I/O and Redirection
Standard I/O

◆ Standard Output (stdout)
  – default place to which programs write

◆ Standard Input (stdin)
  – default place from which programs read

◆ Standard Error (stderr)
  – default place where errors are reported

◆ To demonstrate -- cat
  – Echoes everything you typed in with an <enter>
  – Quits when you press Ctrl-d at a new line -- (EOF)
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`
  - file3 is created if 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

- Generally direct standard output and standard error to the same place:
  - `obelix[1] > 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 in `yourfile`

- In tcsh, to write standard output and standard error into different files:
  - `obelix[2] > (cat myfile > yourfile) >& yourerrorfile`

- In sh (for shell scripts), standard error is redirected differently
  - `cat myfile > yourfile 2> yourerrorfile`
Redirecting Standard Input

- obelix[1] > cat < oldfile > newfile
- A more useful example:
  - obelix[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:
    - obelix[3] > tr aeoiu eoiua
    - obelix[4] > tr a-z A-Z < file1 > file2
– A virtual file that is **always** empty.
– Copy things to here and they disappear.
  
  ```
  cp myfile /dev/null
  mv myfile /dev/null
  ```
– Copy from here and get an empty file.
  
  ```
  cp /dev/null myfile
  ```
– Redirect error messages to this file
  
  ```
  (ls -l > recordfile) >& /dev/null
  ```
  
  Basically, all error messages are discarded.
Filters are programs that:
- Read stdin.
- Modify it.
- 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
  obelix[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:
    \[\text{command1} \mid \text{command2}\]
    – stdout of command1 used as stdin for command2
  – Example:
    \[\text{obelix}[1] > \text{cat readme.txt} \mid \text{grep unix} \mid \text{wc -l}\]

◆ An alternative way (not efficient) is to:
  \[\text{obelix}[2] > \text{grep unix < readme.txt > tmp}\]
  \[\text{obelix}[3] > \text{wc -l < tmp}\]

◆ Can also pipe stderr: \textbf{command1 }\& \textbf{command2}
Redirecting and Pipes (1)

Redirecting with Files

Plumbing the UNIX Way

Redirecting between Programs
Redirecting and Pipes (2)

◆ Note: The name of a command always comes first on the line.

◆ There may be a tendency to say:
  obelix[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!
  obelix[1] > cat readme.txt | grep unix | wc -l
  obelix[2] > grep unix < readme.txt | wc -l
  obelix[3] > grep unix readme.txt | wc -l
The tee Command

- **tee** - replicate the standard output
  - `cat readme.txt | tee myfile`