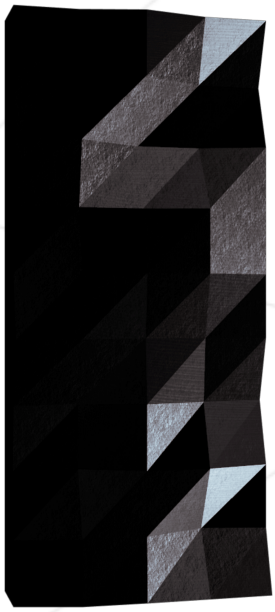

Minishell

Igual de bonito que un shell de verdad

Resumen: El objetivo de este proyecto es crear un shell sencillo. Serï¿½ su propio pequeï¿½o bash o zsh. Va a aprender mucho sobre los process y los file descriptors.

Índice general

Capítulo I

Introducciï¿½n

La existencia de shells estï¿½ ï¿½ntimamente relacionada con la existencia de la informï¿½tica.
Por aquel entonces, los desarrolladores estaban todos de acuerdo en que `comunicar con un ordenador por medio de interruptores 1/0 era muy molesto`.
La consecuencia lï¿½gica fue que se inventï¿½ un medio de comunicaciï¿½n a travï¿½s de lï¿½neas de comandos interactivas con un lenguaje que, hasta cierto punto, se pareciï¿½a al inglï¿½s.

Con `Minishell` va a viajar a travï¿½s del tiempo y se va a enfrentar a los problemas que la gente solï¿½a tener cuando todavï¿½a no existï¿½a `Windows`.

Capítulo II

Reglas comunes

- Su proyecto debe estar programado respetando la Norma. Si tiene archivos o funciones extras, entrarán dentro de la verificación de la norma y, como haya algún error de norma, tendrá un 0 en el proyecto.
- Sus funciones no pueden pararse de forma inesperada (segmentation fault, bus error, double free, etc.) salvo en el caso de un comportamiento indefinido. Si esto ocurre, se considerará que su proyecto no es funcional y tendrá un 0 en el proyecto.
- Cualquier memoria reservada en el montón (heap) tendrá que ser liberada cuando sea necesario. No se tolerará ninguna fuga de memoria.
- Si el proyecto lo requiere, tendrá que entregar un Makefile que compilará sus códigos fuente para crear la salida solicitada, utilizando los flags `-Wall`, `-Wextra` y `-Werror`. Su Makefile no debe hacer relink.
- Si el proyecto requiere un Makefile, su Makefile debe incluir al menos las reglas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los extras, debe incluir en su Makefile una regla `bonus` que añadirá los headers, bibliotecas o funciones que no estén permitidos en la parte principal del proyecto. Los extras deben estar dentro de un archivo `_bonus.{c/h}`. Las evaluaciones de la parte obligatoria y de la parte extra se hacen por separado.
- Si el proyecto autoriza su `libft`, debe copiar sus códigos fuente y y su Makefile asociado en un directorio libft, dentro de la raíz. El Makefile de su proyecto debe compilar la biblioteca con la ayuda de su Makefile y después compilar el proyecto.
- Le recomendamos que cree programas de prueba para su proyecto, aunque ese trabajo **no será ni entregado ni evaluado**. Esto le dará la oportunidad de probar fácilmente su trabajo al igual que el de sus compañeros.
- Deberá entregar su trabajo en el git que se le ha asignado. Solo se evaluará el trabajo que se suba al git. Si Deepthought debe corregir su trabajo, lo hará al final de las evaluaciones por sus pares. Si surge un error durante la evaluación Deepthought, esta última se parará.

Capítulo III

Parte Obligatoria

Nombre del programa	`minishell`
Ficheros de entrega	
Makefile	`Sï¿½`
Argumentos	
Funciones externas autorizadas	`printf, malloc, free, write, open, read, close, fork, wait, waitpid, wait3, wait4, signal, kill, exit, getcwd, chdir, stat, lstat, fstat, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, errno`
Libft autorizada	`Sï¿½`
Descripción	`Escriba un shell`

Su shell tiene que:

- Mostrar un prompt a la espera de un comando nuevo
- Buscar y lanzar el ejecutable adecuado (basï¿½ndose en una variable de entorno PATH o utilizando un `path` absoluto), como en el bash
- Tendrï¿½ que implementar los siguientes builtins (comandos incorporados):
  - `echo` y la opciï¿½n '-n'
  - `cd` con solo direcciï¿½n absoluta o relativa
  - `pwd` sin ninguna opciï¿½n
  - `export` sin ninguna opciï¿½n
  - `unset` sin ninguna opciï¿½n
  - `env` sin ninguna opciï¿½n ni argumento
  - `exit` sin ninguna opciï¿½n
- `;` tendrï¿½ que separar los comandos en la lï¿½nea de comandos

- ’ y " deben funcionar como en el bash, salvo para las multilï¿½neas.
- Las redirecciones <, > y “>>” deben funcionar como en el bash, salvo en las agregaciones de fd
- Las tuberï¿½as (tambiï¿½n llamadas pipes) | deben funcionar como en el bash
- Las variables de entorno ($ seguido de caracteres) deben funcionar como en el bash.
- $? debe funcionar como en el bash
- ctrl-C, ctrl-D y ctrl-\ deben mostrar el mismo resultado que en el bash.

Capítulo IV

Parte Extra

- Si la parte obligatoria no estï¿½ completa, no tiene acceso a los ejercicios extra
- No necesita realizar todos los ejercicios extra
- Redirecciï¿½n "`<<`" como en el bash
- Histï¿½rico y ediciï¿½n de lï¿½nea utilizando termcaps (`man tgetent` para ver algunos ejemplos)
  - Editar la lï¿½nea donde se encuentre el cursor
  - Desplazar el cursor hacia la izquierda o hacia la derecha para editar la lï¿½nea en un lugar preciso. Por supuesto, los caracteres que se aï¿½adan tendrï¿½n que insertarse en medio de los que ya existan, como en el bash.
  - Teclas arriba y abajo para navegar por el historial de comandos, que despuï¿½s podremos editar (inline, no en el historial)
  - Copiar, cortar y pegar todo o parte de una lï¿½nea utilizando una secuencia de teclas de su eleccciï¿½n
  - Desplazarse entre las palabras con las teclas `CTRL+izquierda y CTRL+derecha`
  - Ir directamente al principio o al final de la lï¿½nea utilizando las teclas `home` y `end`
  - Escribir Y editar un comando sobre varias lï¿½neas. En este caso, nos encantarï¿½a que `CTRL+arriba` y `CTRL + abajo` nos permitiesen desplazarnos entre las lï¿½neas, quedï¿½ndonos en la misma columna o en la mï¿½s apropiada.
- `&&` y `||` con los parï¿½ntesis para gestionar las prioridades, como en el bash
- wildcard `*` como en el bash