PHP für Administrations-Skripte - ein Leitfaden

Seitensprung

PHP ist zwar mit Webprogrammierung verheiratet, lässt sich aber auch dazu bewegen, mal auf einer fremden Hochzeit zu tanzen. Eine neue Schnittstelle macht es nun einfacher, mit PHP Administrationswerkzeuge für die Kommandozeile zu schreiben. Carsten Möhrke

PHP für den Administrator? Das hört sich für viele sicherlich erst mal ungewohnt an, denn spontan denken sie eher an die Shell, an Perl oder Python, wenn ein Skript für die Systemverwaltung benötigt wird. Der Webprogrammierer muss aber nicht extra umsatteln, um eine kleine Routineaufgabe zu skripten: Auch mit PHP lassen sich Kommandozeilentools oder ganze Weboberflächen für den Admin schreiben.

Alte und neue Shell-Schnittstelle

Speziell für diesen Zweck bekam PHP ab der Version 4.3.0 eine neue Schnittstelle zur Shell spendiert, das CLI (Command Line Interface). Ein solches SAPI (Server Application Programmers Interface) gab es zwar bereits in den Vorgängerversionen, nur war es für den Einsatz auf Webservern ausgelegt (CGI-SAPI) und stellte deshalb allen Ausgaben einen HTTP-Header voran. Wer auf diese Version angewiesen ist – manche Distributoren lieferten zeitweilig auch neuere PHP-Versionen nur mit dem älteren CGI-SAPI aus –, kann den Header mit der Option »-q« beim PHP-Start unterdrücken.

Allerdings sind damit nicht gleichzeitig die HTML-Tags aus Fehlermeldungen verbannt. Ob lediglich die CGI-Version verfügbar ist, die zudem nicht ganz den Funktionsumfang des Kommandozeilen-Interface erreicht, lässt sich vorab mit »php -v« abfragen. Alle Beispiele dieses Beitrags funktionieren mit beiden Versionen.

PHP-Skripte für den Konsolen-Einsatz unterscheiden sich nicht grundsätzlich von denen, die im Web nutzbar sind: Es steht der gesamte Sprachumfang von PHP zur Verfügung. Damit die Skripte aber auf der Kommandozeile einfach unter ihrem Namen aufrufbar sind, muss der PHP-Interpreter in der ersten Zeile hinter »#!« angegeben werden. Dessen Pfad lässt sich im Bedarfsfall mit »which php« ermitteln. Zusätzlich muss die Skriptdatei Ausführungsrechte erhalten. Das berühmte Hello-Beispiel sähe damit in PHP so aus:

```
#!/usr/bin/php -q
<?php
   echo "Hallo Konsolenwelt\n";
?>
```

Zur Bearbeitung von Textdateien aller Art – zum Beispiel Konfigurations- oder Logfiles, Einträge für die Userverwaltung oder Ähnliches – reichen häufig die PHP-Bordmittel aus. Wo Funktionen des Betriebssystems Verwendung finden sollen, beispielsweise bei der Prozessverwaltung, sind allerdings Linux-Kommandos einzuspannen.

Zu Befehl!

Dafür hält PHP vier Funktionen mit unterschiedlichen Eigenschaften bereit: »exec()«, »system()«, »passthru()« und »shell_exec()«:

- »exec()« führt den Befehl aus, den es als ersten Parameter erhält, und unterdrückt dessen Ausgaben. Lediglich die letzte Zeile der Ausgabe wird als Ergebnis zurückgeliefert. Steht als zweiter Parameter der Name eines Array, füllt Exec() zusätzlich dieses Array zeilenweise mit allen Ausgaben, die normalerweise auf dem Bildschirm gelandet wären. Wer den Exit-Code von Exec() weiterverarbeiten will, erreicht das mit einem dritten Parameter.
- »system()« unterdrückt im Unterschied zu »exec()« die Befehlsausgabe nicht. Sie weist einem optionalen zweiten Parameter den Exit-Code zu.
- »passthru()« funktioniert wie »system()«, allerdings arbeitet die Funktion Binary-safe. Deshalb kann sie beispielsweise Bilder als Ausgabe eines Konverters direkt an einen Webbrowser durchreichen.
- »shell_exec()« arbeitet wie »exec()« in der Ein-Parameter-Version, gibt aber nicht nur die letzte Zeile, sondern die komplette Befehlsausgabe als String zurück. Dieser String enthält – im Unterschied zu den Einträgen im Rückgabe-Array von »exec()« – gegebenenfalls Zeilenschaltungen.

Die beiden Funktionen »exec()« und »shell_exec()« bieten sich besonders in jenen Fällen an, in denen die Textausgaben der externen Kommandos noch in irgendeiner Form zu verarbeiten sind. Die folgenden Beispiele benutzen »exec()«, das am flexibelsten ist. Zurzeit ergibt sich dabei noch ein kleines Problem: PHP liefert den Exit-Code des externen Programms leider nicht immer korrekt zurück (zum Beispiel bei der Verkettung von Befehlen).

Prozesskontrolle

Das folgende PHP-Skript gibt per »exec()« aus, wie viele Prozesse im System laufen:

```
unset ($out);
$cmd="ps axu | wc -l";
$erg=exec($cmd);
$erg-=1; //minus 1 Zeile Überschrift
echo "Zurzeit laufen $erg Prozesse\n";
```

Es löscht zuerst der Inhalt der Variablen »$out«, die für das Array steht, welches die Befehlsausgabe aufnehmen soll. »exec()« überschreibt dieses Array nicht, sondern würde es gegebenenfalls vergrößern. Das Beispiel vermeidet es, die Resultate verschiedener Skriptläufe zu vermengen, deshalb wird das Array geleert. Das Kommando ermittelt die Zeilenzahl des Prozesslistings und subtrahiert dann eine Zeile, die für die Spalten-Überschriften verwendet wurde.

»shell_exec()« und »exec()« übernehmen allerdings nur Daten, die über Stdout kommen. Um auch die Fehlermeldungen abzufangen, die auf Stderr landen, ist der Fehlerkanal mit »2 > &1« auf Stdout umzuleiten.

Häufig will der Programmierer Kommandozeilen-Argumente an ein Skript übergeben. PHP trägt deshalb alle Argumente, die auf der Kommandozeile hinter dem Dateinamen auftauchen, in ein Array namens »$argv« ein. Die Reihenfolge behält es bei. Zusätzlich wird die Anzahl der übergebenen Werte in der Variablen »$argc« gespeichert.

Tabelle 1: Die wichtigsten Ansi-Sequenzen

Bedeutung	Sequenz
Bildschirm löschen	\033[2J
Cursor auf Position *x,y*	\033[*x;y*H
Cursor *n* Zeichen nach links	033[*n*C
Cursor *n* Zeichen nach oben	\033[*n*A
Schriftfarbe Rot	\033[0;31m
Schriftfarbe Schwarz	\033[0;30m
Schriftfarbe Grün	\033[0;32m
Schriftfarbe Hellgrau	\033[0;37m

Gute Argumente

Befehle, mit denen sich Daten direkt von der Tastatur einlesen lassen, kennt PHP nicht. Um trotzdem Eingaben zu ermöglichen, kann man den Stream »stdin« mit Dateifunktionen öffnen und lesen. Auch die verwandten Streams »stdout« und »stderr« sind damit nutzbar:

```
$in=fopen("php://stdin","r");
$err=fopen("php://stdin","w");
$eingabe=trim(fgets($fp, 100));
while (""==$eingabe)
{
    fputs($err,"Sie müssen einen Wert ➥
eingeben\n");
}
```

Wird »stdin« auf diese Weise zum Lesen von Tatstatureingaben verwendet, sollte ein Skript die übergebenen Daten immer mit »trim()« behandeln, das Whitespaces am Stringende entfernt. Um Daten auszugeben, braucht der Anwender nicht auf »stdout« zurückzugreifen, dafür gibt es die Funktionen »echo« und »print«. Die Ausgabe beider Kommandos lässt sich jederzeit umleiten.

Bunt garniert

Für eine etwas komfortablere Anwendung braucht der Programmierer Funktionen für das Löschen des Bildschirms, die Cursorsteuerung oder die farbliche Gestaltung der Ausgaben. Eine gute Lösung hierfür sind Ansi-Sequenzen, die die Shell interpretiert. Bekannt sind sie auch unter der Bezeichnung Escape-Sequenzen, die sich von dem Escape-Zeichen ableitet, mit dem sie beginnen. Zum Beispiel löscht »echo "\033[2J"« den Bildschirm. In Tabelle 1 sind einige der wichtigsten Sequenzen aufgeführt. Eine umfangreichere Liste bietet beispielsweise **[1].**

Die gewählte Farbe bleibt so lange gültig, bis eine andere Farbe eingeschaltet wird. Einen höheren Grad an Portabilität und einen viel größeren Funktionsumfang bieten PHPs Ncurses-Funktionen.

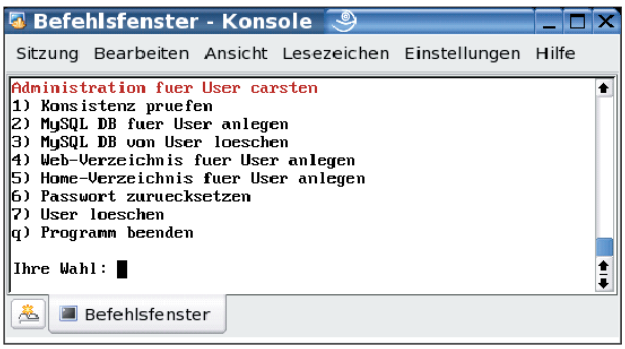

Abbildung 1: Das Menü eines in PHP programmierten Skripts für die Benutzerverwaltung. Das vollständige Listing steht unter [4].

Ausführliche Erläuterungen zu diesem Thema finden sich zum Beispiel auf der PHP- oder der Zend-Homepage [2], [3].

User-Verwaltung in PHP

Das folgende Beispiel demonstriert die besprochenen Funktionen im Zusammenhang. Es stammt aus einem Skript zur Benutzerverwaltung auf einem Schulungsserver. In Abbildung 1 ist das Menü zu sehen, das durch diesen Code dargestellt wird. Listing 1 zeigt einen Ausschnitt, das vollständige Skript steht unter [4] zum Download bereit.

An dem Skript fällt auf, dass es die Standardeingabe zwar mit »fopen()« öffnet, aber nicht mit »fclose()« schließt. Weil PHP offene Streams aber bei Programmende automatisch schließt, verzichtet das Skript der besseren Lesbarkeit wegen darauf. Für Kommandozeilenprogramme ist die Ausführungszeit nicht beschränkt.

PHP eignet sich ideal für die Erstellung eines Webinterface. Diese verlockende Möglichkeit birgt aber auch Gefahren. Jedenfalls sollte man sich bei der Konzeption gründlich Gedanken über die Sicherheitsrisiken machen. Jedes noch so kleine Sicherheitsloch kann fatale Folgen haben. Ein Beispiel:

```
<form method="POST">
    Befehl: <input name="cmd" /><br />
    <input type="submit"value="Ausführen"
/>
</form>
<?php
    if (isset ($_POST['cmd']))
    {
       $ausg=shell_exec($_POST[cmd]);
       echo "<pre>$ausg</pre>";
    }
?>
```

Lösungen wie in diesem Code, die einfach jeden beliebigen Befehl ausführen, sind indiskutabel. Grundsätzlich sollten alle administrativen Skripte nur über eine sichere Verbindung ausführbar sein und keine Möglichkeiten zur Manipulation bieten.

Sonderrechte

Nicht selten benötigen Skripte Root-Rechte. Ein normales PHP-Skript läuft innerhalb einer Webanwendung mit den Rechten des Users, der den Webserver gestartet hat. Einfach diesem Benutzer – meist »www«, »wwwrun« oder »nobody« – mehr Rechte einzuräumen oder den Service gar durch Root starten zu lassen ist eine nahe liegende, aber gefährliche Idee. Alternativen bieten die Kommandos »sudo« oder »su«. Für »su« wäre allerdings das Root-Passwort als Klartext im Skript zu hinterlegen, was ebenfalls unsicher ist.

Das Tool »sudo« hat außerdem den Vorteil, dass es einem Benutzer für bestimmte Befehle Root-Rechte einräumen kann, ohne dass er dafür das Root-Passwort zu kennen braucht. Er authentifiziert sich stattdessen mit seinem eigenen Passwort und erhält daraufhin bestimmte Privilegien, die Root in der Datei »/etc/ sudoers« definiert.

Möchte der Admin beispielsweise dem User »webadmin« gestatten, fremde Pro-

Listing 1: PHP-Menü

```
01 #!/usr/bin/php -q
02 <?php
03 function cls()
04 {
05    echo "\033[2J"; //Bildschirm loeschen
06    echo "\033[0;0H"; // Cursor auf 0,0
07 }
08
09 function text_rot()
10 {
11    echo "\033[0;31m";
12 }
13
14 function text_schwarz()
15 {
16    echo "\033[0;30m";
17 }
18
19 // weitere Funktionen
20
21 // Wurde ein Username uebergeben?
22 $in=fopen("php://stdin","r");
23 $err=fopen("php://stderr","w");
24 if (3==$argc &&
25     "-u"==$argv[1] &&
26     isset($argv[2]))
27 {
28     $user=$argv[2]; // Usernamen auslesen
29 }
30 else
31 {   // Username wurde nicht angegeben
32     fputs($err,"Bitte mit -u user einen
   Usernamen angeben\n");
33     exit(1);
34 }
35 // Funktion um mit /etc/passwd zu
36 // prüfen ob es den User gibt
37 if (false === user_exists($user))
38 {
39     fputs($err,"Username existiert nicht\n");
40     exit(2);
41 }
42
43 while (1) //Endlosschleife zur Verarbeitung
44 {
45     cls();
46     text_rot();
47     echo "Administration fuer User $user\n";
48     text_schwarz();
49     echo "1) Konsistenz pruefen\n";
50     echo "2) MySQL DB fuer User anlegen\n";
51     // Weitere Menuepunkte
52     echo "q) Programm beenden\n\n";
53     echo "Ihre Wahl: ";
54     // Bei der Eingabe Whitespaces entfernen
55     $eingabe=trim(fgets($in,255));
56
57     switch ($eingabe)
58     {
59         case "1": konsistenz_check($in);
60                   break;
61         // Weitere Cases
62         case "q": exit(0);
63
64         // Piepton bei ungueltiger Eingabe
65         default:  echo chr(7);
66     }
67 }
68 ?>
```

zesse mit »kill« zu beenden, trägt er die folgende Zeile in die Datei »/etc/sudoers« ein:

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

Dafür verwendet man am besten den speziellen Editor »visudo«, der dafür sorgt, dass nicht mehrere Benutzer gleichzeitig das Konfigurationsfile bearbeiten können. Außerdem überprüft er, ob die Einträge regelkonform und widerspruchsfrei sind.
Die allgemeine Syntax zur Vergabe von Rechten mit Sudo lautet WER WO = WAS. WER kann ein Username wie in diesem Beispiel oder auch eine Gruppe von Usern sein, die vorher mit Hilfe von »User_Alias« zu definieren wäre. WO bezeichnet den Host, auf dem der User die Rechte erhalten soll. Ein weiterführendes Tutorial zu Sudo bietet die Website von Jochen Lillich [5].

Identitätswechsel

Nach dem Eintrag in Sudos Konfigurationsdatei kann »webadmin« jetzt zwar wie Root jeden Prozess mit »kill« beenden, muss vorher allerdings sein Passwort eingeben. Ein automatisch ablaufendes Skript würde durch diese Aufforderung zur interaktiven Eingabe unterbrochen. Diese Klippe lässt sich umschiffen, indem man Sudo mit dem Parameter »-S« dazu anweist, das Passwort von der Standardeingabe zu lesen. Eine vollständige Befehlszeile würde bis jetzt also etwa lauten: »echo *geheim* | sudo -S kill 13221«
Ist der Benutzer »webadmin« aus der Perspektive der Unix-Prozesse nicht Herr des Webservers – wodurch alle Skripte seine Sonderrechte erben würden, was in den meisten Fällen unerwünscht ist –, dann ist eine Kombination von »su« und »sudo« erforderlich. Der Webserver-User nimmt zuerst mit »su« temporär die Identität »webadmin« an und erhält danach via »sudo« für bestimmte Kommandos die Rechte von Root.

Der Autor

Carsten Möhrke ist selbstständiger Berater und Dozent, Autor des Buchs „Besser PHP programmieren", Inhaber der Firma Netviser und per E-Mail unter [cmoehrke@netviser.de] zu erreichen.

Ein Skript, das diesem Konzept entspricht, könnte damit folgendermaßen aussehen:

```
$user="webadmin";  //sudoer
$pwd="geheim"; // Webadmins Passwort
$befehl="kill -9 ".escapeshellarg($_GET
["pid"]);
$cmd_line="echo $pwd | su -l $user -c
\"echo $pwd | sudo -S $befehl 2>&1\"";
$ret=shell_exec($cmd_line);
```

Das Skript bearbeitet die »pid«, da sie aus einem Formular übernommen wird, erst mit »escape-shellargs()«. Das stellt sicher, dass keine Shell-Injection erfolgen und somit boshafter Code ausgeführt werden kann.

Getrennte Welten

Die sicherste Variante ist jedoch eine komplette Trennung des Webfrontends von dem ausführenden Code, der Root-Rechte benötigt. Bei einem Skript zum Anlegen von neuen Usern könnte das beispielsweise so funktionieren: Mit Hilfe eines Webfrontends gibt der Admin die Benutzernamen für jene Accounts ein, die neu anzulegen sind. Diese schreibt das Skript dann in eine Datenbank oder eine Datei.
Ein zweites Skript, das regelmäßig in kurzen Abständen als Cronjob startet, liest die Daten anschließend aus und legt anhand der Einträge die neuen Benutzer an. Auch hierbei sollte immer »escape-shellargs()« die übergebenen Daten behandeln, um Shell-Injections zu verhindern. Diese Vorgehensweise bietet ein recht hohes Maß an Sicherheit und eignet sich für alle Anwendungsfälle, bei denen der Administrator nicht auf die unmittelbare Rückgabe eines Befehls angewiesen ist. *(jcb)*

Infos

[1] Ansi-Codes im Überblick: [http://www-user.tu-chemnitz.de/~heha/hs_freeware/terminal/terminal.htm]
[2] Manual der Ncurses-Funktionen: [http://www.php.net/ncurses]
[3] Tutorial zu Ncurses: [http://www.zend.com/zend/tut/tut-degan.php]
[4] Listing: [ftp://ftp.linux-magazin.de/2004/12/PHP]
[5] Sudo-Tutorial: [http://www.jochen-lillich.de/artikel/sudo]