Tarea 03: Shell de Unix

El propósito de esta tarea es trabajar un lenguaje imperativo, y entretenerse un poco, haciendo shell scripts. En esta taria necesitará utilizar variables, if, case, for, while, funciones, traps y otras capacidades básicas del lenguaje del bourne shell.

#!/bin/sh

Independientemente de cual es el shel que usted utiliza, para los propósitos de esta tarea se utilizará como nuestro **lenguaje de scripting** el bourne shell (ie. `/bin/sh`), el cual es el Unix shell más básico y portatil.

Note que el bourne shell no es el mismo que bourne again shell (ie. `/bin/bash`). El segundo de puede considerar una extensión del bourne shell original, y como tál, es compatible con él. Aunque muchas lecturas discuten las características en el contexto del bourne again shell, nos enfocaremos el el subset que corresponde al bourne shell.

Actividad 01: Cocine (5 Puntos)

Para esta actividad, ustede debe crear un script del shell, `cocine.sh`, que automáticamente compila a ejecutable todos los archivos fuente que se encuentran en un subdirectorio.

Tarea1 1:

El script `cocine.sh` no tiene argumentos. En vez de eso busca en el directorio actual por archivos cuyo sufijo haga match con el valor de la variable del ambiente `SUFIJOS`. Por default, `SUFIJOS` debe ser `.c` y entonces hacer match con cualquier archivo que termine en `.c`.

Para cada archivo fuente, `cocine.sh` debería compilarlo en un ejecutable. El compilador utilizado se debe obtener de la variable del ambiente `CC` y lar banderas especificadas deben estar en `CFLAGS`. Por default, `CC` debe ser seteado a `gcc` y `CFLAGS` debe ser `-std=gnu99 -Wall`.

Si el proceso falla durante la compilación de los archivos cuyo sufijo hace match con `SUFIJOS`, entonces el script debe hacer `exit` inmediatamente con un status distinto de cero.

Adicionalmente, si la variable `VERBOSE` esta set, entonces `cocine.sh` debe desplegar el el comando que utiliza para compilar cada uno de los archivos fuente.

Aquí algunos ejemplos de `cocine.sh` en acción:

```
# No se preocupe del caso en que no hay archivos fuente en el directorio
$ ls
cocine.sh

# Compile archivos fuente
$ ./cocine.sh
gcc: error: *.c: No such file or directory

# Crea archivo fuente
$ cat > hola.c <<EOF
#include <stdio.h>
int main(int argc, char *argv[]) {
    puts("Hola, Mundo!");
    return 0;
}
EOF

# Compile archivo fuente
$ ./cocine.sh

# Corra el programa compilado
$ ./hola
Hola, Mundo!

# Compile archivo fuente (definiendo VERBOSE)
$ env VERBOSE=1 ./cocine.sh
gcc -std=gnu99 -Wall -o hola hola.c

# Corra programa compilado
$ ./hola
Hola, Mundo!

# Cambie el nombre de los archivos
$ mv hola.c hola.cc

# Compile archivos fuente (redefiniendo CC, CFLAGS, SUFIJOS, VERBOSE)
$ env CC=g++ CFLAGS="-g -Wall" SUFFIXES=.cc VERBOSE=1 ./cocine.sh
g++ -g -Wall -o hola hola.cc

# Corra el archivo compilado
$ ./hola
Hola, Mundo!
```

Pistas

- Puede utilizar basename para eliminar los `SUFIJOS`.
- Para permitir que el valor de variables del ambiente se impongan sobre variables definidas en el script, puede o bien utilizar una sentencia `if` para asignar valor a la variable condicionalmente o bien puede utilizar parameter expansion.
- Para terminar el script cuando la compilación falla, puede utilizar un `if` o short circuit evaluation.

Tarea 2: `README.md`

En su `README.md`, conteste las siguientes preguntas:

1. Describa como implementó el script `cocine.sh` script. En particular, discuta brevemente:

   a. ¿Cómo manejó asignar valores default a las variables?

   b. ¿Cómo iteró por todos los archivos que hacian match con `SUFIJOS`?

   c. ¿Cómo manejó la variable `VERBOSE`.?

   d. ¿Cómo hizo para terminar rápido una vez que la compilación fallaba?

2. Compara utilizar `cocine.sh` con make. ¿Cuáles son las ventajas y desventajas de ambos métodos de compilación automática? ¿Qué utilizará en el futuro (`cocine.sh`, make, o algo distinto)?

Actividad 02: Utilización del Disco (5 Puntos)

Para esta segunda actividad, usted debe crear un script llamado `uso_del_disco.sh`, que dado un directorio, el script lista los `n` archivos o directorios más grandes.

Tarea 1: `uso_del_disco.sh`

El script `uso_del_disco.sh` script acepta dos posibles **flags** y luego una lista de directorios:

```
$ ./uso_del_disco.sh
uso: uso_del_disco.sh [-a -n N] directorio...
```

La bandera `-a` significa que el script debe listar tanto archivos como directorios, y la bandera `-n` indica que solo las primeras `N` entradas deben ser desplegadas (si `N` no se especifica, el default debe ser `10`).

Aquí hay algunos ejemplos de `uso_del_disco.sh` en acción:

```
# Corra en /etc
$ ./uso_del_disco.sh /etc
15M     /etc/
6.5M    /etc/udev
2.0M    /etc/ssl
1.9M    /etc/ssl/certs
1.8M    /etc/ca-certificates/extracted
1.8M    /etc/ca-certificates
1.1M    /etc/pacman.d/gnupg
1.1M    /etc/pacman.d
780K    /etc/ca-certificates/extracted/cadir
340K    /etc/ssh

# Corra en /etc (limite la salida a los primeros 5)
$ ./uso_del_disco.sh -n 5 /etc
15M     /etc
6.5M    /etc/udev
2.0M    /etc/ssl
1.9M    /etc/ssl/certs
1.8M    /etc/ca-certificates/extracted

# Corra en /etc (incluya los archivos)
$ ./uso_del_disco.sh -a /etc
15M     /etc
6.5M    /etc/udev/hwdb.bin
6.5M    /etc/udev
2.0M    /etc/ssl
1.9M    /etc/ssl/certs
1.8M    /etc/ca-certificates/extracted
1.8M    /etc/ca-certificates
```

```
1.1M    /etc/pacman.d/gnupg
1.1M    /etc/pacman.d
780K    /etc/ca-certificates/extracted/cadir

# Corra en /etc (incluya archivos, limítelo a los primeros 5)
$ ./uso_del_disco.sh -a -n 5 /etc
15M     /etc
6.5M    /etc/udev/hwdb.bin
6.5M    /etc/udev
2.0M    /etc/ssl
1.9M    /etc/ssl/certs

# Corra en /etc y /var (limite la salida a los primeros 2 por directorio)
$ ./uso_del_disco.sh -n 2 /etc /var
15M     /etc
6.5M    /etc/udev
7.8G    /var
4.1G    /var/log/journal/4e5d9581840047019e266fb2123b1c90
```

Pistas

- Necesitará utilizar du, sort, and head.
- Para parsear las opciones de la linea de comandos debe utilizar getopts.
- Para suprimir errores irrelevantes de du debe redireccionar el **stderr** a `/dev/null`.

Tarea 3: `README.md`

En su `README.md`, conteste las siguientes preguntas:

1. Describa como implemento su script `uso_del_disco.sh` script. En particular, discuta brevemente:

   a. ¿Cómo parseo los argumentos de la linea de comandos?

   b. ¿Cómo manejó el caso cuando no había argumentos?

   c. ¿Cómo procesó cada argumento/directorio?

   d. ¿Cómo incorporo los comandos de la linea de comandos a los comandos que utilizó para calcular los primeros `N` items de cada directorio?

2. Discuta cual fué el problema más dificil de este punto y porqué. Adicionalmente indentifique cual fue la parte del código que consimió la mayor parte del código. ¿Le sorprende esto? ¿Porqué? / ¿Porqué no

Actividad 03: Taunt (5 Puntos)

Para esta actividad final usted debe crear un script `taunt.sh`

Tarea 1:

El script `taunt.sh` script, no toma argumentos. Cuando es ejecutado debe emitir un mensaje utilizando cowsay, que puede instalar en Linux o bajar de algún servidor autorizado

Después del mensaje inicial, `taunt.sh` debe manejar tres casos:

1. Si no recibe mensaje después de `10` segundos entonces el script `taunt.sh` deberá emitir un mensaje gracioso y terminar/exit.
2. Si recibe la señal `SIGHUP`, entonces debe emitir un mensaje especial y terminar/exit.
3. Si recibe un `SIGINT` o `SIGTERM`, entonces debe emitir un it should emit un choteo y terminar/exit.

Estos escenarios son demostrados en lo que sigue. El contenido de los mensajes son completamente a criterio suyo. Diviértase … pero manténgalo cortés (o no!).

Pistas

- Puede ajustar la variable del ambiente `PATH` dentro de su script para que incluya el directorio en que se encuentra el cowsay.
- cowsay viene con varias figuras ... algunas **No recomendables**.
- Puede decidir hacer funciones en su script para manejar las señales.
- Puede decidir poner a dormir al script varias veces en vez de hacer un solo sleep laaargo.
- Puede utilizar here documents para construir mensajes largos.

Para ayudarle en su testing de chotee `.sh` script, se provee un archivo test_taunt.sh que se puede utilizar de la siguiente manera:

```
# Download script
$ curl -O http://www3.nd.edu/~pbui/teaching/cse.20189.sp16/static/sh/test_taunt.sh

# Make script executable
$ chmod +x test_taunt.sh

# Run test script
$ ./test_taunt.sh
Testing timeout...
 _________________________________
< Uh... What? What do you want? >
 ---------------------------------
   \
    \
        .--.
      |o_o |
      |:_/ |
      //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/


 ____________________________________
/ Ugh... I'm going back to sleep... \
\ ZzZzZ...                          /
 ------------------------------------
   \
    \
        .--.
      |o_o |
      |:_/ |
      //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/

Testing SIGTERM...
 _________________________________
< Uh... What? What do you want? >
 ---------------------------------
   \
    \
        .--.
```

```
      |o_o |
      |:_/ |
      //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/


_______________________________________
/  WTF are you trying to do here?   \
| Terminate me?                     |
|                                   |
| Get lost, pbui! I'm going back to |
\ sleep!                            /
-------------------------------------
  \
    \
        .--.
      |o_o |
      |:_/ |
      //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/

Testing SIGHUP...
_______________________________
< Uh... What? What do you want? >
-------------------------------
  \
    \
        .--.
      |o_o |
      |:_/ |
      //   \ \
    (|     | )
    /'\_   _/`\
    \___)=(___/
```

```
 ______________________________________
/  Hmm... I recognize you pbui... Here's \
| the message you need to give to the    |
| ORACLE:                                |
|                                        |
\ Y29odj0xNDU1NDAyODA2                   /
 --------------------------------------
   \
    \
        .--.
       |o_o |
       |:_/ |
      //   \ \
     (|     | )
     /'\_   _/`\
     \___)=(___/
```

Tarea 2: `README.md`

1. Describa como implementó el script `taunt.sh`. En particular, discuta brevemente:

   a. ¿Cómo manejó las distintas señales?

   b. ¿Cómo le pasó los mensajes [cowsay]?

   c. ¿Cómo manejó timeout?

2. Compare el escribir [shell scripts] con el escribir programas en **C**. ¿Cuál es más facil? ¿Cuál prefiere? ¿Cuando es que usaría una en vez del otro?