Computer Science 35L: Week 0 Bash Worksheet Solutions

Exploring Shell Commands

1. SSH into `lnxsrv11.seas.ucla.edu`.
    a. You will need to be connected to either the UCLA VPN or a UCLA WiFi network.

2. **Manual pages.** Run the command `man man`. What does it say?
    a. You can navigate the man page with the arrow keys. Pressing 'd' goes *down* one page, 'u' goes *up* one page, and 'q' *quits*.
    b. Reading man pages is not easy, but it's a good skill to practice. If you're in a hurry, you might find the TL;DR pages easier to digest.

3. Look through the man or TL;DR page for `shuf`.
    a. What does `shuf` do? How do you use it?
    `shuf` will take lines of text as input and output them in a random order.
    b. Scroll down to the "Author" section of the man page. Who wrote the source code for this program?
    Professor Eggert!

4. **Shuffling text.**
    a. Run the command `seq 10`. If you were to guess, what do you think `seq` does?
    This command will output the numbers 1 through 10, separated by newlines. The command `seq n` will output the numbers 1 through n.
    b. Run the command `seq 10 >numbers.txt`. This will save the output of `seq 10` to a file called `numbers.txt`.
    c. Run the command `shuf numbers.txt`. What did `shuf` do?
    This command will output a random permutation of the numbers 1 through 10, separated by newlines.
    d. Without creating a temporary file, can you instead use a shell pipe to directly pass the output of `seq` to `shuf`?
    `seq 10 | shuf`

5. **Navigating Linux.** Type the following commands:
    a. `mkdir shell-scripts`
    b. `cd shell-scripts`.
    c. Use Emacs to write a shell script called `permutation.sh` that outputs a random ordering of the numbers 1 through 10.
    `#!/bin/bash`
    `seq 10 | shuf`

d. `chmod +x permutation.sh`
e. Test your program!
f. `cat permutation.sh`.
g. `rm permutation.sh`
h. `cd ..`
i. `rmdir shell-scripts`

Encrypted Emails

Problem Statement

The year is 1993. The World Wide Web, which was invented less than five years ago, is still in its infancy. Hackers are everywhere, so you must be careful!

You and your friend would like to use an exciting new service called *LAO Mail* to communicate with each other. However, you are both concerned about the security of *LAO Mail*, and are worried that mischievous hackers may try to send emails on either of your behalf.

To ensure this doesn't happen, the two of you devise the following scheme. When either of you send an email, the last three lines of the email will contain the sender, the recipient, and the date, as follows:

```
from romeo
to juliet
date 01-08
```

By itself, this isn't difficult for a hacker to replicate. To make your emails secure, you use the SHA-1 hash function to convert the above lines into the following hashes:

```
5ad0646549f73c767a14216546fa1d4192ce7525
afb627b3649953927746ed92e6ed3d6e9c0118ab
4e05e3b85cb523bbc1fefc2a1ae0eb49fd50c7a5
```

For *extra* security, you also randomly shuffle the order of these hashes before appending them to the email:

```
4e05e3b85cb523bbc1fefc2a1ae0eb49fd50c7a5
5ad0646549f73c767a14216546fa1d4192ce7525
afb627b3649953927746ed92e6ed3d6e9c0118ab
```

When your friend receives your email, they'll know exactly how to recreate these three hashes. To verify your identity, they can then simply compare the hashes they recreate with the hashes you sent[1]. To save time, you decide to create a shell script to help automate this process.

[1] Of course, this scheme is far from perfect. If a hacker finds out how these hashes are created, they may still be able to impersonate you or your friend. Security isn't easy!

Automating Things

Since you have to append several hashes to every email you send to your friend, you decide to create a shell script to help automate this process. Here's how you want the shell script to work:

```
./sign.sh <FILE> <SENDER'S NAME> <RECIPIENT'S NAME> <DATE>
```

The above command will append the three desired hashes to `FILE`. If you're not sure how to do this, don't worry! We'll guide you through each step of the process :).

Hello, World!

First, let's create a basic "Hello, world!" shell script.

1. Create a file named `sign.sh`. This file will contain your shell script.
2. Copy the following text into `sign.sh`. What do these lines do?

```
#!/bin/bash

echo "Hello, world!"
```

3. Run the command `chmod +x sign.sh`. This allows you to execute your shell script. You can do so with the command `./sign.sh`.
    a. What is the output of `sign.sh`?

If this was your first shell script, congratulations! We hope you write many more in the years to come.

Command Line Arguments

Next, we would like to receive arguments from the command line.

1. Retrieve the first command line argument (`FILE`). Print it to standard output with `echo`.
    a. Hint: You might find the built-in variable `$1` useful.
2. Retrieve the second command line argument (`SENDER'S NAME`). Print out the string `"from <SENDER'S NAME>"`, replacing `<SENDER'S NAME>` with the argument passed in.
    a. Hint: You can expand variables within strings by using double quotes and a dollar sign. For example, if you have a variable named `FOO`, you can run the command `echo "The contents of FOO is $FOO"`. The dollar sign in front of the second `FOO` will cause Bash to replace it with the actual contents of the variable `FOO`. This is known as shell parameter expansion.
3. Retrieve the third command line argument (`RECIPIENT'S NAME`). Print out the string `"to <RECIPIENT'S NAME>"`.

4. Retrieve the last command line argument (`DATE`). Print out the string `"date <DATE>"`.

If you now run the command `./sign.sh file romeo juliet 01-08`, you should see the output:

```
file
from romeo
to juliet
date 01-08
```

Hashing With SHA-1

Let's now hash this output.

1. Look through the man or TL;DR page for `sha1sum`. Note that `sha1sum` can accept input either as a file or through standard input (just like `shuf`).
2. Instead of printing the string `"from <SENDER'S NAME>"`, use `sha1sum` to print its hash.
    a. How can you do this without creating a temporary file?
    b. Hint: You can pass a string to `sha1sum` by using `echo` and a pipe.
3. Likewise, hash the strings `"to <RECIPIENT'S NAME>"` and `"date <DATE>"`.

Running the same command as before, you should now see the output (the hyphens are fine):

```
5ad0646549f73c767a14216546fa1d4192ce7525 -
Afb627b3649953927746ed92e6ed3d6e9c0118ab -
4e05e3b85cb523bbc1fefc2a1ae0eb49fd50c7a5 -
```

Wrapping Up

Almost there!

1. Save each of the three hashes you printed into variables.
    a. You can save the output of a command into a variable by surrounding the command with `$()`.
        - For example, `hello=$(echo "goodbye")` will save `'goodbye'` into `hello`.
2. In a *single command*, print out the three hashes, separated by newlines.
    a. Hint: You might try something along the lines of `echo "$var1\n$var2\n$var3"`. However, this won't work, since Bash doesn't normally interpret `'\n'` as a newline character. Instead, you can use the

command `printf "$var1\n$var2\n$var3\n"` or `echo -e "$var1\n$var2\n$var3"` to format your variable.

3. Use `shuf` to shuffle the order of the hashes.
4. Append the output of `shuf` to the `FILE` argument.
    a. You can use >> to append the output of a command to a file (used in the same way as >, which instead overwrites the file).

You should be done! Don't forget to test your program to make sure it works.

Solution:

```
#!/bin/bash

FILE="$1"
FROM="$2"
TO="$3"
DATE="$4"

from_hash=$(echo "from: $FROM" | sha1sum)
to_hash=$(echo "to: $TO" | sha1sum)
date_hash=$(echo "date: $DATE" | sha1sum)

printf "$from_hash\n$to_hash\n$date_hash\n" |
  shuf >>"$FILE"
```

Bonus: Verifying an Email

If you just can't get enough of shell scripting, try writing a shell script called `verify.sh` that verifies an email ends with the desired hashes. Running the following command should notify you if an email is fraudulent:

```
./verify.sh <FILE> <SENDER'S NAME> <RECIPIENT'S NAME> <DATE>
```

You might find the `tail` and `sort` programs to be useful. You should also know how to write an if statement in Bash.

Solution:

```
#!/bin/bash

FILE="$1"
FROM="$2"
```

```
TO="$3"
DATE="$4"

from_hash=$(echo "from: $FROM" | sha1sum)
to_hash=$(echo "to: $TO" | sha1sum)
date_hash=$(echo "date: $DATE" | sha1sum)

expected_hashes=$(
   printf "%s\n%s\n%s\n" "$from_hash" "$to_hash" "$date_hash"| sort
)
actual_hashes=$(tail -n 3 "$FILE" | sort)

if [ "$expected_hashes" = "$actual_hashes" ]
then
 exit 0
else
 exit 1
fi
```