ICCS207: Term I/2018-19

Lecture 3: Process Management and Shell Scripting

Sunsern Cheamanunkul (sunsern.che@mahidol.edu)

Mahidol University

Process Management

- A process is a running instance of a program or a script, consisting of
  1. the program or script
  2. the corresponding environment, which consists of all required additional information required for the program to run correctly.

Process

- Each process has:
  - A unique process ID (PID)
  - PID of parent process (PPID)
  - User and group number of the owner
  - Priority of the process

ps

- Display running processes with their properties. Without options, only the user's own processes running in the current shell are displayed.

```
scheaman@hamachi:~$ ps
  PID TTY          TIME CMD
 1084 pts/0    00:00:00 bash
 1107 pts/0    00:00:00 bc
 1109 pts/0    00:00:00 bash
 1117 pts/0    00:00:00 bash
 1125 pts/0    00:00:00 ps
```

ps

- common usage: ps aux
  - a = show processes for all users
  - u = display the process's user/owner
  - x = also show processes not attached to a terminal

```
scheaman@hamachi:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 225768  6808 ?        Ss   Sep08   0:06 /sbin/init
root        52  0.0  0.7 226768 59532 ?        Ss   Sep08   0:03 /lib/systemd/
systemd-journald
root        62  0.0  0.0  42108  1876 ?        Ss   Sep08   0:01 /lib/systemd/systemd-
udevd
systemd+   165  0.0  0.0  80016  3736 ?        Ss   Sep08   0:02 /lib/systemd/
systemd-networkd
systemd+   168  0.0  0.0  70736  3900 ?        Ss   Sep08   0:02 /lib/systemd/
systemd-resolved
```


kill

- Sends a signal to processes
- Common usage:
  - Terminates the process with number PID. Can be executed only by the owner of the process or by root.

```
scheaman@hamachi:~$ kill 12313
```


kill

- Many times, the default signal TERM is not enough.

```
scheaman@hamachi:~$ kill -9 12313
```


Creating new process

- Need the path of the executable
  - Relative path

    `./runme.sh`
  - Absolute path

    /bin/ls
- When only the name is specified, the full path will be searched by prepending each PATH defined in $PATH to the name.

Creating new process

PATH=/usr/bin:/bin:/usr/sbin

/usr/bin /bin /sbin

Creating new process

$ zip

/usr/bin/zip /bin/zip /sbin/zip

Creating new process

- What if we have an executable call ‘zip’ in your current directory and want to run that?
- You have to specify the path to tell the shell not to search for it e.g.

$ ./zip

Executable

- Through shell, you could run/execute:
  - Valid binary files
    - compiled and built for the machine
  - Scripts e.g. Bash, Python
    - Shebang at the top tells the shell how to interpret/run the script
      - #!/bin/bash
      - #!/usr/bin/python
      - #!/usr/bin/env python
- The executable must have the “executable” flag turned on for the user.
  - You have use “chmod +x myscript.sh” to grant executable rights.

Shell Scripting

```
#!/bin/bash

echo “Hello” > test.tex
cp test.tex tex.test
cat tex.test
rm tex.test
```


Variable

- You can define local variables in a shell script.
- The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).

```
#!/bin/bash

greeting="Welcome"
user=$(whoami)
day=$(date +%A)

echo "$greeting back $user! Today is $day, which is the
best day of the entire week!"
echo "Your Bash shell version is: $BASH_VERSION. Enjoy!"
```


Another Example

```
#!/bin/bash

# This bash script is used to backup a user's home directory to /tmp/.

user=$(whoami)
input=/home/$user
output=/tmp/${user}_home_$(date +%Y-%m-%d_%H%M%S).tar.gz

tar -czf $output $input
echo "Backup of $input completed! Details about the output backup file:"
ls -l $output
```


Special Variables

- **$#** – represents the total number of arguments (much like argv)
- **$0** – represents the name of the script, as invoked
- **$1, $2, $3, .., $8, $9** – The first 9 command line arguments
- **$*** – all command line arguments
- **$@** – all command line arguments
- **"$@"** – all command line arguments, where each argument is individually quoted.
- **$?** – the exit status of the last program to exit

Quote

- unquoted strings are normally interpreted
- "quoted strings are basically literals -- but $variables are evaluated”
- 'quoted strings are absolutely literally interpreted’
- `commands in quotes like this are executed, their output is then inserted as if it were assigned to a variable and then that variable was evaluated`

Conditional Statements

Description	Numeric Comparison	String Comparison
less than	-lt	<
greater than	-gt	>
equal	-eq	=
not equal	-ne	!=
less or equal	-le	N/A
greater or equal	-ge	N/A
Shell comparison example:	*[100 -eq 50]; echo $?*	*["GNU" = "UNIX"]; echo $?*

Conditional Statements

```
#!/bin/bash

num_a=400
num_b=200

if [ $num_a -lt $num_b ]; then
    echo "$num_a is less than $num_b!"
else
    echo "$num_a is greater than $num_b!"
fi
```


Conditional Statements

```
if [ "$#" -ne 1 ]; then
    echo "Illegal number of parameters"
fi
```

```
if [ "$@" == "your string" ]; then
    echo "YES"
else
    echo "NO"
fi
```

For Loop

```
#!/bin/bash

for i in 1 2 3; do
    echo $i
done
```


While Loop

```
#!/bin/bash

counter=0
while [ $counter -lt 3 ]; do
    let counter+=1
    echo $counter
done
```


Q: What does it do?

```
linuxconfig.org:~$ vi items.txt
linuxconfig.org:~$ cat items.txt
bash
scripting
tutorial
linuxconfig.org:~$ for i in $( cat items.txt ); do echo -n $i | wc -c; done
4
9
8
linuxconfig.org:~$
```


References

- https://linuxconfig.org/bash-scripting-tutorial-for-beginners
- https://www.learnshell.org/