable files
  - `USER` – Logged-in username
  - `TERM` – Current terminal type
  - `DISPLAY` – Display setting for X Window
  - `PS1` – Command prompt format
  - `LANG` – Current language setting

Special Shell Variables

- `$0` – Name of the current shell script or program
- `$$` – Process ID (PID) of the current shell
- `$?` – Exit code of the most recently executed foreground command
- `$!` – PID of the most recently executed background command

Shell Script

- ❑ A text file (typically with a `.sh` extension) that contains commands (shell commands and programs)
- ❑ Interpreted and executed by the shell
- ❑ Can be called from within another shell script
- ❑ Accepts parameters passed from the command line
- ❑ The first line usually begins with: `#!/bin/bash` (called the shebang)
- ❑ Comments are written using the # symbol

First Program: “hello.sh”

- $ cat > hello.sh

```
#!/bin/bash
# This is a comment: simple hello shell script
echo "Enter your name:"
read name
echo "Hello $name, have a nice day!"
```

- $ ./hello.sh

```
bash: ./hello.sh: Permission denied
```

- $ chmod +x hello.sh
- $ ./hello.sh

```
Enter your name:
Nguyen
Hello Nguyen, have a nice day!
```

Positional Parameters

- Parameters are accessed based on their position in the command line
- You can reassign positional parameters using the `set` command (see practice exercises for details)
- Special variables for working with positional parameters:
  - `$#` – Number of parameters passed
  - `$*` – All parameters as a single string
  - `$n` – The nth parameter (e.g., `$1`, `$2`, etc.)
- $ ./myscript source dest

  $0 : ./myscript

  $1 : source

  $2 : dest

Command Substitution

- Executes a command and places its **output** at the current position in the command line
- Two common syntax forms: **backticks** `` `command` `` or `$(command)`
- Examples
  - **$ date**

    ```
    Mon Oct 14 10:48:04 ICT 2003
    ```
  - **$ today=$(date)**

    $ echo $today

    ```
    Mon Oct 14 10:50:04 ICT 2003
    ```
  - **$ ls -l `which tr`**

Arithmetic Expressions

- ❑ Use `let`, `expr`, or `$(())` for arithmetic operations
- ❑ Supported operators: `+`, `-`, `*`, `/`, `%`, `++`, `--`, `==`, `!=`, `>`, `<`, `&&`, `||`
- ❑ Examples:

let “a = 1 + 1” (a = 2)

a=`expr $a “*” 6` (a = 12)

a=$(($a “/” 4)) (a = 3)

let “area = $len * $width”

let “percent = $num / 100”

let “remain = $n % $d”

Conditional Expressions

- Syntax: `[ expression ]` or `test expression`
- **String comparison**: `=`, `!=`, `-n` (not empty), `-z` (is empty)
- **Integer comparison**: `-eq`, `-ne`, `-lt`, `-le`, `-gt`, `-ge`
- **File tests**:
  - `-d` (is directory)
  - `-f` (is regular file)
  - `-x` (is executable)
  - `-e` (exists)
- **Logical operators**: `!` (NOT), `-o` (OR), `-a` (AND)
- Examples

```
[ string1 = string2 ]

[ $num -lt 10 ]

test ! -d mydir && mkdir mydir

[ -f myfile -a -x myfile ] && ./myfile
```

if Condition

Syntax

```
if [ exp ]; then
   statements;
elif [ expr ]; then
   statements;
else
   statements;
fi
```

Example

```
if [ “$1” = “” ]; then
 echo “Enter value:”
 read num
else
 let “num = $1”
fi
```

case Condition

Syntax

```
case $var in
  val1)
    statements;;
  *)
    statements;;
esac
```

Example

```
case $number in
 1)
   echo "You chose option 1"
   ;;
 2)
   echo "You chose option 2"
   ;;
 3)
   echo "You chose option 3"
   ;;
 *)
   echo "Invalid choice"
   ;;
esac
```

for Loop

- Syntax 1:
```
for var [in list]; do
     statements;
done
```
- Syntax 2:
```
for ((exp1; exp2; exp3)); do
     statements;
done
```
- Example
```
let "sum = 0"
for num in 1 3 5; do
let "sum = $sum + $num"
done
echo $sum
```

while Loop

- while expression; do

```
while expression; do
        statements;
done
```

- Example

```
let “num = 1”
while [ $num -lt 10 ]; do
echo $num
let “num = $num + 2”
done
```

Loop Control Statements

- `break` – Exit the current loop
- `continue` – Skip the remaining part of the current loop iteration
- `exit` – Terminate the shell script
- `return` – Return from a function or sourced shell script
- Example:

```
if [ $# -lt 2 ]; then
    echo “Usage: `basename $0` source dest”
    echo
    exit 1 # failure
fi
```

Function

- Declaration

```
func_name() {
    statements;
}
```

- Usage

```
func_name param1 param2 ...
```

```
Example:
greet() {
 echo "Hello, $1"
}
greet "Bob"
```

- Access parameters inside the function using: $1, $2, ...
- Syntax check: `$ sh -n myscript`
- Show commands and arguments during execution:

```
$ sh -x myscript
```

Example

```
isPrime() {
   n=$1

   if [ "$n" -eq 0 ] || [ "$n" -eq
1 ]; then
       return 0   # Not prime
   fi

   for ((i = 2; i < n; i++)); do
       if (( n % i == 0 )); then
           return 0   # Not prime
       fi
   done

   return 1   # Is prime
}
```

```
# The “main” function
read -p "Enter a number: " n

isPrime $n

if [ $? -eq 0 ]; then
   echo "$n is not a prime number"
else
   echo "$n is a prime number"
fi

exit 0
```

Advanced Commands

- String manipulation & colored output
- Arrays & matrices
- File and directory management
- Date and time handling
- Process management
- Scripting utilities and integration

Array Handling

- ❑ Declaration: array[xx] or declare –a array
- ❑ Access element: ${array[i]}
- ❑ Access all elements: ${array[@]} or ${array[*]}
- ❑ Number of elements: ${#array[@]} or ${#array[*]}
- ❑ Remove an element: unset array[1]
- ❑ Alternative declaration: array=([xx]=XXX [yy]=YYY...)

Example

```
# Indexed array
fruits[0]="apple"
fruits[1]="banana"
fruits[2]="cherry"
# Alternate way
declare -a colors
colors=(red green blue)
# Access elements
echo ${fruits[1]}     # Output: banana
echo ${colors[0]}     # Output: red
```

Example

```
# Access All Elements
echo ${fruits[@]}      # Output: apple banana cherry
echo ${colors[*]}      # Output: red green blue
# Count Elements
echo ${#fruits[@]}     # Output: 3
echo ${#colors[*]}     # Output: 3
# Loop Through Array
for fruit in "${fruits[@]}"; do
   echo "Fruit: $fruit"
done
```

Example

```
# Remove an Element
unset fruits[1]          # Removes "banana"
echo ${fruits[@]}        # Output: apple cherry
```

String Handling

- Get String Length

```
${#string}                # Preferred modern syntax
expr length "$string"     # Older syntax
```

Example:

```
stringZ="abcABC123ABCabc"
echo ${#stringZ}          # Output: 15
```

- Find Index of First Match

```
expr index "$string" "$substring"
```

Returns the **position (1-based)** of the first occurrence of any character in `$substring`.

Example:

```
stringZ="abcABC123ABCabc"
echo `expr index "$stringZ" C12`   # Output: 6
```

String Handling

Remove Substring from Beginning

```
${string#pattern}      # Remove shortest match from the beginning
${string##pattern}     # Remove longest match from the beginning
```

Example

```
filename="file_backup_2025.tar.gz"
echo ${filename#*_}     # Output: backup_2025.tar.gz
echo ${filename##*_}    # Output: 2025.tar.gz
```

Replace Substring

```
${string/substring/replacement}     # Replace first match
${string//substring/replacement}    # Replace all matches
```

Example

```
text="The cat chased the cat."
echo ${text/cat/dog}      # Output: The dog chased the cat.
echo ${text//cat/dog}     # Output: The dog chased the dog.
```

Reading a Multi-line File

Method 1: Using a Pipe

```
cat "$FILENAME" | while
read LINE
do
   echo "$LINE"
   :
done
```

Method 2: Redirecting File Input

```
while read LINE
do
   echo "$LINE"
   :
done < "$FILENAME"
```

Debugging Shell Scripts

- ❑ Use `sh -x` to trace script execution step-by-step

```
sh -x script_name.sh
```

- ❑ Example

```
sh -x is_prime.sh
```

This will:

- Show each command **before** it's executed
- Display the **values of variables** as they are expanded

Cutting Strings with cut

- Syntax:

```
cut -d<delimiter> -f<field_number>
```

- Example, input string:

```
"1;2;3;4;5;6"
```

- Goal: Extract the **5th field** (number 5)

```
echo "1;2;3;4;5;6" | cut -d";" -f5
# Output: 5
```

- Notes:

`-d` specifies the **delimiter** (e.g., `;`, `,`, `:`)

`-f` specifies the **field number** to extract (1-based index)

Cutting Strings with awk

- **Syntax:** Print the *n-th* field from a line

```
awk -F<delimiter> '{ print $n }'
```

  - `-F` sets the **field delimiter**
  - Fields are referenced as `$1`, `$2`, ..., `$n`
  - Default delimiter is **whitespace**

- Example: Get list of users in the `root` group

```
cat /etc/group | grep ^root | awk -F":" '{ print $4 }' | tr "," " "
```

- **Explanation**:
  - `cat /etc/group`: shows all groups
  - `grep ^root`: finds the `root` group line
  - `awk -F":" '{ print $4 }'`: gets the **4th field** (list of users)
  - `tr "," " "`: replaces commas with spaces for better readability

Q&A