CS2043 - Unix Tools & Scripting

Cornell University, Spring 2014[1]

Instructor: Bruno Abrahao

January 31, 2014

[1] Slides evolved from previous versions by Hussam Abu-Libdeh and David Slater

Vim: Tip of the day!

Line numbers

Displays line number in Vim:

`:set nu`

Hides line number in Vim:

`:set nonu`

Goes to **line_number**:

:line_number

`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 -ru 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

Character manipulation!

The Translate Command

`tr [options] <char_list1> [char_list2]`

- Translate or delete characters
- char_lists are strings of characters
- By default, searches for characters in char_list1 and replaces them with the ones that occupy the same position in char_list2

Example:

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

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

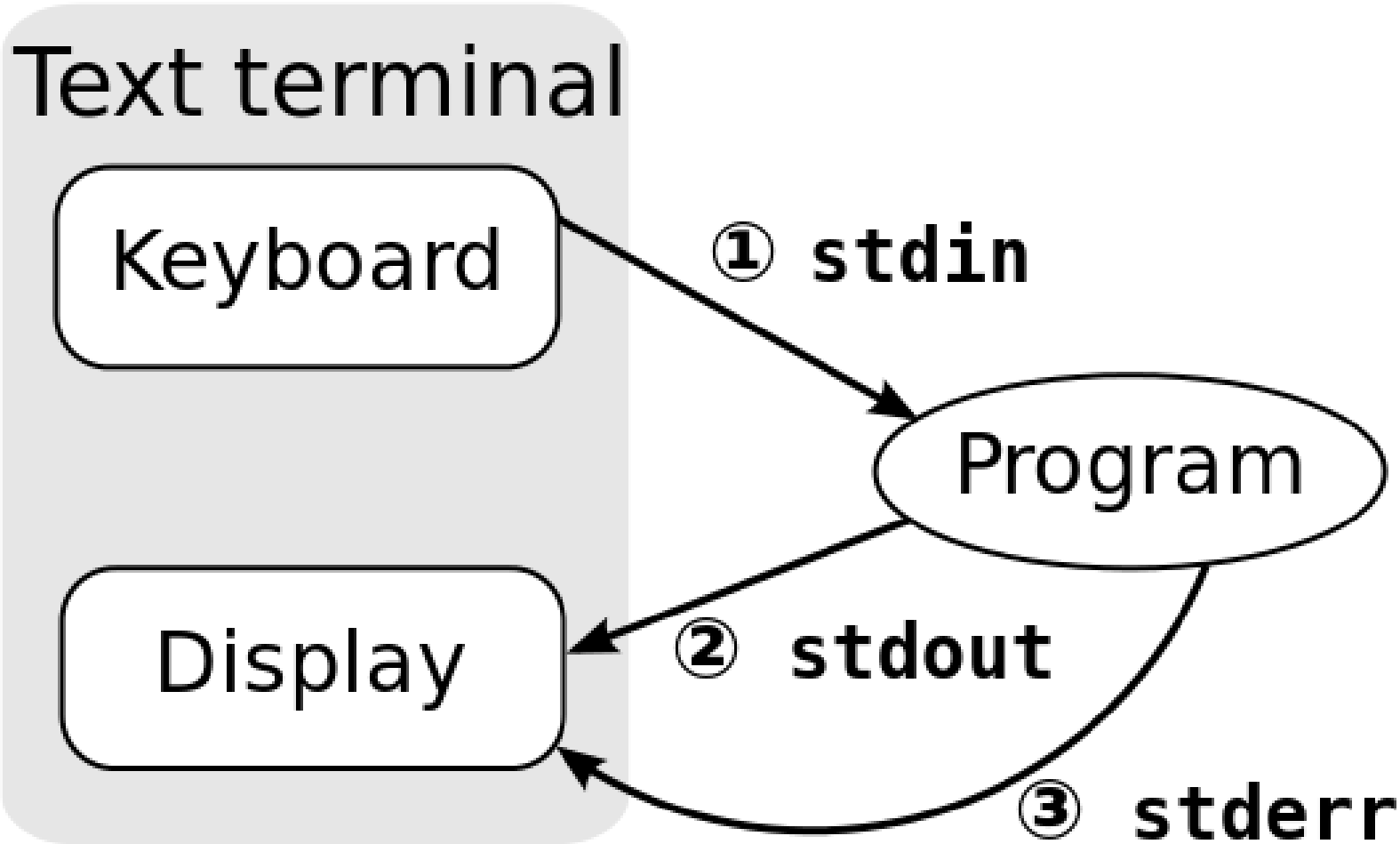
Applications in UNIX are associated with Input/Output (I/O) Streams:

- #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 example

Example:

`ls -al /bin | less`

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

`history | head -20 | tail -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`

Merging 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`

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

Putting things together

Example:

We can put some of these together commands together now to do interesting things.

```
tr 'A-Z ' 'a-z\n' < file.txt | sort | uniq -c | sort -rn | head -n 10
```

Read `file.txt`, convert all letters to lower case, replace spaces with new lines, sort the result, count repeating words, and then sort again in reverse order (printing the most common words first), then display the top 10 entries.

This effectively computes the most frequent words in file.txt (without handling punctuation, we'll deal with that later).

tee party

What if you want to redirect your output to a file and still see it on the stdout?

tee Example

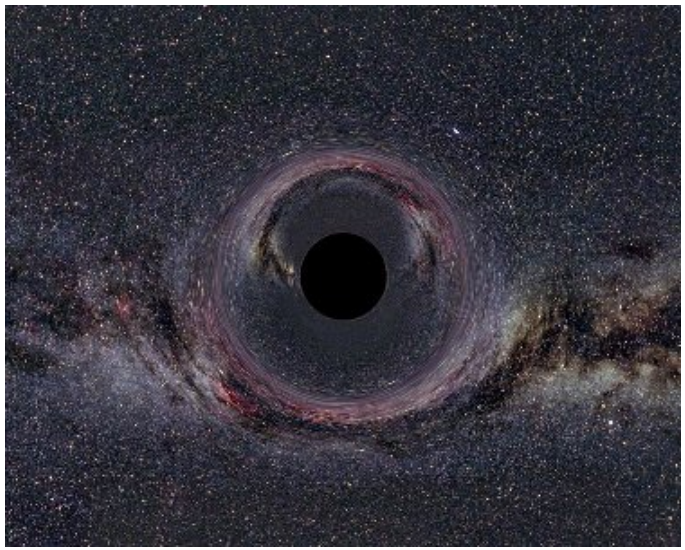
```
ls -1 | tee file.txt
```

/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.



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`.

Revisiting a command
More on `tr`

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

POSIX character sets

tr understands POSIX character sets

Sets

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

Backticks

Backticks around a command: the shell expands this command and replaces the expression within the backticks (including the backticks) with the expression's output.

Examples

The command `whoami` prints the name of the user

- `echo ‘whoami‘`
- `ssh ‘whoami‘@csug01.csuglab.cornell.edu`

Substitution Cypher

Here is how the secret message from hw1 works:
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`