Tarea 3

Explicación Actividad 01 Cocine.sh

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

    - ¿Cómo manejó asignar valores default a las variables?

      Esto se hizo con un la sentencia `if`. Se verifica si el valor del tamaño de la variable es 0, de ser así se le asigna el valor por defecto mencionado en la descripción del problema.

    - ¿Cómo iteró por todos los archivos que hacian match con SUFIJOS?

      Utilizando la sentencia `for archivo in *$SUFIJOS`, lo cual lee todos los archivos en el directorio que coincidan con el sufijo indicado por la variable SUFIJOS.

    - ¿Cómo manejó la variable VERBOSE.?

      Simplemente es un `if` dentro del `for` que verifica si la variable VERBOSE es igual a 1 (TRUE), si es así entonces se hace un echo del comando a ejecutar. De no ser así entonces no se imprime en pantalla el comando.

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

      Luego de la ejecución del comando, se verifica si el valor de salida del último comando ejecutado no es igual a 0 con `if [ "$?" -ne 0 ]`, de no ser 0, entonces se sale del script con `exit 1`, además de un mensaje.

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)?

   En mi experiencia casi no he usado makefiles, pero lo poco que conozco sé que no se debe hacer mucho para poder obtener un archivo que compile, ejecute y borre los archivos ejecutables creados. Además, de que colocando reglas se puede acceder a distintas partes del código. Por otro lado, al usar archivos .sh noté que se debe de realizar muchos if..else dependiendo de lo que se necesite pero esto también ayuda a verificar algunas cosas. Por mi parte, siento que en un futuro voy a utilizar ambos, ya que los dos me parecen bastante convenientes y además pueden ser utiles en diferentes situaciones.

Explicación Actividad 02 uso_del_disco.sh

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

    - ¿Cómo parseo los argumentos de la linea de comandos?

      Para esto usé el comando `getopts` en un `while`. Con un `swtich..case` usando la variable `$opt` fui comparadando para los casos (-a, -n) y realizaba una determinada operación según la entrada. Por ejemplo, para -n, obtuve el valor de su argumento con `$OPTARG`.

    - ¿Cómo manejó el caso cuando no había argumentos?

Si se ejecutaba el script sin argumentos se verificaba con un `if` al inicio el cual espera que la cantidad de argumentos sea diferente de 0. De no ser así, se imprime un mensaje de uso y se sale con exit 1.

- ¿Cómo procesó cada argumento/directorio?

  Por medio de un for, luego de ejecutar `shift` se usó a variable `$@` para obtener cada uno de los directorios luego de las flags.

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

  Para esto usé un pipe luego del comando `du` y `sort`, y luego al comando `head` se le pasó el numero de ítems con `-n "$n"` donde la variable `n` contiene el número de ítems (por defecto 10).

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?

   El problema más difícil fue poder ejecutar el comando `du` correctamente, ya que cuando lo corrió no hacía lo que quería. Además, poder especificar cuando leer solo los directorios y cuando leer todo. Primero traté con `grep "/$"` pero no funcionó, así que tuve que ir probando en la termial diferentes combinaciones del comando `du` hasta poder obtener el resultado deseao. En realidad no me sorprender porque nunca antes había usado ese comando, así que tuve que tomar el tiempo para poder aprender a usarlo de la manera deseada.

Explicación Actividad 03 taunt.sh

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

   - ¿Cómo manejó las distintas señales?

     Para manejar las señales usé el comando `trap`, con esto definí la acción a tomar dependiendo de la señal recibida. Las acciones las dividí en funciones. Para `SIGINT` y `SIGTERM` usé la función `terminar` y para `SIGHUP` usé la función `especial`. Luego dentro del bucle que espera 10 segundos mientras recibe una señal, pregunté si alguna de las señales había sido recibidas con un `if` y preguntando si el archivo existía (`.hup_received` y `.int_term_received`). De recibir alguna se pasa a llamar alguna de las funciones mencionadas.

   - ¿Cómo le pasó los mensajes cowsay?

     Son simplemente un string con el mensaje luego del comando `cowsay`.

   - ¿Cómo manejó timeout?

     Para poder manejar esto tuve que hacer uso de `bash` ya que no encontré ninguna solución para hacerlo con `sh`. Dentro del while se pregunta si la variable especial `$SECONDS` es mayor o igual a 10, de ser así se termina el script.

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?

   Para mi escribir programas en C es más robusto. Se cuenta con un mayor rango de funciones y formas de realizar alguna acción, como por ejemplo, el poder manejar el sleep mientras se recibe alguna entrada. Es mucho más fácil aprender shell scripting que C pero al mismo tiempo con C se aprende a manejar mucho la memoria y tener cuidado con las direcciones de memoria. En mi opinión prefiero C ya que lo encuentro más completo. Pero usaría Shell scripting cuando necesite automatizar alguna tarea que realice en el línea de comandos ya que esto es más rápido y sencillo que hacerlo en C.