



四川大学
国家示范性软件学院
SCU Software college.



Operating System

Lab02 Introduction



四川大学

国家示范性软件学院

SCU Software College



目录

- 1.VMWare 平台交互
- 2.VMWare 其他说明
- 3.Shell编程简介



1.VMWare 平台交互

□ 网络配置

■ Bridge

■ NAT

■ Host-only

□ 与主机交互

■ U盘

■ VMWare Tools

■ SSH

■ samba





虚拟网络编辑器

名称	类型	外部连接	主机连接	DHCP	子网地址
VMnet1	仅主机...	-	已连接	已启用	192.168.13.0
VMnet8	NAT 模式	NAT 模式	已连接	已启用	192.168.150.0

添加网络(E)... 移除网络(O) 重命名网络(W)...

VMnet 信息

☐ 桥接模式(将虚拟机直接连接到外部网络)(B)
 已桥接至(G): 自动设置(U)...

☐ NAT 模式(与虚拟机共享主机的 IP 地址)(N) NAT 设置(S)...

☒ 仅主机模式(在专用网络内连接虚拟机)(H)

☒ 将主机虚拟适配器连接到此网络(V)
 主机虚拟适配器名称: VMware 网络适配器 VMnet1

☒ 使用本地 DHCP 服务将 IP 地址分配给虚拟机(D) DHCP 设置(P)...


子网 IP (I): 子网掩码(M):

△ 需要具备管理员特权才能修改网络配置。 更改设置(C)


文件(F) 编辑(E) 查看(V) 工具(T) 高级(N) 帮助(H)

组织 ▾ 连接到 禁用此网络设备 诊断这个连接 重命名此连接 查看此连接的状态 更改此连接的状态


 VMware Network Adapter
VMnet1
已启用


 VMware Network Adapter
VMnet8
已启用


 本地连接
网络电缆被拔出
Realtek PCIe GBE Family Contr...

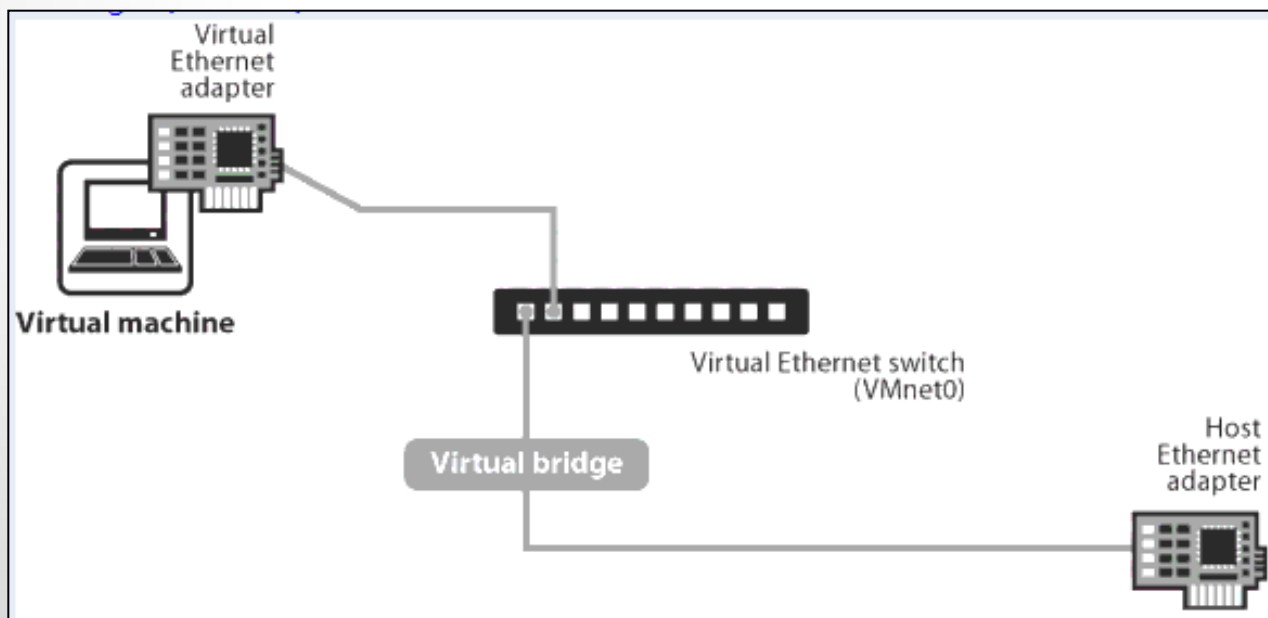

 无线网络连接 4
www.123456
Realtek 8188GU Wireless LAN ...



1.VMWare 平台交互

□ 网络配置--bridge

- 虚拟机和主机IP同网段，但相互独立
- 虚拟机加入主机所在的局域网
- 虚拟机<->主机， 虚拟机<->互联网

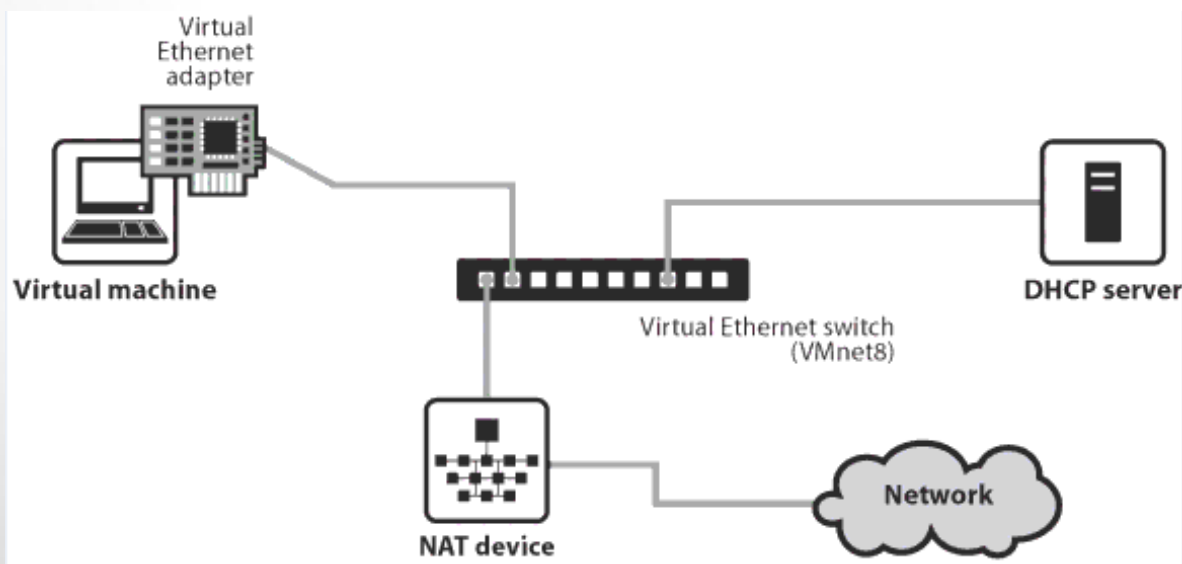




1.VMWare 平台交互

□ 网络配置--NAT

- 虚拟机借助NAT(网络地址转换)功能
- 由vmnet8的DHCP服务器提供的
- 选取自动获取IP
- 虚拟机<->主机, 虚拟机->互联网

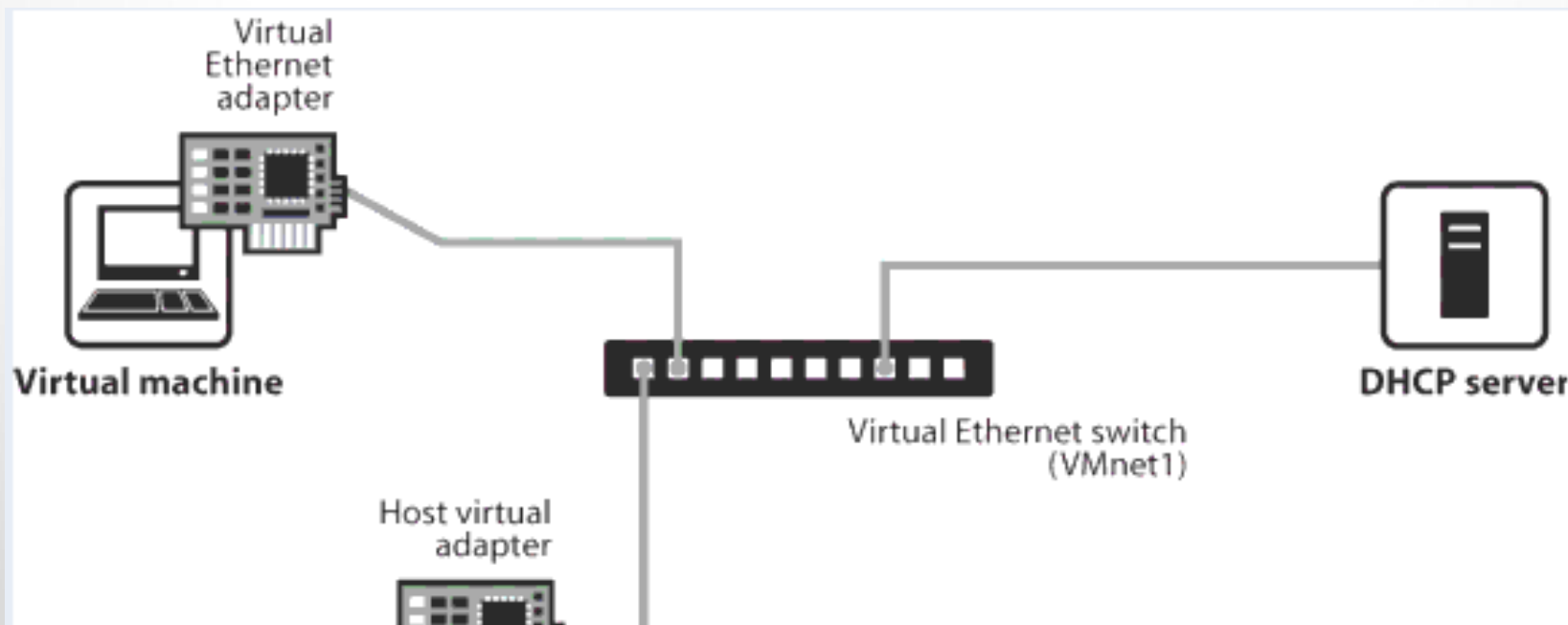




1.VMWare 平台交互

□ 网络配置—Host-only

- 虚拟系统可相互通信，和网络隔离开
- 虚拟机 通过主机真实的网卡进行外网的访问
- 由VMnet1的DHCP服务器来动态分配

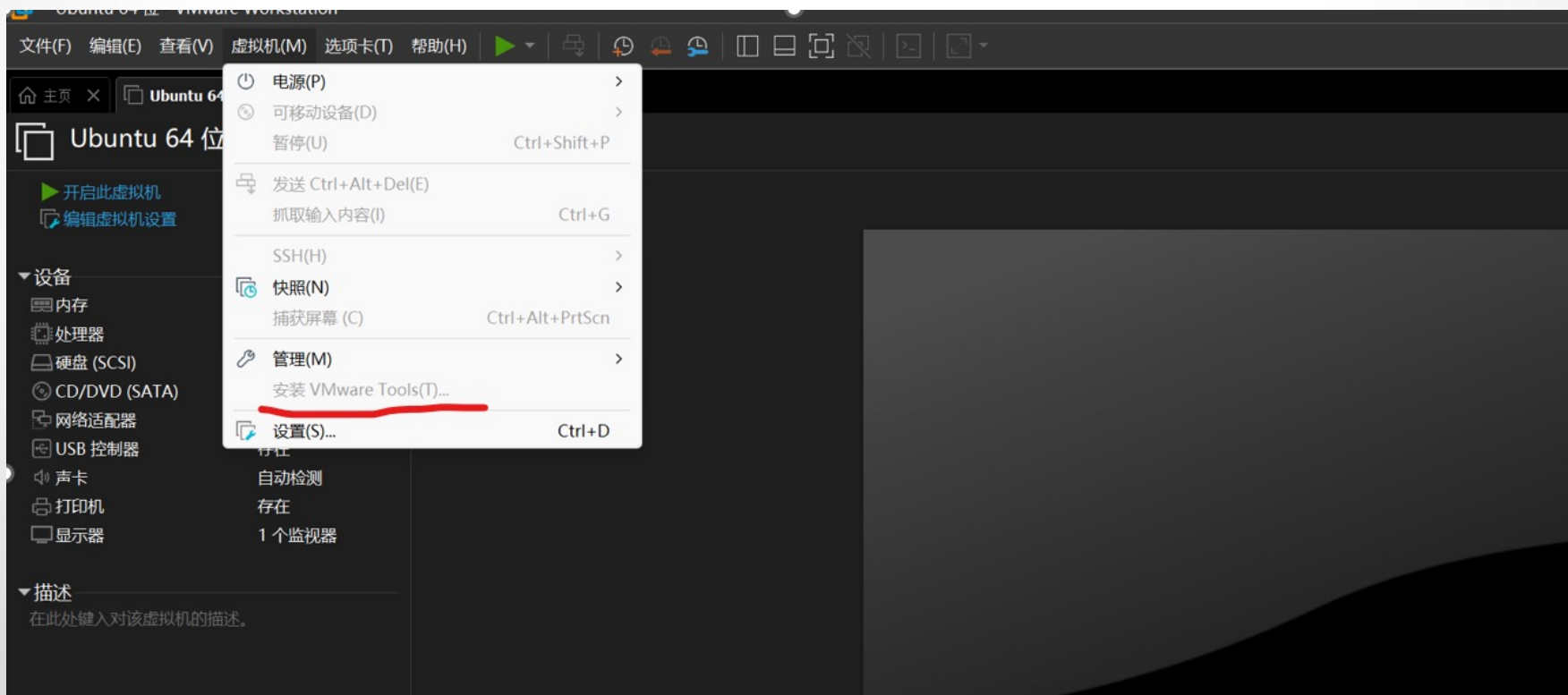


1.VMWare 平台交互

□与主机交互

■U盘

■VMWare Tools





□ 与主机交互

- SSH (Secure Shell) 它是建立在应用层基础上的安全协议，专为远程登录会话和其他网络服务提供安全性的协议
- samba 它是一种在局域网上共享文件和打印机的一种通信协议，它为局域网内的不同计算机之间提供文件及打印机等资源的共享服务。



四川大学

国家示范性软件学院

SCU Software College



目录

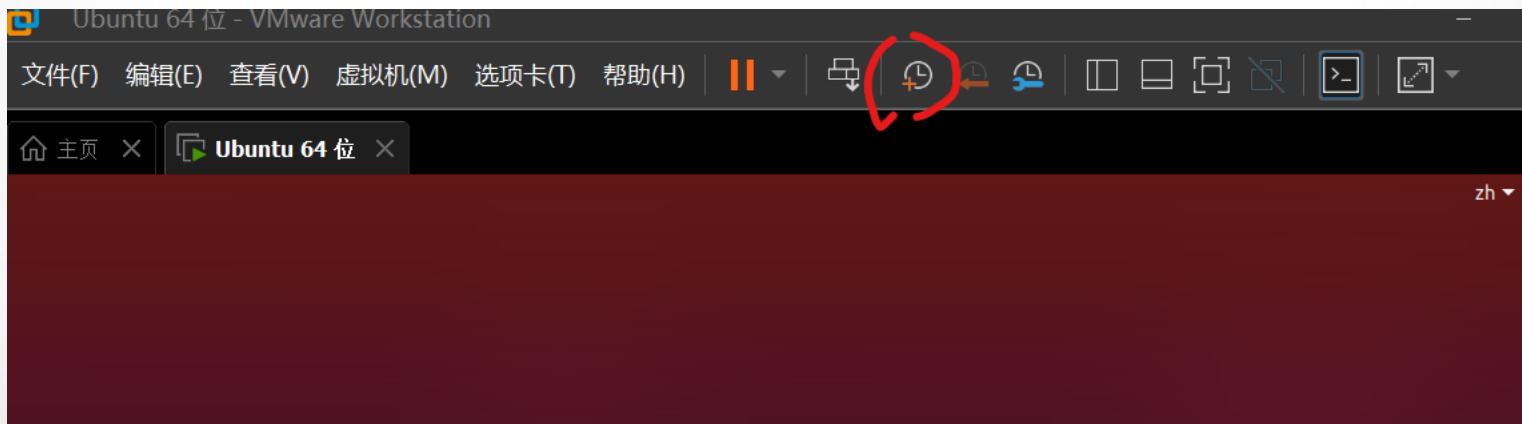
- 1.VMWare 平台交互
- 2.VMWare 其他说明
- 3.Shell编程简介



2.VMWare 其他说明_{1/1}

□快照

- 概念：磁盘快照是虚拟机磁盘文件（VMDK）在某个点及时的副本。
- 好处：如果当系统出现错误时，可通过恢复到快照来保持磁盘文件系统和系统存储。





四川大学

国家示范性软件学院

SCU Software College



目录

- 1.VMWare 平台交互
- 2.VMWare 其他说明
- 3.Shell编程简介



3.Shell编程简介_{1/20}

- **概念：** 是一种用C语言编写的程序，也是用户与Linux操作系统沟通的桥梁。
- **功能：** 用户既可以输入命令执行，又可以利用Shell脚本编程，完成更加复杂的操作
- **常见**
 - `bourne shell` 是一个交换式的命令解释器和命令编程语言
 - `C-shell` 主要是为了让用户更容易的使用交互式功能，并把ALGOL风格的语法结构变成了C语言风格
 - `korn shell` 它完全向上兼容 `Bourne shell` 并包含了
 - `C shell` 的很多特性
 -



3.Shell编程简介_{2/20}

□shell脚本

用普通的文本文件存储一系列等待以后执行的命令，称为脚本（script）。系统管理员经常利用脚本自动执行重复性管理工作。

一个shell程序可以包含：

- （1）命令或其他shell程序
- （2）位置参数
- （3）变量及特殊字符
- （4）表达式
- （5）控制流语句
- （6）函数



3.Shell编程简介_{3/20}

□shell脚本的特点

- 无需编译，解释执行
- 文本文件形式存在
- 强大的正则表达式操作
- 运行速度慢
- 数据类型支持少
- 用于系统管理和文件操作



四川大学

国家示范性软件学院

SCU Software College



3.Shell编程简介_{4/20}

□ shell脚本的编辑与运行

- (1) 建立、编辑脚本 (vi, gedit, ...)
- (2) 修改脚本文件属性为可执行 (chmod)
- (3) 运行脚本 (. /脚本文件名)



3.Shell编程简介 5/20

□例子:

```
#!/bin/sh
```

#!告诉系统其后路径所指定的程序是一个Shell脚本

```
#print hello world in the console window
```

#开头的行就是注释行

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

定义变量a

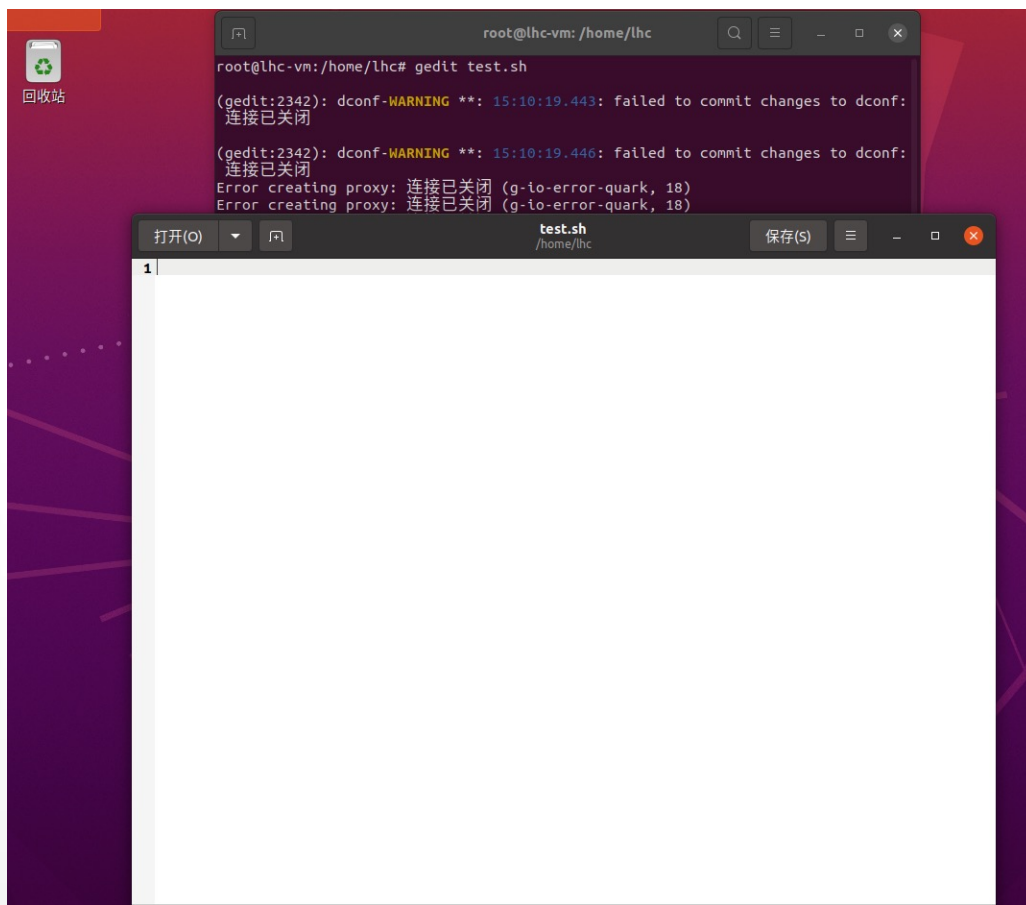
```
echo $a
```

打印变量，输出 hello world



操作:

`gedit filename.sh` #打开图像的文本编辑器





□ 例子:

```
#!/bin/sh          #!告诉系统其后路径所指定的程序是一个Shell脚本
#print hello world in the console window #开头的行就是注释行
a="hello world"    # 定义变量a
echo $a            # 打印变量，输出 hello world
```

```
Error creating proxy: 连接已关闭 (g-io-error-quark, 18)
test.sh
/home/lhc
1 #!/bin/sh
2 #print hello world in the console window #开头的行就是注释行
3 a="hello world"
4 echo $a
hello world
```



`chmod +x filename.sh`

chmod命令是控制用户对文件的权限的命令

- **u** 表示该文件的拥有者，**g** 表示与该文件的拥有者属于同一个群体者，**o** 表示其他以外的人，**a** 表示这三者皆是。
- **+** 表示增加权限、**-** 表示取消权限、**=** 表示唯一设定权限
- **r** 表示可读取，**w** 表示可写入，**x** 表示可执行，**X** 表示只有当该文件是个子目录或者该文件已经被设定过为可执行



Chmod +x test.sh

添加该文件权限为可执行文件

如图：

```
root@lhc-vm:/home/lhc# chmod +x test.sh  
root@lhc-vm:/home/lhc#
```



./test.sh 执行该sh脚本

```
root@lhc-vm:/home/lhc# chmod +x test.sh
root@lhc-vm:/home/lhc# ./test.sh
hello world
root@lhc-vm:/home/lhc#
```



3.Shell编程简介_{6/20}

□shell变量的特点

- (1) shell变量使用之前不需要事先对它进行声明，在第一次使用它的时候创建它。
- (2) shell变量默认情况下，是字符串类型。
- (3) shell变量名区分大小写。
- (4) 弱类型。



3.Shell编程简介_{7/20}

□从标准输入读取变量值

read 变量表

例: `$gedit script.sh`

```
echo "enter your name: "
```

```
read name
```

```
echo "your name is $name"
```

```
root@lhc-vm:/home/lhc# chmod +x test2.sh
root@lhc-vm:/home/lhc# ./test2.sh
enter your name:
lhc
your name is lhc
root@lhc-vm:/home/lhc#
```




□ 注意添加: `#!/bin/sh`

□ "`#!/bin/sh`"是对shell的声明,说明你所用的是那种类型的shell及其路径所在。如果没有声明,则脚本将在默认的shell中执行,默认shell是由用户所在的系统定义为执行shell脚本的shell.如果脚本被编写为在Kornshell ksh中运行,而默认运行shell脚本的为C shell csh,则脚本在执行过程中很可能失败。



3.Shell编程简介_{8/20}

□ 只读变量

■ readonly命令能保护变量，使其不被修改。

■ 例：

```
$passwd=shazam
```

```
$readonly passwd
```

```
$passwd=phoom
```

```
bash:passwd: readonly variable
```

```
$readonly      ——列出所有只读变量
```



```
1 #!/bin/sh
2 password="ossys"
3 readonly password
4 password="1234"
5 readonly
```

this could e.g. happen if you try to connect to
ser, over the native protocol. Don't do that.)
root@lhc-vm:/home/lhc# ./test3.sh
./test3.sh: 4: password: is read only
root@lhc-vm:/home/lhc# gedit test3.sh



3.Shell编程简介^{9/20}

□shell脚本中的运算符

■ 算术运算符

可用的算术运算符：+，-，*，/，%，++，--

- 原生bash不支持简单的数学运算，但是可以通过其他命令来实现，例如 `expr` 。
- `expr` 是一款表达式计算工具，使用它能完成表达式的求值操作。

```
1 #!/bin/sh
2
3 a=10
4 b=20
5
6 val=`expr $a + $b`
7 echo "a + b : $val"
8
9 val=`expr $a - $b`
10 echo "a - b : $val"
11
12 val=`expr $a \* $b`
13 echo "a * b : $val"
14
15 val=`expr $b / $a`
16 echo "b / a : $val"
17
18 val=`expr $b % $a`
19 echo "b % a : $val"
20
```





四川大学

国家示范性软件学院

SCU Software College



```
lhc@lhc-vm:~$ gcc -o op.sh
```

```
lhc@lhc-vm:~$ ./op.sh
```

```
30
```

```
-10
```

```
200
```

```
2
```

```
0
```

```
31 031 0
```



■ 关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

运算符	说明	举例
-eq	检测两个数是否相等，相等返回 <code>true</code> 。	<code>[\$a -eq \$b]</code> 返回 <code>false</code> 。
-ne	检测两个数是否不相等，不相等返回 <code>true</code> 。	<code>[\$a -ne \$b]</code> 返回 <code>true</code> 。
-gt	检测左边的数是否大于右边的，如果是，则返回 <code>true</code> 。	<code>[\$a -gt \$b]</code> 返回 <code>false</code> 。
-lt	检测左边的数是否小于右边的，如果是，则返回 <code>true</code> 。	<code>[\$a -lt \$b]</code> 返回 <code>true</code> 。
-ge	检测左边的数是否大于等于右边的，如果是，则返回 <code>true</code> 。	<code>[\$a -ge \$b]</code> 返回 <code>false</code> 。
-le	检测左边的数是否小于等于右边的，如果是，则返回 <code>true</code> 。	<code>[\$a -le \$b]</code> 返回 <code>true</code> 。



```
1 #!/bin/sh
2 a=10
3 b=20
4 if [ $a -eq $b ]
5 then
6     echo "$a -eq $b : a 等于 b"
7 else
8     echo "$a -eq $b: a 不等于 b"
9 fi
10 if [ $a -ne $b ]
11 then
12     echo "$a -ne $b: a 不等于 b"
13 else
14     echo "$a -ne $b : a 等于 b"
15 fi
16 if [ $a -gt $b ]
```

```
10 -eq 20: a 不等于 b
10 -ne 20: a 不等于 b
10 -gt 20: a 不大于 b
10 -lt 20: a 小于 b
10 -ge 20: a 小于 b
10 -le 20: a 小于或等于 b
```



■ 布尔运算符

运算符	说明	举例
!	非运算，表达式为 true 则返回 false， 否则返回 true。	[! false] 返回 true。
-o	或运算，有一个表达式为 true 则返回 true。	[\$a -lt 20 -o \$b -gt 100] 返回 true。
-a	与运算，两个表达式都为 true 才返回 true。	[\$a -lt 20 -a \$b -gt 100] 返回 false。



```
1 #!/bin/sh
2 a=10
3 b=20
4 if [ $a != $b ]
5 then
6 echo "$a != $b : a 不等于 b"
7 else
8 echo "$a == $b: a 等于 b"
9 fi
10 if [ $a -lt 100 -a $b -gt 15 ]
11 then
12 echo "$a 小于 100 且 $b 大于 15 : 返回 true"
13 else
14 echo "$a 小于 100 且 $b 大于 15 : 返回 false"
15 fi
16 if [ $a -lt 100 -o $b -gt 100 ]
17 then
18 echo "$a 小于 100 或 $b 大于 100 : 返回 true"
```

```
lhc@lhc-vm:~$ ./op2.sh
10 != 20 : a 不等于 b
10 小于 100 且 20 大于 15 : 返回 true
10 小于 100 或 20 大于 100 : 返回 true
10 小于 5 或 20 大于 100 : 返回 false
```



3.Shell编程简介_{10/20}

■ 例：测试1~20中哪些数能整除20

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

```
testval=20
```

```
count=2
```

```
while (($count <= $testval))
```

```
do
```

```
result=$(( $testval % $count))
```

```
if (($result == 0))
```

```
then
```

```
echo "$testval is divisible by $count"
```

```
fi
```

```
count=$(( $count + 1))
```

```
done
```



四川大学

国家示范性软件学院

SCU Software College



3.Shell编程简介_{11/20}

□流程控制

■if then else语句

if 条件命令串

then

条件为真时的命令串

else

条件为假时的命令串

fi



3.Shell编程简介_{12/20}

```
例: #!/bin/sh
```

```
if (($?==0))
```

```
then
```

```
echo "command was sucessful."
```

```
else
```

```
echo "An error was encountered."
```

```
fi
```

```
exit
```

#命令执行后返回值为0，报告命令执行成功，否则报告出错。



3.Shell编程简介_{13/20}

■ for循环

for 变量名 [in数值列表]

do

若干个命令行

done

例: #!/bin/sh

```
for i in 0 1 2 3 4 5 6 7 8 9
```

```
do
```

```
    echo $i
```

```
done
```



3.Shell编程简介_{14/20}

■ while和until循环

while 条件命令串

do

若干个命令行1

done

until 条件命令串

do

若干个命令行2

done

Shell还提供了true和false两条命令用于创建无限循环结构



3.Shell编程简介_{15/20}

例: #!/bin/bash

x=0

while ((\$x<=10))

do

echo \$x

x=\$((x+1))

done

#!/bin/bash

x=0

until ((\$x>10))

do

echo \$x

x=\$((x+1))

done

```
lhc@lhc-vm:~$ ./for.sh
```

```
0
1
2
3
4
5
6
7
8
9
10
```

终端

```
lhc@lhc-vm:~$ ./until.sh
```

```
0
1
2
3
4
5
6
7
8
9
10
```



3.Shell编程简介_{16/20}

■ case条件选择

```
case string in
```

```
exp-1)
```

```
若干个命令行1
```

```
; ;
```

```
exp-2)
```

```
若干个命令行2
```

```
; ;
```

```
.....
```

```
*)
```

#用*通配符来处理无匹配项情况

```
esac
```




3.Shell编程简介_{17/20}

```
例: #!/bin/sh
echo "Enter A, B, C"
read letter
case $letter in
A|a) echo "you entered A" ;;
B|b) echo "you entered B" ;;
C|c) echo "you entered C" ;;
*) echo "not A, B, C"
esac
```



```
you entered B  
lh@lh-vm:~$ ./case.sh  
"Enter A,B,C"  
D  
"not A,B,C"  
lh@lh-vm:~$ ./case.sh  
"Enter A,B,C"  
B  
"you entered B"
```



3.Shell编程简介_{18/20}

□ 函数

■ 函数定义

```
functionName () {  
    命令序列;  
}
```

■ 函数调用

```
functionName  
functionName 位置参数
```

■ 函数返回

```
return 用函数中执行的上一个命令的退出码返回;  
return [value] 用给定的value值返回;
```



```
lhc@lhc-vm:~$ chmod +x test.sh
lhc@lhc-vm:~$ ./test.sh
20 is divisible by 2
20 is divisible by 4
20 is divisible by 5
20 is divisible by 10
20 is divisible by 20
lhc@lhc-vm:~$
```



3.Shell编程简介^{19/20}

■ 全局函数

export命令可以将函数说明为全局函数，使其可以被子shell继承。

■ 函数共享

把要共享的函数单独放在一个文件中，然后在要使用该函数的脚本中，在开始位置用以下格式的命令读取该文件。

. fileName 或 source fileName

■ 变量限制

local 变量名

限制变量只能本地用于当前函数。



3.Shell编程简介_{20/20}

例: #!/bin/sh

f()

{

echo parameter 1=\$1

echo parameter list=\$*

}

#main program.

f 1

f cat my file1 file2

```
lhclhc@lhclhc-vm:~$ ./function.sh
```

```
parameter 1=1
```

```
parameter list=1
```

```
parameter 1=cat
```

```
parameter list=cat my file1 file2
```





四川大学
国家示范性软件学院
SCU Software College



THE END...

THANK YOU~