



## Server-Auslastung

# Grafische Überwachung mit PHP und Bash

Wenn Sie einen Web-Server betreiben, kennen Sie das Problem: Ist die Lastkurve des Rechners von den letzten paar Tagen einzusehen, und könnte man sie per Browser abrufen? Genau das programmieren Sie in diesem Script mit der Bash und PHP.

THOMAS WÖLFER

Für die geschilderte Aufgabe existieren teilweise kommerzielle und teilweise freie Programme, das Interessante am Selber-programmieren ist aber das Lernen. Viele der großen Tools verwenden für die Server-Überwachung zum Teil recht komplexe Programme und setzen zumindest zum Teil auf dem SNMP-Protokoll auf.

Wie Sie sehen werden, geht das für kleinere Aufgaben auch einfacher – die Bash, ein Cron-Job und etwas PHP reichen völlig aus. Das per Cron regelmäßig aufgerufene Bash-Script kümmert sich dabei um die Erfassung der Daten, während das PHP-Programm diese auswertet und eine Webseite daraus baut.

Unter Linux besteht eine Vielzahl von Möglichkeiten, sich die Server-Last anzeigen zu lassen. Eine davon ist das *Uptime*-Kommando. Dieses Kommando gibt eine einzelne Zeile aus. Diese Zeile

enthält alle benötigten Informationen, allerdings auch ein paar, die nicht ganz so wichtig sind. *Uptime* zeigt die folgenden Informationen an:

- aktuelle Uhrzeit
- die *Uptime* des Rechners
- die Anzahl der angemeldeten Benutzer
- und die durchschnittliche Systemlast in der letzten Minute, den letzten 5 Minuten und den letzten 15 Minuten.

Diese durchschnittliche Systemlast ist die Serverlast, die Sie als Administrator interessiert. Dabei werden die „Aus-

schläge“ immer größer, je kürzer die betrachteten Zeiträume sind. Die durchschnittliche Systemlast in der letzten Minute gibt einem einen ungefähren Eindruck der „aktuellen“ Last, während die anderen beiden Werte eine bessere Übersicht über die dauerhafte Rechnerlast bieten. Die Durchschnittswerte werden dabei vom Betriebssystem selbst ermittelt – Sie brauchen also keinen zusätzlichen Rechenaufwand zu betreiben.

## ■ Daten erfassen – und speichern

Damit die Daten für die regelmäßige Analyse zur Verfügung stehen, müssen Sie das *Uptime*-Kommando nur regel-

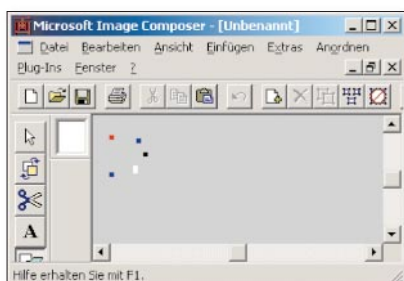
mäßig ausführen und die Ausgabe des Kommandos dabei in eine Datei umleiten – sinnvollerweise immer in die gleiche Datei. Um unter Unix/Linux ein Kommando regelmäßig auszuführen, verwenden Sie einen Eintrag in der Cron-Tabelle des Rechners. Diese befindet sich meist in einer Datei namens „crontab“. Dabei hat jeder User die Möglichkeit, eine eigene Cron-Tabelle zu verwalten. Am besten legen Sie den Job in Ihrer „eigenen“ Cron-Tabelle an. Das Beispiel-Script finden Sie unter dem

```
ssh-nickles - PuTTY
File Edit Search Buffers Windows System Help
1:18pm up 59 min. 1 user, load average: 0.02, 0.02, 0.00
1:33pm up 1:14, 0 users, load average: 0.00, 0.00, 0.00
1:48pm up 1:29, 1 user, load average: 0.00, 0.00, 0.00
2:03pm up 1:44, 0 users, load average: 0.00, 0.00, 0.00
2:18pm up 1:59, 0 users, load average: 0.00, 0.00, 0.00
2:33pm up 2:14, 1 user, load average: 0.00, 0.00, 0.00
2:48pm up 2:29, 1 user, load average: 0.00, 0.00, 0.00
3:03pm up 2:44, 1 user, load average: 0.00, 0.00, 0.00
3:18pm up 2:59, 1 user, load average: 0.00, 0.00, 0.00
3:33pm up 3:14, 1 user, load average: 0.00, 0.00, 0.00
3:48pm up 3:29, 1 user, load average: 0.00, 0.00, 0.00
4:03pm up 3:44, 1 user, load average: 0.00, 0.00, 0.00
4:18pm up 3:59, 1 user, load average: 0.00, 0.00, 0.00
4:33pm up 4:14, 1 user, load average: 0.00, 0.00, 0.00
4:48pm up 4:29, 1 user, load average: 0.00, 0.00, 0.00
5:03pm up 4:44, 2 users, load average: 0.00, 0.00, 0.00
5:18pm up 4:59, 2 users, load average: 0.00, 0.00, 0.00
5:33pm up 5:14, 2 users, load average: 0.00, 0.00, 0.00
5:48pm up 5:29, 2 users, load average: 0.00, 0.00, 0.00
6:03pm up 5:44, 2 users, load average: 0.00, 0.00, 0.00
6:18pm up 5:59, 1 user, load average: 0.00, 0.00, 0.00
6:33pm up 6:14, 1 user, load average: 0.00, 0.00, 0.00
6:48pm up 6:29, 0 users, load average: 0.00, 0.00, 0.00
7:03pm up 6:44, 0 users, load average: 0.25, 0.17, 0.06
7:18pm up 6:59, 0 users, load average: 0.00, 0.00, 0.00
7:33pm up 7:14, 0 users, load average: 0.00, 0.00, 0.00
7:48pm up 7:29, 0 users, load average: 0.00, 0.00, 0.00
8:03pm up 7:44, 0 users, load average: 0.00, 0.00, 0.00
+((Jed B0.99.9) Emacs: check.log (Text) 1/16621 4:30pm--
```

SO SEHEN DIE auszuwertenden Datensätze aus: Lange Dateien voller Zahlen – eine Zeile alle 15 Minuten.

Namen „check.sh“ zusammen mit allen anderen benötigten Dateien auf der Heft-CD und zwar unter dem Namen „check.sh“. Angenommen Sie kopieren dieses Script in das Verzeichnis `/usr/home/ACCOUNT/jobs/`, muss der Eintrag in Ihrer Cron-Tabelle den folgenden Aufbau haben:

```
* * * * * /usr/home/ACCOUNT/jobs/check.sh
```



FÜRS SCRIPT BENÖTIGEN SIE eine Hand voll bunter Pixel: Diese können Sie mit jedem Malprogramm erstellen, oder Sie verwenden die passenden Dateien von der Heft-CD.



Damit wird das Script alle 15 Minuten aufgerufen und zwar in jeder Stunde und

```
ash-nickles - PuTTY
$SERVERNAME = "Beispielserver.de";
$LOGFILENAME = "check.log";

function DisplayTitle()
{
    global $SERVERNAME;
    print "<font face=Verdana,Arial size=2><b>Auslastung des Servers: $SERV
}

function DisplayMenu()
{
    $SHORTPERIODTITLE = 'Minuten-Intervall';
    $MEDIUMPERIODTITLE = '5-Minuten Intervall';
    $LONGPERIODTITLE = '15-Minuten Intervall';

    print '<font face=verdana,arial size=1>';
    print '[<b><a href="check.php3?cmd=1&t=$SHORTPERIODTITLE">$SHORTPERIOD
    print '[<b><a href="check.php3?cmd=2&t=$MEDIUMPERIODTITLE">$MEDIUMPERI
    print '[<b><a href="check.php3?cmd=3&t=$LONGPERIODTITLE">$LONGPERIODTI
    print '<br size=1></font>';
}

-C:\Jed B0.99.9) Emacs: check.php3 (Text) 27/101 4:32pm
```

**WENN SIE DAS SCRIPT** direkt einsetzen möchten, können Sie es mit den beiden Zuweisungen am Anfang des Scripts an Ihren Server anpassen.

an jedem Tag der Woche. In diesem Zusammenhang ein kurzer Hinweis: Das Script selbst schreibt die Daten immer in die gleiche Datei und hängt dabei neu anfallende Daten hinten an. Das führt dazu, dass diese Datei in ihrer Größe immer mehr wächst. Sie müssen also sicherstellen, dass die Datei nicht zu groß wird. Entfernen Sie alle paar Wochen einige Zeilen aus der Datei. Nicht vergessen: Alle 15 Minuten fällt eine neue Zeile an!

## ■ Das Bash-Script

Das Bash-Script selbst ist verhältnismäßig unspektakulär. Alles, was Sie tun müssen, ist, *uptime* aufzurufen und das Resultat des Kommandos in eine Datei umzulernen:

```
#!/bin/sh
/usr/bin/uptime >> /usr/home/
ACCOUNT/jobs/check.log
```

Dabei müssen Sie sicherstellen, dass Sie auch den richtigen Pfad zum *Uptime*-Kommando verwenden: Der abgedruckte Pfad ist zwar wahrscheinlich, muss aber mit dem Pfad auf Ihrem System nicht übereinstimmen. Um herauszufinden, wo sich der *Uptime*-Befehl befindet, können Sie das Kommando *which uptime* verwenden, das Ihnen den kompletten Pfad zum Kommando liefert. Genauso müssen Sie die verwendeten Pfade zum "check.sh"-Kommando in der Cron-Tabelle und den Pfad zur Datei "check.log" im Script selbst anpassen.

Außerdem müssen Sie sicherstellen, dass das Script in die angegebene Datei schreiben darf. Dazu legen Sie diese Datei der Einfachheit halber mit "touch check.log" an und vergeben die passenden Zugriffsrechte für diese Datei. Damit ist der administrative Teil

der Arbeit beendet, und Sie können sich an die Implementierung des PHP-Programmes machen. Wenn Sie übrigens beim Ausprobieren Probleme mit der "check.log"-Datei haben – zum Beispiel, weil Sie in der Kürze der Zeit einfach nicht genügend Daten für einen sinnvollen Kurvenverlauf zusammenbekommen, können Sie den Beispiel-Datensatz von der Heft-CD verwenden.

## ■ PHP – Script beginnt

Einige Grundlagen zu PHP finden Sie an anderer Stelle in diesem Sonderheft in den weiteren PHP-Beispielen und -Programmen.

Das PHP-Script, das Sie nun implementieren werden, ist gar nicht so aufwändig, dafür aber ein wenig trickreich. Für die graphische Darstellung von Kurven und ähnlichen Aufgaben wird meist die GD-Library eingesetzt.

Diese Library ist sehr mächtig – allerdings für das einfache Zeichnen von "hübschen" Kurven nicht notwendig, denn das geht auch mit ein bisschen Logik und kleinen GIF-Dateien.

Für die nun folgenden Kurven benötigen Sie keine GD-Unterstützung auf Ihrem Server, vier GIF-Bilder reichen aus. Diese tragen die Namen red.gif, blue.gif, black.gif und white.gif – und bestehen im Wesentlichen aus einem Pixel in der Farbe des Dateinamens. Diese Pixel-Dateien können Sie mit jedem beliebigen Grafikprogramm erstellen, oder Sie verwenden die GIF-Dateien von der Heft-CD.

Das PHP-Script setzt sich aus drei Funktionen, einem relativ kleinen Hauptteil und einem Bereich für die Anpassung an Ihren Server zusammen.

Diesen Anpassungs-Teil finden Sie in den ersten beiden Programmzeilen des Scripts:

```
$SERVERNAME = "Beispielserver
.de";
$LOGFILENAME = "check.log";
```

Hier werden zwei globale Variable definiert, die an anderen Stellen im Programm verwendet werden. Diese beinhalten den Namen des Web-Servers in der Form, wie er später auf der Webseite angezeigt werden soll, und den Namen der Datei mit den gesammelten Ergebnissen des *Uptime*-Kommandos. Hier können Sie auch einen kompletten Pfad zur Datei angeben, also zum Beispiel */usr/home/ACCOUNT/jobs/check.log*.

Der Hauptteil des Scriptes befindet sich ganz unten in der Datei und kümmert sich um die richtige Reihenfolge beim Aufruf der Funktionen sowie um die Ausgabe der elementaren HTML-Sequenzen:

```
print '<html><head>
</head><body bgcolor=
GAINSBORO>';
DisplayTitle();
DisplayMenu();
if( IsSet( $cmd ))
    RenderUptime( $cmd, $t );
print '</body></html>';
```

Zunächst wird immer der Titel der Seite ausgegeben, dazu dient der Aufruf von *DisplayTitle()*. Danach wird mit *Dis-*

```
check.php3 - Editor
Datei Bearbeiten Format ?
<?php
// check.php3 - graphische darstellung der serverauslastung
// aus dem pc-magazin sonderheft
// copyr'ght(c) 2002 thomas woelfer - www.woelfer.com

$SERVERNAME = "Beispielserver.de";
$LOGFILENAME = "check.log";

function DisplayTitle()
{
    global $SERVERNAME;
    print "<font face=Verdana,Arial size=2><b>Auslastung des Servers:
$SERVERNAME</font><br>";
}

function DisplayMenu()
{
    $SHORTPERIODTITLE = 'Minuten-Intervall';
    $MEDIUMPERIODTITLE = '5-Minuten Intervall';
    $LONGPERIODTITLE = '15-Minuten Intervall';

    print '<font face=verdana,arial size=1>';
    print '[<b><a href="check.php3?cmd=1&t=$SHORTPERIODTITLE">$SHORTPERIODTITLE</a></b>';
    print '[<b><a href="check.php3?cmd=2&t=$MEDIUMPERIODTITLE">$MEDIUMPERIODTITLE</a></b>';
    print '[<b><a href="check.php3?cmd=3&t=$LONGPERIODTITLE">$LONGPERIODTITLE</a></b>';
    print '<br size=1></font>';
}
```

**SIE KÖNNEN DIE ANPASSUNG** des Scriptes auch unter Windows mit Notepad vornehmen. Praktischer ist aber ein Editor auf dem Server.

*playMenu()* ein Menü ausgegeben. Wurde das Script mit dem Parameter „cmd“ aufgerufen, wird außerdem die Funktion *RenderUptime()* bemüht, die sich in Abhängigkeit von zwei Parametern um die Ausgabe und Darstellung der Daten am Bildschirm kümmert.

*DisplayTitle()* erzeugt wie gesagt nur den Titel der Seite. Dazu verwendet es die globale Variable *\$SERVERNAME*, mit der Sie das Script dazu bewegen können, den Namen Ihres Servers zu verwenden:

```
function DisplayTitle() {
```

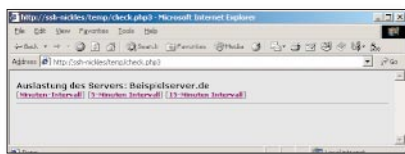


```
global $SERVERNAME;
print "<font face=Verdana,
    ↳Arial size=2><b>
    ↳Auslastung des Servers:
$SERVERNAME<\b></font><
    ↳br>";
}
```

Die Funktion *DisplayMenu()* erzeugt ein Menü, aus dem Sie auswählen können, welche der Kurvenverläufe Sie anzeigen möchten. Dabei werden die URLs mit den richtigen Parametern für den Aufruf von "check.php3" direkt mit ausgegeben.

Auf diese Weise können Sie die Menü-Funktion immer verwenden – egal, welche Seite gerade angezeigt wird. Zusätzlich zum Typ der Kurve wird außerdem der Titel der Kurve mit in den URL-Parametern codiert, sodass Sie im Script diese Titel nur an einer Seite definieren brauchen:

```
function DisplayMenu()
{
    $SHORTPERIODTITLE = 'Minuten-
    ↳Intervall';
    $MEDIUMPERIODTITLE =
    ↳'5-Minuten Intervall';
    $LONGPERIODTITLE =
    ↳'15-Minuten Intervall';
    print '<font face=verdana,
    ↳arial size=1>';
    print "[<b><a
href='check.php3?cmd=1&t=$SHORT
    ↳PERIODTITLE'>$SHORTPERIOD
    ↳TITLE</a></b>] ";
    print "[<b><a
href='check.php3?cmd=2&t=$MEDI
    ↳UM PERIODTITLE'>$MEDIUMPERIOD
    ↳TITLE</a></b>] ";
    print "[<b><a
href='check.php3?cmd=3&t=$LONG
    ↳PERIODTITLE'>$LONGPERIODTITLE<
    ↳</a></b>]";
    print '<chr size=1><
    ↳/font>';
}
```



**DIREKT NACH DEM START** zeigt das Script nur sein Menü an. Wählen Sie aus, welche der Kurven angezeigt werden soll.

## ■ Und nun zur Anzeige ...

Die tatsächliche Anzeige der Kurve erfolgt mit der Funktion *RenderUptime()*. Diese wird mit zwei Parametern aufgerufen. Der eine bestimmt die Art der Kurve, der andere gibt den Titeltext für die Kurve an:

```
function RenderUptime( $idType,
    ↳$strTitle)
{
```

Danach öffnen Sie die Datei mit den *Uptime*-Daten. Wo diese Datei liegt, wird über eine globale Variable definiert – wenn Sie das Script nur anpassen und direkt verwenden wollen, ist es ausreichend nur diese Variable am Anfang des Scripts zu setzen:

```
global $LOGFILE
    ↳NAME;

$logfile = file
    ↳($LOGFILE-
    ↳NAME );
```

Mit dem *file()*-Kommando wird die Datei nicht nur geöffnet, sondern auch komplett gelesen. Sie haben dann eine Variable namens „Slog-file“, die die komplette Datei zeilenweise enthält. Diese Zeilen lesen Sie nun Zeile für Zeile aus:

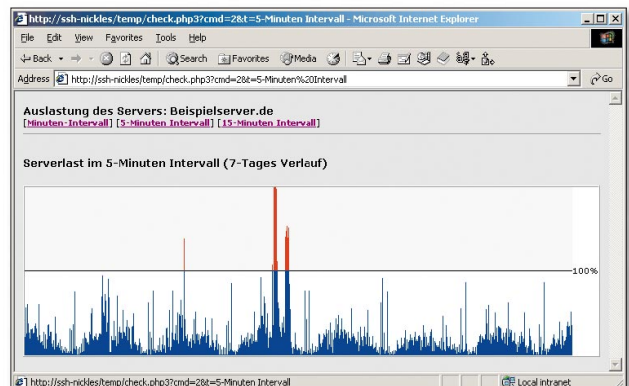
```
while( list(
    ↳$cLine, $line)
    ↳= each(
    ↳$logfile))
{
```

Die einzelnen Zeilen enthalten die Ausgaben des *Uptime*-Kommandos von je einem Aufruf. An dieser Stelle müssen Sie unter Umständen nochmals das Script anpassen und zwar dann, wenn Ihre Version von *Uptime* ein etwas anderes Format bei der Ausgabe verwendet.

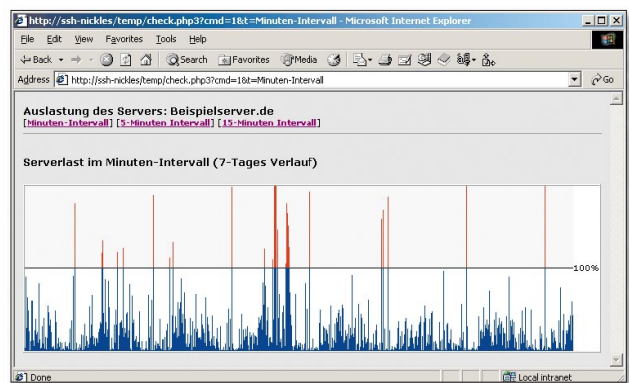
Ob das der Fall ist können Sie leicht an der Grafik erkennen: Ist diese leer, stehen die einzelnen Werte bei Ihnen vermutlich einfach an einer anderen Stelle in der Datei. Zunächst suchen Sie den Anfang der *Uptime*-Daten. Diese stehen hinter dem Wort „average“ in der Zeile:

```
$values = strstr( $line,
    ↳"average:");
```

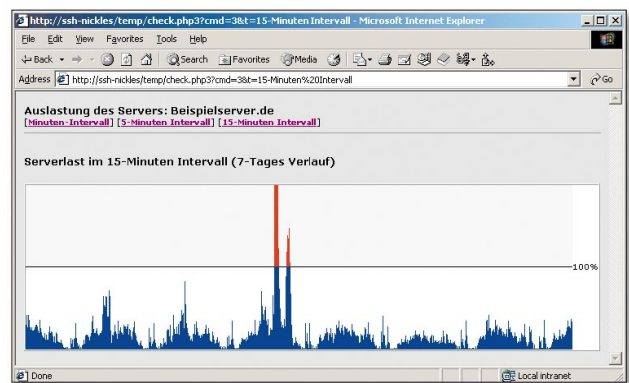
Teilen Sie den String an dieser Stelle auf. Als Trennzeichen verwenden Sie dazu



**DIE AUF FÜNF-MINUTEN-DATEN** basierende Kurve liegt nahe an der Realität: Die ersten „Sprünge“ verschwinden.



**BEI DER KURVE**, die auf den Minuten-Daten basiert sind logischerweise große Sprünge in der Lastverteilung zu sehen.



**DIE KURVE**, die auf 15-Minuten-Daten basiert, beschreibt die tatsächliche Server-Auslastung am genauesten. Der rote Teil symbolisiert dabei Überlast-Perioden.

das Komma: Dieses Zeichen trennt die drei "Uptime"-Angaben voneinander:

```
$arrValues = explode( ',', $
    ↳values);
```

Dadurch erhalten Sie ein Array: Jedes Feld des Arrays enthält nun einen der drei *Uptime*-Werte, allerdings sind diese teilweise von Leerzeichen umgeben. Diese Leerzeichen müssen noch weg. Das geht mit der *trim()*-Funktion:

```
$value = $arrValues[ $idType - 1];
$value = trim( $value);
```

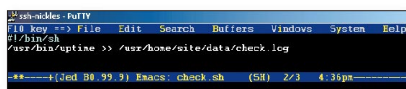




Schließlich speichern Sie den gewünschten Wert – also denjenigen der drei Werte, der zum angeforderten Kurvenverlauf passt – in einem Array. In jedem Schleifendurchlauf wächst das Array dabei um eine Zahl. Am Ende haben Sie ein großes Array, das alle *Uptime*-Werte, die zur angeforderten Kurve passen, enthält.

```
$arrUptime[ $cLine ] = $value;
}
```

Mit diesem Array können Sie nun weiterarbeiten. Allerdings sollte die Grafik auch leidlich hübsch aussehen. Daher

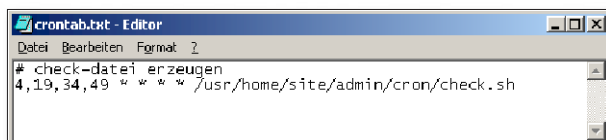


**DAS SHELL-SCRIPT** für die Sammlung der Lasten ist sehr einfach aufgebaut. Im Wesentlichen wird nur ein Kommando aufgerufen.

geben Sie zunächst eine Überschrift für die Grafik aus:

```
print "<br><font face=
↳Verdana,Arial size=2><b>
↳Serverlast im $strTitle (7-Tages
↳Verlauf)</font><br><br>";
```

Nun kommt der trickreiche Teil. Da das Script auf die GD-Library verzichten soll, ist es notwendig die Kurvenverläufe anders als mit echten Mal-Funktionen auszugeben. Das geht am einfachsten mit einer Tabelle. Diese Tabelle besteht im Wesentlichen aus einer Table-Row,



**DER CRONTAB-EINTRAG** für das Starten des Shell-Skripts. Alle 15 Minuten ist eine sinnvolle Periode.

und diese Table-Row enthält für jeden *Uptime*-Wert eine Zelle. Zunächst legen Sie die Tabelle an:

```
print "<table cellpadding=0
↳cellspacing=0>\n";
```

Nun überlegen Sie sich, wie lange die Periode sein soll, die Sie mit Ihrer Grafik abdecken möchten. Die *.log*-Datei enthält alle Daten seit dem ersten Start Ihres Shell-Skriptes. Das sind unter Umständen eine ganze Menge. Es empfiehlt sich daher, die Anzeige auf ein paar Tage einzuschränken.

Im Beispiel werden nur die letzten 7 Tage angezeigt, aber selbst das sind bereits schon über 670 Datenfelder! Um diese Limitierung zu erreichen, errechnen

Sie einen Startwert, den Sie als Index in der „*arrUptime*“ verwenden können:

```
// wir beginnen 7 Tage vor
↳'jetzt'
$vbBegin = count( $arrUptime ) -
↳(24 * 4 * 7);
if($vbBegin < 0) $vbBegin = 0;
```

Nun beginnen Sie die Ausgabe der Table-Row und iterieren über alle Werte ab dem errechneten Startwert bis zum letzten Wert aus dem Array:

```
print '<tr>';
for( $i = $vbBegin; $i < count
↳( $arrUptime);
↳$i++)
```

```
{
```

In jedem Schleifendurchlauf lesen Sie dann einen *Uptime*-Wert aus. Dabei ist zu berücksichtigen, dass das *Uptime*-Prozentzahlen liefert. Benötigt werden aber Integer im Bereich 0 bis hundert (bei Überlast, auch größer als hundert).

Daher müssen Sie den gelieferten Wert noch mit 100.0 skalieren und schließlich in einen Integer umwandeln:

```
$uptime = $arrUptime[ $i ];
$nuptime = (int)($uptime *
↳100.0);
```

Nun folgt eine wichtige Unterscheidung: Ist der *Uptime*-Wert geringer als 100, hat der Server keine Überlast, können Sie im Prinzip eine Zelle ausgeben, die einfach nur ein GIF-Bild mit der Höhe enthält, die in *\$nuptime* enthalten ist. Allerdings wirkt die

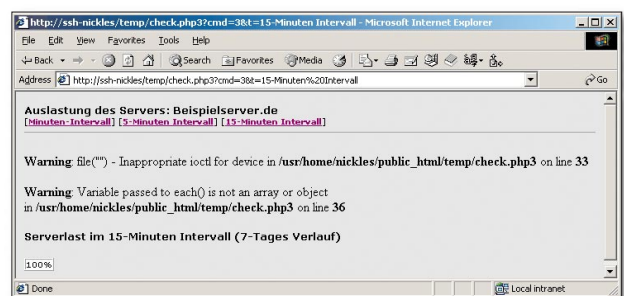
Grafik viel besser, wenn Sie außerdem in jeder Zelle den Wert „100“ besonders markieren. Im Beispiel funktioniert dies dadurch, indem Sie ein schwarzes GIF mit der Größe 1x1 ausgeben, dann ein weißes GIF, das die Höhe 100 - *\$nuptime* hat. Erst dann geben Sie das blaue GIF mit der Höhe *\$nuptime* aus:

```
if ( $nuptime <= 100 )
{
  $nwhite = 100 - $nuptime;
  if( ! $nuptime) $nwhite =
  ↳$nwhite - 1;
  print "<td width=1
  ↳valign=bottom bgcolor=
  ↳whiteSmoke>";
  print "<img src=black.gif
  ↳width=1 height=1><
  ↳img src=white.gif width=1
  ↳height=$nwhite>";
```

```
print "<img src=blue.gif
↳width=1 height=$nuptime
↳border=0></td>\n";
}
```

Besteht eine Überlast, ist die Logik bei der Ausgabe der Bilder ein wenig anders. Hier muss zusätzlich das rote GIF-Bild in passender Höhe ausgegeben werden. Um unschöne Sprünge zu vermeiden, wird als Maximalwert eine hundertprozentige Überlast verwendet.

An diesem Punkt ist der Server ohnehin weit über seine Belastungsfähig-



**WENN SIE BEI** der Konfektionierung des Skriptes einen falschen Pfad zur Datei angeben, kommt eine PHP-Fehlermeldung, wie in diesem Bild.

keit hinaus belastet, sodass eine genauere Darstellung von solchen Lastspitzen keine sinnvolle Information enthalten würde. Alle Werte, die über diesem Grenzwert liegen, werden auf diesen Wert gesetzt. Dann geben sie die Zelle aus:

```
else
{
  $noverload = $nuptime - 100;
  if( $noverload > 100)
  ↳$noverload = 100;
  print "<td width=1
  ↳align=bottom bgcolor=
  ↳whiteSmoke>";
  print "<img src=red.gif
  ↳height=$noverload width=1>
  ↳<br>";
  print "<img src=blue.gif
  ↳width=1 height=1 align=
  ↳center><br>";
  print "<img src=blue.gif
  ↳width=1 height=100 border=0>
  ↳</td>";
}
```

In diesem Beitrag haben Sie erfahren, wie Sie mit einfachen Mitteln ein praktisches Werkzeug zur Server-Überwachung programmieren können. Sogar die Grafiken können Sie ohne aufwändige Grafik-Bibliotheken erzeugen. An Erweiterungsmöglichkeiten des Skriptes bieten sich viele Dinge an. So wäre es zum Beispiel praktisch, wenn das Programm auch die Ausnutzung der Festplatte darstellen könnte. Viel Spaß beim Scripten und überwachen! UR