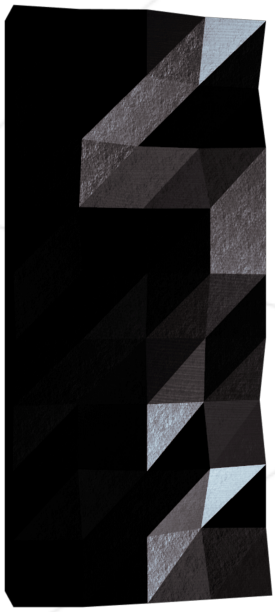

Minishell

Tan bonito como shell

Resumen: El objetivo de este proyecto es que crees un shell sencillo. Sí, tu propio pequeño bash. Aprenderás un montón sobre procesos y descriptores de archivo (file descriptors).

Versión: 9.0

Índice general

Capítulo I

Introducción

La existencia de los shells se remonta a los orígenes de la informática.

Por aquel entonces, todos los programadores estaban de acuerdo en que *comunicarse con un ordenador utilizando interruptores de 1/0 era realmente frustrante.*

Era cuestión de tiempo que llegaran a la idea de crear un software para comunicarse con los ordenadores utilizando líneas de comando interactivas en un lenguaje parecido al utilizado por los humanos.

Gracias a `Minishell`, podrás viajar en el tiempo y volver a los problemas a los que la gente se enfrentaba cuando `Windows` no existía.

Capítulo II

Instrucciones generales

- Tu proyecto deberá estar escrito en C.
- Tu proyecto debe estar escrito siguiendo la Norma. Si tienes archivos o funciones adicionales, estas están incluidas en la verificación de la Norma y tendrás un `0` si hay algún error de norma en cualquiera de ellos.
- Tus funciones no deben terminar de forma inesperada (segfault, bus error, double free, etc) excepto en el caso de comportamientos indefinidos. Si esto sucede, tu proyecto será considerado no funcional y recibirás un `0` durante la evaluación.
- Toda la memoria asignada en el heap deberá liberarse adecuadamente cuando sea necesario. No se permitirán leaks de memoria.
- Si el enunciado lo requiere, deberás entregar un `Makefile` que compilará tus archivos fuente al output requerido con las flags `-Wall`, `-Werror` y `-Wextra`, utilizar cc y por supuesto tu `Makefile` no debe hacer relink.
- Tu `Makefile` debe contener al menos las normas `$(NAME)`, `all`, `clean`, `fclean` y `re`.
- Para entregar los bonus de tu proyecto deberás incluir una regla `bonus` en tu `Makefile`, en la que añadirás todos los headers, librerías o funciones que estén prohibidas en la parte principal del proyecto. Los bonus deben estar en archivos distintos `_bonus.{c/h}`. La parte obligatoria y los bonus se evalúan por separado.
- Si tu proyecto permite el uso de la `libft`, deberás copiar su fuente y sus `Makefile` asociados en un directorio `libft` con su correspondiente `Makefile`. El `Makefile` de tu proyecto debe compilar primero la librería utilizando su `Makefile`, y después compilar el proyecto.
- Te recomendamos crear programas de prueba para tu proyecto, aunque este trabajo **no será entregado ni evaluado**. Te dará la oportunidad de verificar que tu programa funciona correctamente durante tu evaluación y la de otros compañeros. Y sí, tienes permitido utilizar estas pruebas durante tu evaluación o la de otros compañeros.
- Entrega tu trabajo en tu repositorio `Git` asignado. Solo el trabajo de tu repositorio `Git` será evaluado. Si Deepthought evalúa tu trabajo, lo hará después de tus com-

pañeros. Si se encuentra un error durante la evaluación de Deepthought, esta habrá terminado.

Capítulo III

Instrucciones sobre la IA

Contexto

Durante tu proceso de aprendizaje, la IA puede ayudarte con muchas tareas diferentes. Tómate el tiempo necesario para explorar las diversas capacidades de las herramientas de IA y cómo pueden apoyarte con tu trabajo. Sin embargo, siempre debes abordarlas con precaución y evaluar de forma crítica los resultados. Ya sea código, documentación, ideas o explicaciones técnicas, nunca podrás saber con total certeza si tu pregunta está bien formulada o si el contenido generado es el adecuado. Las personas que te rodean son tu recurso más valioso para ayudarte a evitar errores y puntos ciegos.

Mensaje principal:

- Utiliza la IA para reducir las tareas repetitivas o tediosas.
- Desarrolla habilidades de prompting, ya sea para programacion o para otros temas, que beneficiarán tu futura carrera.
- Aprende cómo funcionan los sistemas de IA para anticipar de forma eficiente y evitar los riesgos comunes, sesgos y problemas éticos.
- Sigue trabajando con tus compañeros para desarrollar tanto habilidades técnicas como habilidades transversales.
- Utiliza únicamente contenido generado por IA que entiendas completamente y del cual puedas responsabilizarte.

Reglas para estudiantes:

- Debes tomarte el tiempo necesario para explorar las herramientas de IA y comprender cómo funcionan, para poder utilizarlas de manera ética y reducir los sesgos potenciales.
- Debes reflexionar sobre tu problema antes de dar instrucciones a la IA. Esto te ayuda a escribir preguntas, instrucciones o conjuntos de datos más claros, detalladas y relevantes utilizando un vocabulario preciso.

- Debes desarrollar el hábito de revisar, cuestionar y probar sistemáticamente cualquier contenido generado por la IA.
- Debes buscar siempre la revisión de otras personas, no te limites a confiar en tu propia validación.

Resultados de esta etapa:

- Desarrollar habilidades de prompting tanto generales como de ámbito especifico.
- Aumentar tu productividad con un uso eficaz de las herramientas de IA.
- Seguir fortaleciendo el pensamiento computacional, la resolución de problemas, la adaptabilidad y la colaboración.

Comentarios y ejemplos:

- Ten en cuenta que la IA puede no tener la respuesta correcta porque esa respuesta no esté ni siquiera en Internet. Además, si te da soluciones incorrectas, intenta no insistir y busca ayuda entre las personas que te rodean. Vas a ahorrarte tiempo y vas a sumar en compresión.
- Vas a enfretarte con frecuencia a situaciones (como exámenes o evaluaciones) donde debes demostrar una comprensión real. Prepárate, sigue construyendo tanto tus habilidades técnicas como transversales.
- Explicar tu razonamiento y debatir con otras personas suele revelar lagunas en tu comprensión de un concepto. Prioriza el aprendizaje entre pares.
- Lo normal es que la herramienta de IA que utilices no conozca tu contexto específico (a menos que se lo indiques), así que te dará respuestas genéricas. Si buscas información más adecuada y más precisa en relación a tu entorno cercano, confía en el resto de estudiantes.
- Donde la IA tiende a generar la respuesta más probable, el resto de estudiantes puede proporcionar perspectivas alternativas y matices valiosos. Confía en la comunidad de 42 como un punto de control de calidad.

✓ Buenas prácticas:

> Le pregunto a la IA: "¿Cómo pruebo una función de ordenación?"Me da algunas ideas. Las pruebo y reviso los resultados con otra persona. Refinamos el enfoque de manera conjunta.

✗ Bad practice:

> Le pido a la IA que escriba una función completa, la copio y la pego en mi proyecto. Durante la evaluación entre pares, no puedo explicar qué hace ni por qué. Pierdo credibilidad. Suspendo mi proyecto.

✓ Good practice:

Utilizo la IA para ayudarme a diseñar un parser. Luego, reviso la lógica con otra persona. Encontramos dos errores y lo reescribimos juntos: mejor, más limpio y comprendiendo al 100

✗ Bad practice:

Dejo que Copilot genere mi código para una parte clave de mi proyecto. Compila, pero no puedo explicar cómo maneja los pipes. Durante la evaluación, no puedo justificarlo y suspendo mi proyecto.

Capítulo IV

Parte obligatoria

Nombre de programa	`minishell`
Archivos a entregar	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, flean, re`
Argumentos	
Funciones autorizadas	`readline, rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay, add_history, printf, malloc, free, write, access, open, read, close, fork, wait, waitpid, wait3, wait4, signal, sigaction, sigemptyset, sigaddset, kill, exit, getcwd, chdir, stat, lstat, fstat, unlink, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, perror, isatty, ttyname, ttyslot, ioctl, getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs`
Se permite usar libft	`Sí`
Descripción	`Escribe un shell`

Tu shell deberá:

- Mostrar **una entrada** mientras espera un comando nuevo.
- Tener un **historial funcional**.
- Buscar y ejecutar el ejecutable correcto (basado en la variable `PATH` o mediante el uso de rutas relativas o absolutas).
- Usa como máximo **una variable global** para indicar la recepción de una señal. Piensa en lo que implica: Esta aproximación evita que tu gestor de señales acceda a tus estructuras de datos principales.

¡Cuidado! Esta variable global no puede proporcionar ninguna información o dato más allá del número de la señal recibida. Por lo tanto, está prohibido utilizar estructuras de tipo "norm" como variables globales.

- No interpretar comillas sin cerrar o caracteres especiales no especificados en el enunciado como \ (barra invertida) o ; (punto y coma).
- Gestionar que la ' evite que el shell interprete los metacaracteres en la secuencia entrecomillada.
- Gestionar que la " evite que el shell interprete los metacaracteres en la secuencia entrecomillada exceptuando $ (signo de dólar).
- Implementar **redirecciones**:
  - < debe redirigir la entrada (input).
  - > debe redirigir la salida (output).
  - "<<" debe recibir un delimitador, después leer del input de la fuente actual hasta que una línea que contenga solo el delimitador aparezca. Sin embargo, no necesita actualizar el historial.
  - ">>" debe redirigir la salida (output) en modo de adición (append).
- Implementar **tuberías (pipes)** (carácter |). El salida (output) de cada comando en la cadena de comandos (pipeline) se conecta mediante una tubería (pipe) a la entrada (input) del siguiente comando.
- Gestionar las **variables de entorno** ($ seguidos de caracteres) que deberán expandirse a sus valores.
- Gestionar $?, que deberá expandirse al estado de salida del comando más reciente ejecutado en la pipeline.
- Gestionar ctrl-C ctrl-D ctrl-\, que deberán funcionar como en bash.
- Cuando sea interactivo:
  - ctrl-C imprime una nueva entrada en una línea nueva.
  - ctrl-D termina el shell.
  - ctrl-\ no hace nada.
- Deberá implementar los built-ins:
  - echo con la opción -n.
  - cd solo con una ruta relativa o absoluta.
  - pwd sin opciones.

- `export` sin opciones.
- `unset` sin opciones.
- `env` sin opciones o argumentos.
- `exit` sin opciones.

La función `readline()` puede producir algunos leaks que no necesitas arreglar. Eso **no significa que tu código, sí, el código que has escrito, pueda producir leaks**.

Limítate a hacer lo que pide el enunciado. Cualquier cosa no solicitada no se requiere.
Para cada punto, y en caso de dudas, puedes utilizar bash como una referencia.

Capítulo V

Parte extra

Tu programa deberá implementar los siguientes puntos:

- `&&` y `||` con paréntesis para prioridades.
- Los wildcards `*` deben funcionar para el directorio actual.

Los bonus solo serán evaluados si tu parte obligatoria está PERFECTA. Con PERFECTA queremos naturalmente decir que debe estar completa, sin fallos incluso en el más absurdo de los casos o de mal uso, etc. Significa que si tu parte obligatoria no tiene TODOS los puntos durante la evaluación, tus bonus serán completamente IGNORADOS.

Capítulo VI

Entrega y evaluación

Entrega tu proyecto en tu repositorio `Git` como de costumbre. Solo el trabajo entregado en el repositorio será evaluado durante la defensa. No dudes en comprobar varias veces los nombres de los archivos para verificar que sean correctos.

Durante la evaluación, es posible que se solicite una breve **modificación del proyecto**. Esto puede consistir en ajustar ligeramente el comportamiento, modificar unas cuantas líneas de código o incorporar una característica fácil de implementar.

Puede que este paso **no sea necesario en todos los proyectos**, pero debes tenerlo en cuenta si asi se especifica en la hoja de evaluación.

Este paso sirve para a verificar tu comprensión real de una parte específica del proyecto. La modificación se puede realizar en cualquier entorno de desarrollo que elijas (por ejemplo, tu configuración habitual), y debería ser factible en unos pocos minutos, a menos que se defina un plazo específico como parte de la evaluación.
Por ejemplo, se te puede pedir hacer una pequeña actualización en una función o script, modificar lo que se vería en pantalla o ajustar una estructura de datos para almacenar nueva información, etc.

Los detalles (alcance, objetivo, etc.) se especificarán cada **hoja de evaluación** y pueden variar de una evaluación a otra para el mismo proyecto.