

Shell Programming

Learn the basics of grep, sed, awk, find, sort, uniq, cut, cat, etc.

1. grep Exercise

Obtain the file [grepdata.txt](#). You can save the file on your local system.

Once you have the file, write a series of grep statements that do the following:

- Print all lines that contain a phone number with an extension (the letter x or X followed by four digits).
- Print all lines that begin with three digits followed by a blank. Your answer *must* use the \{ and \} repetition specifier.
- Print all lines that contain a date. Hint: this is a *very* simple pattern. It does not have to work for any year before 2000.
- Print all lines containing a vowel (a, e, i, o, or u) followed by a single character followed by the same vowel again. Thus, it will find “eve” or “adam” but not “vera”. Hint: \{ and \}
- Print all lines that do not begin with a capital S.

Write grep statements that use command-line options along with the pattern to do the following:

- Print all lines that contain CA in either uppercase or lowercase.
- Print all lines that contain an email address (they have an @ in them), preceded by the line number.
- Print all lines that do *not* contain the word Sep. (including the period).
- Print all lines that contain the word **de** as a whole word.

2. sed Exercise

Modify [seddata.txt](#) using sed commands; just copy the text file on your system.

Write a shell file that does the following. It should work on *any* XML file, not just the one provided. That means you can't count on a particular tag always being on a particular line.

- Lines with <article> and </article> should be deleted.
- Replace <title> with Title:, and replace </title> with nothing.
- Replace all <para> and </para> tags with the null string. If the resulting line is empty, delete the line. (You may need to use curly braces to make this happen.)
- Replace all <emphasis> and </emphasis> tags with asterisks. Thus:
--- This is a <emphasis>great</emphasis> bargain.
will become
--- This is a *great* bargain.
- Replace the word web with Web everywhere.
- Replace lines starting with <listing> by ---begin listing
- Replace lines starting with </listing> by ---end listing
- Between the <listing> and </listing>, do these things (you *must* use curly braces to do this!):
 - Replace all occurrences of < with <.
 - Replace all occurrences of > with >.
 - Replace all occurrences of & with &.

- Note: you must do these operations in the order shown above; otherwise, you will get the wrong results!

Note: The & character is a special metacharacter when used in the “replacement” portion of a substitution. For example, if you want to replace the word “and” with “&”, you would do this:

```
s/and/\&/
```

3. awk Exercise

- Using [awkdata.txt](#) file solve the following requirement:

Write an awk script that will compute the average score for every person in the list, the average score for each test, and the average score for each team. If a score is negative, that means the person missed the test, and the score must *not* become part of the average.

Print the output to look like the following. In the list by name, the names must be left-justified in a field of size 10 (hint: %-10s in printf), and the averages must be seven characters wide with two digits to the right of the decimal point (%7.2f).

Name	Average
------	---------

Tom	14.67
Joe	13.00
Maria	15.00
Fred	13.33
Carlos	19.50
Phuong	15.67
Enrique	13.00
Nancy	15.00

Average for Test 1 : 5

Average for Test 2 : 15.75

Average for Test 3 : 22.125

Average for Red Team: 16

Average for Green Team: 13.8889

Average for Blue Team: 14.1667

Notes

- Don't process the first record in the file.
- Your script must use at least one array indexed by number, and at least one array indexed by a string. The usage of the arrays must be relevant to the solution.
- Your script must use at least one for loop.
- Your script must not be tied to this specific data. If the numbers change, or if records are added or deleted, your script must still produce the correct results.

- b. Using [awk2data.txt](#) file solve the following requirement:

Each line of the file contains the name of a point and its x-y coordinates, a semicolon, then the name of a second point and its x-y coordinates. Write an awk script that will do the following:

- Set the field separator to comma and semicolon before processing any lines
- Calculate the distance between the points using this formula:

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

This calculation *must* be done in a user-defined function named `distance`. This function will take four arguments for the x and y coordinates of the two points.

- Calculate the distance between the points using the “city block” formula:

$$city\ block\ distance = |(x_1 - x_2)| + |(y_1 - y_2)|$$

This calculation *must* be done in a user-defined function named `city_block`. This function will take four arguments for the x- and y- coordinates of the two points.

- Print the point names and their coordinates; this will be done by calling a user-defined function named `show_point` twice, once for each point. The point names and coordinates will be followed by the distances. The `show_point` function will take three arguments: the point name, the x-coordinate and the y-coordinate.
- Your script must not be tied to this specific data. If the numbers change, or if records are added or deleted, your script must still produce the correct results.

After you have processed all the records, generate two random numbers in the range 0 to 10 (inclusive). These will be coordinates for point "Q". Then generate two random numbers in the range 0 to 10 (inclusive) as point "R". Print the coordinates for these new points, then calculate and print the distance and city block distance for these new, random points.

Print the output to look like the following. Print only three numbers after the decimal point.

From A (0, 0) to B (3, 4): actual distance 5.000; city block distance 7

From C (4, 7) to D (2, 9): actual distance 2.828; city block distance 4

From E (8, 2) to F (0, 6): actual distance 8.944; city block distance 12

From Q (9, 2) to R (3, 7): actual distance 7.810; city block distance 11

Pseudocode

```
BEGIN { FS="[:;]" }
```

```
function distance( x1, y1, x2, y2 ) {  
  # set variable d to the square root of (x1-x2)*(x1-x2) + (y1-y2)*(y1-y2)  
  # return d  
}
```

```
function city_block( x1, y1, x2, y2 ) {  
  # set variable d1 to x1 - x2  
  # if d1 is less than zero, set d1 to -d1
```

```

# set variable d2 to y1 - y2
# if d2 is less than zero, set d2 to -d2

# set variable result to d1 + d2
# return variable result
}

function show_point( name, x, y ) {
    # use a printf to print the three parameters -- don't use a \n
    # if variable name contains "A",
    # variable x contains 5, and variable y contains 7,
    # the printf should produce:
    # A (5, 7)
}

# $1 is point 1
# $2 is x1
# $3 is y1
# $4 is point 2
# $5 is x2
# $6 is y2

{
    # set variable actual to the distance from ($2, $3) to ($5, $6)
    # set variable city to the city block distance from ($2, $3) to ($5, $6)
    printf "From "
    # call show_point with parameters $1, $2, and $3.
    printf " to "
    # call show_point with parameters $4, $5, and $6
    # now call printf to display the variables actual and city
}

END {
    # seed the random number generator

    # set x1 to the integer part of a random number times 11
    # set y1 to the integer part of a random number times 11
    # set x2 to the integer part of a random number times 11
    # set y2 to the integer part of a random number times 11

    # set variable actual to the distance from (x1,y1) to (x2,y2)
    # set variable city to the city block distance from (x1,y1) to (x2, y2)
    printf "From "
    # call show_point with parameters "Q", x1, and y1
    printf " to "
    # call show_point with parameters "R", x2, and y2
    # now call printf to display the variables actual and city
}

```