Vim编辑器与Shell命令脚本

1 Vim编辑器

2 服务器基础环境配置

3 Yum软件仓库配置

4 编写Shell脚本

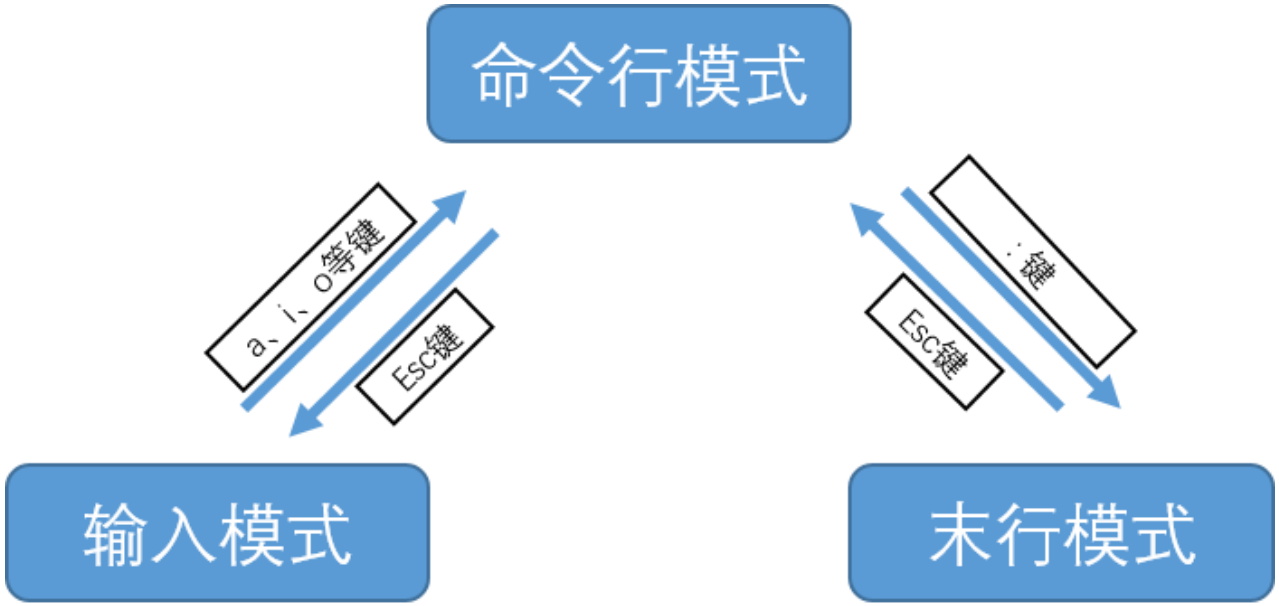
vim编辑器

Vim文本编辑器

Vim之所以能得到广大厂商与用户的认可，原因在于Vim编辑器中设置了三种模式—命令模式、末行模式和编辑模式，每种模式分别又支持多种不同的命令快捷键，这大大提高了工作效率，而且用户在习惯之后也会觉得相当顺手。

三种模式的操作区别以及模式之间的切换方法

- 命令模式：控制光标移动，可对文本进行复制、粘贴、删除和查找等工作。
- 输入模式：正常的文本录入。
- 末行模式：保存或退出文档，以及设置编辑环境。



Vim中常用的命令

命令	作用
dd	删除（剪切）光标所在整行
5dd	删除（剪切）从光标处开始的5行
yy	复制光标所在整行
5yy	复制从光标处开始的5行
n	显示搜索命令定位到的下一个字符串
N	显示搜索命令定位到的上一个字符串
u	撤销上一步的操作
p	将之前删除（dd）或复制（yy）过的数据粘贴到光标后面

末行模式中可用的命令

命令	作用
:w	保存
:q	退出
:q!	强制退出（放弃对文档的修改内容）
:wq!	强制保存退出
:set nu	显示行号
:set nonu	不显示行号
:命令	执行该命令
:s/one/two	将当前光标所在行的第一个one替换成two
:s/one/two/g	将当前光标所在行的所有one替换成two
:%s/one/two/g	将全文中的所有one替换成two
?字符串	在文本中从下至上搜索该字符串
/字符串	在文本中从上至下搜索该字符串

服务器基础环境配置

配置主机名

第1步：使用Vim编辑器修改“/etc/hostname”主机名称文件。

第2步：把原始主机名称删除后追加“eagleslab.com”。注意，使用Vim编辑器修改主机名称文件后，要在末行模式下执行:wq!命令才能保存并退出文档。

第3步：保存并退出文档，然后使用hostname命令检查是否修改成功。

配置网卡信息

第1步：首先切换到/etc/sysconfig/network-scripts目录中（存放着网卡的配置文件）。

第2步：使用Vim编辑器修改网卡文件ifcfg-eno16777736

- 设备类型：TYPE=Ethernet
- 地址分配模式：BOOTPROTO=static
- 网卡名称：NAME=eno16777736
- 是否启动：ONBOOT=yes
- IP地址：IPADDR=192.168.179.10
- 子网掩码：NETMASK=255.255.255.0
- 网关地址：GATEWAY=192.168.179.1
- DNS地址：DNS1=1114.114.114.114

配置网卡信息

```
[root@localhost ~]# cat /etc/sysconfig/network-scripts/ifcfg-ens33
TYPE="Ethernet"
PROXY_METHOD="none"
BROWSER_ONLY="no"
BOOTPROTO="none"
DEFROUTE="yes"
IPV4_FAILURE_FATAL="no"
IPV6INIT="yes"
IPV6_AUTOCONF="yes"
IPV6_DEFROUTE="yes"
IPV6_FAILURE_FATAL="no"
IPV6_ADDR_GEN_MODE="stable-privacy"
NAME="ens33"
UUID="fcb48f5a-7334-443f-82c3-fa58ffbfe1c6"
DEVICE="ens33"
ONBOOT="yes"
IPADDR=192.168.9.101
NETMASK=255.255.255.0
GATEWAY=192.168.9.2
DNS1=114.114.114.114
[root@localhost ~]#
```

配置网卡信息

执行重启网卡设备的命令（在正常情况下不会有提示信息），然后通过ping命令测试网络能否联通。由于在Linux系统中ping命令不会自动终止，因此需要手动按下Ctrl-c键来强行结束进程。

```
[root@eagleslab network-scripts]# systemctl restart network
[root@localhost network-scripts]# ping baidu.com
PING baidu.com (220.181.57.216) 56(84) bytes of data.
64 bytes from 220.181.57.216: icmp_seq=1 ttl=128 time=32.5 ms
64 bytes from 220.181.57.216: icmp_seq=2 ttl=128 time=32.4 ms
^C
--- baidu.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1009ms
rtt min/avg/max/mdev = 32.402/32.496/32.590/0.094 ms
[root@localhost network-scripts]#
```

配置Yum软件仓库

配置Yum软件仓库

第1步：进入到/etc/yum.repos.d/目录中（因为该目录存放着Yum软件仓库的配置文件）。

第2步：使用Vim编辑器创建一个名为eagle.repo的新配置文件（文件名称可随意，但后缀必须为.repo），逐项写入下面的配置参数并保存退出（不要写后面的中文注释）。

- [eagle-mirror]：Yum软件仓库唯一标识符，避免与其他仓库冲突。
- name=eagle：Yum软件仓库的名称描述，易于识别仓库用处。
- baseurl=http://dc.eagleslab.com:8889/Packages：提供的方式包括FTP（ftp://..）、HTTP（http://..）、本地（file:///..）。
- enabled=1：设置此源是否可用；1为可用，0为禁用。
- gpgcheck=0：设置此源是否校验文件；1为校验，0为不校验。
- # gpgkey=file:///media/cdrom/RPM-GPG-KEY-redhat-release：若上面参数开启校验，那么请指定公钥文件地址。

配置Yum软件仓库

第3步：使用“yum makecache”命令检查Yum软件仓库是否已经可用。

```
[root@localhost ~]# vi /etc/yum.repos.d/eagle.repo
[root@localhost ~]# yum makecache
已加载插件：fastestmirror
base                                                | 3.6 kB     00:00
extras                                              | 3.4 kB     00:00
updates                                             | 3.4 kB     00:00
(1/3): extras/7/x86_64/prestodelta                    |  68 kB   00:00
(2/3): updates/7/x86_64/other_db                      | 381 kB   00:00
(3/3): base/7/x86_64/filelists_db                     | 6.9 MB   00:01
Loading mirror speeds from cached hostfile
 * base: mirrors.shu.edu.cn
 * extras: mirrors.shu.edu.cn
 * updates: mirrors.tuna.tsinghua.edu.cn
元数据缓存已建立
```

编写Shell脚本

编写Shell脚本

可以将Shell终端解释器当作人与计算机硬件之间的“翻译官”，它作为用户与Linux系统内部的通信媒介，除了能够支持各种变量与参数外，还提供了诸如循环、分支等高级编程语言才有的控制结构特性。

- 交互式（Interactive）：用户每输入一条命令就立即执行。
- 批处理（Batch）：由用户事先编写好一个完整的Shell脚本，Shell会一次性执行脚本中诸多的命令。

查看SHELL变量可以发现当前系统已经默认使用Bash作为命令行终端解释器了

```
[root@localhost ~]# echo $SHELL
/bin/bash
[root@localhost ~]#
```

编写简单的脚本

使用Vim编辑器把Linux命令按照顺序依次写入到一个文件中，这就是一个简单的脚本了

```
[root@localhost ~]# vim example.sh
#!/bin/bash
#For Example
pwd
ls -al
```

编写简单的脚本

Shell脚本文件的名称可以任意，但为了避免被误以为是普通文件，建议将.sh后缀加上

```
[root@localhost ~]# chmod u+x example.sh
[root@localhost ~]# ./example.sh
/root
总用量 44
dr-xr-x---.  2 root root 4096 7月  21 16:34 .
dr-xr-xr-x. 17 root root 4096 5月  18 2016 ..
-rw-------.  1 root root  929 5月  18 2016 anaconda-ks.cfg
-rw-------.  1 root root  660 7月  19 03:51 .bash_history
-rw-r--r--.  1 root root   18 12月 29 2013 .bash_logout
-rw-r--r--.  1 root root  176 12月 29 2013 .bash_profile
-rw-r--r--.  1 root root  176 12月 29 2013 .bashrc
-rw-r--r--.  1 root root  100 12月 29 2013 .cshrc
-rwxr--r--.  1 root root   36 7月  21 16:34 example.sh
-rw-r--r--.  1 root root  129 12月 29 2013 .tcshrc
-rw-------.  1 root root  636 7月  21 16:34 .viminfo
[root@localhost ~]#
```

接收用户的参数

```
[root@localhost ~]# vim example.sh
#!/bin/bash
echo "当前脚本名称为$0"
echo "总共有$#个参数，分别是#*"
echo "第1个参数为$1,第5个参数为$5。"
[root@localhost ~]# sh example.sh one two three four five six
当前脚本名称为example.sh
总共有6个参数，分别是#*
第1个参数为one,第5个参数为five。
```

判断用户的参数

测试语句格式:[条件表达式]

条件表达式前后应有一个空格，若条件成立则返回数字0，否则便返回其他随机数值。

按照测试对象来划分，条件测试语句可以分为4种：

- 文件测试语句；
- 逻辑测试语句；
- 整数值比较语句；
- 字符串比较语句。

判断用户的参数

文件测试所用的参数：

运算符	作用
-d	测试文件是否为目录类型
-b	判断文件是否为块特殊文件
-e	测试文件是否存在
-f	判断是否为一般文件
-r	测试当前用户是否有权限读取
-w	测试当前用户是否有权限写入
-x	测试当前用户是否有权限执行

判断用户的参数

```
[root@localhost ~]# [ -b /etc/fstab ]
[root@localhost ~]# echo $?
1
[root@localhost ~]# [ -f /etc/fstab ]
[root@localhost ~]# echo $?
0
[root@localhost ~]# [ -e /etc/fstab ] && echo "Exist"
Exist
[root@localhost ~]# [ -e /etc/fstab1 ] && echo "Exist"
    #如果&&前面的命令执行成功了，就会执行&&后面的命令
[root@localhost ~]# [ -e /etc/fstab1 ] || echo "Not Exist"
Not Exist
    #&&是逻辑"与"，||是逻辑"或"
[root@localhost ~]# [ ! -e /etc/fstab1 ] && echo "Not Exist"
Not Exist
    #前面加上!等于"非"，会将结果翻转
[root@localhost ~]# [ ! $USER = root ] && echo "user" || echo "root"
root
```

可用的整数比较运算符

运算符	作用
-eq	是否等于
-ne	是否不等于
-gt	是否大于
-lt	是否小于
-le	是否等于或小于
-ge	是否大于或等于

```
[root@localhost ~]# [ 10 -gt 10 ] || echo "错误"
错误
[root@localhost ~]# [ 10 -eq 10 ] && echo "正确"
正确
```

可用的整数比较运算符

```
[root@localhost ~]# free -m
              total        used        free      shared  buff/cache   available
Mem:           1823         134        1183           8         505        1486
Swap:          2047           0        2047
[root@localhost ~]# free -m | grep Mem:
Mem:           1823         134        1183           8
505        1486
[root@localhost ~]# free -m | grep Mem: | awk '{print $4}'
1183
[root@localhost ~]# FreeMem=`free -m | grep Mem: | awk '{print
$4}'`
[root@localhost ~]# echo $FreeMem
1182
[root@localhost ~]# [ $FreeMem -lt 2048 ] && echo "内存不足"
内存不足
```

常见的字符串比较运算符

运算符	作用
=	比较字符串内容是否相同
!=	比较字符串内容是否不同
-z	判断字符串内容是否为空

```
[root@localhost ~]# echo $LANG
zh_CN.UTF8
[root@localhost ~]# [ $LANG != "en.US.UTF-8" ] && echo "非英语"
非英语
```

流程控制语句——if条件测试语句

if条件语句的单分支结构由if、then、fi关键词组成，而且只在条件成立后才执行预设的命令，相当于口语的“如果……那么……”

```
if 目录不存在
    then 创建该目录
fi
```

```
[root@localhost ~]# vi example.sh
#!/bin/bash
DIR="/root/test"
if [ ! -e $DIR ]
then
        mkdir -p $DIR
fi
[root@localhost ~]# ./example.sh
[root@localhost ~]# ls
anaconda-ks.cfg  example.sh  test
[root@localhost ~]#
```

流程控制语句——if条件测试语句

if条件语句的双分支结构由if、then、else、fi关键词组成，它进行一次条件匹配判断，如果与条件匹配，则去执行相应的预设命令；反之则去执行不匹配时的预设命令，相当于口语的“如果……那么……或者……那么……”

```
if 能够ping通
    then 提示服务器正常工作
    else 报警服务器出现问题
fi
```

流程控制语句——if条件测试语句

```
[root@localhost ~]# vi example.sh
#!/bin/bash
ping -c 3 -i 0.2 -W 3 $1 &> /dev/null
if [ $? -eq 0 ]
then
echo "Host $1 is On-line."
else
echo "Host $1 is Off-line."
fi
[root@localhost ~]# ./example.sh 192.168.0.1
Host 192.168.0.1 is On-line.
[root@localhost ~]# ./example.sh 6.6.6.6
Host 6.6.6.6 is Off-line.
[root@localhost ~]#
```

流程控制语句——if条件测试语句

if条件语句的多分支结构由if、then、else、elif、fi关键词组成，它进行多次条件匹配判断，这多次判断中的任何一项在匹配成功后都会执行相应的预设命令，相当于口语的“如果……那么……如果……那么……”

```
if 分数为85~100之间
    then 判为优秀
elif 分数为70~84之间
    then 判为合格
else
    判为不合格
fi
```

流程控制语句——if条件测试语句

在Linux系统中，read是用来读取用户输入信息的命令，能够把接收到的用户输入信息赋值给后面的指定变量，-p参数用于向用户显示一定的提示信息。

```
[root@localhost ~]# vi example.sh
#!/bin/bash
read -p "Enter your score(0-100): " GRADE
if [ $GRADE -gt 100 ] || [ $GRADE -lt 0 ] ; then
    echo "Error"
elif [ $GRADE -ge 85 ] && [ $GRADE -le 100 ] ; then
    echo "$GRADE is Excellent"
elif [ $GRADE -ge 70 ] && [ $GRADE -le 84 ] ; then
    echo "$GRADE is Pass"
else
    echo "$GRADE is Fail"
fi
```

流程控制语句——for条件循环语句

for循环语句允许脚本一次性读取多个信息，然后逐一对信息进行操作处理，当要处理的数据有范围时，使用for循环语句再适合不过了

```
for 用户名in列表文件
do
    创建用户并设置密码
done
```

流程控制语句——for条件循环语句

```
[root@localhost ~]# vi example.sh
#!/bin/bash
read -p "Enter The Users Password :" PASSWD
for UNAME in `cat userlist.txt`
do
id $UNAME &> /dev/null
if [ $? -eq 0 ]
then
echo "Already exists"
else
useradd $UNAME &> /dev/null
echo "$PASSWD" | passwd --stdin $UNAME &> /dev/null
if [ $? -eq 0 ]
then
echo "$UNAME , Create success"
else
echo "$UNAME , Create failure"
fi
fi
done
```

流程控制语句——wheel条件循环语句

while条件循环语句是一种让脚本根据某些条件来重复执行命令的语句，它的循环结构往往在执行前并不确定最终执行的次数，完全不同于for循环语句中有目标、有范围的使用场景。while循环语句通过判断条件测试的真假来决定是否继续执行命令，若条件为真就继续执行，为假就结束循环。

```
while 未猜中正确价格
do
    反复猜测商品价格
done
```

流程控制语句——wheel条件循环语句

```
[root@localhost ~]# vi example.sh
#!/bin/bash
PRICE=$(expr $RANDOM % 1000)
TIMES=0
echo "商品实际价格为0-999之间，猜猜看是多少？"
while true
do
read -p "请输入您猜测的价格数目：  " INT
let TIMES++
if [ $INT -eq $PRICE ] ; then
echo "恭喜您答对了，实际价格是 $PRICE"
echo "您总共猜 g $TIMES 次"
exit 0
elif [ $INT -gt $PRICE ] ; then
echo "太高了！"
else
echo "太低了！"
fi
done
```

流程控制语句——case条件测试语句

case语句是在多个范围内匹配数据，若匹配成功则执行相关命令并结束整个条件测试；而如果数据不在所列出的范围内，则会去执行星号（*）中所定义的默认命令。

```
case 输入的字符 in
[a-z]|[A-Z])
    提示为字母
    ;;
[0-9])
    提示为数字
    ;;
*)
    提示为特殊字符
esac
```

流程控制语句——case条件测试语句

```
[root@localhost ~]# vi example.sh
#!/bin/bash
read -p "请输入一个字符，并按Enter键确认：  " KEY
case "$KEY" in
[a-z]|[A-Z])
echo "您输入的是 字母"
;;
[0-9])
echo "您输入的是 数字"
;;
*)
echo "您输入的是 空格、功能键或其他控制字符"
esac
```