

# Boletín 2: Redirección, *prompt* y comandos internos

## Ampliación de Sistemas Operativos

Dpto. Ingeniería y Tecnología de Computadores (DITEC)

Universidad de Murcia

Curso 2018/2019

1 Redirecciones en `simplesh`

2 *Prompt* en `simplesh`

3 Comandos internos

# 1. Redirecciones en simplesh

# Redirecciones en simplesh

- Cuando se redirige la salida a un fichero, sus permisos no permiten leer su contenido:

```
1  simplesh> echo linea1 > listado
   simplesh> cat listado
3  cat: listado: Permiso denegado
   simplesh> chmod u+r listado
5  simplesh> cat listado
   linea1
```

- Además, el comportamiento de >> no es el esperado:

```
   simplesh> echo linea1 > listado
2  simplesh> echo linea2 >> listado
   simplesh> cat listado
4  linea2
```

- Y el de > tampoco:

```
   simplesh> echo linea1 > listado
2  simplesh> echo linea2 >> listado
   simplesh> echo linea3 > listado
4  simplesh> cat listado
   linea3
6  linea2
```

- (0.25p) EJERCICIO 1: Modifica `simplesh` para que los ficheros se creen con permisos 700 y la semántica de las redirecciones sea la misma que tendrían si se ejecutasen los comandos en `bash`
- Para realizar el ejercicio, utiliza los *flags* y el *mode* de `open()`

## 2. *Prompt* en simplesh

# El símbolo del sistema o *prompt* de simplesh

- La llamada `readline` muestra el *prompt* y, a continuación, devuelve la línea de comandos introducida por el usuario

```
char* get_cmd()  
2 {  
    char* buf= readline("simplesh> ");  
4  
    if (buf)  
6        add_history(buf);  
8    return buf;  
}
```

- Por ejemplo:

```
1  simplesh> ls > listado # 'buf' apunta a "ls > listado"
```

# El símbolo del sistema o *prompt* de `simplesh`

- (0.25p) EJERCICIO 2: Modifica el *prompt* para que muestre el usuario y el directorio de trabajo actual separados por el carácter @:

```
1  usuario@directorio>_...
```

- Para ello, únicamente se pueden usar llamadas POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas POSIX y/o `glibc` a considerar:
  - `getuid()`
  - `getpwuid()`
  - `getcwd()` (Versión POSIX sin extensiones GNU)<sup>1</sup>
  - `basename()` (Versión POSIX)
  - `sprintf()`

---

<sup>1</sup>Para saber el tamaño máximo que puede tener la ruta absoluta al directorio de trabajo actual, véase `man 3 getcwd`.



### 3. Comandos internos

# Comandos internos versus comandos externos

- Los comandos internos son proporcionados por el propio *shell*
  - `help cd` / `man cd` / `which cd`
- Los comandos externos son programas independientes del *shell*
  - `help ls` / `man ls` / `which ls`
- Algunos comandos son a la vez internos y externos
  - `help pwd` / `man pwd` / `which pwd`
- Comandos como `cd` o `exit` sólo pueden ser internos, ¿por qué?<sup>2</sup>

---

<sup>2</sup>Véase `man 2 chdir` y `man 2 exit`.

- (0.5p) EJERCICIO 3: Implementa el comando interno `cwd`:

```
1  simplesh> cwd  
   cwd: /ruta/a/directorio
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas POSIX y/o glibc a considerar:
  - `getcwd()` (Versión POSIX sin extensiones GNU)
- Para implementar `cwd`, escribe la función `run_cwd()` e inserta llamadas a la misma en `simplesh.c` donde sea necesario

# El comando interno exit

- (1.0p) EJERCICIO 4: Implementa el comando interno exit:

```
2    simplesh> exit  
    [bash]
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas POSIX y/o glibc a considerar: exit
- Para implementar exit, escribe la función run\_exit() e inserta llamadas a la misma en simplesh.c donde sea necesario
- Nótese que exit sólo termina el proceso que ejecuta la llamada
- Verifica que simplesh finaliza siempre que encuentra el comando interno exit (nótese que el segundo echo no se ejecuta)

```
2    simplesh> echo 1 ; exit ; echo 2  
    1  
    [bash]
```

- Comprueba que tras ejecutar exit se libera **toda la memoria**

# El comando interno cd

- (0,5p) EJERCICIO 5: Implementa el comando interno cd dir:

```
1  simplesh> cwd
   cwd: /home/usuario
3  simplesh> cd directorio
   simplesh> cwd
5  cwd: /home/usuario/directorio
```

- Para ello, únicamente se pueden usar llamadas al sistema POSIX o funciones de la biblioteca estándar de C (C11)
- Llamadas POSIX y/o glibc a considerar: `chdir()`
- Para implementar `cd`, escribe la función `run_cd()` e inserta llamadas a la misma en `simplesh.c` donde sea necesario
- Nótese que `chdir()` sólo cambia el directorio actual del proceso que ejecuta la llamada
- Verifica que `simplesh` modifica el directorio actual siempre que encuentra el comando interno `cd`

```
1  simplesh> cwd ; cd directorio ; cwd ; cd .. ; cwd
   cwd: /home/usuario
3  cwd: /home/usuario/directorio
   cwd: /home/usuario
```

# El comando interno cd

- (0.5p) **Opcional:** Implementa `cd [-]`:

```
1  simplesh> cd -
2  run_cd: Variable OLDPWD no definida
3  simplesh> cwd
4  cwd: /home/usuario/directorio
5  simplesh> cd
6  simplesh> cwd
7  cwd: /home/usuario
8  simplesh> cd -
9  simplesh> cwd
10 cwd: /home/usuario/directorio
```

- El directorio por defecto es el valor de la variable de entorno `HOME`, mientras que `OLDPWD` contiene el directorio de trabajo previo<sup>3</sup>
- Llamadas POSIX y/o glibc a considerar: `getcwd()`, `getenv()`, `setenv()` y `unsetenv()`
- Verifica que `simplesh` modifica el directorio actual siempre que encuentra el comando interno `cd`

```
1  simplesh> cwd ; cd ; cwd ; cd - ; cwd
2  cwd: /home/usuario/directorio
3  cwd: /home/usuario
4  cwd: /home/usuario/directorio
```

---

<sup>3</sup> Cuando se inicia `simplesh`, se debe eliminar la variable de entorno `OLDPWD`. Sólo tras la ejecución con éxito del primer comando interno `cd`, se definirá la variable `OLDPWD` con el directorio de trabajo inicial. A partir de entonces, la variable `OLDPWD` se actualizará cada vez que se ejecute con éxito el comando interno `cd`, asignándole el directorio de trabajo previo.

- 1 Escribe la función `run_cwd()` **respetando fielmente** el enunciado
- 2 Verifica que la función `run_cwd()` funciona correctamente
- 3 Escribe una función que dado un comando, determine si es interno
- 4 Usa la función anterior para saber cuando llamar a `run_cwd()`
- 5 Analiza cada llamada a la función `exec_cmd()` dentro de `run_cmd()` para saber dónde hay que llamar a `run_cwd()`, para ejecutar el comando interno `cwd`, en lugar de llamar a `exec_cmd()`
- 6 Escribe e inserta llamadas a las funciones `run_exit()` y `run_cd()` siguiendo la misma estrategia que con `run_cwd()`
- 7 Asegúrate de que los tres comandos internos del boletín exhiben el comportamiento esperado con `./simplesh.py -i boletin2.json`
- 8 Comprueba que no hay fugas de memoria con `./valgrind.sh`

- El comportamiento de `simplesh` cuando se ejecuta cualquiera de los comandos internos tiene que ser similar al de `bash` en estos casos:

```
1  simplesh> echo 1; exit; echo 2
2  1
3  [bash]
4  # El comando "echo 2" no se ejecuta.

5
6  simplesh> echo 1 | exit
7  simplesh>
8  # El comando "echo 1" se ejecuta pero su salida es ignorada por "exit".
9  # El comando "exit" se ejecuta pero simplesh no finaliza su ejecución.
10
11 simplesh> exit &
12 simplesh>

13
14 simplesh> (exit)
15 simplesh>
16 # El comando "exit" se ejecuta pero simplesh no finaliza su ejecución.

17
18 simplesh> exit > salida
19 [bash]
20 # La salida de "exit" se redirige al fichero salida que se crea con tamaño cero.
    # Se requiere el uso de llamadas para manipulación de descriptores de fichero.
```