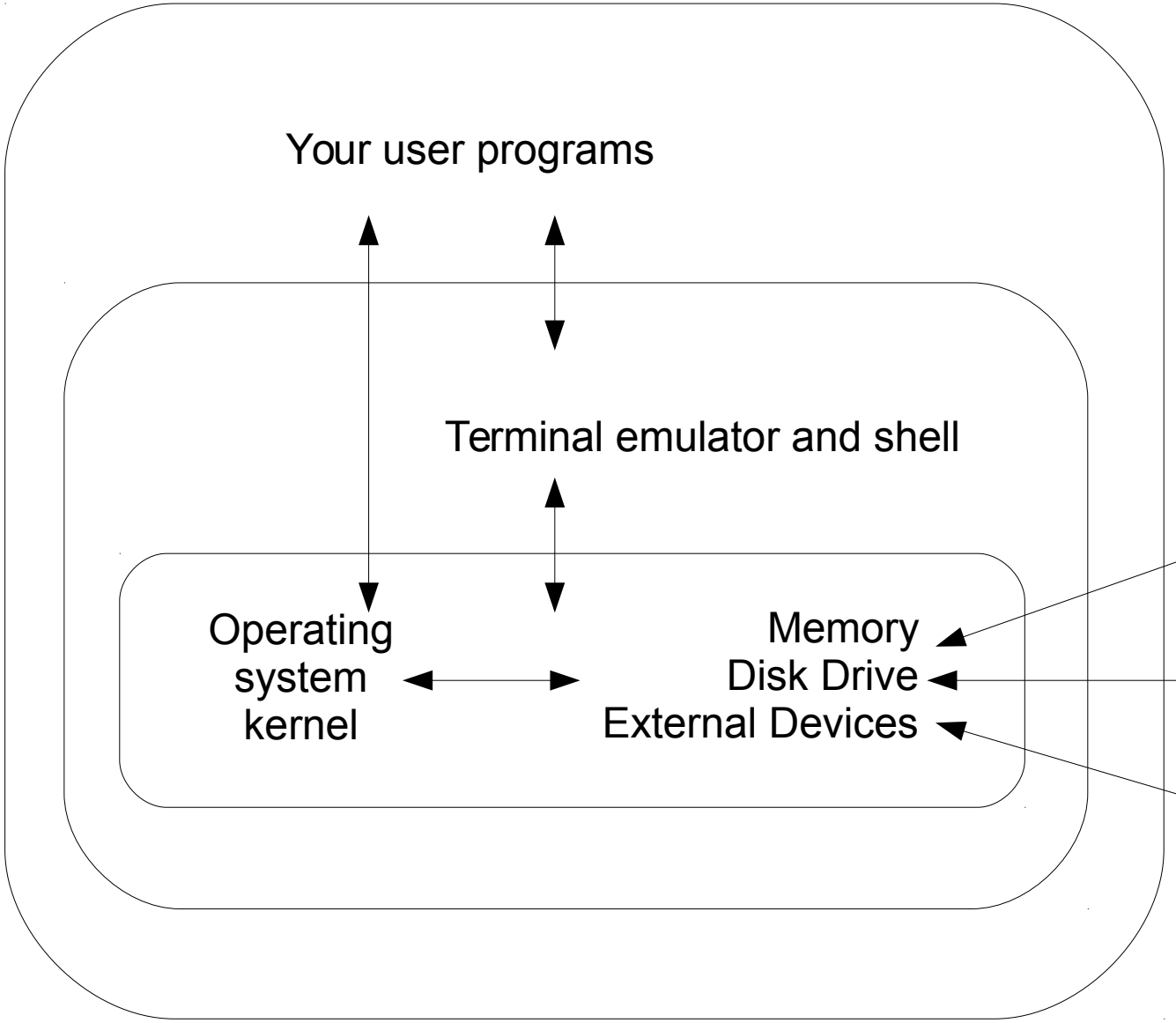
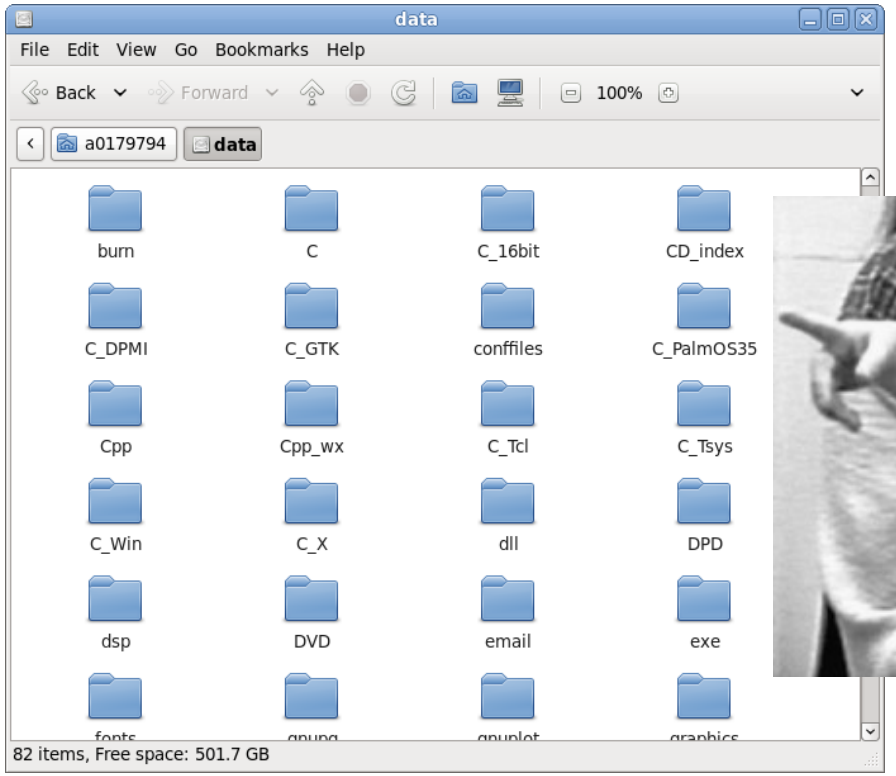
Class Progress

Basics of Linux, gnuplot, C
Visualization of numerical data
Roots of nonlinear equations
(Midterm 1)
Solutions of systems of linear equations
Solutions of systems of nonlinear equations
Monte Carlo simulation
Interpolation of sparse data points
Numerical integration
(Midterm 2)
Solutions of ordinary differential equations

Model of Operating System Processes



“Point and Click” Versus Command Language



Infants progress through a “pointing development” stage, but must then learn language and speech to successfully interact with a complex world

“Point and Click” Versus Command Language

```
i = 0;
while (i < n) {
  if (i != k) {
    j = k + 1;
    while (j < n) {
      coeff[i][j] -= coeff[i][k] * coeff[k][j];
      j++;
    }
    j = 0;
    while (j < n) {
      invert[i][j] -= coeff[i][k] * invert[k][j];
      j++;
    }
  }
  i++;
}
```

```
gcc rlc_integrate.c -o rlc_integrate -lm
./rlc_integrate 6.2832 10 0 0 0 10 0.01 >| rlc_integrate.dat
less rlc_integrate.dat
```

```
plot 'rlc_integrate.dat' using 1:2 with lines, '' using 1:3 with lines
```

Language is the only mechanism to effectively interact with a processor to solve complex numerical problems

Reference Resources on Class Web Page

Basic Linux commands:
http://www.physics.smu.edu/fattarus/LinuxCommands.html

bash shell key commands:
http://www.physics.smu.edu/fattarus/BashCommands.html

less pager key commands:
http://www.physics.smu.edu/fattarus/LessCommands.html

gnuplot commands:
http://www.physics.smu.edu/fattarus/GnuplotCommands.html

Table of hexadecimal digits:
http://www.physics.smu.edu/fattarus/HexDigits.html

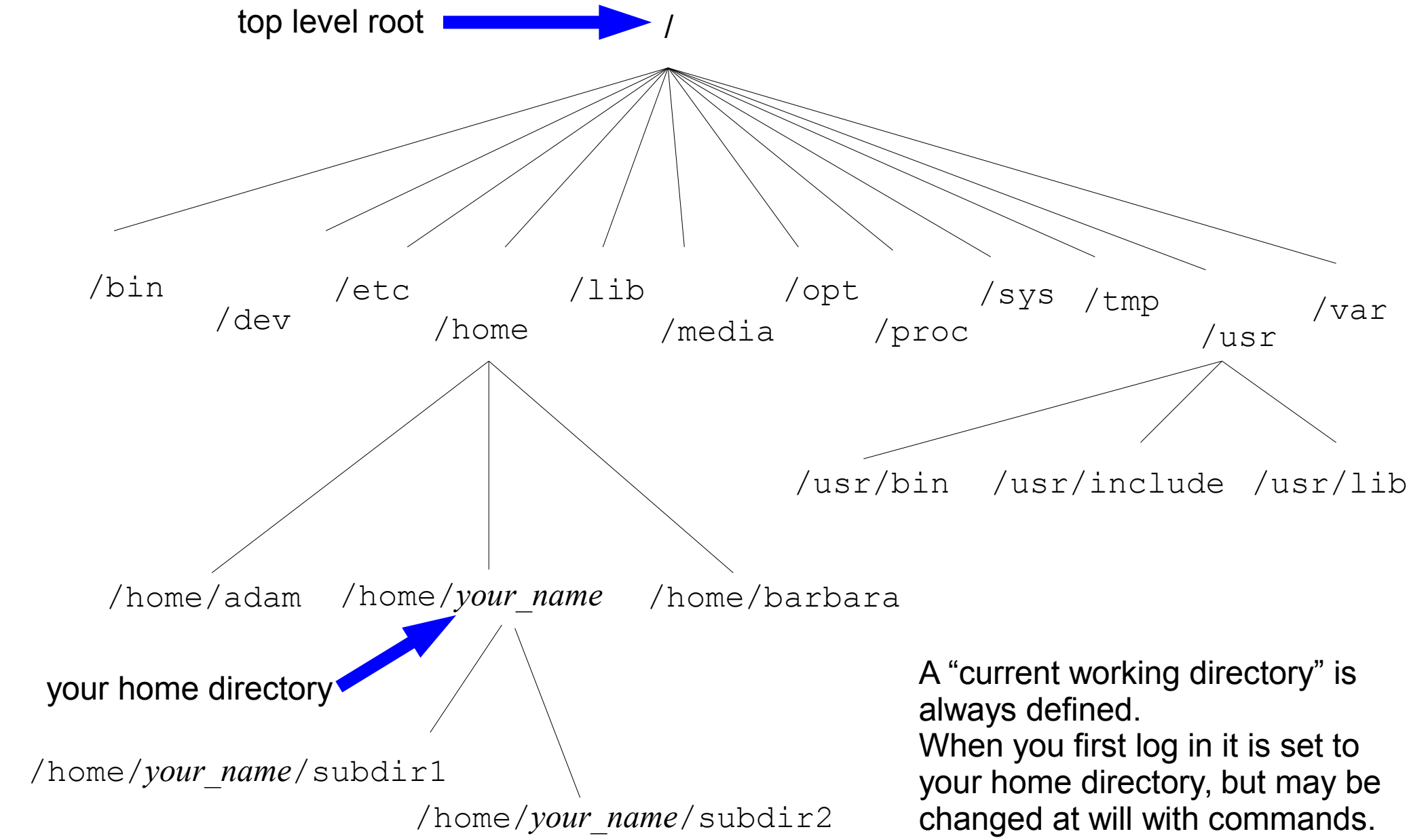
Linux command tutorials:
http://www.physics.smu.edu/fattarus/LinuxTutorial1.html
http://www.physics.smu.edu/fattarus/LinuxTutorial2.html

A Good On-Line Tutorial

`http://www.ee.surrey.ac.uk/Teaching/Unix/index.html`

In addition,
`http://www.tuxfiles.org/linuxhelp/linuxcommands.html`
displays a nice cheat-sheet

Linux Directory Tree Structure



A "current working directory" is always defined.
When you first log in it is set to your home directory, but may be changed at will with commands.

Referring to Files

Absolute, or “fully-qualified” file names:

`/home/my_name/larry`

`/home/my_name/mysubdir/moe`

Note the `/` characters as directory separators

For files in or below the current working directory:

`larry`

`mysubdir/moe`

Use the pseudo-files present in every directory:

`.` refers to current directory `..` refers to one directory level above

to form relative file names:

`../subdir1/larry`

`../../subdir1/subdir2/moe`

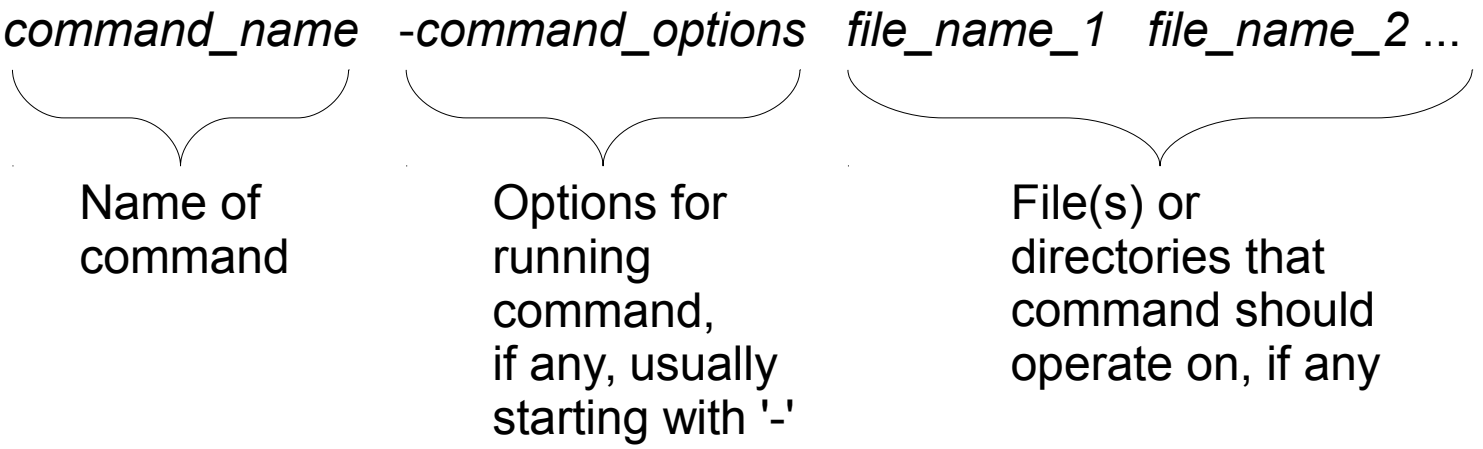
`./curly`

`../subdir1/../subdir2/../subdir1/../subdir2/../subdir1/shemp`

Use the special character '~' as an alias for your home directory:

`~/subdir1/larry`

General Command Syntax



Examples:

```
cp -f larry moe
```
Copy file 'larry' into file 'moe', with a 'force' option (overwrite file 'moe' if it exists)

```
ls -l curly
```
List files in directory 'curly' in long format

```
rm -i shemp
```
Remove file 'shemp' using interactive mode (verify before file is deleted)

Locating Command Executable Files

Each command name is actually the name of the executable program file on the system. These may be programs that come as a standard part of the Linux system or your own executable file generated by compiling your program code.

Show the directories Linux will search for executable program files when a command is entered with

```
echo $PATH
```

If an executable program file is in one of these directories, simply the command name is sufficient. If it is in another directory somewhere, then a full path name (absolute or relative) must be given. If it is in the current directory that means it must be given as

```
./file argument1 ... argumentN
```

File Name “Wildcard” or “Globbing” Syntax

In a command file name, the `?` character matches any single character
In a command file name, the `*` character matches any sequence of characters, including possibly nothing

Example: Suppose in the current directory there are files
`abc.a abc.b abcdef.a abcdef.b abpq.a abrs.b cba.a cba.b`

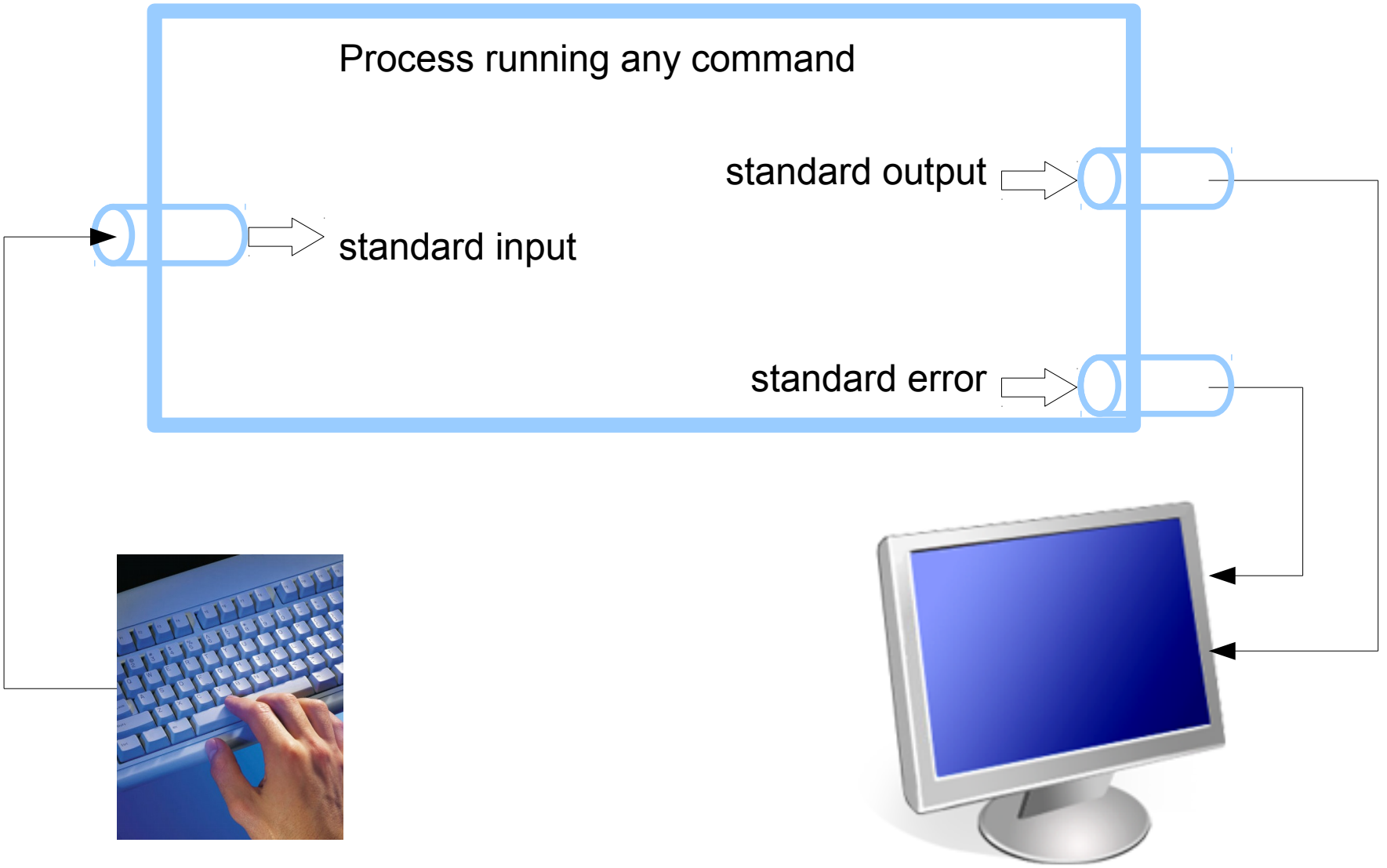
Then when used as a file name on a command line

`abc*` expands to `abc.a abc.b abcdef.a abcdef.b`
`abcdef*` expands to `abcdef.a abcdef.b`
`abc*.a` expands to `abc.a abcdef.a`
`*.b` expands to `abc.b abcdef.b abrs.b cba.b`
`ab*` expands to `abc.a abc.b abcdef.a abcdef.b abpq.a abrs.b`
`*` expands to `abc.a abc.b abcdef.a abcdef.b abpq.a abrs.b cba.a cba.b`
`abc.?` expands to `abc.a abc.b`
`ab??.?` expands to `abpq.a abrs.b`

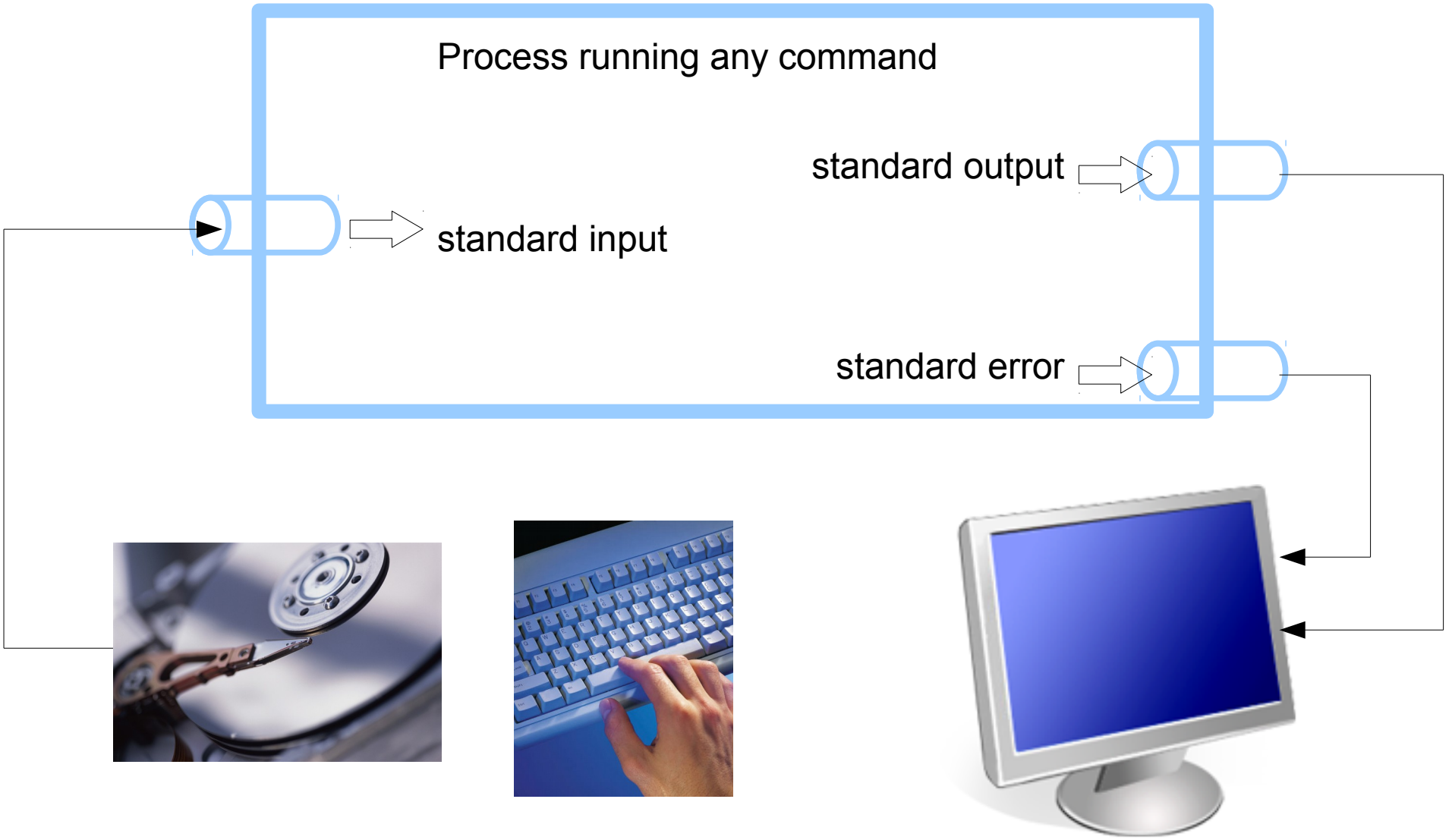
Globbing can refer to subdirectories as well as files

Be careful! Use an `echo` or `ls` command to first verify what files will be matched by your globbing string

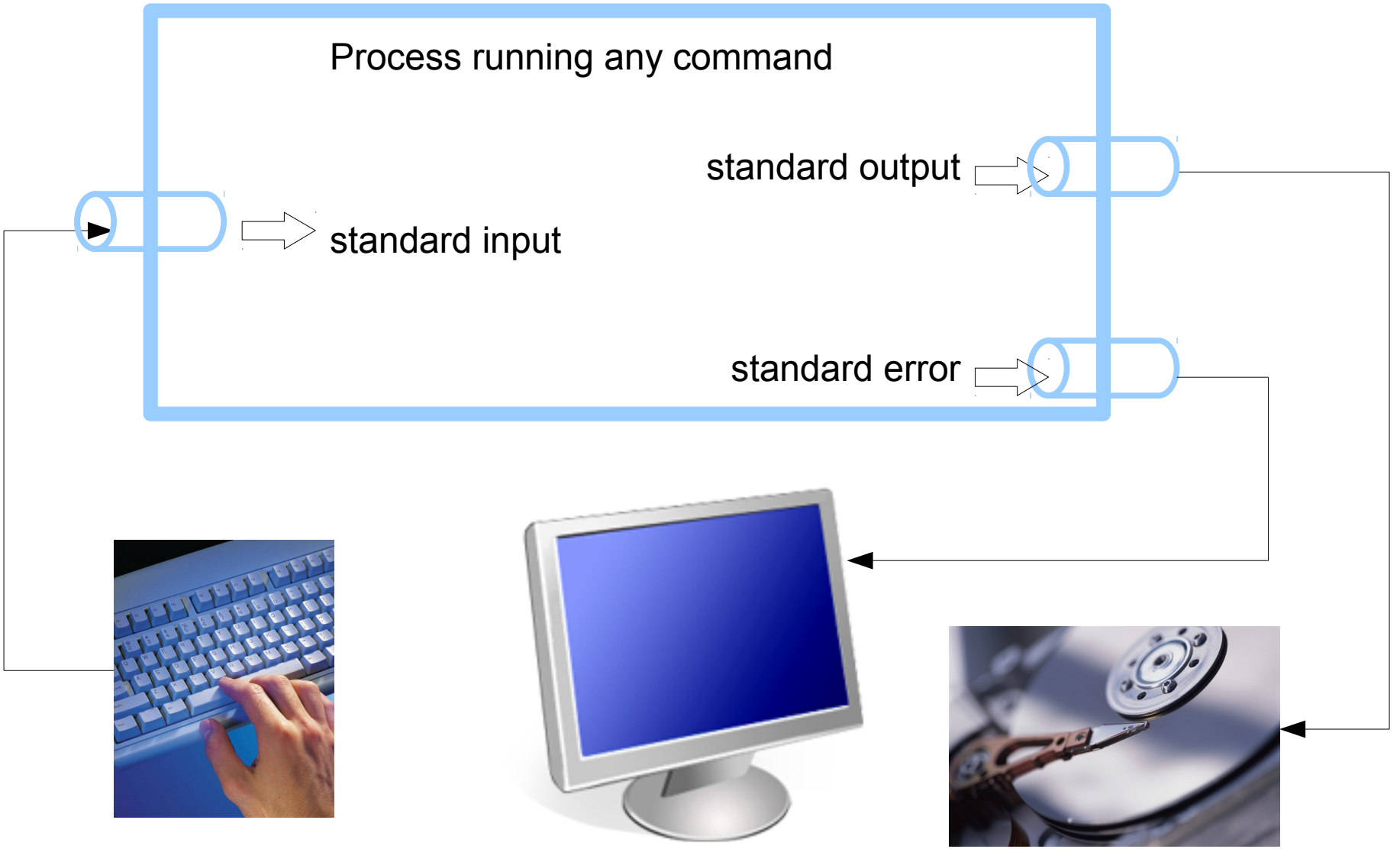
Default Connections for Standard Input and Output



Redirecting Standard Input from a Disk File



Redirecting Standard Output to a Disk File

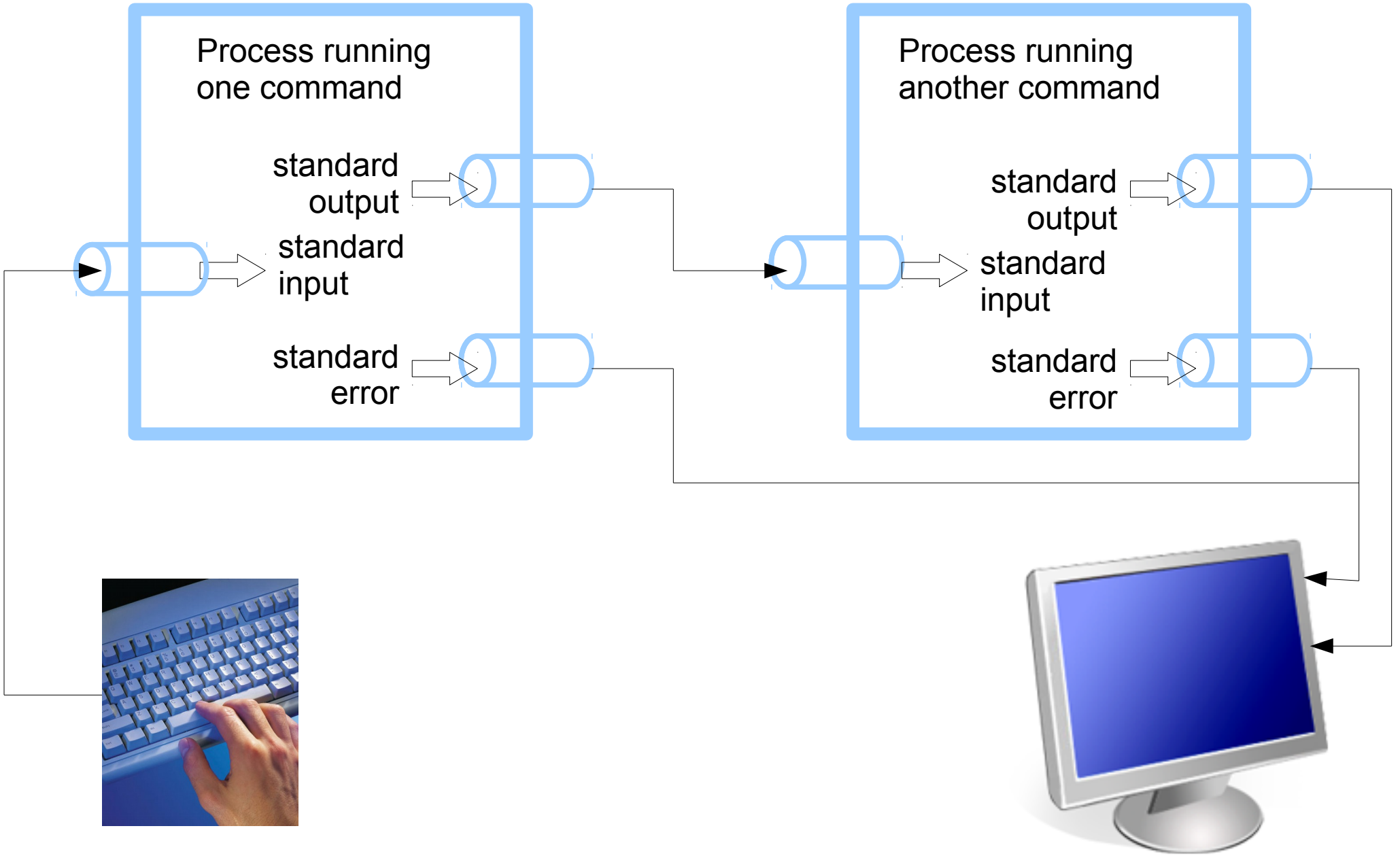


Redirecting output and input on the Command Line

`ls > file_list.txt` to write results of ls command to file
`ls >> file_list.txt` to append results of ls command to contents already in file
`ls >| file_list.txt` to overwrite results of ls command to existing file

`command < data.txt` to feed contents of a file into standard input of a command

Piped Connections for Standard Input and Output



Using pipes on the Command Line

```
ls /usr/bin | less
cat linear.* | less
ls *.c | wc
grep double /usr/include/*.h | less
cat solve*.c | grep maxwell
```

Retrieving Result Codes from Programs

Every program returns an result code to the shell when it exits. The result code is an integer value, zero for a successful execution and nonzero to report an unsuccessful completion. It is placed in the special shell variable named '$?'. The shell command 'echo' can be used to display the contents of all shell variables, including '$?'.

Examples, in a working directory containing text files but no C source files:

```
$ ls *.txt
a.txt b.txt c.txt
$ echo $?
0
$ ls *.c
ls: cannot access *.c: No such file or directory
$ echo $?
2
$ grep John names.txt
John Smith
$ echo $?
0
$ grep Mary names.txt
$ echo $?
1
$
```

Looking Up On-line Manual Pages

- Type "`man 1` *command_name*" to look up documentation of a shell command, with output automatically viewed in "`less`"
- Type "`man 3` *function_name*" to look up documentation of a C function, with output automatically viewed in "`less`"