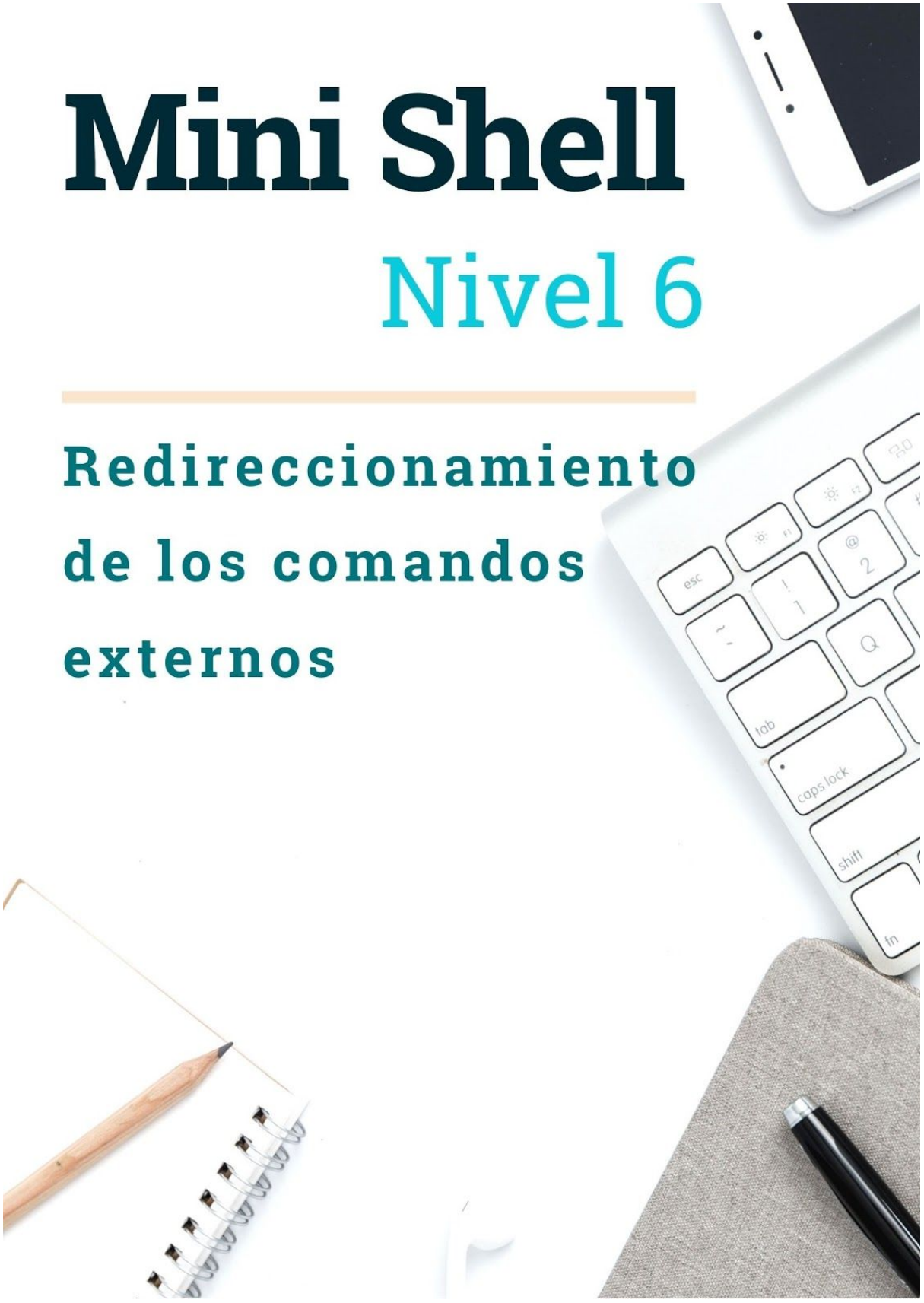
Mini Shell

Nivel 6

Redireccionamiento de los comandos externos



Indice

La salida de cualquiera de los comandos externos ejecutados desde nuestro mini shell tiene que poder ser direccionada a un archivo en vez de aparecer por pantalla.

¿Cuáles son los retos de este nivel?

En este nivel nuestro mini shell tendrá que permitir el redireccionamiento de la salida de un comando externo a un fichero externo, indicando "**>** *fichero*" al final de la línea. [1]

¿Qué conceptos trabajamos con estos retos?

- Flujos estándares
- Redirección. Funciones **dup**() y **dup2**()

¿Cuáles son los detalles de implementación?

Funciones que hay que crear/ampliar/modificar en este nivel:

int is_output_redirection (char **args); Función booleana que recorrerá la lista de argumentos buscando un token '>', seguido de otro token que será un nombre de fichero. En tal caso retornará TRUE (1), significando que se trata de un redireccionamiento, y substituirá el argumento '>' por **NULL** (recordemos que así lo requiere execvp()). En caso contrario devolverá FALSE (0) y no modificará los argumentos. Si se trata de un redireccionamiento (con la sintaxis correcta), se abrirá con **open**() el fichero de salida que se ha indicado en el comando, para obtener su descriptor de fichero. Mediante la llamada al sistema **dup**() o **dup2**() creamos un duplicado de ese descriptor empleando el del *stdout* y luego cerramos el fichero de salida.
int execute_line(char *line);

[1] Nosotros exigiremos que el símbolo '>' esté precedido y seguido de un espacio en blanco (el bash lo detecta y ejecuta aunque no tenga esos espacios, al igual que el '&').

Desde aquí el hijo llamará a la función anterior, *is_output_redirection()*, antes de la llamada a execvp(), para modificar, o no, los argumentos que se le pasarán.

¿Cómo comprobamos que hemos superado este nivel? [2]

- Podéis ejecutar vuestro programa **nivel6.c** e ir escribiendo en la línea de comandos los ejemplos que se exponen a continuación, comparando vuestros resultados con los mostrados aquí

```
uib ~$ ./nivel6
$ pwd > salida.txt
$ cat salida.txt
/home/uib/practicasSO/SOI/miP2/niveles
$ ls -l > listado.txt
$ cat listado.txt
total 6
-rwxrwxr-x 1 uib uib 13456 nov 16 13:19 nivel1
-rwxrwxr-x 1 uib uib 17016 nov 18 12:54 nivel2
-rwxrwxr-x 1 uib uib 14048 nov 10 12:56 nivel3
-rwxrwxr-x 1 uib uib 18056 nov 24 13:38 nivel4
-rwxrwxr-x 1 uib uib 14576 nov 21 11:20 nivel5
-rwxrwxr-x 1 uib uib 18896 nov 25 13:30 nivel6
$ cat > comandos.txt # acabar con Ctrl+D
date
ps
sleep 1
pwd
$ source comandos.txt
vie dic  1 14:17:41 CET 2017
  PID TTY          TIME CMD
24772 pts/18   00:00:00 bash
24789 pts/18   00:00:00 nivel6
24849 pts/18   00:00:00 ps
/home/uib/practicasSO/SOI/miP2/niveles
$
```

[2] En este nivel ya podéis dejar de mostrar los comentarios de las etapas anteriores que aparecían entre [ ] (buscar las ocurrencias del símbolo →)

Anexo 1: Flujos estándares

En UNIX, los procesos se comunican con el exterior a través de **flujos** (*streams*). Conceptualmente, un flujo es una ristra de *bytes* que se puede ir leyendo o sobre la que se puede escribir caracteres. Un flujo puede ser un fichero ordinario, o estar asociado a un dispositivo. Cuando se lee del teclado es porque previamente se ha abierto como flujo de caracteres del que leer. Un proceso, cuando muestra algo por pantalla, está escribiendo caracteres a un flujo de salida.
Los procesos lanzados por un *shell* utilizan dos flujos de particular interés: la **entrada estándar** y la **salida estándar**. La entrada estándar es utilizada para recoger información, y la salida estándar para enviar información al exterior.
La entrada estándar suele ser el teclado. La salida estándar suele ser la pantalla.
Los flujos estándares pueden ser redirigidos a otros ficheros.
Existe también el llamado **error estándar**, flujo donde se vierten los mensajes de error. Habitualmente coincide con la salida estándar, pero se considera un flujo diferente.
Descriptores de fichero:
Cuando se abre un fichero con **open()** se devuelve un **descriptor de fichero** (*file descriptor*), que es un número entero que se empleará en posteriores llamadas a las funciones de manejo de ficheros como **read()** o **write()**.
En UNIX la entrada estándar tiene el descriptor de valor 0, la salida estándar el de valor 1 y la salida de error estándar el de valor 2.

Nombre	Descriptor de fichero	Destino por defecto
entrada estándar (*stdin)*	0	teclado
salida estándar (*stdout*)	1	pantalla
error estándar (*stderr*)	2	pantalla

Anexo 2: Redirección. Llamadas al sistema dup() y dup2()

Para redirigir la entrada o la salida hay que conseguir abrir el fichero de redirección de forma que el sistema le asigne el descriptor 0 ó 1 respectivamente. Teniendo en cuenta que el algoritmo de la llamada **open** reserva el primer descriptor disponible,

empezando a explorar desde el número cero, esto significa que si cerramos la salida estándar (descriptor número 1), el siguiente **open** que se realice asignará al fichero abierto el descriptor 1, quedando automáticamente reasignada la salida estándar. A veces conviene preservar el fichero de entrada o salida estándar actual, para restituirlo en el futuro. Para ello se emplea la llamada **dup**(). También disponemos de la llamada al sistema **dup2**(), con la cual se indica qué descritor de fichero pasa a ser un duplicado de un descriptor abierto.

Ejemplo de uso de las funciones dup y dup2:

```c
/* uso_dup.c - ejemplo de uso de dup() y dup2() - Adelaida Delgado */
// Use: ./uso_dup nombre_fichero comando
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>

int main (int argc, char *args[])
{
  int fd;
  if (argc < 3) {
    printf ("Sintaxis: %s fichero comando\n", args[0]);
    exit (1);
  }

  fd = open (args[1],  O_WRONLY|O_CREAT|O_TRUNC, S_IRUSR|S_IWUSR);
  dup2 (fd, 1);   /* El descriptor 1, de la salida estándar, pasa a ser un duplicado de fd */

  /*También hubiéramos podido hacer:
    close(1); // al cerrar stdout el descriptor 1 pasa a ser el primero libre
    dup(fd);  // crea una copia del descriptor de ficheros fd,
              // utilizando el descriptor de archivo no utilizado
              // con el número más bajo para el nuevo descriptor, o sea 1 (stdout)
  */

  close (fd);
  execvp (args[2], &args[2]); /* Ejecuta comando */
  exit (1);
}
```

Ejemplo de ejecución:

```
uib ~$ ./uso_dup ps.txt ps
uib ~$ cat ps.txt
  PID TTY          TIME CMD
 2454 pts/0    00:00:00 bash
 4997 pts/0    00:00:00 ps
```

Observaciones

- En el ejemplo anterior se pasan los argumentos del programa a través de la línea de comandos. Ver: https://www.tutorialspoint.com/cprogramming/c_command_line_arguments.htm