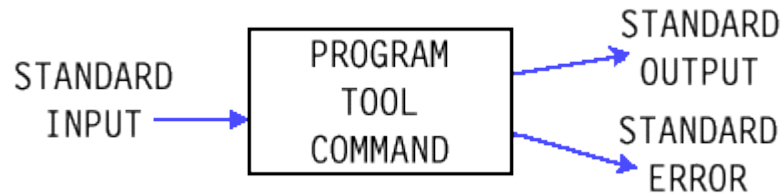


I/O Redirection and Filters

- ◆ Each UNIX command is written as a general purpose program.
- ◆ Inputs come from *STDIN*; outputs go to *STDOUT*.



File Descriptors

- ◆ When a UNIX command is invoked, the shell automatically open 3 files which that command will use for I/O.

File Descriptor	Common Name	Default File
0	STDIN	Keyboard
1	STDOUT	CRT
2	STDERR	CRT

- ◆ Unless you specify differently, the default files will be opened.
- ◆ These default files can easily be changed when the command is invoked.

Redirecting Output

- ◆ To redirect the output of the *ls* command to go into a disk file instead of going to the terminal screen.

`ls -A > MYFILE`

- ◆ To see the contents of *MYFILE*:

```
cat MYFILE
```

```
.profile
adir
bdir
report
code
myfile
```

- ◆ When the output of the *ls* command is redirected into *MYFILE*, the shell opens the following file descriptors.

File Descriptor	Common Name	File
0	STDIN	Keyboard
1	STDOUT	<i>MYFILE</i>
2	STDERR	CRT

Redirecting Input

- ◆ Redirection of input is not as common as redirection of output.
- ◆ Most commands are designed to take their input from files anyway, instead of from *STDIN*.
- ◆ If you call up a command that expects a filename, and you don't provide one, input will come from the keyboard until a *Ctrl-D* is read.
- ◆ Example of a command using *STDIN* as the input file.

```
cat
```

```
Some text is typed here.
```

This text will be stored up and sent to *STDOUT* when a *Ctrl-D* is read.

```
^D (user pressed Ctrl-D)
```

```
Some text is typed here.
```

- ◆ Or you could put the above text into *MYFILE* and direct *cat* to use *MYFILE* as *STDIN*.

```
cat < MYFILE
```

Redirecting Diagnostic Output

- ◆ Let's say you wanted to compile a C program, and send all of the error messages to *ERRORFILE*, rather than to the screen.

```
cc thisfile.c 2>ERRORFILE
```

- ◆ In the case of *STDERR*, you must precede the greater-than by the file descriptor number 2.
- ◆ The file descriptor for *STDOUT* is number 1, and is optional for redirecting.

I/O Redirection

- ◆ The following commands are equivalent:

```
cat < MYFILE  
cat < MYFILE  
cat < MYFILE  
cat < MYFILE
```

- ◆ The following commands are equivalent:

```
ls > MYFILE  
ls > MYFILE
```

Appending

- ◆ This is a special case of redirecting *STDOUT*.
- ◆ Instead of truncating the output file to length zero and directing output to it, the output is appended onto the end of the output file.

◆ Example:

```
cat MYFILE
THIS TEXT EXISTS.
pwd >> MYFILE
cat MYFILE
THIS TEXT EXISTS.
/usr/user1
```

Redirecting Both STDOUT and STDERR

To redirect both STDOUT and STDERR into different files:

```
command 1>output_file 2>error_file
```

In this case, you can run the command *in the background*, since all output will be saved, and you don't have to wait for the command to finish.

```
command 1>output_file 2>error_file &
```

To redirect both STDOUT and STDERR to the same file:

```
command 1>output_file 2>&1
```

If you are using Bash, you can do this in a shorter command line:

```
command >&output_file
```

Discarding STDOUT and/or STDERR

If you want to ignore either or both of the STDOUT or STDERR output, you can redirect it to the special file, called `/dev/null` (a.k.a. the “bit bucket”).

Example:

```
ls -lR >everything 2>/dev/null
```

will throw away any error messages for directories where you don't have read or execute permission, only files and directories that are “visible” to you will be saved.

Pipes



- ◆ Pipes allow the output of one command to become the input to another command.
- ◆ The following example will sort *MYFILE*, and send it to the *lpr* program to be spooled to the printer.

```
cat MYFILE | sort | lpr
```

I/O Redirection vs. Pipes

> < >> 2>	Redirect the output of a command, input to a command, or diagnostic output into some file or I/O device.
	Redirect the output of a command to become the input to another command.

command > *filename*

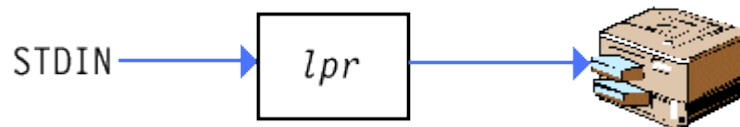
command < *filename*

command >*filename1* 2>*filename2*

command1 | *command2*

Filters

- ◆ Most commands in UNIX are filters.
- ◆ Filters take their input from *STDIN* and put their output to *STDOUT*.
- ◆ Filter commands can occur between two `|`'s in a pipeline.
- ◆ *lpr* is not a filter. Its output must always be to a printer.



tee

- ◆ *tee* is a program which sends its inputs to both *STDOUT* and a file.

```
ls | tee DIRECTORY
```

- ◆ *Since it is a command, you can view the manual page for tee:*

```
man tee
```

- ◆ *tee* can also be used as a filter.

```
ls | tee files | wc -l
```

- ◆ Either way, *tee* (almost) always has a pipe on its left.
- ◆ If *tee* has a pipe on its right, it writes the output to both the filename given and to the pipe, which feeds it to the command immediately following the *tee* (to the right of the pipe.)
- ◆ If *tee* doesn't have a pipe on its right, it writes the output to both the filename given and to *STDOUT*.

