csce215 — UNIX/Linux Fundamentals
Fall 2021 — Lecture Notes: Building with blocks

This document contains slides from the lecture, formatted to be suitable for printing or individual reading, and with some supplemental explanations added. It is intended as a supplement to, rather than a replacement for, the lectures themselves — you should not expect the notes to be self-contained or complete on their own.

(3.1) Last time

Last time we learned some commands for **creating** and **changing** files and directories:

- `vim`
- `mkdir`, `rmdir`
- `mv`, `cp`, `rm`, `rm -r`
- output redirection with `>` and `>>`

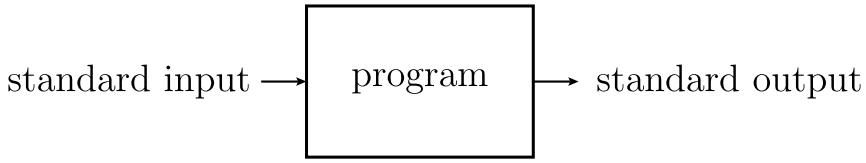
Today, we will learn how to combine multiple programs into larger, more powerful commands.

(3.2) Input and Output

Each time we run a program, that program has access to two primary data streams.

- Input comes from **standard input**.

- Output goes to **standard output**.



(3.3) You already know this

Your favorite programming language has a way to read from standard input:

`System.in`, `cin`, `read`, `sys.stdin`...

And write to standard output:

`System.out`, `cout`, `print()`, `sys.stdout`...

(3.4) Typical behavior

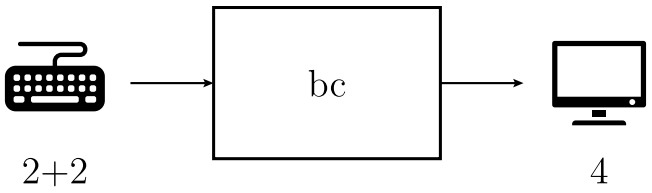
Usually:

- Standard input is connected to the keyboard.
- Standard output is connected to your terminal window.

(3.5) Example: A calculator program

`bc`
Read simple arithmetic expressions from standard input and write the answers to standard output.

```
$ bc
2+2
4
```



(3.6) Review: Output redirection

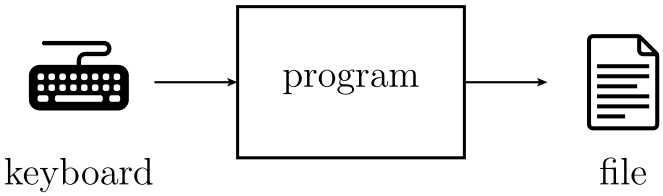
We can **redirect** standard output to go to a file instead of the terminal. We saw this last time.

> and >>

Send the standard output of a command to a file.

> If the file exists, replace it.

>> If the file exists, add to the end.

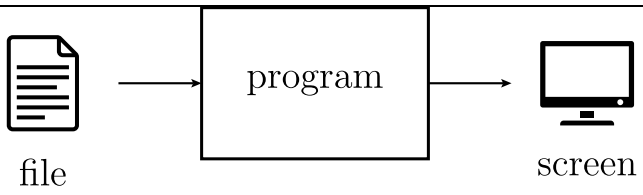


(3.7) Input redirection

We can **redirect** standard input come from a file instead of the keyboard.

<

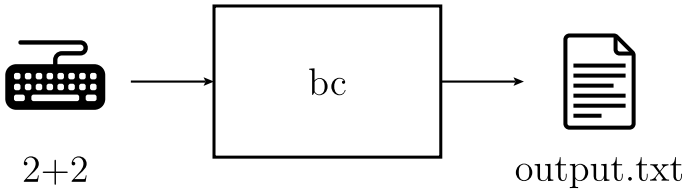
Read the standard input to a command from a file.



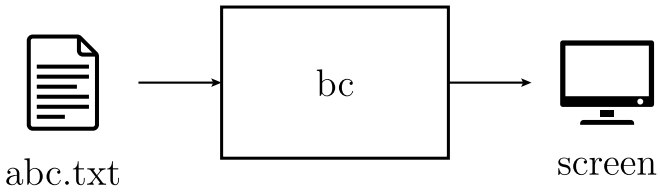
Many programs read from standard input until they reach an **end of file (EOF)** condition. To generate EOF from the keyboard, press Ctrl-D.

(3.8) Examples with bc

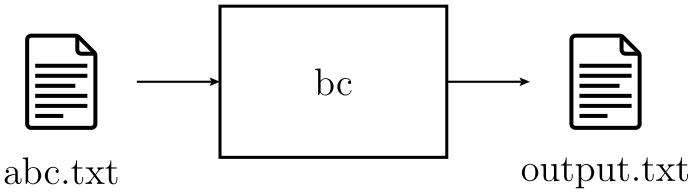
```
$ bc > output.txt
2+2
$cat output.txt
4
```



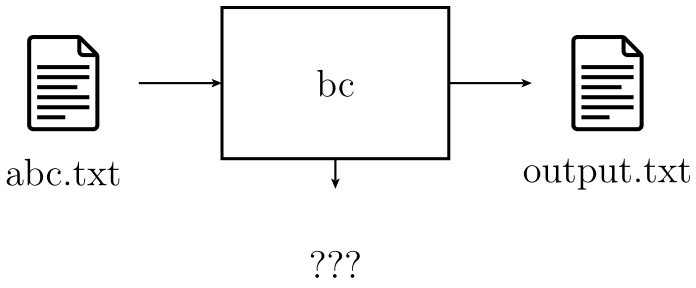

```
$ bc < abc.txt
4
```




```
$ bc < abc.txt > output.txt
```



```
$ echo 2+2+ > abc.txt
$ bc < abc.txt > output.txt
(standard_in) 2: syntax error
```



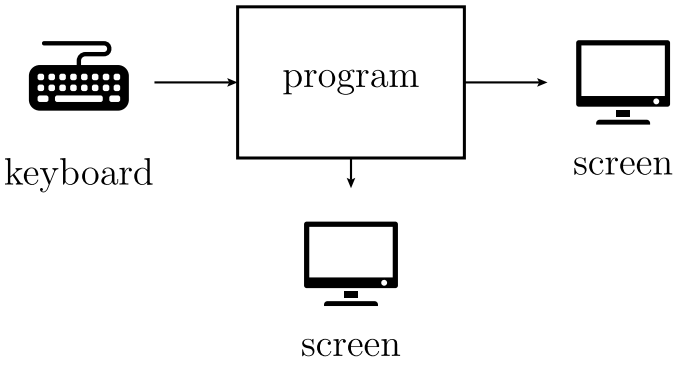
(3.9) Standard Error

In addition to standard input and standard output:

- Error messages go to **standard error**.

Your favorite programming language has a way to write to standard error:

`System.err`, `cerr`, `print(file=sys.stderr,...)`, ...



Key idea: Standard error lets us see error messages, even when we are not looking at standard output.

(3.10) Error redirection

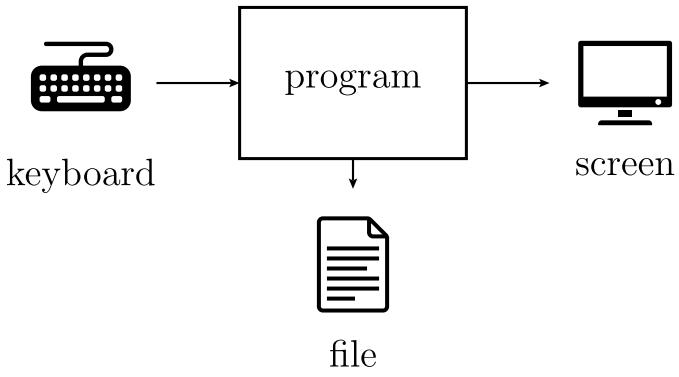
Just like standard output and standard input, we can **redirect** standard error to a file.

2 > and 2 >>

Send the standard error of a command to a file.

2 > If the file exists, replace it.

2 >> If the file exists, add to the end.



(3.11) Example: Catching compile errors

A poorly-written C++ program can generate lots of errors. Here's a mild example:

```
$ g++ broken.cpp
broken.cpp:6:11: warning: missing terminating " character
    6 |    cout << "hello world' >> cin;
      |           ^
broken.cpp:6:11: error: missing terminating " character
    6 |    cout << "hello world' >> cin;
      |           ^~~~~~~~~~~~~~~~~~~~
broken.cpp:1:1: error: include does not name a type
    1 | include <iostreem>
      | ^~~~~~~
broken.cpp: In function int mina():
broken.cpp:6:3: error: cout was not declared in this scope
    6 |    cout << "hello world' >> cin;
      |    ^~~~
broken.cpp:7:1: error: expected primary-expression before } token
    7 | }
      | ^
broken.cpp:7:1: warning: no return statement in function returning non-void [-Wreturn-type]
```

Instead: Capture errors to take a closer look:

```
$ g++ broken.cpp 2> errors
```

(3.12) Multiple commands

Suppose we want the output of one program to be the input of another program. How can we do that?

One (not so great) option:

```
$ program1 > temp
$ program2 < temp
$ rm temp
```

But...

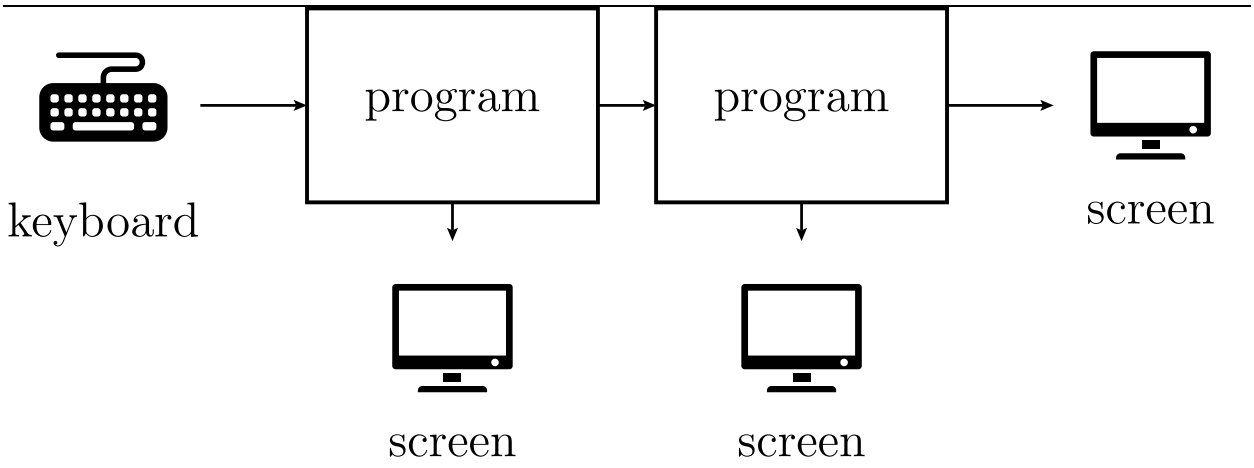
- Three separate steps: Three chances to mess up.
- The programs run one at a time.
- We have to keep track of the temporary file `temp`.

Conclusion:

(3.13) Pipes

A **pipe** runs two or more commands, connect the standard output of each command to the standard input of the next.

(pipe) 
Use the standard output of one command as the standard input of the next.



(3.14) Why?

The example from before becomes simply:

```
$ program1 | program2
```

- One compact line.
- Faster: `program2` can begin processing its input *while `program1` is still producing more output*.
- No temporary files. Data flows directly from one program to the other.

And: It's easy to combine more than 2 programs into a **pipeline** to get the answer we want.

Conclusion:

(3.15) Filters

A program that is designed to be used in pipelines is called a **filter**. There are lots of useful filters.

head

Copy the first 10 lines of standard input to standard output. Ignore the rest.

`-n` k show k lines instead of 10.

tail

Copy the last 10 lines of standard input to standard output. Ignore the rest.

`-n` k show k lines instead of 10.

```
$ cat /usr/share/dict/american-english | head
A
A's
AMD
AMD's
AOL
AOL's
AWS
AWS's
Aachen
Aachen's
```

```
$ ls /dev | tail -n 5
video5
video6
video7
zero
zfs
```

(3.16) More filters

Here are some more examples.

You'll explore these in Lab 3. (Don't forget about the `man` command to find out more about each one!)

`grep`

Find lines that match a pattern.

- `-i` case insensitive
- `-v` find lines that *don't* match

`sort`

Put lines in order.

- `-n` numerical order
- `-r` reverse order

`uniq`

Eliminate duplicated adjacent lines.

- `-c` count duplicates

`wc`

Count the number of lines, words, and characters.

`nl` **'en-ell'**

Number the lines

`tac`

Reverse the order of the lines

tac
Reverse each line, character by character

cut
Extract only part of each line. -c select ranges of characters -f select fields -d'x' specify delimiter character for fields

(3.17) Pipes are powerful!

What camera devices are connected?

```
$ lsusb | grep -i camera
Bus 001 Device 002: ID 13d3:56bb IMC Networks Integrated Camera
```

Show me files here, one page at a time.

```
$ find | less
```

What sort of a file has a cpp extension?

```
$ cat /etc/mime.types | grep cpp
text/x-c++src                                   c++ cpp cxx cc
```

Which course that I've taught uses the most disk space?

```
$ du ~/teaching -d1 | sort -n | tail -n 5
```

How many times was my research cited in papers published at ICRA 2021, accounting for misspellings and different sorts of apostrophe characters, but not counting other researchers with somewhat similar names?

```
grep -r Kane * | grep -v Kanehiro | grep -v Kanerva \
| grep -v Kanehioro | grep -v Johnson | grep -v Kaneko \
| grep -v Kanesiroo | wc -l
```