RedHat Enterprise Linux Essential

Unit 11: **Configuring the Bash Shell – Shell script**

Objectives

Upon completion of this unit, you should be able to:

- Know how to use local and environment variables
- Know how to inhibit variable expansion
- Know how to create aliases
- Understand how the shell parses a command line
- Know how to configure startup files
- Know how to handle input with the read command and positional parameters

Bash Variables

- Variables are named values
  - Useful for storing data or command output
- Set with **VARIABLE=VALUE**
- Referenced with **$VARIABLE**

```
$ HI="Hello, and welcome to $(hostname)."
$ echo $HI
Hello, and welcome to stationX.
```

Environment Variables

- Variables are *local* to a single shell by default
- *Environment variables* are inherited by child shells
  - Set with **export VARIABLE=VALUE**
  - Accessed by some programs for configuration

Some Common Variables

- Configuration variables
  - `PS1`: Appearance of the **bash** prompt
  - `PATH`: Directories to look for executables in
  - `EDITOR`: Default text editor
  - `HISTFILESIZE`: Number of commands in **bash** history
- Information variables
  - `HOME`: User's home directory
  - `EUID`: User's *effective UID*

example PS1

syntax: `PS1='[display content]'`

\!	Display history number
\#	Display number of current command
\$	Display $ or #
\\	Display symbol \
\d	Display current date
\h	Display hostname
\s	Display shell
\t	Display current time
\u	Display user
\W	Display home work current
\w	Display full path home work current

```
PS1='\t \u@\h \s \$'
```

Aliases

- Aliases let you create shortcuts to commands

```
$ alias dir='ls -laht'
```

- Use **alias** by itself to see all set aliases
- Use **alias** followed by an alias name to see alias value

```
$ alias dir
    alias dir='ls -laht'
```

How bash Expands a Command Line

- Split the line into words
- Expand aliases
- Expand curly-brace statements (**{}**)
- Expand tilde statements (**~**)
- Expand variables (**$**)
- Command-substituation (**$()** and ``)
- Split the line into words again
- Expand file globs (***, ?, [abc]**, etc)
- Prepare I/O redirections (**<**, **>**)
- Run the command!

Preventing Expansion

- Backslash (\ ) makes the next character literal

```
$ echo Your cost: \$5.00
 Your cost: $5.00
```

- Quoting prevents expansion
  - Single quotes (') inhibit all expansion
  - Double quotes (") inhibit all expansion, except:
    - $ (dollar sign) - variable expansion
    - ` (backquotes) - command substitution
    - \ (backslash) - single character inhibition
    - ! (exclamation point) - history substitution

Login vs non-login shells

- Startup is configured differently for login and non-login shells
- Login shells are:
  - Any shell created at login (includes X login)
  - su –
- Non-login shells are:
  - su
  - graphical terminals
  - executed scripts
  - any other bash instances

Bash startup tasks: profile

- Stored in /etc/profile (global) and ~/.bash_profile (user)
- Run for login shells only
- Used for
  - Setting environment variables
  - Running commands (eg mail-checker script)

Bash startup tasks: bashrc

- Stored in /etc/bashrc (global) and ~/.bashrc (user)
- Run for all shells
- Used for
  - Setting local variables
  - Defining aliases

Bash exit tasks

- Stored in ~/.bash_logout (user)
- Run when a login shell exits
- Used for
  - Creating automatic backups
  - Cleaning out temporary files

Scripting: Taking input with positional Parameters

- Positional parameters are special variables that hold the command-line arguments to the script.
- The positional parameters available are `$1, $2, $3, etc. .` These are normally assigned to more meaningful variable names to improve clarity.
- `$*` holds all command-line arguments
- `$#` holds the number of command-line arguments

Scripting: Taking input with the read command

- Use **read** to assign input values to one or more shell variables:
  - **-p** designates prompt to display
  - **read** reads from standard input and assigns one word to each variable\
  - Any leftover words are assigned to the last variable
  - `read -p "Enter a filename: " FILE`

```
#!/bin/bash
read -p "Enter several values:" value1 value2
value3
echo "value1 is $value1"
echo "value2 is $value2"
echo "value3 is $value3"
```

While

```
#!/bin/bash
# SCRIPT: method1.sh
# PURPOSE: Process a file line by line with PIPED while-
read loop.

FILENAME=$1
count=0
cat $FILENAME | while read LINE
do
let count++
echo "$count $LINE"
done

echo -e "\nTotal $count Lines read"
```

While with redirect

```
#!/bin/bash
#SCRIPT: method2.sh
#PURPOSE: Process a file line by line with redirected
while-read loop.

FILENAME=$1
count=0

while read LINE
do
let count++
echo "$count $LINE"

done < $FILENAME

echo -e "\nTotal $count Lines read"
```

Thank You !