CSE 391

Shell commands
Streams, Redirection

Slides created by Josh Ervin. Based off slides made by Marty Stepp, Jessica Miller, Ruth Anderson, Brett Wortzman, and Zorah Fung

pollev.com/cse391

- How was the last homework assignment?

AGENDA

- Logistics
- Useful shell commands (wc, more, less, grep)
- Standard in, Standard out
- Input/output redirection
- Pipes

MISC

- During lecture last week, one of the questions we asked was “Do you find command line tools intimidating?”
  - **51%** - Yes, and I feel like other people know more than me
  - **40%** - A little, I’m not very comfortable with the command line
  - **9%** - No, I feel comfortable with the command line
- If you have a question, there are other people that have that question as well!

FILE EXAMINATION

Command	description
`cat`	Print contents of a file
`less`	Output file contents, one page
`more`	Output file contents, one page
`head`	Output number of lines of start of file
`tail`	Output number of lines of end of ile
`wc`	Count words, characters, lines in a file

SEARCHING AND SORTING

Command	description
`grep`	Search given file for pattern
`sort`	Sort input or file, line based
`uniq`	Strip duplicate **adjacent lines**
`find`	Search filesystem
`cut`	Remove section from each line of file

JAVA AND THE COMMAND LINE

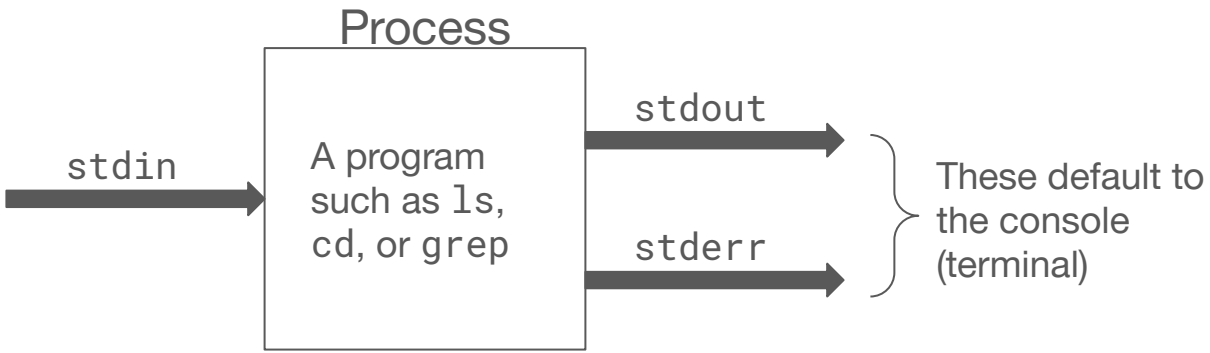
Command	description
`javac ClassName.java`	Compile ClassName
`java ClassName`	Run ClassName
`python, ruby, perl, gcc, go, etc`	Run or compile other files in different languages!

STANDARD STREAMS

- Every unix process has three *streams*, which are abstract locations that tell a program where to read input from and where to write output to.
- There are three standard streams:
  - `stdin` (Standard Input)
  - `stdout` (Standard Output)
  - `stderr` (Standard Error)
- You've likely already seen this before when writing a Java program, the `System.out` in `System.out.println` is referring to `stdout`
- By default, all of these default to the console (they print to the terminal and read from user input into the terminal). However, this can be easily changed.

STANDARD STREAMS

int	stream
`0`	`stdin`
`1`	`stdout`
`2`	`stderr`



STDIN VS PARAMETERS

- One of **the most important distinctions in this class** is the difference between `stdin` and a command's *parameters*.
- A *parameter* is an argument you give on the command line, like so
  - `$ ls dir1`
  - `dir1` is a parameter, it does not come from standard input
- Standard input comes from the user, either from a file or from the console
  - `$ grep "a"`
  - Once you type this command, it accepts input from your keyboard until you close the stream using Ctrl + D

STDIN VS PARAMETERS: JAVA

```java
// Read and print input from stdin
public static void main(String[] args) {
	Scanner console = new Scanner(System.in);
	while (console.hasNext()) {
		System.out.println(console.next());
	}
}
```

```java
// Read and print the parameters
public static void main(String[] args) {
	for (int i = 0; i < args.length; i++) {
		System.out.println(args[i]);
	}
}
```

OUTPUT REDIRECTION

`command > filename`

- Execute **`command`** and redirect its standard output to the given **`filename`**
  - If the file *does not* exist, create the given file.
  - If the file *does* exist, it will **overwrite the given file (BE CAREFUL!!)**
  - To append to a file instead of overwrite it, use >> instead of >
- Examples:
  - Output contents of current directory to files.txt: `ls -l > files.txt`
  - Append output of `wc -l veggies.txt` to files.txt: `wc -l veggies.txt >> files.txt`

INPUT REDIRECTION

```
command < filename
```

- Execute **`command`** and read its standard input from the contents of **`filename`** instead of from the console.
  - If a program usually accepts from user input, such as a console Scanner in Java, it will instead read from the file.
- Notice that this affects user input, not parameters.

STDERR REDIRECTION

```
command 2> filename
```

- Execute **command** and redirect its **standard error** to the given **filename**

```
command 2>&1
```

- Execute **command** and redirect **standard error** to **standard output**

```
command 2>&1 filename
```

- Execute **command**, redirect **standard error** to **standard output**, and redirect **standard output** to **filename**

PIPES

`command1 | command2`

- Execute **`command1`** and send its standard output as standard input to **`command2`**.
- This is essentially shorthand for the following sequence of commands:

```
command1 > filename
command2 < filename
rm filename
```

- This is one of the most powerful aspects of unix - being able to chain together simple commands to achieve complex behavior!

COMBINING COMMANDS

`command1 ; command2`

- Execute **`command1`**, then execute **`command2`**.

`command1 && command2`

- Execute **`command1`**, and if it succeeds, then execute **`command2`**.

THINK: pollev.com/cse391

- Write a command to store all of the lines in fruits.txt that contain the letter **a** into a file called `a.txt`

PAIR: pollev.com/cse391

- Write a command to store all of the lines in fruits.txt that contain the letter **a** into a file called `a.txt`

THINK: pollev.com/cse391

- Suppose we have a file `berries.txt` where each line is the name of a different berry. Write a command that outputs how many berries have names that contain **both** the letter **a** and the letter **e**.

PAIR: pollev.com/cse391

- Suppose we have a file `berries.txt` where each line is the name of a different berry. Write a command that outputs how many berries have names that contain **both** the letter **a** and the letter **e**.

THINK: pollev.com/cse391

- Write a command to output the contents between lines 10 and 15, both inclusive, of the file `veggies.txt`

PAIR: pollev.com/cse391

- Write a command to output the contents between lines 10 and 15, both inclusive, of the file `veggies.txt`

LOGS

- A common exercise in daily software development and operations is looking at log files - basically a status report of what is going on inside the program.
- We can look at the logs for all the CSE course websites by reading the file: `/cse/web/courses/logs/common_log`
- For example, to actively watch the log file and only look for access to our own course website, we could use the following

```
$ tail -f /cse/web/courses/logs/common_log | grep “391”
```