Indian Institute of Technology Jodhpur

Operating Systems Lab (CSL3030)

Dated 7th August 2022

Lab Guidelines:

1. **Lab attendance is mandatory. If you are absent from any lab, you need to provide valid justification or take prior approval. Absence from Lab without informing the instructor will get you a '0' credit for the corresponding evaluation.**
2. Submission of assignment on or before the deadline is mandatory. **For each day delay, 2 marks will be deducted from the assignment.**
3. Plagiarism of any form will not be tolerated and a serious penalty will be imposed if such a case is found. **If detected, you will get zero for that component without any reconsideration**.
4. Evaluation policy discussed in the theory class.
5. For Lab work (10%), every lab session will be evaluated (on a random basis). This component will include attendance, lab activities, viva and interaction with the TAs/instructors.

==========

UNIX Preliminaries and Shell

What is UNIX?

- Mainframe operating system
- Written by Dennis Ritchie, Ken Thompson and Rob Pike (among others) at Bell Labs
- The basis for many modern operating systems, e.g. Linux, BSD, Mac OSX
- Giant, yet simple. Practically runs on every hardware and provided inspiration to the Open Source movement.
- Unlike DOS and Windows, UNIX can be used by several users concurrently- multiuser and multitasking os.

History of UNIX

- Written at Bell Labs in 1969
- First version of BSD is installed in 1974
- Last Bell Labs UNIX (Version 7) is published in 1979
- UCB version called BSD UNIX. Others: Solaris, HP-UX, AIX and so on.
- The GNU Project is started by Richard Stallman
- Linux is a UNIX implementation that is constantly growing with contributions from the Free Software Foundation (formerly, GNU)
- Linus Torvalds writes a monolithic kernel operating system in 1991
- The most popular GNU/Linux flavors include Red Hat, Caldera, SuSE, Debian and Mandrake.

UNIX Philosophy:

"Everything is a file descriptor or a process." – Linus Torvalds

UNIX vs. Linux: a good read link: https://opensource.com/article/18/5/differences-between-linux-and-unix

UNIX Architecture:
The kernel and shell. The kernel interacts with the machine's hardware and the shell with the user.

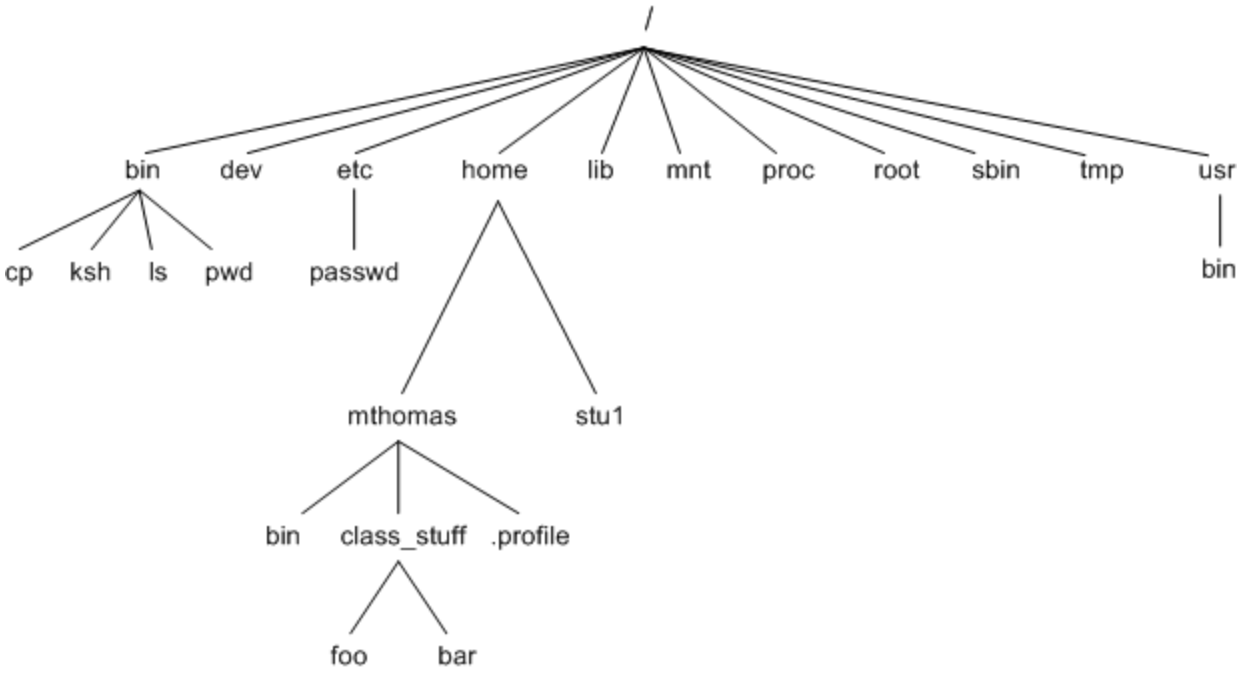
Kernel is the core of the operating system- a collection of routines mostly written in C. It is loaded in the memory when the system is booted and directly communicates with the hardware. User programs access the hardware through kernel by a set of function calls, called system calls.

Shell is the outer part of the os, basically an interface between user and kernel. Shell is a command interpreter that translated the commands into actions. There are several popular Unix shells: sh (the Bourne shell), csh (the C shell), and bash (the Bourne Again shell) are a few.

System administrator, security requirement and logging in with username-password
Command **passwd** and **uname**
/etc/passwd and /etc/shadow

Try basic commands: **date, tput clear, cal, who, ps, ls, cat, wc, echo, more, less**

The Files and Processes: "Files have places and processes have life"-- Kaare Christian.
A file is just an array of bytes and can contain virtually anything. Regular file, directory file and device file.
A process is the name given to a file when it is executed as a program. It is the 'time image' of an executable file.
[Refer Section 4.12 of Sumitabha Das for UNIX file system]



Permissions: UNIX permissions are represented with one bit for each permission in each category, e.g. 700. User(rwx)|Group(___)|Other(___)
Three Permissions I Read I Write I Execute
Three Categories I User I Group I Other
Relative and Absolute permission through **chmod** command
chmod ugo+x filex
chmod a+x filex
chmod +x file1, file2, file3
chmod 666 filex

The PATH variable.

The essential UNIX commands for general use are located in the directories /bin and /usr/bin.
Internal commands and external commands.
Combining commands with a semicolon.

[Section 2.5-2.11 for more details from Sumitabha Das]
What is a utility?

- A program used to process text streams/files
- Called from some command line/shell

Navigation utilities: pwd, ls, cd

File utilities: cp, mv, rm

General Utilities: cal, date, echo, printf (only for BASH), bc
Comparing two files: comm, cmp, diff

Alternatives to diff
Parsing output of diff is hard, so it might be useful to use some kind of “graphical diff” tool, like vimdiff, which opens up two files side-by-side in Vim showing their differences. Try it and you’ll see how much easier to parse this

grep

- Description
  - Print lines matching a pattern
- Synopsys
  - grep [OPTIONS] PATTERN [FILE]
- Examples
  - grep ’@@@’ program.out I ps -axfuw | grep "$USER" | grep "vcs"

Regular Expressions:

- Really powerful/useful
- Complicated, and beyond the scope of this presentation
- Read up on in the grep manual

man

- Description: An interface to the on-line reference manuals (pager)
- Synopsys: man PAGE
- Examples: man grep , man diff

Pager:

- Definition: A program which allows browsing of large text files by breaking them into screen-sized chunks.
- Examples: less, man

These will be useful. . . I cut I touch I tee I xargs I tail I column I find I less
These are harder, but even more useful. . . I sed I awk I patch I vi(m) I fmt I tmux

Text parsing using sed and awk
awk- finds and Replaces text, database sort/validate/index
awk 'pattern{action}' file
Example:
$awk ‘/Book/’ file
$awk -F, ‘$1 _ /Book/’ file
$ awk '/a/{++cnt} END {print "Count = ", cnt}' file.txt

Sed is a stream editor used to perform basic text transformations on an inputs stream.
Example:
$sed ‘s/hello/world/’ input.txt > output.txt
$sed ’30,35d’ input.txt > output.txt
$echo hello world | sed ‘y/abcdefghij/0123456789/’
$sed ‘s/[^,]*,//’ file /* to remove first field of a CSV file*/
$sed ‘s/^.//’ file /* to remove first character of every file*/

Program features:
Methods of Communication

- Standard Text Streams
  - stdin
  - stdout
  - stderr
- Return value/code

What is a return code?

- The integer value a program returns (e.g. return(0);)
- Conventionally, returning zero indicates success, non-zero failure
- Specific values other than zero mean different things for different programs

Standard Text Streams

- stdin: The default input to a program. From a keyboard by default
- stdout: The default output of a program. To a display by default
- stderr: The default error output of a program. To a display by default, requires additional effort to save

Connecting utilities:

- Why do we want to connect utilities?
  - Combination jobs without intermediate files
  - e.g. take the diff of two different grep operations (what you need to do for today’s lab)
- How can we connect utilities?
  - Pipes
  - Redirection

Pipes

What is a pipe?

- Connects the stdout of one program to the stdin of another
- Does not connect stderr

How do I use one?

- Call a program on one side of the | and then call another on the other side I

- e.g. dmesg | less

Problem: What if we want to pipe to utility that uses arguments instead of input?
Solution: xargs
xargs splits input into individual items and calls the program that is its argument once for each input.

Shell. command interpreter, is a program started up after opening of the user session by the **login** process. The shell is active till the occurrence of the <EOT> character, which signals requests for termination of the execution and for informing about that fact the operating system kernel.

Each user obtains their own separate instance of the *sh*. Program *sh* prints out the monitor on the screen showing its readiness to read a next command.

The shell interpreter works based on the following scenario:

1. displays a prompt,
2. waits for entering text from a keyboard,
3. analyses the command line and finds a command,
4. submit to the kernel execution of the command,
5. accepts an answer from the kernel and again waits for user input.

Bourne Again Shell

- What is BASH?
    - Stands for the Bourne Again Shell
    - Created in 1989 by Brian Fox
    - Default shell in most Linux distributions and Mac OSX

The behavior of *shell* simply repeats:

1. Displaying a prompt to indicate that it is ready to accept a next command from its user,
2. Reading a line of keyboard input as a command, and
3. Spawning and having a new process execute the user command.

How does the *shell* execute a user command? The mechanism follows the steps given below:

1. The *shell* locates an executable file whose name is specified in the first string given from a keyboard input.
2. It creates a child process by duplicating itself.
3. The duplicated shell overloads its process image with the executable file.
4. The overloaded process receives all the remaining strings given from a keyboard input, and starts a command execution.

The Shell Environment

There may be distinguished the following variables:

predefined variables,

positional and special parameters, relating to name and arguments of currently submitted command.

Exemplary variables of the **sh** shell:

HOME standard directory for the **cd** command,

IFS (ang. *Internal Field Separators*) used for word splitting after expansion and to split lines into words with the **read** built in command

MAIL mailbox file with alerting on the arrival of the new mail,

PATH a colon-separated search path for commands

PS1(prompt string 1), the primary prompt sting, under the **sh** shell: **$**, **PS2**

(prompt string 2), the secondary prompt string, under the **sh** shell: >, **SHELL** default

program to be run as a subshell,

TERM a terminal type name. Identifies a set of steering sequences appropriate for some particular terminal (exemplary names: *ansi*, *vt100*, *xterm*),

BASH Variables:

- Store data
- Contain text, for the most part No type system

Syntax

- Assignment/Declaration
  - variable=value
- Referencing
  - $variable
  - ${variable}

Variable Scope

- Variables exist inside the shell they're in
- Unless exported
  e.g. export EDITOR=vim
- This is very important in scripts, particularly the shell startup scripts
  (.bashrc, .bash_profile)

Special parameters: these parameters may only be referenced, direct assignment to them is not allowed.

$0 name of the command

$1 first argument of the scipt/ function

$2 second argument of the script/ function

$9 ninth argument of the scipt/ function

**$*** all positional arguments "$*" = "$1 $2 .."

**$@** list of separated all positional arguments "$@" = "$1" "$2" ..

$# the number of arguments of some commands or given to the last **set**,

$? exit status of the most recently executed foreground command,

$! PID of the most recently started backgruond command.

$$ PID of the current shell,

$0-9 also: may be set by the **set** command.

Metacharacters

During resolving of file names and grouping commands into bigger sets, special characters called metacharacters are used.

*	string without the "/" character,
?	any single character,
[ ]	one character from the given set,
[.....]	like [ ], with given scope from the first to the last,
[!.....]	any character except those within the given scope,
#	start of a comment,
\	escape character, preserves the literal value of the following character,
$	a value of a variable named with the following string,
;	commands separator,
‘ ‘	string in accent characters executed as a command with the stdout of the execution as a result of that quotation,
’ ’	preserves the literal value of each character within the quotes
" "	preserves the literal value of all characters within the quotes, with the exception of $, ‘, and \

Input/Output Redirection

After session opening user environment contains the following elements:

standard input (**stdin**) - stream 0,

standard output (**stdout**) - stream 1,

standard o error output (**stderr**) - stream 2.

There are the following redirection operators:

> file	redirect stdout to *file*
>> file	append stdout to *file*
< file	redirect stdin from *file*

<< EOT read input stream directly from the following lines, till EOT word occurence.
n > file redirect output stream with descriptor n to *file*,
n >> file append output stream with descriptor n to *file*,
n>&m redirect output of stream n to input of stream m,
n<&m redirect input of stream n to output of stream m.

Flow Control

IF: the standard structure of the compound

```
if if_list
    then then_list
    [ elif elif_list; then then_list ] ... [ else else_list ]
fi
```

the **if_list** is executed. If its exit status is zero, the **then_list** is executed. Otherwise, each **elif_list** is executed in turn, and if its exit status is zero, the corresponding **then_list** is executed and the command completes. Otherwise, the **else_list** is executed, if present.

```
if cc -c p.c

then ld p.o

else

  echo "compilation error" 1>&2 fi
```

CASE: the standard structure of the compound

```
case word in pattern1) list1;;
  pattern2) list2;;
*) list_default;;
esac
```

a **case** command first expands **word**, and tries to match it against each **pattern** in turn, using the same matching rules as for path-name expansion.

```
case $# in
  0) echo 'usage: man name' 1>&2; exit 2;;
```

Loop Instructions.

In the **sh** command interpreter there are three types of loop instructions:

```
for name [ in word ] ; do list ; done while list; do
        list; done
until list; do list; done
```

for instruction, executed once for each element of the for_list,

while instruction, with loop executed while the condition returns 0 exit

code (while condition is fulfilled),
until instruction, with loop executed until the condition finally returns 0
exit code (loop executed while condition is not fulfilled),
instructions **continue** and **break** may be used inside loops

```
#!/bin/sh
for i in /tmp /usr/tmp do
        rm -rf $i/*
done
```

Testing

- Testing happens in [ ]
- String tests are different than arithmetic tests
- Generally use -lt, -gt, -le, -ge, -eq, -ne
- Tests for files are special, e.g. [-x simv] makes sure that the binary simv has execution permissions

Functions:

- Packages of commands
- Arguments are referenced by position
- Useful for packaging up commonly reused bits
- Syntax:

```
function () {
        commands
}
```

Scripting

1. Write a series of shell commands into a text file
2. On the first line of the file, specify an interpreter. e.g. #!/bin/bash
3. Name it something appropriate
   e.g. test.sh
4. Add execute permissions
   e.g. chmod +x test.sh
5. Run the script
   e.g. $ ./test.sh

Some useful Links:

- https://courses.cs.washington.edu/courses/cse451/16au/readings/ritchie78unix.pdf
- https://www.gnu.org/software/sed/manual/sed.html
- https://www.theunixschool.com/p/awk-sed.html

Exercise (Deadline 14th August, 2022):

1. Write a shell script to find out the number of lines and words in a given file.
2. Write a shell script to translate the current date and time in word format.
3. Write a shell script to find out the three largest directories/files for a given directory.
4. Use awk to print 10 random random numbers between 0 and 100 inclusive
5. Write a command using sed to print first 5 lines of a file.