CS3157: Advanced Programming

Lecture #12

Nov 28 Shlomo Hershkop shlomo@cs.columbia.edu

Outline

- · Update on webthumb
- Shell commands
- Time permitting:
 - dynamic memory allocation (new/delete vs malloc/free)
 - container classes
 - Iterator classesTemplates

 - polymorphism

 Reading - c++core ch 7-9,11-13

Announcements

- Final: 12/21 (wed) 1-4pm in class.
 - Will post details on web
 - Will hold a review session prior
 - Will post online sample questions

webthumb

- Issues
 - Most issues were related because people didn't really understand what they were using.
- Ideas!
- Problems - Local server

 - Ports - Memory frame buffers
 - Half screens
 - Systems
- Explanations

Schedule:

- Will now break from cpp, and cover unix utilities
- · Might have time for some software engineering background
- · Will cover php next week
- Last lab will also be final hw
 - Combine everything we've learned so far into small project using anything you want.

Useful tools & commands

- wc counts characters, words and lines in input
- grep matches regular expression patterns in input
- cut extracts portions of each line from input
- sort sorts lines of input
- · sed stream edits input
- ps displays process list of running processes
- who displays anyone logged in on the system

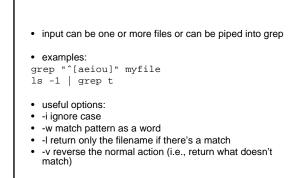
WC

- unix command: counts the number of characters/words/lines in its input input can be a file or a piped command (see below) •
- example:filename = "hello.dat"
- hello

world • usage: unix-prompt\$ wc hello.dat 2 2 12 hello.dat unix-prompt\$ wc -1 hello.dat 2 hello.dat unix-prompt\$ wc -c hello.dat 12 hello.dat unix-prompt\$ wc -w hello.dat 2 hello.dat

grep

- Global Regular Expression Parser
- · one of the most useful tools in unix
 - three standard versions:
 - plain old grep
 - extended grep: egrep
 fast grep: fgrep
- used to search through files for ... regular expressions!
- · prints only lines that match given pattern
- a kind of filter
- · BUT it's line oriented



• examples: grep -i "^[aeiou]" myfile grep -v "^[aeiou]" myfile grep -iv "^[aeiou]" myfile

- how do you list all lines containing a digit?
- how do you list all lines containing a 5?
- how do you list all lines containing a 0?
- how do you list all lines containing 50?
- how do you list all lines containing a 5 and an 0?

cut

- unix command: extracts portions of each line from input
- input can be a file or a piped command
- syntax: cut <-c|f> <-d>
- note that c and +f+ start with 1; default delimiter is TAB

sort

- unix command: sorts lines of input
- input can be a file or a piped command (see below) ٠
- three modes: sort, check (sort -c), merge (sort m)
- syntax: sort <-t> <-n> <-r> <-o> POS1 -POS2+
- note that POS starts with 0; default delimiter is • whitespace

sed

- stream editor · does not change the file it "edits"
- commands are implicitly global
 input can be a file or can be piped into sed
- example: substitute all A for B:
- sed 's/A/B/' myfile
 cat myfile | sed 's/A/B/'
- use the -e option to specify more than one command at a time:
 sed -e 's/A/B/' -e 's/C/D/' myfile
- pipe output to a file in order to save it: :
- sed -e 's/A/B/' -e 's/C/D/' myfile >mynewfile

sed

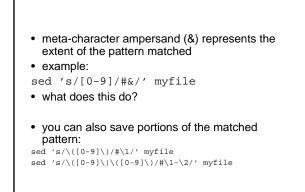
- sed can specify an address of the line(s) to affect
- if no address is specified, then all lines are affected
 if there is one address, then any line matching the address is affected
- if there are two (comma separated) addresses, then all lines between the two addresses
- are affected
- are anected
 if an exclamation mark (!) follows the address, then all lines that DON'T match the
 address are affected
- .
- addresses are used in conjunction with commands
- examples (using the delete (d) command):

- sed '\$d' myfile sed '/^\$/d' myfile sed '/,'under/d' myfile sed '/over/,/under/d' myfile

- · order of commands is important
- input is line oriented
- all editing commands are applied to each line, one at a ٠ time
- then next line is read and editing commands are applied to that linei
- etc
- · for example:
- sed -e 's/pig/cow/' -e 's/cow/horse' myfile
- what does this do?
- is this right???

- delimiter is slash (/)
- backslash (escape) it if it appears in the command, e.g.:

sed 's/\/usr\/bin\//\/usr\/etc/' myfile



- transformation command: y
- example:
- sed 'y/ABC/abc' myfile

```
• print command: p
```

• example:

```
sed '/begin/,/end/p' myfile
sed -n '/begin/,/end/p' myfile
```

```
• what do the following sed commands do?
sed 's/xx/yy' myfile
sed '/BSD/d' myfile
```

```
sed '/^BEGIN/,/^END/p@' myfile
```

- how do you change the content of all your html files to lowercase?
- how do you change all the html commands to lowercase?

shell

sh is the "Bourne shell", the first scripting language
it is a program that interprets your command lines and runs other programs

- it can invoke Unix commands and also has its own set of commands

while (1) {
 print prompt and wait for user to enter input;
 read input from terminal; parse into words; substitute variables; execute commands (execv or builtin); }

- shell commands can be read: - from a terminal == interactive

 - from a file == shell script
- · search path
 - the place where the shell looks for the commands it runs
 - should include standard directories:
 - /bin /usr/bin
 - it should also include your current working directory ()

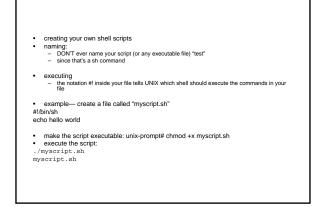
• are you running the Bourne shell? type:

\$SHELL

- if the answer is /bin/sh, then you are
- if the answer is /bin/bash, then that's close enough
- otherwise, you can start the Bourne shell by typing sh at the UNIX prompt ٠
- enter Ctrl-D or exit to exit the Bourne shell and go back to whatever shell you
- were running before...

- capable of both synchronous and asynchronous execution

 - synchronous: wait for completion
 asychronous: in parallel with shell (runs in the background)
- · allows control of stdin, stdout, stderr
- enables environment setting for processes (using inheritance between processes)
- · sets default directory



• quote (')

'something': preserve everything literally and don't evaluate anything that is inside the quotes

• double quote (") "something2": preserve most things literally, but also allow \$ variable expansion (but not ' evaluation)

• backquote (') 'something3': try to execute something as a command

- filename=t.sh #!/bin/sh hello='hi" echo 0=\$hello echo 1='\$hello" echo 3='\$hello" echo 3='\$hello" echo 4='\$hello'"

- filename=hi
 #!/bin/sh
 echo "how did you get in here?"

output= unix\$ t.sh 0=hi 1=\$hello 2=hi 2=hi 3=how did you get in here? 4=how did you get in here? 5='hi'

• single line comments only (no multi-line

shell comments

- comments)
- line begins with # character

Simple commands

- sequence of words
- first word defines command
- .
- first word defines command can be combined with &&, ||, ; to execute commands sequentially: cmd1; cmd2; to execute a command in the background : cmd1& to execute two commands asynchronously: cmd1& cmd2& to execute cmd2 if cmd1 has zero exit status: cmd1 && cmd2 to execute cmd2 only if cmd1 has non-zero exit status: cmd1 || cmd2
- set exit status using exit command (e.g., exit 0 or exit 1)

pipes

- sequence of commands
- connected with |
- · each command reads previous command's output and takes it as input
- example:
- echo "hello world" | wc -w

2

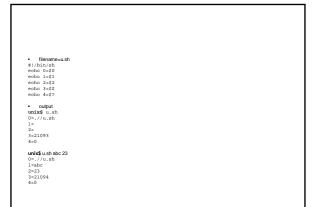
variables

- variables are placeholders for values
 shell does variable substitution
- shell does variable substitution
 \$var or \${var} is the value of the variable
 assignment:
- assignment:

 var-value (with no spaces before or after!)
 let "var = value"
 export var=value

 BUT values go away when shell is done executing
 uninitialized variables have no value

- variables are untyped, interpreted based on context
- standard shell variables: •
- \${N} = shell Nth parameter
 \$\$ = process ID
 \$? = exit status



- shell variables are generally not visible to programs
- . environment variables are a list of name/value pairs passed to sub-processes
- all environment variables are also shell variables, but not vice versa
- show with env or echo \$var
- · standard environment variables include: HOME = home directory

 - PATH = list of directory
 TERM = type of terminal (vt100, ...)
 TZ = timezone (e.g., US/Eastern)

Loops

- similar to C/Java constructs, but with commands
- until test-commands; do consequent-commands; done
- while test-commands; do consequentcommands; done
- for name [in words ...]; do commands; done
- also on separate lines
- break and continue control loop

• while

i=0 while [\$i -lt 10]; do echo "i=\$i" ((i=\$i+1)) # same as let "i=\$i+1" done

• for

for counter in `ls *.c`; do echo \$counter done

if if test-commands; then consequent-commands; [elif more-test-commands; then more-consequents;] [else alternate-consequents;] • colon (:) is a null command example #!/bin/sh #!/Din/sn
if expr \$TERM = "xterm"; then
echo "hello xterm"; else echo "something else"; fi

case test-var in value1) consequent-commands;; value2) consequent-commands;; *) default-commands; esac

- pattern matching:
- ?) matches a string with exactly one character
- ?*) inaccies a string with exactly one or more characters
 [?*) matches a string with one or more characters
 [yY][[YY][[e5][SS]] matches y, Y, yes, YES, yES...
 /*/*[0-9]) matches filename with wildcards like /xxx/yyy/zzz3
 notice two semi-colons at the end of each clause
- stops after first match with a value
- you don't need double quotes to match string values!

example

#!/bin/sh case "\$TERM" in xterm) echo "hello xterm";; vt100) echo "hello vt100";; *) echo "something else";; esac

biggest difference from traditional programming languages

- · shell substitutes and executes
- order:
 - brace expansion
 - tilde expansion
 - parameter and variable expansion
 - command substitution
 arithmetic expansion
 - word splitting

 - filename expansion

Command subing

• replace \$(command) or 'command' by stdout of executing command • can be used to execute content of variables: unix\$ %=1s wyffle.c a.out unix\$ echo %x ls unix\$ echo %x 's wyfile.c a.out unix\$ echo '1s' wh: x: command not found unix\$ echo %(x) wyfile.c a.out unix\$ echo \$(s) wyfile.c a.out

Filename expansion

myfile.c a.out a.b a.out a.b a.out a.b unix\$ ls a? ls:No match. unix\$ ls a? ls:No match. unix\$ ls a.t a.out a.out unix\$ ls a.? a.b a.b	 any word containing "?[[is considered a pattern "matches any sing ? matches any single character [] matches any of the enclosed characters unix\$ 1s
a.b mixi la a* a.out a.b mixi la a? la: No match. mixi la a.t a.out a.b mixi la a.? a.b mixi la a.?? a.out mixi la a.??? a.out mixi la a.l a.j mixi la a.??? a.out mixi la a.l a.b	
unixé la a* a.out a.b unixé la a? la: No match. a.out a.out a.b u.b la a.? u.b la a.?? a.out u.b la a.??? a.out u.b la a.???	
a.out a.b unixé la a? la: No match. unixé la a.* a.out a.b unixé la a.? a.b unixé la a.??? a.out unixé la f.an].b	
 a.b unixi ls a? ls: No match. unixi ls a.* a.b a.b unixi ls a.? a.cr a.out unixi ls a.[m].b 	unix\$ ls a*
unixé la a? la: No match. unixé la a.* a.out a.b unixé la a.? a.b unixé la a.??? a.out unixé la (am).b	a.out
<pre>ls: Yoo match. unix\$ ls a.* a.b unix\$ ls a.7 unix\$ ls a.7? a.out unix\$ ls a.7?? a.out unix\$ ls a</pre>	a.b
umixé la a.* a.out a.b umixé la a.? a.b umixé la a.??? a.out umixé la g.am.b	unix\$ ls a?
a.out a.b unix\$ 1s a.? a.b unix\$ 1s a.??? a.out unix\$ 1s (am).b	ls: No match.
a.b unix\$ 1s a.? a.b unix\$ 1s a.??? a.out unix\$ 1s [am].b	unix\$ ls a.*
unix\$ ls a.? a.b unix\$ ls a.??? a.out unix\$ ls [am].b	a.out
a.b unix\$ 1s a.??? a.out unix\$ 1s [am].b	a.b
unix\$ 1s a.??? a.out unix\$ 1s [am].b	unix\$ ls a.?
a.out unix\$ ls [am].b	a.b
unix\$ ls [am].b	unix\$ ls a.???
	a.out
a.b	unix\$ ls [am].b
	a.b

redirection

- · stdin, stdout and stderr may be redirected
- < redirects stdin (0) to come from a file
- > redirects stdout (1) to go to file
- >> appends stdout to the end of a file
- &> redirects stderr (2)
- >& redirects stdout and stderr, e.g.: 2>&1 sends stderr to the same place that stdout is going
- << gets input from a here document, i.e., the input is what you type, rather than reading from a file

Built in commands

- alias, unalias create or remove a pseudonym or shorthand for a command or series of commands
- jobs, fg, bg, stop, notify control process execution
- command execute a simple command
- cd, chdir, pushd, popd, dirs change working directory echo display a line of text history, fc process command history list
- •
- set, unset, setenv, unsetenv, export shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents
- getopts parse utility options
- hash, rehash, unhash, hashstat evaluate the internal hash table of the contents of directories •
- kill - send a signal to a process

- pwd print name of current/working directory
- shift shell built-in function to traverse either a shell's argument list or a list of field-separated words
- readonly shell built-in function to protect the value of the given variable from reassignment
- source execute a file as a shell script
- suspend shell built-in function to halt the current shell
- test check file types and compare values times shell built-in function to report time usages of the current • shell
- trap, onintr shell built-in functions to respond to (hardware) signals
- type write a description of command type typeset, whence shell built-in functions to set/get attributes and values for shell variables and functions

- limit, ulimit, unlimit set or get limitations on the system resources available to the current shell and its descendents
- umask get or set the file mode creation mask