ULI101

Week 10

Lesson Overview

- Shell Configuration Files
- Shell History
- Alias Statement
- Shell Variables
- Introduction to Shell Scripting
- Positional Parameters
- echo and read Commands
- if and test statements
- for loop

Shell Configuration Files

- Shell configuration files are scripts that are run when you log in, log out, or start a new shell

- /etc/profile belongs to the root user and is the first start-up file that executes when you log in, regardless of shell

- User-specific config files are in the user's home directory:

  ~/.bash_profile runs when you log in

  ~/.bashrc runs when you start an interactive subshell

  ~/.bash_logout runs when you log out

- The start-up files can be used, for example, to:
  - Set the prompt and screen display
  - Create local variables
  - Create temporary Linux commands (aliases)

Shell History

- Many shells keep a history of recently executed command lines in a file
- This history is used by users to save time, when executing same or similar commands over and over
  - Bash uses the up/down arrow keys
  - Use the Ctrl+r to search by keyword
- Bash stores it's history in the ~/.bash_history file

Alias

- A way to create "shortcuts" or temporary commands in UNIX
- Stored in memory, while the user is logged in
- Usually found in .bash_profile
- Syntax:
  alias name=value

  For example: **`alias dir=ls`**
- Even complex command lines can have an alias – enclose the command within double quotes
  For example:
  alias clearfile=”cat /dev/null >”

Shell Variables

- Shell variables are classified in 2 groups
  - System (shell) variables, describing the working environment
  - User-created variables, associated with scripts
- Variables can be read/write or read-only
- Name of a variable can be any sequence of letters and numbers, but it must not start with a number

Common Shell Variables

- Shell environment variables shape the working environment whenever you are logged in
- Common shell variables include:
  - PS1 – primary prompt
  - PWD – present working directory
  - HOME – absolute path to user's home
  - PATH – list of directories where executables are
  - HOST – name of the host
  - USER – name of the user logged in
  - SHELL – current shell
- The set command will display all available variables

The PATH variable

- PATH is an environment variable present in Unix/Linux operating systems, listing directories where executable programs are located
- Multiple entries are separated by a colon (:)
- Each user can customize a default PATH
- The shell searches these directories whenever a command is invoked in the sequence listed
- In case of multiple matches use the which utility to determine which match has a precedence
- On some systems the present working directory may not be included in the PATH by default
- Use ./ prefix or modify the PATH as needed

Assigning a Value

Syntax: `name=value`

For example:

`course=ULI101`

- If variable values are to contain spaces or tabs they should be surrounded by quotes

  For example: `phone="1 800 123-4567"`

Read-Only Variables

- Including the keyword readonly before the command assignment prevents you from changing the variable afterwards
  For example: `readonly phone="123-4567"`
- After a variable is set, it can be protected from changing by using the readonly command
  Syntax: `readonly variable`
  For example: `readonly phone`
- If no variable name is supplied a list of defined read only variables will be displayed

Removing Variables

For example:

```
course=
```

OR

```
unset course
```

- Read-only variables cannot be removed – you must log out for them to be cleared

Variable Substitution

- Whenever you wish to use the value of a variable (its contents), use the variable name preceded by a dollar sign ($)
- This is called variable substitution

  Example:

```
name=Bob
echo $name
```

Introduction to Shell Scripting

- Shell programming
  - Scope ranges from simple day-to-day tasks to large database-driven CGI applications
- Shell-dependent – each shell script is written for a specific shell, such as bash
- First line of each script can specify the path to the program which executes the script - #! statement, for example: **`#!/bin/bash`**
  - Use the which utility to find out path to use: **`which bash`**
  - This must be the first line and nothing can precede it, not even a single space
  - This line is not necessary if the script will be executed in the default shell of the user
- Any line other than first one starting with a # is treated as a comment

Positional Parameters

- Every script can have parameters supplied
- Command line parameters are referred to as $0…$9
- Parameters > $9 can be accessed by using the shift command
  - shift will literally shift parameters to the left by one or more positions
- Can also use the ${ } form
  - This enables direct access to parameters >$9
    For example: ${10}

Positional Parameters

- $* and $@ represent all command line arguments
- "$*" is a single double-quoted string containing values of all arguments separated by a single space
- "$@" is multiple double-quoted strings, each containing the value of one argument
- $# represents the number of parameters (not including the script name)

echo Command

- Displays messages to the terminal followed by a newline
  - Use the –n option to suppress the default newline
- Output can be redirected or piped
- Arguments can be quoted to preserve spaces, double quotes to allow variable substitution or single quotes to disable variable substitution

read command

- The read command allows obtaining user input and storing it into a variable
  - Everything is captured until the Enter key is pressed

Example:

```
echo –n "What is your name? "
read name
echo Hello $name
```

Using Logic

The purpose of the if statement is to execute a command or commands based on a condition

The condition is evaluated by a test command, represented below by a pair of square brackets

```
if [ condition ]
then
   command(s)
fi
```

if Statement Example

Test with a condition
Notice the spaces after “[“ and before “]”

```
read password

if [  ”$password” = ”P@ssw0rd!”  ]
then
  echo ”BAD PASSWORD!”
fi
```

The test Command

- The test command can be used in two ways:
  - As a pair of square brackets: [condition]
  - The test keyword: test condition
- The condition test can result in success (0) or failure (1), unless the negation "not" (!), is used
- The test can compare numbers, strings, and evaluate various file attributes
  - Use = and != to compare strings,
    for example: ["$name" = "Bob"]
  - Use -z and -n to check string length,
    for example: [! -z "$name"]
  - Use -gt, -lt, -eq, -ne, -le, -ge for number,
    for example: [”$salary” -gt 100000]

The Test Command

- Common file test operations include:
  - -e (file exists)
  - -d (file exists and is a directory)
  - -s (file exists and has a size greater than zero)
  - -w (file exists and write permission is granted)
- Check man test for more details

Using Loops

- A for loop is a very effective way to repeat the same command(s) for several arguments such as file names
  Syntax:

Variable "item" will hold one item from the list every time the loop iterates

- for item in list
  do
  command(s)
  done

List can be typed in explicitly or supplied by a command

Loop Examples

```
for addr in $(cat ~/addresses)
do
        mail -s ”Newsletter” $addr < ~/spam/newsletter.txt
done
```

```
for count in 3 2 1 'BLAST OFF!!!'
do
        sleep 1
        echo $count
done
```

```
for id in $(seq 1 1000)
do
        mkdir student_$id
done
```