CS2043 - Unix Tools & Scripting
Lecture 5
Processes and Jobs

Spring 2015 [1]

Instructor: Nicolas Savva

January 30, 2015

[1] based on slides by Hussam Abu-Libdeh, Bruno Abrahao and David Slater over the years

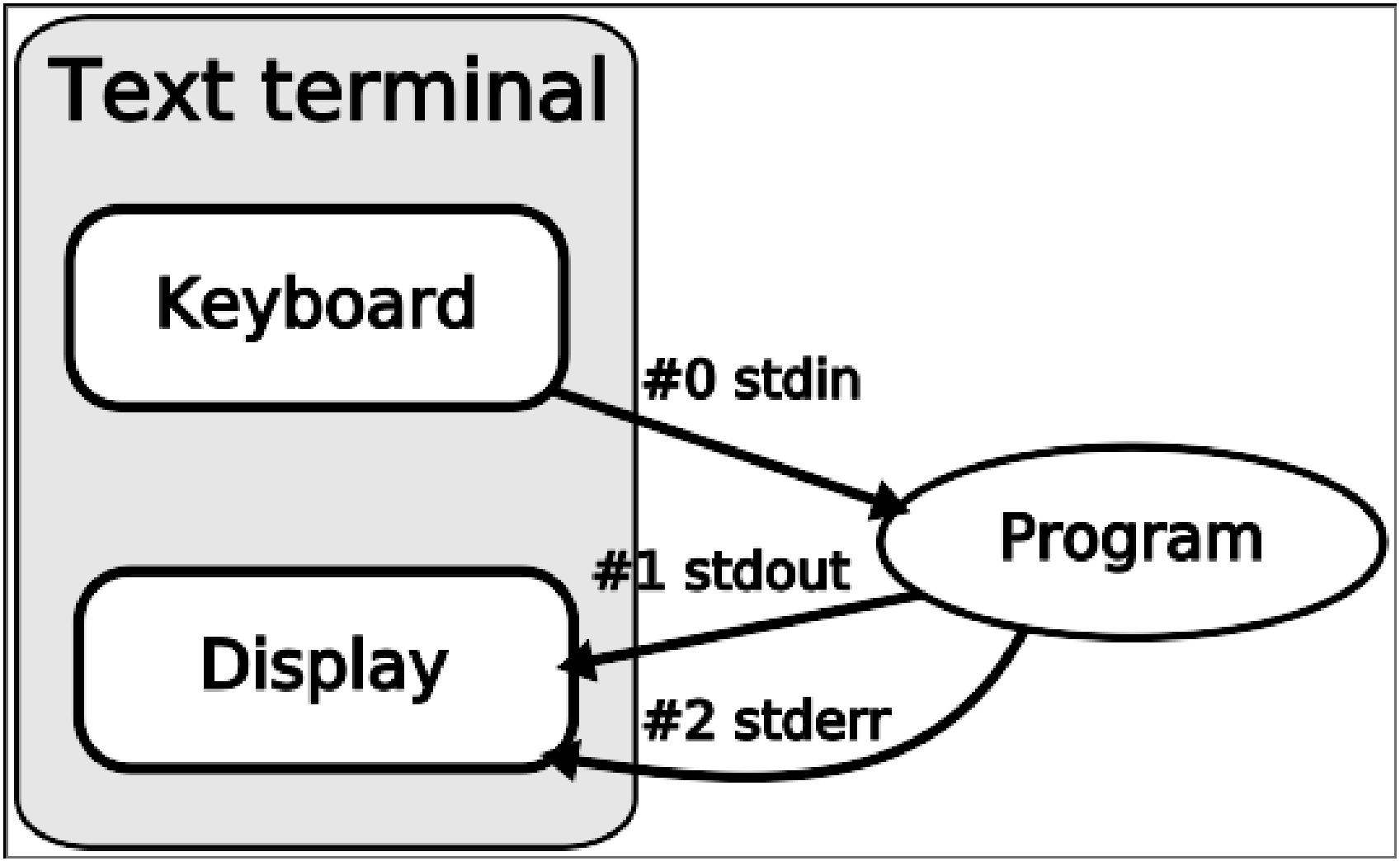
Announcements

- A2 coming out today (due in a week)
- Check if you can `ssh` into CSUGLab
- Course drop deadline next Wednesday (02/04)
- OH are posted on piazza (under Resources/Staff)

Today

- Piping and IO redirection (recap)
- More `tr`
- Processes & jobs

Redirection revisited: Standard Streams



Piping

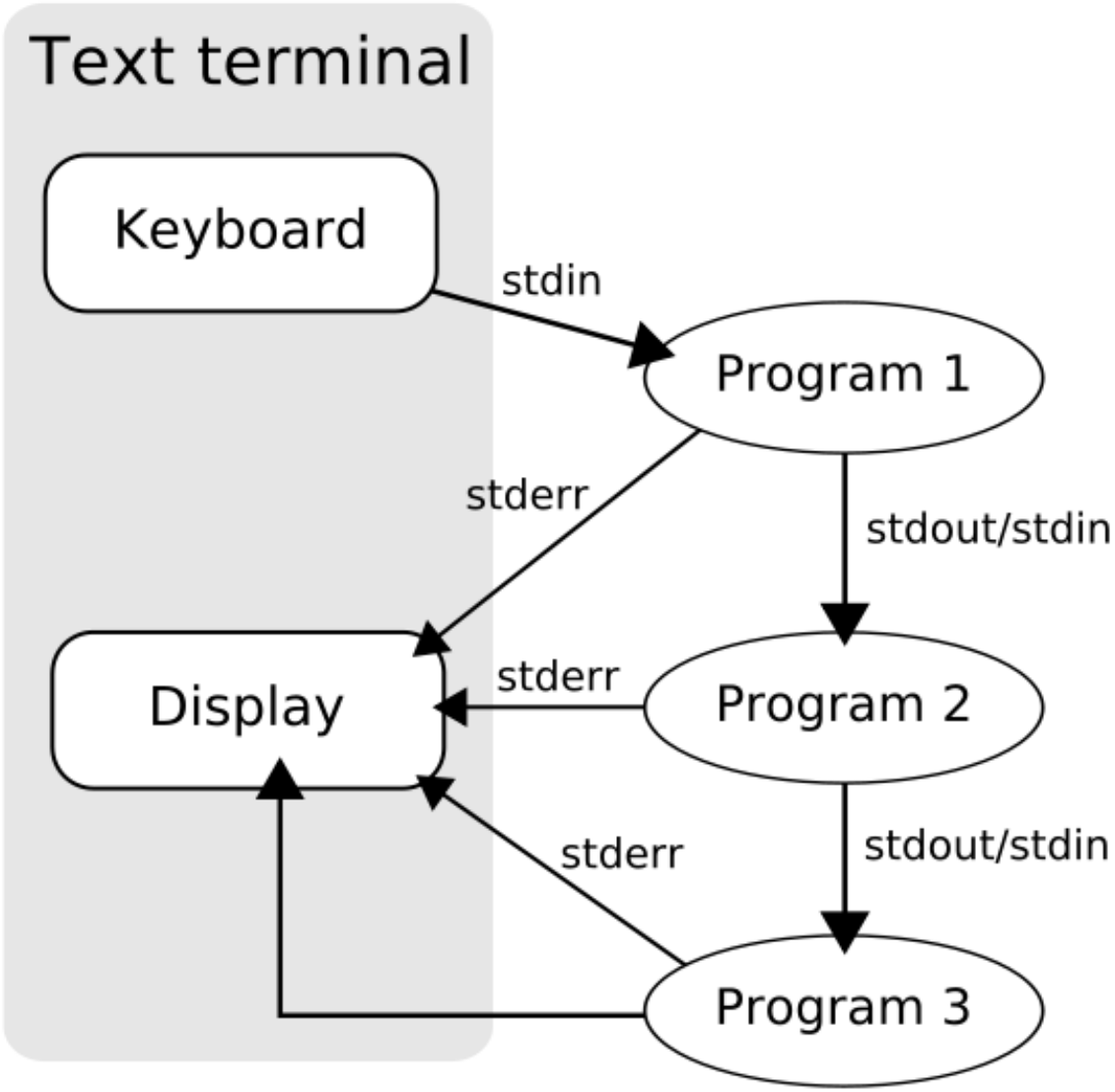
Bash scripting is all about combining simple commands together to do more powerful things. This is accomplished using the "pipe" character

Piping

`<command1> | <command2> | <command3>`

- Passes the output from command1 to input of command2
- Then passes the output from command2 to input of command3
- Works for lots of programs that take input and provide output to the terminal

Piping Streams



Piping Example

Example:

`ls -al /bin | less`

- Allows you to scroll through the long list of programs in /bin

`history | tail -20 | head -10`

- Displays the 10th-19th last commands from the current session

Redirection

To redirect Input/Output streams, use one of `>` `>>` `<`

Input/Output Streams

- to redirect standard input, use the `<` operator
  `command < file`
- to redirect standard output, use the `>` operator
  `command > file`
- to append standard output to contents of a file, use the `>>` operator
  `command >> file`
- to redirect standard error, use the `>` operator and specify the stream by number (2)
  `command 2> file`

Combining streams

You can combine two streams together by using `2>&1`

This says: send standard error to where standard output is going.

Useful for debugging/catching error messages.

Tee

`tee`

Redirect your output to a file and still see it on stdout terminal

Example

```
ls -l ~ / | tee homels.txt
```

Search and Replace

The Translate Command

`tr [options] <set1> [set2]`

- Translate or delete characters
- Sets are strings of characters
- By default, searches for strings matching set1 and replaces them with set2

Example:

`cat somefile | tr 'AEIOU' 'aeiou'` - changes all capital vowels to lower case vowels

tr and sets

- If the two sets passed to tr are the same length then the first character in the first set is replaced by the first in the second and so on.
- If the second is shorter than the first, then they are matched and all extra are changed to the last character
- If the first is shorter, then only those corresponding characters in the second matter

Example:

```
echo "abcdefghijklmnopqrstuvwxyz" | tr 'a-z' 'a'
aaaaaaaaaaaaaaaaaaaaaaaaaa

echo "abcdefghjiklmnopqrstuvwxyz" | tr 'a-z' 'wxyz'
wxyzzzzzzzzzzzzzzzzzzzzzz
```

tr options

tr has some very useful options:

tr options

- `tr -d <set>` - delete all characters that are in the set
- `tr -c <set1> [set2]` - complements set1 before replacing it with set2

Example:

Lets print a file with all non-letters removed:

- `tr -cd 'a-zA-Z' < somefile` - removes non-letters from somefile

tr and character ranges

tr does not understand regular expressions per se (and really for the task it is designed for they don't make sense), but it **does** understand character ranges and POSIX character sets such as `[:alpha:]`

Reminder:

- `[:alnum:]` - alphanumeric characters
- `[:alpha:]` - alphabetic characters
- `[:digit:]` - digits
- `[:punct:]` - punctuation characters
- `[:lower:]` - lowercase letters
- `[:upper:]` - uppercase letters
- `[:space:]` - whitespace characters

Redirection example

Bash processes I/O redirection from left to right, allowing us to do fun things like this:

Example:

Let's delete everything but the numbers from test1.txt, then store them in test2.txt

- `tr -cd '0-9' < test1.txt > test2.txt`

Substitution Cypher

We can use `tr` to do a simple substitution cypher. Create a text file called "cypher" with some reordering of the alphabet. Then to encode we would just do

- `tr 'a-z' 'cat cypher' < file > encodedfile`

and to decode we would do

- `tr 'cat cypher' 'a-z' < encodedfile > decodedfile`

Note the use of backticks around `cat cypher`. By doing this the shell expands this command and passes the output to `tr`.

Backticks ‘ ‘

When using Backticks around a command:
The shell will expand the command and replace the expression within the backticks (and the backtick characters) with the output of the expression after evaluation

Examples

```
echo ‘ls‘
ssh ‘whoami‘@csug07.csuglab.cornell.edu
whoami prints the userid
```

Process? What Process?

Definition

A `process` is an instance of a running program

- More specific than "a program" because it's being executed.
- More specific than "a running program" because the same program can be run multiple times simultaneously or use multiple processes

Example:

Many users could be simultaneously running `ssh` to connect to other servers. In this case, each *instance* of `ssh` is a separate process.

Process Identification

How do we tell one process from another?

- Each process is assigned a unique "Process ID" (or PID) when it is created
- These PIDs are used to differentiate between separate instances of the same program

How do we find out which processes are running, and with which PIDs?

The **P**rocess **S**napshot Command

```
ps [options]
```

- Reports a snapshot of the current running processes, including PIDs

By default, `ps` is not all that useful because it only lists processes started by the user in the current terminal. Instead...

`ps` Options

- `ps -e` – Lists every process currently running on the system
- `ps -ely` – Gives more info about your processes than you'll ever need
- `ps -u username` – Lists all processes for user username.
- NOTE: Options for BSD version are different! (See manpage)

- To see information about a specific process, pipe through `grep`.
- For example,

```
ps -e | grep firefox
```

shows us information about all firefox processes

Priority

Suppose you want to run some long running scientific calculation that might take days and consume 100% of the CPU on some server. Wouldn't it be nice if there was some way to tell the server to give your process less priority with CPU time?

- Remember that although UNIX seems to run tens or hundreds of processes at once, one CPU can only run one process at a time.
- Quick switching back and forth between processes makes it seem as though they are all running simultaneously

UNIX Developers saw this type of situation coming - each process is given a `priority` value when it starts.

Initial Priority

Start a process with a non-default priority:

The `nice` command

```
nice [options] command
```

- Runs command with a specified "niceness value" (default: 10)
- Niceness values range from -20 (highest priority) and 19 (lowest priority)
- Only root can give a process a negative niceness value!
- Commands run without nice have priority 0.

Example:

```
nice -n 10 azureus
```

- Keeps torrent downloads from hogging all our CPU time!

Adjusting Priority

Adjust the niceness of a running process:

The `renice command`

`renice <priority> -p <PID>`

- Changes the niceness of the indicated process to `<priority>`
- Again, only root can go below 0!
- Can only renice processes YOU started!

Example:

`renice 5 -p 10275`

- Sets the niceness of the process with PID 10275 to 5 (slightly lower priority than default)

`renice 19 -u nsavva`

- renice all my processes to 19

Going Rogue...

To end a process running in the foreground simply hit `Ctrl + C`

- What about a background process that stops working?
- What is the UNIX version of CTRL + ALT + DELETE?

The `kill` command

`kill`

`kill [-signal] <PID>`

- Sends the specified signal to the process
- By default, terminates execution

So to terminate a process:

- Look up the process's PID with `ps`
- Use that PID to `kill` the process

The `killall` command

Feeling extra vengeful?

`killall`

```
killall [-signal] <name>
```

Kill processes by name

Example

```
killall firefox
```

Useful Kill Signals

Signal used with `kill` can either be specified by their names or numerical values.

- `TERM` or 15 : Terminates execution (default)
- `HUP` or 1 : Hang-up (restarts the program)
- `KILL` or 9 : Like bleach, can kill anything

Generally speaking, the default `TERM` will get the job done.

Example:

- `kill 9000` : terminates process 9000
- `kill -9 3200` : REALLY kills PID 3200
- `kill -HUP 12118` : Restarts 12118 (handy for servers & daemons)

all in one: top

`top` is a useful little program that lists and dynamically updates information about running programs. It also allows the user to kill and renice other processes.

top

`top [-options]`

- Lists processes by default by percentage of CPU usage
- Customizable display, hit `h` for a list options
- u - show specified user only
- Can manipulate tasks: 'k' kill; 'r' renice

Introduction to Jobs

Jobs

A Job is a process running under the influence of a job control facility.

- What does that mean?!?!?!?!?

- Job control is a built-in feature of most shells, allowing the user to pause and resume tasks, as well as run them in the background (so that the shell is usable while it executes!)

How does this help?

Lets use the `ping` command as an example.

Ping

`ping <server>`

- Measures the network response time (or network latency) to a remote server.
- Sends short bursts to the server, then measures the time until they are returned.
- Can be a good way to check your network connection

Try pinging a reliable location, like `google.com`

Example:

```
ping google.com
```

Why We Need Job Control

As long as `ping` runs, we lose control of our shell. This happens with many applications which run either indefinitely or for long periods:

- Moving large quantities of files
- Compiling source code
- Playing multimedia
- Doing Scientific Computing

Example:

```
mpg123 song.mp3
```

Starting a Job in the background

To run a job in the background, we will use a new command-line operator:

&

`<command> [arguments] &`

- Runs the specified command as a background job
- Unless told otherwise, will send output to the terminal!

Since `cat` runs indefinitely with no arguments, this will illustrate our point:

Example:

```
cat &
```

- Try it without the &!

Backgrounding a Running Job

What if we start a process normally and it's taking too long?

Pausing a Job

Press CTRL + Z to pause a running process!

- When we do this, the shell tells us the paused job's JOB ID
- This Job ID is used like a process's PID
- Once we have a process paused, we can tell it to continue in the background...

The Background Command

bg's Usage

`bg <Job ID>`

- Resumes a paused job in the background
- Without a Job ID resumes the last job placed in the background

how do we find these Job IDs?

the Job Table

`jobs`

- Prints currently running, paused, or recently stopped jobs
- Prints jobs with their Job IDs

Foregrounding an Existing Job

What if we want to resume a job in the foreground?

`fg`'s usage

`fg <Job ID>`

- Resumes a paused job in the foreground
- Again without a Job ID resumes the last command placed in the background

To kill a job, either foreground it and the hit CTRL+C, or you can use the `kill` command with the PID

Dealing with Excess Output

Many programs output continuously as they run. For example `ping` and `play` both clutter up the terminal with output even when they are backgrounded.

- The solution is to use output redirection

Example:

```
ping google.com > testping.log &
```

- When you care about a program's output, redirect it to a log file.

Example:

```
play somesong.mp3 > /dev/null &
```

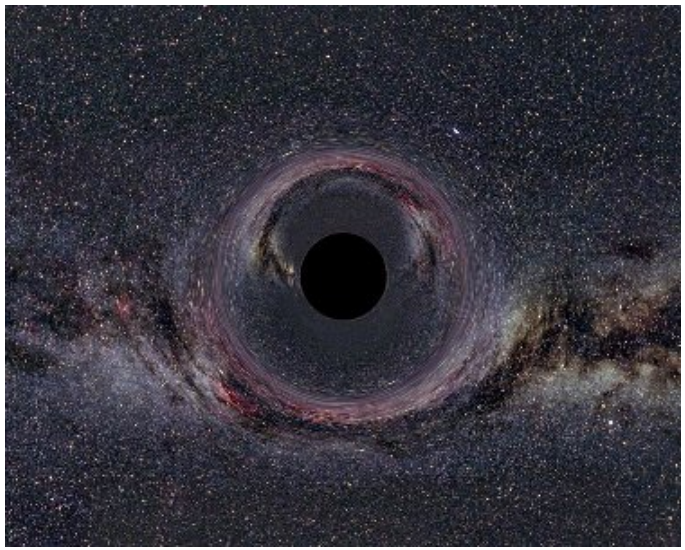
- If the text output doesn't matter, redirect it to `/dev/null`.

/dev/null - the black hole

`/dev/null` is a special file which has the following properties:

- Any user can write to it.
- Anything written to it goes nowhere
- it always reports a successful write.

It works like a black hole for data - you can output to it all day and it will never fill up. Anything you redirect to /dev/null just disappears.



Making Jobs Persist

If you have a **noninteractive** batch job, you can also allow it to continue to run after you logout by using `nohup`

`nohup`

`nohup command`

- command will continue to run after you logout
- output is sent to `nohup.out` if not otherwise redirected
- can be combined with `nice`

Example:

```
nohup nice -15 math < BatchJob.m &
```

Coming up

- Multiplexing terminals: tmux / screen
- `find` / `grep`, and pattern matching
- Regular expressions
- Scripting 101
- sed / (g)awk