作者 Jayzen (/users/75ef7f2c9d47) 2016.07.14 08:55*
写了60898字，被12人关注，获得了14个喜欢
(/users/75ef7f2c9d47)

linux command

第22章
第23章
第24章

编写 Shell 脚本
第25章
第26章
第27章
第28章
第29章
第30章
第31章
第32章
第33章
第34章
第35章
第36章
第37章

第1章
内容略
第2章
bash是shell的增强版本
和shell交互的工具叫终端访问器，一般称为terminal

```
[me@linuxbox ~]$ #用户名@主机名，如果最后一个字符是#，则说明超级管理员用户
```

简单命令

```
cal #显示日历
date #显示日期
df #显示磁盘剩余空间数量
free #显示空闲内存的数量
exit #退出终端
```

第3章：文件系统中跳转
linux的文件结构形式是树结果，根节点是"/",所有的文件都是挂载在这个根节点上面，绝对路径是从这个根节点开始，相对路径参考下面。

```
pwd #显示当前目录
ls #显示所有文件
ls -a #会显示隐藏文件
ls -l #显示长格式文件
cd #改变文件目录
cd - #更改到先前的工作目录
cd ~ #改变到文件当前目录下面
. #当前目录
.. #当前目录的家目录
```

第4章：探究操作系统

```
file filename #在文本中查看文件类型
less filename #查看文件内容，按q退出
cat filename #查看文件内容
```

显示长格式内容以及各自的意义

```
#格式参考command -options arguments
ls -l
#显示结果：
-rw-r--r-- 1 root root 3576296 2007-04-03 11:05 ubuntu.ogg
```

-rw-r--r-- ： 第一个符号表示文件的类型，-表示普通文件，第一个位数表示文件拥有者的权限，第二个三位数表示用户组的权限，第三个三位数表示其他用户的权限
1：表示文件的硬链接数目
root: 表示文件所有者名称
root: 表示用户组名称
3576296：表示文件内容大小
2007-04-03 11:05：文件最后一次修改时间
ubuntu.ogg：表示文件名称

第五章：操作文件和目录
通配符

```
*     匹配任意多个字符（包括零个或一个）
?     匹配任意一个字符（不包括零个）
[characters]     匹配任意一个属于字符集中的字符
[!characters]    匹配任意一个不是字符集中的字符
[[:class:]]     匹配任意一个属于指定字符类中的字符
[:alnum:]     匹配任意一个字母或数字
[:alpha:]     匹配任意一个字母
[:digit:]     匹配任意一个数字
[:lower:]     匹配任意一个小写字母
[:upper]     匹配任意一个大写字母

举例：
[[:upper:]]*     以大写字母开头的文件
[![:digit:]]*     不以数字开头的文件
*[[:lower:]123]     文件名以小写字母结尾，或以 “1”，“2”，或 “3” 结尾的文件
```


常用命令

```
#建立文件夹，可以一次性建立多个文件夹
mkdir dir1 dir2

#复制文件,如果目的文件没有，会创建这个文件
cp file1 file2
cp -r dir1 dir2 #对于文件夹，要添加参数-r(recursive 递归 循环)

#移动文件
#也可以实现重命名
mv file1 file2  #移动 file1 到 file2。如果 file2 存在，它的内容会被 file1 的内容重写。 如果 file2 不存在，则创
mv file1 file2 dir1 #移动 file1 和 file2 到目录 dir1 中。dir1 必须已经存在
mv dir1 dir2 #    如果目录 dir2 不存在，创建目录 dir2，并且移动目录 dir1 的内容到 目录 dir2 中，同时删除目录 dir

#删除文件
rm file 删除文件
rm -r dir 删除文件夹，添加参数 -r
```

硬链接和软链接

```
#硬链接的局限性
1.不能在不同的分区上建立链接
2.不能对文件夹建立链接

#格式
ln name name_one  #建立硬链接
ln -s name name_one #建立软链接

#查看是否是链接
-  #硬链接的形式一般都是“-”
l   #软链接的形式一般都是“l”
ls -li  #通过文件索引号来判断是否是指向同一文件的链接
```

第六章：使用命令

命令的种类：

1、可执行程序

2、shell脚本

3、内建的shell程序

4、别名

常用命令

```
type command #显示命令的类型
which  command #显示可执行程序的位置，针对的是可执行程序
help command #显示命令文档
man command #显示程序手册
```

使用alias创建命令

```
#首先要给这个命令命名，查看这个命令是否已经有
type command

#对命令进行命名
alias foo='cd /usr; ls; cd -'  #注意结构是 alias foo='string'，其中命令间以分号结束

#取消这个命令
unalias foo

#查看已经被别名的命令
alias
```

第七章：重定向

标准输出

在ubuntu下面建立一个文件，一般情况需要改变文件的权限，文件流的前三个看作标准输入，输出和错误，shell 内部参考它们为文件描述符0，1和2

```
#将标准输出导入到文件中
ls -l /usr/bin > ls-output.txt

#清空文件中的内容
ls-output.txt

#使用上面方法，每次都会清空再来，继续添加的方式为
ls -l /usr/bin >> ls-output.txt

#重定向标准错误
ls -l /bin/usr 2> ls-error.txt

#重定向标准输出和错误到同一个文件中，第二种方式为最新方式
ls -l /bin/usr > ls-output.txt 2>&1 #“2>&1”等同于“>&2”
ls -l /bin/usr &> ls-output.txt
```


常用命令

```
cat demo.txt #用来显示简短的文本内容
cat > demo.txt #输入文本内容，ctrl+d结束之后内容就到了demo.txt

#管道线 ，标准输入可以作为标准输出
command1 | command2  #格式
ls -l /usr/bin | less  #ls -l /usr/bin 可以作为标准输出，该输出作为less的标准输入
ls /bin /usr/bin | sort | less  #也可以作为过滤器进行使用

uniq #忽略重复行
wc #打印行，字和字节数
grep #输出匹配行

head/tail – 输出文件开头部分/结尾部分,默认十行
head -n 5 ls-output.txt
tail -n 5 ls-output.txt

tee #从 Stdin 读取数据，并同时输出到 Stdout 和文件
ls /usr/bin | tee ls.txt | grep zip #将ls /usr/bin结果输出到tee ls.txt文件中，并和grep zip建立管道
```

第八章：从 shell 眼中看世界

这个章节内容主要说的是展开，理解就好，但是文字很难表述清楚

```
echo * #展开当前目录下的所有文件
echo $USER #展开这个值 vagrant
echo $((expression)) #计算表达式的公司 echo $((1+1))
echo Number_{1..2} #花括号展开Number_1，Number_2
```

转义字符 “/”,对于特殊符号，可以进行转义

第九章：键盘高级操作技巧

常见命令


```
clear #清屏
tab #进行补全
```

第十章：权限

输入 ls -l 会显示如下内容：

```
-rwxrwxrwx
```

其中第一个符号表示文件类型，其中

```
- #普通文件
d #目录
l #链接
c #字符设备，比如终端机
b #块设备，比如硬盘
```

另外9个字符，3个为1组，分别属于文件拥有者（用户属于一个用户组）、用户组用户，其他用户，其中

```
0    000    ---
1    001    --x
2    010    -w-
3    011    -wx
4    100    r--
5    101    r-x
6    110    rw-
7    111    rwx
```

其中r表示可读，w表示可写，x表示可执行。

```
chmod 777 demo.rb #给文件所有者，用户组，其他用户rwx权限
#还有一种符号权限，详见书中解释
```

设置新生成文件的默认权限umask

```
#以ubuntu为例
umask  #0002

#设置默认权限为0000
umask 0000
#umask一般是三位，osx是三位，ubuntu是四位，第一位可以不用管，书中详细进行了叙述
000 分别对应 rw-rw-rw-
#如果设置为002,则2数字对应的位置被设置成取消权限位
002 分别对应 rw-rw-r--
```


常用命令：

```
su #以其他用户身份登录
#已vagrant ubuntu14.04为例，下面方法可以切换到root用户
su
输入密码：vagrant

#在root用户下面建立其他用户
useradd -m -s /bin/bash deploy（用户名）
其中：-s /bin/bash 指的是用bash进行登录，-m可以直接在/home下面建立deploy文件夹

passwd depoy #修改密码
su depoy #切换到depoly用户

adduser deploy sudo #添加deploy到sudo组
#或者直接在root用户下面给deploy添加sudo操作
adduser deploy --ingroup sudo


#疑问，在deploy没有分配组的时候，可以随意创建文件，但是有组之后就不行了，必须使用sudo
备注：新增加的用户默认是不在sudo组内的，如果要使用sudominlg

groupadd group1 #在root权限下添加用户组
groupdel group1 #删除用户组
adduser deploy group1 #将用户deploy添加到用户组中

chown user1 demo.txt #更改文件所有者

#查看用户组以及相关用户信息
cat /etc/group
```

第十一章：进程

常用命令

```
kill pid #杀死进程
```

第十二章：shell环境

shell在环境中存储了两种数据，一种是环境变量，另外一种是shell变量

printenv #显示环境变量

set #显示环境变量和shell变量

建立shell环境

启动 bash 程序，并且会读取一系列称为启动文件的配置脚本， 这些文件定义了默认的可供所有用户共享的 shell 环境，也分两种情况，分别是是登录 shell 会话，另一个是非登录 shell 会话。

登录shell会话，包括两种情况，一种是按照用户名和密码shell登录，另外一种是通过执行“ bash --login”形式让shell成为登录会话，下面的代码没有包括进去.bashrc,shell在登录情况下真实的执行次序是[Execution sequence for .bash_profile, .bashrc, .bash_login, .profile and .bash_logout’]

```
#以ubuntu为例伪码
execute /etc/profile
IF ~/.bash_profile exists THEN
    execute ~/.bash_profile
ELSE
    IF ~/.bash_login exist THEN
        execute ~/.bash_login
    ELSE
        IF ~/.profile exist THEN
            execute ~/.profile
        END IF
    END IF
END IF

#当用户退出时
IF ~/.bash_logout exists THEN
    execute ~/.bash_logout
END IF
```

非登录shell会话

```
#以ubuntu为例
execute /etc/bash.bashrc
IF ~/.bashrc exists THEN
    execute ~/.bashrc
END IF
```

总结上面的代码：

全局配置(对所有用户都有效)

```
/etc/profile
/etc/bash.bashrc
```

当前用户配置(仅对当前登录用户有效)

```
~/.bashrc
~/.bash_profile#读取这个文件的时候，里面会读取.bashrc文件
~/.bash_login
~/.profile
```


一个启动文件 .bash_profile及相应解释

```
#如果存在这个文件，执行该文件
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
#添加$HOME/bin这个路径到path中，$HOME
PATH=$PATH:$HOME/bin
#告诉 shell 让这个 shell 的子进程可以使用 PATH 变量的内容
export PATH
```

修改后的文件不会立即生效，关闭shell终端，重启会生效，或者终端下执行下面的命令

```
source .bashrc
```

第十三章 : vi 简介
坚持的原则是：用到的时候再查
常用又记不住的快捷键

```
$ #到行的末尾
/  #光标移动到第一行，可以进行全文检索
:%s/Line/line/g  #进行替换
```

第十五章 : 软件包管理
软件包管理是指系统中一种安装和维护软件的方法.
ubuntu 软件安装方式:

```
sudo apt-get update #更新源
sudo apt-get install git
sudo apt-get remove git
```

第十六章 : 存储媒介
内容略

第十七章 : 网络系统

第二十章：正则表达式

一般形式，查找文件名中包含子字符串“zip”的所有文件


```
ls /usr/bin | grep zip
```

grep 程序以这样的方式来接受选项和参数

```
grep [options] regex [file...]

grep zip demo*.rb #显示文件以及包含的相应字符
grep -l zip demo*.rb #打印包含匹配项的文件名，而不是文本行本身
grep -L zip demo*.rb #打印不包含匹配项的文件名
```

元字符

```
^ $ . [ ] { } - ? * + ( ) | \
```

其他字符是普通字符，比如上面提到的zip,表示自身，可以用反斜杠加元字符表示元字符自身，一般表示命令的时候，需要加上双引号，防止展开。

```
#中间的regex如果涉及元字符，必须用引号表示
grep -h '.zip' dirlist*.txt

#对于.zip需要注意的是需要匹配四个字符，而不是三个，zip是不匹配的
```

锚点

```
grep -h '^zip' dirlist*.txt #zip开头
grep -h 'zip$' dirlist*.txt #zip结尾
grep -h '^zip$' dirlist*.txt #zip单独一行
‘^$’  #会匹配空行
```

中括号表达式

```
grep -h '[bg]zip' dirlist*.txt #匹配b或者g中的任意一个
```

否定

```
grep -h '[^bg]zip' dirlist*.txt #匹配不是b或者g中的任意一个，不匹配zip,而且“^”在[]第一个字符有效
```

限定符

```
? - 匹配零个或一个元素
* - 匹配零个或多个元素
+ - 匹配一个或多个元素
{ } - 匹配特定个数的元素
```


第二十二章：格式化输出

常用命令

```
nl – 添加行号 例如：nl demo.rb
fold – 限制文件列宽 例如：fold –w 12
fmt – 一个简单的文本格式转换器 例如（格式成50个字符宽）：fmt –w 50 fmt-info.txt
```

第二十三章：打印

内容略

第二十四章：编译程序

基本命令：

```
#分析程序建立环境，会建立makefile文件
./configure
#构建程序
make
#安装程序
make install
```

第二十五章：编写第一个 Shell 脚本

第一个脚本内容如下：

```
#!/bin/bash
# This is our first script.
echo 'Hello World!'
```

注意下面的几点：

1、这个文件必须是可执行的，所以需要sudo chomod 755 file

至于说文件可执行和文件被ruby解释器执行是两码事，比如建立如下文件：

```
#demo.rb
puts "this is the demo"

#终端执行demo.rb
ruby demo.rb #this is the demo,获得相应的结果
```

上面的例子是ruby 解释执行demo.rb文件，而不是说这个文件是可执行文件，如果是可执行文件应该是进行如下代码所示

```
#demo.rb

#!/usr/bin/env ruby
puts "this is the demo"
```


终端执行文件，必须明确路径

```
./demo.rb #this is the demo
```

2、shebang的解释
"#!"说明的是： 这个 shebang 被用来告诉操作系统将执行此脚本所用的解释器的名字。每个 shell 脚本都应该把这一文本行 作为它的第一行。
比如：

```
#!/bin/bash
#!/usr/bin/env ruby
```

对上面两个名字的解释，一般写shebang的解释器都是些绝对路径，比如#!/bin/bash,这是bash解释器的绝对路径，但是有时候不知道解释器安装在哪里，比如说ruby解释器，所以可以参考第二种方法，#!/usr/bin/env ruby这种方法会从PATH中寻找ruby解释器的位置。

3、解释执行的必须明确路径
在上面的文件中执行demo.rb文件的方法是如下：

```
./demo.rb  #其中“./”表示的是当前的文件夹，结果为 this is the demo

demo.rb
#但是如果直接执行demo.rb,会出现现在结果 bash: hello_world: command not found
```

如果只是执行demo.rb文件，那么会从PATH中寻找可执行程序，发现并没有找到,得到上面的结果。解决的方式是在.bashrc文件中输入如下代码：

```
export PATH=~/workshop/file:"$PATH"
```

然后执行下面的文件，使.bashrc生效

```
source .bashrc
. .bashrc  #source 和“.”同样的意义
```

第二十六章：启动一个项目
变量的定义：

1、变量名可由字母数字字符（字母和数字）和下划线字符组成。
2、变量名的第一个字符必须是一个字母或一个下划线。
3、变量名中不允许出现空格和标点符号。

常量和变量定义举例：

```
CURRENT_TIME=$(date +"%x %r %Z")
a=z
b="a string"            # Embedded spaces must be within quotes.
c="a string and $b"     # Other expansions such as variables can be
                        # expanded into the assignment.
d=$(ls -l foo.txt)      # Results of a command.
e=$((5 * 7))            # Arithmetic expansion.
f="\t\ta string\n"      # Escape sequences such as tabs and newlines.
#常量名也可以被{}包围着
filename="myfile"
mv $filename ${filename}1
#上面实现了变量命名，如果是写mv $filename $filename1,后者没有被定义过，为空值，会出错
```

使用here document

```
cat << _EOF_
  this is the demo
_EOF_
```

here documents 中的单引号和双引号会失去它们在 shell 中的特殊含义

```
foo="some text"
cat << _EOF_
  $foo   #显示结果为：some text
  "$foo" #显示结果为："some text"
  '$foo' #显示结果为：'some text'
   \$foo #显示结果为：$foo
_EOF_
```

cmd << text从命令行读取输入，直到一个与text相同的行结束。除非使用引号把输入括起来，此模式将对输入内容进行shell变量替换。如果使用<<- ，则会忽略接下来输入行首的tab，结束行也可以是一堆tab再加上一个与text相同的内容。

第二十七章：自顶向下设计
注意：return作用是从函数中返回，而exit作用是结束程序
shell函数格式：

```
#第一种形式
function name {
    commands
    #return的作用是从函数中返回
    return
}

#第二种形式
name () {
    commands
    return
}
```


脚本中 shell 函数定义必须出现在函数调用之前，引用形式如下：

```
#!/bin/bash

report_uptime () { return}

cat << _EOF_
  $(report_uptime)
_EOF_
```

局部变量，只在函数的局部范围内有效

```
funct_1 () {
  #必须使用如下的local格式声明局部变量
  local foo
  #局部变量赋值
  foo=1
}
```

第二十八章：流程控制：if 分支结构
参考格式：

```
#!/bin/bash
#第一个等号是赋值
x=5
#第二个等号是比较；if对应用fi封闭；使用[]并且加分号“；”表示结束
if [ $x = 5 ]; then
    echo "x equals 5."
else
    echo "x does not equal 5."
fi
```

退出状态

```
#执行命令的时候，会给系统发送一个值，这个值就是退出状态，在0-255之间，0表示成功，其他值表失败
[me@linuxbox ~]$ ls -d /usr/bin
/usr/bin
[me@linuxbox ~]$ echo $? #0
[me@linuxbox ~]$ ls -d /bin/usr
ls: cannot access /bin/usr: No such file or directory
[me@linuxbox ~]$ echo $? #2

#true表示成功执行，false表示失败
[me@linuxbox~]$ true
[me@linuxbox~]$ echo $? #0
[me@linuxbox~]$ false
[me@linuxbox~]$ echo $? #1

#如果 if 之后跟随一系列命令，则将计算列表中的最后一个命令
[me@linuxbox ~]$ if false; true; then echo "It's true."; fi
It's true.
[me@linuxbox ~]$ if true; false; then echo "It's true."; fi
[me@linuxbox ~]$ $?
3
```


if后面的格式

```
#即是上面两种也都是等价的，test expression和[expression]是等价的。

#之前的例子为
if [ $x=5 ];
#等价的例子为
if test $x= 5;
```

这里的 expression 是一个表达式，其执行结果是 true 或者是 false。当表达式为真时，这个 test 命令返回一个零 退出状态，当表达式为假时，test 命令退出状态为1

"[ ]"的加强版"[[ ]]"

```
#"[[ ]]"是高级形式，相比于前者增加下面两种表达式

string1 =~ regex #例子如下
INT=-5
if [[ "$INT" =~ ^-?[0-9]+$ ]]; then

“==”操作符支持类型匹配
$FILE == foo.*
```

“(( )) ”
被用来执行算术真测试。如果算术计算的结果是非零值，则一个算术真测试值为真。

```
[me@linuxbox ~]$ if ((1)); then echo "It is true."; fi
It is true.
[me@linuxbox ~]$ if ((0)); then echo "It is true."; fi
[me@linuxbox ~]$
```


[]中的内容需要进行转义

因为 test 使用的所有的表达式和操作符都被 shell 看作是命令参数（不像 [[ ]] 和 (( ))）， 对于 bash 有特殊含义的字符，比如说 <，>，(，和)，必须引起来或者是转义

```
[ ! \( $INT -ge $MIN_VAL -a $INT -le $MAX_VAL \) ]
```

逻辑表达式的几种表达方式

```
操作符    测试      [[ ]] and (( ))
AND       -a           &&
OR        -o           ||
NOT       !            !
```

分支的说明：

```
command1 && command2
#先执行 command1，如果并且只有如果 command1 执行成功后， 才会执行 command2。

command1 || command2
对于 || 操作符，先执行 command1，如果并且只有如果 command1 执行失败后， 才会执行 command2
```

第二十九章：读取键盘输入

here字符串

here string可以看成是here document的一种定制形式. 除了COMMAND <<<$WORD, 就什么都没有了，$WORD将被扩展并且被送入COMMAND的stdin中。

最后

变量：

$PATH

$HOME:当前用户的主目录，另外一种表示法是“~”