

## 6. 从awk中读取命令输出

在下面的代码中，echo会生成一个空白行。变量cmdout包含命令grep root /etc/passwd的输出，该命令会打印出包含root的行。

将命令的输出结果读入变量output的语法如下：

```
"command" | getline output ;
```

例如：

```
$ echo | awk '{ "grep root /etc/passwd" | getline cmdout ; print cmdout }'  
root:x:0:0:root:/root:/bin/bash
```

通过使用getline，我们将外部shell命令的输出读入变量cmdout。

awk支持以文本作为索引的关联数组。

## 7. 在awk中使用循环

在awk中可以使用for循环，其格式如下：

```
for(i=0;i<10;i++) { print $i ; }
```

或者

```
for(i in array) { print array[i]; }
```

## 8. awk内建的字符串控制函数

awk有很多内建的字符串控制函数，让我们认识一下其中部分函数。

- ❑ length(string)：返回字符串的长度。
- ❑ index(string, search\_string)：返回search\_string在字符串中出现的位置。
- ❑ split(string, array, delimiter)：用定界符生成一个字符串列表，并将该列表存入数组。
- ❑ substr(string, start-position, end-position)：在字符串中用字符起止偏移量生成子串，并返回该子串。
- ❑ sub(regex, replacement\_str, string)：将正则表达式匹配到的第一处内容替换成replacement\_str。
- ❑ gsub(regex, replacement\_str, string)：和sub()类似。不过该函数会替换正则表达式匹配到的所有内容。
- ❑ match(regex, string)：检查正则表达式是否能够匹配字符串。如果能够匹配，返回非0值；否则，返回0。match()有两个相关的特殊变量，分别是RSTART和RLENGTH。变

量RSTART包含正则表达式所匹配内容的起始位置，而变量RLENGTH包含正则表达式所匹配内容的长度。

## 4.7 统计特定文件中的词频

统计文件中使用单词的频率是一个不错的练习，在这里能够应用我们业已习得的文本处理技巧。词频统计的方法有很多。让我们看看具体的做法。

### 4.7.1 预备知识

我们可以使用关联数组、awk、sed、grep等不同的方式来解决这个问题。**单词**是由空格或句号分隔的字母组合。首先我们应该解析出给定文件中出现的所有单词，然后统计每个单词的出现次数。单词解析可以用正则表达式配合sed、awk或grep等工具来完成。

### 4.7.2 实战演练

我们已经了解了实现原理。现在来动手创建如下的脚本：

```
#!/bin/bash
# 文件名: word_freq.sh
# 用途: 计算文件中单词的词频

if [ $# -ne 1 ];
then
    echo "Usage: $0 filename";
    exit -1
fi

filename=$1

egrep -o "\b[[:alpha:]]+\b" $filename | \

awk '{ count[$0]++ }
END{ printf("%-14s%s\n", "Word", "Count") ;
for(ind in count)
{ printf("%-14s%d\n", ind, count[ind]); }
}'
```

输出如下：

```
$ ./word_freq.sh words.txt
Word          Count
```

```
used          1
this          2
counting      1
```

### 4.7.3 工作原理

`egrep -o "\b[[:alpha:]]+\b" $filename`只用于输出单词。用 `-o` 选项打印出由换行符分隔的匹配字符序列。这样我们就可以在每行中列出一个单词。

`\b` 是单词边界标记符。`[[:alpha:]]` 是表示字母的字符类。`awk`命令用来避免对每一个单词进行迭代。因为`awk`默认会逐行执行`{}`块中的语句，所以我们就不需要再为同样的事编写循环了。借助关联数组，当执行`count[$0]++`时，单词计数就增加。最后，在`END{}`语句块中通过迭代所有的单词，就可以打印出单词及它们各自出现的次数。

### 4.7.4 参考

- 1.7节讲解了`Bash`中的数组。
- 4.6节介绍了`awk`命令。

## 4.8 压缩或解压缩 JavaScript

JavaScript广泛用于网站设计。在编写JavaScript代码时，出于代码可读性与方便维护方面的考虑，我们有必要使用一些空格、注释和制表符。但这些会增加JavaScript文件的大小。随着文件体积的增加，网页载入的时间也随之延长。因此，多数专业网站为了加快页面载入速度，都会对JavaScript文件进行压缩。这多是通过压缩空白字符和换行符的数量来实现的（也被称为minified JS）。对于压缩过的JavaScript，还可以通过加入足够的空白字符和换行符进行解压缩，以便恢复代码的可读性。这则攻略就尝试利用shell中实现类似的功能。

### 4.8.1 预备知识

我们准备写一个JavaScript压缩工具或代码混乱器，当然，还包括与之对应的解压缩工具。来考虑下面的JavaScript代码：

```
$ cat sample.js
function sign_out()
{

    $("#loading").show();
    $.get("log_in",{logout:"True"},
```

```
function(){
  window.location="";
});
}
```

下面是压缩JavaScript所需要完成的工作：

- (1) 移除换行符和制表符；
- (2) 压缩空格；
- (3) 替换注释 /\* 内容 \*/。

要解压缩或者恢复JavaScript的可读性，我们则需要：

- 用 `;\n` 替换 `;`；
- 用 `{\n` 替换 `{`，`\n}` 替换 `}`。

## 4.8.2 实战演练

4

按照之前叙述过的步骤，我们使用下面的命令序列：

```
$ cat sample.js | \
tr -d '\n\t' | tr -s ' ' \
| sed 's:/\*.*\*/::g' \
| sed 's/ \? \([{}()];,;:1\)\ \? /\1/g'
```

得到输出：

```
function sign_out(){$("#loading").show();$.get("log_in",{logout:"True"},
function(){window.location="";}});}
```

接着写一个可以将这些混乱的代码恢复正常的解压缩脚本：

```
$ cat obfuscated.txt | sed 's/;/;\n/g; s/{/{\n\n/g; s/}/\n\n/g'
```

或者

```
$ cat obfuscated.txt | sed 's/;/;\n/g' | sed 's/{/{\n\n/g' | sed 's/}/\n\n/g'
```



该脚本在使用上存在局限：它会删除本不该删除的空格。假如有下列语句：

```
var a = "hello world"
```

两个空格会被转换成一个。这种问题可以使用我们讲解过的模式匹配工具来解决。如果需要处理关键JavaScript代码，最好还是使用功能完善的工具来实现。

### 4.8.3 工作原理

通过执行下面的步骤来进行压缩。

- (1) 移除 '\n' 和 '\t':

```
tr -d '\n\t'
```

- (2) 移除多余的空格:

```
tr -s ' ' 或者 sed 's/[ ]+/ /g'
```

- (3) 移除注释:

```
sed 's:/\*.*\*/::g'
```

因为我们需要使用 /\* 和 \*/，所以用:作为sed的定界符，这样就不必对/进行转义了。

\* 在 sed 中被转义为 \\*。

. \* 用来匹配 /\* 与 \*/ 之间所有的文本。

- (4) 移除 {、}、(、)、;、: 以及 , 前后的空格。

```
sed 's/ \?([{}()];:)\ \?/\1/g'
```

上面的sed语句含义如下。

- ❑ sed代码中的 / \?([{}()];:)\ \?/ 用于匹配，/\1/g 用于替换。
- ❑ \([{}()];:)\ 用于匹配集合 [ { } ( ) ; , : ] (出于可读性方面的考虑，在这里加入了空格) 中的任意一个字符。\' (和 \) 是分组操作符，用于记忆所匹配的内容，以便在替换部分中进行向后引用。对 (和) 转义之后，它们便具备了另一种特殊的含义，进而可以将它们作为分组操作符。位于分组操作符前后的 \? 用来匹配可能出现在字符集合前后的空格。
- ❑ 在命令的替换部分，匹配字符串 (也就是一个可选的空格、一个来自字符集的字符再加一个可选的空格) 被匹配的子字符串所替换。对于匹配的子字符串使用了向后引用，并通过分组操作符() 记录了匹配内容。可以用符号 \1 向后引用分组所匹配的内容。

解压缩命令工作方式如下:

- ❑ s/;/;\n/g 将; 替换为;\n;
- ❑ s/{/{\n\n/g 将 { 替换为 {\n\n;
- ❑ s/}/\n\n}/g 将 } 替换为 \n\n}。

#### 4.8.4 参考

- 2.6节讲解了tr命令。
- 4.5节讲解了sed命令。

### 4.9 按列合并多个文件

很多时候我们需要按列拼接文件，比如要将每一个文件的内容作为单独的一列。而cat命令所实现的拼接通常是按照行来进行的。

#### 4.9.1 实战演练

可以用paste命令实现按列拼接，其语法如下：

```
$ paste file1 file2 file3 ...
```

让我们来尝试一下：

```
$ cat file1.txt
1
2
3
4
5
$ cat file2.txt
slynux
gnu
bash
hack
$ paste file1.txt file2.txt
1slynux
2gnu
3bash
4hack
5
```

默认的定界符是制表符，也可以用-d明确指定定界符，例如：

```
$ paste file1.txt file2.txt -d ","
1,slynux
2,gnu
3,bash
4,hack
5,
```

## 4.9.2 参考

4.4节讲解了如何从文本文件中提取数据。

## 4.10 打印文件或行中的第 $n$ 个单词或列

我们可能有一个包含了多列数据的文件，不过只有其中的一小部分能派上用场。例如在以成绩排序的学生列表中，我们希望得到成绩最高的四名学生。来看看如何实现。

### 4.10.1 实战演练

处理这种任务最为广泛的方法就是借助awk。同样也可以使用cut来实现。

(1) 用下面的命令打印第5列：

```
$ awk '{ print $5 }' filename
```

(2) 也可以打印多列数据，并在各列间插入指定的字符串。

如果要打印当前目录下各文件的权限和文件名，可以使用下列命令：

```
$ ls -l | awk '{ print $1 " : " $8 }'
-rw-r--r-- : delimited_data.txt
-rw-r--r-- : obfuscated.txt
-rw-r--r-- : pastel.txt
-rw-r--r-- : paste2.txt
```

### 4.10.2 参考

- 4.4节讲解了如何从文本文件中提取数据。
- 4.6节讲解了awk命令。

## 4.11 打印行或样式之间的文本

我们有时候可能需要根据某些条件打印文本的某一部分，比如由行号或起止样式所匹配的范围。让我们来看看如何实现这些需求。

### 4.11.1 预备知识

我们可以用awk、grep和sed这类工具来根据条件打印某些文本区域。不过，我仍然觉得awk

是最简单明了的方法。下面就看看如何使用awk完成这项任务。

### 4.11.2 实战演练

- (1) 要打印出从M行到N行这个范围内的所有文本，使用下面的语法：

```
$ awk 'NR==M, NR==N' filename
```

也可以用stdin作为输入：

```
$ cat filename | awk 'NR==M, NR==N'
```

- (2) 把M和N换成具体的数字：

```
$ seq 100 | awk 'NR==4,NR==6'
4
5
6
```

- (3) 要打印处于start\_pattern与end\_pattern之间的文本，使用下面的语法：

```
$ awk '/start_pattern/, /end_pattern/' filename
```

例如：

```
$ cat section.txt
line with pattern1
line with pattern2
line with pattern3
line end with pattern4
line with pattern5

$ awk '/pa.*3/, /end/' section.txt
line with pattern3
line end with pattern4
```

用于awk中的样式为正则表达式。

### 4.11.3 参考

4.6节讲解了awk命令。

## 4.12 以逆序形式打印行

这一节的内容很简单。讲的东西可能看起来用处不大，不过它可以用来在Bash中模拟栈结构，

所以还是有些意思的。下面我们来将一个文件中的文本行以逆序形式打印出来。

### 4.12.1 预备知识

只需要用点awk的小技巧就能完成这项任务。不过，命令tac可以直接搞定。这个命令的名称其实就是反过来书写的cat。

### 4.12.2 实战演练

先来试试tac。

(1) 该命令的语法如下：

```
tac file1 file2 ...
```

它也可以从stdin中读取：

```
$ seq 5 | tac
5
4
3
2
1
```

在tac中，\n是默认的行分隔符。但我们也可以用-s "分隔符"选项指定自己的分隔符。

(2) 使用awk的实现方式如下：

```
$ seq 9 | \
awk '{ lifo[NR]=$0 }
END{ for(lno=NR;lno>-1;lno--){ print lifo[lno]; }
}'
```

在shell脚本中，\可以很方便地将单行命令拆解成多行。

### 4.12.3 工作原理

这个awk脚本非常简单。我们将每一行都存入一个关联数组中，用行号作为数组索引（行号由NR给出），最后由awk执行END语句块。为了得到最后一行的行号，在{ }语句块中使用lno=NR。因此，这个脚本从最后一行一直迭代到第0行，将存储在数组中的各行以逆序方式打印出来。

## 4.13 解析文本中的电子邮件地址和 URL

从给定的文件中解析出所需要的文本是从事文本处理时常见的一项任务。诸如电子邮件地址、URL等都能够借助适合的正则表达式找出来。我们通常需要从包含大量无关字符及单词的电子邮件客户列表或HTML网页中将电子邮件地址解析并提取出来。

### 4.13.1 实战演练

能够匹配一个电子邮件地址的正则表达式如下：

```
[A-Za-z0-9._]+@[A-Za-z0-9.]+\.[a-zA-Z]{2,4}
```

例如：

```
$ cat url_email.txt
this is a line of text contains, <email> #slynux@slynux.com. </email> and email address,
blog "http://www.google.com", test@yahoo.com dfdfdfdddfdf; cool.hacks@gmail.com <br />
<a href="http://code.google.com"><h1>Heading</h1>
```

因为用到了扩展正则表达式（例如+），所以得使用egrep命令。

```
$ egrep -o '[A-Za-z0-9._]+@[A-Za-z0-9.]+\.[a-zA-Z]{2,4}' url_email.txt
slynux@slynux.com
test@yahoo.com
cool.hacks@gmail.com
```

匹配HTTP URL的egrep正则表达式如下：

```
http://[a-zA-Z0-9\-\.\.]+\.[a-zA-Z]{2,4}
```

例如：

```
$ egrep -o "http://[a-zA-Z0-9\-\.\.]+\.[a-zA-Z]{2,3}" url_email.txt
http://www.google.com
http://code.google.com
```

### 4.13.2 工作原理

如果逐个部分进行设计，这些正则表达式其实很简单。在匹配电子邮件地址的正则表达式中，我们都知道电子邮件地址可以采用name@domain.some\_2-4\_letter这种形式。那么，在编写正则表达式时，也要遵循同样的规则：

```
[A-Za-z0-9._]+@[A-Za-z0-9.]+\.[a-zA-Z]{2,4}
```

`[A-Za-z0-9.]+` 表示在表示字面意义的字符 `@` 出现之前，`[]` 中的字符组合应该出现一次或多次（这也正是 `+` 的含义）。然后 `[A-Za-z0-9.]+` 同样应该出现一次或多次（`+`）。样式 `\.` 表示应该呈现一个表示字面意义的“.”（点号），而 `[a-zA-Z]{2,4}` 表示字母的长度应该在2到4之间（包括2和4）。

匹配HTTP URL与匹配电子邮件地址类似，只是不需要匹配`name@`部分。

```
http://[a-zA-Z0-9.]+\.[a-zA-Z]{2,3}
```

### 4.13.3 参考

- 4.2节讲解了如何使用正则表达式。
- 4.5节讲解了`sed`命令。

## 4.14 在文件中移除包含某个单词的句子

只要能写出正确的正则表达式，移除包含某个单词的句子简直就是手到擒来。这里给出了一个解决类似问题的练习。

### 4.14.1 预备知识

`sed`是进行文本替换的不二之选。这样，我们就可以通过`sed`用空白替代匹配的句子。

### 4.14.2 实战演练

先创建一个包含替换文本的文件。例如：

```
$ cat sentence.txt
Linux refers to the family of Unix-like computer operating systems that use the Linux
kernel. Linux can be installed on a wide variety of computer hardware, ranging from
mobile phones, tablet computers and video game consoles, to mainframes and supercomputers.
Linux is predominantly known for its use in servers.
```

我们的目标是移除包含“mobile phones”的句子。可以用下面的`sed`语句来实现：

```
$ sed 's/ [^.]*mobile phones[^.]*\./g' sentence.txt

Linux refers to the family of Unix-like computer operating systems
that use the Linux kernel. Linux is predominantly known for its use
in servers.
```



这里假设没有句子分散在不同的行中。也就是说，句子总是在同一行中起止。

### 4.14.3 工作原理

让我们分析一下sed的正则表达式's/[^.]\*mobile phones[.]\*\./g'。该正则表达式的格式为：'s/匹配模式/替代字符串/g'。它将与匹配样本相匹配的每一处内容都用替代字符串进行替换。

这里的匹配模式是用来匹配整句文本的正则表达式。文件中的每一句话第一个字符都是空格，句与句之间都以“.”来分隔。因此我们需要匹配内容的格式就是：空格+若干文本+需要匹配的字符串+若干文本+句点。一个句子除了作为定界符的句点之外，可以包含任意字符。因此我们要使用[.]. [^.]\*可以匹配除句点之外的任何字符的组合。用来匹配文本的“mobile phone”被放置在两个[.]\*之间。每一个匹配的句子均被//替换（注意，/与/之间没有任何内容）。

4

### 4.14.4 参考

- 4.2节讲解了如何使用正则表达式。
- 4.5节讲解了sed命令。

## 4.15 对目录中的所有文件进行文本替换

我们经常需要将某个目录中的所有文件内的一部分文本替换成另一部分。例如在网站的源文件目录中替换一个URI。解决这个问题最快的方法就是利用shell脚本。

### 4.15.1 实战演练

就目前我们所掌握的知识，首先使用find定位需要进行文本替换的文件。然后使用sed进行实际的替换操作。

假设我们希望将所有.cpp文件中的Copyright替换成Copyleft：

```
$ find . -name *.cpp -print0 | xargs -I{} -0 sed -i 's/Copyright/Copyleft/g' {}
```

### 4.15.2 工作原理

我们使用find在当前目录下查找所有的.cpp文件，然后使用print0打印出以null字符（\0）

作为分隔符的文件列表。(这可以避免文件名中的空格所带来的麻烦。)接着使用通道将列表传递给xargs,后者将对应的文件作为sed的参数,由sed对文件内容进行修改。

### 4.15.3 补充内容

回忆一下,find有一个选项-exec,它可以对每个find查找到的文件执行命令。我们可以使用该选项实现同样的效果:

```
$ find . -name *.cpp -exec sed -i 's/Copyright/Copyleft/g' \{\} \;
```

或者

```
$ find . -name *.cpp -exec sed -i 's/Copyright/Copyleft/g' \{\} \+
```

尽管这两个命令作用相同,但第一个命令会为每个查找到的文件调用一次sed,而在第二个命令中,find会将多个文件名一并传递给sed。

## 4.16 文本切片及参数操作

这则攻略讲述了一些简单的文本替换技术以及Bash中可用的参数扩展简写方法。这些简单的技巧通常能够让我们免于键盘敲击之苦。

### 4.16.1 实战演练

让我们来练练手吧。

替换变量内容中的部分文本:

```
$ var="This is a line of text"
$ echo ${var/line/REPLACED}
This is a REPLACED of text"
```

line被替换成REPLACED。

我们可以通过指定字符串的起始位置和长度来生成子串,语法如下:

```
${variable_name:start_position:length}
```

用下面的命令可以打印第5个字符之后的内容:

```
$ string=abcdefghijklmnpqrstuvwxyz
$ echo ${string:4}
efghijklmnpqrstuvwxyz
```

从第5个字符开始，打印8个字符：

```
$ echo ${string:4:8}
efghijkl
```

起始字符的索引从0开始计数。我们也可以从后向前计数，将最后一个字符索引记为 -1。但如果使用负数作为索引值，则必须将负数放入括号内。(-1)就是最后一个字符的索引。

```
echo ${string:(-1)}
z
$ echo ${string:(-2):2}
yz
```

#### 4.16.2 参考

4.5节讲解了从单词中切分字符。

### 本章内容

- ❑ Web页面下载
- ❑ 以纯文本形式下载网页
- ❑ cURL入门
- ❑ 从命令行访问Gmail
- ❑ 解析网站数据
- ❑ 图片抓取器及下载工具
- ❑ 网页相册生成器
- ❑ Twitter命令行客户端
- ❑ 基于Web后端的定义查询工具
- ❑ 查找网站中的无效链接
- ❑ 跟踪网站变更
- ❑ 以POST方式发送网页并读取响应

## 5.1 入门

Web正在成为反映技术发展的晴雨表和数据处理中心。尽管shell脚本没法像PHP一样在Web上大包大揽，但还是有不少活儿挺适合它。本章会研究一些用于解析网站内容、下载数据、发送数据表单以及网站维护任务自动化之类的例子。我们可以用短短几行脚本就将很多原本需要通过浏览器交互进行的活动自动化。借助命令行工具，利用HTTP协议所提供的功能，我们可以用脚本解决大部分Web自动化的问题。来尽情享受接下来的内容吧。

## 5.2 Web 页面下载

从给定的URL中下载文件或页面很简单。一些命令行下载工具就可以完成这项任务。

### 5.2.1 预备知识

wget是一个用于文件下载的命令行工具，选项繁多且用法灵活。

## 5.2.2 实战演练

用wget可以下载网页或远程文件：

```
$ wget URL
```

例如：

```
$ wget http://slynux.org
--2010-08-01 07:51:20-- http://slynux.org/
Resolving slynux.org... 174.37.207.60
Connecting to slynux.org|174.37.207.60|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 15280 (15K) [text/html]
Saving to: "index.html"

100%[=====] 15,280      75.3K/s   in 0.2s

2010-08-01 07:51:21 (75.3 KB/s) - "index.html" saved [15280/15280]
```

可以指定从多个URL处进行下载：

```
$ wget URL1 URL2 URL3 ..
```

## 5.2.3 工作原理

5

通常下载的文件名和URL中的文件名会保持一致，下载日志或进度被写入stdout。

你可以通过选项-o指定输出文件名。如果存在同名文件，那么会先将该同名文件清空（truncate）再写入下载文件。

也可以用选项-o指定一个日志文件，这样日志信息就不会被打印到stdout了。

```
$ wget ftp://example_domain.com/somefile.img -O dloaded_file.img -o log
```

运行该命令，屏幕上不会出现任何内容。日志或进度信息被写入文件log，输出文件为dloaded\_file.img。

由于不稳定的互联网连接，下载有可能被迫中断。我们可以将重试次数作为命令参数，这样一旦下载中断，wget在放弃之前还会继续进行多次尝试。

用-t指定重试次数：

```
$ wget -t 5 URL
```

或者要求wget不停地重试：

```
$ wget -t 0 URL
```

### 5.2.4 补充内容

wget还有一些其他选项，能够用以应对不同的情况。来看看其中的一部分。

#### 1. 下载限速

当我们的下载带宽有限，却又有多个应用程序共享互联网连接时，进行大文件下载往往会榨干所有的带宽，严重阻滞其他进程。wget命令有一个内建的选项可以限定下载任务能够占有的最大带宽，从而保证其他应用程序流畅运行。

我们可以按照下面的方式使用`--limit-rate`对wget进行限速：

```
$ wget --limit-rate 20k http://example.com/file.iso
```

在命令中用k（千字节）和m（兆字节）指定速度限制。

还可以指定最大下载配额（quota）。配额一旦用尽，下载随之停止。在下载多个文件时，对总下载量进行限制是有必要的，这能够避免在无意中占用过多磁盘空间。

使用`--quota`或`-Q`选项：

```
$ wget -Q 100m http://example.com/file1 http://example.com/file2
```

#### 2. 断点续传

如果使用wget进行的下载在完成之前被中断，可以利用选项`-c`从断点开始继续下载：

```
$ wget -c URL
```

#### 3. 复制整个网站（镜像）

wget有一个选项可以使其像爬虫一样以递归的方式遍历网页上所有的URL链接，并逐个下载。这样一来，我们就能够获得一个网站的所有页面。

要实现这个任务，可以按照下面的方式使用选项`--mirror`：

```
$ wget --mirror --convert-links exampledomain.com
```

或者

```
$ wget -r -N -l -k DEPTH URL
```

`-l`指定页面层级。这意味着wget只会向下遍历指定的页面级数。该选项要与`-r`（recursive，递归选项）一同使用。另外，`-N`表示使用文件的时间戳。URL表示欲下载的网站起始地址。`-k`或`-convert-links`指示wget将页面的链接地址转换为本地地址。



对网站进行镜像时，请三思而行。除非你被许可，否则只应出于个人使用的目的才可以这么做，而且不应该频繁地进行。

#### 4. 访问需要认证的HTTP或FTP页面

一些网页需要HTTP或FTP认证，可以用`--user`和`--password`提供认证信息：

```
$ wget --user username --password pass URL
```

也可以不在命令行中指定密码，而由网页提示并手动输入密码，这就需要将`--password`改为`--ask-password`。

### 5.3 以纯文本形式下载网页

网页其实就是包含HTML标签和其他诸如Javascript、CSS等元素的HTML页面。HTML标记是网页的基础。我们也许需要解析网页来查找特定的内容，这时Bash就能派上用场了。当下载网页时，实际接收到的就是一个HTML文件。这种格式化的数据需要用浏览器查看。

在大多数情况下，解析文本文件要比解析HTML数据来得容易，因为无需再去剥离HTML标签。Lynx是一款颇有玩头的基于命令行的Web浏览器，可以利用它获取纯文本形式的网页。来看看这一切该如何实现。

5

#### 实战演练

用lynx命令的`-dump`选项将网页内容以ASCII编码的形式存储到文本文件中：

```
$ lynx URL -dump > webpage_as_text.txt
```

这个命令会将页面中所有的超链接（`<a href="link">`）作为文本文件的页脚，单独放置在标题为References文本区域。这就省得我们再用正则表达式解析链接了。

例如：

```
$lynx -dump http://google.com > plain_text_page.txt
```

你可以用`cat`命令查看纯文本形式的网页：

```
$ cat plain_text_page.txt
```

### 5.4 cURL 入门

作为一款强力工具，cURL支持包括HTTP、HTTPS、FTP在内的众多协议。它还支持POST、

cookie、认证、从指定偏移处下载部分文件、参照页（referer）、用户代理字符串、扩展头部、限速、文件大小限制、进度条等特性。如果要和网页访问序列（web page usage sequence）以及数据检索自动化打交道，那么cURL定能助你一臂之力。这则攻略将为你展示cURL一系列最为重要的特性。

### 5.4.1 预备知识

在默认情况下，主流Linux发行版中并没有包含cURL，你得使用包管理器进行安装。不过多数发行版都默认附带了wget。

cURL通常将下载文件输出到stdout，将进度信息输出到stderr。要想避免显示进度信息，请使用--silent选项。

### 5.4.2 实战演练

curl命令的用途广泛，诸如下载、发送各种HTTP请求、指定HTTP头部都不在话下。来看看cURL如何做到这一切。

- ❑ 将下载的文件输出到终端（所下载的数据被写入stdout），使用下列命令：

```
$ curl URL
```

- ❑ 要避免curl命令显示进度信息，可以使用--silent选项：

```
$ curl URL --silent
```

- ❑ 选项 -o表明将下载数据写入文件，而非标准输出中。该文件采用的是从URL中解析出的文件名：

```
$ curl URL --silent -o
```

- ❑ 如果需要在下载过程中显示形如 # 的进度条，用 --progress代替 --silent。

```
$ curl http://slynux.org -o index.html --progress
##### 100.0%
```

### 5.4.3 工作原理

cURL如果不将网页或文件写入stdout，就会将其写入和URL中所指定的文件名称相同的文件中。如果在URL中找不到文件名，则会产生错误。因此要确保URL指向的是远程文件。curl http://slynux.org -o -silent就会显示错误信息，这是因为无法从URL中解析出文件名。这种情况下，我们可以使用-o选项来手动指定文件名：

```
$ curl URL --silent -o new_filename
```

### 5.4.4 补充内容

在前面我们学习了如何下载文件以及将HTML页面打印到终端。cURL还包括一些高级选项，让我们接着进行研究。

#### 1. 断点续传

cURL能够从特定的文件偏移处继续下载。可以通过指定一个偏移量来下载部分文件。

```
$ curl URL/file -C offset
```

偏移量是以字节为单位的整数。如果只是想断点续传，那么cURL不需要指定准确的字节偏移。要是你希望cURL推断出正确的续传位置，请使用选项 `-C -`，就像这样：

```
$ curl -C - URL
```

cURL会自动计算出应该从哪里开始续传。

#### 2. 用cURL设置参照页字符串

参照页（`referer`）<sup>①</sup>是位于HTTP头部中的一个字符串，用来标识用户是从哪个页面到达当前页面的。如果用户点击了网页A中的某个链接，那么用户就会转到网页B，网页B头部的参照页字符串会包含网页A的URL。

一些动态网页会在返回HTML页面前检测参照页字符串。例如，如果用户是通过Google搜索来到了当前网页，网页上会附带显示一个Google的logo；如果用户是通过手动输入URL来到当前网页，则显示另一个不同的页面。

网站的作者可以根据条件进行判断：如果参照页是`www.google.com`，那么就返回一个Google页面，否则返回其他页面。

可以用curl命令的 `--referer` 选项指定参照页字符串：

```
$ curl --referer Referer_URL target_URL
```

例如：

```
$ curl --referer http://google.com http://slynux.org
```

#### 3. 用cURL设置cookie

我们可以用curl来指定并存储HTTP操作过程中使用到的cookie。

---

<sup>①</sup> `referer`其实应该是英文单词`referrer`，不过拼错的人太多了，所以编写标准的人也就将错就错了。

要指定cookie，使用 `--cookie "COOKIES"` 选项。

cookies需要以`name=value`的形式来给出。多个cookie之间使用分号分隔。例如：

```
$ curl http://example.com --cookie "user=slynux;pass=hack"
```

如果要将cookie另存为一个文件，使用 `--cookie-jar` 选项。例如：

```
$ curl URL --cookie-jar cookie_file
```

#### 4. 用cURL设置用户代理字符串

如果不指定用户代理（`user agent`），一些需要检验用户代理的网页就无法显示。你肯定碰到过有些陈旧的网站只能在Internet Explorer（IE）下正常工作。如果使用其他浏览器，这些网站就会提示说它只能用IE访问。这是因为这些网站检查了用户代理。你可以用`curl`来设置用户代理。

cURL的 `--user-agent` 或 `-A` 选项用于设置用户代理：

```
$ curl URL --user-agent "Mozilla/5.0"
```

其他HTTP头部信息也可以通过cURL来发送。用 `-H` 头部信息"传递多个头部信息。例如：

```
$ curl -H "Host: www.slynux.org" -H "Accept-language: en" URL
```



各类网页浏览器和爬虫使用了很多不同的用户代理字符串。你可以在这里找到其中的一部分：<http://www.useragentstring.com/pages/useragentstring.php>。

#### 5. 限定cURL可占用的带宽

如果带宽有限，又有多个用户共享，为了平稳流畅地分享带宽，我们可以用 `--limit-rate` 限制cURL的下载速度：

```
$ curl URL --limit-rate 20k
```

在命令中用k（千字节）和m（兆字节）指定下载速度限制。

#### 6. 指定最大下载量

可以用 `--max-filesize` 选项指定可下载的最大文件大小：

```
$ curl URL --max-filesize bytes
```

如果文件大小超出限制，命令返回一个非0的退出码。如果命令正常运行，返回0。

## 7. 用cURL进行认证

可以用cURL的选项 `-u` 完成HTTP或FTP认证。

`-u username:password` 可用来指定用户名和密码。它也可以不指定密码，而在后续的执行过程中按照提示输入密码。例如：

```
$ curl -u user:pass http://test_auth.com
```

如果你喜欢经提示后输入密码，只需要使用 `-u username` 即可。例如：

```
$ curl -u user http://test_auth.com
```

## 8. 只打印响应头部信息（不包括数据部分）

只打印响应头部（response header）有助于进行各种检查或统计。例如，如果要检查某个页面是否能够打开，并不需要下载整个页面内容。只用读取HTTP响应头部就能够知道这个页面是否可用。

检查HTTP头部的一个用法就是在下载之前先获知文件大小。我们可以在下载之前，通过检查HTTP头部中的 `Content-Length` 参数来得知文件的长度。同样还可以从头部检索出其他一些有用的参数。`Last-Modified` 参数能告诉我们远程文件最后的改动时间。

通过 `-I` 或 `--head` 就可以只打印HTTP头部信息，而无须下载远程文件。例如：

```
$ curl -I http://slynux.org
HTTP/1.1 200 OK
Date: Sun, 01 Aug 2010 05:08:09 GMT
Server: Apache/1.3.42 (Unix) mod_gzip/1.3.26.1a mod_log_bytes/1.2 mod_bwlimited/1.4
mod_auth_passthrough/1.8 FrontPage/5.0.2.2635 mod_ssl/2.8.31 OpenSSL/0.9.7a
Last-Modified: Thu, 19 Jul 2007 09:00:58 GMT
ETag: "17787f3-3bb0-469f284a"
Accept-Ranges: bytes
Content-Length: 15280
Connection: close
Content-Type: text/html
```

## 5.4.5 参考

参考5.13节。

## 5.5 从命令行访问 Gmail

Gmail（<http://mail.google.com>）是来自Google的一项被广泛使用的免费电子邮件服务。你可

以使用经过认证的RSS feed来读取个人邮件。我们也能够使用发件人姓名以及邮件主题来解析RSS feed。这样便无须打开网页浏览器就能够查看收件箱中的未读邮件。

### 5.5.1 实战演练

来看下面这个脚本文件，它的作用是通过解析Gmail的RSS feed来显示未读的邮件：

```
#!/bin/bash
#用途： 获取Gmail工具

username='PUT_USERNAME_HERE'
password='PUT_PASSWORD_HERE'

SHOW_COUNT=5 # 需要显示的未读邮件数量

echo
curl -u $username:$password --silent "https://mail.google.com/mail/feed/atom" | \
tr -d '\n' | sed 's:</entry>:\n:g' | \
sed -n 's/.*<title>(.*)</title>.*<author><name>\[^\<]*\]</name><email> \
\[^\<]*\].*/From: \2 [\3] \nSubject: \1\n/p' | \
head -n $(( $SHOW_COUNT * 3 ))
```

输出如下：

```
$ ./fetch_gmail.sh
From: SLYNUX [ slynux@slynux.com ]
Subject: Book release - 2

From: SLYNUX [ slynux@slynux.com ]
Subject: Book release - 1
.
.....共5封未读邮件
```



如果你的Gmail使用双重身份认证，那就必须为此脚本生成一个新建，并使用。你的普通密码不会起作用。

### 5.5.2 工作原理

这个脚本通过用户认证来使用cURL来下载RSS feed。用户认证信息由-u username:password参数提供。也可以只使用-u user，这样就不用事先提供密码，而是由cURL在运行时以交互的方式索要密码。我们将管道命令拆分成不同的部分来解释这个脚本的工作原理。

tr -d '\n'移除了所有的换行符，这样我们就可以将\n作为定界符来重建邮件项。

`sed 's:</entry>:\n:g'`将每一处 `</entry>` 替换成换行符,以保证每一条邮件项独立成行,以便逐条解析邮件。通过查看<https://mail.google.com/mail/feed/atom>页面源代码中用于RSS feed的XML标记,我们可以看出每一条邮件项都对应于 `<entry> TAGS </entry>`。

该脚本接下来的部分如下:

```
sed 's/.*<title>\(.*\)</title.*<author><name>\([^<]*\)</name><email>
\([^<]*\).*/Author: \2 [\3] \nSubject: \1\n/'
```

脚本用`<title>\(.*\)</title>`匹配邮件标题,用`<author><name>\([^<]*\)</name>`匹配发件人姓名,`<email>\([^<]*\)`匹配发件人电子邮件地址。然后使用反向引用:

```
Author: \2 [\3] \nSubject: \1\n
```

用来将匹配内容以易于阅读的格式来替换原先的邮件项。`\1`对应于第一处匹配,`\2`对应于第二处匹配,依次类推。

`SHOW_COUNT=5`用来设置需要在终端中显示的未读邮件数量。

`head`用来显示`SHOW_COUNT*3`<sup>①</sup>行文本。`SHOW_COUNT`乘以3是因为每一封未读邮件的相关信息需要占用3行。

### 5.5.3 参考

- 5.4节讲解了`curl`命令。
- 4.5节讲解了`sed`命令。

## 5.6 解析网站数据

消除不必要的繁枝末节有助于解析网页数据。`sed`和`awk`是完成这项工作的主要工具。在第4章有关`grep`的攻略中,我们见到过一份演员评级列表。那个列表就是通过解析<http://www.johntorres.net/BoxOfficefemaleList.html>得到的。

让我们看看如何使用文本处理工具来解析同样的数据。

### 5.6.1 实战演练

请看下面用于从网站解析演员详细信息的命令:

```
$ lynx -dump -nolist http://www.johntorres.net/BoxOfficefemaleList.html | \
grep -o "Rank-.*" | \
sed -e 's/ *Rank-\([0-9]*\) *\([^\\]\)/\1\t\2/' | \
```

① 这里的`SHOW_COUNT*3`行文本并不包括脚本开始部分由`echo`生成的那一行(空行)。

```
sort -nk 1 > actresslist.txt
```

输入如下：

```
# 由于版面空间限制，只显示前3位演员的信息
1 Keira Knightley
2 Natalie Portman
3 Monica Bellucci
```

### 5.6.2 工作原理

Lynx是一个基于命令行的网页浏览器。它并不会输出一堆原始的HTML代码，而是能够显示网站的文本版本，这个文本版和我们在浏览器中看到的页面一模一样。这样一来，就免去了移除HTML标签的工作。这里用到了lynx的-nolist选项，这是因为不需要给每个链接自动加上数字标号。按照下面的方法用sed解析并格式化以Rank开头的行：

```
sed -e 's/ *Rank-\([0-9]*\) *\(.*)/\1\t\2/'
```

然后再根据评级情况对这些行进行排序。

### 5.6.3 参考

- ❑ 4.5节讲解了sed命令。
- ❑ 5.3节讲解了lynx命令。

## 5.7 图片抓取器及下载工具

当需要下载某个网页上所有的图片时，图片抓取器（image crawler）能够帮上我们的大忙。用不着把HTML源码翻个底朝天来摘出所有的图片，我们可以用脚本解析图像文件并将它们自动下载下来。来看看这是如何实现的。

### 5.7.1 实战演练

用于从网页上抓取并下载图片的Bash脚本如下：

```
#!/bin/bash
#用途：图片下载工具
#文件名：img_downloader.sh

if [ $# -ne 3 ];
then
    echo "Usage: $0 URL -d DIRECTORY"
    exit -1
fi
```

```

for i in {1..4}
do
    case $1 in
        -d) shift; directory=$1; shift ;;
        *) url=${url:-$1}; shift;;
    esac
done

mkdir -p $directory;
baseurl=$(echo $url | egrep -o "https?://[a-z.]+")

echo Downloading $url
curl -s $url | egrep -o "<img src=[^>]*>" |
sed 's/ /tmp/$$.list

sed -i "s|^|/$baseurl/" /tmp/$$.list

cd $directory;

while read filename;
do
    echo Downloading $filename
    curl -s -O "$filename" --silent

done < /tmp/$$.list

```

使用方法:

```
$ ./img_downloader.sh http://www.flickr.com/search/?q=linux -d images
```

## 5.7.2 工作原理

上述图片下载器脚本首先解析HTML页面，除去 <img> 之外的所有标签，然后从 <img> 标签中解析出src="URL" 并将图片下载到指定的目录中。这个脚本接受一个网页URL和用于存放图片的目录路径作为命令行参数。[ \$# -ne 3 ] 用于检查脚本参数数量是否为3个。如果不是，它就会退出运行并返回脚本用法说明。

如果参数是3个，那么就解析URL和目标目录。实现方法如下：

```

while [ -n "$1" ]
do
    case $1 in
        -d) shift; directory=$1; shift ;;
        *) url=${url:-$1}; shift;;
    esac
done

```

这里用到了while循环。有几个参数，它就循环几次。case语句会对第一个参数（\$1）求值，以便匹配-d或是其他需要进行检查的参数。采用这种方法来解析命令行参数的好处在于，可以将-d置于命令行中的任意位置：

```
$ ./img_downloader.sh -d DIR URL
```

或者

```
$ ./img_downloader.sh URL -d DIR
```

shift用来向左移动参数。当使用shift后，\$2的值就被赋给\$1；如果再次使用shift，则\$3的值被赋给\$1，往后依次类推，因此通过\$1就可以对所有的参数进行求值。

如果匹配-d，显然接下来的参数就是目标目录。\*)对应默认匹配（default match）。它能够匹配除了-d之外的任何内容。因此在默认匹配中，如果\$1=""或\$1=URL，就需要采用\$1=URL，避免""将变量url覆盖掉，因此我们使用了url=\${url:-\$1}。如果url不为空，它会返回URL值，否则返回\$1的值。

egrep -o "<img src=[^>]\*>"只打印包括属性值在内的<img> 标签。[>]\*用来匹配除>之外的所有字符，结果就是。

sed 's/ 标签中得到所有图像文件的URL。

图像文件源路径有两种类型：相对路径和绝对路径。绝对路径包含以http://或https://起始的完整URL，相对路径则以/或图像文件名起始。绝对路径例如：http://example.com/image.jpg；相对路径例如：/image.jpg。

对于以/起始的相对路径，应该用基址URL（base URL）把它转换为http://example.com/image.jpg。我们一开始通过解析将基址URL存入变量baseurl中，然后用"s|^/|\$baseurl/|" /tmp/\$\$.list将所有的/替换成baseurl。

```
"s|^/|$baseurl/|" /tmp/$$.list
```

while循环用来对图片的URL列表进行逐行迭代，并用curl下载图像文件。与curl一同使用的--silent 选项可避免下载进度信息出现在屏幕上。

### 5.7.3 参考

- 5.4节讲解了curl命令。
- 4.5节讲解了sed命令。
- 4.3节讲解了grep命令。

## 5.8 网页相册生成器

Web开发人员通常都会为网站设计相册页面，这些页面上包含着多个图像缩略图。点击缩略图，就会出现一幅放大的图片。但如果需要很多图片，每一次都得复制 `<img>` 标签、调整图片大小来创建缩略图、把调整好的图片放进缩略图目录，诸如此类的活儿实在烦琐。我们可以写一个简单的Bash脚本将这些重复的工作以自动化的方式来处理。这样一来，创建缩略图、将缩略图放入对应的目录、生成对应于`<img>`标签的代码片段都可以在短短几秒钟内自动搞定。这则攻略正是要告诉你如何实现刚才所说的一切。

### 5.8.1 预备知识

要完成这项任务，可以通过for循环对当前目录下的所有图片进行迭代。这需要借助一些常见的Bash工具，如cat和convert（来自Image Magick软件包）。我们将用所有的图片生成一个HTML相册index.html（见图5-1）。



图 5-1

### 5.8.2 实战演练

生成HTML相册页面的Bash脚本如下：

```
#!/bin/bash
#文件名：generate_album.sh
#用途：用当前目录下的图片创建相册
```

```
echo "Creating album.."
mkdir -p thumbs
cat <<EOF1 > index.html
<html>
<head>
<style>

body
{
    width:470px;
    margin:auto;
    border: 1px dashed grey;
    padding:10px;
}

img
{
    margin:5px;
    border: 1px solid black;

}
</style>
</head>
<body>
<center><h1> #Album title </h1></center>
<p>
EOF1

for img in *.jpg;
do
    convert "$img" -resize "100x" "thumbs/$img"
    echo "<a href=\"$img\" ><img src=\"$thumbs/$img\" title=\"$img\" /></a>" >>
index.html
done

cat <<EOF2 >> index.html

</p>
</body>
</html>
EOF2

echo Album generated to index.html
```

运行脚本：

```
$ ./generate_album.sh
Creating album..
Album generated to index.html
```

### 5.8.3 工作原理

脚本的起始部分用于生成HTML页面的头部。

接下来，脚本将一直到EOF1的这部分内容（不包括EOF1）重定向到index.html：

```
cat <<EOF1 > index.html
contents...
EOF1
```

页面头部包括HTML和CSS样式表单。

for img in \*.jpg;对每一个文件进行迭代，并执行相应的操作。

convert "\$img" -resize "100x" "thumbs/\$img"将创建宽度为100像素的图像缩略图。

下面的语句会生成所需的<img>标记并将其追加到index.html中：

```
echo "<a href=\"\$img\" ><img src=\"thumbs/\$img\" title=\"\$img\" /></a>" >> index.html
```

最后再用cat添加HTML页脚。

### 5.8.4 参考

1.6节讲解了EOF和stdin重定向。

5

## 5.9 Twitter 命令行客户端

Twitter不仅是最流行的微博平台，同时也是最时髦的在线社交媒体。我们可以使用Twitter API从命令行中读取自己发布的微博。来看看如何实现。

### 5.9.1 预备知识

最近Twitter已经不再允许用户使用普通的HTTP认证（plain HTTP Authentication）进行登录了，我们必须使用OAuth进行自认证（authenticate ourselves）。完整地讲解OAuth超出了本书的范围，因此我们会利用一个代码库以便在Bash脚本中可以轻松使用OAuth。

- (1) 从<https://github.com/livibetter/bash-oauth/archive/master.zip>处下载bash-oauth库，将其解压缩到任意目录中。
- (2) 进入该目录中的bash-oauth-master子目录，以root身份执行make install-all。
- (3) 进入<https://dev.twitter.com/apps/new>注册新的应用，以便能够使用OAuth，如图5-2所示。

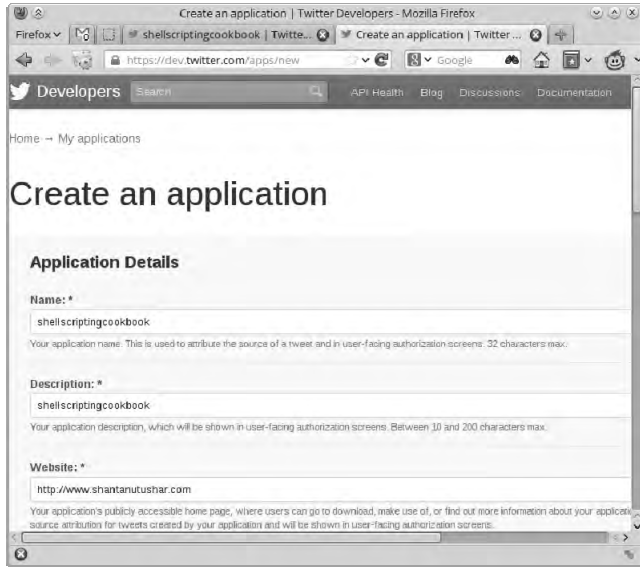


图 5-2

(4) 注册完新的应用之后，进入应用设置，将**Access type**更改为**Read and Write**，如图5-3所示。

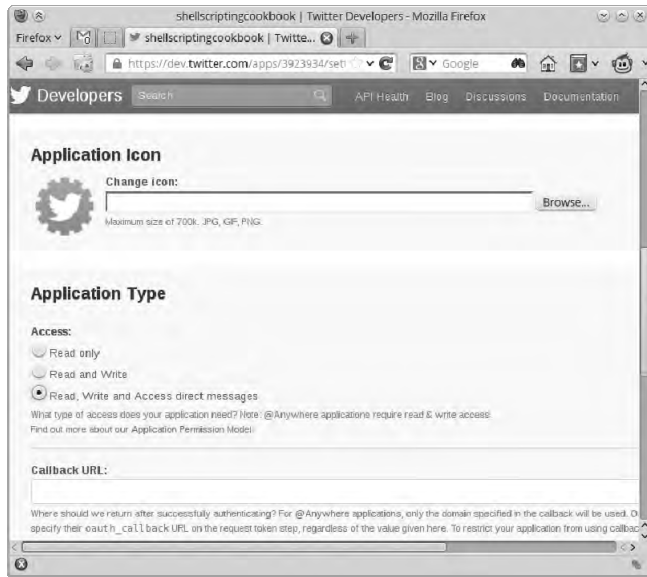


图 5-3

(5) 进入应用的Details部分，注意两个地方：Consumer Key和Consumer Secret，以便在脚本中对其进行替换，如图5-4所示。

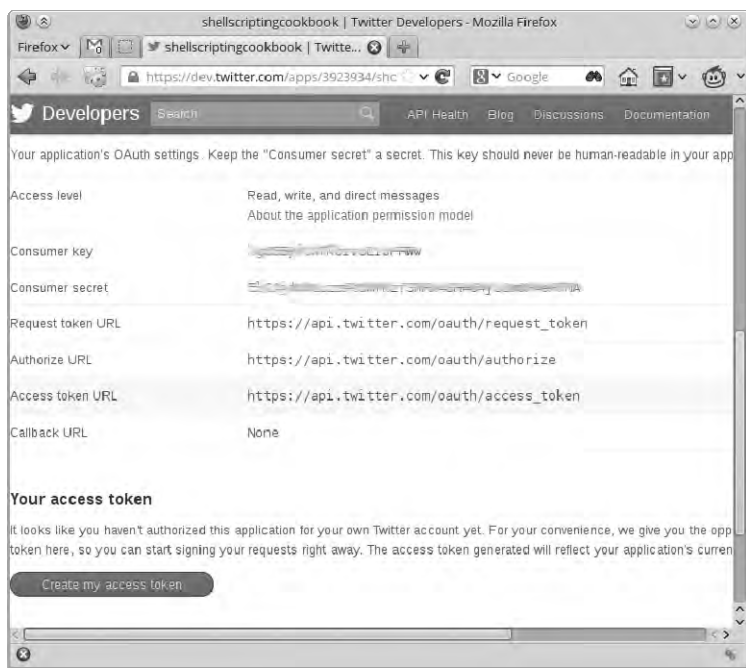


图 5-4

好，接下来就该编写脚本了。

## 5.9.2 实战演练

接下来编写使用代码库的Bash脚本：

```
#!/bin/bash
#文件名: twitter.sh
#用途: twitter客户端基本版

oauth_consumer_key=YOUR_CONSUMER_KEY
oauth_consumer_secret=YOUR_CONSUMER_SECRET
config_file=~/.${oauth_consumer_key}-${oauth_consumer_secret}-rc

if [[ "$1" != "read" ]] && [[ "$1" != "tweet" ]];
then
```

```
    echo -e "Usage: $0 tweet status_message\n    OR\n    $0 read\n"
    exit -1;
fi

source TwitterOAuth.sh
TO_init

if [ ! -e $config_file ]; then
    TO_access_token_helper
    if (( $? == 0 )); then
        echo oauth_token=${TO_ret[0]} > $config_file
        echo oauth_token_secret=${TO_ret[1]} >> $config_file
    fi
fi

source $config_file

if [[ "$1" = "read" ]];
then
    TO_statuses_home_timeline '' 'shantanutushar' '10'
    echo $TO_ret | sed 's/<\([a-z]\)\n<\1/g' | \
grep -e '^<text>' -e '^<name>' | sed 's/<name>/\ - by /g' | \
sed 's$</*[a-z]*>$g'

elif [[ "$1" = "tweet" ]];
then
    shift
    TO_statuses_update '' "$@"
    echo 'Tweeted :)'
fi
```

运行脚本：

```
./twitter.sh read
Please go to the following link to get the PIN:
https://api.twitter.com/oauth/authorize?oauth_token=GaZcfsdnhMO4HiBQuUTdeLJAzeaUam
nOljWGnU
PIN: 4727143
Now you can create, edit and present Slides offline.
- by A Googler
./twitter.sh tweet "I am reading Packt Shell Scripting Cookbook"
Tweeted :)
./twitter.sh read | head -2
I am reading Packt Shell Scripting Cookbook
- by Shantanu Tushar Jha
```

### 5.9.3 工作原理

首先,使用`source`命令将`TwitterOAuth.sh`库包含在脚本中,这样就可以利用其中定义好的函数访问Twitter了。我们需要调用库函数`TO_init`,以便库完成自身的初始化操作。

所有的应用在用户首次使用的时候都需要获取一个OAuth令牌(`token`)以及令牌密钥(`token secret`)。我们需要检查是否已经获取。如果没有,则调用库函数`TO_access_token_helper`。得到之后,将其保存在`config`文件中,以后再执行脚本时,只需对该文件执行`source`命令就可以了。

我们使用库函数`TO_statuses_home_timeline`获取Twitter中发布的内容,该函数会读取Twitter,将得到的XML文件内容存入变量`TO_ret`,然后使用`sed`为每一个标签(`tag`)加上一个换行符,过滤出`<text>`和`<name>`标签,最后再将这些标签删除,替换成便于用户阅读的文本。

要发布新的微博,可以使用库函数`TO_statuses_update`。如果该函数的第一个参数为空,则表明使用默认格式,要发布的内容可以作为第二个参数传递给函数。

### 5.9.4 参考

- 5.4节讲解了`sed`命令。
- 4.3节讲解了`grep`命令。

## 5.10 基于 Web 后端的定义查询工具

网上有大量提供了API的词典工具,利用这些API可以获得能被机器读取(`machine-readable`)的词汇定义。让我们来使用其中的一种API来编写一个词汇定义查询脚本。

### 5.10.1 预备知识

我们用`curl`、`sed`和`grep`来编写词汇查询工具。有关词典的网站数不胜数,你可以注册并免费使用它们的API(限于个人用途)。在这里,我们使用Merriam-Webster的词典API。请按照下列步骤执行。

- (1) 进入<http://www.dictionaryapi.com/register/index.htm>注册账户。选择Collegiate Dictionary和Learner's Dictionary,如图5-5所示。



图 5-5

- (2) 使用新创建的用户登录，进入My Keys获取密钥。记下Learner's Dictionary的密钥，如图5-6所示。



图 5-6

## 5.10.2 实战演练

来看下面这段定义工具脚本：

```
#!/bin/bash
#文件名: define.sh
#用途:用于从 dictionaryapi.com 获取词汇定义

apikey=YOUR_API_KEY_HERE

if [ $# -ne 2 ];
then
    echo -e "Usage: $0 WORD NUMBER"
    exit -1;
fi

curl --silent
http://www.dictionaryapi.com/api/v1/references/learners/xml/$1?key=$apikey | \
grep -o \<dt\>.*\</dt\> | \
sed 's$< /*[a-z]*>$g' | \
head -n $2 | nl
```

运行脚本：

```
$ ./define.sh usb 1
1 :a system for connecting a computer to another device (such as a printer,
keyboard, or mouse) by using a special kind of cord a USB cable/port USB is an
abbreviation of "Universal Serial Bus."How it works...
```

5

## 5.10.3 工作原理

我们通过指定API key (\$apikey) 来使用curl从词典API页面获取数据以及需要查找定义的词汇(\$1)。包含定义的查询结果位于<tag>标签中，可以使用grep来将其选中，然后使用sed删除标签。接着从定义中提取所需要的行数，利用nl在行前加上行号。

## 5.10.4 参考

- 4.5节讲解了sed命令。
- 4.3节讲解了grep命令。

## 5.11 查找网站中的无效链接

一些人采用人工方式来检查网站上的每一个页面，以便找出无效的链接。这对于页面不多的

小网站来说还算可行。但是随着页面数量增多，这种方式就不现实了。要是能够将查找无效链接的工作自动化，那可轻松太多了。我们可以利用HTTP处理工具来查找无效的链接。来看看这是怎样实现的。

### 5.11.1 预备知识

要识别链接并从中找出无效链接，我们可以使用lynx和curl。lynx有一个选项-traversal，能够以递归方式访问网站页面并建立网站中所有超链接的列表。我们可以用curl验证每一个链接的有效性。

### 5.11.2 实战演练

利用curl查找网页上无效链接的Bash脚本如下：

```
#!/bin/bash
#文件名: find_broken.sh
#用途: 查找网站中的无效链接

if [ $# -ne 1 ];
then
    echo -e "$Usage: $0 URL\n"
    exit 1;
fi

echo Broken links:

mkdir /tmp/$$lynx
cd /tmp/$$lynx

lynx -traversal $1 > /dev/null
count=0;

sort -u reject.dat > links.txt

while read link;
do
    output=`curl -I $link -s | grep "HTTP/*OK"`;
    if [[ -z $output ]];
    then
        echo $link;
        let count++
    fi
done < links.txt

[ $count -eq 0 ] && echo No broken links found.
```

### 5.11.3 工作原理

`lynx -traversal URL`会在工作目录下生成数个文件，其中包括`reject.dat`，该文件包含网站中的所有链接。`sort -u`用来建立一个不包含重复项的列表。然后我们迭代每一个链接，并通过`curl -I`检验接收到的响应头部。如果响应头部的第一行包含`HTTP/1.0 200 OK`，就表示该链接正常。显示其他响应信息则表示该链接失效，并将其输出到屏幕。



从名称上来看，`reject.dat`中包含的应该是无效URL的列表。但其实并非如此，`lynx`是将所有的URL全都放到了这个文件中。

`lynx`还生成了一个名为`traverse.errors`的文件，其中包含了所有在浏览过程中存在问题的URL。但是`lynx`只会将返回HTTP 404(not found)的URL，因此会遗漏那些存在其他类型错误的URL（例如HTTP 403 Forbidden）。这就是为什么要手动检查返回状态的原因。

### 5.11.4 参考

- 5.3节讲解了`lynx`命令。
- 5.4节讲解了`curl`命令。

## 5.12 跟踪网站变动

对于Web开发人员和用户来说，跟踪网站的变动情况不无益处。但定期手动检查既麻烦也不现实，因此，我们可以编写一个定期运行的变动跟踪器（`change tracker`）。一旦发生变动，跟踪器便会发出声音或发送提示信息。下面看一看如何编写基础的网站变动跟踪器。

### 5.12.1 预备知识

用Bash脚本跟踪网站变动意味着要在不同的时间检索网站，然后用`diff`命令进行比对。我们可以使用`curl`和`diff`来实现。

### 5.12.2 实战演练

结合各种命令来跟踪网页变更的Bash脚本如下：

```
#!/bin/bash
#文件名: change_track.sh
#用途: 跟踪网页更新

if [ $# -ne 1 ];
then
    echo -e "$Usage: $0 URL\n"
    exit 1;
fi

first_time=0
#非首次运行

if [ ! -e "last.html" ];
then
    first_time=1
    #首次运行
fi

curl --silent $1 -o recent.html

if [ $first_time -ne 1 ];
then
    changes=$(diff -u last.html recent.html)
    if [ -n "$changes" ];
    then
        echo -e "Changes:\n"
        echo "$changes"
    else
        echo -e "\nWebsite has no changes"
    fi
else
    echo "[First run] Archiving.."
fi

cp recent.html last.html
```

分别观察一下网页发生变动与没有变动时脚本track\_changes.sh的输出。

❑ 第一次运行:

```
$ ./track_changes.sh http://web.sarathlakshman.info/test.html
[First run] Archiving..
```

❑ 第二次运行:

```
$ ./track_changes.sh http://web.sarathlakshman.info/test.html
Website has no changes
```

❑ 在网页变动后，第三次运行：

```
$ ./test.sh http://web.sarathlakshman.info/test_change/test.html
Changes:

--- last.html 2010-08-01 07:29:15.000000000 +0200
+++ recent.html      2010-08-01 07:29:43.000000000 +0200
@@ -1,3 +1,4 @@
<html>
+added line :)
<p>data</p>
</html>
```

### 5.12.3 工作原理

脚本用 [ ! -e "last.html" ]; 检查自己是否是首次运行。如果last.html不存在，那就意味着这是首次运行，因此要下载网页并将其复制为last.html。

如果不是第一次运行，那么脚本应该下载一个新的网页副本 ( recent.html )，然后用diff检查差异。如果有变化，则打印出变更信息并将recent.html复制成last.html。

### 5.12.4 参考

5.4节讲解了curl命令。

5

## 5.13 以 POST 方式发送网页并读取响应

POST和GET是HTTP协议中用于发送或检索信息的两种请求类型。在GET请求方式中，我们利用网页的URL来发送参数（名称-值）。而在POST请求方式中，就不是这样了。POST方式用于提交表单。例如提交用户名、密码以及检索登录页面等。

当编写基于网页检索的脚本时，用POST方式发送页面很常见。让我们来看一看POST的使用方法。在编写解析网站数据的shell脚本的过程中，通过POST方式发送数据并检索输出结果来实现HTTP GET和POST请求自动化，是一项非常重要的任务。

### 5.13.1 预备知识

这里我们使用了一个测试网站（<http://book.sarathlakshman.com/lsc/mlogs/>）来提交当前的用户信息，如主机名和用户名。在网站主页上有两个文本域：HOSTNAME和USER以及一个SUBMIT

按钮。当用户输入主机名和用户名，点击SUBMIT按钮后，信息会被存储在网站中。这个过程可以使用一行curl（或wget）自动化POST请求来实现。来看看具体的做法。

### 5.13.2 实战演练

使用curl发送POST请求并读取网站的响应（HTML格式）：

```
$ curl URL -d "postvar=postdata2&postvar2=postdata2"
```

例如：

```
$ curl http://book.sarathlakshman.com/lsc/mlogs/submit.php -d
"host=test-host&user=slynux"
<html>
You have entered :
<p>HOST : test-host</p>
<p>USER : slynux</p>
<html>
```

curl会打印出响应页面。

-d表示以POST方式提交用户数据。-d的字符串参数的形式类似于GET请求。每对var=value之间用&分隔。

也可以利用wget，使用它的--post-data "string"来提交数据。例如：

```
$ get http://book.sarathlakshman.com/lsc/mlogs/submit.php --post-data
"host=test-host&user=slynux" -O output.html
$ cat output.html
<html>
You have entered :
<p>HOST : test-host</p>
<p>USER : slynux</p>
<html>
```

“名称-值”组合的格式同cURL中的一样。



-d参数的内容应该以引用的形式给出。否则，&会被shell解读为该命令需要作为后台进程运行。

### 5.13.3 工作原理

如果你查看网站的源代码（使用网页浏览器的查看源代码选项），你会发现一个与下面类似的HTML表单：

```
<form action="http://book.sarathlakshman.com/lsc/mlogs/submit.php" method="post" >

<input type="text" name="host" value="HOSTNAME" >
<input type="text" name="user" value="USER" >
<input type="submit" >
</form>
```

其中 `http://book.sarathlakshman.com/lsc/mlogs/submit.php` 是目标URL。当用户输入信息并点击SUBMIT按钮时，主机名和用户名就以POST请求的方式被发送到`submit.php`中，然后对应的响应页面被返回到浏览器。

#### 5.13.4 参考

- 5.4节讲解了`curl`命令。
- 5.2节讲解了`wget`命令。

## 本章内容

- ❑ 用tar归档
- ❑ 用cpio归档
- ❑ 用gzip压缩数据
- ❑ 用zip归档和压缩
- ❑ 更快速的归档工具pbzip2
- ❑ 创建压缩文件系统
- ❑ 用raync备份系统快照
- ❑ 用Git进行基于版本控制的备份
- ❑ 用fsarchiver创建全盘镜像

## 6.1 简介

提取快照和备份数据都是我们的日常工作，就服务器或大型数据存储系统而言，定期备份更是不可小视。shell脚本是实现备份自动化最简单的方法之一，如果不能实现自动化，备份的用处就大打折扣了。采用各种压缩方式也值得一试，这样我们便能够减少备份文件的体积。加密是另一种保护数据的常用方法。为了减少加密数据的大小，文件在加密之前通常都要先进行归档和压缩。有很多标准加密算法可以使用，而且也都有相应的shell工具。本章包含了这方面的各类攻略，包括创建和维护文件或文件夹归档、压缩格式以及加密技术。接下来让我们看看这些内容。

## 6.2 用 tar 归档

tar命令可以对文件进行归档。它最初是设计用来将数据存储存储在磁带上。tar可以将多个文件和文件夹保存为单个文件，同时还能保留所有的文件属性，如所有者、权限等。由tar创建的文件通常称为Tarball。在这则攻略里，我们将学习如何使用tar归档。

### 6.2.1 预备知识

所有Unix类操作系统中都默认包含tar命令。它语法简单，文件格式具备可移植性。tar支

持的参数包括：A、c、d、r、u、x、f和v。其中每一个参数都可以依据不同的用途单独使用。

## 6.2.2 实战演练

我们可以使用tar创建归档文件，对已有的归档文件进行操作。

- (1) 用tar对文件进行归档：

```
$ tar -cf output.tar [SOURCES]
```

例如：

```
$ tar -cf output.tar file1 file2 file3 folder1 ..
```

- (2) 使用选项-t列出归档文件中所包含的文件：

```
$ tar -tf archive.tar
file1
file2
```

- (3) 如果需要在归档或列出归档文件列表时获知更多的细节信息，可以使用-v或-vv参数。这个特性叫做“冗长模式”(verbose)，对于大多数命令而言，该模式会在终端中输出更多的细节。例如，利用冗长模式，你可以得到诸如文件权限、所有者所属的分组、文件修改日期等信息。

```
$ tar -tvf archive.tar
-rw-rw-r-- shaan/shaan      0 2013-04-08 21:34 file1
-rw-rw-r-- shaan/shaan      0 2013-04-08 21:34 file2
```



文件名必须紧跟在-f之后，而且-f应该是选项中的最后一个。  
假如你希望使用冗长模式，那就应该像这样使用：

```
$ tar -cvf output.tar file1 file2 file3 folder1 ..
```

6

## 6.2.3 工作原理

命令中的-c代表“创建文件”(creat file)，-f代表“指定文件名”(specify filename)。

我们可以在SOURCE处指定目录或文件名。也可以使用文件名列表或者诸如\*.txt这类通配符来指定。命令执行完毕后，tar会将源文件归档为output.tar。

不能传递上百个文件或文件夹作为tar的参数，毕竟参数数量不是无限制的。如果有很多文件需要归档，那么使用追加选项(详见下文)要更安全些。

## 6.2.4 补充知识

让我们再来看看tar命令的其他特性。

### 1. 向归档文件中添加文件

有时候，我们可能需要向已存在的归档文件再添加一些文件，这时可以使用追加选项-r。

要向已有的归档文件中添加一个文件：

```
$ tar -rvf original.tar new_file
```

创建一个其中包含有文本文件的归档：

```
$ tar -cf archive.tar hello.txt
```

用下面的方法列出归档文件中的内容：

```
$ tar -tf archive.tar
hello.txt
```

接着向归档文件中再添加另一个文件，并列出归档内容：

```
$ tar -rf archive.tar world.txt
$ tar -tf archive.tar
hello.txt
world.txt
```

这个归档文件中现在包含了两个文件。

### 2. 从归档文件中提取文件或文件夹

下面的命令将归档文件的内容提取到当前目录中：

```
$ tar -xf archive.tar
```

选项-x表示提取（exact）。

使用-x时，tar命令将归档文件中的内容提取到当前目录。我们也可以用选项-c来指定需要将文件提取到哪个目录：

```
$ tar -xf archive.tar -C /path/to/extraction_directory
```

这个命令将归档文件的内容提取到指定目录中。它提取的是归档文件中的全部内容。我们可以通过将文件名指定为命令行参数来提取特定的文件：

```
$ tar -xvf file.tar file1 file4
```

上面的命令只提取file1和file4，忽略其他文件。

### 3. 在tar中使用stdin和stdout

进行归档时，我们可以将stdout指定为输出文件，这样另一个命令就可以通过管道来读取（作为stdin），然后进行其他处理或提取内容。

当通过安全shell（Secure Shell，SSH）连接传输数据时，这招很管用。例如：

```
$ tar cvf - files/ | ssh user@example.com "tar xv -C Documents/"
```

在上面的例子中，对files目录中的内容进行了归档并输出到stdout（由'-'指明）。

### 4. 拼接两个归档文件

我们可以用-A选项轻松地合并多个tar文件。

假设我们现在有两个tar文件：file1.tar和file2.tar。可以按照下面的方法将file2.tar的内容合并到file1.tar中：

```
$ tar -Af file1.tar file2.tar
```

查看内容，验证操作是否成功：

```
$ tar -tvf file1.tar
```

### 5. 通过检查时间戳来更新归档文件中的内容

添加选项可以将指定的任意文件加入到归档文件中。如果同名文件已经存在，那么结果就是在归档文件中包含了两个同名的文件。我们可以用更新选项-u指明：只有比归档文件中的同名文件更新时才会被添加。

```
$ tar -tf archive.tar
filea
fileb
filec
```

上面的命令将列出归档文件中的内容。

仅当filea自上次被加入archive.tar后出现了变动才对其进行追加，可以使用：

```
$ tar -uf archive.tar filea
```

如果两个filea的时间戳相同，则什么都不会发生。

可用touch命令修改文件的时间戳，然后再用tar命令：

```
$ tar -uvvf archive.tar filea
-rw-r--r-- slynux/slynux      0 2010-08-14 17:53 filea
```

因为时间戳比归档文件中的同名文件更新，因此执行追加操作。验证如下：

```
$ tar -tf archive.tar
-rw-r--r-- slynux/slynux      0 2010-08-14 17:52 filea
-rw-r--r-- slynux/slynux      0 2010-08-14 17:52 fileb
-rw-r--r-- slynux/slynux      0 2010-08-14 17:52 filec
-rw-r--r-- slynux/slynux      0 2010-08-14 17:53 filea
```

如你所见，一个新的filea被加入了归档文件中。当从中提取文件时，tar会挑选最新的filea进行提取。

## 6. 比较归档文件与文件系统中的内容

有时候需要知道归档文件中的文件与文件系统中的同名文件是否相同。选项-d可以打印出两者之间的差别：

```
$ tar -df archive.tar
afile: Mod time differs
afile: Size differs
```

## 7. 从归档文件中删除文件

我们可以用--delete选项从给定的归档文件中删除文件。例如：

```
$ tar -f archive.tar --delete file1 file2 ..
```

或者

```
$ tar --delete --file archive.tar [FILE LIST]
```

来看另外一个例子：

```
$ tar -tf archive.tar
filea
fileb
filec
```

删除filea：

```
$ tar --delete --file archive.tar filea
$ tar -tf archive.tar
fileb
filec
```

## 8. 压缩tar归档文件

tar命令只能用来对文件进行归档，它并不具备压缩功能。出于这个原因，多数用户在使用归档文件时都会对文件采用某种形式的压缩，这样就能够显著减少文件的体积。归档文件通常被压缩成下列格式之一：

- ❑ file.tar.gz
- ❑ file.tar.bz2
- ❑ file.tar.lzma

不同的tar选项可以用来指定不同的压缩格式：

- ❑ -j指定bunzip2格式；
- ❑ -z指定gzip格式；
- ❑ --lzma指定lzma格式。

这些格式会在随后专门讲解压缩技术的攻略中讨论。

也可以不明确指定上面那些特定的选项来使用压缩功能。tar能够通过查看输出或输入文件的扩展名来进行压缩。为了让tar支持根据扩展名自动进行压缩，使用 -a或 --auto-compress 选项：

```
$ tar acvf archive.tar.gz filea fileb filec
filea
fileb
filec
$ tar tf archive.tar.gz
filea
fileb
filec
```

## 9. 从归档中排除部分文件

通过指定模式可以从归档中排除部分文件。用 --exclude [PATTERN] 排除匹配通配符样式的文件。

例如，排除所有的.txt文件：

```
$ tar -cf arch.tar * --exclude "*.txt"
```



样式应该使用双引号来引用，避免shell对其进行扩展。

也可以将需要排除的文件列表放入文件中，同时配合选项 -x：

```
$ cat list
filea
fileb

$ tar -cf arch.tar * -X list
```

这样就无需对filea和fileb进行归档了。

### 10. 排除版本控制目录

我们通常使用tar归档文件来分发源代码。大多数源代码都是使用版本控制系统进行维护的，如subversion、Git、mercurial、cvs等。版本控制系统中的代码目录包含用来管理版本的特殊目录，如svn或.git。但源代码本身并不需要这些目录，所以不应该将它们包含在源代码的tar归档文件中。

为了在归档时排除版本控制相关的文件和目录，可以使用tar的--exclude-vcs选项。例如：

```
$ tar --exclude-vcs -czvzf source_code.tar.gz eye_of_gnome_svn
```

### 11. 打印总字节数

有时候我们需要知道归档了多少字节。用-totals就可以在归档完成之后打印出总归档字节数：

```
$ tar -cf arc.tar * --exclude "*.txt" --totals
Total bytes written: 20480 (20KiB, 12MiB/s)
```

## 6.2.5 参考

6.4节会讲解gzip命令。

## 6.3 用cpio归档

cpio是另一种类似于tar的归档格式。它用来将多个文件和文件夹存储为单个文件，同时保留所有的文件属性，如权限、文件所有权等。不过，cpio并不像tar那么常用。它多用于RPM软件包（Fedora使用这种格式）、Linux内核的initramfs文件（包含了内核镜像）等。这则攻略将给出几种cpio的用法。

### 6.3.1 实战演练

cpio通过stdin获取输入文件名，并将归档文件写入stdout。我们必须将stdout重定向到一个文件来接收cpio的输出。

(1) 创建测试文件：

```
$ touch file1 file2 file3
```

(2) 将测试文件按照下面的方法进行归档：

```
$ echo file1 file2 file3 | cpio -ov > archive.cpio
```

(3) 列出cpio归档文件中的内容:

```
$ cpio -it < archive.cpio
```

(4) 从cpio归档文件中提取文件:

```
$ cpio -id < archive.cpio
```

### 6.3.2 工作原理

对于归档命令:

- ❑ -o指定了输出;
- ❑ -v用来打印归档文件列表。



通过cpio,我们能够利用文件的绝对路径进行归档。/usr/somedir就是一个绝对路径,因为它是以根目录(/)作为路径的起始。

相对路径不以/开头,而是以当前目录作为起始点。例如, test/file表示有一个目录test,而file位于test目录中。

当进行提取时, cpio会将内容提取到绝对路径中。而tar会移去绝对路径开头的/, 将之转换为相对路径。

在列出给定cpio归档文件所有内容的命令中:

- ❑ -i用于指定输入;
- ❑ -t表示列出归档文件中的内容。

当使用命令进行提取时, -d用来表示提取。cpio在覆盖文件时不会发出提示。

## 6.4 使用 gzip 压缩数据

gzip是GNU/Linux平台下常用的压缩格式。gzip、gunzip、zcat都可以处理这种压缩文件类型。但gzip只能压缩单个文件或数据流,而无法对目录和多个文件进行归档。因此我们需要先创建tar归档文件,然后再用gzip进行压缩。让我们来看看gzip的使用方法。

### 6.4.1 实战演练

gzip可以用于压缩与解压缩。

- (1) 要使用gzip压缩文件，可以使用下面的命令：

```
$ gzip filename
$ ls
filename.gz
```

- (2) 将gzip文件解压缩的方法如下：

```
$ gunzip filename.gz
$ ls
file
```

- (3) 列出压缩文件的属性信息：

```
$ gzip -l test.txt.gz
compressed      uncompressed  ratio uncompressed_name
35              6 -33.3% test.txt
```

- (4) gzip命令可以从stdin中读入文件，也可以将压缩文件写出到stdout。

从stdin读入并将压缩后的数据写出到stdout：

```
$ cat file | gzip -c > file.gz
```

选项 -c 用来将输出指定到stdout。

- (5) 我们可以指定gzip的压缩级别。用 --fast 或 --best 选项分别提供最低或最高的压缩比。

## 6.4.2 补充内容

gzip命令通常与其他命令结合使用。它还有一些高级选项可以用来指定压缩比。让我们来看看gzip的这些特性。

### 1. 压缩归档文件

我们通常将gzip与归档文件结合使用。有两种方法可以创建经由gzip压缩过的归档文件。

#### ❑ 方法1

```
$ tar -czvzf archive.tar.gz [FILES]
```

或者

```
$ tar -cavzf archive.tar.gz [FILES]
```

选项 -a 表明从文件扩展名自动推断压缩格式。

## ❑ 方法 2

首先，创建一个tar归档文件：

```
$ tar -cvvf archive.tar [FILES]
```

压缩tar归档文件：

```
$ gzip archive.tar
```

如果有大量文件（上百个）需要归档及压缩，我们可以采用方法2，并稍作变动。将多个文件作为命令行参数传递给tar的问题在于：tar只能从命令行中接受有限个文件。要解决这个问题，我们可以在循环中使用追加选项（-r）来逐个添加文件：

```
FILE_LIST="file1 file2 file3 file4 file5"

for f in $FILE_LIST;
do
tar -rvf archive.tar $f
done

gzip archive.tar
```

如果要提取经由gzip压缩的归档文件中的内容，可以使用：

```
$ tar -xavvf archive.tar.gz -C extract_directory
```

在上面的命令中，选项-a用于自动检测压缩格式。

## 2. zcat——无需解压缩，直接读取gzip格式文件

zcat命令无需人工干涉，直接就可以.gz文件中的内容提取到stdout。下面的方法可以在保持.gz文件不变的情况下，将其中的文件提取到stdout中：

```
$ ls
test.gz

$ zcat test.gz
A test file
# 文件test包含了一行文本"A test file"

$ ls
test.gz
```

## 3. 压缩率

我们可以指定压缩率，它共有9级，其中：

- 1级的压缩率最低，但是压缩速度最快；
- 9级的压缩率最高，但是压缩速度最慢。

你可以按照下面的方法指定压缩比：

```
$ gzip -5 test.img
```

这应该能在压缩速度和压缩比之间获得一个不错的平衡。

#### 4. 使用bzip2

bzip2是另一种常用的工具，其功能和语法同gzip非常类似。唯一的不同在于bzip2的压缩效率比gzip更高，但花费的时间比gzip更长。

用bzip2进行压缩：

```
$ bzip2 filename
```

解压缩bzip2格式的文件：

```
$ bunzip2 filename.bz2
```

生成tar.bz2文件并从中提取内容的方法同之前介绍的tar.gz类似：

```
$ tar -xjvf archive.tar.bz2
```

其中-j表明该归档文件是bzip2格式。

#### 5. 使用lzma

lzma是另一种压缩工具，它的压缩率甚至比gzip和bzip2更好。

使用lzma进行压缩：

```
$ lzma filename
```

解压缩lzma文件：

```
$ unlzma filename.lzma
```

可以使用tar命令的--lzma选项对生成的tar归档文件进行压缩或提取：

```
$ tar -cvvf --lzma archive.tar.lzma [FILES]
```

或者

```
$ tar -cavvf archive.tar.lzma [FILES]
```

如果要将经过lzma压缩过的tar归档文件中的内容提取到指定的目录中，可以使用：

```
$ tar -xvzf --lzma archive.tar.lzma -C extract_directory
```

其中，-x用于提取内容，--lzma指定使用lzma对归档文件进行解压缩。

我们也可以使用：

```
$ tar -xavvf archive.tar.lzma -C extract_directory
```

### 6.4.3 参考

6.2节讲解了tar命令。

## 6.5 用 zip 归档和压缩

ZIP作为一种流行的压缩格式，在很多平台中都可以看到它的身影。在Linux下，它的应用不如gzip或bzip2那么广泛，但是Internet上的文件通常都采用这种格式。在这这攻略中，我们会看到如何使用zip进行压缩与解压缩。

### 6.5.1 实战演练

来看看zip的各种选项的用法。

- (1) 对归档文件采用ZIP格式进行压缩：

```
$ zip archive_name.zip [SOURCE FILES/DIRS]
```

例如：

```
$ zip file.zip file
```

该命令会生成file.zip。

- (2) 对目录和文件进行递归操作：

```
$ zip -r archive.zip folder1 folder2
```

其中，-r用于指定递归操作。

- (3) 要从ZIP文件中提取内容，可以使用：

```
$ unzip file.zip
```

在完成提取操作之后，unzip并不会删除file.zip（这一点与unlzma和gunzip不同）。

- (1) 如果需要更新压缩文件中的内容, 使用选项 `-u`:

```
$ zip file.zip -u newfile
```

- (2) 从压缩文件中删除内容, 则使用 `-d`:

```
$ zip -d arc.zip file.txt
```

- (3) 列出压缩文件中的内容:

```
$ unzip -l archive.zip
```

### 6.5.2 工作原理

尽管同大多数我们已经讲过的归档、压缩工具类似, 但zip在完成归档之后并不会删除源文件, 这一点与lzma, gzip, bzip2不同。最重要的是, 尽管与tar相像, zip既可以进行归档, 也可以进行压缩, 而单凭tar, 则无法进行压缩操作。

## 6.6 更快速的归档工具 pbzip2

如今大多数计算机都配备了至少两个处理器核心, 这基本上相当于拥有了两块物理CPU。但是仅仅是一块多核CPU并不代表程序可以运行得更快, 重要的是程序自身能够利用多个处理器核心来提高运行速度。

我们目前已经看到的多数压缩命令只能利用单个处理器核心, 所以速度并不会特别快。而pbzip2能够借助多个处理器核心来降低压缩文件所需的时间。

### 6.6.1 预备知识

多数发布版中通常都没有预装pbzip2, 你得使用软件包管理器自行安装。

### 6.6.2 实战演练

来看看如何使用pbzip2来压缩及提取文件。

- (1) 压缩单个文件:

```
pbzip2 myfile.tar
```

pbzip2会自动检测系统中处理器核心的数量, 然后将myfile.tar压缩成myfile.tar.bz2。

- (2) 要将多个文件或目录进行归档及压缩, 可以使用tar配合pbzip2来实现:

```
tar cf myfile.tar.bz2 --use-compress-prog=pbzip2 dir_to_compress/
```

或者

```
tar -c directory_to_compress/ | pbzip2 -c > myfile.tar.bz2
```

(3) 从pbzip2格式的文件中进行提取。

如果是tar.bz2文件，我们可以一次性完成解压缩和提取工作：

```
pbzip2 -dc myfile.tar.bz2 | tar x
```

如果是经过pbzip2压缩过的归档文件，可以使用：

```
pbzip2 -d myfile.tar.bz2
```

### 6.6.3 工作原理

pbzip2在内部使用的压缩算法和bzip2一样，但是它利用pthreads（一个线程库）来同时对多个数据块进行压缩。不过这一切对于用户而言都是透明的，结果就是更快的压缩速度。

同gzip或bzip2一样，pbzip2并不能创建归档文件，它只能对单个文件进行操作。如果要压缩多个文件或目录，还得结合tar来使用。

### 6.6.4 补充内容

pbzip2还有另外一些有用的选项。

#### 1. 手动指定处理器数量

使用pbzip2的-p选项来手动指定处理器核心的数量。如果无法自动检测处理器核心数量或是希望能够释放一些处理核心来进行其他工作，-p选项就能派上用场了。

```
pbzip2 -p4 myfile.tar
```

上面的命令告诉pbzip2使用4个处理器核心。

#### 2. 指定压缩比

像其他压缩工具一样，我们可以使用从1到9的选项来分别指定最快和最优的压缩比。

## 6.7 创建压缩文件系统

squashfs是一种具有超高压缩率的只读型文件系统，这种文件系统能够将2GB~3GB的数据压缩成一个700MB的文件。Linux LiveCD（或是LiveUSB）就是使用squashfs创建的。这类CD利用只读型的压缩文件系统将根文件系统保存在一个压缩文件中。可以使用环回方式将其挂载并

装入完整的Linux环境。如果进程需要某些文件，可以将它们解压，然后载入内存中使用。

如果即需要采用超高的文件压缩率，又希望在无需解压的情况下读取少量文件，那么squashfs就能够大显身手了。解压体积较大的压缩文件可得花上一阵工夫。但如果将文件以环回形式挂载，那速度会变得飞快。因为只有出现访问请求时，对应的那部分压缩文件才会被解压缩。让我们来看看如何使用squashfs。

### 6.7.1 预备知识

squashfs在内部采用了gzip和lzma这类压缩算法。所有的现代Linux发行版都对其提供了支持。但要想创建squashfs文件，则需要额外安装squashfs-tools。

### 6.7.2 实战演练

来看看如何创建并挂载squashfs文件。

(1) 添加源目录和文件，创建一个squashfs文件：

```
$ mksquashfs SOURCES compressedfs.squashfs
```

SOURCES部分可以是通配符或文件、目录路径。

例如：

```
$ sudo mksquashfs /etc test.squashfs  
Parallel mksquashfs: Using 2 processors  
Creating 4.0 filesystem on test.squashfs, block size 131072.  
[=====] 1867/1867 100%
```



还有更多的细节信息会出现在终端上。由于版面的限制，这里就不再列出这些信息了。

(2) 利用环回形式挂载squashfs文件：

```
# mkdir /mnt/squash  
# mount -o loop compressedfs.squashfs /mnt/squash
```

你可以访问/mnt/squashfs访问其中的内容。

### 6.7.3 补充内容

可以指定额外的参数来定制squashfs文件系统。让我们来看看这些命令选项。

### 在创建squashfs文件时排除部分文件

创建squashfs文件时，我们可以排除部分文件。这些文件可以用文件列表或通配符来指定。

使用选项-e，将需要排除的文件列表以命令行参数的方式来指定。例如：

```
$ sudo mksquashfs /etc test.squashfs -e /etc/passwd /etc/shadow
```

其中，选项-e用于将文件passwd和shadow排除在外。

也可以将需要排除的文件名列表写入文件，然后用-ef指定该文件：

```
$ cat excludelist
/etc/passwd
/etc/shadow
```

```
$ sudo mksquashfs /etc test.squashfs -ef excludelist
```

如果希望在排除文件列表中使用通配符，那么可以使用-wildcard选项。

## 6.8 使用 rsync 备份系统快照

备份数据算得上是多数系统管理员日常必备的工作。除了备份本地文件，我们可能还得对Web服务器或远端数据进行备份。rsync可以对位于不同位置的文件和目录进行同步，它利用差异计算以及压缩技术来最小化数据传输量。相对于cp命令，rsync的优势在于使用了高效的差异算法。另外，它还支持网络数据传输。在进行复制的同时，rsync会比较源端和目的端的文件，只有当文件有更新时才进行复制。rsync也支持压缩、加密等多种特性。让我们看看rsync的使用方法。

### 6.8.1 实战演练

下面来看看如何使用rsync进行文件复制及备份。

(1) 将源目录复制到目的端：

```
$ rsync -av source_path destination_path
```

例如：

```
$ rsync -av /home/slynux/data slynux@192.168.0.6:/home/backups/data
```

其中：

- -a表示要进行归档；
- -v表示在stdout上打印出细节信息或进度。

上面的命令会以递归的方式将所有的文件从源路径复制到目的路径。我们可以指定

本地路径，也可以指定远端路径。

- (2) 将数据备份到远程服务器或主机：

```
$ rsync -av source_dir username@host:PATH
```

如果需要在目的端建立一份镜像，只需要定期运行同样的rsync命令即可。它只会对更改过的文件进行复制。

- (3) 用下面的方法将远程主机上的数据恢复到本地主机：

```
$ rsync -av username@host:PATH destination
```



rsync命令用SSH连接远程主机，因此必须使用user@host这种形式设定远程主机的地址，其中user代表用户名，host代表远程主机的IP地址或主机名。而PATH指定需要从中复制数据的远程主机上的路径。

确保远程主机上安装并运行着OpenSSH服务器。如果连接远程主机时不希望输入密码，可以参考7.8节。

- (4) 通过网络进行传输时，压缩数据能够明显改善传输效率。我们可以用rsync的选项-z指定在网络传输时压缩数据。例如：

```
$ rsync -avz source destination
```

- (5) 将一个目录中的内容同步到另一个目录：

```
$ rsync -av /home/test/ /home/backups
```

这条命令将源目录（/home/test）中的内容（不包括目录本身）复制到现有的backups目录中。

- (6) 将包括目录本身在内的内容复制到另一个目录中：

```
$ rsync -av /home/test /home/backups
```

这条命令将包括源目录本身（/home/test）在内的内容复制到新创建的目录backups中。



就路径格式而言，如果我们在源路径末尾使用/，那么rsync会将source\_path尾端目录中的所有内容复制到目的端。

如果没有使用/，rsync会将source\_path尾端目录本身复制到目的端。

以下命令复制test目录到目的端目录中：

```
$ rsync -av /home/test /home/backups/
$ rsync -av /home/test /home/backups
```



如果在destination\_path末尾使用/，那么rsync会将来自源端的内容复制到目的端目录中。

如果没有使用/，rsync会在目的端路径尾部创建一个同名目录，然后将源端内容复制到这个目录中。

## 6.8.2 工作原理

rsync所使用的源路径和目的路径既可以是本地路径，也可以是远程路径。最重要的是，两者皆可以是远程路径。通常使用SSH进行远程连接，由rsync来决定应该复制哪些文件。本地路径和远程路径看起来像这样：

- ❑ /home/slynux/data (本地路径)
- ❑ slynux@192.168.0.6:/home/backups/data (远程路径)

/home/slynux/data指定的是执行rsync命令的那台主机上的绝对路径。slynux@192.168.0.6:/home/backups/data指定的是IP地址为192.168.0.6的主机，以用户slynux的身份登录，路径为/home/backups/data。

## 6.8.3 补充内容

可以通过命令行选项来指定rsync命令的其他功能。让我们来逐项查看。

### 1. 在使用rsync进行归档的过程中排除部分文件

对远端内容进行归档时，有些文件并不需要进行更新。我们可以告知rsync将某些文件从本次操作中排除。有两个选项可以用来排除文件：

```
--exclude PATTERN
```

可以通过通配符指定需要排除的文件。例如：

```
$ rsync -avz /home/code/some_code /mnt/disk/backup/code --exclude "*.txt"
```

该命令不对.txt文件进行备份。

或者我们可以通过一个列表文件指定需要排除的文件。

这可以利用--exclude-from FILEPATH。

## 2. 在更新rsync备份时，删除不存在的文件

默认情况下，rsync并不会在目的端删除那些在源端已不存在的文件。如果要删除这些已不存在文件，使用rsync的 `--delete` 选项：

```
$ rsync -avz SOURCE DESTINATION --delete
```

## 3. 定期进行备份

你可以创建一个cron任务来定期进行备份。

下面是一个简单的例子：

```
$ crontab -ev
```

添加上这么一行：

```
0 */10 * * * rsync -avz /home/code user@IP_ADDRESS:/home/backups
```

上面的crontab条目将rsync调度为每10个小时运行一次。

`*/10`处于crontab语法中的钟点位（hour position），`/10`表明每10小时执行一次备份。如果`*/10`出现在分钟位（minutes position），那就是每10分钟执行一次备份。

请参阅9.7节了解如何配置crontab。

# 6.9 用 Git 进行基于版本控制的备份

人们在备份数据时会采取不同的策略。不过比起将整个源目录复制到备份目录中，并使用日期或时间作为版本号，利用差异备份要更有效，因为前者只会白白浪费存储空间。我们只需要复制那些在备份之后发生变化的那些文件，这叫做增量备份。可以用rsync这类工具来手动创建增量备份，不过恢复这种备份可不是件易事。维护和恢复变更的最好方法是使用版本控制系统。由于代码的变更相当频繁，版本控制系统多用于软件开发与代码维护中。Git是目前最具声望、最高效的版本控制系统。我们可以在非编程环境下用Git进行常规文件的备份。

## 6.9.1 预备知识

我们有一个包含了多个文件和子目录的目录。现在需要跟踪目录内容的变更并对其进行备份。如果数据受损或丢失，必须能够恢复数据之前的备份。我们需要将数据备份到本地主机或远程主机中。使用软件包管理器安装Git，然后来学习它的用法。

## 6.9.2 实战演练

先来看看如何使用Git进行数据备份。

- (1) 进入需要备份的目录：

```
$ cd /home/data/source
```

将其作为需要被跟踪的目录源。

- (2) 设置并初始化远端备份目录。在远程主机中创建备份目录：

```
$ mkdir -p /home/backups/backup.git
$ cd /home/backups/backup.git
$ git init --bare
```

在源主机中执行下列步骤。

- (1) 在源主机中将用户详细信息添加到Git：

```
$ git config --global user.name "Sarath Lakshman"
$ git config --global user.email slynux@slynux.com
```

- (2) 初始化主机中需要进行备份的源目录。在源目录中执行下列命令：

```
$ git init
Initialized empty Git repository in /home/backups/backup.git/

$ git commit --allow-empty -am "Init"
[master (root-commit) b595488] Init
```

- (3) 在源目录中执行下列命令来添加远程Git目录并同步备份：

```
$ git remote add origin user@remotehost:/home/backups/backup.git

$ git push origin master
Counting objects: 2, done.
Writing objects: 100% (2/2), 153 bytes, done.
Total 2 (delta 0), reused 0 (delta 0)
To user@remotehost:/home/backups/backup.git
 * [new branch]      master -> master
```

- (4) 为Git跟踪添加或删除文件。

下面的命令将当前目录下的所有文件和文件夹添加到备份列表中：

```
$ git add *
```

我们可以有条件地添加某些文件到备份列表中：

```
$ git add *.txt
$ git add *.py
```

删除不需要跟踪的文件和文件夹：

```
$ git rm file
```

也可以使用通配符：

```
$ git rm *.txt
```

(5) 检查点或标注备份点。

用下列命令来标注一个附带消息的备份的检查点：

```
$ git commit -m "Commit Message"
```

我们需要定期更新远端的备份，因此得设置一个cron任务（例如，每5个小时进行一次备份）：

创建一个crontab条目：

```
0 */5 * * * /home/data/backup.sh
```

创建脚本/home/data/backup.sh：

```
#!/bin/ bash
cd /home/data/source
git add .
git commit -am "Backup taken at @ $(date)"
git push
```

现在我们就算是完成了备份系统的设置。

(6) 查看所有版本的备份：

```
$ git log
```

(7) 要恢复到之前的某个状态或版本，需要查看一个由32位十六进制串组成的提交ID。  
通过git checkout来使用提交ID。

如果提交ID为 3131f9661ec1739f72c213ec5769bc0abefa85a9，那就是：

```
$ git checkout 3131f9661ec1739f72c213ec5769bc0abefa85a9
```

以下命令可以使转化持久化：

```
$ git commit -am "Restore @ $(date) commit ID: 3131f9661ec1739f72c213ec5769bc0abefa85a9"
```

再次查看有关版本的细节信息：

```
$ git log
```

- (8) 如果工作目录由于某些原因受到了损坏，我们需要用远端的备份来进行修复。按照下面的方法从远端备份中重建损坏的内容：

```
$ git clone user@remotehost:/home/backups/backup.git
```

这条命令将创建一个包含之前全部内容的目录。



尽管就保留版本化的文本文件（包括文档、源代码等等）副本而言，Git表现的算是相当不错了。但最好不要将其用于大量的二进制数据备份。例如用Git来对图片库进行备份/版本管理，就没什么意义了。这是因为如果涉及的是二进制文件，Git保留的是整个文件，而非文件之间的差异，这样会占用大量的磁盘空间。

## 6.10 用 fsarchiver 创建全盘镜像

fsarchiver可以将整个文件系统中的内容保存成一个压缩形式的归档文件。鉴于这种能力，它成为功能最为完备，也最为易用的备份工具之一。

fsarchiver作为partimage（一款知名的文件系统备份工具）的接班人，能够支持像ext4这种较新的文件系统，不过partimage配备了一套小型的图形用户界面，相比之下更易用一些。

6

### 6.10.1 预备知识

fsarchiver默认并没有安装在大多数发布版中。你得用软件包管理器自行安装。更多的信息可以参考<http://www.fsarchiver.org/Installation>。

### 6.10.2 实战演练

- (1) 创建文件系统/分区备份。

使用fsarchiver的savefs选项：

```
fsarchiver savefs backup.fsa /dev/sda1
```

backup.fsa是最终的备份文件，/dev/sda/是要备份的分区。

## (2) 同时备份多个分区。

还是使用savefs选项，将分区作为fsarchiver最后的参数：

```
fsarchiver savefs backup.fsa /dev/sda1 /dev/sda2
```

## (3) 从备份归档中恢复分区。

使用fsarchiver的restfs选项：

```
fsarchiver restfs backup.fsa id=0,dest=/dev/sda1
```

id=0表明我们希望从备份归档中提取第一个分区的内容，将其恢复到由dest=/dev/sda1所指定的分区中。

## (4) 从备份归档中恢复多个分区。

像之前一样，使用restfs选项：

```
fsarchiver restfs backup.fsa id=0,dest=/dev/sda1 id=1,dest=/dev/sdb1
```

我们使用了两组id，dest告诉fsarchiver从备份中将前两个分区的内容恢复到指定的物理分区中。

### 6.10.3 工作原理

和tar的工作原理非常类似，fsarchiver遍历整个文件系统来生成一个文件列表，然后将所有的文件保存在压缩过的归档文件中。不像tar那样只保存文件信息，fsarchiver还要对文件系统进行备份。这意味着它可以很容易地将备份恢复到一个全新的系统中，因为不必再重建文件系统了。

如果你是第一次看到/dev/sda1这样的分区记法，那有必要解释一下。在Linux中，/dev下存放的都是称为设备文件的一类特殊文件，它们分别指向某个物理设备。sda1中的sd指的是SATA disk，接下来的字母可以是a，b，c等，最后跟上分区编号，如图6-1所示。

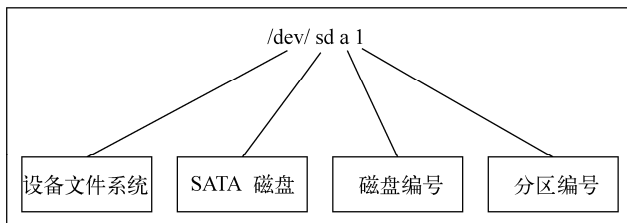


图6-1 显示了Linux中磁盘设备的文件名的各个组成部分

## 本章内容

- ❑ 网络设置
- ❑ 使用ping
- ❑ 列出网络上所有的活动主机
- ❑ 用SSH在远程主机上运行命令
- ❑ 通过网络传输文件
- ❑ 连接无线网络
- ❑ 用SSH实现无密码自动登录
- ❑ 使用SSH进行端口转发
- ❑ 在本地挂载点上挂载远程驱动器
- ❑ 网络流量与端口分析
- ❑ 创建套接字
- ❑ 互联网连接共享
- ❑ 使用iptables架设简易防火墙

## 7.1 简介

联网就是将主机进行互联以形成网络，使得网络中的主机得以交换信息。应用最广泛的网站栈就是TCP/IP，其中每个节点都分配了一个用作标识的独一的IP地址。有很多联网参数，如子网掩码、路由、端口和DNS等，我们需要对这些知识有一个基本的认识。

一些利用网络的应用通过打开并连接到端口来进行运作，端口用于指代某种服务，如数据传输、远程shell登录等。有些管理任务可以通过网络进行。shell脚本也能用来配置网络节点、测试主机是否可用、自动执行远程主机命令等。本章着重介绍网络相关的工具和命令，以及如何用它们解决各种问题。

## 7.2 网络设置

在深入学习与联网相关的攻略之前，有必要简单了解一下网络设置、相关术语以及用于分配IP地址、添加路由等命令。这则攻略会从头开始介绍GNU/Linux中用于联网的各种命令及用法。

### 7.2.1 预备知识

网络接口用于将主机连接到网络。在Linux中通常使用eth0、eth1（指代以太网接口）这种方式来命名网络接口。还有一些其他的接口，如usb0、wlan0等，分别对应USB网络接口、无线LAN。

在这则攻略中会涉及如下命令：ifconfig、route、nslookup和host。

ifconfig命令用于配置及显示网络接口、子网掩码等详细信息。它通常位于/sbin/ifconfig中。

### 7.2.2 实战演练

(1) 列出当前的网络接口配置：

```
$ ifconfig
lo          Link encap:Local Loopback
inet addr:127.0.0.1  Mask:255.0.0.0
inet6addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING  MTU:16436  Metric:1
            RX packets:6078 errors:0 dropped:0 overruns:0 frame:0
            TX packets:6078 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
            RX bytes:634520 (634.5 KB)  TX bytes:634520 (634.5 KB)
wlan0       Link encap:EthernetHWaddr 00:1c:bf:87:25:d2
inet addr:192.168.0.82  Bcast:192.168.3.255  Mask:255.255.252.0
inet6addr: fe80::21c:bfff:fe87:25d2/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
            RX packets:420917 errors:0 dropped:0 overruns:0 frame:0
            TX packets:86820 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:1000
            RX bytes:98027420 (98.0 MB)  TX bytes:22602672 (22.6 MB)
```

ifconfig输出的最左边一列是网络接口名，右边的若干列显示对应的网络接口的详细信息。

(2) 手动设置网络接口的IP地址：

```
# ifconfig wlan0 192.168.0.80
```

你需要使用root账号运行上述命令。192.168.0.80是要设置的IP地址。

使用以下命令设置比IP地址的子网掩码：

```
# ifconfig wlan0 192.168.0.80 netmask 255.255.252.0
```

(3) 自动配置网络接口。如果你连接到一个支持自动分配IP的有线网络中，只需要使用下面的命令就可以配置网络接口了：

```
# dhclient eth0
```

### 7.2.3 补充内容

让我们再看一些其他的基础命令以及用法。

#### 1. 打印网络接口列表

这个单行命令可以打印系统可用的网络接口列表。

```
$ ifconfig | cut -c-10 | tr -d ' ' | tr -s '\n'
lo
wlan0
```

ifconfig输出的前10个字符是用于保留打印网络接口的名称，因此我们用cut命令提取每一行的前10个字符。tr -d ' '删除每一行的所有空格。用tr -s '\n'压缩重复的换行符生成接口名称列表。

#### 2. 显示IP地址

ifconfig会显示系统中所有可用网络接口的详细信息。我们可以限制它只显示某个特定的接口信息：

```
$ ifconfig iface_name
```

例如：

```
$ ifconfig wlan0
wlan0      Link encap:EthernetHWaddr 00:1c:bf:87:25:d2
inet addr:192.168.0.82  Bcast:192.168.3.255
           Mask:255.255.252.0
```

在上面命令输出中，我们感兴趣的是IP地址、广播地址、硬件地址和子网掩码。它们分别为：

- ❑ HWaddr 00:1c:bf:87:25:d2是硬件地址（MAC地址）；
- ❑ inet addr:192.168.0.82是IP地址；
- ❑ Bcast:192.168.3.255是广播地址；
- ❑ Mask:255.255.252.0是子网掩码。

在编写某些脚本时，我们可能需要在脚本中提取某些地址来做进一步的处理。常见的是提取IP地址。要从ifconfig输出中提取IP地址，可以使用：

```
$ ifconfig wlan0 | egrep -o "inet addr:[^ ]*" | grep -o "[0-9.]*"
192.168.0.82
```

第一条命令egrep -o "inet addr:[^ ]\*" 会打印出inet addr:192.168.0.82。模式以inet addr:作为起始，以非空格字符序列（由 [^ ]\* 指定）作为结束。在接下来的管道操作中，打印出数字与点号（.）的组合。

### 3. 硬件地址（MAC地址）欺骗

在某些情况下，需要利用硬件地址对网络上的计算机进行认证或过滤，对此我们可以使用硬件地址欺骗。硬件地址在ifconfig输出中是以HWaddr 00:1c:bf:87:25:d2的形式出现的。

我们能够按照下面的方法在软件层面上进行硬件地址欺骗：

```
# ifconfig eth0 hw ether 00:1c:bf:87:25:d5
```

在上面的命令中，00:1c:bf:87:25:d5是分配的新MAC地址。如果我们需要通过部署了MAC认证的Internet服务提供商才能够访问Internet，这招就能排上用场了。注意，这在机器重启之后就失效了。

### 4. 名字服务器与DNS（域名服务）

互联网最根本的寻址方案是IP地址（采用点分十进制形式，例如202.11.32.75）。然而Internet上的资源（比如网站）是通过被称为URL或域名的ASCII字符组合来访问的，例如google.com就是一个域名，它对应一个（或多个）IP地址。在浏览器中输入IP地址同样可以访问www.google.com。

这种利用符号名对IP地址进行抽象的技术就被称为域名服务（DNS）。当我们输入google.com，计算机使用配置好的DNS服务器将域名解析为对应的IP地址。在本地网络中，我们可以设置本地DNS，以便使用主机名来命名本地网络上的主机。

分配给当前系统的名字服务器可以通过读取/etc/resolv.conf查看。例如：

```
$ cat /etc/resolv.conf
nameserver 8.8.8.8
```

我们可以像下面这样手动添加名字服务器：

```
# echo nameserver IP_ADDRESS >> /etc/resolv.conf
```

该如何获得域名所对应的IP地址呢？获取IP地址最简单的方法就是ping给定的域名，然后查看回应信息。例如：

```
$ ping google.com
PING google.com (64.233.181.106) 56(84) bytes of data.
#64.233.181.106是google.com对应的IP地址。
```

一个域名可以分配多个IP地址。对于这种情况，ping只会显示其中的一个地址。要想获取分配给域名的所有IP地址，就得使用DNS查找工具了。

### 5. DNS查找

有多种基于命令行的DNS查找工具，这些工具会向DNS服务器请求解析IP地址。host和nslookup就是此类DNS查找工具中的两个。

当执行`host`时，它会列出某个域名所有的IP地址。`nslookup`类似于`host`命令，它用于查询DNS相关的细节信息以及名字解析。例如：

```
$ host google.com
google.com has address 64.233.181.105
google.com has address 64.233.181.99
google.com has address 64.233.181.147
google.com has address 64.233.181.106
google.com has address 64.233.181.103
google.com has address 64.233.181.104
```

`host`也可以列出DNS资源记录（DNS resource record）：

```
$ nslookup google.com
Server:      8.8.8.8
Address:     8.8.8.8#53

Non-authoritative answer:
Name:   google.com
Address: 64.233.181.105
Name:   google.com
Address: 64.233.181.99
Name:   google.com
Address: 64.233.181.147
Name:   google.com
Address: 64.233.181.106
Name:   google.com
Address: 64.233.181.103
Name:   google.com
Address: 64.233.181.104

Server:      8.8.8.8
```

上面最后一行对应着用于DNS解析的默认名字服务器。

如果不使用DNS服务器，也可以为IP地址解析添加符号名，这只需要向文件`/etc/hosts`中加入条目即可。

用下面的方法进行添加：

```
# echo IP_ADDRESS symbolic_name >> /etc/hosts
```

例如：

```
# echo 192.168.0.9 backupserver >> /etc/hosts
```

添加了条目之后，任何时候解析`backupserver`，都会返回192.168.0.9。

### 6. 显示路由表信息

多个网络相互连接是很常见的场景。例如在大学里，不同的职能部门可能处于一个网络中。在这种情况下，一个网络中的设备如果想同另一个网络中的设备进行通信，就需要借助某个同时连接了两个网络的设备。这个特殊的设备被称为**网关**，它的作用是在不同的网络中转发分组。

操作系统维护着一个叫做**路由表**的表格，它包含了关于分组如何转发以及通过网络中的哪些节点转发的信息。可以用下面的方法显示路由表：

```
$ route
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref  UseIface
192.168.0.0      *               255.255.252.0   U        2     0    0wlan0
link-local       *               255.255.0.0     U       1000   0    0wlan0
default          p4.local        0.0.0.0         UG        0     0    0wlan0
```

也可以使用：

```
$ route -n
Kernel IP routing table
Destination      Gateway          Genmask         Flags Metric Ref  Use  Iface
192.168.0.0      0.0.0.0          255.255.252.0   U        2     0    0   wlan0
169.254.0.0      0.0.0.0          255.255.0.0     U       1000   0    0   wlan0
0.0.0.0          192.168.0.4     0.0.0.0         UG        0     0    0   wlan0
```

`-n`指定以数字形式显示地址。如果使用`-n`，`route`会以数字形式的IP地址显示每一个条目；否则，如果IP地址具有对应的DNS条目，就会显示符号形式的主机名。

设置默认网关：

```
# route add default gw IP_ADDRESS INTERFACE_NAME
```

例如：

```
# route add default gw 192.168.0.1 wlan0
```

### 7.2.4 参考

- 1.3节讲解了PATH变量。
- 4.3节讲解了grep命令。

## 7.3 使用 ping

`ping`是每位用户都应该首先了解的最基础的网络命令。绝大多数操作系统上都包含了该命

令。ping也是一个验证网络上两台主机连通性的诊断工具，能够找出网络上的活动主机。让我们看看该命令的用法。

### 7.3.1 实战演练

为了检查网络上两台主机之间的连通性,ping命令使用互联网控制消息协议(Internet Control Message Protocol, ICMP)中的echo分组。当向某台主机发送echo分组时,如果分组能够送达且该主机处于活动状态,那么它就会返回一条回应。

检查某台主机是否可以到达:

```
$ ping ADDRESS
```

ADDRESS可以是主机名、域名或者IP地址。

ping会连续发送分组,回应信息将被打印在终端上。用Ctrl+C来停止ping命令。

□ 如果主机可以到达,那么输出信息如下所示:

```
$ ping 192.168.0.1
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=1.44 ms
^C
--- 192.168.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.440/1.440/1.440/0.000 ms

$ ping google.com
PING google.com (209.85.153.104) 56(84) bytes of data.
64 bytes from bom01s01-in-f104.1e100.net (209.85.153.104): icmp_seq=1 ttl=53
time=123 ms
^C
--- google.com ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 123.388/123.388/123.388/0.000 ms
```

□ 如果主机不可到达,则输出如下所示:

```
$ ping 192.168.0.99
PING 192.168.0.99 (192.168.0.99) 56(84) bytes of data.
From 192.168.0.82 icmp_seq=1 Destination Host Unreachable
From 192.168.0.82 icmp_seq=2 Destination Host Unreachable
```

如果主机不可到达,ping返回错误信息“Destination Host Unreachable”。



网络管理员通常会对路由器进行配置，使其不响应ping命令。这样做是为了降低安全风险，因为ping可以被攻击者（使用蛮力）用来获取主机的IP地址。

### 7.3.2 补充内容

除了检查网络上两点之间的连通性，ping命令还可以通过其他选项来获取有用信息。让我们看看ping的其他选项。

#### 1. 往返时间

ping命令可以用来得出网络上两台主机之间的往返时间（Round Trip Time, RTT）。它是分组从源主机到目的主机一来一回的时间。RTT的单位是毫秒，该时间可以从ping命令中获知。例如：

```
--- google.com ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4000ms
rtt min/avg/max/mdev = 118.012/206.630/347.186/77.713 ms
```

其中，最小的RTT是118.012ms，平均RTT是206.630ms，最大RTT是347.186ms。ping输出中的mdev（77.713ms）代表平均偏差（mean deviation）。

#### 2. 限制发送的分组数量

ping命令会不停地发送echo分组并等待回复，直到按下Ctrl+C为止。我们可以用选项 -c 限制所发送的echo分组的数量。用法如下：

```
-c COUNT
```

例如：

```
$ ping 192.168.0.1 -c 2
PING 192.168.0.1 (192.168.0.1) 56(84) bytes of data.
64 bytes from 192.168.0.1: icmp_seq=1 ttl=64 time=4.02 ms
64 bytes from 192.168.0.1: icmp_seq=2 ttl=64 time=1.03 ms

--- 192.168.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.039/2.533/4.028/1.495 ms
```

在上面的例子中，ping命令发送了2个echo分组后就停止发送。果我们需要通过脚本ping一组IP地址来检查主机的状态，那么这个技巧就能派上用场了。

#### 3. ping命令的返回状态

ping命令如果执行顺利，会返回退出状态0；否则，返回非0。执行顺利意味着目的主机能够到达，否则意味着目的主机不可到达。

返回状态可以通过下面的方法轻松获得：

```
$ ping domain -c2
if [ $? -eq 0 ];
then
    echo Successful ;
else
    echo Failure
fi
```

#### 4. Traceroute

当应用程序通过互联网请求服务时，服务器可能位于远端，两者之间通过多个网关或设备节点相连。分组要穿过这些网关才能到达目的地。有一个很有意思的命令traceroute，它可以显示分组途径的所有网关的地址。traceroute信息可以帮助我们搞明白分组到达目的地需要经过多少跳（hop）。中途的网关或路由器的数量给出了一个测量网络上两个节点之间距离的度量（metric）。traceroute的输出如下：

```
$ traceroute google.com
traceroute to google.com (74.125.77.104), 30 hops max, 60 byte packets
1 gw-c6509.lxb.as5577.net (195.26.4.1) 0.313 ms 0.371 ms 0.457 ms
2 40g.lxb-fra.as5577.net (83.243.12.2) 4.684 ms 4.754 ms 4.823 ms
3 de-cix10.net.google.com (80.81.192.108) 5.312 ms 5.348 ms 5.327 ms
4 209.85.255.170 (209.85.255.170) 5.816 ms 5.791 ms 209.85.255.172
  (209.85.255.172) 5.678 ms
5 209.85.250.140 (209.85.250.140) 10.126 ms 9.867 ms 10.754 ms
6 64.233.175.246 (64.233.175.246) 12.940 ms 72.14.233.114
  (72.14.233.114) 13.736 ms 13.803 ms
7 72.14.239.199 (72.14.239.199) 14.618 ms 209.85.255.166
  (209.85.255.166) 12.755 ms 209.85.255.143 (209.85.255.143) 13.803 ms
8 209.85.255.98 (209.85.255.98) 22.625 ms 209.85.255.110
  (209.85.255.110) 14.122 ms
*
9 ew-in-f104.1e100.net (74.125.77.104) 13.061 ms 13.256 ms 13.484 ms
```



如今的Linux发布版中还包括了一个命令mtr，它类似于traceroute，但是能够显示实时刷新的数据。对于检查网络线路质量等问题很有帮助。

## 7.4 列出网络上所有的活动主机

当我们管理大型局域网时，可能需要检查网络上的其他主机是否处于活动状态。一台非活动主机可能有两种情况：要么是没有开机，要么是网络连接有问题。借助shell脚本，我们可以轻易找出并报告网络上的哪一台主机处于活动状态。让我们看看这是如何实现的。

### 7.4.1 预备知识

在这则攻略中，我们采用两种方法。第一种方法使用ping，第二种方法使用fping。在脚本中使用fping更容易些，而且相对于ping，它具有更多的特性。fping默认并没有包含在Linux发布版中，需要用软件包管理器手动安装。

### 7.4.2 实战演练

让我们看看可以找出网络上所有活动主机的shell脚本，以及实现同一目标的另一种方法。

#### (1) 方法1

我们可以用ping命令编写脚本来查询一组IP地址，检查它们是否处于活动状态：

```
#!/bin/bash
# 文件名: ping.sh
#用途: 根据网络配置对网络地址192.168.0进行修改。

for ip in 192.168.0.{1..255} ;
do
    ping $ip -c 2 &> /dev/null ;

    if [ $? -eq 0 ];
    then
        echo $ip is alive
    fi
done
```

输出如下：

```
$ ./ping.sh
192.168.0.1 is alive
192.168.0.90 is alive
```

#### (2) 使用fping

我们可以用已有的命令行工具来查询网络上的主机状态：

```
$ fping -a 192.160.1/24 -g 2> /dev/null
192.168.0.1
192.168.0.90
```

或者，使用：

```
$ fping -a 192.168.0.1 192.168.0.255 -g
```

### 7.4.3 工作原理

在方法1中，我们用ping命令找出网络上的活动主机。这里用了一个for循环对一组IP地址进行迭代。这组IP地址依照表达式192.168.0.{1..255}来生成。像{start..end}这种记法会由shell对其进行扩展并生成一组IP地址，例如192.168.0.1、192.168.0.2、192.168.0.3等，直到192.168.0.255。

ping \$ip -c 2 &> /dev/null会在每次循环中ping对应的IP地址。-c 2将发送的echo分组数量限制为2。&> /dev/null用于将stderr和stdout重定向到/dev/null，使对应的信息不会在终端上打印出来。我们用 \$? 获取退出状态。如果顺利退出，退出状态为0，否则为非0。因此能够ping通的IP地址就被打印出来。

在这个脚本中，ping命令都是依次执行的。尽管所有的IP地址都是彼此独立，但由于编写的是顺序式程序（sequential program），ping命令也只能按顺序执行。每执行一次ping命令，都要经历一段延迟：发送两个echo分组，接收回应或等待回应超时。

### 7.4.4 补充内容

我们已经讨论过了一种找出网络中所有活动主机的方法。接下来再看一些对于已经方法的改进以及另一种全新的实现方法。

#### 1. 并行ping

要是处理255个地址的话，累积下来延迟时间可就不短了。我们可以利用并行方式来加速ping命令的执行。要使ping命令可以并行执行，可将循环体放入( )&。( )中的命令作为子shell来运行，而&会将其置入后台。例如：

```
#!/bin/bash
# 文件名: fast_ping.sh
#用途: 根据网络配置对网络地址192.168.0进行修改。

for ip in 192.168.0.{1..255} ;
do
    (
        ping $ip -c2 &> /dev/null ;

        if [ $? -eq 0 ];
        then
            echo $ip is alive
        fi
    )&
done
wait
```

在for循环体中执行多个后台进程，然后结束循环并终止脚本。要想等所有子进程结束之后再终止脚本，就得使用wait命令。将wait放在脚本最后，它就会一直等到所有的子脚本进程全部结束。

## 2. 使用fping

第二种方法使用了另一个命令fping。它同时可以ping一组IP地址，而且响应速度非常快。fping的选项如下：

- ❑ 选项 -a 指定打印出所有活动主机的IP地址；
- ❑ 选项 -u 指定打印出所有无法到达的主机；
- ❑ 选项 -g 指定从 "IP地址/子网掩码"记法或者"IP地址范围"记法中生成一组IP地址；

```
$ fping -a 192.160.1/24 -g
```

或者

```
$ fping -a 192.160.1 192.168.0.255 -g
```

- ❑ 2>/dev/null将由于主机无法到达所产生的错误信息打印到null设备。

也可以采用命令行参数的方式手动指定一组IP地址，或者从stdin中接收。例如：

```
$ fping -a 192.168.0.1 192.168.0.5 192.168.0.6
# 将IP地址作为参数传递
$ fping -a < ip.list
# 从文件中传递一组IP地址
```

## 7.4.5 参考

- ❑ 1.6节讲解了数据重定向。
- ❑ 1.17节讲解了数字比较。

## 7.5 使用 SSH 在远程主机上运行命令

SSH是一个很有意思的系统管理工具，它能够让你访问远程计算机上的shell，进而执行各种命令。SSH是Secure Shell的缩写，因为它使用加密通道来传输网络数据。这则攻略介绍了在远程主机上运行命令的各种方法。

### 7.5.1 预备知识

GNU/Linux发布版中默认都不包括SSH，需要使用软件包管理器安装openssh-server和openssh-client软件包。SSH服务默认运行在端口22之上。

## 7.5.2 实战演练

(1) 连接运行SSH服务器的远程主机:

```
$ ssh username@remote_host
```

其中:

- ☐ username是远程主机上的用户;
- ☐ remote\_host可以是域名或IP地址。

例如:

```
$ ssh mec@192.168.0.1
The authenticity of host '192.168.0.1 (192.168.0.1)' can't be established.
RSA key fingerprint is 2b:b4:90:79:49:0a:f1:b3:8a:db:9f:73:2d:75:d6:f9.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.1' (RSA) to the list of known hosts.
Password:

Last login: Fri Sep  3 05:15:21 2010 from 192.168.0.82
mec@proxy-1:~$
```

ssh采用交互方式询问用户密码,一旦认证成功,将会为用户返回一个shell。



SSH执行指纹核对 (fingerprint verification) 来确保用户连接到正确的远程主机。这是为了避免**中间人攻击** (man-in-the-middle attack), 在这类攻击中, 攻击者试图假扮成另一台计算机。在第一次连接到服务器上时, SSH默认会存储指纹信息, 在今后的连接过程中核对该指纹。

SSH服务器默认在端口22上运行。但有些SSH服务器并没有使用这个端口。针对这种情况, 可以用SSH的-p port\_num来指定端口。

(2) 连接运行在端口422之上的SSH服务器:

```
$ ssh user@localhost -p 422
```

你可以在远程主机的shell中执行命令。但是在shell脚本中使用SSH时, 并不需要交互式的shell, 因为我们需要执行多条命令并显示或存储命令输出。



每次都要输入密码对于自动化脚本来说显然不实际, 因此要对SSH进行无密码登录配置。7.8节将讲解具体的配置方法。

- (3) 要想在远程主机中执行命令，并将命令输出显示在本地shell中，使用下面的语法：

```
$ ssh user@host 'COMMANDS'
```

例如：

```
$ ssh mec@192.168.0.1 'whoami'
mec
```

可以输入多条命令，在命令之间以分号进行分隔：

```
$ ssh user@host "command1 ; command2 ; command3"
```

例如：

```
$ ssh mec@192.168.0.1 "echo user: $(whoami);echo OS: $(uname)"
Password:
user: mec
OS: Linux
```

在这个例子中，在远程主机上执行的命令是：

```
echo user: $(whoami);
echo OS: $(uname)
```

写成一般的形式就是：

```
COMMANDS="command1; command2; command3"
$ ssh user@hostname "$COMMANDS"
```

我们也可以在命令序列中用子shell操作符传递一个更复杂的子shell。

- (4) 让我们来编写一个基于SSH的shell脚本，它用来收集一组远程主机的运行时间（uptime）。运行时间是系统加电运行的时间，uptime命令可以用来显示系统加电后运行了多久。

假设在IP\_LIST中的所有系统都有一个用户test。

```
#!/bin/bash
# 文件名: uptime.sh
# 用途: 系统运行时间监视器

IP_LIST="192.168.0.1 192.168.0.5 192.168.0.9"
USER="test"

for IP in $IP_LIST;
do
    uptime=$(ssh ${USER}@${IP} uptime | awk '{ print $3 }' )
    echo $IP uptime: $uptime
done
```

输出如下：

```
$ ./uptime.sh
192.168.0.1 uptime: 1:50,
192.168.0.5 uptime: 2:15,
192.168.0.9 uptime: 10:15,
```

### 7.5.3 补充内容

让我们看看ssh命令的另一些选项。

#### 1. SSH的压缩功能

SSH协议也支持对数据进行压缩传输。当带宽有限时，这一功能很方便。用ssh命令的选项-c启用这一功能：

```
$ ssh -C user@hostname COMMANDS
```

#### 2. 将数据重定向至远程shell命令的stdin

有时候我们需要将一些数据重定向到远程shell命令的stdin。下面是实现方法：

```
$ echo 'text' | ssh user@remote_host 'echo'
text
```

或者

```
# 将文件中的数据重定向
$ ssh user@remote_host 'echo' < file
```

在远程主机上，echo打印出通过stdin接收到的数据，但这些数据却是从本地主机传递到远程shell的stdin中的。

#### 3. 在远程主机中执行图形化命令

如果你打算使用刚才介绍过的方法来运行一些需要显示某种形式的GUI的命令，就会碰上类似于“无法显示”之类的错误。这是因为ssh shell无法连接到远程主机上的X服务器。对此，你需要像这样设置变量\$DISPLAY：

```
ssh user@host "export DISPLAY=:0 ; command1; command2""
```

这将启用远程主机上的图形化输出。如果你想在本地主机上也显示图形化输出，使用SSH的X11转发选项（forwarding option）：

```
ssh -X user@host "command1; command2"
```

该命令会在远程主机上执行，但图形化输出会显示在本地主机上。

### 7.5.4 参考

7.8节讲解了如何实现在不输入密码的情况下，自动登录执行远程命令。

## 7.6 通过网络传输文件

计算机联网的主要目的就是资源共享。在资源共享方面，使用最多的是文件共享。有多种方法可以用来在网络中传输文件。这则攻略就讨论了如何用常见的协议FTP、SFTP、RSYNC和SCP传输文件。

### 7.6.1 预备知识

用来在网络上传输文件的命令多数都已默认包含在安装好的Linux中。通过FTP传输文件可以使用lftp命令，通过SSH连接传输文件可以使用sftp，RSYNC使用SSH与rsync命令，scp通过SSH进行传输。

### 7.6.2 实战演练

文件传输协议（File Transfer Protocol，FTP）是一个古老的用于在网络主机之间传输文件的文件传输协议。我们可以用lftp命令访问FTP服务器来进行文件传输。只有远程主机上安装有FTP服务器才能使用FTP。很多公共网站都是用FTP共享文件，它使用端口21。

要连接FTP服务器传输文件，可以使用：

```
$ lftp username@ftphost
```

它会提示你输入密码，然后显示一个像下面那样的登录提示符：

```
lftp username@ftphost:~>
```

你可以在提示符后输入命令，如下所示。

- ☐ 用cd directory改变目录。
- ☐ 用lcd改变本地主机的目录。
- ☐ 用mkdir创建目录。
- ☐ 列出远程机器当前目录下的文件使用ls。
- ☐ 用get filename下载文件：

```
lftp username@ftphost:~> get filename
```

- ☐ 用put filename从当前目录上传文件：

```
lftp username@ftphost:~> put filename
```

❑ 用quit退出lftp会话。

lftp提示符支持命令自动补全。

### 7.6.3 补充内容

让我们看看其他可用于网络文件传输的技术及命令。

#### 1. FTP自动传输

ftp是另一个可用于FTP文件传输的命令。相比较而言，lftp的用法更灵活。lftp和ftp为用户启动一个交互式会话（通过显示消息来提示用户输入）。如果我们不使用交互模式，而是希望进行自动文件传输，又该怎么做呢？下面的脚本可以用来实现FTP自动传输：

```
#!/bin/bash
# 文件名: ftp.sh
# 用途: 自动化 FTP 传输
HOST='domain.com'
USER='foo'
PASSWD='password'
ftp -i -n $HOST <<EOF
user ${USER} ${PASSWD}
binary
cd /home/slynux
puttestfile.jpg
getserverfile.jpg
quit
EOF
```

上面的脚本包含下列结构：

```
<<EOF
DATA
EOF
```

这是用来通过stdin向FTP命令发送数据。1.6节中已讲解了重定向到stdin的各种方法。

在示例脚本中，ftp的选项-i关闭用户的交互会话，user \${USER} \${PASSWD}设置用户名和密码，binary将文件模式设置为二进制模式。

#### 2. SFTP（Secure FTP，安全FTP）

SFTP是一个类似于FTP的文件传输系统，它运行在SSH连接之上并模拟成FTP接口。它不需要远端运行FTP服务器来进行文件传输，但必须安装并运行OpenSSH服务器。SFTP是一个交互式命令，提供了命令提示符。

下面的命令可用于文件传输。对于特定主机、用户和密码的自动化FTP会话来说，命令都是一样的。

```
cd /home/slynux
put testfile.jpg
get serverfile.jpg
```

运行sftp:

```
$ sftp user@domainname
```

和lftp类似，输入quit命令可以退出sftp会话。

SSH服务器有时候并不在默认的端口22上运行。如果它在其他端口运行，我们可以在sftp中用 -oPort=PORTNO来指定端口号。例如：

```
$ sftp -oPort=422 user@slynux.org
```

-oPort应该作为sftp命令的第一个参数。

### 3. rsync命令

作为一款重要的命令行工具，rsync广泛用于网络文件复制及系统备份。6.8节详细讲解了rsync的用法。

### 4. SCP（Secure Copy Program，安全复制程序）

就传统的远程复制工具rcp而言，SCP是一项更安全的文件复制技术。文件均通过SSH加密通道进行传输。我们可以像下面这样轻松地将文件传输到远程主机：

```
$ scp filename user@remotehost:/home/path
```

该命令会提示你输入密码，可以用SSH自动登录功能来免于输入密码。7.8节会讲解SSH自动登录。因此，用scp传输文件无需特定的脚本。一旦实现了SSH自动登录，scp就可以直接执行，而不再需要提示输入密码。

命令中的remotehost可以是IP地址或域名。scp命令的格式是：

```
$ scp SOURCE DESTINATION
```

SOURCE或DESTINATION可以采用形如username@localhost:/path的格式。例如：

```
$ scp user@remotehost:/home/path/filename filename
```

上面的命令将远程主机中的文件复制到当前目录，并采用给定的文件名。

如果SSH没有运行在端口22，使用 -oPort，并采用和sftp相同的语法。

## 5. 用scp进行递归复制

使用scp的-r选项，我们可以在两台网络主机之间对文件夹进行递归复制：

```
$ scp -r /home/slynux user@remotehost:/home/backups
# 将目录/home/slynux递归复制到远程主机中
```

scp的-p选项能够在复制文件的同时保留文件的权限和模式。

## 7.6.4 参考

1.6节讲解了如何用EOF实现标准输入。

## 7.7 连接无线网络

配置以太网很简单，因为它使用物理线缆，无需认证之类的特殊要求。但是无线LAN可能就得进行认证了，比如WEP密钥以及所要连接的无线网络的ESSID (Extended Service Set Identification, 扩展服务集标识)。ESSID是无线网络的名称。让我们看看如何用脚本来连接无线网络。

### 7.7.1 预备知识

我们需要用ifconfig分配IP地址和子网掩码才能连接上有线网络。对于无线网络来说，还需要其他工具（如iwconfig和iwlist）来配置更多的参数。

### 7.7.2 实战演练

连接启用了WEP (Wried Equivalent Privacy, 有线等效加密) 的连接无线LAN的脚本如下：

```
#!/bin/bash
#文件名: wlan_connect.sh
#用途: 连接无线 LAN

#根据你的设置修改下面的参数
##### PARAMETERS #####
IFACE=wlan0
IP_ADDR=192.168.1.5
SUBNET_MASK=255.255.255.0
GW=192.168.1.1
HW_ADDR='00:1c:bf:87:25:d2'
#如果不想使用物理地址欺骗，把上面这一行注释掉

ESSID="homenet"
WEP_KEY=8b140b20e7
FREQ=2.462G
#####
```

```
KEY_PART=""

if [[ -n $WEP_KEY ]];
then
    KEY_PART="key $WEP_KEY"
fi
#设置新的配置之前先关闭接口
/sbin/ifconfig $IFACE down

if [ $UID -ne 0 ];
then
    echo "Run as root"
    exit 1;
fi

if [[ -n $HW_ADDR ]];
then
    /sbin/ifconfig $IFACE hw ether $HW_ADDR
    echo Spoofed MAC ADDRESS to $HW_ADDR
fi

/sbin/iwconfig $IFACE essid $ESSID $KEY_PART freq $FREQ

/sbin/ifconfig $IFACE $IP_ADDR netmask $SUBNET_MASK

route add default gw $GW $IFACE

echo Successfully configured $IFACE
```

### 7.7.3 工作原理

命令ifconfig、iwconfig和route必须以root用户身份运行，因此在脚本一开始要检查是否为root用户。

无线LAN需要**ssid**、**密钥**、**频率**等参数。**ssid**是我们想要连接的无线网络名称。一些网络需要用**WEP**密钥进行认证，**WEP**密钥通常是一个5位或10位十六进制数口令。频率则是分配给特定网络的，**iwconfig**命令用它将无线网卡同对应的无线网络联系起来。

我们可以用iwlist工具扫描并列出的无线网络。用下面的命令进行扫描：

```
# iwlist scan
wlan0      Scan completed :
           Cell 01 - Address: 00:12:17:7B:1C:65
           Channel:11
           Frequency:2.462 GHz (Channel 11)
           Quality=33/70   Signal level=-77 dBm
```

```
Encryption key: on
ESSID: "model-2"
```

可以从扫描结果中的Frequency:2.462 GHz (Channel 11)一行中提取频率参数。



为简单起见这个示例使用WEP，但需要说明的是，就目前来讲WEP是不安全的。如果你要负责管理无线网络，最好使用Wi-Fi保护访问2（WPA2）加密方式，以确保安全。

### 7.7.4 参考

1.17节讲解了字符串比较。

## 7.8 用 SSH 实现无密码自动登录

SSH广泛用于脚本自动化，它使得我们可以在远程主机上执行命令并读取输出。SSH通常使用用户名和密码进行认证，在其执行过程中会提示输入密码。但是在自动化脚本中要求用户输入密码就显然不实际了。因此需要将登录过程自动化。SSH包含了一个内建特性，可以用SSH密钥实现自动登录。这则攻略描述了如何创建SSH密钥并协助实现自动登录。

### 7.8.1 预备知识

SSH采用了非对称加密技术，认证密钥包含两部分：一个公钥和一个私钥。我们可以通过ssh-keygen命令创建认证密钥。要想实现自动化认证，公钥必须放置在服务器中（将其加入文件~/.ssh/authorized\_keys），与公钥对应的私钥应该放入登录客户机的~/.ssh目录中。另一些与SSH相关的配置信息（例如，authorized\_keys文件的路径与名称）可以通过修改文件/etc/ssh/sshd\_config进行配置。

### 7.8.2 实战演练

设置SSH自动化认证需要两步：

- (1) 创建SSH密钥，这用于登录远程主机；
- (2) 将生成的公钥传给远程主机，并将其加入文件~/.ssh/authorized\_keys中。

输入命令ssh-keygen创建SSH密钥，并指定加密算法类型为RSA：

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/slynux/.ssh/id_rsa):
Created directory '/home/slynux/.ssh'.
```

```

Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/slynux/.ssh/id_rsa.
Your public key has been saved in /home/slynux/.ssh/id_rsa.pub.
The key fingerprint is:
f7:17:c6:4d:c9:ee:17:00:af:0f:b3:27:a6:9c:0a:05 slynux@slynux-laptop
The key's randomart image is:
+--[ RSA 2048 ]-----+
|           .           |
|            o . .      |
|       E      o o      |
|    ...oo |           |
|      .S .+  +o.      |
|      . . . = . . .   |
|    .+.o...|           |
|      . . + o. .      |
|      ..+             |
+-----+

```

你需要输入一个口令来生成一对公钥和私钥。如果不输入的话，也可以生成密钥，但是这样做可不安全。我们可以编写监控脚本，利用自动登录来登入多台主机。对于这种情况，在运行ssh-keygen命令时，不要填入口令，这样就能够避免在脚本运行时向你索要口令了。

现在 ~/.ssh/id\_rsa.pub和 ~/.ssh/id\_rsa已经生成了。id\_dsa.pub是生成的公钥，id\_dsa是生成的私钥。公钥必须添加到远程服务器 ~/.ssh/authorized\_keys文件中，这台服务器也正是我们想从当前主机自动登入的那台服务器。

要添加一个密钥文件，可以使用：

```

$ ssh USER@REMOTE_HOST "cat >> ~/.ssh/authorized_keys" < ~/.ssh/id_rsa.pub
Password:

```

在上面的命令中要提供登录密码。

自动登录就已经设置好了。从现在开始，SSH在运行过程中就不会再提示输入密码。你可以用下面的命令来测试：

```

$ ssh USER@REMOTE_HOST uname
Linux

```

这样就不会提示你输入密码了。



多数Linux发布版中有一个叫做ssh-copy-id的工具，它可以自动将私钥加入远程服务器的authorized\_keys文件中。用法如下：

```
ssh-copy-id USER@REMOTE_HOST
```

## 7.9 使用 SSH 进行端口转发

端口转发允许其他计算机利用你的主机来连接到远程服务器上的特定服务。举例而言，假设你的主机分配的IP地址是192.168.1.2，而且还拥有互联网连接。如果你将主机端口8000上的流量转发到www.kernel.org的端口80上，那么其它计算机就可以使用浏览器通过访问http://192.168.1.2:8000来进入Linux内核站点了。来看看如何实现这一切。

### 7.9.1 实战演练

你可以将本地主机端口上的流量转发到另一台主机上，也可以将远程主机端口上的流量转发到其它主机。按照下面的方法，一旦端口转发设置完毕，你会得到一个shell提示符。在进行转发的过程中，这个shell必须保持打开状态，什么时候想停止端口转发，只需要退出该shell就可以了。

- (1) 使用下列命令将本地主机端口8000上的流量转发到www.kernel.org的端口80上：

```
ssh -L 8000:www.kernel.org:80 user@localhost
```

将上述命令中的user替换成本地主机上的用户名。

- (2) 使用下列命令将远程主机端口8000上的流量转发到www.kernel.org的端口80上：

```
ssh -L 8000:www.kernel.org:80 user@REMOTE_MACHINE
```

将上述命令中的REMOTE\_MACHINE替换成远程主机的主机名或IP地址，将user替换成使用SSH进行访问的用户名。

### 7.9.2 补充内容

使用非交互模式或者反向端口转发可以使端口转发发挥更大的作用。来看一看这两者的用法。

#### 1. 非交互式端口转发

如果你只是想设置端口转发，而不希望总是保持一个打开状态的shell，那么可以像下面这样使用ssh：

```
ssh -fL 8000:www.kernel.org:80 user@localhost -N
```

-f指定ssh在执行命令前转入后台运行，-L指定远程主机的登录名，-N告诉ssh无需执行命令，只进行端口转发。

#### 2. 反向端口转发

反向端口转发是SSH最强大的特性之一。如果你有一台无法通过互联网进行访问的主机，但

是又希望其他用户可以访问到这台主机上的服务，那就是反向端口转发大显身手的时候了。如果你使用SSH访问一台可以通过互联网访问的远程主机，那么就可以在这台主机上设置反向端口转发，将流量转发到运行服务的本地主机上。

反向端口转发的设置同端口转发非常类似：

```
ssh -R 8000:localhost:80 user@REMOTE_MACHINE
```

上述命令会将远程主机端口8000上的流量转发到本地主机的端口80上。和之前一样，别忘了把REMOTE\_MACHINE替换成远程主机的主机名或IP地址。

利用这种方法，如果你在远程主机上浏览http://localhost，那么实际连接的是运行在本地主机端口8000上的Web服务器。

## 7.10 在本地挂载点上挂载远程驱动器

在执行数据读写操作时，如果可以通过本地挂载点访问远程主机文件系统，那就再好不过了。SSH是网络中最常用的文件传输协议，因此可以利用它和sshfs来实现这一需求。sshfs允许你将远程文件系统挂载到本地挂载点上。让我们看看实现方法。

### 7.10.1 预备知识

GNU/Linux发布版默认并不包含sshfs。请使用软件包管理器来自行安装。sshfs是FUSE文件系统软件包的一个扩展，允许它支持的操作系统像使用本地文件系统一样挂载各类数据。

有关FUSE更多的信息，请访问<http://fuse.sourceforge.net/>。

### 7.10.2 实战演练

将位于远程主机上的文件系统挂载到本地挂载点上：

```
# sshfs -o allow_other user@remotehost:/home/path /mnt/mountpoint
Password:
```

在收到提示时输入用户密码。现在位于远程主机/home/path中的数据就可以通过本地挂载点/mnt/mountpoint来访问了。

完成任务后，可用下面的方法卸载：

```
# umount /mnt/mountpoint
```

### 7.10.3 参考

7.5节讲解了ssh命令。

## 7.11 网络流量与端口分析

网络端口是网络应用程序必不可少的参数。应用程序在主机上打开端口，通过远程主机中打开的端口实现远程通信。出于安全方面的考虑，必须留意系统中开放及关闭的端口。恶意软件和Rootkit可能会利用特定的端口及服务运行在系统中，从而使攻击者可以对数据及资源进行未经授权的访问。通过获取开放端口列表以及运行在端口之上的服务，便可以分析并抵御Rootkit，而且这些信息还有助于对其进行清除。开放端口列表不仅能够协助检测恶意软件，而且还能够收集开放端口的相关信息，以便调试网络应用程序。它可以用来分析端口连接及端口侦听功能是否正常。这则攻略讨论了各种用于端口分析的工具。

### 7.11.1 预备知识

很多命令可用来列出端口以及运行在端口上的服务（例如lsof和netstat）。这些命令都默认包含在所有的GNU/Linux发布版中。

### 7.11.2 实战演练

要列出系统中的开放端口以及运行在端口上的服务的详细信息，可以使用以下命令：

```
$ lsof -i
```

COMMAND	PID	USER	FD	TYPE	DEVICE	SIZE/OFF	NODE	NAME
firefox-b	2261	slynux	78u	IPv4	63729	0t0	TCP	localhost:47797->localhost:42486 (ESTABLISHED)
firefox-b	2261	slynux	80u	IPv4	68270	0t0	TCP	slynux-laptop.local:41204->192.168.0.2:3128 (CLOSE_WAIT)
firefox-b	2261	slynux	82u	IPv4	68195	0t0	TCP	slynux-laptop.local:41197->192.168.0.2:3128 (ESTABLISHED)
ssh	3570	slynux	3u	IPv6	30025	0t0	TCP	localhost:39263->localhost:ssh (ESTABLISHED)
ssh	3836	slynux	3u	IPv4	43431	0t0	TCP	slynux-laptop.local:40414->boney.mtveurope.org:422 (ESTABLISHED)
GoogleTal	4022	slynux	12u	IPv4	55370	0t0	TCP	localhost:42486 (LISTEN)
GoogleTal	4022	slynux	13u	IPv4	55379	0t0	TCP	localhost:42486->localhost:32955 (ESTABLISHED)

lsof的每一项输出都对应着一个打开了特定端口的服务。输出的最后一列类似于：

```
laptop.local:41197->192.168.0.2:3128
```

输出中的laptop.local:41197对应本地主机，192.168.0.2:3128对应远程主机。41197是本地主机当前的开放端口，3128是连接远程主机服务所需要使用的端口。

要列出本地主机当前的开放端口，可以使用：

```
$ lsof -i | grep ":[0-9]\+-->" -o | grep "[0-9]\+" -o | sort | uniq
```

### 7.11.3 工作原理

第一个grep中使用的正则表达式：`[0-9]\+-->`用来从lsof输出中提取主机端口部分（`:34395-->`）；第二个grep用来提取端口号（数字）。同一个端口可能会有多个连接，因此相同的端口也许会出现多次。为了保证每个端口只显示一次，将端口号排序并打印出不重复的部分。

### 7.11.4 补充内容

让我们看看另一些用来查看开放端口以及网络流量相关信息的工具。

#### 用netstat查看开放端口与服务

netstat是另一个可用于网络服务分析的命令。讲解netstat的所有特性超出了这则攻略的范围。这里只看如何用它列出服务与端口号。

用netstat -tnp列出开放端口与服务：

```
$ netstat -tnp
(Not all processes could be identified, non-owned process info
will not be shown, you would have to be root to see it all.)
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
PID/Program name
tcp        0      0 192.168.0.82:38163      192.168.0.2:3128
ESTABLISHED 2261/firefox-bin
tcp        0      0 192.168.0.82:38164      192.168.0.2:3128      TIME_
WAIT      -
tcp        0      0 192.168.0.82:40414      193.107.206.24:422
ESTABLISHED 3836/ssh
tcp        0      0 127.0.0.1:42486         127.0.0.1:32955
ESTABLISHED 4022/GoogleTalkPlug
tcp        0      0 192.168.0.82:38152      192.168.0.2:3128
ESTABLISHED 2261/firefox-bin
tcp6       0      0 :::1:22                 :::1:39263
ESTABLISHED -
tcp6       0      0 :::1:39263              :::1:22
ESTABLISHED 3570/ssh
```

## 7.12 创建套接字

对于已有的应用，例如文件传输、远程shell等，我们有现成的工具可以使用（ftp和ssh）。

但有时候你可能需要一些定制的网络应用。比如写一个脚本，当远程客户连接到主机时执行某些操作。在这则攻略中，我们会创建简单的套接字并用它们进行通信。

### 7.12.1 预备知识

要实现这一切，需要创建网络套接字，使得我们可以通过TCP/IP网络进行数据传输。最简单的方法就是使用netcat命令（或nc）。我们需要两个套接字：一个用来侦听，一个用来连接。

### 7.12.2 实战演练

- (1) 设置侦听套接字：

```
nc -l 1234
```

这会在本地主机的端口1234上创建一个侦听套接字。

- (2) 连接到该套接字：

```
nc HOST 1234
```

如果是在运行着侦听套接字的主机上执行该命令，那么需要将HOST更换成localhost，或将其更换成其他主机的IP地址或主机名。

- (3) 要想发送消息，只需要在执行第2步操作的主机终端中输入信息并按回车键就行了。消息会出现在执行第1步操作的主机终端中。

### 7.12.3 补充内容

网络套接字可不仅能用于发送文本。接着往下看。

在网络上进行快速文件复制

我们可以利用netcat和shell重定向来简化网络中的文件复制操作。

- (1) 在接收端执行下列命令：

```
nc -l 1234 > destination_filename
```

- (2) 在发送端执行下列命令：

```
nc HOST 1234 < source_filename
```

## 7.13 互联网连接共享

在如今的计算世界里，我们平时都在使用着各种类型的设备。桌面计算机、上网本、笔记本

电脑、平板电脑、智能耳机，等等。我们需要将这些设备都连入互联网，这通常是通过无线路由器实现的。如果手边没有路由器（或者路由器坏掉了），但又需要共享使用互联网，该怎么办呢？没问题！Linux的iptables再加上几个脚本便可助你一臂之力。

### 7.13.1 预备知识

在这则攻略中，我们使用iptables设置网络地址转换（Network Address Translation, NAT），使得多个联网设备能够共享互联网连接。你需要使用iwconfig命令来获得无线接口的名称。

### 7.13.2 实战演练

- (1) 连接到互联网。在这里我们假设使用的是有线网络连接，通过eth0连接到互联网。请按照你个人的实际情况进行修改。

- (2) 使用发布版自带的网络管理工具，创建一个adhoc无线连接，设置如下：

- ❑ IP地址：10.99.66.55
- ❑ 子网掩码：255.255.0.0（16）

- (3) 使用下面的shell脚本来实现互联网连接共享：

```
#!/bin/bash
#文件名：netsharing.sh
echo 1 > /proc/sys/net/ipv4/ip_forward
iptables -A FORWARD -i $1 -o $2 -s 10.99.0.0/16 -m conntrack
--ctstate NEW -j ACCEPT
iptables -A FORWARD -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT
iptables -A POSTROUTING -t nat -j MASQUERADE
```

- (4) 执行脚本：

```
./netsharing.sh eth0 wlan0
```

其中，eth0是连接到互联网的接口，wlan0是用于共享互联网连接的无线接口。

- (5) 将设备连接到刚才创建的无线网络：

- IP地址：10.99.66.56（以此类推）。
- 子网掩码：255.255.0.0。



要想更为便捷，可以在主机上安装DHCP和DNS服务器，这样就不必手动配置IP地址了。你可以使用一个叫做dnsmasq的工具来进行DHCP和DNS操作。

## 7.14 使用 iptables 架设简易防火墙

防火墙是一种网络服务，它可以用来过滤、阻塞不需要的网络流量，允许正常的网络流量通过。Linux中最为强大的工具非iptables莫属，它目前已经被集成到了内核中。

### 7.14.1 实战演练

如今所有的Linux发布版中默认都包含了iptables。我们接下来会看到一些典型的iptables的配置方法。

(1) 阻塞发送到特定IP地址的流量：

```
#iptables -A OUTPUT -d 8.8.8.8 -j DROP
```

如果在执行iptables命令之前，从另一个终端中运行PING 8.8.8.8，你会看到如下内容：

```
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.  
64 bytes from 8.8.8.8: icmp_req=1 ttl=56 time=221 ms  
64 bytes from 8.8.8.8: icmp_req=2 ttl=56 time=221 ms  
ping: sendmsg: Operation not permitted  
ping: sendmsg: Operation not permitted
```

ping命令在执行到第三次的时候失败了，这是因为我们使用iptables将所有发送到8.8.8.8的流量给丢弃了。

(2) 阻塞发送到特定端口的流量：

```
#iptables -A OUTPUT -p tcp -dport 21 -j DROP  
$ ftp ftp.kde.org  
ftp: connect: Connection timed out
```

### 7.14.2 工作原理

iptables是Linux系统中用来架设防火墙的标准命令。iptables中的第一个选项-A表明向链(chain)中添加一条新的规则，该规则由后续参数给出。链就是一组规则的集合，在这则攻略中我们使用的是OUTPUT链，它可以对所有出站(outgoing)的流量进行控制。

在第一个例子中，-d指定了所要匹配的分组目的地址。随后使用-j来使iptables丢弃(DROP)符合条件的分组。

在第二个例子中，`-p`指定该规则是适用于TCP，`-dport`指定了对应的端口。这样我们就可以阻塞所有出站的FTP流量了。

### 7.14.3 补充内容

在使用`iptables`命令时，你可能希望清除对`iptables`链所做出的所有改动。可以使用下面的命令：

```
#iptables --flush
```

## 本章内容

- ❑ 统计磁盘使用情况
- ❑ 计算命令的执行时间
- ❑ 登录用户、启动日志及启动故障的相关信息
- ❑ 列出1小时内占用CPU最多的10个进程
- ❑ 用watch监视命令输出
- ❑ 记录文件及目录访问
- ❑ 用logrotate管理日志文件
- ❑ 用syslog记录日志
- ❑ 通过监视用户登录找出入侵者
- ❑ 监视远程磁盘的健康情况
- ❑ 找出系统中用户的活动时段
- ❑ 电源使用的测量与优化
- ❑ 监视磁盘活动
- ❑ 检查磁盘及文件系统错误

## 8.1 简介

操作系统是由一系列不同用途的系统软件组成的。为了了解这些软件是否工作正常，最好能够对其进行监视。我们可以使用一项被称为日志记录（logging）的技术，借助这项技术，应用程序在运行的时候，会将重要的信息写入某个文件中。该文件的内容能够用来了解特定程序或守护进程的操作过程。如果应用程序或服务发生了崩溃，这些信息有助于我们进行调试和故障修复。

在本章中，我们要和监视系统活动的各种命令打交道，同时还要学习日志技术及其使用方法。

## 8.2 监视磁盘使用情况

磁盘空间是一种有限资源。我们经常要统计存储介质（例如硬盘）的使用情况，以便确定可用空间。当可用空间开始捉襟见肘时，我们就得找到大体积的文件，删除或移走它们，以便腾出空间。除此之外，磁盘使用管理也用于shell脚本环境。这则攻略将会讲解用于磁盘管理的各类命令及其选项。

### 8.2.1 预备知识

df和du是Linux中用于统计磁盘使用情况的两个重要命令。df是disk free的缩写，du是disk usage的缩写。让我们看看如何用它们执行各种涉及磁盘使用情况的任务。

### 8.2.2 实战演练

找出某个文件（或多个文件）占用的磁盘空间：

```
$ du FILENAME1 FILENAME2 ..
```

例如：

```
$ du file.txt
4
```



统计结果默认以字节作为计量单位。

要获得某个目录中所有文件的磁盘使用情况，并在每一行中显示各个文件的磁盘占用详情，可以使用：

```
$ du -a DIRECTORY
```

-a递归地输出指定目录或多个目录中所有文件的统计结果。



执行du DIRECTORY也可以输出类似的结果，但是它只会显示子目录使用的磁盘空间，而不显示每个文件占用空间的情况。要想显示文件的磁盘使用情况，必须使用-a。

例如：

```
$ du -a test
4  test/output.txt
4  test/process_log.sh
4  test/pcpu.sh
16 test
```

使用du DIRECTORY的输出如下：

```
$ du test
16 test
```

### 8.2.3 补充内容

让我们看看du命令的其他用法。

#### 1. 以KB、MB或块（block）为单位显示磁盘使用情况

命令du默认显示文件占用的总字节数，但是以标准单位KB、MB或GB显示磁盘使用情况更方便人们阅读。要采用这种更友好的格式进行打印，可以使用选项 -h：

```
du -h FILENAME
```

例如：

```
$ du -h test/pcpu.sh
4.0K test/pcpu.sh
# 可以接受多个文件参数
```

或者

```
# du -h DIRECTORY
$ du -h hack/
16K hack/
```

#### 2. 显示磁盘使用总计

假如我们需要计算所有文件或目录总共占用了多少磁盘空间，那么显示单个文件的使用情况显然没什么用。du的选项 -c可以输出作为命令参数的所有文件和目录的磁盘使用情况，它会在输出结果末尾加上一行总计。语法如下：

```
$ du -c FILENAME1 FILENAME2..
```

例如：

```
du -c process_log.sh pcpu.sh
4 process_log.sh
4 pcpu.sh
8 total
```

或者

```
$ du -c DIRECTORY
```

例如：

```
$ du -c test/
16 test/
16 total
```

或者

```
$ du -c *.txt
# 通配符
```

-c可以同 -a、-h等选项组合使用。如果不使用-c，也可以得到同样的输出。唯一不同的是选项-c会添加一行磁盘使用情况总计。

另一个选项 -s (summarize, 合计) 则只输出合计数据。它可以配合 -h打印出人们易读的格式。这个命令在实践中经常使用，其语法如下：

```
$ du -s FILE(s)
$ du -sh DIRECTORY
```

例如：

```
$ du -sh slynux
680K  slynux
```

### 3. 用特定的单位打印文件

我们可以强制du使用特定的单位打印磁盘使用情况。举例如下。

❑ 打印以字节（默认输出）为单位的文件大小：

```
$ du -b FILE(s)
```

❑ 打印以KB为单位的文件大小：

```
$ du -k FILE(s)
```

❑ 打印以MB为单位的文件大小：

```
$ du -m FILE(s)
```

❑ 打印以指定块为单位的文件大小：

```
$ du -B BLOCK_SIZE FILE(s)
```

其中BLOCK\_SIZE以字节为单位。

下面的例子中包含了这些命令：

```
$ du pcpu.sh
4  pcpu.sh
$ du -b pcpu.sh
439  pcpu.sh
$ du -k pcpu.sh
4  pcpu.sh
$ du -m pcpu.sh
1  pcpu.sh
$ du -B 4 pcpu.sh
1024  pcpu.sh
```

#### 4. 从磁盘使用统计中排除部分文件

有时候我们需要从磁盘使用统计中排除部分文件。可以使用两种方法指定需要排除的文件：

##### (1) 通配符

按照下面的方法指定一个通配符：

```
$ du --exclude "WILDCARD" DIRECTORY
```

例如：

```
$ du --exclude "*.txt" FILES(s)
# 排除所有的.txt文件
```

##### (2) 排除列表

从文件中获取需要排除的文件列表：

```
$ du --exclude-from EXCLUDE.txt DIRECTORY
# EXCLUDE.txt包含了需要排除的文件列表
```

还有其他的选项可以很方便地用来限制磁盘使用情况统计的范围。我们可以用 `--max-depth` 指定 `du` 应该遍历的目录层次的最大深度。将深度指定为1，可以统计当前目录下的所有文件占用内存的情况。将深度指定为2，可以统计当前目录以及下一级子目录中文件占用内存的情况，但不包括第二级子目录。

例如：

```
$ du --max-depth 2 DIRECTORY
```

当使用 `du` 时，要确保对其遍历的目录和文件拥有适合的读权限。



可以用选项 `-x` 来限制 `du`，使它只能够对单个文件系统进行遍历。假设运行 `du DIRECTORY`，它会递归地遍历每一个可能存在的子目录。目录层次中的某个子目录可能就是一个挂载点（例如，`/mnt/sda1` 是 `/mnt` 的子目录，同时也是设备 `/dev/sda1` 的挂载点）。`du` 会遍历挂载点并统计设备文件系统的磁盘使用情况。为了避免 `du` 的这种行为，可以在使用 `du` 其他选项的同时加上选项 `-x`。`du -x /` 会将 `/mnt/` 中的所有挂载点排除在磁盘使用统计之外。

#### 5. 找出指定目录中最大的10个文件

我们经常会碰上找出大体积文件的活儿，一般是要将这些大文件删除或移走。可以用 `du` 和 `sort` 轻松地完成这项任务：

```
$ du -ak SOURCE_DIR | sort -nrk 1 | head
```

其中，`-a`使得`du`遍历`SOURCE_DIR`并计算其中所有文件的大小。由于指定了选项`-k`，输出的第一列会包含以KB为单位的文件大小，第二列则会包含文件或文件夹的名称。

`sort`对第一列依数值逆序排序。`head`用来显示前10行输出。例如：

```
$ du -ak /home/slynux | sort -nrk 1 | head -n 4
50220 /home/slynux
43296 /home/slynux/.mozilla
43284 /home/slynux/.mozilla/firefox
43276 /home/slynux/.mozilla/firefox/8c22khxc.default
```

上面这个单行脚本的缺点在于它在结果中包含了目录。如果我们只是需要找出最大的文件而不是目录，可以按照下面的方法改进脚本，使它只输出最大的文件：

```
$ find . -type f -exec du -k {} \; | sort -nrk 1 | head
```

利用`find`替`du`将文件过滤出来，而无需使用`du`进行递归遍历。

## 6. 磁盘可用空间信息

`du`提供磁盘使用情况信息，而`df`提供磁盘可用空间信息。`df`的`-h`选项会以易读的格式打印磁盘空间信息。例如：

```
$ df -h
Filesystem                Size      Used Avail Use% Mounted on
/dev/sda1                  9.2G      2.2G   6.6G  25% /
none                      497M      240K   497M    1% /dev
none                      502M      168K   501M    1% /dev/shm
none                      502M       88K   501M    1% /var/run
none                      502M        0   502M    0% /var/lock
none                      502M        0   502M    0% /lib/init/rw
none                      9.2G      2.2G   6.6G  25% /var/lib/ureadahead/debugfs
```

## 8.3 计算命令执行时间

当测试一个应用程序的效率或比较不同的算法时，其执行时间非常重要。好算法的执行时间应该很短。让我们看看如何来计算执行时间。

### 8.3.1 实战演练

(1) 要测试执行时间，只需要将`time`放在想要运行的命令之前。例如：

```
$ time COMMAND
```

命令COMMAND会执行并生成输出。除此之外，time命令会将命令的执行时间添加到stderr中。例如：

```
$ time ls
test.txt
next.txt
real    0m0.008s
user    0m0.001s
sys     0m0.003s
```

输出中分别显示了执行该命令所花费的real时间、user时间以及sys时间。



time命令的可执行二进制文件位于/usr/bin/time，还有一个shell内建命令也叫做time。当运行time时，默认调用的是shell的内建命令。shell内建的time命令选项有限。因此，如果我们需要使用另外的功能，就应该使用可执行文件time的绝对路径（/usr/bin/time）。

- (2) 可以用选项-o filename将相关的时间统计信息写入文件：

```
$ /usr/bin/time -o output.txt COMMAND
```

文件名应该出现在选项-o之后。

要将命令执行时间添加到文件而不影响其原有内容，使用选项-a以及-o：

```
$ /usr/bin/time -a -o output.txt COMMAND
```

- (3) 我们也可以使用选项-f，利用格式字符串来格式化时间输出。格式字符串由对应于特定选项的参数组成，这些参数以 % 作为前缀。real时间、user 时间、sys 时间的格式字符串分别如下：

```
❑ real: %e
❑ user: %U
❑ sys: %S
```

通过结合参数字符串，我们就可以创建格式化输出：

```
$ /usr/bin/time -f "FORMAT STRING" COMMAND
```

例如：

```
$ /usr/bin/time -f "Time: %U" -a -o timing.log uname
Linux
```

其中，%U是user时间的参数。

格式化输出生成后被写入标准输出，命令的执行时间信息被写入标准错误。我们可以用重定向操作符(>)对格式化输出重定向，用错误重定向操作符(2>)对时间信息重定向。

例如：

```
$ /usr/bin/time -f "Time: %U" uname> command_output.txt 2>time.log
$ cat time.log
Time: 0.00
$ cat command_output.txt
Linux
```

(4) 使用参数%Z显示系统页面大小：

```
$ /usr/bin/time -f "Page size: %Z bytes" ls> /dev/null
Page size: 4096 bytes
```

这里并不需要被计时命令的输出（即ls），因此为了使其不显示在终端中，我们将标准输出重定向到了/dev/null设备。

还有很多可用格式字符串，请阅读man time以获取更多的细节信息。

### 8.3.2 工作原理

有以下三种不同类型的时间。

- ❑ **Real**时间指的是挂钟时间（wall clock time），也就是命令从开始执行到结束的时间。这段时间包括其他进程所占用的时间片（time slice）以及进程被阻塞时所花费的时间（例如，为等待I/O操作完成所用的时间）。
- ❑ **User**时间是指进程花费在用户模式（内核之外）中的CPU时间。这是唯一真正用于执行进程所花费的时间。执行其他进程以及花费在阻塞状态中的时间并没有计算在内。
- ❑ **Sys**时间是指进程花费在内核中的CPU时间。它代表在内核中执行系统调用所使用的时间，这和库代码（library code）不同，后者仍旧运行在用户空间。与“user时间”类似，这也是真正由进程使用的CPU时间。

通过time命令可以获得进程的很多细节信息，包括退出状态、接收到的信号数量、进程上下文的切换次数等。每一个参数都可以用适合的格式字符串显示。

表8-1展示了一些可以使用的参数。

表 8-1

参 数	描 述
%C	进行计时的命令名称以及命令行参数
%D	进程非共享数据区域的大小，以KB为单位
%E	进程使用的real时间（挂钟时间），显示格式为[小时:]分钟:秒
%x	命令的退出状态
%k	进程接收到的信号数量
%W	进程被交换出主存的次数
%Z	系统的页面大小。这是一个系统常量，但在不同的系统中，这个常量值也不同
%P	进程所获得的CPU时间百分比。这个值等于user+system时间除以总运行时间。结果以百分比形式显示
%K	进程的平均总（data+stack+text）内存使用量，以KB为单位
%w	进程主动进行上下文切换的次数，例如等待I/O操作完成
%c	进程被迫进行上下文切换的次数（由于时间片到期）

## 8.4 收集与当前登录用户、启动日志及启动故障的相关信息

收集与操作环境、当前登录用户、主机加电时间、启动故障等相关的信息很有用处。在这则攻略会讲解一些用于收集活动主机信息的相关命令。

### 8.4.1 预备知识

接下来将介绍以下命令：who、w、users、uptime、last和lastb。

### 8.4.2 实战演练

(1) 获取当前登录用户的相关信息：

```
$ who
slynux pts/0 2010-09-29 05:24 (slynuxs-macbook-pro.local)
slynux tty7 2010-09-29 07:08 (:0)
```

该命令会显示出登录名、用户所使用的TTY、登录时间以及登录用户的远程主机名（或者X显示信息）。

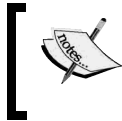


TTY（该术语取自TeleTYperwriter）是与文本终端相关联的设备文件。当用户生成一个新终端时，对应的设备文件就会出现在/dev/中（例如/dev/pts/3）。可以通过输入并执行命令tty来获得当前终端的设备路径。

(2) 获得有关登录用户更详细的信息：

```
$ w
07:09:05 up 1:45, 2 users, load average: 0.12, 0.06, 0.02
USER      TTY      FROM    LOGIN@   IDLE   JCPU   PCPU   WHAT
slynux    pts/0    slynuxs 05:24    0.00s   0.65s  0.11s  sshd: slynux
slynux    tty7     :0       07:08    1:45m   3.28s  0.26s  gnome-session
```

第一行列出了当前时间，系统运行时间，当前登录的用户数量以及过去的1分钟、5分钟、15分钟内的系统平均负载。接下来的每一行显示了每一个登录用户的详细信息，其中包括登录名、TTY、远程主机、登录时间、空闲时间、自该用户登录后所使用的总CPU时间、当前运行进程所使用的CPU时间以及进程所对应的命令行。



uptime命令输出中的平均负载（load average）是表明系统负载量的一个参数。在第9章我们会对此进行详细地解释。

(3) 列出当前登录主机的用户列表：

```
$ users
slynux slynux slynux hacker
```

如果某个用户打开了多个伪终端，那么该用户会被多次显示。在上面的输出中，用户slynux打开了3个伪终端。排除重复用户出现的最简单的方法是使用sort和uniq进行过滤：

```
$ users | tr ' ' '\n' | sort | uniq
slynux
hacker
```

利用tr将' '替换成'\n'，然后用sort和uniq为每个用户生成唯一的输出。

(4) 查看系统已经加电运行了多长时间：

```
$ uptime
21:44:33 up 3:17, 8 users, load average: 0.09, 0.14, 0.09
```

单词up之后的时间表明了系统已经加电运行了多久。我们可以编写一个简单的单行脚本来提取运行时间：

```
$ uptime | grep -Po '\d{2}\:\d{2}\:\d{2}'
```

这条命令利用grep和perl风格的正则表达式来提取由冒号分隔的3组两位数字。

(5) 获取上一次启动以及用户登录会话的信息：

```
$ last
slynux    tty7           :0                Tue Sep 28 18:27    still logged in
```

```
reboot    system boot  2.6.32-21-generic Tue Sep 28 18:10 - 21:46 (03:35)
slynux    pts/0        :0.0              Tue Sep 28 05:31 - crash (12:39)
```

last命令可以提供登录会话信息。它实际上是一个系统登录日志，包括了登录tty、登录时间、状态等信息。

last命令以日志文件/var/log/wtmp作为输入日志数据。它也可以用选项-f明确地指定日志文件。例如：

```
$ last -f /var/log/wtmp
```

(6) 获取单个用户登录会话的信息：

```
$ last USER
```

(7) 获取重启会话（reboot session）信息：

```
$ last reboot
reboot    system boot  2.6.32-21-generi Tue Sep 28 18:10 - 21:48 (03:37)
reboot    system boot  2.6.32-21-generi Tue Sep 28 05:14 - 21:48 (16:33)
```

(8) 获取失败的用户登录会话信息：

```
# lastb
test      tty8          :0              Wed Dec 15 03:56 - 03:56 (00:00)
slynux    tty8          :0              Wed Dec 15 03:55 - 03:55 (00:00)
```

你必须以root用户的身份运行lastb。

## 8.5 列出 1 小时内占用 CPU 最多的 10 个进程

CPU是一种重要的资源，最好能够跟踪某个阶段内占用CPU最多的进程。通过监视一段时间内CPU的使用情况，我们可以找出长期占用CPU的进程并对其进行优化，提高CPU使用效率。我们在这则攻略中讨论了进程监视与日志记录。

### 8.5.1 预备知识

ps命令用于收集系统中进程的详细信息。这些信息包括CPU使用情况、正在执行的命令、内存占用、进程状态等。记录在一小时内的CPU占用情况，然后灵活地运用ps以及文本处理就可以找出占用CPU最多的10个进程。关于ps命令的更多细节，请参考第9章。

### 8.5.2 实战演练

让我们看看用于监视并计算一小时内CPU使用情况的shell脚本：

```
#!/bin/bash
#文件名: pcpu_usage.sh
#用途: 计算1个小时内进程的CPU占用情况

SECS=3600
UNIT_TIME=60

#将SECS更改成需要进行监视的总秒数
#UNIT_TIME是取样的时间间隔, 单位是秒

STEPS=$(( $SECS / $UNIT_TIME ))

echo Watching CPU usage... ;

for((i=0;i<STEPS;i++))
do
    ps -eocomm,pcpu | tail -n +2 >> /tmp/cpu_usage.$$
    sleep $UNIT_TIME
done

echo
echo CPU eaters :

cat /tmp/cpu_usage.$$ | \
awk '
{ process[$1]+=$2; }
END{
    for(i in process)
    {
        printf("%-20s %s\n",i, process[i]) ;
    }

    }' | sort -nrk 2 | head

rm /tmp/cpu_usage.$$
#删除临时日志文件
```

输出如下:

```
$ ./pcpu_usage.sh
Watching CPU usage...
CPU eaters :
Xorg          20
firefox-bin   15
bash          3
evince        2
pulseaudio    1.0
pcpu.sh       0.3
wpa_suplicant 0
```

```
wnck-applet      0
watchdog/0       0
usb-storage      0
```

### 8.5.3 工作原理

在上面的脚本中,主要的输入源是`ps -eo comm,pcpu`,其中`comm`表示命令名( `command name` ),`pcpu`表示CPU使用率( `CUP usage in percent` )。该命令输出所有进程名及CPU使用率。每个进程对应一行输出。因为需要监视一小时内CPU的使用情况,所以我们得在一个每次迭代时间为60秒的`for`循环中不停地用`ps -eo comm,pcpu | tail -n +2`来获取CPU的使用统计数据,并将这些数据添加到文件 `/tmp/cpu_usage.$$` 中。60秒的迭代时间通过`sleep 60`来提供。这就使得每分钟执行一次`ps. tail -n +2`用来将 `ps` 输出中的头部和 `COMMAND %CPU` 剥除。

`cpu_usage.$$` 中的 `$$` 表示当前脚本的进程 ID。假设进程 ID为1345,那么在脚本执行时它会被替换成 `/tmp/cpu_usage.1345`。因为这是一个临时文件,所以我们把它放在 `/tmp`中。

统计文件在1小时后就准备妥当了,文件中包含了60项,分别对应每分钟的进程状态。然后用`awk`求出每个进程总的CPU使用情况。我们用了一个关联数组来统计CPU使用情况。其中进程名作为数组索引。最后根据总的CPU使用情况依数值逆序排序,并通过`head`获得前10项。

### 8.5.4 参考

- ❑ 4.6节讲解了`awk`命令。
- ❑ 3.13节讲解了`tail`命令。

## 8.6 使用 watch 监视命令输出

我们可能需要在一段时期内以固定的间隔时间不断监视某个命令的输出。例如在复制大文件时,我们希望看到不断增长的文件大小。要做到这一点,我们可以使用`watch`命令来执行`du`命令,不断地重复输出。这则攻略会告诉你实现方法。

### 8.6.1 实战演练

`watch`命令可以用来在终端中以固定的间隔监视命令输出。该命令语法如下:

```
$ watch COMMAND
```

例如:

```
$ watch ls
```

或者

```
$ watch 'COMMANDS'
```

例如：

```
$ watch 'ls -l | grep "^d"'  
# 只列出目录
```

命令默认每2秒更新一次输出。

我们可以用 `-n SECOND`指定更新输出的时间间隔。例如：

```
$ watch -n 5 'ls -l'  
#以5秒为间隔，监视ls -l的输出
```

## 8.6.2 补充内容

让我们再来看一下`watch`命令的其他特性。

**突出标示`watch`输出中的差异**

`watch`有一个选项可以将时间间隔前后的命令输出的差异之处以不同颜色突出标示出来。选项 `-d`可以启用这一功能：

```
$ watch -d 'COMMANDS'
```

## 8.7 记录文件及目录访问

记录文件及目录访问对于跟踪文件和目录的变化很有帮助。这则攻略会讲解如何记录这种类型的访问。

### 8.7.1 预备知识

`inotifywait`命令可以用来收集有关文件访问的信息。Linux发布版默认并没有包含这个命令，你得用软件包管理器自行安装`inotify-tools`。这个命令还需要将`inotify`支持编译入Linux内核，好在大多数新的GNU/Linux发布版已经都在内核中启用了`inotify`。

### 8.7.2 实战演练

来看看用来监视目录访问的shell脚本：

```
#!/bin/bash  
#文件名：watchdir.sh
```

```
#用途:监视目录访问
path=$1
#将目录或文件路径作为脚本参数
```

```
inotifywait -m -r -e create,move,delete $path -q
```

输出样例如下:

```
$ ./watchdir.sh .
./ CREATE new
./ MOVED_FROM new
./ MOVED_TO news
./ DELETE news
```

8.7.3 工作原理

上面的脚本记录给定了路径中文件或目录的创建、移动以及删除。选项 -m表明要持续监视变化，而不是在事件发生之后退出。-r允许采用递归形式监视目录（忽略符号链接）。-e 指定需要监视的事件列表。-q 用于减少冗余信息，只打印出所需要的信息。命令输出可以被重定向到日志文件。

我们可以从事件列表中添加或删除事件。一些重要的事件如表8-2所示。

表 8-2

事 件	描 述
访问 (access)	读取文件
修改 (modify)	文件内容被修改
属性 (attrib)	文件元数据被修改
移动 (move)	移动文件操作
创建 (create)	生成新文件
打开 (open)	打开文件操作
关闭 (close)	关闭文件操作
删除 (delete)	文件被删除

8.8 用 logrotate 管理日志文件

日志文件是Linux系统维护中必不可少的组成部分，它可以帮助跟踪系统内各种服务中出现的事件。这既有助于排查问题，也可以为活动主机提供统计数据。随着时间的推移，日志文件会变得越来越大，因而对于日志文件的管理就必不可少。我们将用一种被称为轮替（rotation）的技术来限制日志文件的体积，一旦它超过了限定的大小，就对其内容进行抽取（strip），同时将

日志文件中的旧条目存储到日志目录中的归档文件内。旧的日志文件就会得以保存以便随后参阅。让我们看看如何对日志文件进行轮替及存储。

### 8.8.1 预备知识

`logrotate`是每一位Linux系统管理员都应该了解的命令。它能够将日志文件的大小限制在给定的SIZE内。日志记录程序会将信息添加到日志文件中。最新添加的信息总是出现在日志文件的尾部。`logrotate`根据配置文件扫描特定的日志文件。它在保留日志文件中最新添加的100KB内容（假设指定SIZE = 100k）的同时，将剩下的数据（较旧的日志数据）移入新文件`logfile_name.1`。随着该日志文件（`logfile_name.1`）中的内容越来越多，逐渐超出了SIZE规定的定额，`logrotate`就会再用最近的内容更新日志文件，然后用较旧的内容创建`logfile_name.2`。整个过程可以轻松地使用`logrotate`进行配置。`logrotate`还可以将旧的日志文件压缩成`logfile_name.1.gz`、`logfile_name.2.gz`等。是否选择压缩旧日志文件也可以通过`logrotate`来配置。

### 8.8.2 实战演练

`logrotate`的配置目录位于`/etc/logrotate.d`。如果列出这个目录中的内容，你会发现很多其他的日志文件配置。

我们可以为自己的日志文件（比如`/var/log/program.log`）编写一个特定的配置：

```
$ cat /etc/logrotate.d/program
/var/log/program.log {
missingok
notifempty
size 30k
    compress
weekly
    rotate 5
create 0600 root root
}
```

这就是全部的内容。其中，`/var/log/program.log`指定了日志文件路径。旧的日志文件归档之后也放入同一个目录中。

### 8.8.3 工作原理

表8.3列出了配置文件中各个参数的含义。

表 8-3

参 数	描 述
missingok	如果日志文件丢失，则忽略；然后返回（不对日志文件进行轮替）
notifempty	仅当源日志文件非空时才对其进行轮替
size 30k	限制实施轮替的日志文件的大小。可以用1M表示1MB
compress	允许用gzip压缩较旧的日志
weekly	指定进行轮替的时间间隔。可以是weekly、yearly或daily
rotate 5	这是需要保留的旧日志文件的归档数量。在这里指定的是5，所以这些文件名将会是program.log.1.gz、program.log.2.gz等直到program.log.5.gz
create 0600 root root	指定所要创建的归档文件的模式、用户以及用户组

表8-3中的选项都是可选的。我们只在logrotate配置文件中指定所需的选项即可。如果想了解更多的可用选项，请参考logrotate的手册页（<http://linux.die.net/man/8/logrotate>）。

8.9 用 **syslog** 记录日志

通常与各种守护进程和应用程序相关的日志文件都会放在/var/log目录中, 因为这是存储日志文件的公共目录。如果你读过日志文件，就会看出它们都采用了一种通用的格式。在Linux系统中，由守护进程syslogd使用syslog协议负责在 /var/log中创建并写入日志信息。每一个标准应用进程都可以利用syslog记录日志信息。在这则攻略中，我们将讨论如何在脚本中用syslogd进行日志信息的记录。

8.9.1 预备知识

日志文件有助于我们推断系统出现了什么故障。在编写重要的应用程序时，应当将应用程序的执行过程记录在日志文件中。我们接下来将学习使用命令logger通过syslogd记录日志。在学习如何往日志文件中写入信息之前，先看看Linux中一些重要的日志文件，如表8-4所示。

表 8-4

日志文件	描 述
/var/log/boot.log	系统启动信息
/var/log/httpd	Apache Web服务器日志
/var/log/messages	发布内核启动信息
/var/log/auth.log	用户认证日志
/var/log/dmesg	系统启动信息
/var/log/mail.log	邮件服务器日志
/var/log/Xorg.0.log	X服务器日志

### 8.9.2 实战演练

下面来看看如何使用logger创建及管理日志信息。

- (1) 向系统日志文件/var/log/message中写入日志信息：

```
$ logger LOG_MESSAGE
```

例如：

```
$ logger This is a test log line
```

```
$ tail -n 1 /var/log/messages
```

```
Sep 29 07:47:44 slynux-laptop slynux: This is a test log line
```

/var/log/messages是一个通用日志文件。如果使用logger命令，它默认将日志信息记录到 /var/log/messages中。

- (2) 如果要记录特定的标记（tag），可以使用：

```
$ logger -t TAG This is a message
```

```
$ tail -n 1 /var/log/messages
```

```
Sep 29 07:48:42 slynux-laptop TAG: This is a message
```

Syslog处理/var/log下的多个日志文件。但是当logger发送消息时，它用标记字符串来确定应该记录到哪一个日志文件中。syslogd使用与日志相关联的TAG来决定应该将其记录到哪一个文件中。你可以从/etc/rsyslog.d/目录下的配置文件中看到标记字符串以及与其相关联的日志文件。

- (3) 要将另一个日志文件的最后一行记录到系统日志中，可以使用：

```
$ logger -f /var/log/source.log
```

### 8.9.3 参考

3.13节讲解了head和tail命令。

## 8.10 通过监视用户登录找出入侵者

日志文件可以用于收集系统状态的详细有关信息。下面是一个脚本相关的问题描述。

我们有一个通过SSH连接到Internet的系统。很多攻击者试图登入这个系统，因此需要利用shell脚本来设计一个入侵检测系统。入侵者定义为：屡次试图登入系统达两分钟以上，并且期间的登录过程全部失败。凡是这类用户都应该被检测出来并生成包含以下细节信息的报告：

- ❑ 试图登录的账户;
- ❑ 试图登录的次数;
- ❑ 攻击者的IP地址;
- ❑ IP地址所对应的主机;
- ❑ 进行登录的时间段。

### 8.10.1 预备知识

我们可以编写一个shell脚本，对日志文件进行扫描并从中收集所需要的信息。为了处理SSH登录失败的情况，还知道用户认证会话日志会被记录在日志文件/var/log/auth.log中。脚本需要扫描这个日志文件来检测出失败的登录信息，执行各种检查来获取所需要的数据。我们可以用host命令找出IP地址所对应的主机。

### 8.10.2 实战演练

接下来编写一个入侵检测脚本，它利用用户认证日志文件来生成有关入侵者的报告：

```
#!/bin/bash
#文件名:intruder_detect.sh
#用途:入侵报告工具，以auth.log作为日志文件
AUTHLOG=/var/log/auth.log

if [[ -n $1 ]];
then
    AUTHLOG=$1
    echo Using Log file : $AUTHLOG
fi

LOG=/tmp/valid.$$log
grep -v "invalid" $AUTHLOG > $LOG
users=$(grep "Failed password" $LOG | awk '{ print $(NF-5) }' | sort | uniq)

printf "%-5s|%-10s|%-10s|%-13s|%-33s|%-33s\n" "Sr#" "User" "Attempts" "IP address"
"Host_Mapping" "Time range"

ucount=0;

ip_list="$(egrep -o "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" $LOG | sort | uniq)"

for ip in $ip_list;
do
    grep $ip $LOG > /tmp/temp.$$log
```

```

for user in $users;
do
    grep $user /tmp/temp.$$log> /tmp/$$.log
    cut -c-16 /tmp/$$.log > $$$.time
    tstart=$(head -1 $$$.time);
    start=$(date -d "$tstart" "+%s");
    tend=$(tail -1 $$$.time);
    end=$(date -d "$tend" "+%s")

    limit=$(( $end - $start ))

    if [ $limit -gt 120 ];
    then
        let ucount++;

        IP=$(egrep -o "[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+" /tmp/$$.log | head -1 );

        TIME_RANGE="$tstart-->$tend"

        ATTEMPTS=$(cat /tmp/$$.log|wc -l);

        HOST=$(host $IP | awk '{ print $NF }' )

        printf "%-5s|%-10s|%-10s|%-10s|%-33s|%-s\n" "$ucount" "$user" "$ATTEMPTS" "$IP"
"$HOST" "$TIME_RANGE";
    fi
done
done

rm/tmp/valid.$$log/tmp/$$.log $$$.time/tmp/temp.$$log 2>/dev/null

```

输出样例如下：

```

slynux@slynux-laptop:~$ ./intruder_detect.sh sampleauth.log
Using Log file : sampleauth.log

```

Sr#	User	Attempts	IP address	Host Mapping	Time range
1	alice	3	203.110.250.34	attk1.foo.com	Oct 29 05:28:59 -->Oct 29 05:31:59
2	bob1	3	203.110.251.31	attk2.foo.com	Oct 29 05:21:52 -->Oct 29 05:29:52
3	bob2	3	203.110.250.34	attk1.foo.com	Oct 29 05:22:59 -->Oct 29 05:25:52
4	gvraju	20	203.110.251.31	attk2.foo.com	Oct 28 04:37:10 -->Oct 29 05:19:09
5	root	21	203.110.253.32	attk3.foo.com	Oct 29 05:18:01 -->Oct 29 05:37:01

### 8.10.3 工作原理

在intruder\_detect.sh脚本中，我们将auth.log文件作为输入。也可以用脚本的命令行参数来提

供一个日志文件作为输入，或者默认读取/var/log/auth.log。我们只需要记录合法用户的登录日志详情。如果有非法用户企图登录，则类似于Failed password for invalid user bob from 203.83.248.32 port 7016 ssh2的日志就会出现在auth.log中。因此我们需要排除日志文件中所有包含invalid的行。grep命令的反转选项（-v）可以用来移除对应非法用户的所有日志内容。下一步是找出试图登录并且失败的用户列表。对于密码错误，SSH会记录类似的日志信息：sshd[21197]: Failed password for bob1 from 203.83.248.32 port 50035 ssh2，所以我们应找出所有包含“failed password”的行。

接下来，要找出所有不重复的IP地址以提取对应于每一个IP地址的日志行。提取IP地址列表可以用匹配IP地址的正则表达式和egrep命令来完成。用for循环对IP地址进行迭代，并用grep找出对应的日志行并将其写入临时文件。日志行中倒数第6个单词是用户名（例如bob1），可用awk命令提取用户名（倒数第6个单词）。NF返回最后一个单词的列号，因此NF-5就是倒数第6个单词的列号。我们再用sort和uniq生成一个没有重复的用户列表。

现在我们要收集表明登录失败的日志行，这些行中包含了用户名。for循环用来读取对应每一位用户的日志行，并将这些行写入临时文件。每一行的前16个字符是时间戳，可用cut命令进行提取。一旦得到了一个用户所有登录失败的时间戳，就要检查第一次和最后一次试图登录之间的时间差。第一行日志对应第一次登录，最后一行日志对应最后一次登录。我们用head -1提取首行，用tail -1提取末行。这样就有了首次登录（tstart）和末次登录（tends）的字符串格式的时间戳。使用date命令，我们可以将字符串形式的日期转换成Unix纪元时的总秒数（1.10节讲解了Unix纪元时）。

变量start和end中包含着秒数，分别对应于字符串形式的起始时间戳。对这两个时间求差，并检查差值是否大于2分钟（即120秒）。如果某个用户被确认为入侵者，其对应的细节信息就要生成日志。IP地址可以用正则表达式和egrep命令从日志中提取。试图登录的次数就是含有某个用户的日志行数。这个行数可以用wc命令获得。IP地址到主机名的映射可以将IP地址作为host命令的参数，然后从命令输出中提取。时间范围可以用已经提取到的时间戳来打印。最后，删除脚本使用过的临时文件。

上节的脚本旨在演示一个用于扫描日志并从中生成报告的模型。我们尽力让脚本更短小简单，以避免过于复杂，因此难免存在一些bug。你可以使用更好的处理逻辑来改进该脚本。

## 8.11 监视远程磁盘的健康情况

网络是由不同用户的多台主机组成。它需要对远程主机的磁盘使用情况实施集中监视。网络的系统管理员得每天记录网络中所有主机的磁盘使用。日志的每一行应该包含日期、主机IP地址、设备、设备容量、占用空间、剩余空间、使用比例、健康状况等细节信息。如果远程主机中的任

意分区使用率超过了80%，那么健康状态应该被设置为ALERT，否则就可以设置为SAFE。这则攻略将演示如何编写一个可以收集网络远程主机详细信息的监视脚本。

### 8.11.1 预备知识

我们需要从网络中收集每台主机的磁盘使用情况信息，然后写入中央主机的日志文件中。可以将负责收集信息并写入日志的脚本调度为每天的特定时间执行。可以使用SSH来登录远程系统收集磁盘使用情况。

### 8.11.2 实战演练

首先得在网络中的所有远程主机上设置一个共用账户。这个账户供脚本disklog登录系统使用。我们需要为这个账户配置SSH自动登录（7.8节讲解了自动登录的配置方法）。假设在所有配置了自动登录的远程主机上都有一个叫做test的用户，那么来看看这个shell脚本：

```
#!/bin/bash
#文件名: disklog.sh
#用途: 监视远程系统的磁盘使用情况

logfile="diskusage.log"

if [[ -n $1 ]]
then
    logfile=$1
fi

if [ ! -e $logfile ]
then

    printf "%-8s %-14s %-9s %-8s %-6s %-6s %-6s %s\n" "Date" "IP address" "Device"
"Capacity" "Used" "Free" "Percent" "Status" > $logfile
fi

IP_LIST="127.0.0.1 0.0.0.0"
#提供远程主机IP地址列表

(
for ip in $IP_LIST;
do
    #slynux是用户名，可以按照实际情况进行修改
    ssh slynux@$ip 'df -H' | grep ^/dev/ > /tmp/$$.df

    while read line;
```

```

do
    cur_date=$(date +%D)
    printf "%-8s %-14s " $cur_date $ip
    echo $line | awk '{ printf("%-9s %-8s %-6s %-6s %-8s", $1, $2, $3, $4, $5); }'

    pusg=$(echo $line | egrep -o "[0-9]+%")
    pusg=${pusg/%/};
    if [ $pusg -lt 80 ];
    then
        echo SAFE
    else
        echo ALERT
    fi

    done< /tmp/$$.df
done

) >> $logfile

```

我们可以用cron以固定的间隔来调度脚本执行，例如在crontab中写入以下条目，就可以在每天上午10点运行该脚本：

```
00 10 * * * /home/path/disklog.sh /home/user/diskusg.log
```

执行命令crontab -e，然后添加上面一行。也可以手动执行脚本：

```
$ ./disklog.sh
```

脚本的输出样例如下：

```

slynux@slynux-laptop:~/book$ cat diskusage.log
Date      IP address      Device    Capacity Used   Free   Percent Status
12/15/10  127.0.0.1       /dev/sda1 9.9G      2.4G   7.0G   26%     SAFE
12/15/10  0.0.0.0        /dev/sda1 9.9G      2.4G   7.0G   26%     SAFE

```

### 8.11.3 工作原理

在脚本disklog.sh中，我们可以提供日志文件路径作为命令行参数，否则脚本使用默认的日志文件。如果日志文件不存在，它会将日志文件头部写入新文件中。-e logfile用来检查文件是否存在。远程主机的IP地址列表被存储在变量IP\_LIST中，彼此之间以空格分隔。要确保在IP\_LIST中列出的所有远程系统中都有用户test，并且SSH已经配置了自动登录。for循环用来对IP地址进行逐个迭代。通过ssh使用远程命令df -H获取磁盘剩余空间。这项数据被存储在一个临时文件中。while循环用来逐行读取这个文件。利用awk提取并打印数据，同时一并打印的还有日期。用egrep提取使用率，并将%删除以获取使用率的数值部分。检查得到的数值，看是否超过了80。如果不足80，将状态设置为SAFE；如果大于或等于80，则将状态设置为ALERT。打

印出来的所有数据要被重定向到日志文件中。因此代码被放入子`shell()`中，并将标准输出重定向到日志文件。

### 8.11.4 参考

9.7节讲解了`crontab`命令。

## 8.12 找出系统中用户的活跃时段

考虑一个使用共享主机的Web服务器。每天都会有很多用户登录和注销，用户活动都被记入了服务器的系统日志。这则攻略是一项实践任务：利用系统日志找出每个用户在服务器上停留了多久，并根据其时间长短来进行分级，最后生成一份包含等级、用户名、首次登录时间、末次登录时间、登录次数以及总使用时长等细节信息的报告。让我们看看如何解决这个问题。

### 8.12.1 预备知识

`last`命令用来列出系统中有关用户登录会话的细节。这些会话数据被存储在`/var/log/wtmp`文件中。通过分别累计各用户的会话时间，就能得出他们的总使用时间。

### 8.12.2 实战演练

研究一下这个用来找出活跃用户并生成报告的脚本：

```
#!/bin/bash
#用户名: active_users.sh
#用途:查找活跃用户

log=/var/log/wtmp

if [[ -n $1 ]];
then
    log=$1
fi

printf "%-4s %-10s %-10s %-6s %-8s\n" "Rank" "User" "Start" "Logins" "Usage hours"

last -f $log | head -n -2 > /tmp/ulog.$$

cat /tmp/ulog.$$ | cut -d' ' -f1 | sort | uniq> /tmp/users.$$

(
while read user;
do
```

```

grep ^$user /tmp/ulog.$$ > /tmp/user.$$
minutes=0

while read t
do
    s=$(echo $t | awk -F: '{ print ($1 * 60) + $2 }')
    let minutes=minutes+s
done< <(cat /tmp/user.$$ | awk '{ print $NF }' | tr -d '()')

firstlog=$(tail -n 1 /tmp/user.$$ | awk '{ print $5,$6 }')
nlogins=$(cat /tmp/user.$$ | wc -l)
hours=$(echo "$minutes / 60.0" | bc)

printf "%-10s %-10s %-6s %-8s\n" $user "$firstlog" $nlogins $hours
done< /tmp/users.$$

) | sort -nrk 4 | awk '{ printf("%-4s %s\n", NR, $0) }'
rm /tmp/users.$$ /tmp/user.$$ /tmp/ulog.$$

```

输出样例如下：

```

$ ./active_users.sh

```

Rank	User	Start	Logins	Usage	hours
1	easyibaa	Dec 11	531	349	
2	demoproj	Dec 10	350	230	
3	kjayaram	Dec 9	213	55	
4	cinenews	Dec 11	85	139	
5	thebenga	Dec 10	54	35	
6	gateway2	Dec 11	52	34	
7	softl132	Dec 12	49	25	
8	sarathla	Nov 1	45	29	
9	gtsminis	Dec 11	41	26	
10	agentcde	Dec 13	39	32	

### 8.12.3 工作原理

在脚本active\_users.sh中，需要提供日志文件作为命令行参数，否则就使用默认的日志文件wtmp。命令last -f用来打印日志文件的内容。日志文件的第一列是用户名。我们用cut从中提取第一列，然后用sort和uniq找出不重复的用户。对于每一位用户，用grep找出其对应登录会话的日志行并写入一个临时文件。日志的最后一列是用户登录会话的时长。为了找出用户总的使用时间，需要累加所有的会话时长。使用时间的格式是（小时：秒），需要用一个简单的awk脚本将其转换成分钟。

要提取用户的会话时长，得使用awk命令。要移除括号，得使用tr -d。用<( COMMANDS )操作符将使用时长字符串列表作为标准输入传递给while循环，其作用就相当于文件输入。利用date命令将每一个时长字符串转换成秒数，并累加到变量seconds中。把出现在最后一行的用

户的首次登录时间提取出来。登录次数就是日志的行数。要根据总的使用时间来计算每位用户的等级,数据记录以使用时长为键,进行降序排列。用sort命令的-nr选项指定按照数值逆序排列。-k4指定键的列号(即使用时长)。最后,sort的输出被传递给awk。awk命令为每一行添加上行号,这个行号就是每一位用户的等级。

## 8.13 电源使用的测量与优化

必须时刻关注电源的消耗,尤其对于笔记本电脑、平板电脑等移动设备而言更是如此。Linux系统中有几个工具可以测量电源的使用情况,powertop就是其中之一,我们将在本攻略中学习它的用法。

### 8.13.1 预备知识

powertop在多数Linux发布版中都没有被预装,你得使用软件包管理器自行安装。

### 8.13.2 实战演练

来看看如何使用powertop来对电源使用进行测量及优化。

(1) powertop的用法相当简单,只需要运行以下命令即可:

```
# powertop
```

随后powertop就会开始进行测量,测量结束之后会显示出有关电源使用情况、耗电最多的进程等详细信息,如图8-1所示:

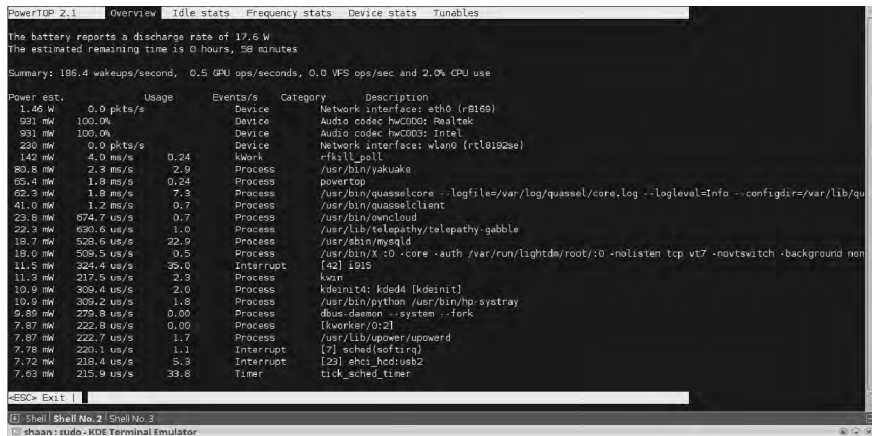


图 8-1

(2) 生成HTML格式的报表:

```
# powertop --html
```

powertop会进行一段时间的测量, 然后生成一份默认名称为PowerTOP.html的HTML报表, 你可以使用Web浏览器进行浏览。

(3) 优化电源使用:

在powertop运行时, 可以使用箭头键切换到Tunables标签。该标签下包含了一系列可由powertop调节的选项, 借以降低电源消耗。只需要选中希望调节的选项, 按回车键将选项从Bad切换到Good, 如图8-2所示。

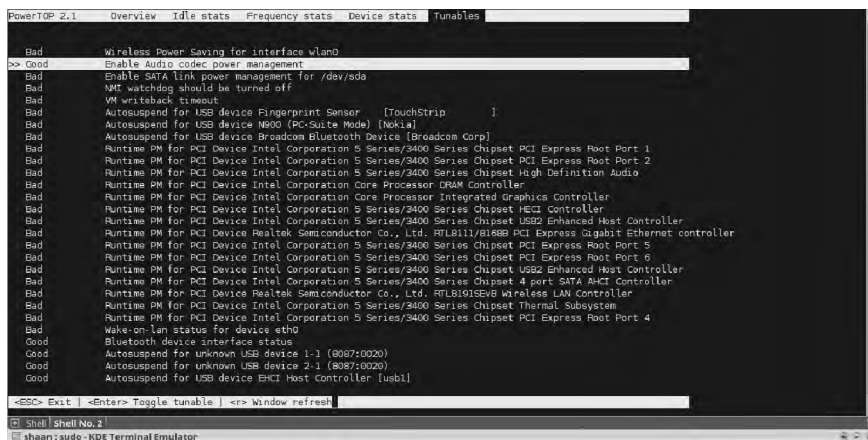


图 8-2

如果想要监视可移动设备的电池消耗情况, 需要拔掉设备的充电器, 使用powertop对电池进行测量。

## 8.14 监视磁盘活动

监视类工具的名称流行以top (进程监视命令) 作为结尾, 依照这种命名习惯, 监视磁盘I/O的工具就叫做iotop。

### 8.14.1 预备知识

iotop在多数Linux发布版中都没有被预装, 你得使用软件包管理器自行安装。

### 8.14.2 实战演练

使用*iotop*进行I/O监视的方法多种多样，在这则攻略中会涉及其中的几种。

(1) 交互式监视：

```
# iotop -o
```

*iotop*的-o选项只显示出那些正在进行I/O活动的进程。该选项有助于减少输出干扰。

(2) 用于shell脚本的非交互式用法：

```
# iotop -b -n 2
```

这使得*iotop*打印出两次统计数据，然后退出。如果你希望在shell脚本中获得输出结果并对其进行处理，这种用法就能派上用场了。

(3) 监视特定进程

```
# iotop -p PID
```

使用你希望对其进行监视的进程的PID，*iotop*就只会输出该进程的统计信息。



在大多数现代Linux发布版中，无需先查找PID，然后再提供给*iotop*。你可以使用*pidof*命令，将之前的命令写成如下形式：

```
# iotop -p 'pidof cp'
```

## 8.15 检查磁盘及文件系统错误

数据是计算机系统最重要的东西。因而对存储在物理介质上数据的一致性进行监视自然也就很重要了。

### 8.15.1 预备知识

我们将会使用标准工具*fsck*来检查文件系统错误。该命令应该已经预装在所有的Linux发布版中了。如果没有预装，请使用软件包管理器自行安装。

### 8.15.2 实战演练

来看看如何使用*fsck*的各种选项对文件系统错误进行检查和修复。

(1) 要检查分区或文件系统的错误，只需要将路径作为*fsck*的参数：

```
# fsck /dev/sdb3
fsck from util-linux 2.20.1
```

```
e2fsck 1.42.5 (29-Jul-2012)
HDD2 has been mounted 26 times without being checked, check forced.
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
HDD2: 75540/16138240 files (0.7% non-contiguous), 48756390/64529088 blocks
```

- (2) 检查/etc/fstab中所配置的所有文件系统：

```
# fsck -A
```

该命令会依次检查/etc/fstab中列出的文件系统。fstab文件对磁盘及其挂载点之间的映射关系进行了配置，以便于更便捷地挂载文件系统。这也使得在系统启动的过程中能够挂载某些文件系统。

- (3) 指定fsck自动修复错误，无需询问是否进行修复：

```
# fsck -a /dev/sda2
```

- (4) 模拟fsck要执行的操作：

```
# fsck -AN
fsck from util-linux 2.20.1
[/sbin/fsck.ext4 (1) -- /] fsck.ext4 /dev/sda8
[/sbin/fsck.ext4 (1) -- /home] fsck.ext4 /dev/sda7
[/sbin/fsck.ext3 (1) -- /media/Data] fsck.ext3 /dev/sda6
```

该命令会打印出fsck将要执行什么样的操作（检查所有的文件系统）。

### 8.15.3 工作原理

fsck不过是一个前端程序而已，对于各类文件系统而言，都有其针对性的fsck程序。当执行fsck时，它会自动检测文件系统类型，然后运行对应的fsck.fstype命令，其中fstype是文件系统的类型。如果我们在ext4文件系统中执行fsck，它最终会调用fsck.ext4命令。

正因为如此，你会发现fsck本身只支持所有这些特定文件系统工具所共有的选项。要查找更详细的选项，请查找特定工具（如fsck.ext4）的手册页。

如果怀疑文件系统受损，需要使用fsck进行修复时，对fsck操作进行模拟则不无益处。因为必须确保fsck不会执行我们不希望进行的操作。比如说，fsck可能会将某些扇区标为坏扇区，但我们却希望从中恢复数据。在这种情况下，我们只让fsck做一个演习，打印出所要进行的操作，而不是真正进行操作。

### 本章内容

- ❑ 收集进程信息
- ❑ 杀死进程以及发送或响应信号
- ❑ 向用户终端发送消息
- ❑ 采集系统信息
- ❑ 使用 `/proc` 采集信息
- ❑ 使用 `cron` 进行调度
- ❑ 从 `Bash` 中读写 `MySQL` 数据库
- ❑ 用户管理脚本
- ❑ 图像文件的缩放及格式转换
- ❑ 在终端下进行截屏
- ❑ 管理多个终端

## 9.1 简介

GNU/Linux 的生态系统是由运行的程序、服务、所连接的设备、文件系统、用户等组成的。系统管理的主要目的在于对整个系统形成一个概观并对操作系统进行整体上的管理。我们应该掌握用于系统信息收集及资源管理的常用命令以及实践用法，这也有助于编写脚本和使用自动化工具。本章将介绍一些此类工具的用法。

## 9.2 收集进程信息

进程是程序的运行实例（`running instance`）。运行在一台计算机中的多个进程都被分配了一个称为进程 ID（`PID`）的唯一标识数字。同一个程序的多个实例可以同时运行，但是它们的 `PID` 却互不相同。进程包括多种属性，例如拥有该进程的用户、进程使用的内存数量、进程占用的 `CPU` 时间等。这则攻略将学习如何收集有关进程的信息。

### 9.2.1 预备知识

和进程管理相关的重要命令是 `top`、`ps` 和 `pgrep`。让我们看看如何收集进程信息。

## 9.2.2 实战演练

`ps`是收集进程信息的重要工具。它提供的信息包括：拥有进程的用户、进程的起始时间、进程对应的命令行路径、PID、进程所属的终端（TTY）、进程使用的内存、进程占用的CPU等。例如：


```
$ ps
  PID TTY          TIME CMD
 1220 pts/0    00:00:00 bash
 1242 pts/0    00:00:00 ps
```

`ps`命令通常结合一系列参数使用。如果不使用任何参数，`ps`将显示运行在当前终端（TTY）中的进程。第一列显示进程ID（PID），第二列是TTY（终端），第三列是进程启动后过去的时间，最后一列是CMD（进程所对应的命令）。

为了包含更多的信息，可以使用`-f`（表示full）来显示多列，如下所示：

```
$ ps -f
  UID          PID  PPID  C  STIME TTY          TIME CMD
 slynux       1220   1219  0  18:18 pts/0    00:00:00 -bash
 slynux       1587   1220  0  18:59 pts/0    00:00:00 ps -f
```

上面的`ps`命令没有什么用处，因为它没有提供除当前终端外的任何进程信息。要获取运行在系统中的每个进程的信息，使用选项`-e`（every）。选项`-ax`（all）也可以生成同样的输出。

 选项`-x`和`-a`用来解除`ps`默认设置的TTY限制。通常使用不带参数的`ps`命令，以便只打印出其所属终端上的进程。

运行如下命令之一：`ps -e`，`ps -ef`，`ps -ax`或`ps -axf`。

```
$ ps -e | head
  PID TTY          TIME CMD
  1 ?           00:00:00 init
  2 ?           00:00:00 kthreadd
  3 ?           00:00:00 migration/0
  4 ?           00:00:00 ksoftirqd/0
  5 ?           00:00:00 watchdog/0
  6 ?           00:00:00 events/0
  7 ?           00:00:00 cpuset
  8 ?           00:00:00 khelper
  9 ?           00:00:00 netns
```

输出列表很长。我们使用`head`进行了过滤，所以只列出了前10项。

`ps`命令支持显示包括进程名及PID在内的多种信息。`ps`默认在不同的列中显示这些信息，

其中的大多数都没什么用处。可以用-o来指定想要显示的列，以便只打印出我们需要的内容。与进程相关的参数可以通过与此参数对应的命令选项指定。参数列表以及-o的用法在接下来会进行讨论。

用ps显示所需要的输出列：

`$ ps [OTHER OPTIONS] -o parameter1,parameter2,parameter3 ..`



-o的参数以逗号操作符(,)作为定界符。值得注意的是，逗号操作符与它分隔的参数之间是没有空格的。在大多数情况下，选项-o都是和选项-e结合使用的(-oe)，因为它需要列出运行在系统中的每一个进程。但是如果-o需要使用某些过滤器，例如列出特定用户拥有的进程，那么就不再使用-e。-e和过滤器结合使用没有任何实际效果，依旧会显示所有的进程。

示例如下，其中comm表示COMMAND，pcpu表示CPU占用率：

```
$ ps -eo comm,pcpu | head
COMMAND      %CPU
init          0.0
kthreadd      0.0
migration/0   0.0
ksoftirqd/0   0.0
watchdog/0    0.0
events/0      0.0
cpuset        0.0
khelper       0.0
netns         0.0
```

### 9.2.3 工作原理

选项-o可以使用不同的参数，这些参数及其描述如表9-1所示。

表 9-1

参 数	描 述
pcpu	CPU占用率
pid	进程ID
ppid	父进程ID
pmem	内存使用率
comm	可执行文件名

(续)

参 数	描 述
cmd	简单命令 <sup>①</sup>
user	启动进程的用户
nice	优先级
time	累计的CPU时间
etime	进程启动后流逝的时间
tty	所关联的TTY设备
euid	有效用户ID
stat	进程状态

9.2.4 补充内容

让我们看看其他一些进程控制命令的用法。

1. top

top对于系统管理员来说是一个极为重要的命令。它默认会输出一个占用CPU最多的进程列表。输出结果每隔几秒就会更新。该命令的用法如下所述。

\$ top

除了若干个占用CPU最多的进程外，还会显示进程相关的一些其他参数，如图9-1所示。

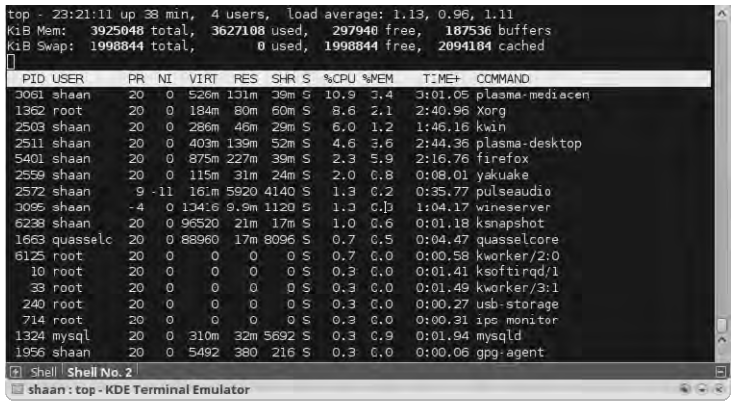


图 9-1

① 简单命令是我们平时使用最频繁的一类命令。它是由空白字符分隔的一系列单词，以shell控制操作符作为结尾。第一个单词指定要执行的命令，余下的单词作为命令参数。shell控制操作符可以是换行符，或者是|, &&, &, ;, ;, ;, |, |&, (, )。

## 2. 根据参数对ps输出进行排序

可以用`--sort`将ps命令的输出根据特定的列进行排序。在参数前加上+（升序）或-（降序）来指定排序方式：

```
$ ps [OPTIONS] --sort -parameter1,+parameter2,parameter3..
```

例如，要列出占用CPU最多的10个进程：

```
$ ps -eo comm,pcpu --sort -pcpu | head
COMMAND      %CPU
Xorg          0.1
hald-addon-stor 0.0
ata/0        0.0
scsi_eh_0     0.0
gnome-settings- 0.0
init         0.0
hald         0.0
pulseaudio   0.0
gdm-simple-gree 0.0
```

进程依据CPU占用率进行降序排序，用head命令提取前10个进程。

我们可以用grep从ps的输出中提取与给定进程名或其他参数相关的条目。要找出与Bash进程相关的条目，可以使用：

```
$ ps -eo comm,pid,pcpu,pmem | grep bash
bash         1255  0.0  0.3
bash         1680  5.5  0.3
```

## 3. 找出给定命令名所对应的进程ID

假设某个命令有多个实例正在运行，我们可能需要识别这些进程的PID。该信息可以使用ps或pgrep命令得到。按照下面的方式使用ps：

```
$ ps -C COMMAND_NAME
```

或者

```
$ ps -C COMMAND_NAME -o pid=
```

用户自定义格式指示符`-o`先前已经讲解过了。但是这里你可以看到pid后面加上了`=`，这会去掉ps输出中PID一列的列名。在参数后加上`=`就可以移除列名。例如：

```
$ ps -C bash -o pid=
1255
1680
```

这条命令列出了所有Bash进程的进程ID。

除此之外，还有一个很方便的工具pgrep。你可以用它获得特定命令的进程ID列表。例如：

```
$ pgrep COMMAND
$ pgrep bash
1255
1680
```



pgrep只需要命令名的一部分作为输入参数来提取Bash命令，诸如pgrep ash或pgrep bas都能够奏效。但是ps需要你输入命令准确的全名。

pgrep可以接受多个输出过滤选项。如果不使用换行符作为定界符，而是要自行指定可以像下面这样：

```
$ pgrep COMMAND -d DELIMITER_STRING
$ pgrep bash -d ":"
1255:1680
```

指定进程的用户（拥有者）列表：

```
$ pgrep -u root,slynux COMMAND
```

其中root和slynux都是用户名。返回所匹配的进程数量：

```
$ pgrep -c COMMAND
```

#### 4. 根据真实用户或ID以及有效用户或ID过滤ps输出

可以用ps根据指定的真实/有效用户名或ID对进程进行分组。指定的参数可以用来过滤ps的输出：通过检查每一个输出条目是否属于参数列表中指定的有效用户或真实用户，并只显示匹配的条目。实现方法如下：

- ❑ 用 -u EUSER1,EUSER2 ...，指定有效用户列表；
- ❑ 用 -U RUSER1,RUSER2 ...，指定真实用户列表。

例如：

```
$ ps -u root -U root -o user,pcpu
```

该命令会显示以root作为有效用户ID和真实用户ID的所有进程，以及用户、CPU占用率列。



在大多数情况下，-o都是和-e结合成-eo的形式。但是当使用过滤器的时候，-o应该像上面那样单独使用。

## 5. 用TTY过滤ps输出

可以通过指定进程所属的TTY选择ps的输出。用选项 `-t` 指定TTY列表：

```
$ ps -t TTY1, TTY2 ..
```

例如：

```
$ ps -t pts/0,pts/1
  PID TTY          TIME CMD
 1238 pts/0      00:00:00 bash
 1835 pts/1      00:00:00 bash
 1864 pts/0      00:00:00 ps
```

## 6. 进程线程的相关信息

通常与进程线程相关的信息在ps输出中是看不到的。我们可以用选项 `-L` 在ps输出中显示线程的相关信息。这会显示出两列：NLWP和NLP。NLWP是进程的线程数量，NLP是ps输出中每个条目的线程ID。例如：

```
$ ps -eLf
```

或者

```
$ ps -eLf --sort -nlwp | head
UID      PID PPID  LWP  C  NLWP STIME TTY          TIME CMD
root      647   1   647  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   654  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   656  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   657  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   658  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   659  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   660  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   662  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
root      647   1   663  0    64 14:39 ?           00:00:00 /usr/sbin/
console-kit-daemon --no-daemon
```

该命令列出了线程数最多的10个进程。

## 7. 指定输出宽度以及所要显示的列

我们可以使用用户自定义的输出格式指示符 `-o` 来指定在ps输出中所要显示的列。另一种指

定输出格式的方法是使用标准选项。可以按照你自己的使用方式来进行应用。尝试以下选项:

```
❑ -f    ps -ef
❑ u     ps  -e u
❑ ps    ps  -e w (w表示宽松输出)
```

## 8. 显示进程的环境变量

了解某个进程依赖哪些环境变量, 这类信息我们通常都用得着。进程的运行方式可能极其依赖某组环境变量。我们可以利用环境变量调试并修复与进程相关的问题。

要在ps的输出条目中同时列出环境变量, 可以使用:

```
$ ps -eo cmd e
```

例如:

```
$ ps -eo pid,cmd e | tail -n 3
1162 hald-addon-acpi: listening on acpid socket /var/run/acpid.socket
1172 sshd: slynux [priv]
1237 sshd: slynux@pts/0
1238 -bash USER=slynux LOGNAME=slynux HOME=/home/slynux
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
MAIL=/var/mail/slynux SHELL=/bin/bash SSH_CLIENT=10.211.55.2 49277 22
SSH_CONNECTION=10.211.55.2 49277 10.211.55.4 22 SSH_TTY=/dev/pts/0 TERM=xterm-color
LANG=en_IN XDG_SESSION_COOKIE=d1e96f5cc8a7a3bc3a0a73e44c95121a-1286499339.
592429-1573657095
```

这种跟踪环境方法的用途之一就是解决apt-get软件包管理器的故障。如果你是用HTTP代理连接Internet, 可能得设置环境变量http\_proxy=host:port。如果你在脚本中忘记设置该环境变量, 那么apt-get就没法选择代理服务器并将返回一个错误。这时就得查看环境变量, 查找问题所在。

我们可能需要借助crontab这类调度工具来使一些应用程序能够自动运行。但是这类应用也许要依赖某些环境变量。假设需要在某个特定时间打开一个窗口化的GUI应用程序。我们用crontab将其安排到指定的时间, 但结果却事与愿违:

```
00 10 * * * /usr/bin/windowapp
```

这是因为窗口化的应用程序总是依赖于环境变量DISPLAY。要想获得所需的环境变量, 首先手动运行windowapp, 然后运行ps -C windowapp -eo cmd e。找出对应的环境变量, 将其添加到crontab中的命令名之前, 这个问题就可以搞定了:

```
00 10 * * * DISPLAY=:0 /usr/bin/windowapp
```

这里的DISPLAY=:0 是从ps输出中获取的。

## 9. which、whereis、file、whatis与平均负载

有一些命令可以用来探究其他的命令。让我们来看一下。

### ❑ which

which命令用来找出某个命令的位置。我们在终端输入命令时无需知道对应的可执行文件位于何处。

当输入命令时，终端会在一组位置中查找这个命令，如果能够找到，那么就执行该可执行文件。这一组位置由环境变量PATH指定。例如：

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

我们可以导出PATH并添上我们自己的命令搜索位置。如果要将/home/slynux/bin添加到PATH中，可以使用以下命令：

```
$ export PATH=$PATH:/home/slynux/bin
# /home/slynux/bin被添加到PATH中
```

which命令会输出作为其参数的命令所在的位置。例如：

```
$ which ls
/bin/ls
```

### ❑ whereis

whereis与which命令类似，但它不仅返回命令的路径，还能够打印出其对应的命令手册的位置以及命令源代码的路径（如果有的话）。例如：

```
$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

### ❑ file

file命令是一个既有趣又使用频繁的命令，可用来确定文件的类型：

```
$ file FILENAME
```

该命令会打印出与该文件类型相关的细节信息。

例如：

```
$ file /bin/ls
/bin/ls: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.15, stripped
```

### ❑ whatis

whatis命令会输出作为参数的命令的简短描述信息。这些信息是从命令手册中解析得来的。例如：

```
$ whatis ls
ls (1)                - list directory contents
```

### **apropos**



有时候我们需要搜索和某个单词相关的命令是否存在。那么可以搜索包含该字符串命令的手册页。为此，我们可以使用以下命令：

```
apropos COMMAND
```

### □ 平均负载

平均负载是系统总负载量的一个重要参数。它指明了系统中可运行进程总量的平均值。平均负载由三个值来指定，第一个值指明了1分钟内的平均值，第二个值指明了5分钟内的平均值，第三个值指明了15分钟内的平均值。

这三个值可以通过运行uptime获得。例如：

```
$ uptime
12:40:53 up 6:16, 2 users, load average: 0.00, 0.00, 0.00
```

## 9.2.5 参考

9.7节讲解了如何调度任务。

## 9.3 杀死进程以及发送或响应信号

终结进程是我们通常都会碰到的事儿，包括需要终结某个程序的所有实例。命令行提供了多种用于终结程序的方法。在类Unix环境中与进程相关的一个重要概念就是信号。信号是一种进程间通信机制，它用来中断运行中的进程以执行某些操作。终止程序也是通过使用信号技术来实现的。这则攻略介绍了信号及其用法。

### 9.3.1 预备知识

信号是Linux中的一种进程间通信机制。我们可以使用特定的信号来中断进程。每一种信号都同一个整数值相关联。当进程接收到一个信号时，它会通过执行对应的信号处理程序（signal handler）来进行响应。在shell脚本中同样可以发送、接收并响应信号。kill是用于终止进程的信号。像Ctrl+C、Ctrl+Z这种事件也会发送各自的信号。kill命令可用来向进程发送信号，而trap命令用来处理所接收的信号。

### 9.3.2 实战演练

(1) 列出所有可用的信号：

```
$ kill -l
```

该命令会打印出信号编号（signal number）以及对应的信号名称。

(2) 终止进程：

```
$ kill PROCESS_ID_LIST
```

kill命令默认发出一个TERM信号。进程ID列表使用空格作为进程ID之间的定界符。

(3) 要通过kill命令向进程发送指定的信号，可以使用：

```
$ kill -s SIGNAL PID
```

参数SIGNAL要么是信号名称，要么是信号编号。尽管有很多信号可用于不同的目的，但经常用到的其实只有那么几个，具体如下所示。

- ☐ SIGHUP 1——对控制进程或终端的终结进行挂起检测（hangup detection）。
- ☐ SIGINT 2——当按下Ctrl + C时发送该信号。
- ☐ SIGKILL 9——用于强行杀死进程。
- ☐ SIGTERM 15——默认用于终止进程。
- ☐ SIGTSTP 20——当按下Ctrl + Z时发送该信号。

(4) 我们经常要强行杀死进程，可以使用：

```
$ kill -s SIGKILL PROCESS_ID
```

或者

```
$ kill -9 PROCESS_ID
```

### 9.3.3 补充内容

让我们看看其他一些用于终止以及向进程发送信号的命令。

#### 1. 相关的kill命令

kill命令以进程ID作为参数。在kill命令系列中还有其他命令可以接受命令名作为参数，向对应的进程发送信号。

killall命令通过命令名终止进程：

```
$ killall process_name
```

通过名称向进程发送信号：

```
$ killall -s SIGNAL process_name
```

通过名称强行杀死进程：

```
$ killall -9 process_name
```

例如：

```
$ killall -9 gedit
```

通过名称以及所属用户名指定进程：

```
$ killall -u USERNAME process_name
```

如果需要在杀死进程前进行确认，可以使用killall的-i选项。

pkill命令和kill命令类似，不过默认情况下pkill接受的是进程名，而非进程ID。例如：

```
$ pkill process_name
$ pkill -s SIGNAL process_name
```

SIGNAL是信号编号。pkill不支持信号名称。pkill提供了很多和kill相同的选项。要了解更多信息，请参阅pkill的命令手册。

## 2. 捕捉并响应信号

trap命令在脚本中用来为信号分配信号处理程序。一旦使用trap将某个函数分配给一个信号，那么当脚本运行收到该信号时，其对应的函数就会开始执行。

命令语法如下：

```
trap 'signal_handler_function_name' SIGNAL LIST
```

SIGNAL LIST以空格分隔，它可以是信号编号或者信号名称。

下面是一个能够响应信号SIGINT的shell脚本：

```
#!/bin/bash
#文件名: sighandle.sh
#用途: 信号处理程序

function handler()
{
    echo Hey, received signal : SIGINT
}

echo My process ID is $$
###$是一个特殊变量，它可以返回当前进程/脚本的进程ID
trap 'handler' SIGINT
```

#handler是信号SIGINT的信号处理程序的名称

```
while true;
do
    sleep 1
done
```

在终端中运行这个脚本。当脚本运行时，如果按Ctrl+C，就会显示一条消息，这是通过执行与信号关联的信号处理程序实现的。Ctrl+C会发出一个SIGINT信号。

通过使用一个无限循环while来保持进程运行。这样就可以使它能够响应另一个进程以异步方式发送的信号。用来保持进程一直处于活动状态的循环通常称为事件循环（event loop）。

我们可以用kill命令以及脚本的进程ID向脚本发送信号：

```
$ kill -s SIGINT PROCESS_ID
```

上面脚本的PROCESS\_ID在执行时会被打印出来。或者也可以用ps命令找出脚本的进程ID。

如果没有为信号指定信号处理程序，那么将会调用由操作系统默认分配的信号处理程序。一般来说，按下Ctrl+C会终止程序，因为这是操作系统提供的处理程序的默认行为。不过这里我们自定义的信号处理程序指定了在接收到信号后所执行的操作。

通过trap命令，我们能够为任何可用的信号（kill -l）定义处理程序，也可以为多个信号指定单个信号处理程序。

## 9.4 向用户终端发送消息

系统管理员可能需要向网络中所有用户或特定用户的终端发送消息。这则攻略将指导你如何完成这项任务。

### 9.4.1 预备知识

wall命令用来向当前所有登录用户的终端写入消息。它可以将消息传递给一台服务器中所有的登录用户或是多台分散主机中的用户。给所有的用户发送消息未必总是靠谱。在Linux系统中，终端是作为设备存在的。因此那些打开的终端在/dev/pts/中都会有对应的设备节点文件。向特定的设备写入数据将会在对应的终端中显示出消息。

### 9.4.2 实战演练

向终端中所有的当前登录用户发送广播消息：

```
$ cat message | wall
```

或者

```
$ wall< message
Broadcast Message from slynux@slynux-laptop
      (/dev/pts/1) at 12:54 ...
```

```
This is a message
```

消息概要（message outline）会显示谁（哪个用户、哪台主机）发送了这则消息。如果其他用户发送了消息，只有在“写入消息”选项启用的情况下该消息才会显示在当前终端中。在绝大多数发布版中，“写入消息”选项都是默认启用的。如果消息的发送者是root用户，那么不管“写入消息”选项是否启用，消息都会显示出来。

要允许写入消息，可以使用：

```
$ mesg y
```

要禁止写入消息，可以使用：

```
$ mesg n
```

让我们写一个给指定用户终端发送消息的脚本：

```
#!/bin/bash
#文件名: message_user.sh
#用途: 用于向指定用户的终端发送消息的脚本
USERNAME=$1

devices=`ls /dev/pts/* -l | awk '{ print $3,$10 }' | grep $USERNAME | awk '{ print $2 }'`
for dev in $devices;
do
    cat /dev/stdin > $dev
done
```

运行脚本：

```
./message_user.sh USERNAME < message.txt
#通过stdin传递消息，USERNAME作为参数
```

输出如下：

```
$ cat message.txt
A message to slynux. Happy Hacking!
#./message_user.sh slynux < message.txt
#因为消息要发送给指定的用户，因此要以root用户身份运行message_user.sh
```

这时slynux的终端就会接收到该消息。

### 9.4.3 工作原理

目录/dev/pts中包含着与每一位系统终端中登录用户所对应的字符设备。我们可以通过查看ls -l输出设备文件的属主来得知谁登入了哪个终端。这些信息可以用awk来获得,然后借助grep提取出对应于指定用户的行。用户名作为脚本的首个参数被存储在变量USERNAME中。\$devices中包含了给定用户的终端列表,该列表通过for循环来进行迭代。/dev/stdin包含传递给当前进程的标准输入数据,这些数据被重定向到对应的终端设备(TTY)。

## 9.5 采集系统信息

从命令行中收集当前系统信息对于记录系统数据来说非常重要。各种系统信息包括主机名、内核版本、Linux发布版名称、CPU信息、内存信息、磁盘分区信息等。这则攻略将为你演示在Linux中采集系统信息的各种途径。

### 实战演练

- (1) 打印当前系统的主机名:

```
$ hostname
```

或者

```
$ uname -n
```

- (2) 打印Linux内核版本、硬件架构等详细信息:

```
$ uname -a
```

- (3) 打印内核发行版本:

```
$ uname -r
```

- (4) 打印主机类型:

```
$ uname -m
```

- (5) 打印CPU相关信息:

```
$ cat /proc/cpuinfo
```

获取处理器名称:

```
$ cat /proc/cpuinfo | sed -n 5p
```

cpuinfo的第5行包含处理器的名称。

(6) 打印内存的详细信息:

```
$ cat /proc/meminfo
```

打印系统可用内存总量:

```
$ cat /proc/meminfo | head -1
MemTotal:      1026096 kB
```

(7) 列出系统的分区信息:

```
$ cat /proc/partitions
```

或者

```
$ fdisk -l #如果没有输出, 切换到root用户执行该命令
```

(8) 获取系统的详细信息:

```
$ lshw #建议以root用户来执行
```

## 9.6 使用 proc 采集信息

在GNU/Linux操作系统中, /proc是一个在内存中的伪文件系统 (pseudo filesystem)。它的引入是为了提供一个可以从用户空间读取系统参数的接口。我们能够从中获取到大量的信息。下面来看看如何使用它。

### 实战演练

如果查看 /proc, 你会发现有很多文件和目录。其中的一些我们在本章的其他攻略中已经讲解过了。你可以对 /proc 及其子目录下的文件使用 cat 来获取信息。所有内容都是易读的格式化文本。

系统中每一个运行的进程在 /proc 中都有一个对应的目录。目录名和进程ID相同。

以 Bash 为例, 它的进程ID是4295 (pgrep bash), 那么就会有一个对应的目录 /proc/4295。进程对应的目录中包含了大量有关进程的信息。/proc/PID 中一些重要的文件如下所示。

- ❑ environ: 包含与进程相关的环境变量。使用 cat /proc/4295/environ, 可以显示所有传递给该进程的环境变量。
- ❑ cwd: 是一个到进程工作目录 (working directory) 的符号链接。
- ❑ exe: 是一个到当前进程所对应的可执行文件的符号链接。

```
$ readlink /proc/4295/exe
/bin/bash
```

❑ fd: 包含了进程所使用的文件描述符。

## 9.7 用 cron 进行调度

我们经常会需要安排脚本在某个时间或每隔一段时间来运行。GNU/Linux系统包含了各种可用于任务调度的工具。cron就是其中之一，它通过守护进程cron使得任务能够按照固定的时间间隔在系统后台自动运行。cron利用的是一个叫做cron表的文件，这个文件中存储了需要执行的脚本或命令的调度列表以及执行时间。一个常见的用法是设置在免费时段（一些ISP提供免费使用时间，通常是在午夜），从Internet上进行下载。用户完全不需要在夜里熬红双眼等待下载。只需要编写一个cron条目，然后调度下载即可。你也可以安排当免费时段结束后自动断开Internet连接并关机。

### 9.7.1 预备知识

所有的GNU/Linux发布版默认都包含了cron调度工具。只要在cron表中写入条目，对应的命令就会在指定的时间执行。crontab命令用来添加作业（job）。cron表不过是一个简单的文本文件而已，每位用户都有自己的副本。

### 9.7.2 实战演练

要想进行任务调度，我们得知道cron表的格式。一项cron作业指定了需要执行的脚本或命令的路径以及执行时间。

- (1) 创建一项cron作业，在每天中每小时的第2分钟执行脚本test.sh:

```
02 * * * * /home/slynux/test.sh
```

- (2) 在每天的第5、6、7小时执行脚本:

```
00 5,6,7 * * * /home/slynux/test.sh
```

- (3) 在周日的每个小时执行脚本script.sh:

```
00 */12 * * * 0 /home/slynux/script.sh
```

- (4) 在每天凌晨2点关闭计算机:

```
00 02 * * * /sbin/shutdown -h
```

- (5) 现在，让我们看看如何调度一项cron作业。利用crontab命令进行调度的方法有很多种。

使用选项 -e来编辑cron表：

```
$ crontab -e
02 02 * * * /home/slynux/script.sh
```

输入crontab -e后，会打开默认的文本编辑器（通常是vi）供用户输入cron作业并保存。该cron作业将会在指定的时间被调度执行。

- (6) 如果我们在脚本中调用crontab进行任务调度，那么有另外两种方法可供使用。

- ① 创建一个文本文件（例如task.cron），并写入cron作业。将文件名作为命令参数，运行crontab：

```
$ crontab task.cron
```

- ② 在行内（inline）指定cron作业，而无需创建单独的文件。例如：

```
crontab<<EOF
02 * * * * /home/slynux/script.sh
EOF
```

cron作业需要写在crontab<<EOF和EOF之间。

### 9.7.3 工作原理

cron表中的每一个条目都由6部分组成，并按照下列顺序排列：

- ❑ 分钟（0~59）
- ❑ 小时（0~23）
- ❑ 天（1~31）
- ❑ 月份（1~12）
- ❑ 工作日（0~6）
- ❑ 命令（在指定时间执行的脚本或命令）

前5部分指定了开始执行某个命令实例的时间。还有其他一些选项也可用来指定调度时间。

星号（\*）指定命令应该在每个时间段执行。也就是说，如果\*是写在cron作业中的小时字段中，那么命令就会每小时执行一次。与此类似，如果你希望在某个特定时段执行命令，那么就在对应的时间字段中指定时段，并用逗号分隔（例如要在5分钟和10分钟时运行命令，那就在分钟字段中输入"5,10"）。还有另一个不错的选项可以让我们以特定的时间间隔运行命令。在分钟字段使用\*/5，可以每5分钟运行一次命令。这个技巧可以用在任何时间字段。一个cron表条目

是由一项或多项cron作业组成的。cron表条目中的每一行都是一项作业。

执行cron作业所使用的权限同执行crontab命令所使用的权限相同。如果你需要执行要求更高权限的命令，例如关闭计算机，那么就要以root用户身份执行crontab。

在cron作业中指定的命令需要使用完整路径。这是因为执行cron作业时的环境与终端所使用的环境不同，环境变量PATH可能都没有设置。如果命令运行时需要设置某些环境变量，就应该明确地进行设定。

### 9.7.4 补充内容

crontab命令还包括其他选项。让我们看看其中一部分。

#### 1. 指定环境变量

很多命令需要正确的设置环境变量才能够运行。我们可以在用户的cron表中插入一行变量赋值语句来设置环境变量。

例如，如果你使用的是代理服务器连接互联网，要调度某个需要使用互联网的命令，就得设置HTTP代理环境变量http\_proxy，可以用下面的方法来完成：

```
crontab<<EOF
http_proxy=http://192.168.0.3:3128
00 * * * * /home/slynux/download.sh
EOF
```

#### 2. 在系统启动时运行命令

有时候需要在系统启动时运行特定的命令。实现这一需求的方法有很多，cron便是其中之一。（其他一些方法是将命令添加到/etc/rc.d中，但这并不能保证在所有的发布版中都行得通。）

要在启动时运行命令，请将下面一行加入crontab：

```
@reboot command
```

这样就会以你所具有的用户身份来运行指定的命令。如果需要以root用户的身份运行命令，需要编辑root用户的crontab。

#### 3. 查看cron表

我们可以用选项-l列出cron表中现有的内容：

```
$ crontab -l
02 05 * * * /home/user/disklog.sh
```

`crontab -l`会列出当前用户cron表中的已有条目。

我们也可以通过选项 `-u` 来查看其他用户的cron表：

```
$ crontab -l -u slynux
09 10 * * * /home/slynux/test.sh
```

当使用选项 `-u` 时，你必须作为root用户以获取更高的权限。

#### 4. 删除cron表

可以使用选项 `-r` 删除当前用户的cron表：

```
$ crontab -r
```

要删除其他用户的cron表，可以使用：

```
# crontab -u slynux -r
```

这需要以root用户身份以获得更高的权限。

## 9.8 从 Bash 中读写 MySQL 数据库

MySQL是一款应用广泛的数据库管理系统。以PHP、Python、C++等语言编写的应用程序通常使用它作为存储系统。从shell脚本中访问并处理MySQL数据库也很有意思。我们可以编写脚本将文本文件或CSV（Comma Separated Value，逗号分隔值文件）的内容写入数据表，与MySQL进行交互来读取、处理数据。例如，可以从shell脚本中执行查询语句来读取存储在留言板程序的数据库中的所有电子邮件地址。在这则攻略中，我们会看到如何从Bash中读写MySQL数据库。假设现在有以下问题。

我有一个包含学生详细信息的CSV文件。我需要将文件的内容插入到一个数据表中。要保证为每一个系生成一个单独的排名列表。

### 9.8.1 预备知识

要处理MySQL数据库，系统中必须安装mysql-server和mysql-client软件包。Linux发布版中默认并没有包含这些工具。由于MySQL要使用用户名和密码进行认证，在安装MySQL服务器时还得设置用户名和密码。

### 9.8.2 实战演练

前面提出的这个问题可以用sort、awk等工具解决，或者用一个SQL数据库的数据表也可以搞定。

我们接下来要编写3个脚本，分别用于创建数据库及数据表、向数据表中插入学生数据、从数据表中读取并显示处理过的数据。

创建数据库及数据表的脚本如下：

```
#!/bin/bash
#文件名: create_db.sh
#用途: 创建MySQL数据库和数据表

USER="user"
PASS="user"

mysql -u $USER -p$PASS <<EOF 2> /dev/null
CREATE DATABASE students;
EOF

[ $? -eq 0 ] && echo Created DB || echo DB already exist
mysql -u $USER -p$PASS students <<EOF 2> /dev/null
CREATE TABLE students(
id int,
name varchar(100),
mark int,
dept varchar(4)
);
EOF

[ $? -eq 0 ] && echo Created table students || echo Table students already exist

mysql -u $USER -p$PASS students <<EOF
DELETE FROM students;
EOF
```

将数据插入数据表的脚本如下：

```
#!/bin/bash
#文件名: write_to_db.sh
#用途: 从CSV中读取数据并写入MySQLdb

USER="user"
PASS="user"

if [ $# -ne 1 ];
then
    echo $0 DATAFILE
    echo
    exit 2
fi

data=$1
```

```

while read line;
do

    oldIFS=$IFS
    IFS=,
    values=($line)
    values[1]="\"`echo ${values[1]} | tr ' ' '#' '\\"
    values[3]="\"`echo ${values[3]}\"`\"

    query=`echo ${values[@]} | tr ' #' ', ' `
    IFS=$oldIFS

    mysql -u $USER -p$PASS students <<EOF
INSERT INTO students VALUES($query);
EOF

```

```

done< $data
echo Wrote data into DB

```

查询数据库的脚本如下：

```

#!/bin/bash
#文件名: read_db.sh
#用途:从数据库中读取数据

USER="user"
PASS="user"

depts=`mysql -u $USER -p$PASS students <<EOF | tail -n +2
SELECT DISTINCT dept FROM students;
EOF`

for d in $depts;
do

echo Department : $d

result=`mysql -u $USER -p$PASS students <<EOF
SET @i:=0;
SELECT @i:=@i+1 as rank,name,mark FROM students WHERE dept="$d" ORDER BY mark DESC;
EOF`

echo "$result"
echo

done

```

作为输入的CSV文件（studentdata.csv）中的数据如下：

```
1,Navin M,98,CS
2,Kavya N,70,CS
3,Nawaz O,80,CS
4,Hari S,80,EC
5,Alex M,50,EC
6,Neenu J,70,EC
7,Bob A,30,EC
8,Anu M,90,AE
9,Sruthi,89,AE
10,Andrew,89,AE
```

按照以下顺序执行脚本：

```
$ ./create_db.sh
Created DB
Created table students

$ ./write_to_db.sh studentdat.csv
Wrote data into DB

$ ./read_db.sh
Department : CS
rank name mark
1 Navin M 98
2 Nawaz O 80
3 Kavya N 70

Department : EC
rank name mark
1 Hari S 80
2 Neenu J 70
3 Alex M 50
4 Bob A 30

Department : AE
rank name mark
1 Anu M 90
2 Sruthi 89
3 Andrew 89
```

### 9.8.3 工作原理

我们现在来逐个讲解上面的脚本。第一个脚本create\_db.sh用来创建数据库students，并在其中创建数据表students。我们需要MySQL的用户名和密码来访问或修改数据库中的内容。变量USER和PASS用来存储用户名和密码。mysql命令用于对MySQL进行操作。该命令可以用-u指定

用户名，用`-pPASSWORD`指定密码，其他命令参数是数据库名。如果将数据库名作为`mysql`命令的参数，那么就将使用该数据库，否则就必须用`use database_name`明确地为SQL查询语句指明使用哪一个数据库进行查询。`mysql`命令通过标准输入（`stdin`）接受查询。通过`stdin`提供多行输入的简便方法是使用`<<EOF`。出现在`<<EOF`和`EOF`之间的文本被作为`mysql`的标准输入。在`CREATE DATABASE`语句中，为了避免显示错误信息，我们将`stderr`重定向到`/dev/null`。同样，在创建数据表时，我们也将`stderr`重定向到`/dev/null`，以忽略可能出现的任何错误。然后用退出状态变量`$?`来检查`mysql`命令的退出状态，以获知是否已经存在同名的数据库或数据表。如果已经存在，则会显示出一条提示信息；否则，就进行创建。

接下来的脚本`write_to_db.sh`接受包含学生数据的CSV文件名。我们用`while`循环读取CSV文件的每一行，因此在每次迭代中都会接收到一行以逗号分隔的数值。然后将行内的数值放入SQL查询语句中。要实现这个目的，最简单的方法是用数组存储CSV文件行中的数据项。我们知道数组赋值的形式为`array=(val1 val2 val3)`，其中内部字段分隔符（IFS）是空格。文本行用逗号分隔数值，因此只需要将IFS修改成逗号（`IFS=,`），就可以轻松地赋值给数组。文本行中以逗号分隔的数据项分别是`id`、`name`、`mark`和`department`。`id`和`mark`是整数，而`name`和`department`是字符串（字符串必须进行引用）。`name`中也可以包含空格。这样一来就和IFS产生了冲突。因此我们应该将`name`中的空格替换成其他字符（`#`），在构建查询语句时再替换回来。为了引用字符串，数组中的值要加上`\"`作为前缀和后缀。`tr`用来将`name`中的空格替换成`#`。最后通过将空格替换成逗号，将`#`替换成空格来构造出查询语句并进行查询。

第三个脚本`read_db.sh`用来查找各系并打印出每个系的学生排名列表。第一个查询用来找出各系的名称。我们用一个`while`循环对每个系进行迭代，然后进行查询并按照成绩从高到低的顺序显示学生的详细信息。`SET @i:=0`是一个SQL构件（SQL construct），用来设置变量`i=0`。在每一行中，变量`i`都会增加并作为学生排名来显示。

## 9.9 用户管理脚本

GNU/Linux是一个多用户操作系统，多个用户可以同时登录并执行多种操作。有一些管理任务会涉及用户管理，这包括为用户设置默认shell、禁用某个账户、禁用某个shell账户、添加新用户、删除用户、设置密码、设置账户有效期等。这则攻略旨在编写一个可以处理此类任务的用户管理工具。

### 9.9.1 实战演练

让我们看看这个用户管理脚本：

```
#!/bin/bash
#文件名: user_adm.sh
```

#用途： 用户管理工具

```
function usage()
{
    echo Usage:
    echo Add a new user
    echo $0 -adduser username password
    echo
    echo Remove an existing user
    echo $0 -deluser username
    echo
    echo Set the default shell for the user
    echo $0 -shell username SHELL_PATH
    echo
    echo Suspend a user account
    echo $0 -disable username
    echo
    echo Enable a suspended user account
    echo $0 -enable username
    echo
    echo Set expiry date for user account
    echo $0 -expiry DATE
    echo
    echo Change password for user account
    echo $0 -passwd username
    echo
    echo Create a new user group
    echo $0 -newgroup groupname
    echo
    echo Remove an existing user group
    echo $0 -delgroup groupname
    echo
    echo Add a user to a group
    echo $0 -addgroup username groupname
    echo
    echo Show details about a user
    echo $0 -details username
    echo
    echo Show usage
    echo $0 -usage
    echo

    exit
}

if [ $UID -ne 0 ];
then
    echo Run $0 as root.
    exit 2
```

```

fi

case $1 in

    -adduser) [ $# -ne 3 ] && usage ; useradd $2 -p $3 -m ;;
    -deluser) [ $# -ne 2 ] && usage ; deluser $2 --remove-all-files;;
    -shell)   [ $# -ne 3 ] && usage ; chsh $2 -s $3 ;;
    -disable) [ $# -ne 2 ] && usage ; usermod -L $2 ;;
    -enable) [ $# -ne 2 ] && usage ; usermod -U $2 ;;
    -expiry) [ $# -ne 3 ] && usage ; chage $2 -E $3 ;;
    -passwd) [ $# -ne 2 ] && usage ; passwd $2 ;;
    -newgroup) [ $# -ne 2 ] && usage ; addgroup $2 ;;
    -delgroup) [ $# -ne 2 ] && usage ; delgroup $2 ;;
    -addgroup) [ $# -ne 3 ] && usage ; addgroup $2 $3 ;;
    -details) [ $# -ne 2 ] && usage ; finger $2 ; chage -l $2 ;;
    -usage) usage ;;
    *) usage ;;

esac

```

输出如下:

```

# ./user_admin.sh -details test
Login: test                Name:
Directory: /home/test      Shell: /bin/sh
Last login Tue Dec 21 00:07 (IST) on pts/1 from localhost
No mail.
No Plan.
Last password change       : Dec 20, 2010
Password expires           : never
Password inactive          : never
Account expires            : Oct 10, 2010
Minimum number of days between password change : 0
Maximum number of days between password change : 99999
Number of days of warning before password expires : 7

```

## 9.9.2 工作原理

脚本user\_admin.sh可以用来执行多种用户管理任务。你可以参考usage()中的内容来学习这个脚本的用法。当用户给出的参数不正确或使用-usage选项时, 函数usage()用来显示脚本不同选项的使用方法。case语句用来匹配命令参数, 并根据参数执行对应的命令。脚本user\_admin.sh合法的命令选项是:-adduser, -deluser, -shell, -disable, -enable, expiry, -passwd, -newgroup, -delgroup, -addgroup, -details和-usage。如果匹配了\*)分支, 那就意味着用户输入了错误的选项, 因此要调用usage()。对于每一个匹配分支, 使用[ \$# -ne 3 ] && usage。它用来检测参数个数。如果命令参数个数不等于要求的数量, 则调用函数usage()并退出脚本。要运行用户管理命令, 需要以超级用户身份执行脚本。因此要检查用户ID是否为0 (root

用户的用户ID是0)。如果用户ID非0,则表明脚本不是以root用户身份执行的,因此显示出要求以root用户身份运行脚本的提示消息并退出。

下面来逐个讲解每个选项。

❑ **-useradd**

useradd命令可以用来创建新用户。命令语法如下:

**useradd USER -p PASSWORD**

❑ 选项-m用来创建home目录。也可以用选项-c FULLNAME提供用户的全名。

❑ **-deluser**

deluser命令用来删除用户。命令语法如下:

**deluser USER**

❑ **--remove-all-files**用来删除与用户相关的所有文件,包括home目录。

❑ **-shell**

chsh命令用来修改用户的默认shell。命令语法如下:

**chsh USER -s SHELL**

❑ **-disable**和**-enable**

usermod命令用来处理和用户账户相关的若干属性信息。usermod -L USER和usermod -U USER分别用来锁定和解锁用户账户。

❑ **-expiry**

chage命令用来处理用户账户的过期信息。命令语法如下:

**chage -E DATE**

其他选项如下:

- **-m MIN\_DAYS** (将更改密码的最小天数修改成MIN\_DAYS);
- **-M MAX\_DAYS** (设置密码有效的最大天数);
- **-W -WARN\_DAYS** (设置在前几天提醒需要更改密码)。

❑ **-passwd**

passwd命令用来更改用户密码。命令语法如下:

**passwd USER**

命令会提示输入新的密码。

#### ❑ -newgroup和addgroup

addgroup命令会为系统添加一个新的用户组。命令语法如下：

```
addgroup GROUP
```

要将已有的用户添加到一个组，可以使用：

```
addgroup USER GROUP  
-delgroup
```

delgroup命令会删除一个用户组。命令语法如下：

```
delgroup GROUP
```

#### ❑ -details

finger USER命令会显示用户信息，这包括用户的home目录、上一次登录的时间、默认shell等。chage -l命令会显示用户账户的过期信息。

## 9.10 图像文件的缩放及格式转换

我们大家都会使用数码相机，也会从网上获取数码照片。如果需要处理大量图像文件，我们可以轻松地用脚本进行批量处理。这通常会涉及到的任务是调整照片的大小，还有转换图像格式（例如，将JPEG格式转换成PNG格式）。当从数码相机中取到照片时，大分辨率的图片体积都比较大，我们可能需要减少图片的大小，以便于存储并通过电子邮件发送。因此就需要调整图片的大小来降低分辨率。这则攻略将讨论如何用脚本管理图片。

### 9.10.1 预备知识

我们要用到convert命令，它来自Imagemagick软件包，该软件包中包含了大量出色的图像处理工具，拥有各类选项的丰富，能够处理多种图像格式。大多数GNU/Linux发布版中并没有预装Imagemagick。你得自己手动安装。更多的信息请访问[www.imagemagick.org](http://www.imagemagick.org)。

### 9.10.2 实战演练

将一种图像格式转换为另一种图像格式：

```
$ convert INPUT_FILE OUTPUT_FILE
```

例如：

```
$ convert file1.png file2.png
```

我们可以通过指定缩放比或输出图像的宽度和高度来调整图像。

指定WIDTH（宽度）或HEIGHT（高度）来缩放图像：

```
$ convert image.png -resize WIDTHxHEIGHT image.png
```

例如：

```
$ convert image.png -resize 1024x768 image.png
```

必须提供WIDTH或HEIGHT，这样才能够自动计算其他数值，以便在保留图像比例的同时进行缩放。

```
$ convert image.png -resize WIDTHx image.png
```

例如：

```
$ convert image.png -resize 1024x image.png
```

指定百分比缩放图像：

```
$ convert image.png -resize "50%" image.png
```

让我们看一个用于图像管理的脚本：

```
#!/bin/bash
#文件名: image_help.sh
#用途: 图像管理脚本

if [ $# -ne 4 -a $# -ne 6 -a $# -ne 8 ];
then
    echo Incorrect number of arguments
    exit 2
fi

while [ $# -ne 0 ];
do

    case $1 in
        -source) shift; source_dir=$1 ; shift ;;
        -scale) shift; scale=$1 ; shift ;;
        -percent) shift; percent=$1 ; shift ;;
        -dest) shift ; dest_dir=$1 ; shift ;;
        -ext) shift ; ext=$1 ; shift ;;
        *) echo Wrong parameters; exit 2 ;;
    esac;

done
```

```

for img in `echo $source_dir/*` ;
do
    source_file=$img
    if [[ -n $ext ]];
    then
        dest_file=${img%.*}.$ext
    else
        dest_file=$img
    fi

    if [[ -n $dest_dir ]];
    then
        dest_file=${dest_file##*/}
        dest_file="$dest_dir/$dest_file"
    fi

    if [[ -n $scale ]];
    then
        PARAM="-resize $scale"
    elif [[ -n $percent ]];
    then
        PARAM="-resize $percent%"
    fi

    echo Processing file : $source_file
    convert $source_file $PARAM $dest_file

done

```

下面是输出样例，将目录sample\_dir中的图像调整到原来的20%：

```

$ ./image_help.sh -source sample_dir -percent 20%
Processing file :sample/IMG_4455.JPG
Processing file :sample/IMG_4456.JPG
Processing file :sample/IMG_4457.JPG
Processing file :sample/IMG_4458.JPG

```

将图像宽度调整到1024像素：

```
$ ./image_help.sh -source sample_dir -scale 1024x
```

把-ext png加入上面的命令，使文件格式转换成PNG。

将文件缩放或转换到指定的目录：

```

$ ./image_help.sh -source sample -scale 50% -ext png -dest newdir
# newdir作为目的目录

```

### 9.10.3 工作原理

上面的脚本image\_help.sh可以接受多个命令行参数，例如-source、-percent、-scale、-ext和-dest等。每个选项的简短描述如下。

- ❑ -source用于指定图像源目录。
- ❑ -percent用于指定缩放比例，-scale用于指定缩放宽度与高度。
- ❑ -percent与-scale不能同时出现，只能使用其中之一。
- ❑ -ext用于指定目标文件格式。-ext是一个可选的选项。如果没有指定，那么不执行格式转换。
- ❑ -dest为缩放或转换格式后的文件指定目标目录。该选项也是可选的。如果没有指定，目标目录则和源目录相同。脚本的第一步就是检查命令行参数的数量是否正确，可以出现的参数数量分别是4、6或8。

借助while循环和case语句，我们将命令行参数解析到对应的变量。\$#是一个特殊的变量，它可以返回参数的数量。shift命令每执行一次，就将命令行参数向左移动一个位置，这样我们就不需要使用变量\$1、\$2、\$3等，而只用一个\$1就可以对命令参数逐个访问了。case语句用来匹配\$1的值，就像C语言中的switch语句一样。如果匹配了某个case分支，就执行对应的语句。每一个case分支都以;;作为结尾。一旦将所有的参数都解析到变量percent、scale、source\_dir、ext和dest\_dir中，就用for循环对源目录中的每一个文件进行迭代，并执行对应的转换操作。

如果变量ext已定义（也就是说-ext作为命令参数出现），就将目标文件的扩展名从source\_file.extension更改为source\_file.\$ext。接下来检查是否提供了-dest选项。如果指定了目标目录，则使用文件名切片将源路径中的目录替换成目标目录，从而形成目标文件路径。随后构造出convert命令的参数，用以执行缩放（-resize widthx 或-resize perc%）。参数构造完毕之后，用对应的参数执行convert命令。

### 9.10.4 参考

2.12节讲解了如何提取部分文件名。

## 9.11 从终端截图

截图是另一种常见的计算机用户日常操作。对于维护GUI应用以及自动化管理的系统管理员而言，这种操作更为重要。当特定事件出现时，关键就是要抓取截图，以便知道GUI应用到底出了什么事。

### 9.11.1 预备知识

我们要使用一个来自ImageMagick软件包的工具，该工具包在上一个攻略中已经有所涉及。请使用软件包管理器自行安装。

### 9.11.2 实战演练

下面我们使用ImageMagick中的import工具进行截图。

(1) 取整个屏幕：

```
$ import -window root screenshot.png
```

(2) 手动选择部分区域进行抓取：

```
$ import screenshot.png
```

(3) 抓取特定窗口：

```
$ import -window window_id screenshot.png
```

使用命令xwininfo，点击需要抓取截图的窗口来获得window\_id。然后将window\_id传递给import命令的-window选项。

## 9.12 管理多个终端

如果你频繁地使用shell，大概会注意到有时候需要同时访问多个终端。如果你使用的是如Konsole这类图形化终端模拟器，就得借助多个标签（tab）才能满足这种需求。

但是如果如果没有图形化终端模拟器该怎么办呢？或者说登录到远程主机，希望使用多个shell，又该怎么办呢？在后一种情况中，打开多个ssh连接基本上只能浪费网络带宽，拖慢运行速度而已。接下来我们会看到如何在避免这些问题的同时获得多个shell。

### 9.12.1 预备知识

要实现这一点，需要使用一个叫做GNU screen的工具。如果你使用的发布版中没有预装screen，请使用软件包管理器自行安装。

### 9.12.2 实战演练

❑ 创建新的screen窗口：从shell中运行screen来创建一个新的screen。你会看到一条欢迎消

息以及一些该软件的相关信息。按空格键或回车键开始。接着会获得一个新的shell用以输入命令。要创建新窗口（基本上相当于一个新shell），使用Ctrl+A+C（区分大小写）。

- ❑ 查看已打开的窗口列表：在运行screen时，如果要查看所有打开窗口，使用Ctrl+A+"。
- ❑ 在窗口之间切换：我们通常需要以下一个/前一个（next/previous）的方式在打开的窗口之间进行切换。可以使用Ctrl+A和Ctrl+N切换到下一个窗口，使用Ctrl+A和Ctrl+P切换到前一个窗口。
- ❑ 关联与脱离screen：screen具备一个有用的特性，能够让你保持、载入screen会话，用screen的术语来说，叫做脱离（detaching）与关联（attaching）。使用Ctrl+A和Ctrl+D脱离当前screen会话。要关联到一个已有的screen会话，可以使用：

```
screen -r -d
```

该命令告诉screen关联到上一个screen会话。如果已脱离的会话不止一个，screen会用列表输出会话，然后可以使用下述命令：

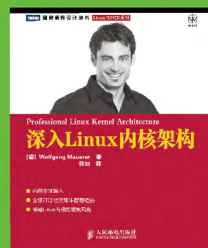
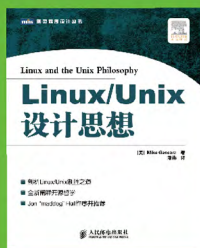
```
screen -r -d PID
```

这里，PID是你想关联到的screen会话的PID。

Linux/GNU是一款非凡的操作系统，拥有稳定可靠且极其强大的开发环境。作为与操作系统进行沟通的原生界面，shell能够控制整个操作系统的运作，是与Linux进行交互最灵活的手段。本书向读者展现了如何有效地利用shell完成复杂的任务。从shell的基础知识开始，学习简单命令的用法，对各类文件进行操作。随后讲解了文本处理、Web交互、备份、监视以及其他系统管理任务。第2版进行了全面修订，精选极具实用价值的技巧，让你的日常工作更加轻松。

- 考察了各类日常任务以及如何利用shell命令更快速地解决问题。
- 编写脚本从Web中挖掘数据，然后利用几行代码进行处理。
- 综合利用多种工具来解决问题。
- 在脚本中与简单的Web API进行交互。
- 任务的执行及自动化，例如利用归档工具实现自动备份和恢复。
- 创建及维护文件和文件夹归档，利用shell进行压缩和加密。
- 利用shell脚本设置以太网和无线局域网。
- 利用日志监视网络活动。

## 好书推荐



**[PACKT]**  
PUBLISHING

图灵社区: iTuring.cn

热线: (010)51095186 转 600

分类建议 计算机 / 程序设计 / Linux

人民邮电出版社网址: www.ptpress.com.cn

ISBN 978-7-115-33921-8



9 787115 339218 >

ISBN 978-7-115-33921-8

定价: 59.00元

欢迎加入

# 图灵社区

## 最前沿的IT类电子书发售平台

电子出版的时代已经来临。在许多出版界同行还在犹豫彷徨的时候，图灵社区已经采取实际行动拥抱这个出版业巨变。作为国内第一家发售电子图书的IT类出版商，图灵社区目前为读者提供两种DRM-free的阅读体验：在线阅读和PDF。

相比纸质书，电子书具有许多明显的优势。它不仅发布快，更新容易，而且尽可能采用了彩色图片（即使有的书纸质版是黑白印刷的）。读者还可以方便地进行搜索、剪贴、复制和打印。

图灵社区进一步把传统出版流程与电子书出版业务紧密结合，目前已实现作者网上交稿、编辑网上审稿、按章发布的电子出版模式。这种新的出版模式，我们称之为“敏捷出版”，它可以让读者以较快的速度了解到国外最新技术图书的内容，弥补以往翻译版技术书“出版即过时”的缺憾。同时，敏捷出版使得作、译、编、读的交流更为方便，可以提前消灭书稿中的错误，最大程度地保证图书出版的质量。

## 最方便的开放出版平台

图灵社区向读者开放在线写作功能，协助你实现自出版和开源出版梦想。利用“合集”功能，你就能联合二三好友共同创作一部技术参考书，以免费或收费的形式提供给读者。（收费形式须经过图灵社区立项评审。）这极大地降低了出版的门槛。只要有写作的意愿，图灵社区就能帮助你实现这个梦想。成熟的书稿，有机会入选出版计划，同时出版纸质书。

图灵社区引进出版的外文图书，都将在立项后马上在社区公布。如果你有意翻译哪本图书，欢迎你来社区申请。只要你通过试译的考验，即可签约成为图灵的译者。当然，要想成功地完成一本书的翻译工作，是需要有坚强的毅力的。

## 最直接的读者交流平台

在图灵社区，你可以十分方便地写文章、提交勘误、发表评论，以各种方式与作译者、编辑人员和其他读者进行交流互动。提交勘误还能够获赠社区银子。

你可以积极参与社区经常开展的访谈、审读、评选等多种活动，赢取积分和银子，积累个人声望。

ituring.com.cn