Nhập Môn CNTT – Thực Hành

Shell commands
Streams, Redirection

AGENDA

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

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

You’re working on a project and have been leaving comments with the text “TODO” near all the things you still need to finish. Your project is stored in the `project` directory in your current directory.

Write a command to find and print all of the lines that have a TODO on them in the `project` folder.

Hint: Use the “wildcard” to select all files as part of a path.

```
$ grep “TODO” project/*
```

C AND THE COMMAND LINE

Command	description
`gcc filename.c`	Compile C file
`./a.out`	Run the c program
`python, ruby, perl, go, etc`	Run or compile other files in different languages!

What is the command to compile all C files in the current directory?

Hint: How can you use wildcards to find all C files?

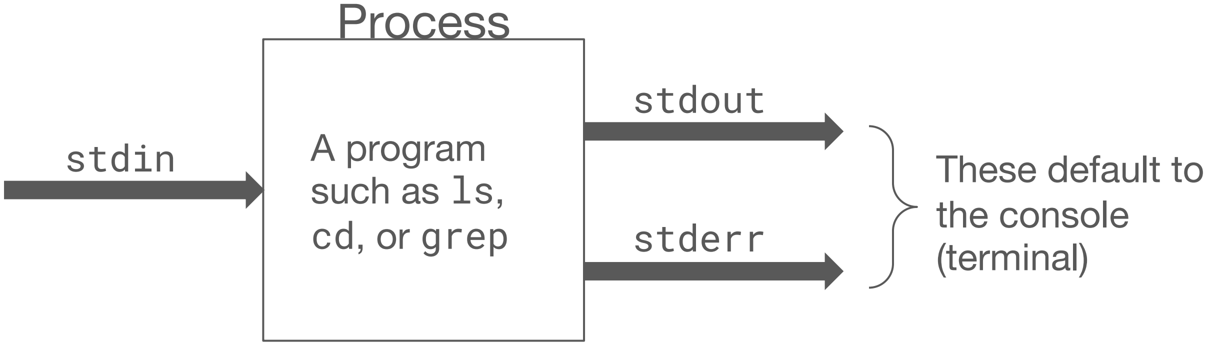
```
$ gcc -Wall *.c -o sayhello
```

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)
- 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



Process

stdin

A program such as ls, cd, or grep

stdout

stderr

These default to the console (terminal)

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

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.

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

```
$ grep “a” fruits.txt > a.txt
```

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!

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**.

```
$ grep “a” berries.txt | grep “e” | wc -l
```

COMBINING COMMANDS

```
command1 ; command2
```

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

```
command1 && command2
```

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

FIND

- **`find`** is a program for searching your filesystem for certain files.
- For example, to list all java files in the current directory and all subdirectories, recursively, we would run the following
    - `$ find -name “*.c”`
- This is commonly used with **`xargs`.** For instance, to compile all c files in the current directory and all subdirectories recursively
    - `$ find -name “*.c” | xargs gcc`
- Note that find has a plethora of options and flags, but we will most commonly use find with the `-name` and `-type` flags

COMMAND SUBSTITUTION

`$(command)`

- Another powerful tool is command substitution. It executes the given command and places that string literally into the given context.
- For example, to compile all C files in the current directory and subdirectories recursively, we can run the following
    - `$ gcc $(find -name “*.c”)`

STDERR REDIRECTION

- We've learned that we can redirect standard error using the 2> operator.
- Sometimes, however, we want standard error and standard out to go to the same location. We can do that with the following syntax:
  - `$ command > out.txt 2>&1`
- To understand this command, this reads as "redirect standard out to `out.txt`, redirect standard error to the same place as standard out"

TEE

- Sometimes, we want to redirect the output of a command to both a file and to the console. Do do this, we can pipe the output of a command to `tee`
  - `$ command | tee file.txt`
- To redirect both standard output and standard error to a file, and to the console, we use the following
  - `$ command 2>&1 | tee file.txt`

CUT

```
cut -d<DELIMITER> -f<FIELD>
```

- `cut` is a simple program to split lines based on a given delimiter.
- For example, to split the string “a,b,c,d,e” on commas and get the second entry, we would use the following:
  - $ echo “a,b,c,d,e” | cut -d, -f2
  - Note: the echo program simply prints the given string to standard out

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 a service by using: `$ journalctl`
- For example, to actively watch the log file and only look for access to our own course website, we could use the following

```
$ sudo journalctl —u <some-name> -f | grep “503”
```