CST8177 – Linux II

Processes

Todd Kelley
kelleyt@algonquincollege.com

Topics

- elinks, mail
- processes
- nice
- ps, pstree, top
- job control, jobs, fg, bg
- signals, kill, killall
- crontab, anacron, at

elinks

- elinks is a text-based (character mode) web browser
- we will use it to enable our scripts to retrieve web pages
- in assignment 7, we use it to retrieve a weather webpage
- `elinks -dump -no-numbering -no-references <URL>`
- Example

```
elinks -dump -no-numbering -no-references \
'http://m.weather.gc.ca/city/pages/on-118_e.html'
```

- Could grep this to extract information (maybe with -A option)

mail command

- use the `mail` command to send outgoing and read incoming email on the CLS
- Sending outgoing email (bold font shows what the user types)

```
$ mail username@example.com
Cc:
Subject: First Message from CLS
This is a test message.
^D
$
```

reading mail

- text mode mail reader
- incoming email is stored in

  `/var/spool/mail/<username>`
- use the `mail` command to read it
- you'll see a list of messages, each preceded by a number (the header list)
- enter a number to see that message
- enter `h` to see the header list again
- when you enter `q`, mail will quit and messages you read will be stored in `~/mbox`
- `mail -f` to see the messages in `~/mbox`

Processes

- Any program we run executes as a process
- Processes have the following attributes
  - a process id: PID
  - a parent process id: PPID
  - a nice number (related to priority)
  - controlling terminal
  - Real (RUID) and effective (EUID) user id
  - Real (RGID) and effective (EGID) group id
- Also:
  - a current working directory
  - a umask value
  - an environment (values of environment variables)

ps command

- We have already been using the ps command to print out information about processes running on the system
- ps –ef or ps aux piped to grep is common
- there are many options for printing specific info in a specific way: man ps or ps –h
- ps –l # long format
- ps –f versus ps –fw

top command

- top displays some system information, and a list of processes, ordered on a column
- the most important keys are ?, h, and q (according to man page)
- load average: 5min, 10min, 15min
- load average is number of processes running or in uninterruptable state (disk IO, others)
- no exact rule, but if load average is more than 1-1.5 times the number of CPUs, the machine is overloaded in some way and you have a problem (your mileage may vary)

Other commands

- pstree: connects parents and children in a pictorial display
- free: memory usage
- vmstat: processes, memory, and more

Process states

- Runnable: ready to go
- Sleeping: choosing not to go
- Stopped: suspended indefinitely, as in ^Z
- Uninterruptable Sleep: waiting on a disk I/O operation, or similar
- Zombie or Defunct: process has completed, but it's still in the process table waiting for parent to take action

Nice command

- Each process has a priority, which you can control with the nice command
- -20 highest priority, 19 lowest priority
- nice [-n increment] command
- nice -n 10 long_command # 10 is default
- only superuser can specify negative increments
- For processes already running:
  - renice priority -p PID or renice -n increment -p PID

Job Control

- your shell can run several processes for you at once
- we can run commands in the background
  - command &
- we can put a running command in the background
  - ^Z
- what jobs are there?
  - jobs
- resume a stopped job
  - bg %N # background, where N is a job number
  - fg %N # foreground

Sending signals: kill command

- When we type ^C when a process is running in the foreground, the process receives a SIGINT signal, which by default would cause a process to terminate.
- SIGINT: ^C (default), similar to SIGTERM
- SIGHUP: terminal has been closed
- SIGTERM: clean up if necessary, then die
- SIGKILL: die right now
- We can send these signals to a process with the kill command

Send a signal to kill a process

- kill -SIGNAL PID #send SIGNAL to process PID
- When system shuts down, it
  - sends all processes a SIGTERM
  - waits a few seconds (5 or 10)
  - sends all processes a SIGKILL
- Why not just wait for the SIGTERM to finish?
- Because SIGTERM can be handled, possibly ignored, it's optional
- SIGKILL cannot be handled – it works unless the process is in an uninterruptible state (maybe disk I/O, NFS)

When kill -9 PID doesn't work

- If kill -9 PID (kill -SIGKILL PID) as root doesn't kill the process, it is in an uninterruptible state
- if uninterruptible processes don't become interruptible, there may be a system problem (bad disk, misconfigured NFS filesystem, etc)
- Reboot may be the only way to get rid of them

What are the other signals?

- summary of all the POSIX signals: http://en.wikipedia.org/wiki/Unix_signal

Scheduling tasks (cron)

- To run a command regularly and automatically, we use the cron facility
- The cron daemon process every minute checks to see if commands specified in crontab files need to be run
- for now, we're concerned only with our user crontab files, which are
  - /var/spool/cron/*
  - for example, /var/spool/cron/user1 is user1's crontab file

Configuring your cron job

- full details from man 5 crontab
  - recall that is how we read section 5 of the manual (section 5 of the manual is file formats)
- man crontab will give info about the crontab command (in default section 1 of the manual)
- create a file containing your cron instructions (see next slide), naming that file, say, myuser.crontab
- run the crontab command to submit that file's contents to be your user's crontab file: crontab < myuser.crontab
- alternatively, you can edit your user's live crontab file: crontab -e

crontab format (man 5 crontab)

- All fields must contain a value of some valid kind
- Field are separated by one or more spaces
- Asterisk (*) indicates the entire range

```
# .---------------- minute (0 - 59)
# |   .------------- hour (0 - 23)
# |   |   .--------- day of month (1 - 31)
# |   |   |   .------ month (1 - 12)
# |   |   |   |   .--- day of week (0 – 7, both 0 and 7 are Sunday)
# |   |   |   |   |
  0  6   1  *  *  /home/user/bin/mycommand
  1  6  15  *  *  /home/user/bin/anothercommand > /dev/null 2>&1
```

crontab format (cont'd)

- ranges with dash are allowed: first-last
- * means every value first-last
- lists are allowed: first,second,third
- steps indicated with '/' are allowed after ranges or asterisk:
  - */2 means every second one
  - 1-7/2 means 1,3,5,7

common crontab options

- crontab -l
  - list the contents of your current live crontab file
- crontab -e
  - edit the contents of your current live crontab file
- crontab
  - read the new contents of for your crontab file from stdin
- crontab -r
  - remove your current crontab file

example crontab

- see man 5 crontab for example crontab
- really, see the example: man 5 crontab
- things to watch out for
  - input for your commands (they run without anyone to type input)
  - output of commands (if you don't (re)direct output, the output will be emailed – better if you handle it)
  - error output of commands (same as for output above)
  - summary: it's best if your commands in a crontab are arranged with input and output already handled, not relying on output to be emailed by cron
  - if you want to email, do it explicitly in your command somehow, and test that command before putting it into your crontab

at command

- at command runs a set of commands at a later time
- at command takes a TIME parameter and reads the set of commands from standard input
- example (run commands at 4pm 3 days from now)
  - at 4pm + 3 days
    rm -f /home/usr/foo
    touch /home/usr/newfoo
    ^D
- other at-related commands: atrm, atq
- for details: man at
- as with cron, you must be aware of how your commands will get their input (if any) and what will happen to their output (if any)