Shell Scripting

Week 2

Administration

- My office hour:

  Thursdays 09:30–11:30 @BH 2432

- Assignment 2 is due Saturday 4/15 11:55pm on CCLE
  - Probably the most difficult assignment
  - Start early
- http://web.cs.ucla.edu/classes/spring17/cs35L/assign.html
- Lab part: buildwords, lab2.log

  Homework part: sameln

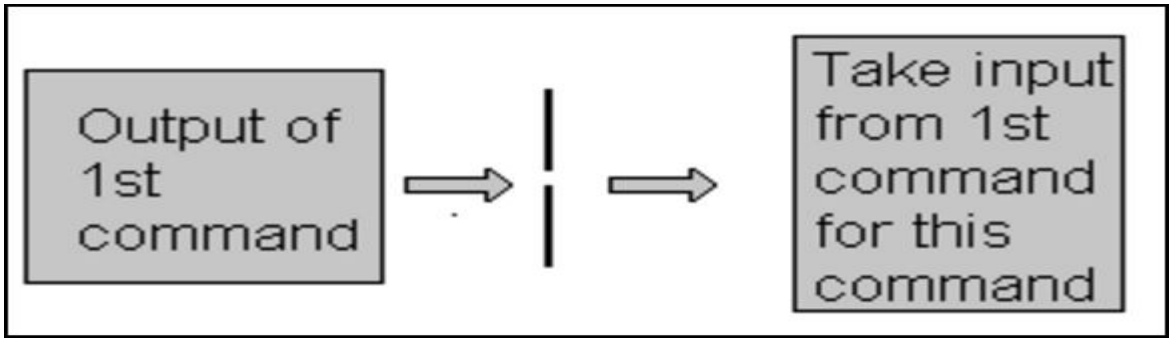
The Basics: Redirection

- > *file*: (over)write stdout to a file
- >> *file*: append stdout to a file
- < *file*: use contents of a file as stdin

Pipe

Command 1 | command 2

- **connect the output of command1 to the input of command 2 without any temporary file.**



$ **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

Text Processing Tools

- sort: sorts text alphabetically
- wc: count the number of lines, words, and bytes
- head: extract top of files (default 10 lines)
- tail: extracts bottom of files (default 10 lines)

ex: head -n 20 lab2.txt

Q: how to get line 10 to line 20 of a file?

The tr command

- Abbreviated as translate or transliterate
- Usage

  tr [options] [set1] [set2]

- Function: replace the elements in set1 with corresponding elements from set2

  Example: tr [a-z] [A-Z]

-- replace all lowercase letters from input with uppercase letters

-- input can be stdin, file, pipe

-- tr [:lower:] [:upper:]

Options in tr command

tr [options] [set1] [set2]

- -C , -- complement

  - use the complement of set 1

- -d, -- delete

  - delete characters in SET1, do not translate

- -s, -- squeeze-repeats

  - replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character

Question: tr -c 'A-Za-z' '[\n*]’

Examples of tr command

- Usage: as a part of pipeline
  - e.g. cat assign2.html | tr -cs 'A-Za-z' '[\n*]' > pre
- Eliminate everything except alphabet characters, also duplicate words
  - tr -cs 'A-Za-z' '[\n*]'
- Transform all upper cases characters to lower cases
  - tr '[:upper:]' '[:lower:]'
- Delete all left-over blanks
  - tr -d '[:blank:]'

sed command

sed [options] … {script-only-if-no-other-script} [input-file] ...

- Stream editor for filtering and transforming text
- Modify input as specified by the commands
- Use sed to search patterns and replace
  - sed ‘s/regexp/replacement/”
  - Ex: sed 's/h.llo/world/'

```
sed 's/\*/\+/g'
sed 's/ /\n/g'
    sed 's/:.*//'
```

Note: g means global, operate on the whole line

sed command

sed [options] … {script-only-if-no-other-script} [input-file] ...

- Use sed to delete lines with certain number or pattern

- sed '3d'

  (delete the third line)

- sed '3,$d'

  (delete from the third line to the end)

- sed '/pattern/d'

  (delete lines with certain pattern)

- sed '/pattern1/ , /pattern2/d'

  (delete a range of lines, pattern1 turns on the deletion, pattern 2 turns off the deletion)

Compare difference between files

- diff
  - usage: diff [option] [file1] [file2]
  - function: compare files line by line
- comm
  - usage: comm [option] [file1] [file2]
  - function: compare sorted files line by line

Laboratory -- Spell-checking Hawaiian

- Finish the script *buildword*:
  - Basic Structure: Using a pipeline of tr and sed commands
  - Input: read html from stdin
  - Output: a sorted list of unique words
  - Usage: cat foo.html bar.html | ./buildwords
  - Preprocess
    - Delete whatever before/after the html <table> tag
    - Eliminate html tags, extract words

Laboratory -- Spell-checking Hawaiian

- First clean the web pages:
  - Eliminate characters expect a-z and A-Z
  - Eliminate characters out of Hawaiian alphabet
  - Transform upper cases to lower cases
- Use commands: tr and sed
- Try basic regular expressions

Laboratory -- Spell-checking Hawaiian

- Finish the script *buildword* (continue):
  - Change upper case characters to lower case
  - Treat ` as ’
  - Remove any misspelled Haiwaiin language
  - Hints: don’t leave unnecessary information behind (e.g. duplication, empty lines, spaces, html tags)

grep command

grep [options] pattern [file...]

- Global regular expression print
- Process text line by line and print lines matching a specified pattern
- Options
- f: obtain patterns from file, one per line
- i: ignore cases in both patterns and input files
- Similar commands
- grep uses basic regular expression (BRE)
- egrep (-E) uses extend regular expression (ERE)
- fgrep (-F) uses fixed strings instead of regular expressions

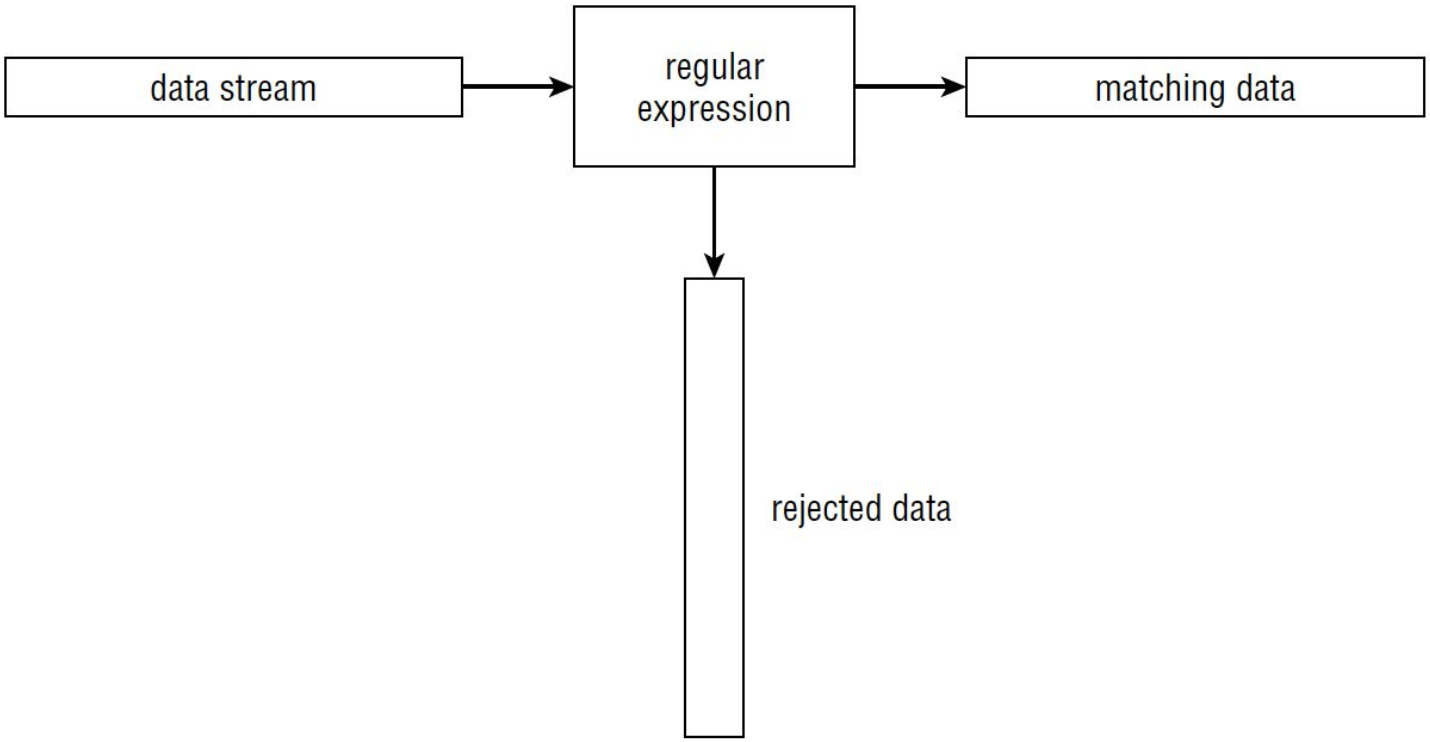
Simple grep

```
$ who                          Who is logged on
tolstoy tty1 Feb 26 10:53
tolstoy pts/0 Feb 29 10:59
tolstoy pts/1 Feb 29 10:59
tolstoy pts/2 Feb 29 11:00
tolstoy pts/3 Feb 29 11:00
tolstoy pts/4 Feb 29 11:00
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)

$ who | grep -F austen          Where is austen logged on?
austen pts/5 Feb 29 15:39 (mansfield-park.example.com)
austen pts/6 Feb 29 15:39 (mansfield-park.example.com)
```

Regular Expressions

- Notation that represents a text pattern
  - ex: starts with the letter a
- Match/filter data against a regular expression



Regular Expressions

- Different applications use different types of regular expression
  - Programming languages (i.e. Java, Perl, Python)
  - Linux utilities (i.e. sed,grep)
  - Mainstream applications (i.e. MySQL)
- Regular Expression Engine
  - Interprets regular patterns and use patterns to match text
  - The POSIX Basic Regular Expression (BRE) engine
  - The POSIX Extended Regular Expression (ERE) engine

Special characters in Regular expression

- Quantification (the number of previous occurrences)
  - ? (0 or 1)
  - * (0 or more)
  - + (1 or more)
  - {} (specified number)
- Alternation
  - [] (any character in the range)
  - | (one case or another)
- Anchors
  - ^ (beginning of a line)
  - $ (end of a line)
- Group
  - ()

Regular expressions

Character	BRE / ERE	Meaning in a pattern
\	Both	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	Both	Match any single character except NUL. Individual programs may also disallow matching newline.
*	Both	Match any number (or none) of the single character that immediately precedes it. For EREs, the preceding character can instead be a regular expression. For example, since . (dot) means any character, .* means "match any number of any character." For BREs, * is not special if it's the first character of a regular expression.
^	Both	Match the following regular expression at the beginning of the line or string. BRE: special only at the beginning of a regular expression. ERE: special everywhere.

Regular Expressions (cont'd)

$	Both	Match the preceding regular expression at the end of the line or string. BRE: special only at the end of a regular expression. ERE: special everywhere.
[...]	Both	Termed a bracket expression, this matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. (Caution: ranges are locale-sensitive, and thus not portable.) A circumflex (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. A hyphen or close bracket (]) as the first character is treated as a member of the list. All other metacharacters are treated as members of the list (i.e., literally). Bracket expressions may contain collating symbols, equivalence classes, and character classes (described shortly).
\{*n*,*m*\}	BRE	Termed an *interval expression*, this matches a range of occurrences of the single character that immediately precedes it. \{*n*\} matches exactly n occurrences, \{*n*,\} matches at least n occurrences, and \{*n*,*m*\} matches any number of occurrences between n and m. n and m must be between 0 and RE_DUP_MAX (minimum value: 255), inclusive.
\(\)	BRE	Save the pattern enclosed between \(and \) in a special *holding space*. Up to nine subpatterns can be saved on a single pattern. The text matched by the subpatterns can be reused later in the same pattern, by the escape sequences \1 to \9. For example, **\(ab\).*\1** matches two occurrences of ab, with any number of characters in between.

Regular Expressions (cont'd)

\n	BRE	Replay the nth subpattern enclosed in \(and \) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
{n,m}	ERE	Just like the BRE \{n,m\} earlier, but without the backslashes in front of the braces.
+	ERE	Match one or more instances of the preceding regular expression.
?	ERE	Match zero or one instances of the preceding regular expression.
|	ERE	Match the regular expression specified before or after.
( )	ERE	Apply a match to the enclosed group of regular expressions.

Examples

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 (e.g., toltoy, tolstoy, tolWHOtoy, and so on)

BRE Special Character Classes

Class	Description
`[[:alpha:]]`	Match any alphabetical character, either upper or lower case.
`[[:alnum:]]`	Match any alphanumeric character 0–9, A–Z, or a–z.
`[[:blank:]]`	Match a space or Tab character.
`[[:digit:]]`	Match a numerical digit from 0 through 9.
`[[:lower:]]`	Match any lower-case alphabetical character a–z.
`[[:print:]]`	Match any printable character.
`[[:punct:]]`	Match a punctuation character.
`[[:space:]]`	Match any whitespace character: space, Tab, NL, FF, VT, CR.
`[[:upper:]]`	Match any upper-case alphabetical character A–Z.