CS2043 - Unix Tools & Scripting
Lecture 4
More Unix Tools

Spring 2015 [1]

Instructor: Nicolas Savva

January 28, 2015

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

Course Logistics

- Last day to enroll!
- Assignment 1 is due tonight
- Late policy: total of 5 days for the course
- Additional OH and support (beginning next week)

Today

- Accessing Remote Resources
- Variables
- More useful commands
- Piping, input/output redirection

ssh

You can use "secure shell" (`ssh`) to connect to a remote machine.

```
ssh [username@]<remote machine name or IP address>
```

- If the username is omitted, local username will be used.
- Remote machine has to be configured to accept `ssh` connections:
  - `ssh` daemon (service) has to be running and listening on an open port (by default 22)

Executing remote commands

`ssh` can be used to execute commands on the remote machine

Example

```
ssh nsavva@csug01.csuglab.cornell.edu ls
```

this will execute `ls` on `csug01.csuglab.cornell.edu` and output the result to the screen before `ssh` terminates the connection

- You can use the `-f` flag to put `ssh` into the background before executing the remote command
- You can use the `-Y` flag to forward X11 (graphical windows/user interface) to the local machine

Run firefox on the remote machine

```
ssh -Y nsavva@csug01.csuglab.cornell.edu firefox
```

Identity files

- You can use an identity file to authenticate with the remote machine instead of using your username/password.
- An identity file allows you to authenticate yourself with a "*pass phrase*" (which could be empty).
- Identity files are typically a pair of public/private keys used for asymmetric key cryptography (e.g., RSA).

Identity files

To use identity files,

1. Create an identity file using `ssh-keygen`

create identity files using RSA encryption

```
ssh-keygen -t rsa
```

2. Append the generated public key file (by default `~/.ssh/id_rsa.pub`) to the `~/.ssh/authorized_keys` file on the remote machine.
3. `ssh` to the remote machine using the `-i` flag to use the identity file, and specify the private file corresponding to the public file appended at the remote machine

```
ssh nsavva@example.com -i ~/.ssh/id_rsa
```

ssh configuration file

You can configure `ssh` to use customized settings when connecting to a particular host without having to set the corresponding flags every time. The file `~/.ssh/config` contains these settings.

Sample config

```
host rgblab
hostname maxwell.cs.cornell.edu

host tesla
hostname tesla.cs.cornell.edu
user nsavva
ForwardX11 yes
IdentityFile ~/.ssh/id_rsa
```

Here, `ssh rgblab` connects to `maxwell.cs.cornell.edu` and `ssh tesla` connects to `tesla.cs.cornell.edu` with username `nsavva` and identity `~/.ssh/id_rsa` and enable X11 forwarding.

sftp

- Transfer files securely between local and remote machines.
- Operates over an encrypted `ssh` transport.
  - same connection settings as `ssh`
- Uses an interactive console to interact with the user
  - unless the `-b [batchfile]` option is used to use batch files
- Useful sftp commands:
  - `help` : to see a list of commands and help on them
  - `put` : upload a file to the remote machine
  - `get` : download a file from the remote machine
  - `cd / pwd` : change directory / print current directory on remote machine
  - `lcd / lpwd` : change directory / print current directory on local machine

scp

- Copy files securely over a network using an encrypted `ssh` transport.

copy file to remote machine

```
scp file nsavva@remote_machine:
```

copy file from remote machine

```
scp nsavva@remote_machine:file .
```

- The ':' is necessary after the remote machine name. A path on the remote machine starting from the user's home directory can be specified after the colon ':'.

copy directories using the `-r` flag

```
scp -r pics_dir nsavva@remote_machine:
```

Other useful commands

wget

```
wget [OPTIONS] [URL...]
```

Download a file from a remote location over HTTP. Popular options:

- `-r` : recursive
- `-c` : continue a partial download

curl

```
curl [OPTIONS] [URL...]
```

Transfer data from/to web servers.

For more info on these commands, consult the `man` pages.

Variables

- Bash scripting is powerful (you could write a web server just using bash scripting)
- We need variables to really get anything done
- All variables preceded by the dollar sign ($)
- The contents of any variable can be listed using the `echo` command
- `Two types of variables: Environment and Local`

Example:

```
echo My shell is $SHELL and the username is $USER
My shell is /bin/zsh and the username is nsavva
```

Environment Variables

- Environment Variables are used by the system to define aspects of operation
- The shell passes environment variables to its child processes
- Examples:
  - $SHELL : which shell will be used by default
  - $PATH : a list of directories to search for binaries
  - $HOSTNAME : the hostname of the machine
  - $HOME : the home directory for the current user
- To get a list of all current environment variables use the `env` command

New Environment Variable:

Set a new environment variable using `export`

```
nsavva@x200t:~$ export X=42
nsavva@x200t:~$ echo $X
42
```

Local Variables

We can define local variables which only exist in the current shell:

New Environment Variable:

Set a new environment variable using `export`

```
nsavva@x200t:~$ x=7
nsavva@x200t:~$ echo $x
7
```

Note: You cannot have a space after the x nor before the 7

- The main difference between environment and local variables is that the environment variables are passed to child processes while local variables are not.
- A copy is passed (variable changes in the child processes are not reflected in parent)
- We will talk more about this in a few lectures

Listing and Removing Variables

- `env` : displays all environment variables
- `set` : displays all shell/local variables
- `unset name` : remove a shell variable
- `unsetenv name` : remove an environment variable

`wc`

- How many lines of code are in my new awesome program?
- How many words are in this document?
- Good for bragging rights

Word, Character, Line, and Byte count with `wc`

- `wc -l` : count the number of lines
- `wc -w` : count the number of words
- `wc -m` : count the number of characters
- `wc -c` : count the number of bytes

sort

Sorts the lines of a text file alphabetically.

- `sort -r -u file`
  - sorts the file in reverse order and deletes duplicate lines.
- `sort -n -k 2 -t : file`
  - sorts the file numerically by using the second column, separated by a colon

Example

Consider a file (`numbers.txt`) with the numbers 1, 5, 8, 11, 62 each on a separate line, then:

```
$ sort numbers.txt
1
11
5
62
8
```

```
$ sort numbers.txt -n
1
5
8
11
62
```

uniq

- `uniq file` - Discards all but one of successive identical lines
- `uniq -c file` - Prints the number of successive identical lines next to each line

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

Some Simple Examples

Example:

`echo *` prints everything in the directory, separated by spaces. Let's separate them by newlines instead:

- `echo * | tr ' ' '\n'` – replaces all spaces with newlines

Example:

Let's print a file in all uppercase:

- `tr 'a-z' 'A-Z' < test.txt` - prints the contents of text.txt in all caps

Pipes and redirection

- `tr` only receives input from *standard input* (stdin)
  - i.e. keyboard input
- What if we want to operate on files?
  1. Piping: `cat somefile | tr 'AEIOU' 'aeiou'`
  2. Input redirection: `tr 'AEIOU' 'aeiou' < somefile`

Pipes and input/output redirection are important and useful throughout UNIX.

Redirection revisited

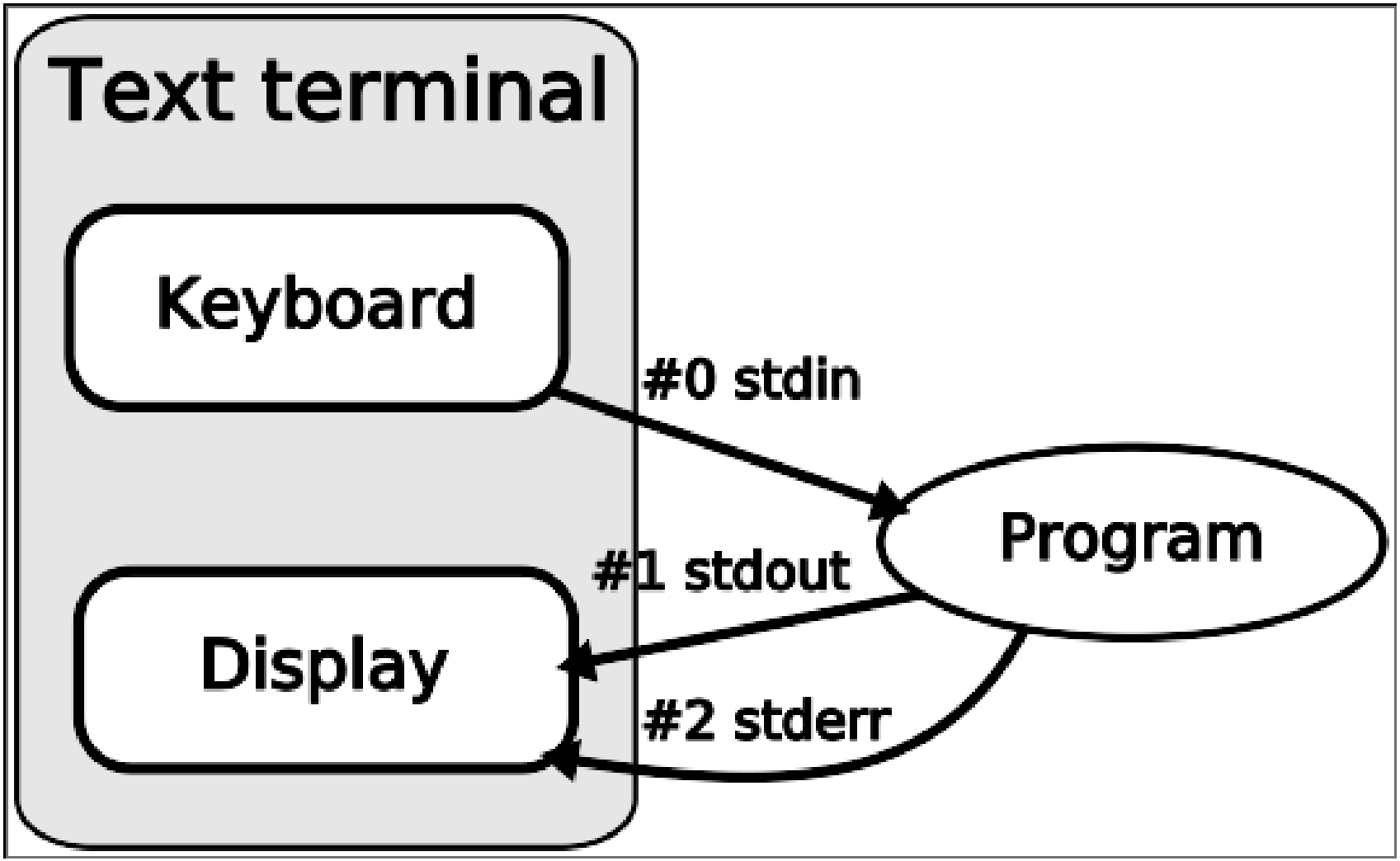
Applications in UNIX are associated with Input/Output (I/O) Streams:

- #0 : Standard input stream; STDIN
  (usually keyboard)
- #1 : Standard output stream; STDOUT
  (usually terminal console)
- #2 : Standard error stream; STDERR
  (depends on system setting, but usually terminal console)

UNIX Philosophy

In UNIX you will find many tools that specialize in one or a few things, and they do them really well! To get more complex functionality combine one ore more tools by piping or I/O redirection

Standard Streams



Piping

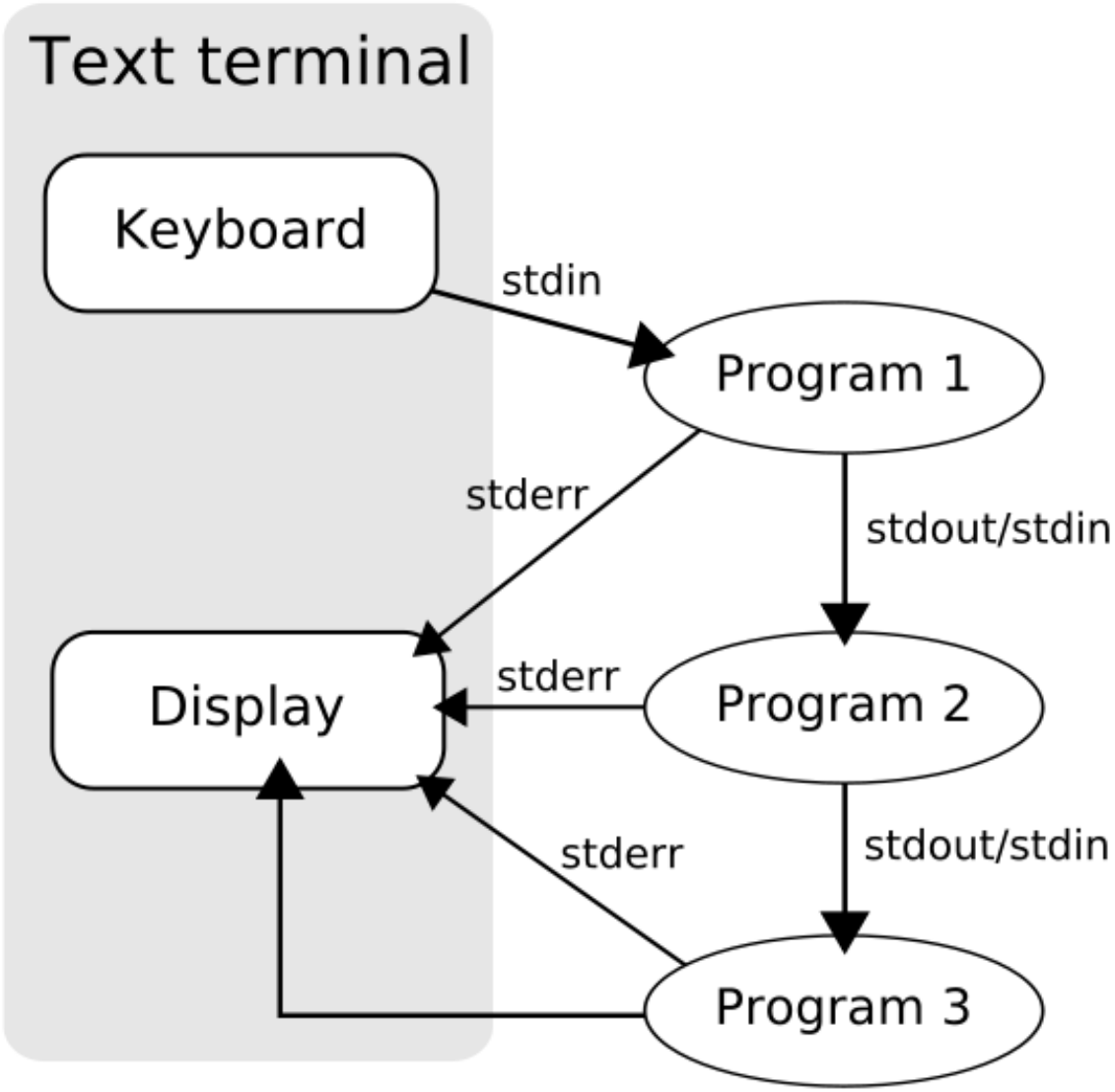
Bash scripting is all about combining simple commands together to do more powerful things. This is accomplished using the "pipe" character

Piping

`<command1> | <command2>`

- Passes the output from command1 to input of command2
- 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 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.

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`

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 &!

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 &
```

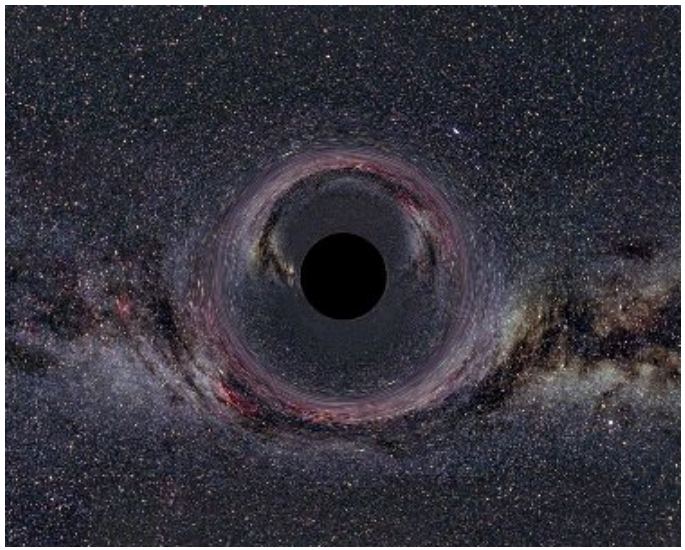
- If the text output doesn't matter, redirect it to `/dev/null`.

/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.



Tee

`tee`

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

Example

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

Next Time

- Processes and Jobs
- Multiplexing terminals: tmux / screen
- find grep, and pattern matching