
CSE 391

Lecture 3

bash shell continued:
I/O redirection, streams, xargs, processes

slides created by Marty Stepp, modified by Jessica Miller and Ruth Anderson

<http://www.cs.washington.edu/391/>

Lecture summary

- A bit more on combining commands
- Processes and basic process management

File examination

command	description
cat	output a file's contents on the console
more or less	output a file's contents, one page at a time
head, tail	output the first or last few lines of a file
wc	count words, characters, and lines in a file
du	report disk space used by a file(s)
diff	compare two files and report differences

- Let's explore what we can do here...

Searching and sorting

command	description
grep	search a file for a given string (useful options: <code>-v</code> and <code>-i</code>)
sort	convert an input into a sorted output by lines
uniq	strip duplicate (adjacent) lines
find	search for files within a given directory
locate	search for files on the entire system
which	shows the complete path of a command
cut	extract a column of text from a file

- `grep` is actually a very powerful search tool; more later...
- *Exercise* : Display the contents of the text file names `.txt` in alphabetical order.

Review: Redirection and Pipes

- ***command* > *filename***
 - Write the output of ***command*** to ***filename*** (>> to append instead)
- ***command* < *filename***
 - Use ***filename*** as the input stream to ***command***
- ***command1* | *command2***
 - Use the console output of ***command1*** as the input to ***command2***
- ***command1* ; *command2***
 - Run ***command1*** and then run ***command2***
- ***command1* && *command2***
 - Run ***command1***, if completed without errors then run ***command2***

Streams in the Shell

- Stdin, stdout, stderr
 - These default to the console
 - Some commands that expect an input stream will thus read from the console if you don't tell it otherwise.
- *Example:* `grep hi`
 - What happens? Why?

We can change the default streams to something other than the console via redirection.

Output redirection

command > *filename*

- run *command* and write its output to *filename* instead of to console;
 - think of it like an arrow going from the command to the file...
 - if the file already exists, it will be overwritten (be careful)
- >> appends rather than overwriting, if the file already exists
- *command* > /dev/null suppresses the output of the command

- Example: `ls -l > myfiles.txt`
- Example: `java Foo >> Foo_output.txt`
- Example: `cat > somefile.txt` (writes console input to the file until you press ^D)

- *Exercise* : List the vegetables in `veggies.txt` in alphabetical order in the file `sortedVeggies.txt`.

Input redirection

command < *filename*

- run *command* and read its input from *filename* instead of console
 - whenever the program prompts the user to enter input (such as reading from a Scanner in Java), it will instead read the input from a file
 - some commands don't use this; they accept a file name as an argument
- again note that this affects *user input*, not *parameters*
- useful with commands that can process standard input or files:
 - e.g. `grep`, `more`, `head`, `tail`, `wc`, `sort`, `uniq`, `write`
- Example: `java Guess < input.txt`
- *Exercise* : run `HelloWorld` taking input from `names.txt`
- *Exercise* : run it again and write the output to `output.txt`

Tricky Examples

- Amongst the top 250 movies in movies.txt, display the third to last movie that contains "The" in the title when movie titles are sorted.
- The wc command can take multiple files: wc names.txt student.txt
 - Can we use the following to wc on every txt file in the directory?
 - `ls *.txt | wc`
- Find the disk space usage of the man program
 - Hints: use which and du...
 - Does `which man | du` work?

Answers posted in lecture_commands.txt after lecture

Command Substitution

command1 \$(*command2*)

- run *command2* and pass its console output to *command1* as a parameter;
- best used when *command2*'s output is short (one line)

- Finish the example!
 - du \$(which man)

xargs

command	description
xargs	run each line of input as an argument to a specified command

- xargs allows you to repeatedly run a command over a set of lines
 - often used in conjunction with `find` to process each of a set of files
- Example: Remove all my `.class` files from my home directory (and its subdirectories).

```
find ~ -name "*.class" | xargs rm
```
- Find the disk usage of `man` using `xargs`
 - `which man | xargs du`

tee

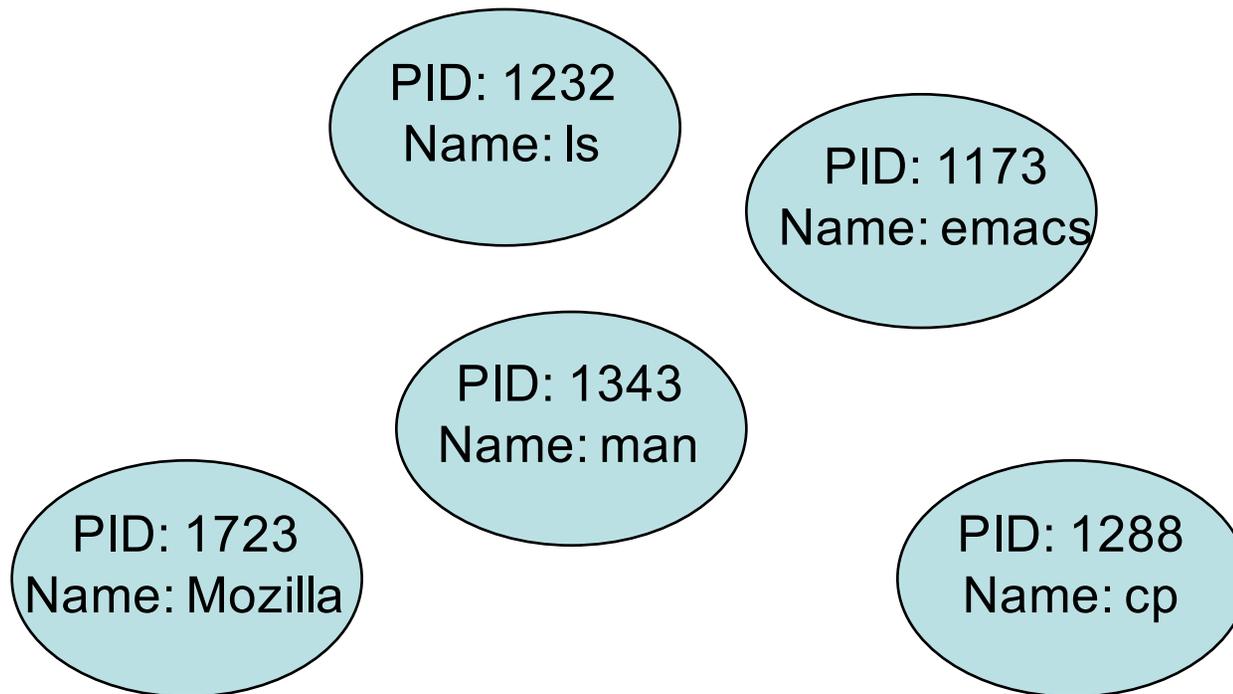
command	description
tee	copies standard input to standard output (similar to cat) and also outputs to one or more files

- Example: Run the HelloWorld java program, and produce output to the console as well as to a file.

```
Java HelloWorld | tee output.txt
```

Processes

- **process:** a program that is running (essentially)
 - when you run commands in a shell, it launches a process for each command
 - Process management is one of the major purposes of an OS



Process commands

command	description
<code>ps</code> or <code>jobs</code>	list processes being run by a user; each process has a unique integer id (PID)
<code>top</code>	show which processes are using CPU/memory; also shows stats about the computer
<code>kill</code>	terminate a process by PID (sometimes <code>-KILL</code> is needed)
<code>killall</code>	terminate several processes by name

- use `kill` or `killall` to stop a runaway process (infinite loop)
 - similar to `^C` hotkey, but doesn't require keyboard intervention

Background processes

command	description
&	(special character) when placed at the end of a command, runs that command in the background
^Z	(hotkey) suspends the currently running process
fg, bg	resumes the currently suspended process in either the foreground or background

- On the VM, if you run a graphical program like `gedit` or `emacs` from the shell, the shell will lock up waiting for the graphical program to finish
 - instead, run the program in the background, so the shell won't wait:
`$ emacs homework1.txt &`
 - if you forget to use `&`, suspend `emacs` by hitting `^Z` at the TERMINAL (NOT in `emacs`), then run `bg` from the terminal
 - lets play around with an infinite process...

screen

command	description
screen	A program that allows you to run multiple shells at once

- Create a new screen session
 - screen
- Creates a new screen session with the given name
 - screen -S session_name
- Reattach to a session
 - screen -r [optional-screen-id]
- Detach from a screen session with Ctrl+a d

screen

- When you create a new screen, it will open a new window with a new shell that looks very similar like the window you just came from.
- `Ctrl+a ?` will list commands that may be used within a screen
- Using screen is similar to having processes run in the background. When you are in a screen session, if you run a command, you can detach from the screen session, returning to your original window and your command will continue to run.
- If you create a screen session on attu then log off attu, your screen session will still be there when you return