Shell Handout

Zilogic Systems

1. Processes

- Process is a program in execution.
- A process is always created by another process.
- When process X creates a new process Y, X is called the parent of Y.
- Each process is given a unique number called the process ID. The processes can be referred to using the process ID.
- `init` is the first program to be executed, and hence the first process. It has process ID of 1.
- The first user process, is usually the shell.

2. Listing Processes

- The `ps` command is used to list processes.

```
❶    ❷              ❸    ❹
  PID TTY          TIME CMD
 6423 tty1     00:00:00 bash
 6457 tty1     00:00:00 ps
```

❶ The process ID of the task.
❷ The terminal associated with the task.
❸ The total CPU time used by the process
❹ Command used to start the process.

- Without arguments, the `ps` command lists processes started by the user in the current terminal.
- The command to list all processes in the system the `-A` option can be used as shown below.

```
$ ps -A
```

- There options for filtering processes for filtering processes based on user, command, and process ID.
  - `ps -U <username-list>`
  - `ps -C <command-list>`
  - `ps -p <pid-list>`
- The parent child relation ship of processes can be displayed as tree, using the `pstree` command.

Try Out

- List all processes belonging to `root`.
- List all processes corresponding to the program `getty`.
- List the process tree of the system.

3. Resource Utilisation

- The `top` command can be used to watch the resource utilisation of processes.

- The `top` command is a full screen application. `top` displays, one screen full of processes and their information. Some of important fields are listed below
  - `PR` - priority of the process
  - `VIRT` - the amount of virtual memory used by the task
  - `RES` - the amount of physical memory used by the task
  - `SHR` - the amount of shared memory used by the task
  - `S` - the state of the process
  - `%CPU` - the task’s share of the elapsed CPU time since the last screen update
  - `%MEM` - the task’s currently used share of available physical memory.
- Key strokes within `top`.
  - `M` key stroke - Sort on memory usage
  - `P` key stroke - Sort on CPU usage
  - `q` key stroke - quit

Try Out

- Find out the process that occupies the greatest amount of memory in the system.
- Find out the process that uses the CPU the most.

4. Job Control

- Shell does not provide prompt till the last entered command is completed. Further commands cannot be entered till the previous command completes execution.
- If this behaviour is not desired, the program can be run in the background: `cmd &`
- Each command entered in the shell is given a job no. When a command is executed in the background the shell prints the shell prints the job no. and the process id.

```
$ sleep 10 &
[1] 11274
```

- Sometime it is required to background, a command that has already been started. This can be achieved by first stopping the job using `Ctrl-Z`, and then running `bg <job-no>`.
- Stopping a job freezes its execution, no instructions in the program are executed, till the process is continued, for example by the `bg` command.

```
$ sleep 10
^Z
[1]+ Stopped
$ bg %1
[1]+ sleep 10 &
```

- Usually job numbers are prefixed by a percentage character.
- We could end up with lots of stopped jobs, or jobs running in the background. In case we would like to know what jobs are stopped or running in the background, the `jobs` command can be used.
- A stopped job or a background-ed job can be brought to foreground using `fg <job-no>`.

5. Terminator!

- The job running in the foreground, can be immediately terminated using the `Ctrl-C` key stroke.

- Processes running in the background or in other terminals can be terminated using, the `kill` command. The command can be invoked as `kill <pid>` or `kill <job-no>`.
- Processes can also be killed by specifying the command name they were started with. But since there could be more than one processes started with the same command name, this could potentially kill more than one process. The command to terminate processes by command name is `killall` and the general syntax is `killall <command-name>`

Try Out

1. Suspend and resume process
   - Start `top`.
   - Press Ctrl-Z to suspend `top`
   - Resume `top` in the foreground.
2. Suspend and background process
   - Make a copy of `/usr/bin/inkscape` to your home directory.
   - Press Ctrl-Z during the copy, to suspend the copy.
   - Resume the command in the background.
3. Run in background
   - Make a copy of `/usr/bin/inkscape` in the background.
4. Killing processes
   - Start multiple copies in the background.
   - Kill a copy process with the kill command.
   - Check the file size to verify that the copy command terminated.

6. I/O Redirection

- Programs usually produce some output. By default the output goes to the terminal.
- The output can be saved to file by using the output redirection operator `>`. The following example, stores the output of `ls` to `myfile`.

```
$ ls > myfile
```

- Note that, when the output is redirected to a file, the original contents of the file will be lost.
- If instead the output should be appended to the existing contents of the file the `>>` redirection operator can be used.
- Apart from regular output, a command can print errors.
- To redirect errors prefix the output redirection operator with `2` as `2>` and `2>>`.

Try Out

- Store the output of `pstree` to a file.
- Perform a file copy using the `cat` command and redirection operators.
- Switch to the home directory.
- Join the 3 files `elements1.txt`, `elements2,txt`, `elements3.txt` into a single file using a single `cat` command and the redirection operator.
- Join the 3 files using multiple `cat` commands and the redirection with append operator.

7. Shell Variables

- Used to store information in memory for later use by the user or the shell.
- To set the value of variable

```
$ myvar=value
```

- Note that there is no space around the `=` sign.
- To retreive the value of the variable prefix the variable name with a `$` sign.

```
$ myvar2=$myvar1
```

- The `echo` command is used to print the string passed as arguments.
- To view the value of a variable, the `echo` command can be used.

```
$ echo $myvar
```

- TAB completion also works with shell variables, in `bash`.
- The values of the variables are stored internally as a string of characters.
- There are no integer variables.
- The variables are not preserved across re-boots.

Try Out

- Store the string `/usr/share/iceweasel/icons` in a variable `icons`
- List the contents of the directory using the variable.
- Copy the contents of the directory to `/tmp` using the variable.

8. Quoting

- Certain characters have special meaning to the shell, like `*`, `?`, `>`, `&`, `$`, etc.
- If a command contains these characters they will be specially interpreted by the shell.
- Quoting can be used to prevent the special interpretation of these characters.

Quoting a Single Character

- To remove the special meaning of single character, prefix the character by a `\`.
- Example to copy a file called `m&n`, the following command can be used.

```
$ cp m\&n.txt /tmp
```

Quoting a String of Characters

- To prevent the shell from interpreting special characters in a string of characters, surround the string by a single quote.
- The above example can be repeated with a single quote.

```
$ cp 'm&n.txt' /tmp
```

- The double quotes is similar to single quote, except that `$` retains its special meaning.
- To copy a file called `m&n-1.txt`, the following command can be used.

```
$ i=1
$ cp "m&n-$i.txt" /tmp
```

Try Out

- The directory `quoted` in the home directory contains files with special characters.
- Try copying each file to the directory `slash`, using the `\` quote.
- Try copying each file to the directory `single`, using the `'` quote.
- Try copying each file to the directory `double`, using the `"` quote.

9. Special Shell Variables

`PATH`. Specifies a list of directories in which the shell should look for commands. The directories are separated by a `:`.

```
$ echo $PATH
/usr/local/bin:/bin:/usr/bin
```

When the user types `ls` in the command line, the shell looks for `ls` in `/usr/local/bin`, `/bin` and `/usr/bin`. The first match is executed.

To add a directory to existing list of directories, the following command can be used.

```
$ PATH=$PATH:/path/to/new/directory ❶
$ PATH=/path/to/new/directory:$PATH ❷
```

❶ The directory is added to the end of the list.
❷ The directory is added to the start of the list.

Note that the directory added to `PATH` should be an absolute path.

Try Out

- Take a backup of your original `PATH` using

```
BPATH=$PATH
```

- There are two binaries `ls` and `hello` in `~/mybin`
- From some other directory invoke `hello`, the command will fail.
- Add the directory `mybin` to the end of `PATH`
- Invoke `ls` and `hello`.
- Add the directory `mybin` to the beginning of `PATH`.
- Invoke `ls` and `hello`.
- Restore your original `PATH`.

```
PATH=$BPATH
```

10. Scripting

- Hello World
  1. Hello world script

```
#!/bin/sh ❶

echo "Hello World" ❷
```

❶ Sha bang sequence
❷ Shell command

- Making it executable

```
chmod +x hello.sh
```

- Executing a shell script

```
$ ./hello.sh
```

Try Out

- Create a Hello World shell script.

Beyond Hello World

1. A script to create a backup of a directory

Backup script.

```
#!/bin/sh
                              ❶
zipfile="/tmp/backup-$(date +%d%m%y).zip" ❷
backdir="/home/vijaykumar/projects/smash"

zip -r $zipfile $backdir ❸
```

❶ The `$()` performs command substitution. The command is replaced by the commands output.
❷ The `date` command displays the current date in the specified format.
❸ The `zip` command is used create a ZIP file. The `-r` option specifies that directories are to be zipped recursively.

Looping Constructs

- The `for` statement executes the loop body once for each item in the list specified after `in`.
- The body for the `for` loop is enclosed between `do` and `done`.

`for` loop sample.

```
for x in 1 2 3
do
echo Hello $x
done
```

Sample code output.

```
Hello 1
Hello 2
Hello 3
```

A script to convert all .bmp files to .jpg files

Convert BMP files to JPEG files.

```
#!/bin/sh

for file in *.bmp
do
  file=$(basename $file .bmp) # ❶
  echo "Converting $file.bmp to $file.jpg ..."
  convert $file.bmp $file.jpg # ❷
done
```

❶ `basename` command removes displays filename without extension. The filename is the first argument and extension is the second argument.

❷ `convert` command can be used to convert between file types.

11. Further Reading

- Introduction to Linux: Processes: Processes inside out - http://www.tldp.org/LDP/intro-linux/html/sect_04_01.html
- Bash Reference Manual: Bash Features: Job Control - http://www.gnu.org/software/bash/manual/html_node/Job-Control.html#Job-Control
- FreeBSD Handbook: Unix Bascis: Processes and Daemons - http://www.freebsd.org/doc/handbook/basics-processes.html
- Debian Reference: GNU/Linux tutorials: Unix-like text processing - http://www.debian.org/doc/manuals/reference/ch-tutorial.en.html
- Bash Programming - Introduction HOWTO http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
- An Introduction to the Unix Shell by Steve Bourne. The author of the original bourne shell. http://partmaps.org/era/unix/shell.html