CS100: Software Tools & Technologies Lab I

Linux Commands and Shell Scripting

Vishwesh Jatala

Assistant Professor

Department of EECS

Indian Institute of Technology Bhilai

vishwesh@iitbhilai.ac.in

2022-23 W

Remote Connection: ssh

- You can use “secure shell” (ssh) to connect to a remote machine.
- ssh [username@]<remote machine name or IP address>
- If the username is omitted, local username will be used.
  - Remote machine has to be configured to accept ssh connections:
  - ssh daemon (service) has to be running and listening on an open port (by default 22)

Remote Transfer: scp

- Copy files securely over a network using an encrypted ssh transport.
- copy file to remote machine
  - scp file [username]@remote machine:

- copy file from remote machine
  - scp [username]@remote machine:file .

Downloading

- wget [options] URL
  - Download a file from a remote location over HTTP.

Working with Process and Jobs

- A process is an instance of a running program
- Each process is assigned a unique "Process ID" (or PID) when it is created
- These PIDs are used to differentiate between separate instances of the same program

ps

- ps [options]
- Reports a snapshot of the current running processes, including PIDs
- By default, ps is not all that useful because it only lists processes started by the user in the current terminal. Instead...
- ps Options
  - -ef – Lists every process currently running on the system with details.

Kill

- kill <PID>
  - Look up the process's PID with ps
  - Use that PID to kill the process

Scripting!

Why?

- Execute a program many times with various inputs
- ./a.out <input1> <input2>
- Example:
  - TA evaluation

Example

- Consider a program takes input as a file and gives some output
  - ./a.out file.txt
- If you have to execute the program for many files in a directory how to automate it?

Example

- Consider you have two programs
  - ./program1 file1
  - ./program2 file2
- Execute program1 if file1 start with a and program2 otherwise.

Shell Scripting

- Helps in automation!
- Simple and easy to use
- Running a program or creating a program environment.
- Helps in solving complex tasks easily

Shell Scripting

- A script is very similar to a program, although it is usually much simpler to write
- It is executed from source code
- Shell scripts are scripts designed to run within a command shell like bash.

Variable

- **Local Variables**
  - To get anything done we need variables.
  - To read the values in variables, precede their names by a dollar sign ($).
  - We can print the contents of any variable using the echo command

Expression Evaluations

- **let evaluates all expressions following the equal sign**
  - VAR1=2
  - let VAR2=$VAR1+15
  - let VAR2++
  - echo $VAR2
  - 18

Environment Variables

- Used by the system to define aspects of operation.
- Examples:
  - $SHELL - which shell will be used by default
  - $PATH - a list of directories to search for binaries
  - $HOSTNAME - the hostname of the machine
  - $HOME - current user's home directory
- How to a get all the environment variables.
  - env

Quotes

- 3 different types of quotes to enclose strings, and they have different meanings:
  - **Single quotes (‘):** preserves the literal value of each character. A single quote may not occur between single quotes, even when preceded by a backslash.
  - **Double quotes (“):** preserves the literal value of all characters within the quotes, with the exception of $
  - **Back quotes (`):** Executes the command within the quotes.

Examples

- echo "$USER owes me $ 1.00"
  - abrahao owes me $ 1.00
- echo '$USER owes me $ 1.00'
  - $USER owes me $ 1.00
- echo "I am $USER and today is `date`"
  - I am abrahao and today is Wed Feb 11 16:23:30 EST 2009

Command Line Variables (arguments)

- When we pass arguments to a bash script, we can access them in a very simple way:
  - $1, $2, ... $10, $11 : are the values of the first, second
  - etc arguments
  - $0 : The name of the script
  - $# : The number of arguments
  - $@ : All the arguments, "$@" expands to "$1" "$2" ... "$n"

Examples

- mult.sh file
  - lex x=$1*$2
    - echo $x
- Usage: ./multi.sh 5 10

Operators

- a++, a--: Post-increment/decrement
- ++a, --a : Pre-increment/decrement
- a+b, a-b : Addition/subtraction
- a*b, a/b : Multiplication/division
- a%b : Modulo
- a**b : Exponential
- a>b, a<b : Greater than, less than
- a==b, a!=b : Equality/inequality
- =, +=, -= : Assignments

Conditional Statements

- if [expression]
- then
- statement1
- else
- statement2
- fi

Test Expression Evaluation

- First to compare two numbers:
  - n1 -eq n2 : tests if n1 = n2
  - n1 -ne n2 : tests if n1 6 = n2
  - n1 -lt n2 : tests if n1 < n2
  - n1 -le n2 : tests if n1 ≤ n2
  - n1 -gt n2 : tests if n1 > n2
  - n1 -ge n2 : tests if n1 ≥ n2
- If either n1 or n2 is not a number, the test fails.

What does this program do?

```
arg=`grep $2 $1 | wc -l`
arg2=`grep $3 $1 | wc -l`
if [ $arg -lt $arg2 ]
then
        echo "$3 is more frequent"
elif [ $arg -eq $arg2 ]
then
        echo "Equally frequent"
else
        echo "$2 is more frequent"
fi
```

Usage: ./ifeq.sh <file> string1 string2

String comparison

- To perform tests on strings use
  - s1 == s2 : s1 and s2 are identical
  - s1 != s2 : s1 and s2 are different
  - s1 : s1 is not the null string
- Make sure you leave spaces! s1==s2 will fail!

Example

```
#Initializing two variables
a=”First”
b=”Second”

if [ $a == $b ]
then
    #If they are equal then print this
    echo "a is equal to b"
else
    #else print this
    echo "a is not equal to b"
fi
```

Path Testing

- If path is a string indicating a path, we can test if it is a valid path, the type of file it represents and the type of permissions associated with it:
  - -e path : tests if path exists
  - -f path : tests if path is a file
  - -d path : tests if path is a directory
  - -r path : tests if you have permission to read the file
  - -w path : tests if you have write permission
  - -x path : tests if you have execute permission

Path Testing Example

```
if [ -f $1 ]
then
    Perform some actions
else
    echo "This script needs a file as its input dummy!"
fi
```

Combining Expressions

- You can combine tests:

```
if [ testexp1 -a testexp2 ]
then
    cmd
fi
```

- -a : and
- -o : or
- !testexp1 : not

Loops!

While Loop

```
while [ condition ];
do
    # statements
    # commands
done
```

What does this program do?

```
#!/usr/bin/bash

a=7
while [ $a -gt 4 ];
do
    echo $a
    ((a--))
done

echo "Out of the loop"
```

For Loop

```
for var in list ; do
    commands
done
```

For Loop

```
for i in 1 2 3 4; do echo $i; done
for i in {1..4}; do echo $i; done
```

For Loop Example

```
i=0
for f in "$@"
do
    j=`wc -l < $f`
    i=$(($i+$j))
done
echo $i
```

Recall that $@ expands to all arguments individually quoted ("arg1" "arg2" etc).

What does the program do?

References

- Miscellaneous resources from internet
- Lecture notes from https://www.cs.cornell.edu/courses/cs2043/2014sp/

Thank you!