

CST8177 – Linux II

Review of Fundamentals

Todd Kelley
kelleyt@algonquincollege.com

Topics

- ▶ The CST8207 course notes
- ▶ GPL
- ▶ the shell
- ▶ SSH (secure shell)
- ▶ the Course Linux Server
- ▶ RTFM
- ▶ vi
- ▶ general shell review

CST8207 course notes

- ▶ Linux I introduces many fundamental topics to give you a good basic foundation
- ▶ In Linux II we build on that same foundation so it will be good to refer back to the course notes:

<http://teaching.idallen.com/cst8207/13f/notes/>

Especially these

- ▶ 120_shell_basics.html
- ▶ 140_man_page_RTFM.html
- ▶ 150_arguments_and_options.html
- ▶ 160_pathnames.html
- ▶ 170_home_and_HOME.html
- ▶ 180_finding_files.html
- ▶ 185_find_and_xargs.html
- ▶ 190_glob_patterns.html
- ▶ 200_redirection.html
- ▶ 300_vi_text_editor.html
- ▶ 320_shell_variables.html
- ▶ 350_startup_files.html
- ▶ 400_search_path.html

And these

- ▶ 440_quotes.html
- ▶ 450_file_system.html
- ▶ 455_links_and_inodes.html
- ▶ 457_disk_usage.html
- ▶ 460_symbolic_links.html
- ▶ 500_permissions.html
- ▶ 510_umask.html
- ▶ 520_package_management.html
- ▶ 580_system_log_files.html
- ▶ 600_processes_and_jobs.html
- ▶ 630_crontab_at_job_scheduler.html
- ▶ 700_users_and_groups.html
- ▶ 720_partitions_and_file_systems.html
- ▶ 750_booting_and_grub.html
- ▶ 900_unix_command_list.html

Linux Licensing: GPL (for example)

- ▶ You should be aware that we all use GNU and Linux (and other Free software) *under license*
- ▶ Q: who cares? A: your employer
- ▶ When you receive a copy of GPL software, you are automatically granted a license from the copyright holder, and you have obligations
- ▶ Roughly, If you don't give copies to others, no worries
- ▶ Roughly, If you give copies to others
 - 1. You must give the source code along with binary; OR
 - 2. You must provide a written offer to provide source code; OR
 - 3. for other special conditions or possibilities, read the GPL

Again, who cares?

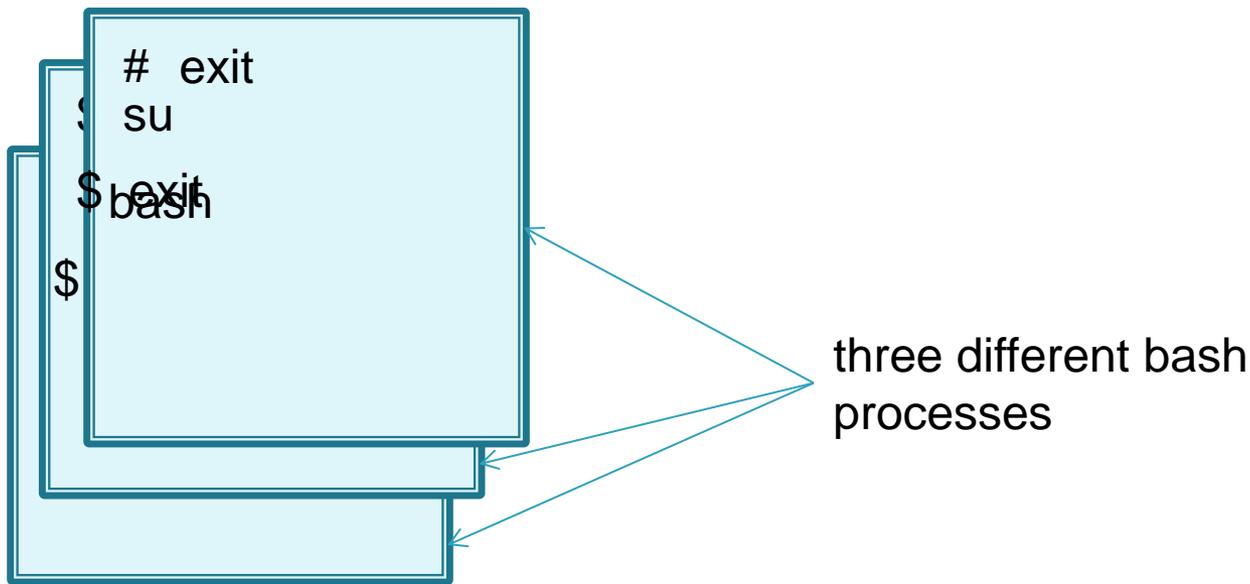
- ▶ When you get a job, it will be incredibly important that your employer (through your work for them) is not found to be out of compliance with the GPL
- ▶ It gets serious when you (on behalf of your employer through their facilities) provide copies of software to others because you may inadvertently deny those others some rights
- ▶ Do not consider this legal advice: when/if the time comes, consult your employer's legal department
- ▶ **“We always considered Open Source software to be a free-for-all under all circumstances. Why didn't anyone warn us?”** -- I just did. That is all this was for.

The shell

- ▶ http://teaching.idallen.com/cst8207/13f/notes/120_shell_basics.html
- ▶ The shell is a program that is executed for us automatically when we log in, and we control by typing text for it to read
- ▶ Normally we are asking the shell to run programs for us on certain arguments, which we also type
- ▶ This is the *command line*
- ▶ Basically, the process is this
 1. The shell prints a prompt on our terminal or terminal emulator screen (“screen”)
 2. We type a command and “enter” (“return”)
 3. The shell reads what we typed, interpreting special characters like space, GLOB characters, quotation marks, etc
 4. The shell carries out the operation in the way we asked (we might see output from that operation on our screen)
 5. repeat at step 1.

Sub-shells

- ▶ When you invoke `bash`, or `su`, or `sudo -s` you begin talking to a *sub-shell*
- ▶ schematically for illustration:



Sub-shells (cont'd)

- ▶ closer to what we actually see :

```
$ bash
bash
$ su
password:
# exit
exit
$ exit
exit
$
```

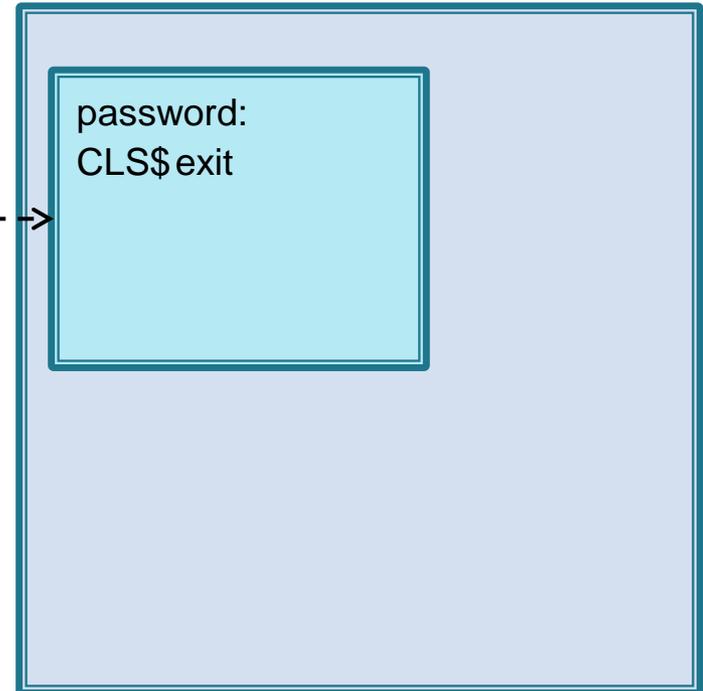
SSH and the Course Linux Server(CLS)

- ▶ all the details:
http://teaching.idallen.com/cst8207/14w/notes/070_course_linux_server.html
- ▶ SSH (secure shell) is a program that allows us to securely invoke a shell *on a remote computer*
- ▶ On Windows: putty.exe
- ▶ schematically (abbreviated):

a terminal window on your
local computer

```
$ ssh user@cst8177.idallen.ca
$
```

cst8177.idallen.ca (remote computer)



SSH and CLS (cont'd)

- ▶ what we'd see locally (abbreviated)

local terminal window

```
$ ssh user@cst8177.idallen.ca  
password:  
CLS $ exit  
$
```

SSH to CLS

- ▶ http://teaching.idallen.com/cst8207/14w/notes/070_course_linux_server.html
- ▶ cst8177-alg.idallen.ca represents an *internal* IP address that works only on campus: when on campus, use this one
- ▶ cst8177.idallen.ca must be used when off campus
- ▶ login id is your algonquin userid
- ▶ password is given verbally by your Prof(s) or another student
- ▶ Change your password at your first opportunity (if you haven't already)
- ▶ If you have firewalled internet access, you might try connecting to the CLS on Port 443 with
 - `ssh -p 443 cst8177.idallen.ca`

Editing a text file

▶ Text Editors

- Windows
 - notepad, wordpad (gui required)
- Unix
 - vi (vim), emacs, nano, pico
 - gedit (gui required – no good for CLS)
- ▶ You need to be able to edit text files without a GUI
 - start the editor
 - move around
 - make a change
 - save and quit
- ▶ **Vi:** http://teaching.idallen.com/cst8207/13f/notes/300_vi_text_editor.html
- ▶ long into the CLS and issue the command, vimtutor

Looking things up

- ▶ http://teaching.idallen.com/cst8207/13f/notes/140_man_page_RTFM.html
- ▶ It's normal for Unix users of all kinds (novice to expert) to consult the manual (man pages) often.
- ▶ `$ man man`
 - Read the man page for the man command
- ▶ `$ man -k listing`
 - Print out man page titles that include the text `listing`
 - Try using this command with the text `list` instead of `listing`
 - What did you notice?
 - less detail in your search terms means more search results
 - More detail in your search terms gives less search results

Looking things up (cont'd)

- ▶ It's a required skill to be able to find information in technical documentation, (“grep-ing through documents”)
- ▶ We want you to get lots of practice looking things up
 - You get better and faster at looking things up the more you do it
 - Knowing where to look and what to look for
 - You get the answer you were looking for and acquire the knowledge
- ▶ Here's the normal process when you encounter a concept that you don't know or it's become vague or you've forgotten
 - search for the term in the manual
 - Often you'll get too much information, including information that is much more advanced than you need – that's normal, use the search facility
 - search for the term in the course notes
 - All of the CST8207 course notes are available in text form on the CLS
 - search for the term on the web (be careful)
 - ask your professor or lab instructor
 - This includes situations where you have trouble with any of the above!

Commands, programs, scripts, etc.

Command

A directive to the shell typed at the prompt. It could be a utility, a program, a built-in, or a shell script.

Program

A file containing a sequence of executable instructions. Note that it's not always a binary file but can be text (that is, a script).

Script

A file containing a sequence of text statements to be processed by an interpreter like bash, Perl, etc.

Every program or script has a stdin, stdout, and stderr by default, but they may not always be used.



Filter

A program that takes its input from stdin and send its output to stdout. It is often used to transform a stream of data through a series of pipes.

Scripts are often written as filters.

Utility

A program/script or set of programs/scripts that provides a service to a user. (ls, grep, sort, uniq, many many more)

Built-in

A command that is built into the shell. That is, it is not a program or script as defined above. It also does not require a new process, unlike those above.



History

A list of previous shell commands that can be recalled, edited if desired, and re-executed.

Token

The smallest unit of parsing; often a word delimited by white space (blanks or spaces, tabs and newlines) or other punctuation (quotes and other special characters).

stdin

The standard input file; the keyboard; the file at offset 0 in the file table.

stdout

The standard output file; the terminal screen; offset 1 in the file table.



stderr

The standard error file; usually the terminal screen; offset 2 in the file table.

Standard I/O (Numbered 0, 1, and 2, in order)

stdin, stdout, **and** stderr

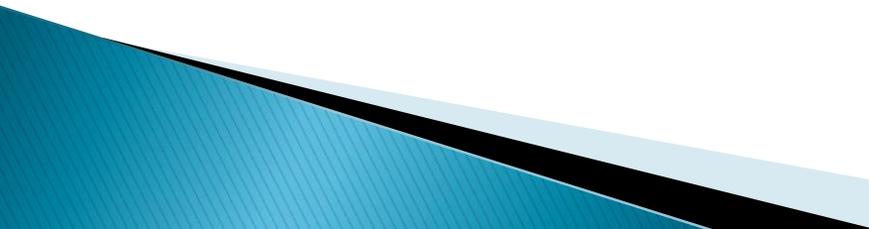
Pipe

Connects the stdout of one program to the stdin of the next; the "|" (pipe, or vertical bar) symbol.

A command line that involves this is called a pipeline

Redirect

To use a shell service that replaces stdin, stdout, or stderr with a regular named file.



Process

http://teaching.idallen.com/cst8207/13f/notes/770_processes_and_jobs.html

- **A process is what a script or program is called while it's being executed. Some processes (called daemons) never end, as they provide a service to many users, such as crontab services from crond.**
- **Other processes are run by you, the user, from commands you enter at the prompt. These usually run in the foreground, using the screen and keyboard for their standard I/O. You can run them in the background instead, if you wish.**
- **Each process has a PID (or pid, the process identifier), and a parent process with its own pid, known to the child as a ppid (parent pid). You can look at the running processes with the ps command or examine the family relationships with pstree.**
- **Example: print out a full-format listing of all processes:**

```
ps -ef
```

Child process

- Every process is a child process, with the sole exception of process number 1 – the init process.
- A child process is forked or spawned from a parent by means of a system call to the kernel services.
- Forking produces an exact copy of the process, so it is then replaced by an exec system call.
- The forked copy also includes the environment variables and the file table of the parent.
- This becomes very useful when redirecting standard I/O, since a child can redirect its own I/O without affecting its parent.
- Each non-builtin command is run as a child of your shell (builtins are part of the shell process: man builtin).

History

•The command history is a list of all the previous commands you have executed in this session with this copy of the shell. It's usually set to some large number of entries, often 1000 or more.

•Use `echo $HISTSIZE` to see your maximum entries

•You can reach back into this history and pull out a command to edit (if you wish) and re-execute.

•To make the history of all your simultaneous sessions is captured, do

```
shopt -s histappend
```

In your `.bashrc`

Some history examples

- **To list the history:**

System prompt> history | less

- **To repeat the last command entered:**

System prompt> !!

- **To repeat the last ls command:**

System prompt> !ls

- **To repeat the command from prompt number 3:**

System prompt> !3

- **To scroll up and down the list:**

Use arrow keys

- **To edit the command:**

Scroll to the command and edit in place

Redirection

- **Three file descriptors are open and available immediately upon shell startup: stdin, stdout, stderr**
 - **These can be overridden by various redirection operators**
 - **Following is a list of most of these operators (there are a few others that we will not often use; see man bash for details)**
 - **If no number is present with > or <, 0 (stdin) is assumed for < and 1 (stdout) for >; to work with 2 (stderr) it must be specified, like 2>**
- 

Operator

Behaviour

Individual streams

< filename

Redirects stdin from filename

> filename

Redirects stdout to filename

>> filename

Appends stdout **onto** filename

2> filename

Redirects stderr to filename

2>> filename

Appends stderr **onto** filename

Combined streams

&> filename

Redirects both stdout **and** stderr to filename

>& filename

Same as &>, **but do not use**

&>> filename

Appends both stdout **and** stderr **onto** filename

>>& filename

Not valid; produces an error

Operator

Behaviour

Merged streams

2>&1

Redirects stderr to the same place as stdout, which, if redirected, must already be redirected

1>&2

Redirects stdout to the same place as stderr, which, if redirected, must already be redirected

**Special stdin processing ("here" files),
mainly for use within scripts**

<< string

Read stdin using string as the end-of-file indicator

<<- string

Same as <<, but remove leading TAB characters

<<< string

Read string into stdin

Command aliases

- **To create an alias (no spaces after alias name)**

```
alias ll="ls -l"
```

- **To list all aliases**

```
alias or alias | less
```

- **To delete an alias**

```
unalias ll
```

- **Command aliases are normally placed in your `~/.bashrc` file (first, make a back-up copy; then use `vi` to edit the file)**

- **If you need something more complex than a simple alias (they have no arguments or options), then write a bash function script (that topic is coming soon).**

Filename Globbing and other Metacharacters

Metacharacter

Behaviour

\

Escape; use next char literally

&

Run process in the background

;

Separate multiple commands

\$xxx

Substitute variable xxx

?

Match any single character

*

Match zero or more characters

[abc]

Match any one char from list

[!abc]

Match any one char not in list

(cmd)

Run command in a subshell

{cmd}

Run in the current shell

Simple Quoting

- **No quoting:**

```
System Prompt$ echo $SHELL  
/bin/bash
```

- **Double quote: "**

```
System Prompt$ echo "$SHELL"  
/bin/bash
```

- **Single quote: '**

```
System Prompt$ echo '$SHELL'  
$SHELL
```

Observations:

Double quotes allow variable substitution;

Single quotes do not allow for substitution.

Quoting

- ▶ Backslash `\` removes the special meaning of the special character that follows it
- ▶ Single quotes remove the special meaning from all special characters
 - Cannot include single quote inside single quotes not even with backslash
- ▶ Double quotes remove the special meaning from special characters, except `!` `$` `\` ```
 - This means history, variable expansion, command substitution and backslash escaping all work inside double quotes

Escape and Quoting

- Escape alone:**

```
Prompt$ echo \$SHELL  
$SHELL
```

- Escape inside double quotes:**

```
Prompt$ echo "$SHELL"  
$SHELL
```

- Escape inside single quotes:**

```
Prompt$ echo '$SHELL'  
\$SHELL
```

Observations:

Escape leaves the next char unchanged;

Double quotes obey escape (process it);

Single quotes don't process it (treat literally)

Filespecs and Quoting

System Prompt\$ ls

a b c

System Prompt\$ echo *

a b c

System Prompt\$ echo "*"

*

System Prompt\$ echo '*'

*

System Prompt\$ echo *

*

Observation:

Everything prevents file globs

Backquotes and Quoting

```
System Prompt$ echo $(ls) # alternate  
a b c
```

```
System Prompt$ echo `ls` # forms  
a b c
```

```
System Prompt$ echo "`ls`"  
a  
b  
c
```

```
System Prompt$ echo `ls`  
`ls`
```

Observations:

Single quotes prevent command processing

Summary so far

Double quotes allow variable substitution

"\$SHELL" becomes /bin/bash

Single quotes do not allow for substitution

'\$SHELL' becomes \$SHELL

Escape leaves the next char unchanged

\\$SHELL becomes \$SHELL

Double quotes obey escape (process it);

"\\$SHELL" becomes \$SHELL

Single quotes don't process it (treat it literally)

'\\$SHELL' becomes \\$SHELL

Everything prevents file globs

"*" '*' * each become *

Single quotes prevent command processing

`|s` becomes `|s`

Escaping quotes

```
System Prompt$ echo ab"cd
```

```
> "
```

```
abcd
```

```
System Prompt$ echo ab\"cd
```

```
ab"cd
```

```
System Prompt$ echo 'ab\"cd'
```

```
ab\"cd
```

```
System Prompt$ echo "ab"cd"
```

```
> "
```

```
abcd
```

More quote escapes

```
System Prompt$ echo "ab\"cd"  
ab"cd
```

```
System Prompt$ echo don't  
> '  
dont
```

```
System Prompt$ echo don\t  
don't
```

```
System Prompt$ echo "don't"  
don't
```

```
System Prompt$ echo 'don't'  
> '  
dont
```

Observations

Unbalanced quotes cause a continuation prompt

Unescaped quotes are removed (but their meaning is applied)

"hello" becomes hello

"\$HOME" becomes /home/username

Quoting protects quotes, as does \ escaping

"don't" and don\t are the same, and OK

Single quotes are more restrictive than double

System Prompt\$ echo '\$USER' "\$USER"

\$USER someusername



(In class) Warm-up for next lecture:

Add a Unix command

- create a simple shell script
- make it executable
- copy it to a directory that is in our \$PATH
- presto, we have extended UNIX