

Uso de PHP en scripts de administración

Operación Comando

La mayoría de los administradores suelen usar la shell, Perl o Python si necesitan programar un script de administración. Pero para los programadores de la web no es necesario aprender otros lenguajes para escribir una tarea de rutina. PHP proporciona a los administradores la potencia para programar herramientas de línea de comandos e incluso interfaces web completas. **POR CARSEN MÖHRKE**



Los scripts escritos en PHP para la consola tienen básicamente la misma forma que scripts escritos para la web: los programadores pueden usar todo el rango de características de PHP. Para permitir de forma sencilla, que los programas se ejecuten en la línea de comandos, se necesita especificar el intérprete de PHP en la primera línea del fichero de scripts de la siguiente forma `#!/`. Si no se conoce el path al intérprete, tecleando `which php` se averiguará. También, los ficheros de scripts necesitan tener activado el permiso de ejecución. Echémosle un vistazo al típico ejemplo "Hello, World" en PHP:

```
#!/usr/bin/php -q
<?php
    echo "Hello, World\n";
?>
```

Las herramientas de PHP es lo que se necesita para cualquier clase de manipulación de texto, como modificación de ficheros de

configuración, ficheros de registro, entradas de gestión de usuarios y cosas así. Si se necesita acceder a las funciones de sistema operativo, para tareas como administración de procesos, se necesitará añadir algunos comandos Linux.

Línea de Comandos

PHP tiene cuatro funciones diseñadas para la ejecución de comandos del sistema: `exec()`, `system()`, `passthru()` y `shell_exec()`:

*`exec()` ejecuta el comando que se le pasa como primer argumento y elimina cualquier salida del comando. La última línea del comando se devuelve como resultado. Si el segundo parámetro es el nombre de un array, `exec()` rellena el array línea a línea con la salida que debería aparecer en la pantalla. Si se necesita procesar el código de salida del comando `exec()` se puede usar un tercer parámetro.

*`system()` no suprime la salida al contrario que `exec()`. Tiene un segundo parámetro opcional para el código de salida.

*`passthru()` funciona como `system()`, pero en modo binario compatible. Esto permite al comando pasar imágenes generadas como salida convertidas para un navegador web.

*`shell_exec()` funciona como una versión simple de parámetros de `exec()`, pero en vez de extraer la última línea, devuelve una cadena con la salida completa del comando. La cadena contiene caracteres de nueva línea en contraposición a las entradas que devolvía el comando `exec()` en el array de valores.

Tanto `exec()` como `shell_exec()` son útiles en casos donde se necesite manipular el texto de salida de un comando externo. Los ejemplos que vienen a continuación hacen uso de la función `exec()`, que es mucho más flexible que las otras funciones. Nos encontramos con un pequeño fallo técnico relacionado con esto en la actualidad: PHP no siempre devuelve el código de salida correcto del comando externo (por ejemplo cuando concatenamos comandos).

Interfaces para la Shell Antiguos y Modernos

Las versiones de PHP 4.3.0 o posteriores tienen un nuevo interfaz para la shell para programar herramientas desde la línea de comandos: el CLI (Command Line Interface). Las versiones previas de PHP (incluso algunas nuevas) tienen un interfaz de programación llamado Server Application Programming Interface (SAPI), pero SAPI fue diseñado para usarse en servidores web (CGI-SAPI) y añade una cabecera HTTP a la salida. (Se puede suprimir la cabecera colocando la opción `-q` cuando se lanza el intérprete de PHP, pero esto no eliminará las etiquetas HTML de los mensajes de error). Se puede comprobar si se tiene la versión CGI, que no tiene el mismo alcance de características que tiene la interfaz de línea de comandos, tecleando `php -v`. El ejemplo de este artículo funcionará con cualquier versión.

Control de Procesos

El siguiente script PHP usa `exec()` para mostrar el número de procesos que se ejecutan en el sistema:

```
unset ($out);
$cmd="ps axu | wc -l";
$erg=exec($cmd,$out);
$erg-=1; //minus 1 line header
echo "Number of active
processes is $erg\n";
```

El fichero primero elimina el contenido de la variable `$out`, que representa el array donde se almacenará la salida del comando. `exec()` no sobrescribiría el array sino que simplemente le añadiría al final la nueva salida. El ejemplo impide que se mezclen los resultados de múltiples scripts vaciando primero el array. El comando detecta el número de líneas en la lista de procesos y resta una línea de las cabeceras de columna.

`shell_exec()` y `exec()` tan sólo pueden manejar datos de `stdout`. Para capturar la salida de error estándar desde `stderr`, se necesita redireccionar la salida de error a `stdout`: `2 >&1`.

Los programadores a menudo necesitan pasar argumentos de la línea de comandos al script. Para permitir que esto pueda hacerse, PHP introduce los argumentos que siguen al nombre del fichero en la línea de comandos dentro de un array llamado `$argv`, conservando el orden original. Además, el número de argumentos se almacena en la variable `$argc`.

Buenos Argumentos

PHP no soporta comandos que lean datos directamente desde el teclado. Para proporcionar soporte a las entradas de teclado, los programadores pueden abrir el flujo `stdin` como un fichero y leer desde él. Esto permite el uso de los flujos `stdout` y `stderr` relacionados:

```
$in=fopen("php://stdin","r");
$error=fopen("php://stdin","w");
$input=trim(fgets($fp, 100));
while ("=="$input)
{
    fputs($error,"Please input a ↵
    value\n");
}
```

Si se usa `stdin` para leer las entradas de teclado de esta forma, el script debería usar la función `trim()` para eliminar los espacios en blanco al final de la cadena. No hay necesidad de usar `stdout` para la salida de datos, ya que PHP cuenta con las funciones `echo` y `print` para realizar estas tareas. La salida de ambos comandos pueden ser redireccionadas.

Coloreado

Para diseñar aplicaciones más cómodas, los programadores necesitan funciones que borren la pantalla, que controlen el

Tabla 1: Secuencias de Escape importantes

Significado	Secuencia
Borrar Pantalla	<code>\033[2J</code>
Posicionar el cursor en x,y	<code>\033[x;yH</code>
Coloca el cursor n posiciones a la izquierda	<code>\033[nC</code>
Coloca el cursor n posiciones arriba	<code>\033[nA</code>
Color de Fuente rojo	<code>\033[0;31m</code>
Color de Fuente negro	<code>\033[0;30m</code>
Color de Fuente verde	<code>\033[0;32m</code>
Color de Fuente gris	<code>\033[0;37m</code>

cursor y que añadan color a la salida. Las secuencias ANSI [1] proporcionan una solución útil, ya que la shell es capaz de interpretarlas. Estas secuencias empiezan con el carácter Escape y son conocidas como secuencias de Escape. Por ejemplo, la secuencia `echo "\033[2J"` limpia la pantalla. La Tabla 1 lista unas cuantas de las secuencias más importantes.

El color seleccionado permanece válido hasta que se selecciona un color diferente. Las funciones de PHP `ncurses` proporcionan mayor portabilidad y un rango mucho más amplio de características. Véase los ejemplos de las páginas web de PHP o Zend en [2], [3] para más detalle.

Gestión de Usuarios

El siguiente ejemplo muestra las funciones referidas a aquellas en el contexto de un script para la gestión de usuarios de un servidor con el fin de que se vean ejemplos prácticos. El Listado 1 es una muestra y el script completo está disponible para descargarse desde [4].

Nótese que el script abre la entrada estándar con `fopen()` pero no hace uso de `fclose()` para cerrarla. Ya que PHP automáticamente cierra todos los flujos abiertos cuando finaliza el programa, el script evita cerrar el flujo por razones de legibilidad. El tiempo de ejecución para los programas de la línea de comandos no está restringido.

PHP es ideal para crear interfaces web. Esta atractiva opción tiene algunas pegadas ocultas. Cuando se diseña una aplicación se necesita considerar los riesgos potenciales de seguridad. El más pequeño de los agujeros de seguridad podría tener consecuencias fatales. Un ejemplo:

```
01 <form method="POST">
02 Command: <input name=
"cmd" /><br />
03 <input type=
"submit" value=
"Execute" />
04 </form>
05 <?php
06 if (isset ($_POST['cmd']))
07 {
08 $outp=shell_exec
09 ($_POST[cmd]);
10 echo "<pre>$outp</pre>";
11 }
12 >>
```

Las soluciones que ejecutan comandos arbitrarios (como esta) están fuera de lugar. Los scripts de administración tan sólo deberían ser ejecutables dentro de un entorno seguro y no deberían permitir ninguna posibilidad de que sean manipulados.

Privilegios Especiales

Los scripts a menudo necesitan privilegios de root. Un script PHP basado en web normalmente se ejecuta con los privilegios del identificador de usuario del servidor web. Podría parecer obvio darle a este usuario (típicamente `www`, `wwwrun` o `nobody`) muchos más privilegios, pero esto es extremadamente peligroso. Comandos como `sudo` o `su` proporcionan una alternativa. Sin embargo, `su` requiere la práctica insegura de añadir la clave de root como texto en claro dentro del script.

La herramienta `sudo` también puede asignar privilegios de root para acciones específicas a un usuario sin la necesidad de que este usuario conozca la clave de root. En vez de ello, los usuarios se autentican con su propia clave y se les asignan los privilegios específicos por el root en el fichero `/etc/sudoers`. Si un administrador necesita asignar al usuario `webadmin` el derecho de terminar procesos con el comando `kill`, él o ella podría añadir la siguiente línea a `/etc/sudoers`:

```
webadmin ALL = /bin/kill, ↵
/usr/bin/killall
```

Es una buena idea usar el editor `visudo` para esto. El editor se asegura de que sólo un usuario puede editar el fichero

de configuración al mismo tiempo. Además se asegura de que las entradas concuerden con las reglas y no entren en conflicto.

La sintaxis genérica para asignar privilegios con sudo es: QUIEN DONDE = QUE. QUIEN puede ser un nombre de usuario, como en nuestro ejemplo, o un grupo de usuarios que deben ser definidos usando un *User_Alias*. DONDE significa el host donde el usuario tendrá estos privilegios.

Cambio de Identidad

Gracias a las entradas en el fichero de configuración de sudo, el usuario *webadmin* puede ahora “matar” procesos de forma arbitraria después de introducir su clave para la cuenta de usuario. Un script automatizado podría pararse y preguntar por la clave antes de continuar. Suele colocarse el parámetro *-S* para indicarle a sudo que lea la contraseña desde la entrada estándar. La línea de comandos sería como sigue: `echo clave | sudo -S kill 13221`

Si el usuario *webadmin* no es el usuario root del servidor web desde el punto de vista de los procesos de Unix (esto implicaría la herencia de los privilegios del usuario que probablemente no sería deseada en la mayoría de los casos), con toda seguridad se necesitará una combinación de *su* y *sudo*. El usuario del servidor web primero asumirá la identidad de *webadmin* temporalmente ejecutando *su* y entonces es cuando se le asignan los privilegios de root específicos por medio de *sudo*. Un script que usa esta solución puede parecerse a este:

```
$user="webadmin"; //sudoer
$pwd="clave"; ➤
// Webadmin password
$befehl="kill -9➤
".escapeshellarg($_GET
["pid"]);
$cmd_line="echo $pwd | ➤
su -l $user -c
\"echo $pwd | sudo -S ➤
$command 2>&1\"";
```

```
$ret=shell_exec($cmd_line);
```

El script primero usa *escape-shellargs()* para manipular el *pid*, que lo procesa desde un formulario. Esto elimina los ataques potenciales a la shell, que por otro lado se podrían insertar en el código de forma maliciosa. ■

RECURSOS

- [1] Códigos ANSI: http://en.wikipedia.org/wiki/ANSI_escape_code
- [2] Manual de Funciones Ncurses: <http://www.php.net/ncurses>
- [3] Tutorial Ncurses: <http://www.zend.com/zend/tut/tut-degan.php>
- [4] Listado: <http://www.linuxmagazine.com.es/Magazine/Downloads/03>

EL AUTOR

Carsten Möhrke es un consultor freelance y profesor, es el autor de “Better PHP Programming” y el CEO de Netviser. Puede ponerse en contacto con Carsten en cmoehrke@netviser.de.

Listado 1: Menú PHP

```
01 #!/usr/bin/php -q
02 <?php
03 function cls()
04 {
05 echo "\033[2J"; // Borrar
    Pantalla
06 echo "\033[0;0H"; // Cursor
07 to 0,0
08 }
09
10 function text_red()
11 {
12 echo "\033[0;31m";
13 }
14
15 function text_black()
16 {
17 echo "\033[0;30m";
18 }
19
20 // Funciones adicionales
21 // ¿Se ha insertado nombre de
    usuario?
22 $in=fopen("php://stdin","r");
23
24 $err=fopen("php://stderr","w")
    ;
24 if (3==$argc &&
25 "-u"==$_argv[1] &&
26 isset($_argv[2]))
27 {
28 $user=$_argv[2]; // leer
29 username
30 }
31 else
32 { // no hay nombre de usuario
33 fputs($err,"Por favor utilice
    -u para insertar un nombre de
    usuario\n");
34 exit(1);
35 }
36 // Comprobar existencia
    usuario
37 // con etc/passwd
38 if (false
    ===user_exists($user))
39 {
40 fputs($err,"Nombre de usuario
    no existe\n");
41 exit(2);
42 }
43
44 while (1) //Procesado infinito
45 loops
46 {
47 cls();
48 text_red();
49 echo "Administración para
    usuario $user\n";
50 text_black();
51 echo "1) Comprobar
    consistencia\n";
52 echo "2) Crear usuario MySQL
    DB\n";
53 // Más elementos del menú
54 echo "q) Salir Programa\n\n";
55 echo "Escoja una opción: ";
56 // Eliminar blancos e entrada
57
58 $input=trim(fgets($in,255));
59
60 switch ($input)
61 {
62 case "1":
63 consistency_check($in);
64 break;
65 // Más casos
66 case "q": exit(0);
67 // beep en caso de entrada
    incorrecta
68 default: echo chr(7);
69 }
70 }
71 ?>
```