

2.4. 查看脚本生成的输出文件。

```
[student@servera ~]$ cat output.txt
This is my first bash script

#####
#
```

► 3. 向 **firstscript.sh** 脚本中添加更多命令，执行它并查看输出。

3.1. 使用 **vim** 文本编辑器编辑 **firstscript.sh**

```
[student@servera ~]$ vim firstscript.sh
```

3.2. 将以下粗体标出的行附加到文件 **firstscript.sh**。

```
#!/bin/bash
#
echo "This is my first bash script" > ~/output.txt
echo "" >> ~/output.txt
echo "#####" >> ~/output.txt
echo "LIST BLOCK DEVICES" >> ~/output.txt
echo "" >> ~/output.txt
lsblk >> ~/output.txt
echo "" >> ~/output.txt
echo "#####" >> ~/output.txt
echo "FILESYSTEM FREE SPACE STATUS" >> ~/output.txt
echo "" >> ~/output.txt
df -h >> ~/output.txt
echo "#####" >> ~/output.txt
```

3.3. 使用 **chmod** 命令使 **firstscript.sh** 文件可执行。

```
[student@servera ~]$ chmod a+x firstscript.sh
```

3.4. 执行 **firstscript.sh** 脚本。

```
[student@servera ~]$ ./firstscript.sh
```

3.5. 查看脚本生成的输出文件。

```
[student@servera ~]$ cat output.txt
This is my first bash script

#####
LIST BLOCK DEVICES

NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0     11:0    1 1024M  0 rom
vda     252:0    0   10G  0 disk
└─vda1  252:1    0   10G  0 part /
vdb     252:16   0    5G  0 disk
```

```
#####
# FILESYSTEM FREE SPACE STATUS
#
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        892M    0  892M   0% /dev
tmpfs          915M    0  915M   0% /dev/shm
tmpfs          915M   17M  899M   2% /run
tmpfs          915M    0  915M   0% /sys/fs/cgroup
/dev/vda1       10G  1.5G  8.6G  15% /
tmpfs         183M    0  183M   0% /run/user/1000
#####
```

► 4. 删除练习文件并从 `servera` 注销。

4.1. 删除脚本文件 `firstscript.sh` 和输出文件 `output.txt`。

```
[student@servera ~]$ rm firstscript.sh output.txt
```

4.2. 从 `servera` 注销。

```
[student@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$
```

## 完成

在 `workstation` 上，运行 `lab console-write finish` 脚本来完成本练习。

```
[student@workstation ~]$ lab console-write finish
```

本引导式练习到此结束。

# 使用循环更高效地运行命令

## 培训目标

学完本节后，您应能够：

- 使用 **for** 循环迭代列表。
- 通过命令和脚本评估退出代码。
- 使用运算符执行测试。
- 使用 **if** 语句创建条件结构。

## 使用循环来迭代命令

系统管理员在其日常活动中经常会遇到重复任务。重复任务可能表现为对目标多次执行某个操作，例如在 10 分钟内，每隔一分钟检查进程来查看其是否已完成。任务重复还可能表现为一次对多个目标执行某个操作，例如对系统上的每个数据库进行备份。**for** 循环是 Bash 提供的多个 shell 循环结构之一，并且可用于任务迭代。

## 从命令行处理项目

Bash 的 **for** 循环结构使用以下语法。

```
for VARIABLE in LIST; do  
    COMMAND VARIABLE  
done
```

循环按顺序逐一处理 LIST 中提供的字符串，并且在处理列表中的最后一个字符串之后退出。列表中的每个字符串临时存储为 VARIABLE 的值，而 **for** 循环执行包含在其结构中的命令块。变量的命名是任意的。通常，变量值由命令块中的命令进行引用。

可以通过多种方式来为 **for** 循环提供字符串列表。可以是用户直接输入的字符串的列表，或者是通过不同类型的 shell 扩展生成，如变量扩展、花括号扩展、文件名扩展或命令替换。下面部分示例演示了可以向 **for** 循环提供字符串的不同方式。

```
[user@host ~]$ for HOST in host1 host2 host3; do echo $HOST; done  
host1  
host2  
host3  
[user@host ~]$ for HOST in host{1,2,3}; do echo $HOST; done  
host1  
host2  
host3  
[user@host ~]$ for HOST in host{1..3}; do echo $HOST; done  
host1  
host2  
host3  
[user@host ~]$ for FILE in file*; do ls $FILE; done  
filea
```

```

fileb
filec
[user@host ~]$ for FILE in file{a..c}; do ls $FILE; done
filea
fileb
filec
[user@host ~]$ for PACKAGE in $(rpm -qa | grep kernel); \
do echo "$PACKAGE was installed on \
$(date -d @$($rpm -q --qf "%{INSTALLTIME}\n" $PACKAGE))"; done
abrt-addon-kerneloops-2.1.11-12.el7.x86_64 was installed on Tue Apr 22 00:09:07
EDT 2014
kernel-3.10.0-121.el7.x86_64 was installed on Thu Apr 10 15:27:52 EDT 2014
kernel-tools-3.10.0-121.el7.x86_64 was installed on Thu Apr 10 15:28:01 EDT 2014
kernel-tools-libs-3.10.0-121.el7.x86_64 was installed on Thu Apr 10 15:26:22 EDT
2014
[user@host ~]$ for EVEN in $(seq 2 2 10); do echo "$EVEN"; done
2
4
6
8
10

```

## 在脚本中使用退出代码

在处理完自己的所有内容后，脚本会退出到调用它的进程。但是，有时候可能需要在完成之前退出脚本，比如在遇到错误条件时。可以通过在脚本中使用 **exit** 命令来实现这一目的。当脚本遇到 **exit** 命令时，脚本将立即退出且不会对脚本的其余内容进行处理。

可以使用可选的整数参数（**0** 到 **255** 之间，表示退出代码）来执行 **exit** 命令。退出代码是进程完成返回的代码。退出代码值 **0** 表示没有错误。所有其他非零值都表示存在错误的退出代码。您可以使用不同的非零值来区分遇到的不同类型错误。此退出代码传回到父进程，后者将它存储在 **\$?** 变量中，并可通过 **\$?** 进行访问，如以下示例所示。

```

[user@host bin]$ cat hello
#!/bin/bash
echo "Hello, world"
exit 0

[user@host bin]$ ./hello
Hello, world

[user@host bin]$ echo $?
0

```

如果不使用任何参数调用 **exit** 命令，那么脚本将退出并且将最后执行的命令的退出状态传递给父进程。

## 测试脚本输入

为确保脚本不会由于意外情况而轻易中断，一种良好的做法是不要进行与输入有关的假设。如命令行参数、用户输入、命令替换、变量扩展及文件名扩展。可以使用 Bash 的 **test** 命令来执行完整性检查。

与所有命令一样，**test** 命令会在完成后生成一个退出代码，该退出代码存储为值 \$?。要查看测试的结论，请在执行 **test** 命令后立即显示 \$? 的值。再次说明，退出状态值 0 表示测试成功，而非零值表示测试失败。

执行测试时要用到多种运算符。运算符可用于确定某个数值是大于、大于等于、小于、小于等于，还是等于另一个数值。它们可用于测试某个文本字符串与另一个文本字符串是相同还是不同。运算符也可用于评估变量是否具有某个值。



### 注意

除了这里所讲的比较运算符之外，还有其他多种运算符可用于 shell 脚本编写。**test(1)** 的 man page 中列出了重要的条件表达式运算符及其说明。此外，**bash(1)** 的 man page 中还介绍了运算符的使用和评估，但这对于初学者来说读起来会非常困难。建议学员借助专门针对 shell 编程的书籍和课程，进一步满足他们的高级 shell 脚本编写需求。

以下示例演示了如何使用 Bash 的数字比较运算符来使用该**test** 命令

```
[user@host ~]$ test 1 -gt 0 ; echo $?
0
[user@host ~]$ test 0 -gt 1 ; echo $?
1
```

可以使用 Bash 的测试命令语法 [ <TESTEXPRESSION> ] 来执行测试。也可以使用 Bash 的较新扩展测试命令语法 [[ <TESTEXPRESSION> ]]（在 Bash 版本 2.02 及更高版本中可用，可提供诸如通配模式匹配和正则表达式模式匹配等功能）执行这些测试。

以下示例演示了使用 Bash 的 test 命令语法和 Bash 的数字比较运算符

```
[user@host ~]$ [ 1 -eq 1 ]; echo $?
0
[user@host ~]$ [ 1 -ne 1 ]; echo $?
1
[user@host ~]$ [ 8 -gt 2 ]; echo $?
0
[user@host ~]$ [ 2 -ge 2 ]; echo $?
0
[user@host ~]$ [ 2 -lt 2 ]; echo $?
1
[user@host ~]$ [ 1 -lt 2 ]; echo $?
0
```

以下示例演示了 Bash 的字符串比较运算符的使用。

```
[user@host ~]$ [ abc = abc ]; echo $?
0
[user@host ~]$ [ abc == def ]; echo $?
1
[user@host ~]$ [ abc != def ]; echo $?
0
```

以下示例演示了 Bash 的字符串一元运算符的使用。

```
[user@host ~]$ STRING=''; [ -z "$STRING" ]; echo $?
0
[user@host ~]$ STRING='abc'; [ -n "$STRING" ]; echo $?
0
```

### 注意

测试中括号内的空格字符是必需的，因为它们用于分隔测试表达式中的词语和元素。借助内置解析规则，shell 的命令解析例程会通过识别空格和其他元字符将所有命令行划分成相应的词语和运算符。有关这种高级概念的完整处理方式，请参阅 man page **getopt(3)**。左方括号字符 (`[]`) 本身就是 **test** 命令的一个内置别名。无论是命令、子命令、选项、参数还是其他令牌元素，shell 词语始终用空格分隔。

## 条件结构

简单的 shell 脚本表示从头到尾执行的命令的集合。条件结构允许用户在 shell 脚本中包含决策，以便仅当满足特定条件时才执行脚本的特定部分。

### 使用 if/then 结构

Bash 中最简单的条件结构是 if/then 结构，其语法如下。

```
if <CONDITION>; then
    <STATEMENT>
    ...
    <STATEMENT>
fi
```

通过此结构，如果满足给定条件，将采取一个或多个操作。如果不满足给定条件，则不采取任何操作。前面演示的数字、字符串和文件测试经常用于在 **if/then** 语句中测试条件。末尾的 **fi** 语句用于结束 **if/then** 结构。以下代码段演示了使用 **if/then** 结构来启动 **psacct** 服务（如果其未处于活动状态）。

```
[user@host ~]$ systemctl is-active psacct > /dev/null 2>&1
[user@host ~]$ if [ $? -ne 0 ]; then
> sudo systemctl start psacct
> fi
```

### 使用 if/then/else 结构

**if/then** 结构可以进一步扩展，以便能够根据是否满足条件来采取不同的操作集合。使用 **if/then/else** 结构可实现此目标。

```
if <CONDITION>; then
    <STATEMENT>
    ...
    <STATEMENT>
else
    <STATEMENT>
```

```

...
<STATEMENT>
fi

```

以下代码段演示了使用 **if/then/else** 语句来启动 psacct 服务（如果其未处于活动状态）和停止该服务（如果其处于活动状态）。

```

[user@host ~]$ systemctl is-active psacct > /dev/null 2>&1
[user@host ~]$ if [ $? -ne 0 ]; then
> sudo systemctl start psacct
> else
> sudo systemctl stop psacct
> fi

```

## 使用 **if/then/elif/then/else** 结构

最后，**if/then/elif/then/else** 结构可以进一步扩展以测试多个条件：在满足某个条件时执行不同的操作集合。以下示例中显示了其结构：

```

if <CONDITION>; then
    <STATEMENT>
    ...
    <STATEMENT>
elif <CONDITION>; then
    <STATEMENT>
    ...
    <STATEMENT>
else
    <STATEMENT>
    ...
    <STATEMENT>
fi

```

在此条件结构中，Bash 将按照显示的顺序测试条件。在发现某个条件成立后，Bash 将执行与该条件相关联的操作，然后跳过条件结构的其余部分。如果所有条件均不成立，Bash 将执行 **else** 子句中枚举的操作。

以下代码段演示了使用 **if/then/elif/then/else** 语句来运行 mysql 客户端（如果 mariadb 服务处于活动状态），运行 psql 客户端（如果 postgresql 服务处于活动状态）或运行 sqlite3 客户端（如果 mariadb 和 postgresql 服务均未处于活动状态）。

```

[user@host ~]$ systemctl is-active mariadb > /dev/null 2>&1
MARIADB_ACTIVE=$?
[user@host ~]$ sudo systemctl is-active postgresql > /dev/null 2>&1
POSTGRESQL_ACTIVE=$?
[user@host ~]$ if [ "$MARIADB_ACTIVE" -eq 0 ]; then
> mysql
> elif [ "$POSTGRESQL_ACTIVE" -eq 0 ]; then
> psql
> else
> sqlite3
> fi

```



## 参考文献

**bash(1)** man page

## ► 指导练习

# 使用循环更高效地运行命令

在本练习中，您将使用循环从多个服务器高效地显示主机名。

### 成果

您应能够创建一个 **for** 循环，用于迭代命令行和 shell 脚本中的项目列表。

### 在你开始之前

在 workstation 上，以 student 用户身份并使用 student 作为密码进行登录。

在 workstation 上，运行 **lab console-commands start** 命令。该命令将运行一个起始脚本，它将确定 servera 和 serverb 主机是否可从网络访问。如果不可用，脚本会提醒您。

```
[student@workstation ~]$ lab console-commands start
```

- 1. 使用 **ssh** 和 **hostname** 命令将 servera 和 serverb 的主机名显示到标准输出。

```
[student@workstation ~]$ ssh student@servera hostname  
servera.lab.example.com  
[student@workstation ~]$ ssh student@serverb hostname  
serverb.lab.example.com
```

- 2. 创建 **for** 循环以提高执行同一任务的效率。

```
[student@workstation ~]$ for HOST in servera serverb  
do  
ssh student@${HOST} hostname  
done  
servera.lab.example.com  
serverb.lab.example.com
```

- 3. 创建 shell 脚本来执行同一 **for** 循环。

3.1. 创建 **/home/student/bin** 目录以包含 shell 脚本。

```
[student@workstation ~]$ mkdir ~/bin
```

3.2. 验证新创建的目录是否位于 **PATH** 的环境变量中。

```
[student@workstation ~]$ echo $PATH  
/home/student/.local/bin:/home/student/bin::/usr/local/bin:/usr/bin:/usr/local/  
sbin:/usr/sbin
```

- 3.3. 在 `/home/student/bin/printhostname.sh` 中创建 shell 脚本来执行 `for` 循环。  
使用 `cat` 命令验证 `printhostname.sh` 的内容。

```
[student@workstation ~]$ vim ~/bin/printhostname.sh
[student@workstation ~]$ cat ~/bin/printhostname.sh
#!/bin/bash
#Execute for loop to print server hostname.
for HOST in servera serverb
do
    ssh student@${HOST} hostname
done
exit 0
```

- 3.4. 确保新创建的脚本可以执行。

```
[student@workstation ~]$ chmod +x ~/bin/printhostname.sh
```

- 3.5. 从主目录运行该脚本。

```
[student@workstation ~]$ printhostname.sh
servera.lab.example.com
serverb.lab.example.com
```

- 3.6. 验证脚本的退出代码是否为 0。

```
[student@workstation ~]$ echo $?
0
```

## 完成

在 `workstation` 上，运行 `lab console-commands finish` 脚本来结束本练习。

```
[student@workstation ~]$ lab console-commands finish
```

本引导式练习到此结束。

# 使用正则表达式匹配命令输出中的文本

## 培训目标

学完本节后，学员应能够：

- 创建可匹配所需数据的正则表达式。
- 使用 **grep** 命令将正则表达式应用于文本文件。
- 使用 **grep**，通过竖线命令搜索文件和数据。

## 编写正则表达式

正则表达式提供了一种便于查找特定内容的模式匹配机制。**vim**、**grep** 和 **less** 命令都可以使用正则表达式。诸如 Perl、Python 和 C 等编程语言在使用模式匹配条件时，也都会使用正则表达式。

正则表达式自成体系，也就是说，该语言有其自身的语法和规则。本章节将探讨创建正则表达式时所用的语法，并展示一些正则表达式的示例。

## 描述简单的正则表达式

最为简单的正则表达式是完全匹配。如果正则表达式中的字符与正在搜索的数据中的类型和顺序均匹配，即为完全匹配。

假设用户正在以下文件中查找出现的所有 **cat** 模式：

```
cat  
dog  
concatenate  
dogma  
category  
educated  
boondoggle  
vindication  
chilidog
```

**cat** 与 c 加 a 加 t（中间没有其他字符）完全匹配。使用 **cat** 作为正则表达式来搜索上一个文件将返回以下匹配项：

```
cat  
concatenate  
category  
educated  
vindication
```

## 匹配行首和行尾

以上部分对文件使用了完全匹配正则表达式。请注意，无论正则表达式在行的哪个位置（行首、行中或行尾）执行操作，均会匹配搜索字符串。使用行定位符控制正则表达式在哪个位置上查找匹配项。

要在行首搜索，请使用脱字符号 (^)。要在行尾搜索，请使用美元符号 (\$)。

在使用上述文件的情况下，**^cat** 正则表达式将匹配两个词语。**\$cat** 正则表达式找不到任何匹配的词语。

```
cat
dog
concatenate
dogma
category
educated
boondoggle
vindication
chilidog
```

要想在文件中查找以 **dog** 为结尾的行，请使用该精确表达式和行尾定位符来创建正则表达式 **dog\$**。将 **dog\$** 应用到文件将会找到两个匹配项：

```
dog
chilidog
```

要查找一行中的唯一词语，请同时使用行首和行尾定位符。例如，要查找作为一行中唯一词语的 **cat**，请使用 **^cat\$**。

```
cat dog rabbit
cat
horse cat cow
cat pig
```

## 向正则表达式中添加通配符和倍数

正则表达式使用句点或点(.) 来匹配除换行符之外的任何单个字符。正则表达式 **c.t** 将搜索包含 **c** 加任何单个字符加 **t** 的字符串。匹配示例包括：**cat**、**concatenate**、**vindication**、**c5t** 和 **c\$t**。

使用不受限制的通配符时，您将无法预测与通配符匹配的字符。要匹配特定字符，请将不受限制的通配符替换为可接受的字符。将正则表达式更改为 **c[aou]t** 会匹配以下模式：以 **c** 开头，后面跟着 **a**、**o** 或 **u**，然后是 **t**。

倍数是常与通配符一起使用的机制。倍数应用到正则表达式中的前一位字符。更为常用的倍数之一是星号 (\*)。在正则表达式中使用时，该倍数表示匹配前一表达式的零项或多项。您可以在表达式中使用 \*，而不仅限于字符。示例：**c[aou]\*t**。正则表达式 **c.\*t** 将匹配 **cat**、**coat**、**culvert** 乃至 **ct** (**c** 和 **t** 之间没有字符)。任何以 **c** 开头，后面跟着零个或多个字符，最后以 **t** 结尾的数据。

另一种类型的倍数将会指示模式中前面字符的期望个数。`'c.\{2\}t'` 是使用显式倍数的一个示例。该正则表达式将匹配以 **c** 开头，后面跟着任意两个字符，最后以 **t** 结尾的任何词语。`'c.\{2\}t'` 与以下示例中的两个词语匹配：

```
cat
coat convert
cart covert
cypher
```



### 注意

由于正则表达式常含有 shell 元字符（如 `$`、`*` 和 `{}`），建议练习使用单引号来括起正则表达式。这样可确保字符由命令解释，而不是由 shell 解释。



### 注意

本课程介绍了两种不同的元字符文本解析系统：shell 模式匹配（也称为文件通配或文件名扩展）和正则表达式。由于这两个系统都使用相似的元字符，如星号（`*`），但在元字符解释和规则方面存在差异，因此在完全掌握每种系统之前，二者之间可能会存在混淆。

模式匹配是一项命令行解析技术，它可以轻松指定多个文件名，并且主要支持在命令行上表示文件名的模式。正则表达式则设计为用文本字符串来表示任何形式或模式，无论内容有多么复杂。在内部，正则表达式受到大量文本处理命令（如 **grep**、**sed**、**awk**、**python**、**perl**）和诸多应用程序的支持，在解释规则上存在与命令有关的细微差异。

## 正则表达式

选项	描述
.	句点(.) 匹配任何单个字符。
?	前面的项目是可选的，且最多匹配一次。
*	前面的项目将匹配零次或多次。
+	前面的项目将匹配一次或多次。
{n}	前面的项目恰好匹配 n 次。
{n,}	前面的项目匹配 n 次或更多次。
{,m}	前面的项目最多匹配 m 次。
{n,m}	前面的项目至少匹配 n 次，但不超过 m 次。
[:alnum:]	字母数字字符：' <code>[:alpha:]</code> ' 和 ' <code>[:digit:]</code> '；在 'C' 语言环境和 ASCII 字符编码中，它等同于 ' <code>[0-9A-Za-z]</code> '。
[:alpha:]	字母字符：' <code>[:lower:]</code> ' 和 ' <code>[:upper:]</code> '；在 'C' 语言环境和 ASCII 字符编码中，它等同于 ' <code>[A-Za-z]</code> '。

选项	描述
[:blank:]	空白字符：空格和制表符。
[:cntrl:]	控制字符。在 ASCII 中，这些字符对应八进制代码 000 到 037 和 177 (DEL)。在其他字符集中，它们为对等字符（如果有）。
[:digit:]	数字：0 1 2 3 4 5 6 7 8 9。
[:graph:]	图形字符：'[:alnum:]' 和 '[:punct:]'。
[:lower:]	小写字母；在 'C' 语言环境和 ASCII 字符编码中，它对应于 a b c d e f g h i j k l m n o p q r s t u v w x y z。
[:print:]	可打印字符：'[:alnum:]'、'[:punct:]' 和空格。
[:punct:]	标点符号；在 'C' 语言环境和 ASCII 字符编码中，它对应于 !"#\$%&()' * +,-./:;<=>@[\]^_`{ }~。在其他字符集中，它们为对等字符（如果有）。
[:space:]	空格字符；在 “C” 语言环境中，它对应于制表符、换行符、垂直制表符、换页符、回车符和空格。
[:upper:]	大写字母；在 'C' 语言环境和 ASCII 字符编码中，它对应于 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z。
[:xdigit:]	十六进制数字：0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f。
\b	匹配词语两侧的空字符串。
\B	匹配词语中间的空字符串。
\<	匹配词语开头的空字符串。
\>	匹配词语末尾的空字符串。
\w	匹配词语组分。'[_[:alnum:]]' 的同义词。
\W	匹配非词语组分。'[^_[:alnum:]]' 的同义词。
\s	匹配空格。'[:space:]' 的同义词。
\S	匹配非空格。'[^[:space:]]' 的同义词。

## 通过 GREP 匹配正则表达式

用作部分分发的 **grep** 命令使用正则表达式来隔离匹配的数据。

### 使用 grep 命令隔离数据

**grep** 命令提供了一个正则表达式和一个文件（该文件中应有此正则表达式的匹配项）。

```
[user@host ~]$ grep '^computer' /usr/share/dict/words
computer
computerese
computerise
```

```
computerite
computerizable
computerization
computerize
computerized
computerizes
computerizing
computerlike
computernik
computers
```

**注意**

由于正则表达式常含有 shell 元字符（如 \$、\* 和 {}），建议练习使用单引号来括起正则表达式。这样可确保字符由 **grep** 解释，而不是由 shell 解释。

通过竖线运算符 (|)，**grep** 命令可与其他命令一起使用。例如：

```
[root@host ~]# ps aux | grep chrony
chrony      662  0.0  0.1  29440  2468 ?          S     10:56   0:00 /usr/sbin/chronyd
```

## grep 选项

**grep** 命令具有许多有用的选项，用于调整 **grep** 如何使用提供的正则表达式来处理数据。

### 常见 grep 选项表

选项	功能
<b>-i</b>	使用所提供的正则表达式，但不会强制区分大小写（运行不区分大小写的操作）。
<b>-v</b>	仅显示不包含正则表达式匹配项的行。
<b>-r</b>	将递归地匹配正则表达式的数据搜索应用到一组文件或目录中。
<b>-A NUMBER</b>	显示正则表达式匹配项之后的行数。
<b>-B NUMBER</b>	显示正则表达式匹配项之前的行数。
<b>-e</b>	如果使用多个 <b>-e</b> 选项，则可以提供多个正则表达式，并将与逻辑 OR 一起使用。

**grep** 还有很多其他选项。使用 **man page** 可以对它们进行研究。

## grep 示例

以下示例中会用到各种配置文件和日志文件。

默认情况下，正则表达式会区分大小写。对 **grep** 使用 **-i** 选项可以执行区分大小写的搜索。以下示例将搜索模式 **serverroot**。

```
[user@host ~]$ cat /etc/httpd/conf/httpd.conf
...output omitted...
ServerRoot "/etc/httpd"

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 80
...output omitted...
```

```
[user@host ~]$ grep -i serverroot /etc/httpd/conf/httpd.conf
# with "/", the value of ServerRoot is prepended -- so 'log/access_log'
# with ServerRoot set to '/www' will be interpreted by the
# ServerRoot: The top of the directory tree under which the server's
# ServerRoot at a non-local disk, be sure to specify a local disk on the
# same ServerRoot for multiple httpd daemons, you will need to change at
ServerRoot "/etc/httpd"
```

如果您知道哪些内容不是自己要查找的，那么 **-v** 选项会非常有用。**-v** 选项仅显示与正则表达式不匹配的行。在以下示例中，将返回不含正则表达式 **server** 的所有行（不管大小写如何）。

```
[user@host ~]$ cat /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

172.25.254.254 classroom.example.com classroom
172.25.254.254 content.example.com content
172.25.254.254 materials.example.com materials
172.25.250.254 workstation.lab.example.com workstation
### rht-vm-hosts file listing the entries to be appended to /etc/hosts

172.25.250.10 servera.lab.example.com servera
172.25.250.11 serverb.lab.example.com serverb
172.25.250.254 workstation.lab.example.com workstation

[user@host ~]$ grep -v -i server /etc/hosts
127.0.0.1 localhost localhost.localdomain localhost4 localhost4.localdomain4
::1 localhost localhost.localdomain localhost6 localhost6.localdomain6

172.25.254.254 classroom.example.com classroom
172.25.254.254 content.example.com content
172.25.254.254 materials.example.com materials
172.25.250.254 workstation.lab.example.com workstation
```

```
### rht-vm-hosts file listing the entries to be appended to /etc/hosts
172.25.250.254 workstation.lab.example.com workstation
```

要查找不受命令行影响的文件，请使用 **-v** 选项。在以下示例中，正则表达式将会匹配所有以 # 或 ; 开头的行（# 和 ; 是表示行将被解释为注释的典型字符）。然后，输出中将略去这些行。

```
[user@host ~]$ cat /etc/ethertypes
#
# Ethernet frame types
#      This file describes some of the various Ethernet
#      protocol types that are used on Ethernet networks.
#
# This list could be found on:
#          http://www.iana.org/assignments/ethernet-numbers
#          http://www.iana.org/assignments/ieee-802-numbers
#
# <name>    <hexnumber> <alias1>...<alias35> #Comment
#
IPv4      0800     ip ip4      # Internet IP (IPv4)
X25      0805
ARP      0806     ether-arp   #
FR_ARP    0808           # Frame Relay ARP      [RFC1701]
...output omitted...

[user@host ~]$ grep -v '^#[;]' /etc/ethertypes
IPv4      0800     ip ip4      # Internet IP (IPv4)
X25      0805
ARP      0806     ether-arp   #
FR_ARP    0808           # Frame Relay ARP      [RFC1701]
```

对 **grep** 命令使用 **-e** 选项将允许您一次搜索多个正则表达式。以下示例使用了 **less** 和 **grep** 的组合，它可以查找 **/var/log/secure** 日志文件中出现的所有 **pam\_unix**、**root** 用户和已接受的公钥。

```
[root@host ~]# cat /var/log/secure | grep -e 'pam_unix' \
-e 'user root' -e 'Accepted publickey' | less
Mar 19 08:04:46 jegui sshd[6141]: pam_unix(sshd:session): session opened for user
root by (uid=0)
Mar 19 08:04:50 jegui sshd[6144]: Disconnected from user root 172.25.250.254 port
41170
Mar 19 08:04:50 jegui sshd[6141]: pam_unix(sshd:session): session closed for user
root
Mar 19 08:04:53 jegui sshd[6168]: Accepted publickey for student from
172.25.250.254 port 41172 ssh2: RSA SHA256:M8ikhcEDm2tQ95Z0o7ZvufqEix0Fst
+wowZLNzNlBT0
```

要在用 **vim** 或 **less** 打开的文件中搜索文本，请使用斜杠字符 (/) 并输入要查找的模式。按 **Enter** 开始搜索。按 **N** 查找下一个匹配项。

```
[root@host ~]# vim /var/log/boot.log
...output omitted...
[^[[[0;32m OK ^[[0m Reached target Initrd Default Target.^M
Starting dracut pre-pivot and cleanup hook...^M
[^[[[0;32m OK ^[[0m Started dracut pre-pivot and cleanup hook.^M
Starting Cleaning Up and Shutting Down Daemons...^M
Starting Plymouth switch root service...^M
Starting Setup Virtual Console...^M
[^[[[0;32m OK ^[[0m Stopped target Timers.^M
[^[[[0;32m OK ^[[0m Stopped dracut pre-pivot and cleanup hook.^M
[^[[[0;32m OK ^[[0m Stopped target Initrd Default Target.^M
/Daemons

[root@host ~]# less /var/log/messages
...output omitted...
Feb 26 15:51:07 jegui NetworkManager[689]: <info> [1551214267.8584] Loaded device
plugin: NMTeamFactory (/usr/lib64/NetworkManager/1.14.0-14.el8/libnm-device-
plugin-team.so)
Feb 26 15:51:07 jegui NetworkManager[689]: <info> [1551214267.8599] device (lo):
carrier: link connected
Feb 26 15:51:07 jegui NetworkManager[689]: <info> [1551214267.8600] manager:
(lo): new Generic device (/org/freedesktop/NetworkManager/Devices/1)
Feb 26 15:51:07 jegui NetworkManager[689]: <info> [1551214267.8623] manager:
(ens3): new Ethernet device (/org/freedesktop/NetworkManager/Devices/2)
Feb 26 15:51:07 jegui NetworkManager[689]: <info> [1551214267.8653] device
(ens3): state change: unmanaged -> unavailable (reason 'managed', sys-iface-
state: 'external')
/device
```



### 参考文献

regex(7) 和 grep(1) man page

## ► 指导练习

# 使用正则表达式匹配命令输出中的文本

在本实验中，您将搜索系统日志和命令输出中的文本，以便更高效地查找信息。

## 成果

您应该能高效地搜索日志文件和配置文件中的文本。

## 在你开始之前

以 student 用户身份并使用 student 作为密码登录 workstation。

在 workstation 上，运行 **lab console-regex start** 命令。此命令将运行一个起始脚本，它将确定 servera 计算机是否可从网络访问。此外，它还将安装 postfix 软件包。

```
[student@workstation ~]$ lab console-regex start
```

- 1. 使用 **ssh** 命令，以 student 用户身份登录 servera。系统已配置为使用 SSH 密钥来进行身份验证，因此不需要提供密码。

```
[student@workstation ~]$ ssh student@servera  
...output omitted...  
[student@servera ~]$
```

- 2. 使用 **sudo -i** 命令，切换为 root 用户。student 用户的密码为 student。

```
[student@servera ~]$ sudo -i  
[sudo] password for student: student  
[root@servera ~]#
```

- 3. postfix 软件包现已由 **start** 脚本安装好。使用 **grep** 命令查找 postfix 与 postdrop 组和用户的 GID 和 UID。要减少 **grep** 命令的输出，则显示始于某个特定 **Start Time** 的所有日志。

3.1. 使用 **date** 命令可确定当前时间。

```
[root@servera ~]# date  
Fri Mar 22 08:23:56 CET 2019
```

3.2. 对 **grep** 命令使用日期、开始时间和 GID 选项可以查找 postfix 和 postdrop 用户的 GID 和 UID。实验设置脚本已在当前时间之前几分钟开始运行。搜索 **/var/log/secure** 日志文件时，要考虑到这一点。

```
[root@servera ~]# grep '^Mar 22 08:2.*GID' /var/log/secure
Mar 22 08:20:04 servera groupadd[2514]: group added to /etc/group: name=postdrop,
GID=90
Mar 22 08:20:04 servera groupadd[2514]: new group: name=postdrop, GID=90
Mar 22 08:20:04 servera groupadd[2520]: group added to /etc/group: name=postfix,
GID=89
Mar 22 08:20:04 servera groupadd[2520]: new group: name=postfix, GID=89
Mar 22 08:20:04 servera useradd[2527]: new user: name=postfix, UID=89, GID=89,
home=/var/spool/postfix, shell=/sbin/nologin
```

- 4. 修改正则表达式，以查找 **/var/log/maillog** 文件中的前两条消息。请注意，该搜索中没有使用脱字符号 (\)，因为您搜索的并不是一行中的第一个字符。

```
[root@servera ~]# grep 'postfix' /var/log/maillog | head -n 2
Mar 22 08:21:02 servera postfix/postfix-script[3879]: starting the Postfix mail
system
Mar 22 08:21:02 servera postfix/master[3881]: daemon started -- version 3.3.1,
configuration /etc/postfix
```

- 5. 您需要找到 Postfix 服务器的 **queue** 目录的名称。要获取有关队列的所有信息，请搜索 **/etc/postfix/main.cf** 配置文件。使用 **-i** 选项可忽略大小写区分。

```
[root@servera ~]# grep -i 'queue' /etc/postfix/main.cf
# testing. When soft_bounce is enabled, mail will remain queued that
# The queue_directory specifies the location of the Postfix queue.
queue_directory = /var/spool/postfix
# QUEUE AND PROCESS OWNERSHIP
# The mail_owner parameter specifies the owner of the Postfix queue
# is the Sendmail-compatible mail queue listing command.
# setgid_group: The group for mail submission and queue management
```

- 6. 确认 **postfix** 将消息写入 **/var/log/messages**。使用 **less** 命令加斜杠字符 (/) 搜索文件。按 **n** 移动到与搜索相匹配的下一个条目。使用 **q** 键退出 **less** 命令。

```
[root@servera ~]# less /var/log/messages
...output omitted...
Mar 22 07:58:04 servera systemd[1]: Started Postfix Mail Transport Agent.
...output omitted...
Mar 22 08:12:26 servera systemd[1]: Stopping Postfix Mail Transport Agent...
Mar 22 08:12:26 servera systemd[1]: Stopped Postfix Mail Transport Agent.
...output omitted...
/Postfix
```

- 7. 使用 **ps aux** 命令确认 postfix 服务器当前是否正在运行。通过与 **grep** 命令组合使用，可以减少 **ps aux** 的输出。

```
[root@servera ~]# ps aux | grep postfix
root      3881  0.0  0.2 121664  5364 ?        Ss   08:21   0:00 /usr/
libexec/postfix/master -w
postfix   3882  0.0  0.4 147284  9088 ?        S    08:21   0:00 pickup -l -t unix
-u
postfix   3883  0.0  0.4 147336  9124 ?        S    08:21   0:00 qmgr -l -t unix -
u
```

- 8. 确认 qmgr、cleanup 和 pickup 队列是否配置正确。对 **grep** 命令使用 **-e** 选项可匹配同一文件中的多个条目。相应的配置文件为 **/etc/postfix/master.cf**。

```
[root@servera ~]# grep -e qmgr -e pickup -e cleanup /etc/postfix/master.cf
pickup    unix n      -      n      60      1      pickup
cleanup   unix n      -      n      -       0      cleanup
qmgr      unix n      -      n      300     1      qmgr
#qmgr     unix n      -      n      300     1      oqmgr
```

- 9. 从 servera 注销。

```
[root@servera ~]# exit
logout
[student@servera ~]$ exit
logout
Connection to servera closed.
[student@workstation ~]$
```

## 完成

在 workstation 上，运行 **lab console-regex finish** 脚本来完成本练习。

```
[student@workstation ~]$ lab console-regex finish
```

本引导式练习到此结束。

## ► 开放研究实验

# 提高命令行生产率

### 任务执行清单

在本实验中，您将创建一个 Bash 脚本，它可以过滤并从不同的主机中获取相关信息。

### 成果

您应能够：

- 创建 Bash 脚本并将其输出重定向到文件。
- 使用循环来简化代码。
- 使用 **grep** 和正则表达式过滤相关内容。

### 在你开始之前

以 **student** 用户身份并使用 **student** 作为密码登录 **workstation**。

在 **workstation** 上，运行 **lab console-review start** 命令。该命令将运行一个起始脚本，它将确定 **workstation**、**servera** 和 **serverb** 计算机是否可从网络访问。如果不可用，脚本会提醒您。此外，它还将安装 **vim-enhanced** 和 **util-linux** 软件包（如果需要），配置 **sudo**，并在 **servera** 和 **serverb** 上准备 **/var/log/secure** 的内容。

```
[student@workstation ~]$ lab console-review start
```

1. 在 **workstation** 上创建 **/home/student/bin/bash-lab** 脚本文件。
2. 编辑新创建的脚本文件，使之符合 **servera** 和 **serverb** 主机所要求的以下信息。系统已配置为使用 SSH 密钥来进行身份验证；不需要提供密码。

命令或文件	要求的内容
<b>hostname -f</b>	获取所有输出。
<b>echo "#####"</b>	获取所有输出。
<b>lscpu</b>	只获取以字符串 <b>CPU</b> 开头的行。
<b>echo "#####"</b>	获取所有输出。
<b>/etc/selinux/config</b>	忽略空行。忽略以 # 开头的行。
<b>echo "#####"</b>	获取所有输出。
<b>/var/log/secure</b>	获取所有“Failed password”条目
<b>echo "#####"</b>	获取所有输出。

将所需的信息保存到新文件 `/home/student/output-servera` 和 `/home/student/output-serverb` 中。



### 注意

您可以在 `servera` 和 `serverb` 主机上使用 `sudo`, 无需密码。请记住使用循环来简化脚本。您也可以将多个 `grep` 命令与竖线字符 (`|`) 连接在一起使用。

3. 执行 `/home/student/bin/bash-lab` 脚本, 然后在 `workstation` 上查看输出内容。

## 评估

在 `workstation` 上, 运行 `lab console-review grade` 命令来确认是否成功完成本练习。

```
[student@workstation ~]$ lab console-review grade
```

## 完成

在 `workstation` 上, 运行 `lab console-review finish` 脚本来完成本练习。

```
[student@workstation ~]$ lab console-review finish
```

本实验到此结束。

## ► 解决方案

# 提高命令行生产率

### 任务执行清单

在本实验中，您将创建一个 Bash 脚本，它可以过滤并从不同的主机中获取相关信息。

### 成果

您应能够：

- 创建 Bash 脚本并将其输出重定向到文件。
- 使用循环来简化代码。
- 使用 **grep** 和正则表达式过滤相关内容。

### 在你开始之前

以 **student** 用户身份并使用 **student** 作为密码登录 **workstation**。

在 **workstation** 上，运行 **lab console-review start** 命令。该命令将运行一个起始脚本，它将确定 **workstation**、**servera** 和 **serverb** 计算机是否可从网络访问。如果不可用，脚本会提醒您。此外，它还将安装 **vim-enhanced** 和 **util-linux** 软件包（如果需要），配置 **sudo**，并在 **servera** 和 **serverb** 上准备 **/var/log/secure** 的内容。

```
[student@workstation ~]$ lab console-review start
```

1. 在 **workstation** 上创建 **/home/student/bin/bash-lab** 脚本文件。

- 1.1. 在 **workstation** 上，创建 **/home/student/bin/** 文件夹（如果需要）。

```
[student@workstation ~]$ mkdir -p /home/student/bin
```

- 1.2. 使用 **vim** 创建并编辑 **/home/student/bin/bash-lab** 脚本文件。

```
[student@workstation ~]$ vim ~/bin/bash-lab
```

- 1.3. 插入以下文本并保存文件。

```
#!/bin/bash
```

- 1.4. 使您的脚本文件成为可执行文件。

```
[student@workstation ~]$ chmod a+x ~/bin/bash-lab
```

2. 编辑新创建的脚本文件，使之符合 **servera** 和 **serverb** 主机所要求的以下信息。系统已配置为使用 SSH 密钥来进行身份验证；不需要提供密码。

命令或文件	要求的内容
<code>hostname -f</code>	获取所有输出。
<code>echo "#####"</code>	获取所有输出。
<code>lscpu</code>	只获取以字符串 <b>CPU</b> 开头的行。
<code>echo "#####"</code>	获取所有输出。
<code>/etc/selinux/config</code>	忽略空行。忽略以 # 开头的行。
<code>echo "#####"</code>	获取所有输出。
<code>/var/log/secure</code>	获取所有 “Failed password” 条目。
<code>echo "#####"</code>	获取所有输出。

将所需的信息保存到新文件 `/home/student/output-servera` 和 `/home/student/output-serverb` 中。



### 注意

您可以在 `servera` 和 `serverb` 主机上使用 `sudo`, 无需密码。请记住使用循环来简化脚本。您也可以将多个 `grep` 命令与竖线字符 (`|`) 连接在一起使用。

2.1. 使用 `vim` 打开并编辑 `/home/student/bin/bash-lab` 脚本文件。

```
[student@workstation ~]$ vim ~/bin/bash-lab
```

2.2. 将以下粗体标出的行附加到脚本文件 `/home/student/bin/bash-lab`。



### 注意

以下是如何实现所请求的脚本的示例。在 Bash 脚本编写中，您可以采用不同的方法并得到相同的结果。