

# Usando señales de Unix con Perl



por  
Matias Palomec  
<matias@programado.org>



# Introducción

- Las señales son **eventos que envía el sistema operativo** (u otros programas) informando de "**eventos de importancia**". Por su naturaleza, suelen **interrumpir o modificar** la ejecución normal del programa, forzando al programa a **manejar las señales inmediatamente**.

•

- Cada señal puede tener un **manejador propio** dentro del programa, que es una función que es llamada cuando el programa recibe la señal indicada. La función es llamada de una forma anacrónica, es decir que no hace falta tener en ninguna parte del programa que llame directamente a esta función (salvo en el manejador). Cuando la señal es enviada al proceso, el sistema operativo **para la ejecución** del proceso y "fuerza" a que se ejecute el manejador de la señal. Cuando el manejador de la señal termina, el **proceso continúa** la ejecución en donde se había quedado, como si no hubiese pasado nada.

# listando las señales

- Cada señal es un número y tiene un nombre asociado (simbólico) en `/usr/include/signal.h` (depende del **\*nix** la locación del archivo). Se pueden ver con `"kill -l"`

# ejemplo

- Entendiendo el:
  - `Control + C`
    -
- No todos los programas deben terminar con el `Ctrl + C`, por eso se los puede “atrapar” y ejecutar una acción arbitraria

# perl

```
#!/usr/bin/perl -w

$SIG{INT} = sub {
    print "Eps, porque el ^C\n";
};

print "Antes del sleep\n";
sleep 60;
print "Luego del sleep\n";

__END__
```

# la salida

```
matias@homero:~/perl$ ./uno.pl
```

```
Antes del sleep
```

*<se presiona el control + C>*

```
Eps, porque el ^C
```

```
Luego del sleep
```

```
matias@homero:~/perl$
```

# analogías

- A continuación otros lenguajes de programación

# ruby

```
#!/usr/bin/ruby -w

trap("INT") {
  print "Eps, porque el ^C\n"
}

print "Antes del sleep\n"
sleep 60
print "Luego del sleep\n"

__END__
```

# la salida

```
matias@homero:~/perl$ ./ruby_^C.rb  
Antes del sleep
```

*<se presiona el control + C>*

```
Eps, porque el ^C
```

*<se presiona el control + C>*

```
Eps, porque el ^C
```

```
Luego del sleep
```

```
matias@homero:~/perl
```

# C

```
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

void trapeo(int sig_num) {
    printf("Eps, porque el ^C\n");
}

int main( void ) {
    signal(SIGINT, trapeo);

    printf("Antes del sleep\n");
    sleep(60);
    printf("Luego del sleep\n");
}
#EOF#
```

# la salida

```
matias@homero:~/perl$ ./c^C
```

```
Antes del sleep
```

*<se presiona el control + c>*

```
Eps, porque el ^C
```

```
Luego del sleep
```

```
matias@homero:~/perl$
```

# bash

```
#!/bin/bash
```

```
trap 'echo "Eps, porque el ^C"' 2
```

```
echo "Antes del sleep"
```

```
sleep 60
```

```
echo "Luego del sleep"
```

# la salida

```
matias@carosaII:~/perl$ ./bash_^C.pl
```

```
Antes del sleep
```

*<se presiona el control + c>*

```
Eps, porque el ^C
```

```
Luego del sleep
```

```
matias@carosaII:~/perl$
```

# manejando timeouts

- Las entradas de texto suelen tener problemas (entre el teclado y el respaldo del asiento), por eso suelen dar *timeouts* arbitrarios. Para no tener que esperar *in-eternum* se puede generar código que termine (o continúe) si el usuario no ingresa nada.

```
sub ginfo {
  my $b = "";
  eval {
    local $SIG{ALRM} = sub {
      alarm 0;
      print "ufff, me canse\n" && die;
    };
    alarm 2;
    print "Ingrese texto: ";
    $b = <STDIN>;
  };
  return "nada ingresado" if($@);
  chop($b);return $b;
}
```

```
print "tengo: \" . ginfo() . "\"\n";
```

# explicación

- Dentro del eval, al cumplirse el timeout (**alarm 2**), se ejecuta la subrutina guardada en el manejador **\$SIG{"ALRM"}** que resetea la alarma, imprime un mensaje por pantalla y por último ejecuta el die (aborta la ejecución).
- El eval atrapa el **die** del manejador de la señal y carga la variable de entorno de error de sintaxis (ver perlvar)

# explicación en detalle

- [...]
- `read(3, "#!/usr/bin/perl -w\n\nsub ginfo {\n"..., 4096) = 288`
- `fcntl64(3, F_SETFD, FD_CLOEXEC) = 0`
- `brk(0x8190000)`
- `= 0x8190000`
- 
- `# -> Se lee el código`

•

# continuación

- `rt_sigaction(SIGALRM, NULL, {SIG_DFL}, 8) = 0`
- `rt_sigprocmask(SIG_BLOCK, [ALRM], [], 8) = 0`
- `rt_sigaction(SIGALRM, {SIG_DFL}, {SIG_DFL}, 8) = 0`
- `rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0`
- `rt_sigprocmask(SIG_BLOCK, [ALRM], [], 8) = 0`
- `rt_sigaction(SIGALRM, {0x80afde0, [], 0}, {SIG_DFL}, 8) = 0`
- `rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0`
- 
- # → La alarma global queda con la opción por defecto (SIG\_DFL), pero la alarma del bloque (ALRM) se asigna de otra forma.

•

# continuación

- 
- alarm(2) = 0
- 
- # -> Se activa la alarma
- 
- write(1, "Ingrese texto: ", 15Ingrese texto:) = 15
- read(0, 0x8170d40,4096) = ? ERESTARTSYS (To be restarted)
- --- SIGALRM (Alarm clock) @ 0 (0) ---
- 
- # -> Se pregunta al usuario por el ingreso de texto pero luego se recibe la alarma.

# continuación

- sigreturn() = ? (mask now [])
- rt\_sigprocmask(SIG\_BLOCK, [ALRM], NULL, 8) = 0
- alarm(0) = 0
- 
- # -> Se cancela el SIGALRM
- 
- write(1, "ufff, me canse\n", 15ufff, me canse
- ) = 15
- 
- # -> Se escribe el mensaje
-

- `rt_sigprocmask(SIG_UNBLOCK, [ALRM], NULL, 8) = 0`
- `rt_sigprocmask(SIG_BLOCK, [ALRM], [], 8) = 0`
- `rt_sigaction(SIGALRM, {SIG_DFL}, {0x80afde0, [], 0}, 8) = 0`
- `rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0`
- 
- # -> Al terminar el bloque se elimina el trapeo del SIGALRM.
- 
- `write(1, "tengo: \"nada ingresado\"\n", 24tengo: "nada ingresado") = 24`
- `close(3)`
- `= 0`
- `exit_group(0)`
- `= ?`
- 
- 
- # -> Termina el archivo.

# usos y costumbres

- Una de las señales más importantes para ser “bien” programada es el:
  - **SIGHUP**
    - 
    - Esta señal debería “**reiniciar**” internamente el proceso, es decir, cerrar todos los archivos abiertos, **releer la configuración**, y cargar todo desde cero.

# usos y costumbres

- **SIGINT**: Es el “Ctrl + C”, se interrumpe la ejecución del programa
- **SIGABRT**: Es el “Ctrl + \”, parecido al INT, pero no tan ignorado (el INT se lo suele ignorar)
- **SIGALRM**: Es una alarma interna que se setea con el alarm (o via programas externos).

# usos y costumbres

- **SIGTERM**: Indica al programa terminar todo (cerrando todos los archivos de forma ordenada)
- **SIGSEGV**: Violación de segmento (causa que el programa aborte con ese error de memoria)

-

# usos y costumbres

- **SIGUSR1** y **SIGUSR2**: Señales definidas por el usuario (son dos en todos los \*nix)
  - **SIGCHLD**: El status del proceso hijo cambió (terminó el proceso)
  - **SIGSTP**: stop (inevitable/inbloqueable)
  - **SIGSTOP**: el “Ctrl+Z” del teclado (stop)

# usos y costumbres

- **SIGCONT**: Continuar con la ejecución (saca el stop del SIGTSTP y SIGSTOP)
- **SIGXCPU**: Se excedió el límite de CPU
- **SIGXFSZ**: Se excedió el límite de tamaño para un archivo

# recomendación de usuario

- **SIGHUP**

- (reconfigurar todo sin tener que reiniciar)

- **SIGTERM**

- (guardar los archivos ordenadamente, indicando que se cerró bien)

- **SIGINT**

- (realizar acciones previas a interrumpir algo por la mitad)

# Grupo de Perl de Capital Federal

Sitio web:

<http://cafe.pm.org/>

Lista de correo:

<http://mail.pm.org/mailman/listinfo/cafe-pm>

Próxima reunión:

Baviera (Juramento y Ciudad de la Paz)

Viernes 6 de Julio 19:08

url de la presentación:  
[http://programado.org/?q=charla\\_20070630](http://programado.org/?q=charla_20070630)



licencia de la charla:  
<http://creativecommons.org/licenses/by-nc-sa/2.5/ar/>

