Homework 02: Scripting the Shell

The goal of this homework assignment is to allow you to practice shell scripting and explore networking in a Unix environment.

For this assignment, record your responses to the following activities in the `README.md` file in the `homework02` folder of your **assignments** GitLab repository along with any other scripts you create and push your work by **11:59 AM Saturday, February 3**.

Activity 0: Pulling Changes

Before starting this homework assignment, you should first perform a `git pull` to retrieve any changes in your remote GitLab repository:

```
$ cd path/to/repository                    # Go to assignments repository

$ git checkout master                      # Make sure we are in master branch

$ git pull --rebase                        # Get any remote changes not present locall
```

You will need to do this from time to time since the TAs will be grading your assignments and merging in your previous work. To integrate these changes into your local repository, you will need to do a `git pull` to retrieve it from the remote GitLab repository.

Note: If you get the following messages when you do a `git push` :

```
! [rejected]        master -> master (fetch first)
error: failed to push some refs to 'https://bitbucket.org/djasek12/assignments '
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first merge the remote changes (e.g.,
hint: 'git pull') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Or:

```
! [rejected]        master -> master (fetch first)
error: failed to push some refs to '...'
To prevent you from losing history, non-fast-forward updates were rejected
Merge the remote changes before pushing again.  See the 'Note about fast-forwards'
section of 'git push --help' for details
```

This means you need to perform a `git pull` and merge in the remote changes before performing a `git push` as described above.

Don't forget to create a new branch for this assignment:

```
$ git checkout -b homework02               # Create homework02 branch and check it out

$ cd homework02                            # Go into homework02 folder
```

Activity 1: Saving the World (2 Points)

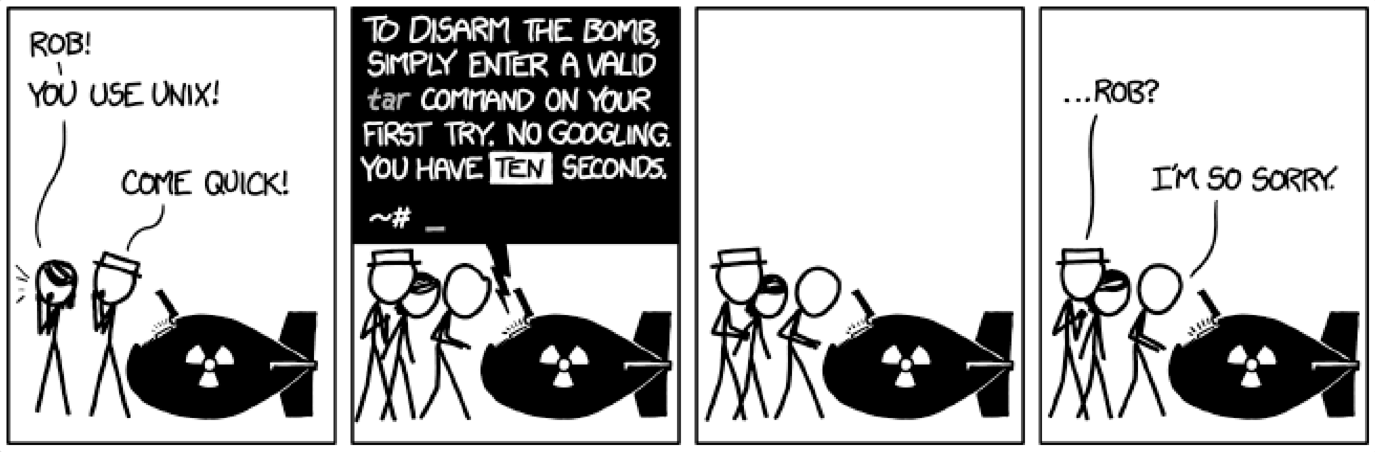
For the first activity, you are to extend the initial prototype below to create a script called `extract.sh` which will unpack any tarball or archive:

```
#!/bin/sh

# Initial prototype

case $1 in
    *.tar.gz)
        tar xzvf $1
        ;;
esac
```

This would be handy should you ever face the following situation:


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

Task 1: `extract.sh`

The `extract.sh` script takes a list of archives as arguments. For each of these archives, the script determines which command is necessary to extract or unpack the archive based on the file extension and then executes the appropriate command. The script should support the following extensions:

```
.tgz, .tar.gz         # gzip
.tbz, .tar.bz2        # bzip2
.txz, .tar.xz         # xz
.zip, .jar            # zip
```

If no arguments are based to the script, then `extract.sh` should display a **usage** message and exit with an error code. Likewise, if an archive with an unknown extension is provided, the script should emit the message `"Unknown archive format: "` followed by the name of the file, and exit with an error code.

Here is an example of `extract.sh` in action:

```
# Display usage message
$ ./extract.sh
Usage: extract.sh archive1 archive2...

# Download test archive
$ curl -LO https://www3.nd.edu/~pbui/teaching/cse.20289.sp18/static/tar/cmatrix-1.2a.

# Extract test archive
$ ./extract.sh cmatrix-1.2a.tar.gz
cmatrix-1.2a/
cmatrix-1.2a/NEWS
cmatrix-1.2a/TODO
cmatrix-1.2a/aclocal.m4
cmatrix-1.2a/README
cmatrix-1.2a/configure
cmatrix-1.2a/configure.in
cmatrix-1.2a/cmatrix.1
cmatrix-1.2a/cmatrix.c
cmatrix-1.2a/config.guess
cmatrix-1.2a/install-sh
cmatrix-1.2a/cmatrix.spec
cmatrix-1.2a/cmatrix.spec.in
cmatrix-1.2a/matrix.fnt
cmatrix-1.2a/config.sub
cmatrix-1.2a/missing
cmatrix-1.2a/mkinstalldirs
cmatrix-1.2a/Makefile.am
cmatrix-1.2a/Makefile.in
cmatrix-1.2a/mtx.pcf
cmatrix-1.2a/config.h.in
cmatrix-1.2a/matrix.psf.gz
cmatrix-1.2a/stamp-h.in
cmatrix-1.2a/AUTHORS
cmatrix-1.2a/INSTALL
cmatrix-1.2a/ChangeLog
cmatrix-1.2a/acconfig.h
cmatrix-1.2a/COPYING

# Remove extracted folder
$ rm -fr cmatrix-1.2a
```

Hints

The **hints** may spoil the surprises in the journey, so only view them when you are stuck: **Toggle Hints**

Task 2: `test_extract.sh`

To aid you in testing the `extract.sh` script, we are providing you with test_extract.sh which you can use as follows:

```
# Download script
$ curl -LO http://www3.nd.edu/~pbui/teaching/cse.20289.sp18/static/sh/test_extract.sh

# Make script executable
$ chmod +x test_extract.sh

# Run test script
$ ./test_extract.sh
extract.sh test successful!
```

⚠ Test Scripts

Note, the test scripts are meant as a **smoke test** but are not considered exhaustive or comprehensive. That is, if there is a failure during testing then that indicates that something is **most likely** wrong with your script. On the other hand, if the test script succeeds, then you can be confident in the functionality of your script, but it is **not guaranteed** it is **100% correct**.

Task 3: `README.md`

In your `README.md` , describe how you implemented the `extract.sh` script. In particular, briefly discuss:

1. How you checked and handled having no arguments.
2. How you determined which command to use for each argument.
3. What was the most challenging aspect of writing this script and how you overcame this issue.

Activity 2: Predicting the Future (4 Points)

For the second activity, you are to create a script called `fortune.sh` , which is your own modified version of the **TROLL** from Homework 01. In this case, your cute ASCII friend will tell the future ala a Magic 8-ball.

Task 1: `fortune.sh`

The `fortune.sh` script, like the **TROLL**, takes no command line arguments. When executed, it should emit a message using cowsay, which can be found in the following directory on the student machines:

```
/afs/nd.edu/user15/pbui/pub/bin
```

After the initial message, the `fortune.sh` script should prompt the user for a question. Finally, it should display its prediction, which is one of the following messages:

```
It is certain
It is decidedly so
Without a doubt
Yes, definitely
You may rely on it
As I see it, yes
Most likely
Outlook good
Yes
Signs point to yes
Reply hazy try again
Ask again later
Better not tell you now
Cannot predict now
Concentrate and ask again
Don't count on it
My reply is no
My sources say no
Outlook not so good
Very doubtful
```

Note: The `fortune.sh` script should randomly choose one of these as the prediction each time a question is asked.

During the execution of the script, if the user enters a **blank** question, then `fortune.sh` should prompt the user again until a non-empty question is entered.

Moreover, if `fortune.sh` receives a `SIGHUP` , `SIGINT` , or `SIGTERM` signal, then it should **emit** a special message and **exit** with a failure error code.

</> Not The TROLL

Unlike the `TROLL` , your program should **quit** after **one non-blank question**. It should only loop if the user enters in a blank question (and prompt the user again).

Here are some examples of `fortune.sh` in action:

```
# Ask a question
$ ./fortune.sh
 _______________________________________
/ Hello pbui, what question do you have \
\ for me today?                         /
 ---------------------------------------
  \
   \
       .--.
      |o_o |
      |:_/ |
     //   \ \
    (|     | )
   /'\_   _/`\
   \___)=(___/

Question? Will the patriots win the superbowl?
 _______________
< Very doubtful >
 ---------------
  \
   \
       .--.
      |o_o |
      |:_/ |
     //   \ \
    (|     | )
   /'\_   _/`\
   \___)=(___/

# Ask a question (with some blanks)
$ ./fortune.sh
 _________________________________
< Hi pbui, what can I do for you? >
 ---------------------------------
  \
   \
       .--.
      |o_o |
      |:_/ |
     //   \ \
    (|     | )
   /'\_   _/`\
   \___)=(___/

Question?
Question?
Question?
Question? Will it snow today?
 ___________________
< Don't count on it >
 -------------------
  \
   \
       .--.
      |o_o |
      |:_/ |
     //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/

# Exit Early with Control-C
$ ./fortune.sh
 _________________________________
< Why pbui, what is on your mind? >
 ---------------------------------
  \
   \
       .--.
      |o_o |
      |:_/ |
     //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/

Question? ^C __________________
< Leaving so soon? >
 ------------------
  \
   \
       .--.
      |o_o |
      |:_/ |
     //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/
```

Hints

The **hints** may spoil the surprises in the journey, so only view them when you are stuck: Toggle Hints

Task 2: `test_fortune.sh`

To aid you in testing the `fortune.sh` script, we are providing you with test_fortune.sh which you can use as follows:

```
# Download script
$ curl -LO http://www3.nd.edu/~pbui/teaching/cse.20289.sp18/static/sh/test_fortune.sh

# Make script executable
$ chmod +x test_fortune.sh

# Run test script
$ ./test_fortune.sh
...
```

Note: Unlike the previous test, this script merely runs your `fortune.sh` script and tries to interact with it. It does not check whether or not it worked (because it has no way of knowing your messages will be), so this is simply a way for you to automate running your `fortune.sh` through the basic scenarios.

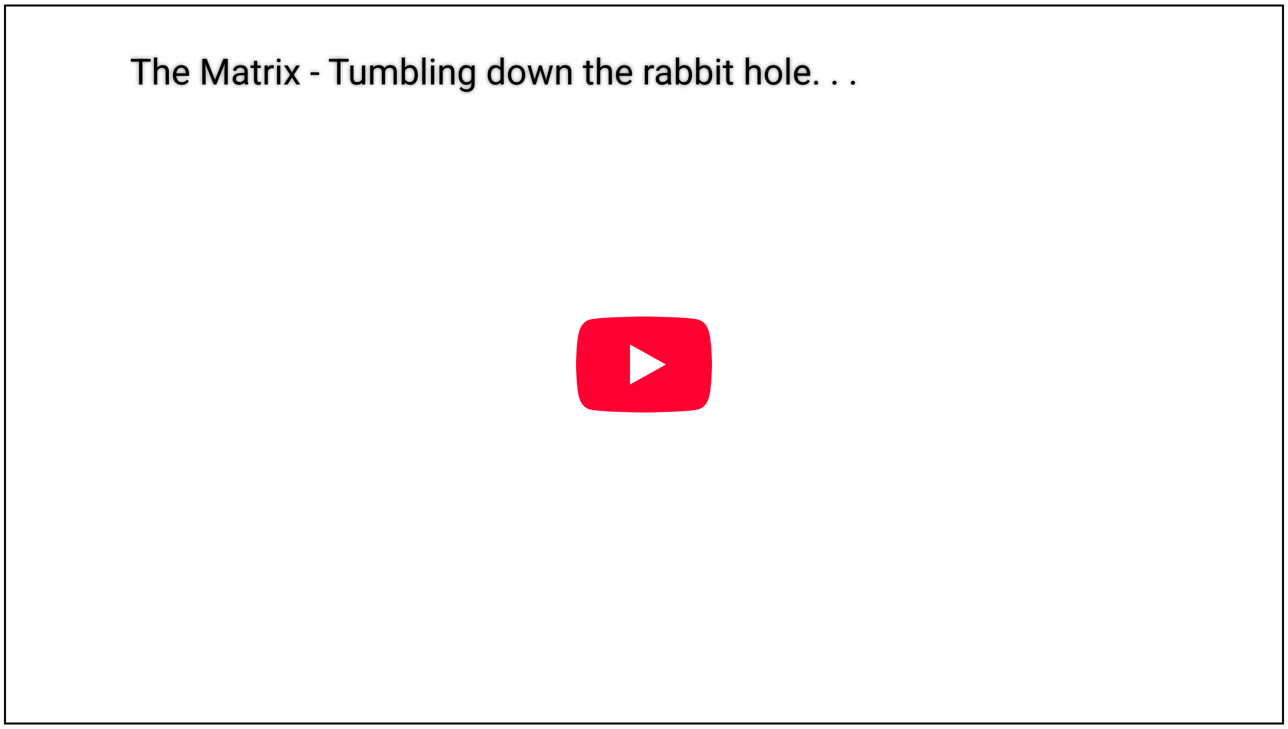
Task 3: `README.md`

In your `README.md` , describe how you implemented the `fortune.sh` script. In particular, briefly discuss:

1. How you displayed random messages to the user.
2. How you handled signals.
3. How you read input from the user.
4. What was the most challenging aspect of writing this script and how you overcame this issue.

Activity 3: Meeting the Oracle (4 Points)

For the third activity, you are to go down the **rabbit hole** and talk to the Oracle:



Of course, access to the Oracle is heavily restricted, so you will need to use your hacking skills to locate her and gain an audience with her. Once you connect with her, she will hopefully give you a message about how to use your **awesome Unix skills**.

To begin your journey, you will have to find the entrance to her sanctuary. Thanks to a glitch in the Matrix we have been able to narrow her location to the machine `xavier.h4x0r.space`. Scan that machine for a HTTP port in the `9000 - 9999` range and then proceed down the path of discovery and enlightenment.

As you travel to the Oracle, record what you witnessed and what commands you executed in your `README.md`. Be as thorough as possible, since we will need this log to reconstruct your path and hopefully use it for future meetings. Here is an example of what your `README.md` for this activity should look like:

```
### Activity 3: Meeting with Oracle

1. My first step was to scan `xavier.h4x0r.space` for a HTTP port:

        $ command ...
        Output...

    As can be seen, there are `X` ports in the `9000` - `9999` range.

2. Next, I tried to access the HTTP server:

        $ command ...
        Output...

...
```

At the end of your journey, reflect on what the Oracle told you and what you learned in your journey down the rabbit hole.

Student Machines and AFS

This activity requires that you work on the **student** machines or have access to **AFS** on your Unix machine.

Hints

The **hints** may spoil the surprises in the journey, so only view them when you are stuck: Toggle Hints

Guru Point (1 Extra Credit Point)

For **extra credit**, you are to customize your favorite `$EDITOR` by adding useful programming or development extensions and plugins that will make your Unix programming environment more productive. Here are some sources of inspiration:

Nano

- Nano Text Editor and nanorc Tips and Tricks

Vim

- Vim as your IDE
- Vim: revisited
- How to turn Vim into a full-fledged IDE

Emacs

- C/C++ Development Environment for Emacs
- Mastering Emacs
- Org mode

If you come across any other tutorials or resources, please share them on Slack.

Linux Users Group

The Notre Dame Linux Users Group will be having a meeting that covers using and customizing Vim on **Tuesday, January 30** at **6:00 PM** in **B011 DeBartolo Hall**. All folks interested in Linux or Open Source software are welcome.

To get credit for this Guru Point, show your text editor customizations to the instructor or a TA to verify.

Feedback

If you have any questions, comments, or concerns regarding the course, please provide your feedback at the end of your `README.md`.

Submission

To submit your assignment, please commit your work to the `homework02` folder of your `homework02` branch in your **assignments** GitLab repository:

```
$ cd path/to/repository                    # Go to assignments repository
$ git checkout master                      # Make sure we are in master branch
$ git pull --rebase                        # Make sure we are up-to-date with GitLab
$ git checkout -b homework01               # Create homework01 branch and check it out
$ cd homework02                            # Go to Homework 02 directory
...
$ $EDITOR extract.sh                       # Edit script
$ git add extract.sh                       # Mark changes for commit
$ git commit -m "homework02: activity 1"   # Record changes
...
$ $EDITOR fortune.sh                       # Edit script
$ git add fortune.sh                       # Mark changes for commit
$ git commit -m "homework02: activity 2"   # Record changes
...
$ $EDITOR README.md                        # Edit appropriate README.md
$ git add README.md                        # Mark changes for commit
$ git commit -m "homework02: README"       # Record changes
...
$ git push -u origin homework02            # Push branch to GitLab
```

Remember to create a merge request and assign the appropriate TA from the Reading 02 TA List.