I/O and Redirection

Standard I/O (1)

Normally each Unix process has three streams opened when it starts; one for input, one for output, and one for diagnostic or error messages.

- Standard Input (stdin)
  - default place from which programs read
  - File descriptor: 0
- Standard Output (stdout)
  - default place to which programs write
  - File descriptor: 1
- Standard Error (stderr)
  - default place where errors are reported
  - File descriptor: 2

Standard I/O (2)

- For terminal,
  - The default standard input is the keyboard,
  - The default standard output is the display
  - The default standard error is the display.
- To demonstrate -- cat
  - Echoes everything you typed in with an <enter>
  - Quits when you press Ctrl-d at a new line -- (EOF)
- To redirect (change) the default
  - Use < to redirect standard input (the same as 0<)
  - Use > to redirect standard output (the same as 1>)
  - Use 2> to redirect standard error

Redirecting Standard Output

- cat file1 file2 > file3
  - concatenates file1 and file2 into file3
  - file3 is created if not there
- cat file1 file2 >| file3
  - file3 is clobbered if there
- cat file1 file2 >> file3
  - warning if file3 is not there
  - file3 is appended to if it is there
- cat > file3
  - file3 is created from whatever user provides from standard input

Redirecting Standard Error

- To write standard output and standard error into different files:

```
compute[1] > cat myfile > yourfile 2> yourerrorfile
```

- Generally direct standard output and standard error to the same place:

```
compute[2] > cat myfile &> yourfile
```

  - If myfile exists, it is copied into yourfile
  - If myfile does not exist, an error message

    ```
    cat: myfile: No such file or directory
    ```

    is copied into yourfile

- A more general way is
  - `cat myfile > yourfile 2>&1`
  - stdout goes to yourfile and stderr goes to where stdout goes

Redirecting Standard Input

- compute[1] > cat < oldfile > newfile
- A more useful example:
  - compute[2] > tr string1 string2
    - read from standard input.
    - character *n* of string1 translated to character *n* of string2.
    - results written to standard output.
  - Example of use:

```
compute[3] > tr aeoiu eoiua <file1 >file2
compute[4] > tr eoiua aeoiu <file2 >file3
compute[5] > tr a-z A-Z < file1 > file2
```

/dev/null

- /dev/null
  - A virtual file that is <u>always</u> empty.
  - Copy things to here and they disappear.
    - `cp myfile /dev/null`
  - Copy from here and get an empty file.
    - `cp /dev/null myfile`
  - Redirect error messages to this file
    - `ls -l > recordfile 2> /dev/null`
    - Basically, all error messages are discarded.

Filters (1)

- Filters are programs that:
  - Read from stdin.
  - Modify it (may do nothing).
  - Write the results to stdout.
- Filters typically do not need user input.
- Example:
  - tr (translate):
    - Read stdin
    - Echo to stdout, translating some specified characters
- Many filters can also take file names as operands for input, instead of using stdin.

Filters (2)

- grep patternstr:
  - Read stdin and write lines containing patternstr to stdout

```
compute[1] > grep "unix is easy"  < myfile1  > myfile2
```

  - Write all lines of myfile1 containing phrase ***unix is easy*** to myfile2
- wc:
  - Count the number of chars/words/lines on stdin
  - Write the resulting statistics to stdout
- sort:
  - Sort all the input lines in alphabetical order and write to the standard output.

Pipes

- The pipe:
  - Connects stdout of one program with stdin of another
  - General form:

    ```
    command1 | command2
    ```

    - stdout of command1 used as stdin for command2
  - Example:

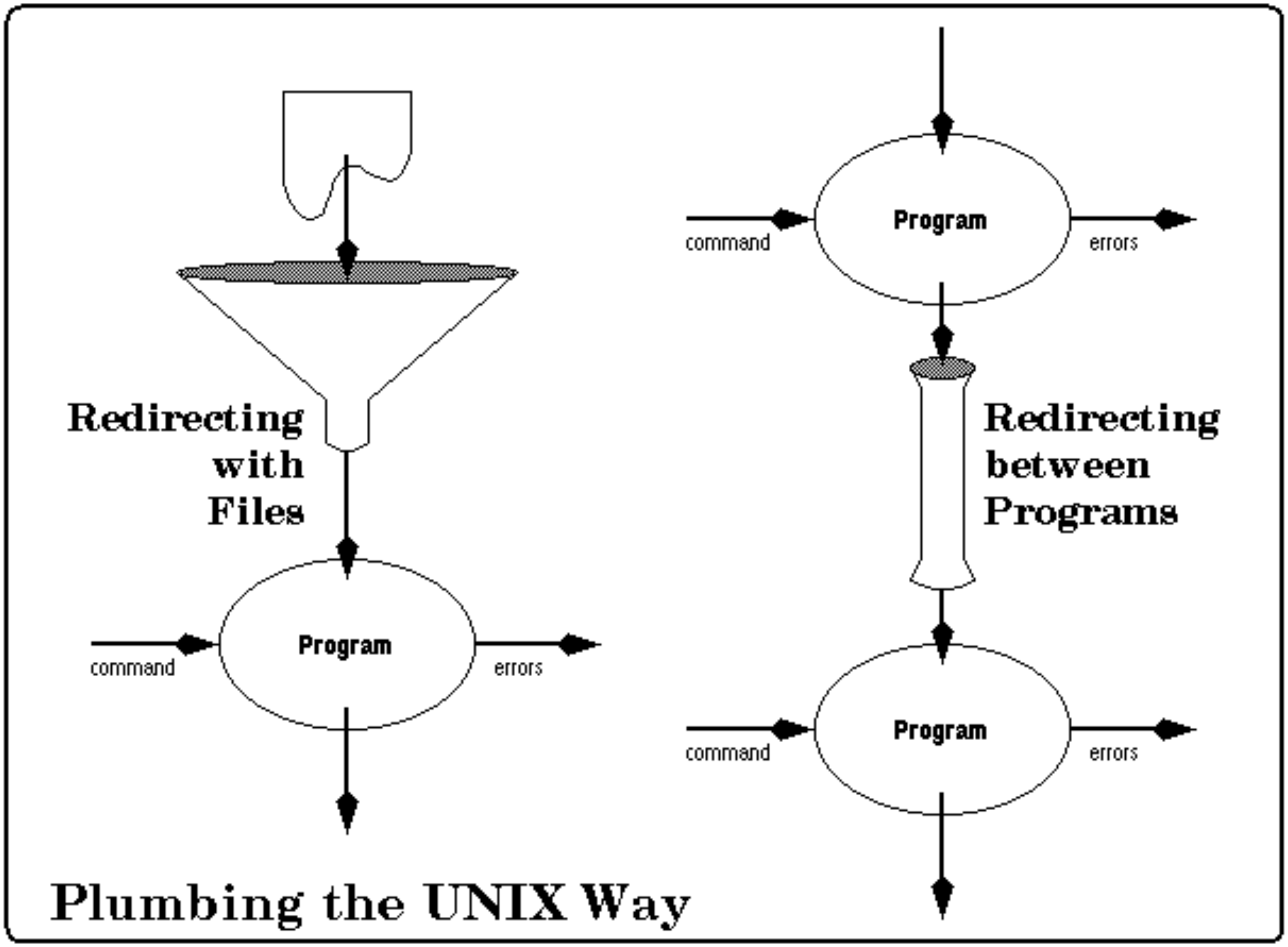
    ```
    compute[1] > cat readme.txt | grep unix | wc -l
    ```

- An alternative way (not efficient) is to:

  ```
  compute[2] > grep unix < readme.txt > tmp
  compute[3] > wc -l < tmp
  ```

- Can also pipe stderr: `command1 |& command2`

Redirecting and Pipes (1)



Redirecting and Pipes (2)

- Note: The name of a command always comes first on the line.
- There may be a tendency to say:

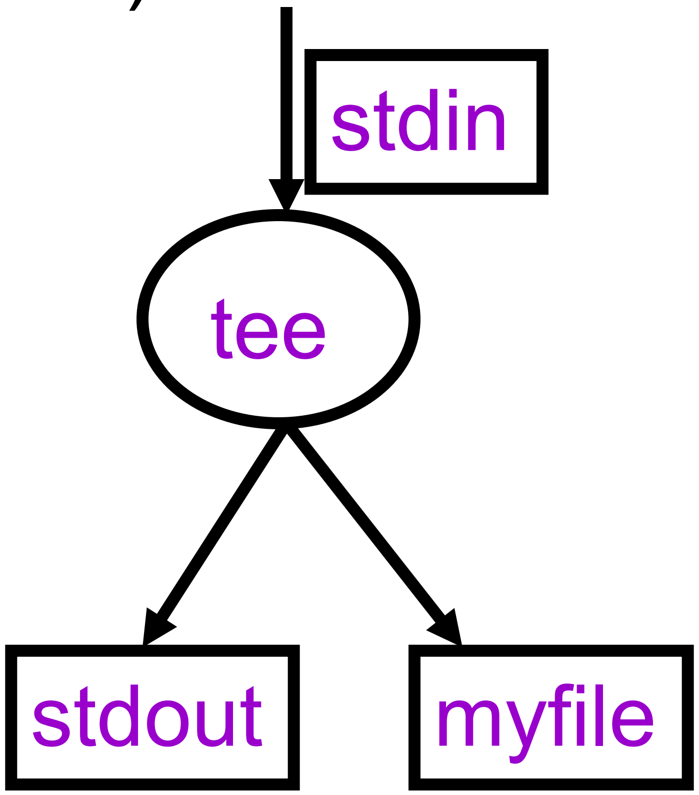
```
compute[1] > readme.txt > grep unix | wc -l
```

  - This is WRONG!!!
  - Your shell will go looking for a program named readme.txt
- To do it correctly, many alternatives!

```
compute[2] > cat readme.txt | grep unix | wc -l
compute[3] > grep unix < readme.txt | wc -l
compute[4] > grep unix readme.txt | wc -l
compute[5] > grep -c unix readme.txt
```

The tee Command

- tee - replicate the standard output
  - cat readme.txt | tee myfile
  - tee –a (append)


stdin
tee
stdout
myfile