Lecture 4

Redirecting standard I/O & Pipes

COP 3353 Introduction to UNIX

Standard input, output and error

- standard input (0: stdin)
  - The default place where a process reads its input (usually the terminal keyboard)
- standard output (1: stdout)
  - The default place where a process writes its output (typically the terminal display)
- standard error (2: stderr)
  - the default place where a process can send its error messages (typically the terminal display)

Redirecting standard I/O

- Standard input and output can be redirected providing a great deal of flexibility in combining programs and unix tools
- Can redirect standard input from a file using <

```
a.out < input12
```

  - any use of stdin will instead use input12 in this example
- Can redirect standard output to a file using >

```
testprog1 > testout1
cal > todaycal
a.out < input12 > testout
```

  - the stdout of a.out is directed to file testout1 in this example
- Can also redirect stderr and / or stdout at the same time

Appending to a file

- The >> operator *appends* to a file rather than redirecting the output to a file

```
cat textinfo >assign4
prog1.exe >>assign4
prog2.exe >>assign4
cat endinfo >>assign4
```

Pipes

- Pipes allow the standard output of one program to be used as the standard input of another program
- The pipe operator '|' takes the input from the command on the left and feeds it as standard input to the command at the right of the pipe
- Examples

```
ls | sort -r
prog1.exe < input.dat | prog2.exe |
  prog3.exe >output.dat
ls -l | cut -c 38-80
```

- Pipes are more efficient as compared to using intermediate files

Another Example

```
du -sc * | sort -n | tail
```

- The *du* command is for disk usage (default is in blocks of 512 bytes). The s and c flags are for summarize and give a grand total respectively
- the *sort* -n command will sort by numeric value
- *head* and *tail* commands print out a few lines at the head or tail of the file respectively
- http://learnlinux.tsf.org.za/courses/build/shell-scripting/ch01s04.html

Separating commands

- Multiple instructions on one line
  - separate instructions by ‘;’

```
ls -l; cal; date
```

- Suppose you need to continue a command to the next line - use the ‘\’ to do so and then continue your command on the next line

```
cat filename | sort \
| wc
```