Lab 2

Unix - Tools of the Trade

Ref: Unix Shell Programming by Kochan and Wood

CSCI-2500 - Fall 2015

Lab Objectives

- TA will lecture for about 30 minutes on Unix topics
- Practice working with file substitutions, standard input, standard output, and pipes
- Learn how basic regular expressions are used in Unix programs (pattern matching)
- Explore common Unix command programs
- Use the Unix octal dump tool to examine file contents
- Complete the lab exercise sheet for credit

File Substitution (Globbing)

- Suppose you want to print the contents of all your files to the terminal. You could type all the file names with the `cat` command:

```
bash-3.2$ ls
lab1 lab2 lab3 lab4
bash-3.2$ cat lab1 lab2 lab3 lab4
. . . .
```

- But you can also get the same results if you type the following:

```
bash-3.2$ cat *
. . . .
```

- This also works with the echo command:

```
bash-3.2$ echo *
lab1 lab2 lab3 lab4

bash-3.2$ echo * : *
lab1 lab2 lab3 lab4 : lab1 lab2 lab3 lab4
```

File Substitution (Globbing)

- The * can also be used in combination with other characters to limit the filenames that are substituted. Suppose we have the following files in our current directory:

```
bash-3.2$ ls
a
b
c
lab1
lab2
lab3
lab4
```

- To display the contents of just the files beginning with lab, you can type:

```
bash-3.2$ cat lab*
. . . .
```

File Substitution (Globbing)

- The * is not limited to the end of the file name; it can be used at the beginning or in the middle as well:

```
bash-3.2$ echo *ab3
lab3
```

- The asterisk (*) matches zero or more characters, meaning that x* matches the file x, as well as x1, x2, xabc, and so on. To question mark (?) matches exactly one character:

```
bash-3.2$ ls
a
b
c
lab1
lab2
lab3
lab4
```

```
bash-3.2$ echo ?
a  b  c
```

File Substitution (Globbing)

- Another way to match a single character is to give a list of the characters in brackets `[]`. For example `[abc]` matches one letter `a`, `b`, or `c`. You can also specify a range of characters in the brackets (e.g. `[0-9]` matches the character `0` *through* `9`.

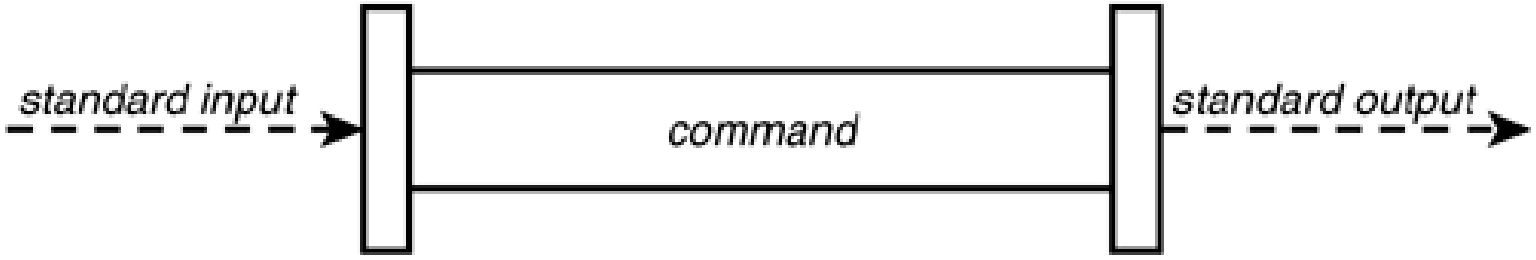
- If the first character following the `[` is a `!` then the mixing sense of the match is inverted. That is, any character is matched except those enclosed in the brackets. So `[!A-Z]` matches any character except a uppercase letter, and `*[!p]` matches any file that doesn't end with the lowercase letter p.

File Substitution (Globbing)

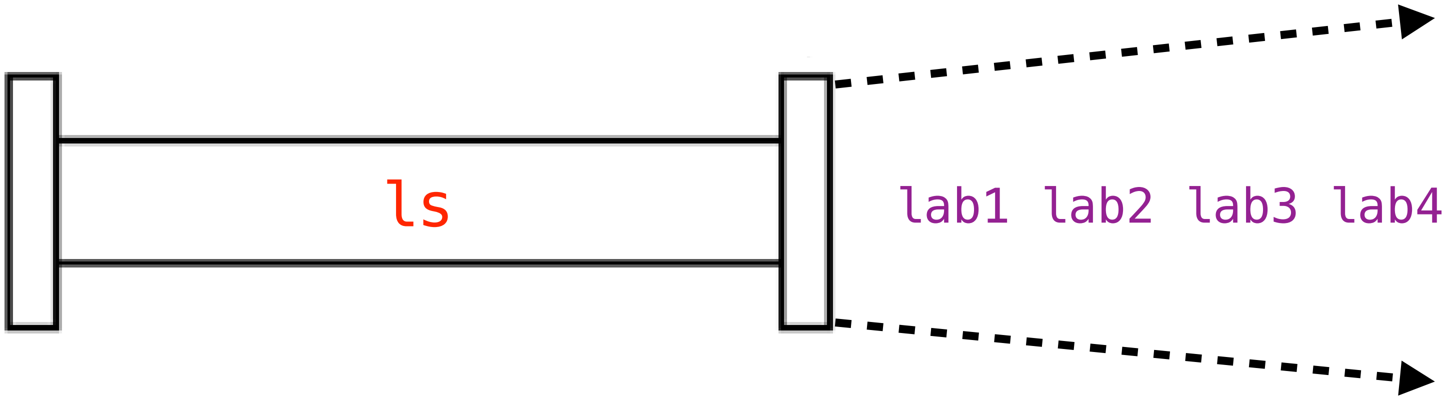
Example	***Result***
`echo a*`	Print the names of the files beginning with a
`cat *.c`	Print all files ending .c
`rm *.*`	Removes all files containing a period
`ls x*`	List all files beginning with x
`rm *`	Remove all files in the current directory (Careful!)
`echo a*b`	Print the names of all files beginning with a and ending with b
`cp ../programs/* .`	Copy all files from ../programs into the current directory
`ls [a-z]*[!0-9]`	List files that begin with a lowercase letter and don't end with a digit

Standard Input/Output

- Most Unix system commands take input from the terminal input (keyboard) and send the resulting output back to the terminal:



- Recall the `ls` command. The `ls` command queries the file system for a list of files and sends the results to standard output. There is no standard input:



Standard Input/Output

- Now recall the `wc` command. If the `wc` command is invoked without an input file argument, `wc` will attempt to get input from standard input:

```
bash-3.2$ wc -l
This is text that
is typed on the
standard input device.

[Ctrl+d]

3

bash-3.2$
```

Need to enter 'Ctrl-D' to signify the end of standard input

command standard output

Standard Output Redirection

- Standard output can be directed away from the terminal by using the > or >> operators.
- The > operator can be used to direct standard output to a file (will overwrite contents of an existing file):

```
bash-3.2$ ls lab[1-2] > output_file
bash-3.2$ cat output_file
lab1
lab2
```

- The >> operator can also be used to direct standard output to a file but will append to an already existing file:

```
bash-3.2$ ls lab[1-2] >> output_file
bash-3.2$ cat output_file
lab1
lab2
lab1
lab2
```

Input Redirection

- The input of a command be also be redirected from a file using the < operator. Of course, only commands that normally take their input from standard input can have their input redirected from a file in this manner. Let's look at some examples from the `wc` command: Command with standard file argument:

```
bash-3.2$ wc -l lab1
     125 lab1
```

Command with input redirection (notice the difference in output):

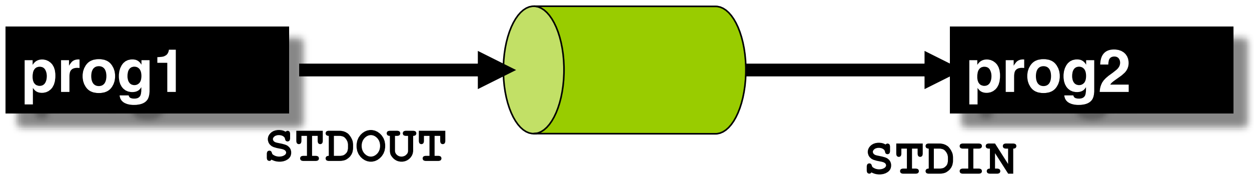
```
bash-3.2$ wc -l < lab1
     125
```

Note: It is possible to combine input and output redirection

```
bash-3.2$ wc -l < lab1 > wc_output
bash-3.2$ cat wc_output
     125
```

Pipes

- The Unix system allows a user to effectively connect two commands together. This is known as a pipe and allows you to take the output from one command and feed it directly into the input of another command. A pipe is represented by the operator |.



- Example of a pipe between the ls and wc commands:

```
bash-3.2$ ls
lab1 lab2 lab3 lab4
bash-3.2$ ls | wc -l
     4
```

Standard Error

- In addition to standard input and output there is another place where most Unix commands write their error messages. It is also associated with the terminal by default. Example:

```
bash-3.2$ ls
lab1 lab2 lab3 lab4
bash-3.2$ ls n*
n* not found
```

- Verify n* not found is NOT standard output:

```
bash-3.2$ ls n* > out_file
n* not found
bash-3.2$ cat out_file
bash-3.2$
```

- Users must redirect standard error differently:

```
bash-3.2$ ls n* 2> error_file
bash-3.2$ cat error_file
n* not found
```

Tips for Unix Commands

- Unix allows users to type more than one command per line (use the semicolon to separate them). Example for getting your current working directory and the date:

```
bash-3.2$ date; pwd
Sat Jan 31 12:15:30 EST 2015; /home/calonge
```

- Commands can also be sent to the background for processing (i.e. users don't have to wait for them in the terminal). Use the & operator after a command to do this (note: the # 1258 below is the process id):

```
bash-3.2$ cat massive_file > another_file &
[1] 1258
bash-3.2$
```

- A list of background jobs is provided by the `jobs` command (list all processes with the `ps` command):

```
bash-3.2$ jobs
[1] Running cat massive_file > another_file &
```

Unix and Regular Expressions

- Many Unix tools and regular expressions go hand-in-hand. Regular expressions provide a consistent and convenient way of specifying patterns to be matched.
- Note: The shell recognizes a limited form of regular expressions when you use filename substitution (i.e. globbing).
- Regular expressions recognized by most Unix programs are far more sophisticated than those recognized by the shell.
- Please be advised that the asterisk (∗) and the question mark (?) characters are treated differently by Unix commands/programs than by the shell.

Common Regular Expressions

Notation	***Meaning***	***Example(s)***	***Matches***
.	any character	a..	a followed by any two characters
^	beginning of line	^lab	lab only if it appears at the beginning of a line
$	end of line	x$ ^INSERT$ ^$	x only if it is the last character on the line a line containing just the characters INSERT a line that contains no characters

Common Regular Expressions

Notation	*Meaning*	*Example(s)*	*Matches*
*	zero or more occurrences of previous regular expression	x*	zero or more consecutive x ’s
		xx*	one or more consecutive x ’s
		.*	zero or more characters
		w.*s	w followed by zero or more characters followed by an s
[chars]	any character in chars	[sS]	lower- or uppercase s
		[a-z]	lowercase letter
		[a-zA-Z]	lower- or uppercase letter

Common Regular Expressions

Notation	***Meaning***	***Example(s)***	***Matches***
[^chars]	any character NOT in chars	[^0-9]	any nonnumeric character
		[^a-zA-Z]	any non alphabetic character
\{min,max\}	at least *min* and at most *max* occurrences of previous regular expressions	x\{3,\}	at least 1 and at most 5 x's
		[0-9]\{3,9\}	anywhere from 3 to 9 successive digits
		[0-9]\{3\}	exactly 3 digits
		[0-9]\{3,\}	at least 3 digits

Some Tools of the Trade

Command	***Description***	***Example(s)***
cut	extract exact columns or fields from a file	cut -c1-5,12-14 file > file1 cut -f1,3,5-9 file > file2
paste	the inverse of cut; puts lines of files together	paste file1 file2 > newfile
tr	translate command; replaces or removes specific characters from the standard input stream	tr a-z A-Z < file1 > file2
grep	utility for searching plain-text data sets for lines matching a regular expression	grep Chuck phonebook.txt grep ‘[A-Z]’ list.txt grep ‘[0-9]’ phonebook.txt grep ‘[A-Z]…[0-9]’ list.txt
sort	sorts a file or input from standard input	sort -n number_file.txt sort -u duplicates_file.txt sort phone_book.txt

Some Tools of the Trade

Command	***Description***	***Example(s)***
uniq	removes duplicate lines from a file or standard input	uniq duplicates_file.txt newfile uniq names_file.txt uniq -d duplicates_file.txt
od	(octal dump) used to output the contents of a file in different formats with the octal format being the default. Useful for working with binary files.	od -b input_file od -x input_file

- Other useful (and very powerful) Unix programs are sed (streaming editor) and awk (text processing and data extraction/reporting tool).

Steps for Lab Credit and Additional Reading

- Work through the problems on the exercise sheet
- Suggestions:
  - Use the command man pages
  - Experiment with command options
  - Ask questions if you are having trouble
- A comprehensive list of Unix commands can be found in “Linux in a Nutshell” by Siever et al.
- Other recommended references:
  - “Unix Shell Programming” by Kochan and Wood (Unix shell basics and scripting)
  - “sed and awk” by Dougherty and Robbins (authoritative guide on sed and awk programs)