CS35L Software Construction Laboratory

Lab 5 - Sneha Shankar

Week 2; Lecture 1

Read and Practice

- head
- tail
- sort
- comm
- cmp
- tr: translate or delete characters
  - echo "password a1b2c3" | tr -d [:digit:] -> password abc
  - echo “abc” | tr [:lower:] [:upper:] -> ABC

UNIX Wildcards

- A *wildcard* is a character that can stand for all members of some class of characters
- The * wildcard
  - The character * is a wildcard and matches **zero or more character(s)** in a file (or directory) name. (ls list* or ls *list)
- The ? Wildcard
  - The character ? will match **exactly one character**. (ls ?list OR ls list?)
- The [] Wildcard
  - A pair of [] represents any of the characters enclosed by them (ls *[0-9]*)

Regular Expressions (regex)

- A regex is a special text string for describing a certain search pattern
- Quantification
  - How many times of previous expression?
  - Most common quantifiers: ?(0 or 1), *(0 or more), +(1 or more)
- Alternation
  - Which choices?
  - Operators: [] and |
  - E.g Hello | World , [A B C]
- Anchors
  - Where?
  - Characters: ^(beginning) and $(end)

regex contd...

- ^ start of line
- $ end of line
- \ turn off special meaning of next character
- [] match any of enclosed characters, use – for range
- [^] match any characters except those enclosed in []
- . match a single character of any value
- * match 0 or more occurrences of preceding character/expression
- + match 1 or more occurrences of preceding character/expression

regex contd...

Expression	Matches
tolstoy	The seven letters tolstoy, anywhere on a line
^tolstoy	The seven letters tolstoy, at the beginning of a line
tolstoy$	The seven letters tolstoy, at the end of a line
^tolstoy$	A line containing exactly the seven letters tolstoy, and nothing else
[Tt]olstoy	Either the seven letters Tolstoy, or the seven letters tolstoy, anywhere on a line
tol.toy	The three letters tol, any character, and the three letters toy. Anywhere on a line
tol.*toy	The three letters tol, any sequence of zero or more characters, and the three letters toy. Anywhere on a line

sed

- stream editor, modifies the input as specified by the command(s)
- Can be used for:
- Printing specific lines or address ranges
  - sed –n ‘1p’ sedFile.txt
  - sed –n ‘1,5p’ sedFile.txt
  - sed –n ‘1~2p’ sedFile.txt
- Deleting text
  - sed '1~2d' sedFile.txt
- Substituting text - s/regex/replacement/flags
  - sed 's/cat/dog/' file.txt
  - sed 's/cat/dog/g' file.txt
  - sed 's/<[^>]*>//g' a.html
  - `sed 's/`***`regExpr`***`/`***`replText`***`/'` filename

sed contd..

- sed –n 12,18p file.txt
- sed 12,18d file.txt
- sed ‘1~3d’ file.txt
- sed ‘1,5 s/line/Line/g’ file.txt
- sed ‘/pattern/d’ file.txt
- sed ‘/regexp/!d’ file.txt

grep

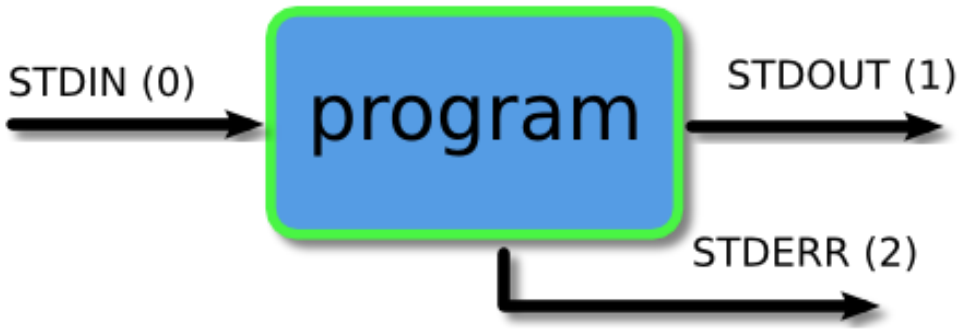
- A Unix command to search files/text for the occurrence of a string of characters that matches a specified pattern
- Usage:
  - grep [option(s)] pattern [file(s)]
- grep –r '*.txt' *
- grep –c 'line' file.txt
- grep –n 'line' file.txt
- ls –l | grep *txt

awk

- awk is more than a command; it's a programming language by itself
- Utility/language for data extraction
- awk views a text file as records and fields
- Usage:
  - awk '/search pattern/ {Actions}' file
- Examples:
  - awk '{print;}' file.txt // print the file line by line; default behaviour
  - awk '/Hello/ {print;}' file.txt // prints lines which matches Hello
  - awk '{print $1,$2;}' file.txt // prints only specific fields
  - awk –F'Hello' '{print $2}' // prints second column in between the occurrences of the specified pattern

Piping and Redirection

- Every program we run on the command line automatically has three data streams connected to it.
  - STDIN (0) - Standard input (data fed into the program)
  - STDOUT (1) - Standard output (data printed by the program, defaults to the terminal)
  - STDERR (2) - Standard error (for error messages, also defaults to the terminal)



Piping and redirection is the means by which we may connect these streams between programs and files to direct data in useful ways.

More About Piping and Redirection

- Basic I/O Redirection
  - Most programs read from stdin
  - Write to stdout
  - Send error messages to stderr
  - Try: $ cat

    // With no arguments, read standard input, write standard output
  - Task: Piping and Redirection
  - Create a file test.txt with numbers 1-5 in descending order in each line
    - Delete all the new line characters (with tr command and redirection) and redirect output to test1.txt
    - Now, first sort the file and then repeat the above step; but instead of redirection, now append the output to test1.txt

The Shell and OS

- What is a shell?
  - Outermost cover of the kernel
  - User's interface to the OS
- Use it to run your programs

Compiled Languages v/s Scripting Languages

- Compiled Languages
  - Examples?
    - C,C++,Java
  - First Compiled
  - Source code to object code; then executed
  - Run faster
  - Applications:
    - Typically run inside a parent program like scripts, more compatible during integration, can be compiled and used on any platform (eg. Java)

- Scripting Languages
  - Examples?
    - Python, JavaScript, Shell Scripting
  - No compilation required. Directly interpreted!
  - Interpreter reads program, translates into internal form and executes
  - Runs slower than a high level lang
  - Applications:
    - Automation, Extracting information from a data set, Less code intensive

Shell Script

- A computer program designed to be run on a shell (UNIX/Linux)
- All shell commands can be executed inside a script
- Why use a shell script?
  - Simplicity
  - Portability
  - Ease of development

Basic Shell Constructs

- Shell recognizes three fundamental kinds of commands:
  - **Built-in commands**: Commands that the shell itself executes (e.g.: echo)
  - **Shell functions**: Self-contained chunks of code, written in shell language
  - **External Commands**: mainly external utilities; backtick often associated
    - number=`ll | wc -l` // This is an external command
    - echo $number

Self-Contained Scripts: The #! First Line

- When the shell runs a program, it asks the kernel to start a new process and run the given program in that process.
- It knows how to do this for compiled programs but for a script, the kernel will fail, returning a "not executable format file" error so it'll start a new copy of /bin/sh (the standard shell) to run the program.
- But if there is more than one shell installed on the system, we need a way to tell the kernel which shell to use for a script

  #! /bin/csh –f

  #! /bin/awk –f

  #! /bin/sh

Task: Understanding a Shell Script

- Create a file testFile.sh
- Write a statement to print “Hello World” inside it
- Run the File with ‘sh testFile.sh’. What do you observe?
- Now, add #!/bin/sh to the first line and repeat the above step
- Did you see any difference?
- Why?
- Now, rename the file to testFile.txt and run it again.
- Did you see any difference?
- Why?

Variables in shell script

- Start with a letter or underscore and may contain any number of following letters, digits, or underscores
- Hold string variables
- $ myvar=this_is_a_long_string_that_does_not_mean_much //Assign value
- $ echo $myvar //Print the value

this_is_a_long_string_that_does_not_mean_much

- first=sneha middle=s last=shankar — Multiple assignments allowed on one line
- fullname="$first $middle $last" — Double quotes required here, for concatenating
- fullname="abc xyz mno" — Use quotes for whitespace in value
- oldname=$fullname — Quotes not needed to preserve spaces in value

Variables in shell script contd…

- **Escape Character ** - Literal value of following character
  - echo \|
- **Single Quote** - Literal Meaning of all within ‘’
  - $hello=1
  - $str=‘$hello’
  - echo $str -> $hello
- **Double Quote** - Literal meaning except for $, ` and \.
  - $hello=1
  - $str=“abc$hello”
  - echo $str -> abc1

Variables in shell script contd…

- Special Variables: certain characters reserved as special variables
  - $: PID of current shell
  - #: number of arguments the script was invoked with
  - n: nth argument to the script
  - ?: exit status of the last command executed
  - echo $$; echo $#; echo $2; echo $?;
- scalar variable vs array variable:
  - array_name[index]=value; echo ${array_name[index]}

Loops

- **for var in list_values**
  do
    command 1
    ..
    command n
  done
- **while condition**
  do
    command 1
    ..
    command n
  done

```
ALL=`ls -a $dir | sort`
declare -a ARRAY
count=0
for FILE in $ALL
        do
        ARRAY[$count]=$FILE
        .....
        done
```

```
for i in "${ARRAY[@]}"
        do
        …
        done
```

Conditional and Unconditional Statements

- **Conditional**
  - if…then…fi
  - if…then…else…fi
  - if…then…elif..then…fi
  - case…esac
- **Unconditional**
  - break
  - continue

```
#!/bin/sh

a=10
b=20

if [ $a == $b ]
then
   echo "a is equal to b"
elif [ $a -gt $b ]
then
   echo "a is greater than b"
elif [ $a -lt $b ]
then
   echo "a is less than b"
else
   echo "None of the condition met"
fi
```

```
#!/bin/sh

FRUIT="kiwi"

case "$FRUIT" in
   "apple") echo "Apple pie is quite tasty."
   ;;
   "banana") echo "I like banana nut bread."
   ;;
   "kiwi") echo "New Zealand is famous for kiwi."
   ;;
esac
```