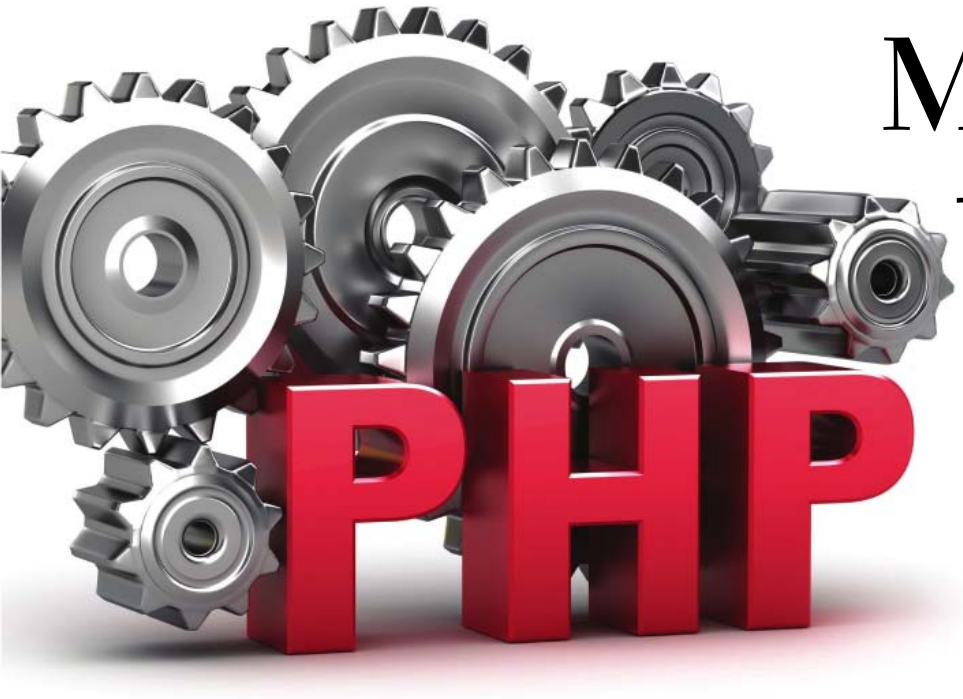

Momento terminal

PHP em linha de comando está no pedaço há mais de 10 anos, o que torna a linguagem e suas completas bibliotecas eminentemente adequadas para a caixa de ferramentas de qualquer administrador que gerencia servidores online.

por Tim Schürmann

A linguagem PHP está quase inseparavelmente associada a aplicativos online, também pode processar tarefas recorrentes, complexas ou simplesmente chatas na linha de comando. Um componente que tornou isso possível foi introduzido na versão 4.3, que remonta ao ano de 2002. O interpretador de linha de comando [1] fica escondido em um pacote à parte na maioria das distribuições. Normalmente, usa o mesmo nome do interpretador – por exemplo, no Debian 6 (Squeeze) e no Ubuntu, é `php5-cli` – mas, para o openSUSE 12.1, só precisamos do pacote `php5` completo.

Diga de uma vez!

A interface em linha de comando aceitará qualquer script PHP normal. Como poderíamos esperar, a saída não termina em um navegador, mas é enviada para a saída-padrão. Um simples programa "Olá Mundo" se parece com:

```
#!/usr/bin/php
<?php
echo "Olá mundo\n";
exit(0);
?>
```

A primeira linha (shebang, que contém o `#!`) permite aos usuários marcarem o arquivo como executável e chamá-lo como fariam com um script Shell normal. No entanto, também é possível iniciá-lo ao digitar:

```
php olamundo.php
```

O `\n` junto ao `Olá mundo` insere uma quebra de linha na saída, e o `exit(0)` gera um código de saída correspondente.

Como primeiro passo, é necessário certificar-se de que realmente é chamada a versão em linha de comando do PHP, pois o código CGI-PHP, por exemplo, executa scripts em um servidor online. Para maiores detalhes, consulte o **quadro 1**.

Antes de iniciar um script pela primeira vez, é possível validá-lo, usando:

```
php -l Script.php
```

que identificará a digitação e erros de sintaxe. Claro, podemos enviar comandos PHP para o interpretador através da entrada-padrão, o que significa que é possível conectar o script a outros programas com pipes:

```
echo "<?php echo 'Olá'; ?>" |
➥ php> cumprimento.txt
```

No contexto da linha de comando orientado para texto, os administradores frequentemente operam malabarismos com strings, e concatenar várias sequências é algo bastante simples de se fazer com o PHP:

```
$arquivo = "ferias";
$sufixo = "png";
echo "/home/tim/" . $arquivo
. ".$sufixo \n";
```

O PHP sempre usa um sinal de dólar para designar variáveis – programadores Bash estarão familiarizados com essa abordagem. Os pontos fora das aspas concatenam as strings; O PHP substitui automaticamente as variáveis dentro das aspas com seu conteúdo.

Emprestado do C

Para operações mais complexas, o PHP possui funções que se assemelham às suas contrapartes na linguagem de programação C. Por exemplo, o `strlen($name)` retorna o comprimento da string em `$name`, ao passo que `printf()` retorna uma string com a ajuda dos marcadores também usados em C:

```
tim@ubuntu: ~
tim@ubuntu:~$ php example.php Tim says hello
number of arguments: 4
argumente[0] = example.php
argumente[1] = Tim
argumente[2] = says
argumente[3] = hello
tim@ubuntu:~$
```

Figura 1 O nome do arquivo também pode ser avaliado como argumento.

```
$conta=5;
printf("Existem %d mensagens
➥ \n",$conta);
```

O PHP oferece muitas maneiras de dissecar strings. Por exemplo, o script `explode()` divide uma string de acordo com um delimitador definido – um espaço, neste caso – e coloca as partes em um array:

```
<?php
$texto = "Olá Senhor Miller";
$parte = explode(" ", $texto);
foreach ($dividido as $numero =>
➥ $parte) {
echo "$parte\n";
}
?>\
echo $dividido[0];
```

Este script disseca o texto `Olá Senhor Miller` nos espaços e salva as partes individuais em um array. O loop `foreach()` itera então contra o array e exibe o conteúdo linha por linha.

Os arrays no PHP são o que as outras linguagens chamam de dicionários ou tabelas de hash. Estas estruturas de dados armazenam um valor sob um índice; neste exemplo, o índice `0` refere-se à primeira palavra.

Os programadores também podem procurar por sequências usando expressões regulares em um estilo que lembra o Perl:

```
preg_condiz ("/Senhor/",
"Olá Senhor Miller", $condiz);
echo $condiz[0];
```

A função `preg_match()` armazena todos os condizentes em `$condiz`; neste exemplo, `$condiz[0]` contém a string `Senhor`.

Argumentos sólidos

É aconselhável ter cautela ao alimentar scripts PHP projetados para a Internet com a interface em linha de comando; ele simplesmente exibe qualquer texto do script que esteja fora do `<?... php?>`. Na pior das hipóteses, veremos apenas uma parte muito longa de HTML aparecendo na tela. Além disso, o PHP-CLI não tem os arrays `$_GET`, `$_ POST` e `$_COOKIE`, que é onde os dados devolvidos pelo navegador normalmente acabam.

Ao invés disso, todos os argumentos entregues na linha de comando acabam no array `$_SERVER['argv']`, que indica quantos argumentos foram passados. A **listagem 1** mostra um exemplo de todos os argumentos passados. Note a partir dos resultados da **figura 1**, o primeiro argumento é sempre o nome do arquivo de script que foi executado.

Se usarmos o `php` para chamar um script e também quisermos passar argumentos que comecem com um traço (`-`), como em:

```
php exemplo.php -h
```

a interface interpretará o argumento e, neste caso, mostrará seu próprio texto de ajuda. O separador de argumentos (`--`) ajudará aqui:

```
php exemplo.php -- -h
```

Este problema não ocorre se adicionarmos um shebang (linha com `#!/usr/bin/php`) no script – como em "Olá Mundo" – e em seguida tornarmos o script executável e o invocarmos diretamente.

```
tim@ubuntu: ~
tim@ubuntu:~$ php example.php
User of process: tim
UID: 1000
GID: 1000
Home directory: /home/tim
tim@ubuntu:~$ sudo php example.php
[sudo] password for tim:
User of process: root
UID: 0
GID: 0
Home directory: /root
tim@ubuntu:~$
```

Figura 2 O `posix_getpwuid()` retorna informações sobre a conta de usuário utilizada para executar o script.

Abridor de latas

Como muitas outras linguagens de programação, o PHP usa a função `fopen()` para abrir e retornar um manipulador de arquivo. Todas as outras rotinas de entrada e saída precisam então deste ponteiro para o arquivo. Uma dessas rotinas é `fgets()`, que lê uma linha de texto do arquivo:

```
$manipular = fopen("teste.txt",
➥ "r");
$linha = trim(fgets($manipular));
fclose($manipular);
```

A função `trim()` remove todos os espaços em branco (ou seja, espa-

ços e tabulações) no início e final de cada linha, e `fclose()` fecha o arquivo. Uma série de constantes predefinidas existe para facilitar a vida dos desenvolvedores. Ao invés de primeiro abrir a entrada-padrão com o comando:

```
fopen("php://stdin", "r");
```

e depois lendo uma linha, os programadores PHP podem usar a entrada STDIN diretamente:

```
$linha = fgets(STDIN);
```

A `STDOUT` e a `STDERR`, saída-padrão e erro-padrão, seguem o exemplo. Analisar um arquivo linha por linha também é brincadeira de criança com o PHP (**listagem 2**). Na **linha 4**, `file()` abre um arquivo e retorna todas as suas linhas como um array. Um loop `foreach` itera contra o array e exibe cada linha na tela.

Quadro 1: PHP como um CLI e um CGI

Interpretadores PHP estão disponíveis não apenas para a linha de comando, mas também para a interface CGI, que utiliza servidores online para iniciar programas externos. A versão em linha de comando difere da versão CGI principalmente em termos de saída. Por exemplo, o interpretador CGI sempre retorna um cabeçalho HTTP; a saída do script "Olá Mundo" seria, então, parecida com esta:

```
X-Powered-By: PHP/5.3.6-13ubuntu3.6
Content-type: text/html
Olá Mundo!
```

Obviamente, não precisamos deste cabeçalho na linha de comando. Um simples condicional `if` testa o script para ver se está sendo executado na variante de linha de comando, conforme desejado:

```
if (PHP_SAPI === 'cli')
{
[...]
}
```

Aliás, é possível usar este teste para verificar se o script não executa em um servidor online por engano. Outra maneira de determinarmos se estamos chamando a versão em linha de comando é emitir o comando `php -v`. A saída deve conter uma observação para o efeito que estamos usando (`CLI`).

Listagem 1: Avaliação de argumentos

```
01 <?php
02 echo "Número de argumentos: " . $_SERVER['argc'] . "\n";
03
04 $argumentos = $_SERVER['argv'];
05 foreach ($argumentos as $nr => $argumento) {
06 echo "argumentos[$nr] = $argumento\n";
07 }
08
09 exit(0);
10 ?>
```

Listagem 2: Análise de um arquivo de texto

```
01 <?php
02
03 $nomedoarquivo ="teste.txt";
04 $linhas = file($nomedoarquivo);
05
06 foreach ( $linhas as $numero => $linha ) {
07 $linha = trim($linha);
08 echo "$numero : $linha\n";
09 }
10
11 exit(0);
12 ?>
```

Aplicação prática

O PHP tem uma função correspondente que permite analisar um arquivo CSV: `fgetcsv()`; a **listagem 3** mostra o código de exemplo. Na **linha 2**, `fopen()` abre o arquivo `lista.csv`, então `fgetcsv()` pega a primeira linha do arquivo, extrai os campos de dados individualmente, e logo os coloca no array `$dado`. O loop `foreach()` subsequente gera todos os elementos do array na ordem correta.

O loop `while()` repete este procedimento até que o `fgetcvs()` atinja o fim do arquivo CSV e já não possa retornar nenhuma linha. É difícil imaginar uma forma mais conveniente ou um caminho mais curto de se fazer isso. O terceiro parâmetro em `fgetcsv()` especifica o delimitador; na **listagem 3**, as entradas estão separadas por vírgulas. Se estas vírgulas forem substituídas por dois pontos, poderemos facilmente analisar o arquivo de sistema `/etc/passwd`:

```
$arquivo = fopen("/etc/passwd", "r");
$dado = fgetcsv($file, 0, ":");
```

A contraparte ao `fgetcsv()` é o `fputcsv()`, que armazena quase automaticamente uma linha correspondente em um arquivo CSV. Para que isso aconteça, ele simplesmente precisa passar o caractere separador desejado e um array contendo todos os dados para a linha:

```
$registro = array("tims",
➥ "Tim","Schürmann");
$arquivo = fopen("teste.csv", "w");
fputcsv($arquivo, $registro, ",");
```

Na linha de comando, inúmeras funções POSIX são muito úteis. Por exemplo, `posix_geteuid()` retorna a ID do usuário sob a qual o script está sendo executado. Podemos então fornecer esta ID para o `posix_getpwuid()` (**listagem 4**)

para exibir um array de informações para a conta do usuário (**figura 2**). A função `posix_access()` (**listagem 5**) descobre as permissões de um arquivo (**figura 3**).

Manipulação de arquivos

O PHP tem uma função correspondente para mais ou menos qualquer operação de sistema de arquivos. Por exemplo, `chmod()` muda as permissões de arquivos, `ls_dir()` certifica-se de que o nome passado é um diretório, `copy()` copia um arquivo, e `mkdir()` cria um novo diretório:

```
mkdir("/home/tim/novo", 0700);
```

O segundo argumento define as permissões. Neste exemplo, somente o usuário que criou o script tem permissão para acessar um diretório. A função `scandir()` também fornece uma maneira fácil de manipular os arquivos em um diretório:

```
<?php
$arquivos = scandir('Imagens');
foreach ($arquivos as $arquivo) {
echo "$arquivo\n";
};
?>
```

Esse código usa o `scandir()` para exibir todos os arquivos no subdiretório `Imagens`. A **figura 4** mostra a função `getcwd()`, que avalia o diretório de trabalho atual.

A função `exec` executa programas externos, por exemplo. Os administradores podem usar praticamente qualquer extensão PHP na linha de comando, o que pode ser um recurso muito útil. Por exemplo, é possível acessar um arquivo Gzip que foi criado com as correspondentes funções Zlib e direcionar seu conteúdo para o console:

```
readgzfile("teste.txt.gz");
```

Outras funções semelhantes abrem e manipulam arquivos ZIP e RAR.

As funções de criptografia dão aos programadores uma maneira rápida de gerar uma impressão digital única de um arquivo – este exemplo usa a abordagem de hash MD5:

```
echo hash_file("md5", "teste.txt");
```

As funções que introduzi neste artigo são apenas um pequeno trecho a partir do enorme conjunto de recursos que satisfarão até os desejos

```
tim@ubuntu: ~
tim@ubuntu:~$ php example.php
file exists
file is readable
tim@ubuntu:~$ sudo php example.php
file exists
file is readable
file is writeable
tim@ubuntu:~$
```

Figura 3 O script da **listagem 5** executado como uma conta de usuário normal e como *root*.

Listagem 3: Análise de um arquivo CSV

```
01 <?php
02 $arquivo = fopen("lista.csv", "r");
03 while (($dado = fgetcsv($arquivo, 0, ",")) !== FALSE) {
04
05 echo "Conteúdo da linha:\n";
06 foreach ( $dado as $nr => $entrada ) {
07 $entrada = trim($entrada);
08 echo "$entrada\n";
09 }
10
11 }
12 fclose($arquivo);
13 exit(0);
14 ?>
```

Listagem 4: Propriedades do processo

```
01 <?php
02 $uid = posix_geteuid();
03 $usuario = posix_getpwuid($uid);
04 echo "Usuário do processo: $usuario[name] \n";
05 echo "ID do usuário: $usuario[uid] \n";
06 echo "ID do grupo: $usuario[gid] \n";
07 echo "Pasta pessoal: $userario[dir] \n";
08 ?>
```

Listagem 5: Permissões de arquivo

```
01 <?php
02 $existir = posix_access("/etc/passwd", POSIX_F_OK);
03 if($existir) echo "o arquivo existe\n";
04
05 $ler = posix_access("/etc/passwd", POSIX_R_OK);
06 if($ler) echo "o arquivo pode ser lido\n";
07
08 $escrever = posix_access("/etc/passwd", POSIX_W_OK);
09 if($escrever) echo "o arquivo pode ser escrito\n";
10
11 $executar = posix_access("/etc/passwd", POSIX_X_OK);
12 if($executar) echo "o arquivo é executável\n";
13 ?>
```

mais exóticos [2]. Por exemplo, ainda há a opção de se criar uma pequena interface para o usuário baseada em texto com o uso de funções `Ncurses`.

Conclusão

Embora o PHP possa não ser a linguagem de programação que imediatamente vem à mente quando buscamos uma linguagem de linha de comando, verificamos que este executa esta tarefa quase que perfeitamente. Graças a uma enorme biblioteca de funções, até mesmo a tarefa mais exigente irá normalmente se resumir a um par de linhas bem definidas, e um código fácil de ler.

A ferramenta em linha de comando também oferece um modo interativo que podemos habilitar digitando `php -a`. Este modo também suporta o serviço de autocompletar: pressionar a tecla [**Tab**] automaticamente completa os comandos que começamos a digitar. Caso tenhamos uma máquina com o PHP instalado, podemos facilmente dar um descanso ao Bash e nos voltarmos para o PHP-CLI como alternativa. ■

```
tim@ubuntu: ~
tim@ubuntu:~$ more scripts/example.php
<?php
        echo getcwd(), "\n";
?>
tim@ubuntu:~$ php scripts/example.php
/home/tim
tim@ubuntu:~$
```

Figura 4 O script `getcwd()` mostra que o diretório atual é aquele em que o usuário invocou o interpretador de linha de comando.

Mais informações

[1] PHP-CLI: `http://php.net/manual/en/features.commandline.php`

[2] Referência de funções PHP: `http://php.net/manual/en/funcref.php`

Gostou do artigo?

Queremos ouvir sua opinião. Fale conosco em cartas@linuxmagazine.com.br

Este artigo no nosso site: http://lnm.com.br/article/7422