SCRIPTING AND COMPUTER ENVIRONMENTS [CSE 505]

ASSIGNMENT 3 – *Shell Scripting*

Due Date : 27th September 2013 9:00 PM

1. **This assignment has 11 questions divided into 4 levels of difficulty :**
   EASY , INTERMEDIATE, ADVANCED and CHALLENGE

2. **There is only 1 CHALLENGE question applicable to you based on your branch.** However, PGSSP students and other non M-Tech students can attempt any but one domain question.

3. **Data Validation is compulsory for every script** that you write. You must take care of the number of arguments. You will lose points if you do not do so.

4. **There are questions with extra credits.** They will fetch you more marks, but not more than the maximum that has been allocated.

5. **Be careful about your submissions**. You must strictly follow the upload format. Failure to do so will make you lose points or may even fetch you a zero.

6. **Follow message formats as specified** . It is recommended that you copy paste the messages that you are required to print as errors and warnings to avoid penalties.

7. **Plagiarism will not be tolerated**. Matching cases will be awarded zero. Be honest in your attempt to solve this assignment.

8. **Any stray file in your submission will fetch you a zero**. Also, you cannot use any helper codes/scripts written in other languages in your script. Our detection schemes will check for such cases and award zero.

9. **Post deadline, no queries shall be entertained** regarding the assignment. You are solely responsible for your submission being valid in terms of the upload format , as well as the answers.

EASY (10 Points each)

1. Write a bash script that echoes itself to stdout, but backwards.

-- Requirements:

*Do not use rev. Using rev will fetch you a zero.

*All whitespaces in the script MUST be preserved

*Reverse of Reversed script must be identical to original

-- Usage/Presentation Requirements :

* Name the script 1.sh

* Usage: bash 1.sh

* Check for commandline arguments to be zero

* Ensure that output is to stdout

-- Errors to handle and message Format :

'Error: Invalid Number of Arguments!' -- If number of arguments is not 0

2. Write a bash script that backs itself up, that is, copies itself to a file named backup.sh.

-- Requirements:

* All whitespaces in the script MUST be preserved

* Do not use cp. Use of cp will fetch you a zero.

-- Usage/Presentation Requirements:

* Name the script 2.sh

* Usage: bash 2.sh

* Check for commandline arguments to be zero

* Output must be as backup.sh

-- Errors to handle and message Format :

'Error: Invalid Number of Arguments!' -- If number of arguments is not 0

3. Write a script to *lowercase* all file/directory names in a directory. Input will be a single directory. If a lowercase file/directory of the same name already exists in that directory, the program should warn the user and NOT overwrite the existing file/directory.

-- Usage/Presentation Requirements:

* Name the script 3.sh
* Usage: bash 3.sh <input_dir_name>
* Check for commandline arguments to be 1 and to be a directory

-- Errors to handle and message Format :

'Error: Invalid Number of Arguments!' -- If number of arguments is not 1

'Error: Input not a directory' -- If argument passed is not a dir

-- Warnings to handle and message Format :

'Warning: Not overwriting <Filename>' -- If a lowercase file/dir exists

EXAMPLE

Assume a directory tree :

```
/home/raghav/TA/Assignment 3
               | ---------> A
                              | ---------> b
                                           | ---------> f1
                                           | ---------> f2
               | ---------> f3
               | ---------> f4
               | ---------> F4
```

Now Running

bash 3.sh /home/raghav/Assignment 3/

Will rename files only in Directory "Assignment 3"and none of its descendants. Here the files are A,f3,f4 and F4 which are desired to be renamed to a,f3,f4 and f4. However, F4 cannot be renamed to 'f4' as another file with that name already exists.

Warning: Not overwriting f4

You must strictly follow this presentation format.

4. Solve a quadratic equation of the form Ax^2 + Bx + C = 0. Have a script take as arguments the coefficients, A, B, and C, and return the solutions to five decimal places.

-- Usage/Presentation Requirements:

* Name the script 4.sh
* Usage: bash 4.sh <A> < B> <C>
* Check for commandline arguments to be 3
* Check output is corrected to 5 decimal places

-- Errors to handle and message Format :

'Error: Invalid Number of Arguments!' -- If number of arguments is not 3

'Error: Input not a number' -- If any of the arguments isn't a number

-- Outputs to handle and message Format :

'No Solutions were found' -- If no solution can be found

'<root1>,<root2>' -- If solutions were found as 2 roots root1 and root2

EXAMPLES

Example test runs :

bash 4.sh 1 4 4
-2.00000,-2.00000

bash 4.sh 12 9 8
No Solutions were found

INTERMEDIATE : (15 Points each)

5. Write a recursive script that recursively lists all files i.e is equivalent to 'ls -R'. Arguments will work like those provided to the ls command. (Hint: Use export and readlink to store absolute path of the script)

-- Requirements:

*You cannot use recursive functions or ls command. You will get a 0 if you do so.

*You must use the same script for each directory you find

*Can take any number of commandline arguments. For each of them, call this script

*If no commandline argument is provided, work on current directory (just like ls!).

-- Usage/Presentation Requirements:

* Name the script 5.sh

* Usage: bash 5.sh <any number of files/directories>

* Support any number of commandline arguments, including zero.

* Validate incoming inputs. Do not stop on Errors.

-- Errors to handle and message Format :

* 'Error: Invalid File/Directory!' -- If an inexistent file/directory is passed

-- Outputs to handle and message Format :

* Output will be a directory along with the files in it. If you encounter a directory in here, call your script to traverse this directory. Here is the format to be followed:

<PathName of Directory>:

<file1>

<file2>

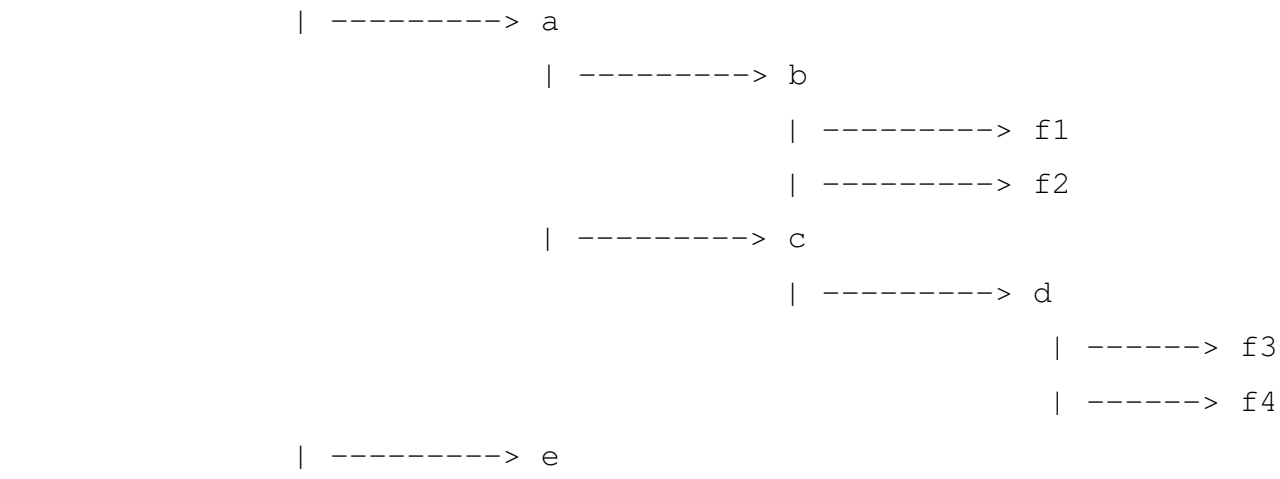
.

.

EXAMPLE

Assume a directory tree :

/home/raghav/TA/Assignment 3



Running

bash 5.sh /home/raghav/Assignment 3/

Will produce :

/home/raghav/TA/Assignment 3:
/home/raghav/TA/Assignment 3/a:
/home/raghav/TA/Assignment 3/a/b:
f1
f2
/home/raghav/TA/Assignment 3/a/c:
/home/raghav/TA/Assignment 3/a/c/d:
f3
f4
e

You must strictly follow this presentation format.

6. Write a script that produces a visualization tree of a directory. You must take only 1 argument as input and it must be a directory.

-- Requirements:

* You must take only one command line argument
* The argument passed must be a directory
* Using the tree command will fetch you zero

-- Usage/Presentation Requirements:

* Name the script 6.sh
* Usage: bash 6.sh <directoryname>

-- Errors to handle and message Format :

'Error: Invalid Number of Arguments!' -- If number of arguments is not 1

'Error: Input not a directory' -- If argument passed is not a dir

-- Outputs to handle and message Format :

|--<space><filename> OR

|<space><space><space>|--<space><filename> OR

|<space><space><space>|<space><space><space>|--<space><filename> etc

i.e do **|<space><space><space>|..** as many times as depth and finally print **|--<space><filename>**

EXAMPLE

Running

```
bash 6.sh mydir
```

Will produce :

```
mydir
|-- a
|-- b
|-- c
|-- Directory1
|    |-- da
|    |-- db
|    |-- Directory12
|    |    |-- dca
|-- e
```

7. Assume that you are an administrator of a lab machine. The labs are open on all days except Sunday.

Write a shell script that logs all accesses to the files in `/etc` during the course of a single day (Take it to be the previous day and NOT 24 hrs from now).

This information should include the filename and access time. Write this data as tabular (tab-separated) formatted records in a logfile called "AccessLog".

(Hint: Use sudo to get over 'permission denied' messages)

Extra Credit : (5 Points ~ Only if both are answered)

1) Find a way to run this everyday at 12:01 AM or 0001 hrs.

2) Find a way to run this every monday and generate a log for accesses made on sundays.

-- Requirements:

* Your script must not take any commandline arguments. Exit if any.
* Output File Name must be '**AccessLog**'
* Each line in 'AccessLog' must be <filename><tab><accesstime>
* AccessTime must have these Fields

  <MonthName><space><Day Number><space> <HH>:<MM>
* For extra credit, type your answers in file 7Extra

-- Usage/Presentation Requirements:

* Name the script 7.sh
* **(Extra Credits Only)** Name the file containing answers to extra credits as 7Extra
* Usage: bash 7.sh

-- Errors to handle and message Format :

'Error: Invalid Number of Arguments!' -- If number of arguments is > 0

EXAMPLE: Log entries in 'AccessLog' should be as follows:

```
/etc/xdg/menus/gnomecc.menu   Sep 11 08:59
/etc/xdg/sni-qt.conf     Sep 11 15:28
/etc/xdg/Trolltech.conf  Sep 11 15:50
```

8. Implement, as a script, a "safe" delete command, 8.sh. Filenames passed as command-line arguments to this script are not deleted, but instead gzipped if not already compressed (use file to check), then moved to a ~/TRASH directory. Upon invocation, the script checks the ~/TRASH directory for files older than 48 hours and permanently deletes them.

Extra credit (5 Points):

Rewrite this script, which can handle files and directories recursively. This would give it the capability of "safely deleting" an entire directory structure.

-- Requirements:

* Your script can take any number of commandline arguments
* If ~/TRASH does not exist, you must create it
* For extra credit, create another script in file 8Extra.sh

-- Usage/Presentation Requirements:

* Name the script 8.sh
* **(Extra Credits Only)** Name the file containing answers to extra credits as 8Extra.sh
* Usage: bash 8.sh <file to delete> <file to delete> .. <file to delete>
* **(Extra Credits Only)**

  Usage: bash 8Extra.sh <file/dir to delete> .. <file/dir to delete>
* The script compresses those files provided in command line first, and then proceeds to delete files older than 48 hours and still residing in ~/TRASH
* The script doesnt quit on error.
* The script doesn't produce any output apart from Error Messages.

-- Errors to handle and message Format :

'Error: Invalid File!' -- If an inexistent file is passed

(Extra Credits Only)

'Error: Invalid File/Directory!' -- If an inexistent file/directory is passed

ADVANCED : (20 Points each)

9. Write a script that implements C String operations strcmp, strcat, strlen, strtok, strstr, and use it in another script that reads 2 strings and then performs all these operations.

-- Requirements:

* Create two scripts 9a.sh and 9b.sh in the same directory.
* Define the String functions in 9a.sh
* 9b.sh takes 2 String Inputs via command line and MUST USE 9a.sh as `. ./9a.sh`
* 9b.sh then executes these (Follow this order strictly) and PRINTS THEIR VALUE
    strcmp $1 $2
    strlen $1
    strlen $2
    strcat $1 $2
    strtok $1 $2
    strstr $1 $2

-- Usage/Presentation Requirements:

* Name the scripts 9a.sh and 9b.sh
* 9a.sh will never be called. However, 9b.sh must take only TWO arguments.
* Usage: bash 9b.sh STRING1 STRING2
* 9b.sh will quit on Error. The only error is invalid number of arguments.
* Return Values of Functions:
    strcmp-- A number identical to strcmp in C
    strlen -- A number identical to strlen in C
    strstr -- A number which indicates start index of searched substring or -1
    strtok -- A string which is the first token before delimiters
    strcat -- A string which concatenates two inputs

-- Errors to handle and message Format : (9b.sh)

'Error: Invalid Number of Arguments!' -- If number of arguments is != 2

10. Write a script to check and validate passwords. The objective is to flag "weak" or easily guessed password candidates.

To be considered acceptable, a password must meet the following minimum qualifications:

- **Minimum length of 8 characters**
- **Must contain at least one numeric character**
- **Must contain at least one of the following non-alphabetic characters: @, #, $, %, &, *, +, -, =**

Do a dictionary check (Use /usr/share/dict/words) on every sequence of at least four consecutive alphabetic characters in the password under test. This will eliminate passwords containing embedded "words" found in a standard dictionary.

The input to your script will be a file containing passwords, one per line. You must generate an output file 'PasswordTest' whose lines will say WEAK or STRONG corresponding to the input password line.

-- Requirements:

* Your script can take only 1 Commandline Argument, the passwords file
* Enforce the rules on each line of the passwords file. If they pass, write STRONG to '**PasswordTest**' else write WEAK

-- Usage/Presentation Requirements:

* Name the script 10.sh
* Usage: bash 10.sh <name of password file>
* Script will quit if invalid file is given as input.
* Output must be in file '**PasswordTest**'

-- Errors to handle and message Format :

'Error: Invalid File!' -- If a non existent file is passed

'Error: Invalid Number of Arguments!' -- If number of arguments is != 1

CHALLENGE : (40 Points each)

M.Tech Students : Attempt the Question that applies to your Domain ; Others : Pick one

11) *For Computer Science students only*

A file contains list of roll numbers of students who copied an assignment from each other. Write a shell script which reads the file and outputs groups of all students who copied the assignments together. For example, if file has:

201305641 201305581

201305641 201305051

201305051 201305581

201305051 201305021

201305021 201305051

201306532 201305111

201306532 201205121

201305641 201205874

201305532 201305182

then output would be:

201205121 201306532 201305111

201205874 201305641 201305581 201305051 201305021

201305182 201305532

-- Requirements:

* Your script can take only 1 Commandline Argument, the copy checker result file.
* Identify the groups who have copied from each other. For every group print a line to the file '**CopyGroups**'
* For example, **201205874 201305641 201305581 201305051 201305021** will be the second line in '**CopyGroups**' for the given file

-- Usage/Presentation Requirements:

* Name the script 11.sh

* Usage: bash 11.sh <name of copy checker result file>

* Script will quit if an invalid file is given as input.

-- Errors to handle and message Format :

'Error: Invalid File!' -- If a non existent file is passed

'Error: Invalid Number of Arguments!' -- If number of arguments is != 1

12) *For VLSI students only*

Assembler :

Consider a hypothetical machine :

(i)This machine has 5 registers of 64 bit each,viz. AX BX CX DX EX where AX is an accumulator while the rest are general purpose. All of them must store data in binary.

(ii)We now have an assembly code for this machine, which you have to evaluate. The assembly code instructions work on memory locations as well as registers. However, all memory locations must be declared and optionally initialized before being used. This is done by a DECLARE directive, which appears as the first line in the assembly program. (Thus, you will declare all memory variables in the first line ONLY)

+----------Assembly Code ----------+

```
DECLARE var1,var2,var3=value,var4,...
<Any instruction that deals with any of var1 var2 var3 var4 or AX
BX CX DX EX >
```

+----------Assembly Code ----------+

The code will refer to memory locations, that you must implement as variables. They can store data in any format you wish (binary/decimal).

Each instruction can have either a memory location or a register as an operand. Memory locations are just variable names containing alphabets.

(iii) The assembly code has these instructions which you are required to implement :

DECLARE (<variable[=val]>)*

A variable name or variable=value. This is a list of such declarations separated by commas. ***You can assume*** *variable names to have only letters of the English Alphabet.*

Example: `DELCARE A`

`DECLARE A=2,B`

`DECLARE A=2,B,C=1,OUTPUT`

ADD <register>,<register> ;; Add contents of two registers and store in AX

SUB <register>,<register> ;; Subtract contents of two registers and store in AX

MUL <register>,<register> ;; Multiply contents of two registers and store in AX

DIV <register>,<register> ;; Divide contents of two registers and store in AX
;; Throws a DIVIDEBYZERO exception

STORE <register>,<MemoryLoc> ;; Store contents of register to Memory Loc

LOAD <MemoryLoc>,<register> ;; Store contents of Memory Loc to register

MOV <register1>,<register2> ;; Move contents of Register1 to Register 2

PRINTREG <register> ; Print contents of the register. Must be in binary

PRINTMEM <MemLocn> ; Print contents of memory location. Must be in decimal

SHL <register> ; Shift contents of register LEFT by 1 bit

SHR <register> ; Shift contents of register RIGHT by 1 bit

ROL <register> ; Rotate contents of register LEFT by 1 bit

ROR <register> ; Rotate contents of register RIGHT by 1 bit

(iv)Now given an assembly code in this language, your script must execute all the instructions. Your script must handle these exceptions while doing so :

1) DECLARATION ERROR (If Declaration format doesn't match or not found in the first line)

2) INSTRUCTION ERROR (If an invalid instruction is found)

3) DIVIDEBYZERO ERROR

Format for exception is a message : 'ERROR: <exception name>' ; Print it and then exit your code

Example :

```
echo 'ERROR: INSTRUCTION ERROR'
```

(Hint : You can use bc ; Use printf with %064 for 64bit output)

(Sample Input can be found in 12_sample.txt)

-- Requirements:

* Your script can take only 1 Commandline Argument, the assembly code file.
* For every line of the assembly code, evaluate it and modify necessary registers and memory locations.
* You can assume that apart from the declaration, input assembly instructions are VALID. **You dont have to perform any validation for the instruction format.**

-- Usage/Presentation Requirements:

* Name the script 12.sh
* Usage: bash 12.sh <name of assembly code file>
* Script will quit if it encounters any exception

-- Errors to handle and message Format :

'Error: Invalid File!' -- If an inexistent file is passed

'Error: Invalid Number of Arguments!' -- If number of arguments is != 1

'ERROR: <exception name>' -- If an exception is encountered

An illustration is shown in the next page.

Example :

Let this file be 'program.asm'

```
DECLARE A=2,B,C=1,O
LOAD A,BX       (BX=2)
ADD BX,BX       (AX=2+2=4)
SHL AX          (AX=8)
STORE AX,B      (B=8)
LOAD C,CX       (CX=1)
MUL AX,CX       (AX=8*1=8)
STORE AX,O      (O=8)
PRINTMEM O      (Will Print 8)
PRINTREG AX     (Will Print 8 in 64bit format)(Hint: Use printf)
```

Now when you run

bash 12.sh program.asm

Output will be

```
8
0000000000000000000000000000000000000000000000000000000000001000
```

13) *For CSIS students only*

Cryptography:

Perform encryption and decryption.

Given a file with text in it (having a-z A-Z 0-9, space and dot(.)), encrypt the file using the scheme given below (next page), write encrypted text in a separate file. Encrypted File name should be 13_Encrypted.txt

Read the encrypted data from this file and decrypt it using reverse scheme and write the decrypted data into another file. File name should be 13_Decrypted.txt .Make sure your original file data and decrypted file data match.

Make separate functions to encrypt and decrypt the data and write them in a single script file 13.sh

Encoding Rule:

Perform a transformation on the input stream using an encpryption key, (which is a sequence of column numbers) as follows:

1)Divide the stream into a matrix, with the number of columns=no of characters in the key.

2)If a row doesnt fill completely, append it with blanks ('')*

3)Now, output the matrix columnwise as specified in the encryption key.

Example :

PlainText = "attack post poned until two am"

Let encryption key be **4312567**

Steps:

1) Length of key=7 . Therefore, you need to generate a matrix of 7 **columns.**

2)Distribute the characters of the plaintext 7 per row as :

```
a t t a c k
p o s t   p o
n e d   u n t
i l   t w o
a m
```

3) The last five columns in row 5 need to be appended with blanks. So use a character '*' at these places and generate the matrix as follows:

```
a t t a c k
p o s t   p o
n e d   u n t
i l   t w o
a m * * * * *
```

4) Now based on the ecryption key, output columnwise. i.e The key here is 4312567. Therefore, label columns 1-7 as 4312567. Now output column with label 1 (which is col 3), then column with label 2 (which is col 4) and so on.

```
4 3 1 2 5 6 7
a t t a c k
p o s t   p o
n e d   u n t
i l   t w o
a m * * * * *
    ^ ^
```

Therefore, the encrypted text is : '**tsd *at t*toelmapniac uw*kpno* ot *'**

i.e **tsd *** | **at t*** | **toelm** | **apnia** | **c uw*** | **kpno*** | **ot ***

This should be the output to 13_Encrypted.txt

Decoding Rule:

The decrpytion is performed by a reverse transformation. First, the ciphertext characters must be transformed using the following rule:

1)Divide the stream into a matrix, with the number of rows=no of characters in the decryption key.

2)Now, output the matrix row wise as specified in the decryption key and delete the trailing blanks.

Example:

We start from our ecrypted value, which was '**tsd *at t*toelmapniac uw*kpno* ot *'**

Encryption key was **4312567** (Col 4 will be Col1,Col 3 will be Col2)
i.e 1->4 2->3 3->1 4->2 5->5 6->6 7->7
Therefore, the Decryption key will be reverse of these pairs,
i.e 4->1 3->2 1->3 2->4 5->5 6->6 7->7
If we sort these pairs we get
1->3 2->4 3->2 4->1 5->5 6->6 7->7
Thus,Decryption key is **3421567**

Steps:

1) Length of key=7 . Therefore, you need to generate a matrix of 7 **rows.** Thus you will have (Length of String)/keylength columns

2)Distribute the characters of the string 'tsd *at t*toelmapniac uw*kpno* ot * ' across 7 rows as (Length of String/keylength characters per row i.e 35/7=5 characters per row)

```
t s d   *
a t   t *
t o e l m
a p n i a
c   u w *
k p n o *
  o t   *
```

3) Now based on the decryption key, rearrange the rows. i.e The key here is **3421567.** Therefore, label rows 1-7 as 3421567. Now output row with label 1 (which is row 3), then row with label 2 (which is col 4) and so on.

```
3 t s d   *
4 a t   t *
2 t o e l m
1 a p n i a
5 c   u w *
6 k p n o *
7   o t   *
```

After reordering

```
1 a p n i a
2 t o e l m
3 t s d   *
4 a t   t *
5 c   u w *
6 k p n o *
7   o t   *
```

4) Now output columnwise and remove trailing zeroes

The columnwise output is : 'attack post poned until two am*****'

Therefore, the decrypted text is : 'attack post poned until two am'

-- Requirements:

* Your script can take only 2 Commandline Arguments, the file to encode and encryption key
* You have to generate the decryption key yourself

-- Usage/Presentation Requirements:

* Name the script 13.sh
* Usage: bash 13.sh <File to encode> <Encryption Key>
* Your script will not output anything to stdout. The encrypted and decrypted messages MUST be stored in files `13_Encrypted.txt` and `13_Decrypted.txt` respectively.

-- Errors to handle and message Format :

'Error: Invalid Number of Arguments!' -- If number of arguments is not 2

'Error: Invalid File!' -- If a non existent file is passed

14) *For BioInformatics students only*

You are given two files "14_ref.txt" and "14_sample.txt" which contain a part of Illumina mRNA chip expression data for the BRCA1 gene. This gene is a tumor suppressor gene and is involved in resistance to Breast Cancer in Humans.

Write a shell script for the following:

a. Distance calculation b/w corresponding cells in the expression data files. e.g. : for first cell (ILMN_1725881), the values will be 196.6413 and 187.7823. Write the distance values for each cell to a file : "output.txt". Use minkowski distance calculating algorithm.

b. Cluster the cells into 4 clusters using distance values calculated in step a. Perform unweighted hierarchical clustering. Use any threshold of your choice. Write the cluster number and the values of that cluster to the file "output.txt". (use of any simple approach for clustering will be fine).

c. Calculate the median distance between each of the 4 clusters.

d. Find the total distance b/w the genes. Assuming that each cell had 30 consecutive bp's of BRCA1 gene (from start) and using all the results, what all inferences can you derive about :-

- conserved regions in the gene

- positions of mutation in the gene

- extent of divergence

Requirements will be specified soon