UCLA CS35L

Week 3

Monday

Reminders

- Start thinking about Week 10 Presentation Topics/Teams
  - Signup link under CCLE – Week 10
- Assignment 2 due tonight (4/13)
- Assignment 3 is longer and due next **Friday** (4/24)

- Anonymous feedback for Daniel - https://forms.gle/tZwuMbALe825DBVn8

More Linux Commands

I/O Redirection in the Shell

Most programs read from stdin (input to terminal)

Then write to stdout (output to terminal)

Send error messages to stderr

echo hello -> writes to stdout

cat non-existent-file -> writes to stderr

cat -> waits for stdin, and then writes to stdout

I/O Redirection – Pipeline Operator |

Lets you **PIPE** output from one command as input to a second command.

```
$ who
george              pts/2  Dec 31 16:39 (valley-forge.example.com)
betsy         pts/3  Dec 27 11:07 (flags-r-us.example.com)
benjamin      dtlocal       Dec 27 17:55 (kites.example.com)
jhancock      pts/5  Dec 27 17:55 (:32)
Camus         pts/6  Dec 31 16:22
tolstoy             pts/14       Jan 2 06:42
```

```
$ who | wc -l          Count users
6
```

I/O Redirection – Pipeline Operator |

Note – you can chain multiple times!

```
$ who
george                  pts/2  Dec 31 16:39 (valley-forge.example.com)
betsy           pts/3  Dec 27 11:07 (flags-r-us.example.com)
benjamin        dtlocal         Dec 27 17:55 (kites.example.com)
jhancock        pts/5  Dec 27 17:55 (:32)
Camus           pts/6  Dec 31 16:22
tolstoy                 pts/14          Jan 2 06:42

$ who | grep “Dec 27”
betsy           pts/3  Dec 27 11:07 (flags-r-us.example.com)
benjamin        dtlocal         Dec 27 17:55 (kites.example.com)
jhancock        pts/5  Dec 27 17:55 (:32)

$ who | grep “Dec 27” | wc -l                Count users
3
```

I/O Redirection – <, >, and >>

< takes the file instead of the terminal as stdin

```
tr a b < some-file.txt
```

> Writes to the file after erasing the existing content (overwrites)

```
echo ‘this will be the new text in the file’ > myFile
```

>> appends to the file

```
echo ‘new line at end of file’ >> myFile
```

One more file redirection 2>

2> Redirects stderr to a file

```
cat non-existent-file 2> error.log
```

2>&1 Redirects stderr to stdout

```
cat non-existent-file 2>&1
```

echo

- Echo prints output of a command to stdout

```
$ echo Now is the time for all good men
Now is the time for all good men
$ echo to come to the aid of their country.
to come to the aid of their country.
```

There is also fancier output with printf, which can refer to its man page for

tr

- Translate or delete characters
- tr SET1 SET2
    - Translate characters in SET1 to SET2
    - e.g. echo "hello world" | tr "o" "a“
- tr -d SET
    - Deletes any characters in SET
    - e.g. echo "hello world" | tr -d "lo“
- tr -s SET
    - "Squeezes” consecutive, repeated letters into single occurrence
    - e.g. echo "hello world" | tr -s "l"

sort

Check the man page

```
echo -e “abc\nabc\ndef\n” > hi
sort hi
sort –u hi
```

Locale and commands

- A set of environmental variables that define character encodings for your shell session
- Locale
  - Prints information about the current locale environment standard output
  - Gets its data from the LC_* environment variables
- LC_TIME
  - Date and Time formats
- LC_COLLATE
  - Order for comparing and sorting

Locale and commands cont.

- LC_ALL
  - Determines values for ALL locale categories
- LC_COLLATE=‘C’
  - Sorting is in ASCII order. Note that ASCII is both a character set and an encoding. Other encodings include UTF-8, etc
- LC_COLLATE=‘en_US’
  - Sorting is case insensitive except when the two strings are otherwise equal and one has an uppercase earlier than the other.

Locale and assignment

- Keep in mind for assignment, need to run
  - `export LC_ALL=‘C’`
  - Will make sure sorting is done byte-wise so your results will match

Regular Expressions (Regex)

What is Regex

- A text string with special notation to describe a search pattern
- Incredibly powerful, but there are a lot of rules so it gets complicated

Anchors

- ^ - match an expression at the beginning of a line/string
- $ - match an expression at the end of a line/string
- Examples

`tolstoy`	tolstoy anywhere on line
`^tolstoy`	tolstoy at beginning of line
`tolstoy$`	tolstoy at the end of a line
`^tolstoy$`	A line containing only tolstoy and nothing else

Character Sets

- […] goes around a range of characters that you want to match
- Examples

`[ABC]`	Match any character in the set “ABC”
`[A-Za-z]`	Match any lowercase or uppercase alphabet character
`[^0-9]`	Match a string that does NOT contain any digits NOTE - ^ inside a capture group is a NOT
`\w`	Match alphanumeric and underscore [A-Za-z0-9_]
`\d`	Match any digit [0-9]
`\s`	Match any whitespace (spaces, tabs, line breaks)

Quantifiers

- Used to match a specific **number** of occurences
- Examples

`.`	Match any single character
`*`	Match 0 or more preceding character
`?`	Match 0 or 1 of preceding character
`+`	Match 1 or more of preceding character
`{n}`	Match exactly n occurrence of preceding character
`{n,}`	Match n or more occurrences of preceding character
`{n,m}`	Match n to m occurrences of preceding character

Examples

- Match string that has “abc”
  - Examples to match = abc, habc, abcdefg
  - Answer – `abc`
- Match string that has “ab” followed by one or more “c”
  - Examples to match = abc, abcc, abccc
  - Answer - `abc+`
- Match string that starts with “w”
  - Examples to match = world, wish, whale
  - Answer = `^w`

Examples cont

- Match string that has one alphabet character and then the number 3
  - Examples to match – D3, someString3, chars34789
  - Answer – `[A-Za-z]3`
- Match a string that contains only letters, dots, hyphens, and underscores
  - Examples – myFile, UC.LA, nine_of-spades
  - Answer – `^[-._A-Za-z]*$`
- Match string that has 3 or more occurrences of o
  - Examples – Goooooooogle
  - Answer - `{3,}o`

Capturing Groups

- Use parentheses to create a capture group
  - (abc){3}
- Backreference a capture group to reuse it
  - Use \# where # is based on order of appearance (starts at 1)
  - Match HTML tag
  - <([A-Z][A-Z0-9]*)\b[^>]*>.*?</\1>

Alternation

- Use | to represent OR in regex
- Examples
  - yes | no
  - me | myself | I

POSIX Match Groups

- Built-in Matching Groups

`[:alpha:]`	Alphabetic
`[:digit:]`	Numeric
`[:alnum:]`	Alphanumeric
`[:blank:]/[:space:]`	Space and tab / all whitespace
`[:graph:]`	Non-space
`[:lower:]/[:upper:]`	Lowercase / uppercase

Regex Helpful Resources

- When in doubt – use websites like https://regexr.com/
  - You can put in sample strings and test your regex patterns to see in real-time what you are matching against
- Regex cheat sheets are handy in case you forget
  - https://www.rexegg.com/regex-quickstart.html

Commands with Regex

Basic vs Extended Regex

- **Basic Regular Expressions (BRE):** ? + { }) _ | are normal characters
  - Need to escape these to use as metacharacters
- **Extended Regular Expressions (ERE)** will treat them with special meaning
  - Need to escape if you want use a metacharacter as a normal character
- The commands we’ll talk about use BRE by default
- NOTE – to escape a character means to use a backslash \ in front
  - * \? etc

grep

- Searches input for line matching the search term
- By default uses BRE, but can change behavior
- grep –E
  - uses ERE
- grep –F
  - matches fixed strings instead of regex

grep examples

- Commonly used when piping in input from another source
- echo “my text” | grep my
- echo “123a456” | grep [A-Za-z]
- echo “Gooogle” | grep –E o{3}
- ls –l | grep “.txt”
- cat error.log | grep “^Connection Failed”

find

- Talked about in week 1
- Extra options to customize that may be helpful this week
  - -name – search by name
  - -regex –specify a regex pattern for search
  - -type – search by file type (e.g. directory)
  - -maxdepth DEPTH – descend into directories to a given depth
  - -prune – ignore a directory and the files under it
  - -exec – execute command using each matched file as an argument

find examples

- find ~/cs35l –maxdepth 1 –name “answer.txt”
- find . –type d –exec echo {} >> matches.txt \;
  - {} represents found file
  - \; completes command for -exec

sed

- Utility to replace text using regex pattern matching
- Structure
  - `sed “s/ORIGINAL/REPLACEMENT/[FLAGS]” [FILENAME]`
- Note:
  - `sed –E` uses ERE
  - `g` flag specifies Global Search, otherwise default is to only replace the first match on each line

sed examples

- `sed ‘s/this/that’ file`
  - Replace **first** occurrence of “this” with “that” for every line in the file
- `sed ‘s/this/that/g’ file`
  - Replace **every** occurrence of “this” with “that” in file
- `sed ‘s/hi//g’ file`
  - Remove every occurrence of hi from file
- `echo “hi and goodbye” | sed –E ‘s/hi|bye//g’`
  - Remove every occurrence of “hi” and “bye” from echo’d input

Double vs Single Quotes

- Double and Single Quotes can both be used to enclose strings and search phrases
- If you need to match against ‘ then use “ to enclose your string phrase
- May still need to escape special characters like
  - Single-quote ‘
  - Back-tick `