CS320 Assignment #3

Purpose

This assignment is designed to familiarize you with C and bash working together.

Requirements

This assignment consists of six major requirements.

1) Developing solutions to the four problems below.
2) Ensuring that your programs compile and run correctly using the tools available in a standard linux distribution.
3) Documenting your solutions correctly.
4) Organizing your *git repository* correctly.

Problem

1) In this assignment you have been hired by a mid size corporation to perform some corporate espionage. Your target is a set of encrypted documents. In this project you will obtain, break and decode these documents for your employer.
    a. Step 1.

Your employer, Silly Parts Inc, know that an employee for their rival company, Terrible Tribbles, is also in your class. The Terrible Tribbles employee, Yoko Jamika, is known to you and you know that they do not have very good security practices. It is probable that they use the same password on a number of accounts. If you could find their password for the class you should be able to login to their named account on the Terrible Tribbles san diego server. After you log into the server you should be able to find a set of encrypted files, and the encryption executable in their directory

For this first step, you will need to write a script(prog3_1.sh) that takes three arguments. The first argument will be a password file (You do know where to obtain this information right? You found them in the last assignment). The second argument will be a target server, for this assignment the Terrible Tribbles server is hosted at: `sd.lindeneau.com`. The third argument will be a name (first and last separated by a space), you know that in companies it is common to use an employee's first name as their user name (among other combinations, but for this assignment we are using the first name).
This script will obtain Yoko's password from the provided source (argument 1). It will then copy all of the files in Yoko's home directory on the target server (argument 2) with the `.enc` extension, in addition to the encryption executable: `encryptor` to your current directory. You are encouraged to use the scp program, which will allow you to securely copy files from a remote server to your current local directory. An example of copying the remote `encryptor`

executable to the current directory is:

```
expect -c "
        spawn scp <USERNAME>@<TARGET>:encryptor .
        expect password: { send <PASSWORD>\r }
        sleep 1
        exit
"
```

More info here: http://www.tamas.io/automatic-scp-using-expect-and-bash/

Your script should echo the local filename of every file copied (including the encryptor executable).

b. The second part of this assignment will be to figure out how to break the encryption. Fortunately for us, Yoko's encryption program is custom made and is based upon one of the oldest, and most broken, encryption schemes: the Caesar Cipher. We don't have any cleartext files, but we do have the encryptor executable, which has the key embedded into the executable. To get ahold of some cleartext we can simply generate some known plaintext (we just pick some text… literally anything you want) and then pass it into the encryptor executable on standard in. The encryptor program will output the encrypted text on standard out. To break the encryption, we need to figure out the offset between the plaintext and the resulting ciphertext.

   To accomplish this you will need to write a c program(prog3_2.c) and a script(prog3_2.sh). The c program will calculate the offset used in the encryptor program and the script will automate the process.

   The c program will take a command line argument and a phrase on standard in. The command line argument will be the plaintext and the standard input will be the ciphertext. Your program prog3_2.c should then output the difference between between the characters that are passed on the command line and the characters that are input through standard input. (The minimum number of characters you will need to figure this out is 1. Yes 1. You only need to have 1 character where you know both the plain and cypher text to break a Caesar Cipher). An example of a call to this program is:

   ```
   echo "k" | ./prog3_2 g
   ```

   Which should output `4` on standard out. (The difference between g and k is 4 characters. All of the characters will have the same difference).

   Your script should automate the process of calling the encryptor program with some generated plaintext (something you choose) and then calling prog3_2 with the plaintext

and the ciphertext. Your script should echo the difference between the plaintext and the ciphertext. Your script does not need to take any arguments.

c. The third part of this program will also consist of both a c program(prog3_3.c) and a script(prog3_3.sh). The purpose of this section is to decipher the encrypted files you obtained in part 1. The c program will implement the Caesar Cipher. There are number of resources on how to do this online. Your program should take a command line argument, which will be the offset, while the input through standard in will be the part that will be rotated(encrypted/decrypted). To simplify this assignment you will only need to rotate characters that are alpha characters (letters). No other characters should be rotated. You can also assume that the input will only be a single line long.

   Your script will take a single command line argument, which will be the rotation argument to the Caesar c program you wrote above. Your script should then call the prog3_3 you wrote above on each of the `.enc` files you got in step 1. Your script should output a `.dec` file for each `.enc` file that exists. It should echo the name of each `.dec` file it generates. (You will be able to use the negative of the number you found in step 2. This is how you decrypt Caesar Cipher encryptions.)

2) You must ensure that your code can be run on a standard Debian based linux distribution using the shell tools of your choice. You may develop your solution on any machine you desire, as long as the final solution works on a standard linux distrubtion.
3) Your solution must have a complete comment header as is detailed below. During runtime, each of your solutions to section 1 must output a correct **title string** as the first line printed. It should be in this format:

   `Assignment #1-1, <NAME>, <EMAIL>`

   Each problem should follow this format, with the second number incremented for each problem in part 1. For example the second problem in part one should have the title string:

   `Assignment #1-2, Scott Lindeneau, slindeneau@gmail.com`

4) You must place a copy of your solutions inside a repository named cs320Assignment2 in your git profile on gitlab. Your files **MUST** be named **`prog3_1.sh`**, **`prog3_2.sh`**,**`prog3_2.c`**, **`prog3_3.sh`**,**`prog3_3.c`** and must not be modified after the turn in time. The modified timestamp for each **`.c/.sh`** file in your **`repository`** will be used as the submission time for that file. If it is after the due date, it will be counted as late.

   You are also responsible for two additional files that will be part of every assignment (after assignment #1). You **MUST** place a copy of the programming rubric in your repository. You must not rename it (it should be: cs320programmingrubric.pdf) and by placing it in the repository you should also be reading it and acknowledging that this is the rubric by which you will be graded.

You will also place a README.md file[1] in your assignment directory. This file should contain a description **IN YOUR OWN WORDS** of the project along with a short description of each file, what it is, how to compile and/or run it. These descriptions may be short as long as they are accurate.

YOU MAY NOT HAVE ANY OTHER FILES IN YOUR GIT REPOSITORY. If you have any other files in your git repository the assignment will be considered **not** complete.

[1] If you are familiar with reddit commenting/posting then you are familiar with markdown. They use markdown exclusively for user provided content. Most git repos use md as the format of choice for readme files and install text.
Here is a good cheat sheet: https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet

Example Output

```
Assignment #3-1, Scott Lindeneau, slindeneau@gmail.com
encryptor
file1.enc
file2.enc
file3.enc

Assignment #3-2, Scott Lindeneau, slindeneau@gmail.com
7

Assignment #3-3, Scott Lindeneau, slindeneau@gmail.com
file1.dec
file2.dec
file3.dec
```

Additional Details

- You do not need to do input checking. You can assume that the input value is a valid input.

Late Policy

Late programs will be accepted with a penalty of 5% per day for three days after due date.

Cheating Policy

There is a zero tolerance policy on cheating in this course. You are expected to complete all programming assignments on your own. Collaboration with other students in the course is not permitted. You may discuss ideas or solutions in general terms with other students, but you must not exchange code. (Remember that you can get help from me. This is not cheating, but is in fact encouraged.) I will examine your code carefully. Anyone caught cheating on a programming assignment or on an exam will receive an "F" in the course, and a referral to Judicial Procedures.

Extra Credit

The problem should be titled Assignment #3-4 and saved in a file prog3_4.sh.

Extra credit is graded on a complete/incomplete basis and will be added to your class total at the end of the semester.

Ok. So lets automate most of the process. Write a script that takes four command line arguments, the first will be an encryptor program (one that does not take command line arguments and simply encrypts the stdin), the second will be a breaker program, the third will be a general purpose ceaser cipher encryptor program (the one you wrote for part three) and the fourth will be an encrypted file. Your script should determine the rotation for the encryptor program using the breaker, and then decrypt the encrypted file and print its contents to stdout. No other output should be displayed. No other user input should be necessary.