

第十二章 shell编程

查看系统一共包含哪些shell

```
cat /etc/shells
```

更改用户默认的shell

```
chsh -s
```

```
usermod -s
```

```
vi /etc/passwd
```

用户无法登录系统，但是用户可以改密码，且管理员无法查看密码，怎么做

```
which passwd
```

```
echo 'which passwd' >> /etc/shells
```

```
useradd -s /usr/bin/passwd
```

[linux请问如何让root用户也不能随意更改root用户密码](#)

```
cd /etc
```

```
grub-md5-crypt
```

生成一个用MD5加密的密码，复制这个MD5密码，然后vi/etc/grub.conf添加密码栏

```
password --md5 MD5密码(刚才生成的)
```

保存退出，重启计算机。只有拥有这个密码才可以从SHELL下修改root密码，没有这个密码，是无法通过SHELL下修改root密码的

- 设置记录历史命令的条数
 - #history 查看历史命令
 - 记录条数设置: /etc/profile
 - 修改 HISTSIZE 参数（默认为1000条）
- 调用历史命令
 - !n: 执行历史记录中的第n条命令
 - !str: 执行历史记录中以“str”开头的命令

上下方向键

```
/etc/profile
```

```
.bashrc_profile
```

bash在用户登陆时从四个文件中读取环境设定：

- 全局设置文件：
 - /etc/profile
 - /etc/bashrc
- 用户设置文件：
 - ~/.bashrc
 - ~/.bash_profile

全局设置作用于所有用户

用户设置作用于当前用户

使系统环境设置文件能够即时生效，而无需重新登陆

`source .bash`

定义新变量或赋值

变量名=变量值

`#export 变量名` 将变量变为环境变量

`#env` 查看系统环境变量

`#echo $变量名` 显示变量值

`#unset 变量名` 取消变量（临时）

`echo $PATH>> .bashrc`

`PATH=9999: XXXXXXXXXXXX` `PATH` 我们只需要添加到程序所在的目录即可

`export PATH`

`source .bash`

• 有三种执行shell脚本的方式：

`sh /路径/脚本名` （无需X权限）

`/路径/脚本名` （需X权限）

`./路径/脚本名` （需X权限）

- 使用read将数据读入
 - read username
 - 从标准输入读取数据来为username这个变量赋值
- Example (CHAP11_01.sh) :

```
#!/bin/bash
echo -e "Please input Your Name:"
read yourname
echo -e "Your Name is: $yourname"
```

脚本编程:

read 在脚本执行中, 给后面指定的变量赋值

```
#mkdir test && touch ./test/file1
#mkdir test && touch ./test/file2

#mkdir test2 || touch ./test2/file1
#mkdir test2 || touch ./test2/file2
```

• 逻辑运算符

- **Shell**命令行支持在同一行的两条命令之间使用逻辑运算符

• && (逻辑与)

- 当前一条命令执行成功时再执行后一条指令

• || (逻辑或)

- 当前一条命令执行失败时再执行后一条指令

• ! (逻辑否)

- 当指定的条件不成立时, 返回结果为真

./configure && make && make install

条件测试

• 测试文件状态

- 格式: [操作符 文件或目录]

• 常用的测试操作符

• -d: 测试是否为目录 (Directory)

• -e: 测试目录或文件是否存在 (Exist)

- f: 测试是否为文件 (File)
- r: 测试当前用户是否有权限读取 (Read)
- w: 测试当前用户是否有权限写入 (Write)
- x: 测试当前用户是否可执行 (Excute) 该文件
- L: 测试是否为符号连接 (Link) 文件

•整数值比较

- 格式: [整数1 操作符 整数2]

•常用的测试操作符

- eq: 等于 (Equal)
- ne: 不等于 (Not Equal)
- gt: 大于 (Greater Than)
- lt: 小于 (Lesser Than)
- le: 小于或等于 (Lesser or Equal)
- ge: 大于或等于 (Greater or Equal)

•字符串比较

- 格式: [字符串1 = 字符串2]

[字符串1 != 字符串2]

[-z 字符串]

•常用的测试操作符

- = 字符串内容相同
- != 字符串内容不同, !号表示相反的意思
- z 字符串内容为空

•使用带命令行参数的Shell脚本

- #command [option1] [option2].....

• 引用Shell脚本命令行参数

- \$0 命令名本身
- \$1 第一个参数(option1)
- \$2 第二个参数(option2)
-
- \$9 第九个参数(option9)

•\$* 程序的所有参数

•\$# 程序的参数个数

\$? 上一个指令的返回值（无误0 有误1）

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

```
RATE=`df -hT | grep "/boot" | awk '{print $6}' | cut -d "%" -f1`
```

```
if [ $RATE -gt 80 ]
```

```
then
```

```
    echo "Warning,DISK is full!"
```

```
fi
```

```
#!/bin/bash
X=`df -hT | grep "hda1" | awk '{print $6}' | cut -d "%" -f1`
if [ $X -gt 25 ]
then
    echo "Warning, The HDA1 DISK is Full! $X%" > ./temp.txt
    mail -s "DISK FULL" root < ./temp.txt
    rm -rf ./temp.txt
fi
~
~
"test.sh" 8L, 221C
```

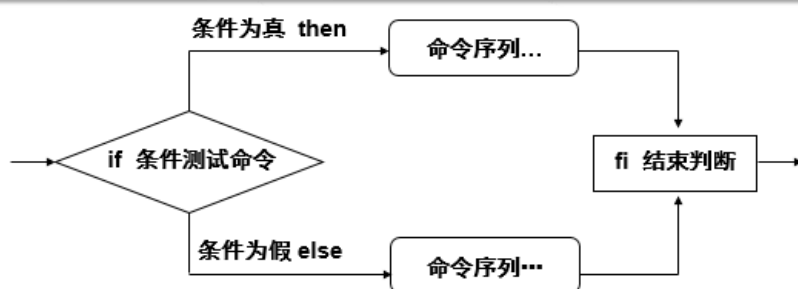
单分支语句

• if条件语句（双分支）

– 当“条件成立”、“条件不成立”时执行不同操作

– 语法格式：

```
if 条件判断语句
then 动作
else 动作
fi
```



双分支语句

• if条件语句（双分支）示例（一）：

– Example（CHAP11_05.sh）：

```
#!/bin/sh
read -p "Enter a password: " pwd_entered
if ["$pwd_entered" = "password"]; then
    echo Password is correct
else
    echo Password is incorrect
fi
```

多分支语句

- if条件语句（多分支）

- 相当于if语句嵌套，针对多个条件执行不同操作
- 语法格式：

```
if 条件判断语句
    then    动作
elif 条件判断语句
    then    动作
elif ...
    else    动作
fi
```

y n

如果我们需要用户输入是或否

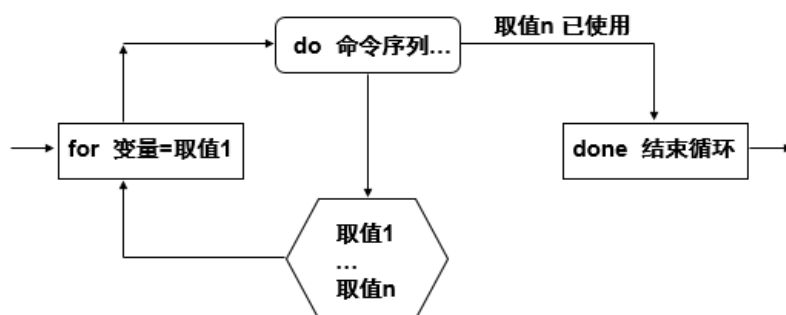
1. y Y YES
2. n N NO
3. sadasd
4. 什么也不输入

for循环语句

- for循环语句

- 根据变量的不同取值，重复执行一组命令操作。
- 语法格式：

```
for 变量名 in 取值列表
do 命令序列
done
```



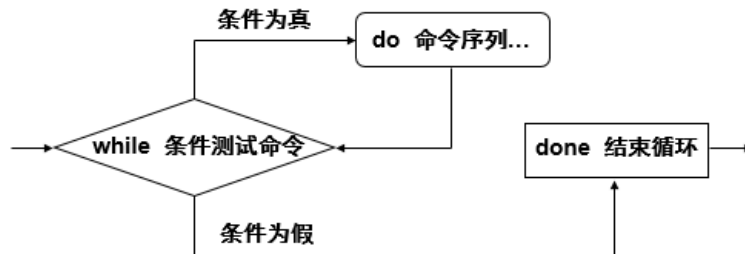
最多借位30位

每个子网是4个ip

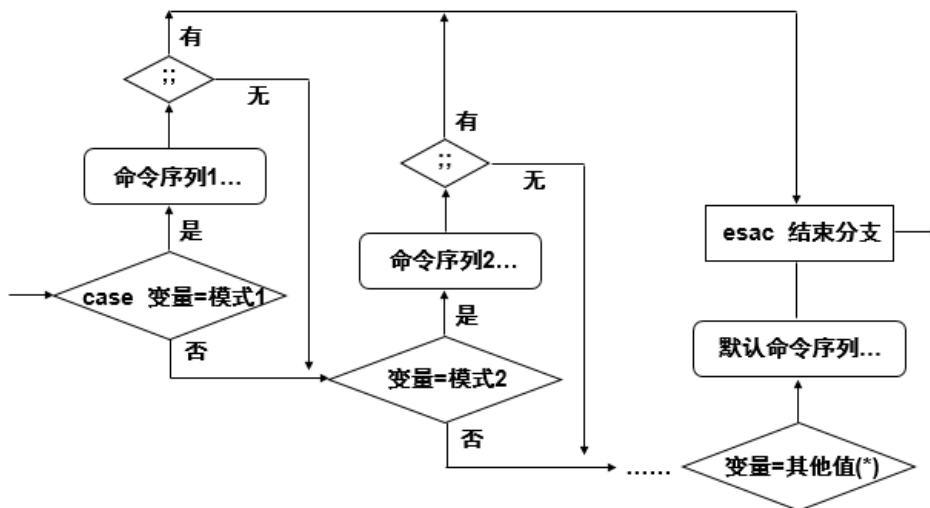
while循环语句

- 重复测试指定的条件，只要条件成立则反复执行对应的命令操作。
- 语法格式：

```
while 命令或表达式
do
    命令列表
done
```



case多重分支语句



• case多重分支语句示例（一）：

- 编写脚本文件 mydb.sh，用于控制系统服务mysqld
- 当执行 ./mydb.sh start 时，启动mysqld服务
- 当执行 ./mydb.sh stop 时，关闭mysqld服务
- 如果输入其他脚本参数，则显示帮助信息

```
#!/bin/bash
case $1 in
    start)
        echo "Start MySQL service."
        ..
        ;;
    stop)
        echo "Stop MySQL service."
        ..
        ;;
    *)
        echo "Usage: $0 start|stop"
        ..
        ;;
esac
```

case多重分支语句示例（二）：

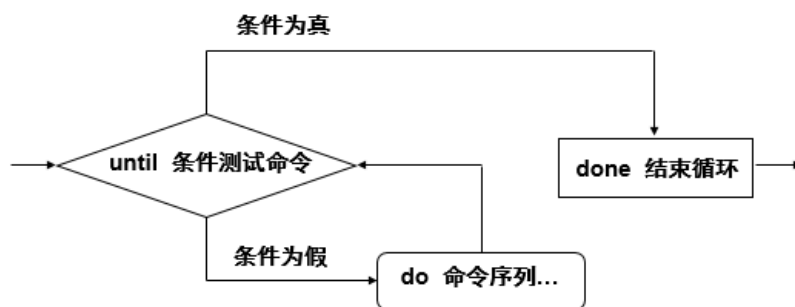
- 提示用户从键盘输入一个字符，判断该字符是否为字母、数字或者其它字符，并输出相应的提示信息。

```
#!/bin/bash
read -p "Press some key, then press Return:" KEY
case "$KEY" in
  [a-z][A-Z])
    echo "It's a letter."
    ;;
  [0-9])
    echo "It's a digit."
    ;;
  *)
    echo "It's function keys、Spacebar or other keys. "
esac
```

until循环语句

- until语句根据条件执行重复操作。
- 语法格式：

```
until 条件测试命令
do
  命令序列
done
```



• until循环语句示例：

- Example (CHAP11_09.sh) :

```
#!/bin/sh
read -p "Enter a password:" pwd_entered
clear
until [ "$pwd_entered" = "123456" ]
do
    echo -e "Sorry, Try again!"
    read -p "Enter a password:" pwd_entered
done
echo -e "Correct password entered!"
```


用于迁移位置变量，将 \$1~\$9 依次向左传递

- 例如，若当前脚本程序获得的位置变量如下：
 - \$1=file1、\$2=file2、\$3=file3、\$4=file4
- 则执行一次shift命令后，各位置变量为：
 - \$1=file2、\$2=file3、\$3=file4
- 再次执行shift命令后，各位置变量为：
 - \$1=file3、\$2=file4

• shift迁移语句示例：

- 通过命令行参数传递多个整数值，并计算总和。

```
#!/bin/bash
Result=0
while [ $# -gt 0 ]
do
    Result=`expr $Result + $1`
    shift
done
echo "The sum is : $Result"
```

```
[root@localhost ~]# ./sumer.sh 12 34 56
The sum is : 102
```

break语句

- 在for、while、until等循环语句中，用于跳出当前所在的循环体，执行循环体后的语句。

```
while 命令
do
    .....
    .....
    break
    .....
done
.....
```




跳出循环

• continue语句

- 在for、while、until等循环语句中，用于跳过循环体内余下的语句，重新判断条件以便执行下一次循环。

```
while
do
    .....
    .....
    continue
    .....
done
.....
```



继续下次循环

• Shell函数概述

- 在编写Shell脚本程序时，将一些需要重复使用的命令操作，定义为公共使用的语句块，即可称为函数
- 合理使用Shell函数，可以使脚本内容更加简洁，增强程序的易读性，提高执行效率

```
function 函数名 {  
    命令序列  
}
```

• Shell函数应用示例：

- 在脚本中定义一个加法函数，用于计算2个整数的和。
- 调用该函数计算（12+34）、（56+789）的和。

```
#!/bin/bash  
add() {  
    echo `expr $1 + $2`  
}  
add 12 34  
add 56 789
```

```
[root@localhost ~]# sh addfun.sh  
46  
845
```