UNIX II:grep, awk, sed

October 30, 2017

File searching and manipulation

- In many cases, you might have a file in which you need to find specific entries (want to find each case of NaN in your datafile for example)

- Or you want to reformat a long datafile (change order of columns, or just use certain columns)

- Can be done with writing python or other scripts, today will use other UNIX tools

grep: global regular expression print

- Use to search for a pattern and print them
- Amazingly useful! (a lot like Google)

```
GREP(1)                   BSD General Commands Manual                  GREP(1)

NAME
     grep, egrep, fgrep, zgrep, zegrep, zfgrep -- file pattern searcher

SYNOPSIS
     grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwxZ] [-A num] [-B num] [-C[num]]
          [-e pattern] [-f file] [--binary-files=value] [--color[=when]]
          [--colour[=when]] [--context[=num]] [--label] [--line-buffered]
          [--null] [pattern] [file ...]

DESCRIPTION
     The grep utility searches any given input files, selecting lines that
     match one or more patterns.  By default, a pattern matches an input line
     if the regular expression (RE) in the pattern matches the input line
     without its trailing newline.  An empty expression matches every line.
     Each input line that matches at least one of the patterns is written to
     the standard output.

     grep is used for simple patterns and basic regular expressions (BREs);
     egrep can handle extended regular expressions (EREs).  See re_format(7)
     for more information on regular expressions.  fgrep is quicker than both
     grep and egrep, but can only handle fixed patterns (i.e. it does not
     interpret regular expressions).  Patterns may consist of one or more
     lines, allowing any of the pattern lines to match a portion of the input.

     zgrep, zegrep, and zfgrep act like grep, egrep, and fgrep, respectively,
     but accept input files compressed with the compress(1) or gzip(1) com-
     pression utilities.
```

grep

Basic syntax: >>> grep <pattern> <inputfile>

>>> grep Oklahoma one_week_eq.txt

2017-10-28T09:32:45.970Z,35.3476,-98.0622,5,2.7,mb_lg,,133,0.329,0.3,us,us1000ay0b,2017-10-28T09:47:05.040Z,"11km WNW of Minco, Oklahoma",earthquake,1.7,1.9,0.056,83,reviewed,us,us

2017-10-28T04:08:45.890Z,36.2119,-97.2878,5,2.5,mb_lg,,41,0.064,0.32,us,us1000axz3,2017-10-28T04:22:21.040Z,"8km S of Perry, Oklahoma",earthquake,1.4,2,0.104,24,reviewed,us,us

2017-10-27T18:39:28.100Z,36.4921,-98.7233,6.404,2.7,ml,,50,,0.41,us,us1000axpz,2017-10-28T02:02:23.625Z,"33km NW of Fairview, Oklahoma",earthquake,1.3,2.6,,,reviewed,tul,tul

2017-10-27T10:00:07.430Z,36.2851,-97.506,5,2.8,mb_lg,,25,0.216,0.19,us,us1000axgi,2017-10-27T19:39:37.296Z,"19km W of Perry, Oklahoma",earthquake,0.7,1.8,0.071,52,reviewed,us,us

2017-10-25T15:17:48.200Z,36.2824,-97.504,7.408,3.1,ml,,25,,0.23,us,us1000awq6,2017-10-25T21:38:59.678Z,"19km W of Perry, Oklahoma",earthquake,1.1,5,,,reviewed,tul,tul

2017-10-25T11:05:21.940Z,35.4134,-97.0133,5,2.5,mb_lg,,157,0.152,0.31,us,us1000awms,2017-10-27T21:37:47.660Z,"7km ESE of McLoud, Oklahoma",earthquake,1.7,2,0.117,19,reviewed,us,us

2017-10-25T01:50:53.100Z,36.9748,-99.4244,8.115,2.9,ml,,197,,0.64,us,us1000awir,2017-10-26T00:52:01.343Z,"23km NE of Buffalo, Oklahoma",earthquake,2,7.6,,,reviewed,tul,tul

2017-10-24T23:18:09.000Z,35.3787,-98.0931,7.72,2.7,ml,,91,,0.49,us,us1000awhe,2017-10-26T00:47:37.010Z,"13km W of Union City, Oklahoma",earthquake,2.4,5.7,,,reviewed,tul,tul

2017-10-23T15:57:10.890Z,36.6565,-97.8019,5,2.6,mb_lg,,39,0.2,0.15,us,us1000avxp,2017-10-23T18:30:47.642Z,"17km SSW of Medford, Oklahoma",earthquake,1.2,1.8,0.132,15,reviewed,us,us

grep

- Lots of useful options available **(read the man page!)**
- -w : look for a whole word
- -i : ignore case
- -v : omit matching lines
- -c: provide a count of matching lines

grep

What is a regular expression?

```
GREP(1)                    BSD General Commands Manual                    GREP(1)

NAME
     grep, egrep, fgrep, zgrep, zegrep, zfgrep -- file pattern searcher

SYNOPSIS
     grep [-abcdDEFGHhIiJLlmnOopqRSsUVvwxZ] [-A num] [-B num] [-C[num]]
          [-e pattern] [-f file] [--binary-files=value] [--color[=when]]
          [--colour[=when]] [--context[=num]] [--label] [--line-buffered]
          [--null] [pattern] [file ...]

DESCRIPTION
     The grep utility searches any given input files, selecting lines that
     match one or more patterns.  By default, a pattern matches an input line
     if the regular expression (RE) in the pattern matches the input line
     without its trailing newline.  An empty expression matches every line.
     Each input line that matches at least one of the patterns is written to
     the standard output.

     grep is used for simple patterns and basic regular expressions (BREs);
     egrep can handle extended regular expressions (EREs).  See re_format(7)
     for more information on regular expressions.  fgrep is quicker than both
     grep and egrep, but can only handle fixed patterns (i.e. it does not
     interpret regular expressions).  Patterns may consist of one or more
     lines, allowing any of the pattern lines to match a portion of the input.

     zgrep, zegrep, and zfgrep act like grep, egrep, and fgrep, respectively,
     but accept input files compressed with the compress(1) or gzip(1) com-
     pression utilities.
```

Regular Expression

- Set of characters that specify a pattern
- Makes changing and searching for text easy just from the command line.
- Regular expressions are accepted input for grep, sed, awk, perl, vim and other unix commands.
- It’s all about syntax…. (and because it’s UNIX, it’s a little cryptic)
- http://www.regular-expressions.info/quickstart.html

Simple Regular Expression Symbols

Generally a good idea to surround regular expression with single quotes on command line to protect it from being interpreted by the shell.

- . (period) --- matches any single character
- B --- matches uppercase B
- b --- matches lowercase b
- * --- matches zero or more occurrences of preceding character
- ^ --- goes to beginning of a line
  - Example – search a file where # is used to comment lines

  >>> grep ^# filename

  Will pull out all the lines where # is the first character in line
- $ --- end of the line

Simple Regular Expression Symbols

- \ --- looking for a symbol

```
[Macintosh-5:~/Desktop/lab10] sbilek% grep \* one_week_eq.txt
*time,latitude,longitude,depth,mag,magType,nst,gap,dmin,rms,net,id,updated,place,type,horizontalError,depthError,magError,magNst,status,locationSource,magSource
```

- [] --- matches member of the range within the brackets

```
[Macintosh-5:~/Desktop/lab10] sbilek% grep 'm[bl]' one_week_eq.txt | head
2017-10-28T09:32:45.970Z,35.3476,-98.0622,5,2.7,mb_lg,,133,0.329,0.3,us,us1000ay0b,2017-10-28T09:47:05.040Z,"11km WNW of Minco, Oklahoma",earthquake,1.7,1.9,0.056,83,reviewed,us,
us
2017-10-28T08:00:33.180Z,47.6909,147.5345,378.91,4.1,mb,,141,3.328,0.81,us,us1000axzx,2017-10-28T08:59:27.040Z,"169km SW of Vostok, Russia",earthquake,4.5,10.1,0.054,93,reviewed,
us,us
2017-10-28T06:27:50.180Z,13.3098,50.8856,10,4.6,mb,,120,12.616,0.98,us,us1000axzm,2017-10-28T06:41:27.040Z,"160km N of Bereeda, Somalia",earthquake,11.7,1.9,0.095,33,reviewed,us,
us
2017-10-28T05:54:27.290Z,19.0921669,-155.4701691,43.3,2.68,ml,40,181,0.06337,0.23,hv,hv61958436,2017-10-28T06:00:15.610Z,"12km S of Pahala, Hawaii",earthquake,0.9,1.3,0.85,3,auto
matic,hv,hv
2017-10-28T05:36:53.360Z,33.4991667,-116.8006667,4.85,2.56,ml,92,30,0.01317,0.22,ci,ci38032328,2017-10-28T06:43:42.844Z,"9km NE of Aguanga, CA",earthquake,0.17,0.76,0.107,25,auto
matic,ci,ci
2017-10-28T04:08:45.890Z,36.2119,-97.2878,5,2.5,mb_lg,,41,0.064,0.32,us,us1000axz3,2017-10-28T04:22:21.040Z,"8km S of Perry, Oklahoma",earthquake,1.4,2,0.104,24,reviewed,us,us
2017-10-28T01:52:41.340Z,-22.7612,-176.684,141.36,4.6,mb,,85,6.553,0.88,us,us1000axy9,2017-10-28T02:11:44.040Z,"231km SW of Vaini, Tonga",earthquake,11.5,7.4,0.095,36,reviewed,us
,us
2017-10-27T22:43:37.596Z,59.5819,-153.6007,115.1,3.4,ml,,,,0.51,ak,ak17105789,2017-10-28T03:01:26.040Z,"76km ESE of Old Iliamna, Alaska",earthquake,,0.4,,,reviewed,ak,ak
2017-10-27T21:32:06.380Z,39.4143,54.9324,28.96,4.4,mb,,135,2.9,1.1,us,us1000axvh,2017-10-27T22:08:48.040Z,"37km NE of Gumdag, Turkmenistan",earthquake,9,7.4,0.134,16,reviewed,us,
us
2017-10-27T20:38:53.200Z,-30.5362,-71.9326,21.74,4.2,mb,,118,0.29,1.01,us,us1000axtw,2017-10-27T22:11:27.040Z,"70km W of Ovalle, Chile",earthquake,6.2,3.6,0.186,8,reviewed,us,us
```

- [^] --- matches anything except what's in the bracket

- Non-printable characters:
  - \t : for a tab character
  - \r : for carriage return
  - \n : for new line
  - \s : for a white space

Sed – stream editor

- Command line tool for editing files line by line, largely used for substitution
- Like grep for searching, but can replace found pattern with something else
- Want to change every instance of mb to ml in my file?

>>> sed s/mb/ml filename

```
[Macintosh-5:~/Desktop/lab10] sbilek% more short_mb_mb.txt
2017-10-28T09:32:45.970Z,35.3476,-98.0622,5,7.7,mb_lg,,133,0.329,0.3,us,us1000ay0b,2017-10-28T09:47:05.040Z,"11km WNW of Minco, Oklahoma",e
2017-10-28T08:00:33.180Z,47.6909,147.5345,378.91,4.1,mb,,141,3.328,0.81,us,us1000axzx,2017-10-28T08:59:27.040Z,"169km SW of Vostok, Russia"
2017-10-28T06:27:50.180Z,13.3098,50.8856,10,4.6,mb,,120,12.616,0.98,us,us1000axzm,2017-10-28T06:41:27.040Z,"160km N of Bereeda, Somalia",ea
2017-10-28T05:54:27.290Z,19.0921669,-155.4701691,43.3,2.68,ml,40,181,0.06337,0.23,hv,hv61958436,2017-10-28T06:00:15.610Z,"12km S of Pahala,
2017-10-28T05:36:53.360Z,33.4991667,-116.8006667,4.85,2.56,ml,92,30,0.01317,0.22,ci,ci38032328,2017-10-28T06:43:42.844Z,"9km NE of Aguanga,
2017-10-28T04:08:45.890Z,36.2119,-97.2878,5,2.5,mb_lg,,41,0.064,0.32,us,us1000axz3,2017-10-28T04:22:21.040Z,"8km S of Perry, Oklahoma",eart
2017-10-28T01:52:41.340Z,-22.7612,-176.684,141.36,4.6,mb,,85,6.553,0.88,us,us1000axy9,2017-10-28T02:11:44.040Z,"231km SW of Vaini, Tonga",e
2017-10-27T22:43:37.596Z,59.5819,-153.6007,115.1,3.4,ml,,,,0.51,ak,ak17105789,2017-10-28T03:01:26.040Z,"76km ESE of Old Iliamna, Alaska",ea
2017-10-27T21:32:06.380Z,39.4143,54.9324,28.96,4.4,mb,,135,2.9,1.1,us,us1000axvh,2017-10-27T22:08:48.040Z,"37km NE of Gumdag, Turkmenistan"
2017-10-27T20:38:53.200Z,-30.5362,-71.9326,21.74,4.2,mb,,118,0.29,1.01,us,us1000axtw,2017-10-27T22:11:27.040Z,"70km W of Ovalle, Chile",ear
[Macintosh-5:~/Desktop/lab10] sbilek% sed s/mb/ml/ short_mb_mb.txt
2017-10-28T09:32:45.970Z,35.3476,-98.0622,5,7.7,ml_lg,,133,0.329,0.3,us,us1000ay0b,2017-10-28T09:47:05.040Z,"11km WNW of Minco, Oklahoma",e
2017-10-28T08:00:33.180Z,47.6909,147.5345,378.91,4.1,ml,,141,3.328,0.81,us,us1000axzx,2017-10-28T08:59:27.040Z,"169km SW of Vostok, Russia"
2017-10-28T06:27:50.180Z,13.3098,50.8856,10,4.6,ml,,120,12.616,0.98,us,us1000axzm,2017-10-28T06:41:27.040Z,"160km N of Bereeda, Somalia",ea
2017-10-28T05:54:27.290Z,19.0921669,-155.4701691,43.3,2.68,ml,40,181,0.06337,0.23,hv,hv61958436,2017-10-28T06:00:15.610Z,"12km S of Pahala,
2017-10-28T05:36:53.360Z,33.4991667,-116.8006667,4.85,2.56,ml,92,30,0.01317,0.22,ci,ci38032328,2017-10-28T06:43:42.844Z,"9km NE of Aguanga,
2017-10-28T04:08:45.890Z,36.2119,-97.2878,5,2.5,ml_lg,,41,0.064,0.32,us,us1000axz3,2017-10-28T04:22:21.040Z,"8km S of Perry, Oklahoma",eart
2017-10-28T01:52:41.340Z,-22.7612,-176.684,141.36,4.6,ml,,85,6.553,0.88,us,us1000axy9,2017-10-28T02:11:44.040Z,"231km SW of Vaini, Tonga",e
2017-10-27T22:43:37.596Z,59.5819,-153.6007,115.1,3.4,ml,,,,0.51,ak,ak17105789,2017-10-28T03:01:26.040Z,"76km ESE of Old Iliamna, Alaska",ea
2017-10-27T21:32:06.380Z,39.4143,54.9324,28.96,4.4,ml,,135,2.9,1.1,us,us1000axvh,2017-10-27T22:08:48.040Z,"37km NE of Gumdag, Turkmenistan"
2017-10-27T20:38:53.200Z,-30.5362,-71.9326,21.74,4.2,ml,,118,0.29,1.01,us,us1000axtw,2017-10-27T22:11:27.040Z,"70km W of Ovalle, Chile",ear
[Macintosh-5:~/Desktop/lab10] sbilek%
```

Sed

>>> sed s/mb/ml filename

- Basic structure for substitution:
  - s --- is the command that indicates substitution
  - delimiter
    - Can be anything you want, slash (/) is common, so is _ or :
    - But if you need to search something that has a / will need to quote the slash using backslash \

    >>> sed 's/\/usr\/local\/bin/\/usr\/bin' file

    Will change /usr/local/bin to /usr/bin for lines in file that contain /usr/local/bin
  - regular expression or pattern to search for
  - replacement

- If want to do a search and replace globally (in entire file), put "/g" at end. Otherwise it will replace only the first instance found on each line

  >>> sed 's/\/usr\/local\/bin/\/usr\/bin/g' file

- Sed uses regular expressions, same as grep

awk

- Programming language available on most Unix-like OS
- Developed in 1970s (name comes from first letters of last names of developers)
- Useful for manipulating text files
  - One of the most useful unix tools you can develop
- Also able to do floating point math
- Structured as a sequence of patterns and then actions do perform when patterns are found
- Used on text files: columns = fields; lines = records

awk vs nawk vs gawk

- Different versions exist
- awk – original
- nawk – “new awk”, version used on Macs as “awk”
- gawk – GNU awk, standard on linux, compatible with awk and nawk. Can access this on Macs as well – use “gawk” or set an alias for it

- A few minor differences in syntax between versions

Using awk

- Can call it from the command line:

  >>> awk [options] '{commands}' variables infile

  >>> awk –f scriptfile variables infile

- Or create an executable awk script
  - File contains:

    #!/usr/bin/awk

    some set of commands

  >>> chmod +x test.awk

  >>> ./test.awk

awk and text

- awk commands are applied to every record (=line) of a file
- it is designed to separate the data in each line into a field (=column)
- essentially, each field becomes a member of an array so that the first field is $1, second field $2, third field $3 …
- $0 refers to the entire record

awk: Field separators

- the default field separator is one or more white spaces

$1	$2	$3	$4	$5	$6	$7	$8	$9	$10	$11
1	1918	9	22	9	54	49.29	-1.698	98.298	15.0	ehb

- the field separator may be modified by resetting the FS built in variable
- Example:

```
[Macintosh-5:~/Desktop/lab10] sbilek% grep -v '^#' /etc/passwd | head -n1
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
```

Separator is ":", so reset it.

```
[Macintosh-5:~/Desktop/lab10] sbilek% grep -v '^#' /etc/passwd | head -n1 | awk -F":" '{print $1, $3}'
nobody -2
```

awk - print

- One of the most common commands used in awk scripts is print

- awk is not sensitive to white space in the commands

```
>>> awk –F”:”  ‘{ print $1 $3}’  /etc/passwd
nobody-2
```

- two solutions to this

```
>>> awk –F”:”  ‘{ print $1 “  “ $3}’  /etc/passwd
>>> awk –F”:”  ‘{ print $1, $3}’  /etc/passwd
nobody -2
```

- any string or numeric text can be explicitly output using “”

Assume a starting file like so:

1 1 1918 9 22 9 54 49.29 -1.698 98.298 15.0 0.0 0.0 ehb FEQ x

>>> awk '{print "latitude:",$9,"longitude:",$10,"depth:",$11}’ earthquake.txt

latitude: -1.698 longitude: 98.298 depth: 15.0

latitude: 9.599 longitude: 92.802 depth: 30.0

latitude: 4.003 longitude: 94.545 depth: 20.0

1 1 1918 9 22 9 54 49.29 -1.698 98.298 15.0 0.0 0.0 ehb FEQ x

- you can specify a newline in two ways

```
>>> awk '{print "latitude:",$9; print "longitude:",$10}’  earthquake.txt
>>> awk '{print "latitude:",$9”\n”,”longitude:",$10}’  earthquake.txt
```

latitude: -1.698
longitude: 98.298

awk and if

- If statements are very useful in awk:

```
[Macintosh-5:~/Desktop/lab10] sbilek% grep -v '^#' /etc/passwd | head -n3 | awk -F":" '{print $1, $3}'
nobody -2
root 0
daemon 1
[Macintosh-5:~/Desktop/lab10] sbilek% grep -v '^#' /etc/passwd | head -n3 | awk -F":" '{if ($1=="root") print $1, $3}'
root 0
```

awk and math

- Big advantage – it does floating point math (remember bash does not)
- it stores all variables as strings, but when math operators are applied, it converts the strings to floating point numbers if the string consists of numeric characters
- All basic arithmetic is left to right associative

    - + : addition
    - - : subtraction
    - * : multiplication
    - / : division
    - % : remainder or modulus
    - ^ : exponent
    - other standard C programming operators
- Assignment operators
    - = : set variable equal to value on right
    - += : set variable equal to itself plus the value on right
    - -= : set variable equal to itself minus the value on right
    - *= : set variable equal to itself times the value on right
    - /= : set variable equal to itself divided by value on right
    - %= : set variable equal to the remainder of itself divided by the value on the right
    - ^= : set variable equal to the itself to the exponent following the equal sign

awk relational operators

- Returns 1 if true and 0 if false
- All relational operators are left to right associative

  - < : test for less than
  - <= : test for less than or equal to
  - > : test for greater than
  - >= : test for greater than or equal to
  - == : test for equal to
  - != : test for not equal

awk logical operators

- Boolean operators return 1 for true and 0 for false
    - && : logical AND; tests that both expressions are true
        - left to right associative
    - || : logical OR ; tests that one or both of the expressions are true
        - left to right associative
    - ! : logical negation; tests that expression is true

Useful awk built-in variables

- FS: Field Separator (separates columns)
- NR: record number (line number)
- OFS : output field separator
  - Default is whitespace
- ORS : output record separator
  - Default is \n (newline)
- OFMT : output format for numbers
- NF : number of fields in the current record

Using variables in awk

- 1. Assign the shell variables to awk variables after the body of the script, but before you specify the input file

```
awk '{print v1, v2, NF, NR}' v1=$VAR1 file1 v2=$VAR2 file2
```

Or

- 2. Use the -v switch to assign the shell variables to awk variables.

```
awk –v v1=$VAR1 –v v2=$VAR2 '{print v1, v2}' input_file
```

More awk …

- Developing more complex programs in awk
  - Use of for loops, while loops, if/then/else
  - Format output
  - Define functions
  - Matching regular expressions

- Worth spending time exploring websites/books on awk functionality – it will likely become one of your most used tools.