



Linux操作系统

单位：杭州电子科技大学
通信工程学院



8.3 Eclipse开发环境

开发工具:

Eclipse

Codeblocks

Nemiver

IntelliJ IDEA

Netbeans

.....

开发工具哪家强?



8.3 Eclipse开发环境

- Eclipse是著名的跨平台的自由集成开发环境（IDE）。最初主要用来Java语言开发，现在可以通过安装插件使其作为C++、Python、PHP等其他语言的开发工具。Eclipse的本身只是一个框架平台，但是众多插件的支持，使得Eclipse拥有较佳的灵活性。许多软件开发商以Eclipse为框架开发自己的IDE。本节将使用eclipse搭建c语言集成开发环境。



8.3.1 安装openjdk

1. 安装

运行eclipse需要有JDK(Java Development Kit)的支持，可以安装openjdk和oraclejdk，这里安装的是openjdk，如下图所示。

```
root@malimei-virtual-machine:/usr/bin# apt-get install openjdk-8-jdk
正在读取软件包列表... 完成
正在分析软件包的依赖关系树
正在读取状态信息... 完成
将会同时安装下列软件：
ca-certificates-java fonts-dejavu-extra java-common libgif7 libice-dev
libpthread-stubs0-dev libsm-dev libx11-dev libx11-doc libxau-dev libxcb1-dev
libxdmcp-dev libxt-dev openjdk-8-jdk-headless openjdk-8-jre
openjdk-8-jre-headless x11proto-core-dev x11proto-input-dev x11proto-kb-dev
xorg-sgml-doctools xtrans-dev
建议安装：
default-jre libice-doc libsm-doc libxcb-doc libxt-doc openjdk-8-demo
openjdk-8-source visualvm icedtea-8-plugin fonts-ipafont-gothic
fonts-ipafont-mincho fonts-wqy-microhei fonts-wqy-zenhei fonts-indic
下列【新】软件包将被安装：
ca-certificates-java fonts-dejavu-extra java-common libgif7 libice-dev
libpthread-stubs0-dev libsm-dev libx11-dev libx11-doc libxau-dev libxcb1-dev
libxdmcp-dev libxt-dev openjdk-8-jdk openjdk-8-jdk-headless openjdk-8-jre
openjdk-8-jre-headless x11proto-core-dev x11proto-input-dev x11proto-kb-dev
xorg-sgml-doctools xtrans-dev
升级了 0 个软件包，新安装了 22 个软件包，要卸载 0 个软件包，有 244 个软件包未被
```

2.选择

选择jdk版本，因为只安装了一个jdk，不用选择了，执行结果如下图所示。

```
root@lex-virtual-machine:/home/lex/桌面# java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~20.10-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
root@lex-virtual-machine:/home/lex/桌面#
```

如果安装了多个jdk，需要选择要执行的jdk,如下图所示。

```
root@malimei-virtual-machine:/usr/lib1/eclipse# update-alternatives --config java
有 2 个候选项可用于替换 java (提供 /usr/bin/java)。

  选择                路径                                      优先级    状态
-----
0                /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java    1081      自动模
式
1                /java/jdk1.8.0_221/bin/java                        300      手动模
式
* 2                /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java    1081      手动模
式

要维持当前值[*]请按<回车键>，或者键入选择的编号：
```



3.测试

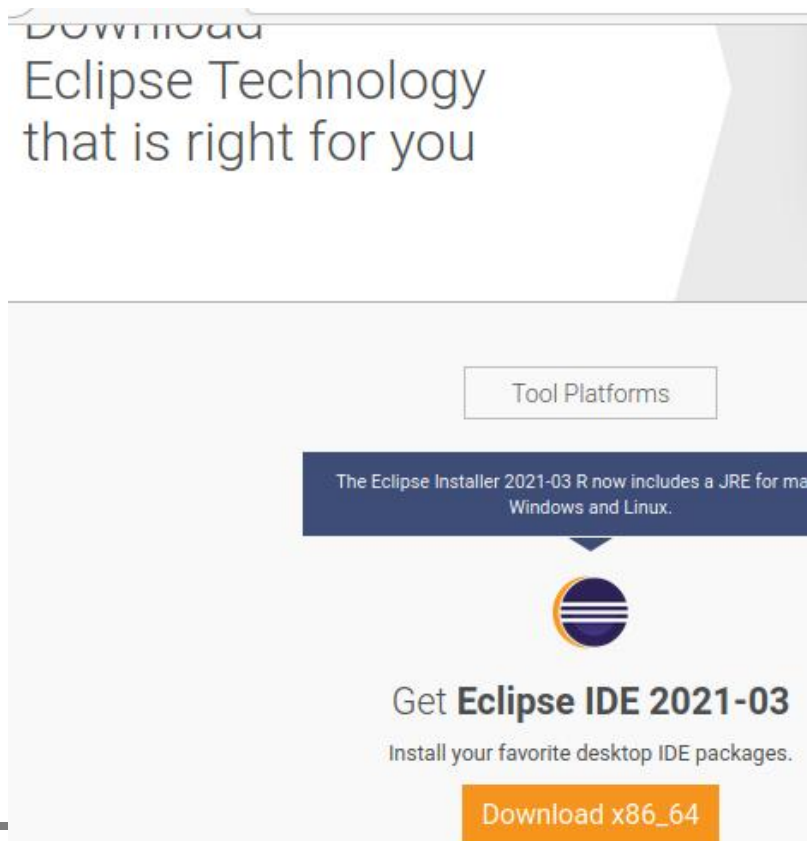
测试jdk安装是否安装成功，如果安装成功如下 图所示。

```
root@lex-virtual-machine:/home/lex/桌面# java -version
openjdk version "1.8.0_292"
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~20.10-b10)
OpenJDK 64-Bit Server VM (build 25.292-b10, mixed mode)
root@lex-virtual-machine:/home/lex/桌面#
```

8.3.2配置eclipse的集成开发环境

1.下载

(1) 下载eclipse的集成开发环境，官网下载网址为：<https://www.eclipse.org/downloads/>，打开网页后，显示如下图所示。



(2) 选择Eclipse ***R Packages, 单击右边Linux 64-bit 按钮, 如下图所示, 单击Download按钮下载即可.



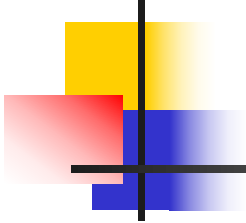


2.安装

(1) 把下载后的文件放到到/usr/lib/目录下，如下图所示.

```
eclipse-inst-jre-linux64.tar.gz  
lex@lex-virtual-machine:~/下载$ sudo cp eclipse-inst-jre-linux64.tar.gz /usr/lib  
[sudo] password for lex:  
lex@lex-virtual-machine:~/下载$
```

```
lex@lex-virtual-machine:/usr/lib$ ls eclipse*  
eclipse-inst-jre-linux64.tar.gz  
lex@lex-virtual-machine:/usr/lib$
```



(2) 将eclipse*.tar.gz文件解压到/usr/lib/目录，如下图所示。

```
lex@lex-virtual-machine:/usr/lib$ sudo tar -xzf eclipse-inst-jre-linux64.tar.gz
eclipse-installer/
eclipse-installer/p2/
eclipse-installer/p2/org.eclipse.equinox.p2.engine/
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/1615359430962.profile.gz
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/.lock
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/.data/
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/.data/org.eclipse.equinox.internal.p2.touchpoint.eclipse.actions/
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/.data/org.eclipse.equinox.internal.p2.touchpoint.eclipse.actions/jvmargs
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/1615359430965.profile.gz
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/1615359434865.profile.gz
eclipse-installer/p2/org.eclipse.equinox.p2.engine/profileRegistry/DefaultProfile.pro
file/1615359435183.profile.gz
```

(3) 解压后在该目录下，生成一个名为eclipse-installer的子目录，进入该目录，执行./eclipse-inst文件，执行结果如下图所示。

```
lex@lex-virtual-machine:/usr/lib$ cd eclipse-installer/  
lex@lex-virtual-machine:/usr/lib/eclipse-installer$ ls  
artifacts.xml  eclipse-inst      features  p2          readme  
configuration  eclipse-inst.ini  icon.xpm  plugins  
lex@lex-virtual-machine:/usr/lib/eclipse-installer$ ./eclipse-inst
```

type filter text



The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Maven and Gradle integration



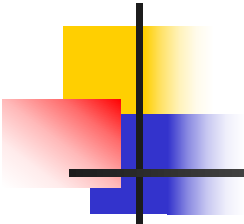
Eclipse IDE for Enterprise Java and Web Developers

Tools for developers working with Java and Web applications, including a Java IDE, tools for JavaScript, TypeScript, JavaServer Pages and Faces, Yaml, Markdown, Web..



Eclipse IDE for C/C++ Developers

An IDE for C/C++ developers.



(4) eclipse启动，如图8.61所示，单击Launch按钮。

Applicable licenses will be discovered and prompted later in the installation process.
Avoid such interruptions by accepting the licenses that govern Eclipse content now.



Eclipse Foundation Software User Agreement Versions

There are currently three versions of the Eclipse Foundation Software User Agreement. You may review them and accept them now, or wait until you are prompted to review and accept them later.

- [Eclipse Foundation Software User Agreement Version 2.0](#)
- [Eclipse Foundation Software User Agreement Version 1.1](#)
- [Eclipse Foundation Software User Agreement Version 1.0](#)


Version 2.0

Eclipse Foundation Software User Agreement

Decide Later

Accept Now

就可以编写C/C++程序了，如下图所示



Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

[details](#)

Java 11+ VM

//download.eclipse.org/justj/jres/15/updates/release/15.0.2


Installation Folder

/home/lex/eclipse/cpp-2021-03

☒ create start menu entry

☒ create desktop shortcut

INS



Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers.

[BACK](#)

a 11+ VM

JRE 15.0.2 - http://download.eclipse.org/justj/jres/15/updates/relea

Installation Folder

/home/lex/eclipse/cpp-2021-03

☒ create start menu entry

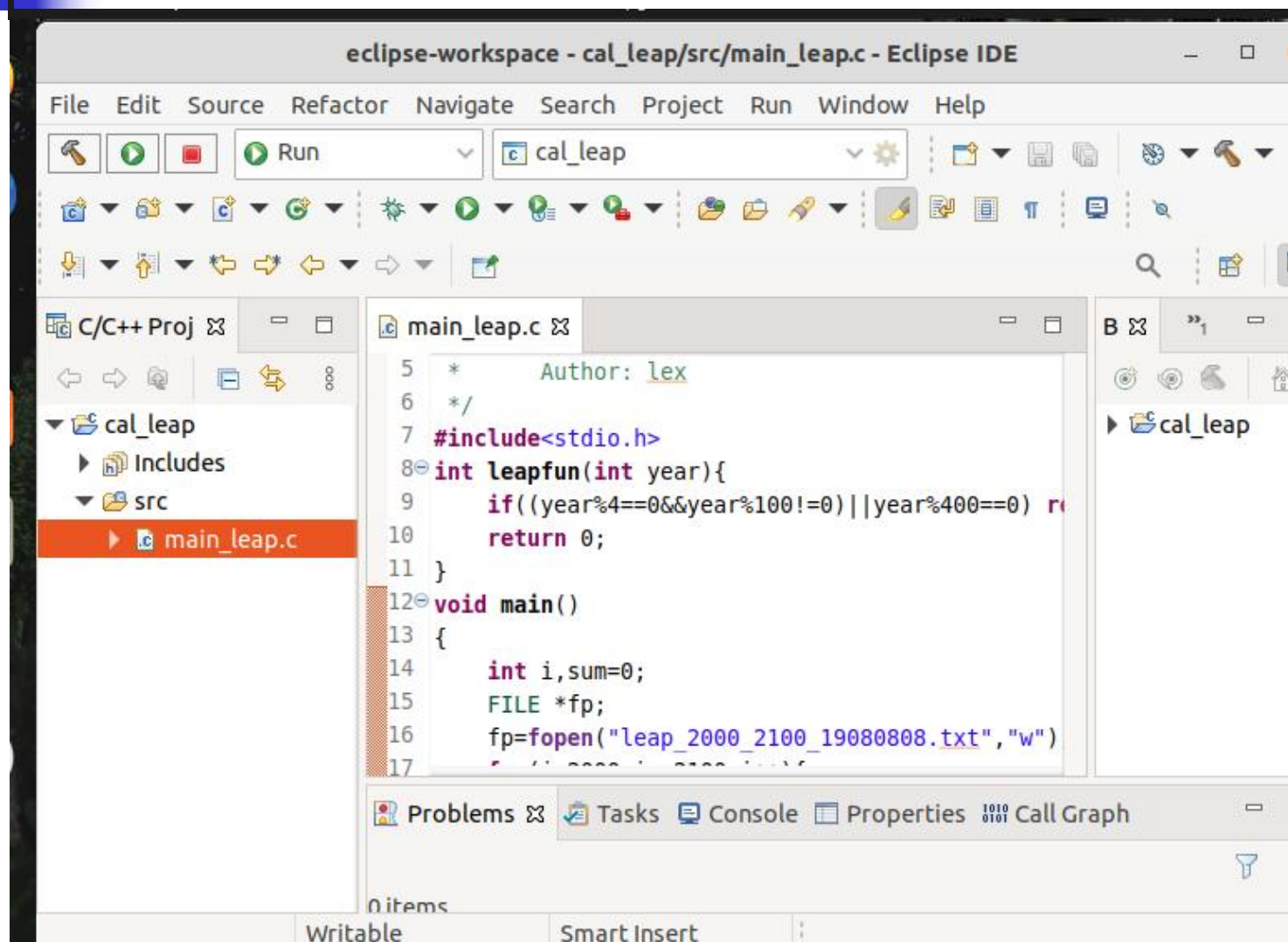
☒ create desktop shortcut

▶ LAUNCH

show readme file

13/17

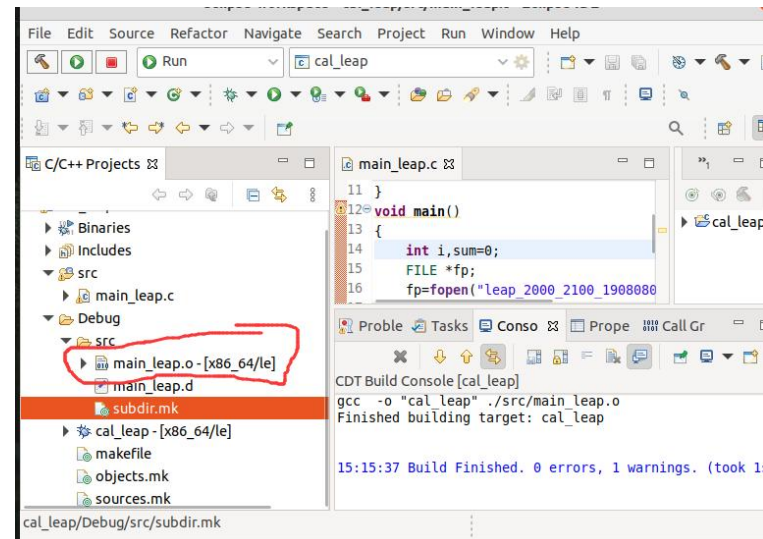
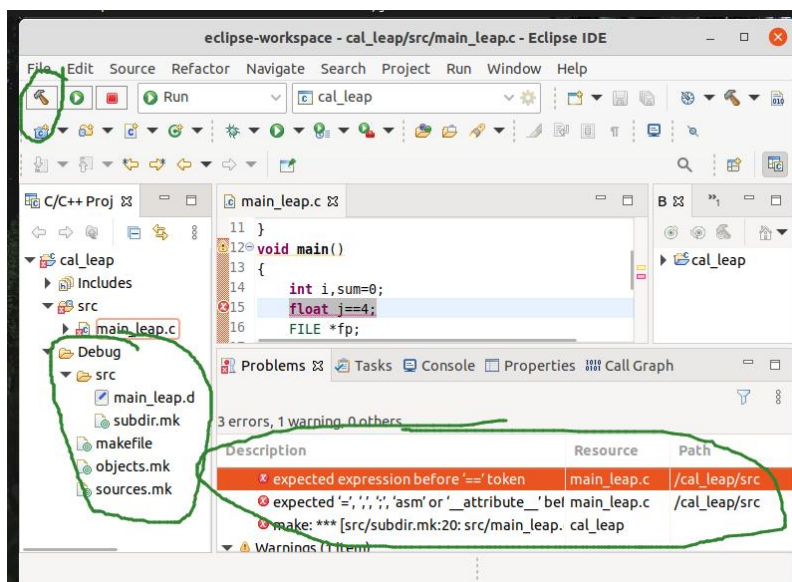
例：计算闰年



The screenshot shows the Eclipse IDE interface. The title bar reads "eclipse-workspace - cal_leap/src/main_leap.c - Eclipse IDE". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar contains icons for Run, Stop, and other development actions. The left sidebar shows a project tree with "cal_leap" containing "Includes" and "src" folders, with "main_leap.c" selected. The main editor displays the code for "main_leap.c". The code includes a comment "Author: lex", a header inclusion, and a function "leapfun" that checks for leap years. The "main" function initializes variables and opens a file "leap_2000_2100_19080808.txt".

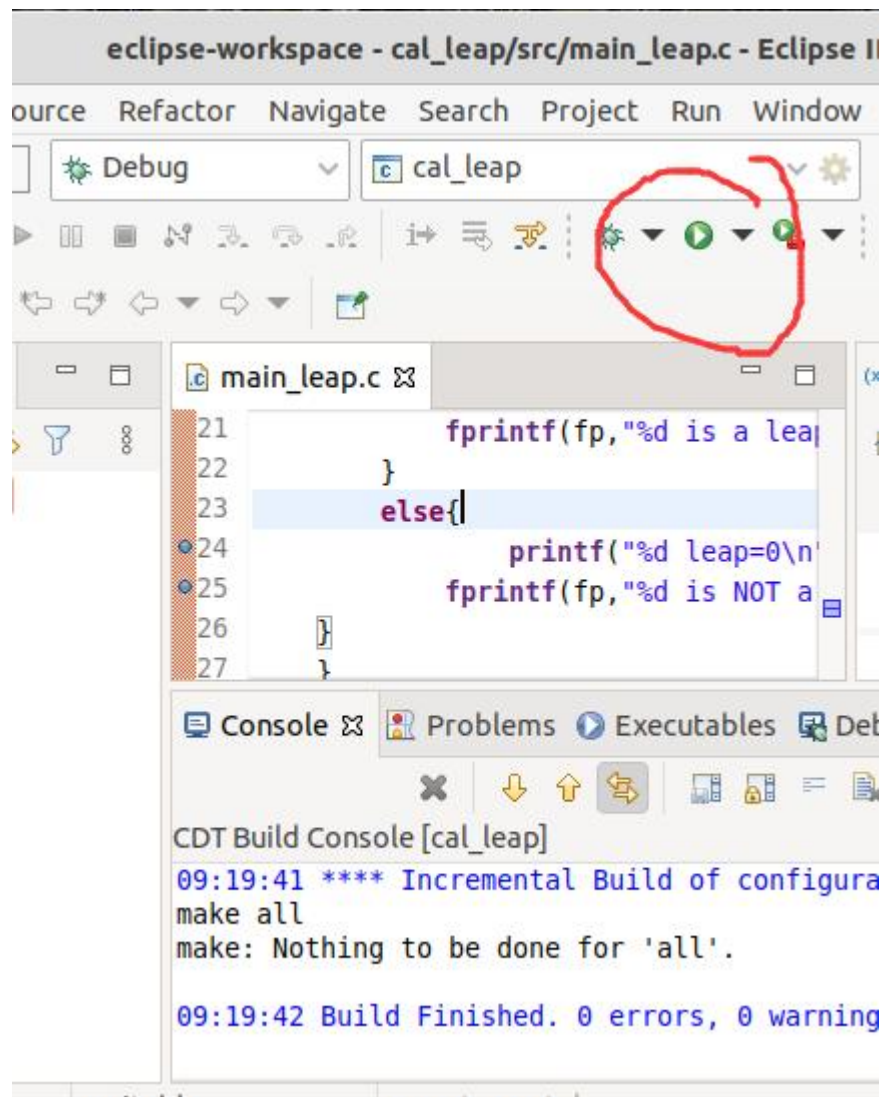
```
5  *   Author: lex
6  */
7  #include<stdio.h>
8  int leapfun(int year){
9      if((year%4==0&&year%100!=0)||year%400==0) re
10     return 0;
11 }
12 void main()
13 {
14     int i,sum=0;
15     FILE *fp;
16     fp=fopen("leap_2000_2100_19080808.txt","w")
17     f
```

编译链接，在Eclipse中，这个步骤称为Build Project。点击工具栏上的锤子图标



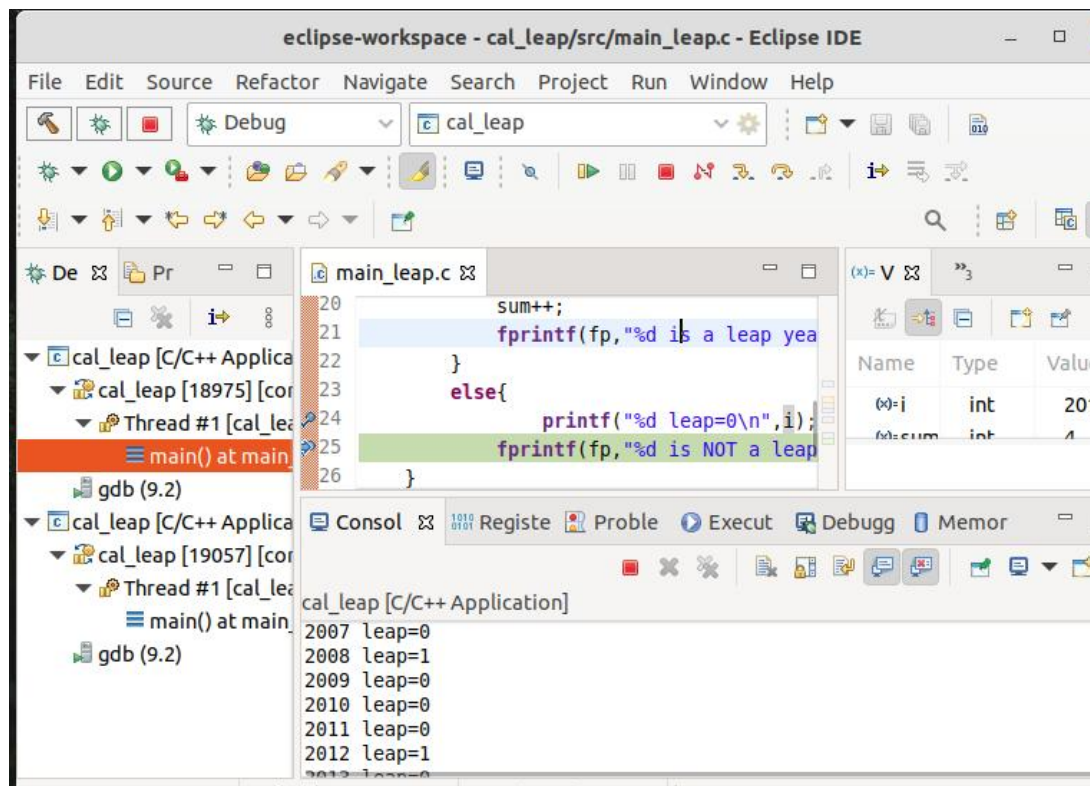
- 在Build时，若程序中有错误，Eclipse会将它显示出来。行首的红色或者黄色标记分别表示错误和警告，如图所示。错误的具体原因，会显示在屏幕的下方。
- 如果有错误，Eclipse是不能生成最终的可执行文件的，在执行程序前，必须要修正所有的错误。修改过源代码之后，Eclipse不会自动重新Build该项目，需要我们手工执行该操作，可以再次点击工具栏上的锤子型图标，也可以使用快捷键ctrl-b。

若Build Project成功，则可以运行程序。点击工具栏上的绿色Play图标或按快捷键“Ctrl-F11”即可运行程序。运行程序后，输入和输出都将在源代码编辑窗口下方的ConsoleView中进行。



调试:

- 双击行号前的竖框加断点,
- F6 单步调试
- F5 进入函数
- F7 跳出函数
- F8 继续执行到下一个断点处
- Ctrl+R 运行到光标处
- Ctrl+F2 终结调试





Linux操作系统

单位：杭州电子科技大学
通信工程学院



第九章 Shell及其编程

9.1 Shell 概述

9.2 Shell脚本执行方式

9.3 Shell脚本变量

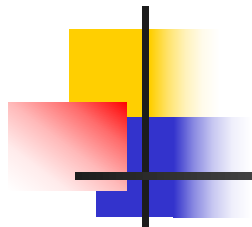
9.4 数组

9.5 Shell的输入输出

9.6 运算符和特殊字符

9.7 Shell语句

9.8 综合应用



通常情况下，命令行每输入一次命令就能够得到**系统的响应**，如果需要输入多条命令才能得到结果，那么这种操作效率无疑是很低的，**使用Shell程序或Shell脚本**就可以很好地解决这个问题，Shell编程语言具有普通编程语言的很多特点，比如它也有循环结构和分支控制结构等，用这种编程语言编写的Shell程序与其他应用程序具有同样的效果。

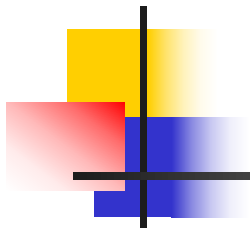


9.1 Shell概述

Shell就是可以接受用户**输入命令**的**程序**，它隐藏了操作系统低层的细节。

- Unix下的图形用户界面Gnome和KDE，有时也被叫做“虚拟shell”或“图形shell”。
- Linux操作系统下的**shell既是用户交互界面，也是控制系统的脚本语言**。
在Linux系列操作系统下，shell是控制系统启动、X Window启动、和很多其他实用工具的脚本解释程序。
- 每个Linux系统的用户可以**拥有他自己的用户界面或Shell**，用以满足自己专门的Shell需要。

通俗地讲,shell脚本就是写有一堆系统命令+简单的shell语法(变量、if判断、循环语句等)的一个文件,执行这文件能把所有命令一次性都执行了并实现一定的目的。



主要有下列版本的Shell:

(1) Bourne Shell: 是贝尔实验室开发的。

(2) C Shell: 是SUN公司Shell的BSD版本。

(3) Korn Shell: 是对Bourne Shell的发展, 在大部分内容上与Bourne Shell兼容。

(4) BASH: 是GNU的Bourne Again Shell, 是GNU操作系统上默认的shell。



9.1.1 Bourne Shell

第一个标准Linux Shell是1970年底在V7 Unix（AT&T第7版）中引入，以其资助者Stephen Bourne的名字命名。Bourne shell 是一个交换式的命令解释器和命令编程语言。

shell先读取/etc/profile文件和\$HOME /.profile文件。/etc/profile文件为所有用户定制环境，\$HOME/.profile文件为本用户定制环境。shell读取用户输入。

9.1.2 C Shell

C Shell 是Bill Joy在上世纪80年代早期，在Berkeley的加利福尼亚大学开发的。目的是让用户更容易的使用交互式功能，并把ALGOL风格，适于数值计算的语法结构变成了C语言风格。它新增了命令历史、别名、文件名替换、作业控制等功能。



9.1.3 Korn Shell

在很长一段时间里，只有两类shell供选择，Bourne shell用来编程，C shell用来交互。AT&T贝尔实验室的David Korn开发了Korn shell。Korn shell结合了所有的C shell的交互式特性，并融入了Bourne shell的语法，新增了数学计算，进程协作（coprocess）、编辑（inline editing）等功能。Korn Shell 是一个交互式的命令解释器和命令编程语言，它符合POSIX标准。

9.1.4 Bourne Again Shell

Bourne Again Shell，简称bash。也是GNU计划的一部分，用来替代Bourne shell。bash是大多数类Unix系统以及Mac OS X v10.4默认的shell，被移植到多种系统中。bash的语法针对Bourne shell的不足做了很多扩展。bash的命令语法很多来自Korn shell 和C shell。bash作为一个交互式的shell，按下TAB键即可自动补全已部分输入的程序名，文件名，变量名等等。



9.1.5查看用户shell

(1) 使用命令`cat /etc/shells`来查看/bin/目录下Ubuntu支持的Shell。

```
lex@lex-virtual-machine:/usr/lib/eclipse-installer$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
```

(2) `echo $SHELL`命令查看当前用户的shell。

```
lex@lex-virtual-machine:/usr/lib/eclipse-installer$ echo $SHELL
/bin/bash
```

(3) 其他用户的shell，可以在/etc/passwd文件中看到，并且可以修改，但要具有超级用户权限。



9.2 Shell脚本执行方式

9.2.1 Shell脚本概述

shell脚本是利用shell的功能所写的一个纯文本程序，将各类shell命令预先放入到一个文件中，方便一次性执行的一个程序文件，方便管理员进行设置或者管理。它与Windows下的批处理相似，一个操作执行多个命令。shell script 提供了数组、循环、条件以及逻辑判断等功能，可以直接以shell来写程序。



9.2.2 执行shell脚本几种方式

1. Shell脚本执行过程

- (1) shell脚本中，命令、参数间的多个空白以及空白行都会被忽略掉；
 - (2) 读到一个Enter符号（CR）或分号“；”时尝试开始执行该行（或该串）的命令；
 - (3) 如果一行的内容太多，则可以使用“\[Enter]”来扩展至下一行。
- 比如输出一行长的字符串，如图9.3所示。在shell脚本中，任何加在#后面的数据将全部被视为批注文字而被忽略。

```
lex@lex-virtual-machine:/usr/lib/eclipse-installer$ echo "this is a very \  
> long string."  
this is a very long string.
```

2. Shell脚本执行方式

(1) 直接命令执行

- 使用文本编辑器编辑生成脚本文件

```
lex@lex-virtual-machine: ~/lesson/lesson8  
echo "this is the shell script test"
```

使用ls -l命令，可以看到test.sh没有执行的权限。

```
lex@lex-virtual-machine:~/lesson/lesson8$ ls -l  
total 4  
-rw-r--r-- 1 root root 37  5月  4 18:57 test.sh
```

- 设置shell脚本文件的权限为可执行后再提示符下执行。

```
lex@lex-virtual-machine:~/lesson/lesson8$ sudo chmod a+x test.sh  
lex@lex-virtual-machine:~/lesson/lesson8$ ls -l  
total 4  
-rwxr-xr-x 1 root root 37  5月  4 18:57 test.sh  
lex@lex-virtual-machine:~/lesson/lesson8$ ./test.sh  
this is the shell script test  
lex@lex-virtual-machine:~/lesson/lesson8$
```



(2) sh/bash [选项] 脚本名

打开一个shell读取并执行脚本中命令。

执行：sh/bash 文件名.sh sh/bash ./

sh或bash在执行脚本过程中，选项如下：

- n 不要执行script，仅检查语法的问题。
- v 在执行script前，先将script的内容输出到屏幕上。
- x 进入跟踪方式，显示所执行的每一条命令，并且在行首显示一个“+”号。

```
lex@lex-vm:~/lesson/lesson8$ sh -v test.sh
echo "this is the shell script test"
this is the shell script test
lex@lex-vm:~/lesson/lesson8$ sh -x test.sh
+ echo this is the shell script test
this is the shell script test
```


(3) source 脚本名或者 ./脚本名

①在当前bash环境下读取并执行脚本中命令。

```
lex@lex-vm:~/lesson/lesson8$ source ./test.sh  
this is the shell script test
```

注意：执行shell脚本文件时，直接命令方式需要设置脚本为可执行权限，sh/bash和source 模式不需要脚本文件具有可执行权限。

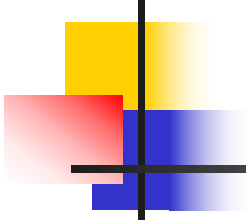
```
lex@lex-vm:~/lesson/lesson8$ ls -l  
total 4  
-rw-r--r-- 1 root root 37 5月 4 20:33 test.sh  
lex@lex-vm:~/lesson/lesson8$ ./test.sh  
bash: ./test.sh: Permission denied  
lex@lex-vm:~/lesson/lesson8$ sh test.sh  
this is the shell script test  
lex@lex-vm:~/lesson/lesson8$ source ./test.sh  
this is the shell script test  
lex@lex-vm:~/lesson/lesson8$
```



9.3 Shell脚本变量

shell脚本变量就是在shell脚本程序中保存的系统和用户所需要的各种各样的值。

- (1) 系统变量
- (2) 环境变量
- (3) 用户自定义变量



9.3.1 系统变量

Shell常用的系统变量

按键	命令
\$#	命令行参数的个数
\$n	当前程序的第n个参数, n=1,2,...9
\$0	当前程序的名称
\$?	执行上一个指令或函数的返回值
\$*	以“参数1 参数2 ...”形式保存所有参数
\$@	以“参数1”“参数2”... 形式保存所有参数
\$\$	本程序的(进程ID号)PID
#!	上一个命令的PID
\$-	显示shell使用的当前选项, 与set命令功能相同



例：分析名为sysvar.sh脚本的运行结果。sysvar.sh脚本的代码如下：

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

```
#to explain the application of system variables.
```

```
echo "\$1 = $1 ; \$#2 = $2; \&3=$3;\%4=$4 ";
```

```
echo "the number of parameter is $ # "; $#
```

```
echo "the return code of last command is $?";
```

```
echo "the script name is $0 ";
```

```
echo "the parameters are $* ";
```

```
echo "the parameters are @$ ";
```

```
echo " the last command pid is $! ";
```

```
echo " the current pid is $$ ";
```

◆ 反斜线 (\)

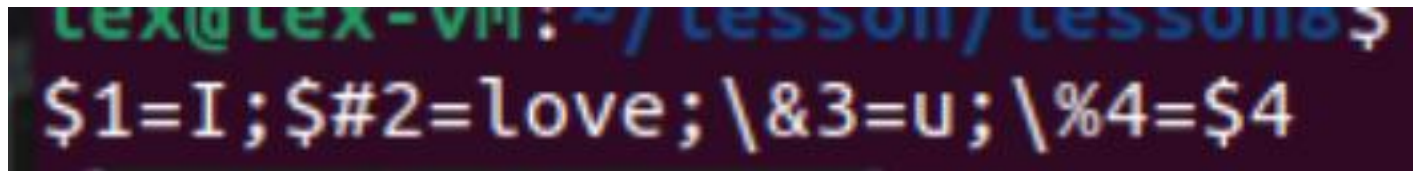
反斜线是转义字符，它告诉Shell不要对其后面的那个字符进行特殊处理，只当做普通字符即可。

\出现后会判断后面是否出现系统变量，如果出现，则将后面出现系统变量字符当作普通字符，如果不出现，则\也作为普通字符输出。



例：某shell脚本的输入参数为I love u forever,其中某一行文本如下所示，请写出执行该命令后的输出结果。

`echo "\$1=$1;\$#2=$2;\&3=$3;\%4=\$4 ";`



```
lex@lex-vm: ~/lesson/lesson8$  
$1=I;$#2=love;\&3=u;\%4=$4
```

Shell中的几个特殊字符\$ \ ` ""

- \$符号在双引号中具有特殊意义，即取变量值或其他
- 反斜杠也可以将特殊字符的特殊含义屏蔽掉，使特殊字符失去特殊含义
- 反引号是命令替换，将其中的字符串作为命令来执行
- 双引号对\$符号不起作用而单引号可以将特殊字符的特殊意义屏蔽掉，使其能显示为字符本身

例1: (a)echo \$? (b)echo "\$?"
(c)echo '\$?' (d)echo \ \$? (e)echo "\\$?"

```
lex@lex-vm:~/lesson/lesson8$ echo $?  
0  
lex@lex-vm:~/lesson/lesson8$ echo "$?"  
0  
lex@lex-vm:~/lesson/lesson8$ echo '$?'  
$?  
lex@lex-vm:~/lesson/lesson8$ echo \ $?  
$?  
lex@lex-vm:~/lesson/lesson8$ echo "\$?"  
$?
```

例2: a=123,比较(a)echo a (b)echo \$a
(c)echo \ \$a (d)echo \' (e)echo \' (f)echo \\
(g)echo \\

```
lex@lex-vm:~/lesson/lesson8$ a=123  
lex@lex-vm:~/lesson/lesson8$ echo a  
a  
lex@lex-vm:~/lesson/lesson8$ echo $a  
123  
lex@lex-vm:~/lesson/lesson8$ echo \ $a  
$a  
lex@lex-vm:~/lesson/lesson8$ echo \'  
'  
lex@lex-vm:~/lesson/lesson8$ echo \\  
\  
lex@lex-vm:~/lesson/lesson8$
```

例3: b=`date`,查看echo b和 echo \$b的输出结果

```
lex@lex-vm:~/lesson/lesson8$ b=`date`  
lex@lex-vm:~/lesson/lesson8$ echo b  
b  
lex@lex-vm:~/lesson/lesson8$ echo $b  
2021年 05月 04日 星期二 22:22:02 CST
```

例4: c=`cat sysvar.sh`,查看echo c; echo \$c;echo
“\$c”;echo “\ \$c”的输出结果

```
lex@lex-vm:~/lesson/lesson8$ c=`cat sysvar.sh`  
lex@lex-vm:~/lesson/lesson8$ echo c  
c  
lex@lex-vm:~/lesson/lesson8$ echo $c  
#!/bin/sh #to explain the application of system variables. ec  
ho "\$1=$1;\$#2=$2;\&3=$3;\%4=\$4"; echo "the number of param  
eter is $#"; echo "the return code of last command is $?"; ec  
ho "the script name is $0"; echo "the parameters are $*"; ech  
o "the parameters are $@"; echo "the last command pid is $!";  
echo "the current pid is $$";  
lex@lex-vm:~/lesson/lesson8$ echo "$c"  
#!/bin/sh  
#to explain the application of system variables.  
echo "\$1=$1;\$#2=$2;\&3=$3;\%4=\$4";  
echo "the number of parameter is $#";  
echo "the return code of last command is $?";  
echo "the script name is $0";  
echo "the parameters are $*";  
echo "the parameters are $@";  
echo "the last command pid is $!";  
echo "the current pid is $$";  
lex@lex-vm:~/lesson/lesson8$ echo "\ $c"  
$c
```

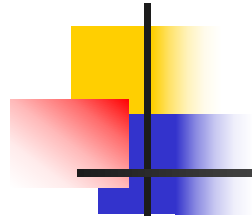



9.3.2 环境变量

- 登入系统就获得一个shell，它占据一个进程，输入的命令都属于这个shell进程的子进程，选择此shell后，获得一些环境设定，即环境变量。
- 环境变量约束用户行为，也帮助实现很多功能，包括：
 - ◆ (1)主目录的变换；
 - ◆ (2)自定义显示符的提示方法；
 - ◆ (3)设定执行文件查找的路径等。

常用的环境变量

按键	命令
PATH	命令搜索路径,以冒号为分隔符.但当前目录不在系统路径里
HOME	用户home目录的路径名,是cd命令的默认参数
COLUMNS	定义了命令编辑模式下可使用命令行的长度
EDITOR	默认的行编辑器
VISUAL	默认的可视编辑器
FCEDIT	命令fc使用的编辑器
HISTFILE	命令历史文件
HISTSIZE	命令历史文件中最多可包含的命令条数
HISTFILESIZE	命令历史文件中包含的最大行数
HISTORY	显示历史命令
IFS	定义shell使用的分隔符
LOGNAME	用户登录名
MAIL	指向一个需要shell监视修改时间的文件.当该文件修改后,shell发送消息You hava mail给用户
MAILCHECK	SHELL检查MAIL文件的周期,单位是秒
MAILPATH	功能与MAIL类似.但可以用一组文件,以冒号分隔,每个文件后可跟一个问号和一条发向用户的消息
SHELL	SHELL的路径名
TERM	终端类型
TMOUT	SHELL自动退出的时间,单位为秒,0为禁止SHELL自动退出
PROMPT_COMMAND	指定在主命令提示符前应执行的命令
PS1	主命令提示符
PS2	二级命令提示符,命令执行过程中要求输入数据时用



例：使用env命令查看环境变量。

为了方便查看，使用重定向命令（输出重定向）将环境变量存储到envs文件中，命令：`env > envs`，然后使用编辑器打开该文件，如图所示。

```
lex@lex-vm:~/lesson/lesson8$ cat envs
SHELL=/bin/bash
SESSION_MANAGER=local/lex-vm:@/tmp/.ICE-unix/1324,unix/lex-vm:/tmp/.ICE-unix/1324
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LANGUAGE=zh_CN:en
LC_ADDRESS=zh_CN.UTF-8
GNOME_SHELL_SESSION_MODE=ubuntu
L Files E=zh_CN.UTF-8
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
LC_MONETARY=zh_CN.UTF-8
SSH_AGENT_PID=1270
GTK_MODULES=gail:atk-bridge
PWD=/home/lex/lesson/lesson8
LOGNAME=lex
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
```

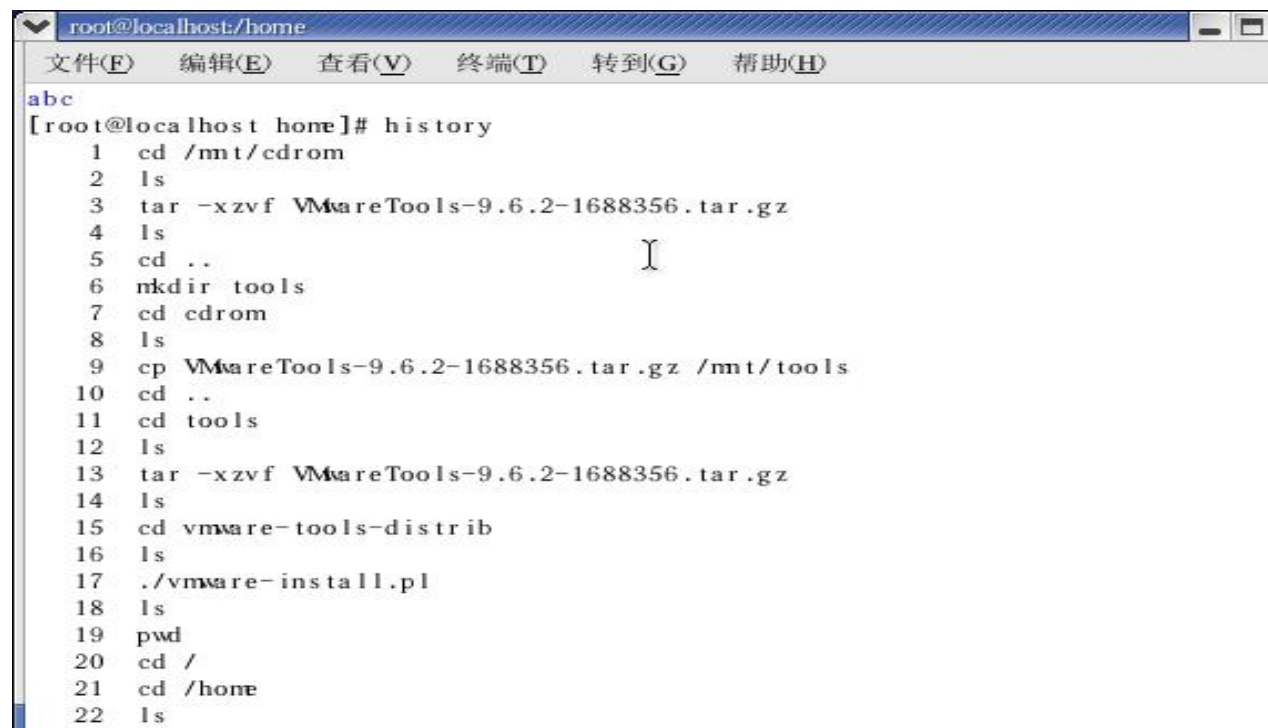


在Linux系统中有几个最有用的bash环境变量，这些变量名及功能如下：

(1) history命令用来显示历史命令。

history 命令： history [n]

当 history 命令没有参数时，整个历史命令列表的内容将被显示出来。
如图所示。使用 n 参数的作用是仅有最后 n 个历史命令会被列出。例如，history 5 显示最后 5 个命令。



```
root@localhost:/home
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
abc
[root@localhost home]# history
 1 cd /mnt/cdrom
 2 ls
 3 tar -xzf VMwareTools-9.6.2-1688356.tar.gz
 4 ls
 5 cd ..
 6 mkdir tools
 7 cd cdrom
 8 ls
 9 cp VMwareTools-9.6.2-1688356.tar.gz /mnt/tools
10 cd ..
11 cd tools
12 ls
13 tar -xzf VMwareTools-9.6.2-1688356.tar.gz
14 ls
15 cd vmware-tools-distrib
16 ls
17 ./vmware-install.pl
18 ls
19 pwd
20 cd /
21 cd /home
22 ls
```

(2) **alias**命令别名，命令别名通常是其他命令的缩写，用来减少键盘输入。

命令格式有两种：

alias 显示机器中已经定义的别名 **永久别名的设置 .bashrc**

alias [alias-name='original-command']

其中，alias-name是用户给命令取的别名，original-command是原来的命令和参数。如图所示。

永久更改别名：
1 进入 /home/lex
2 Vi .bashrc 文件
永久别名的设置 .bashrc



```
root@localhost:/mnt/tools/vmware-tools-distrib
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
bash: cd: /mnt/tools/vmware-tools-distrib: 没有那个文件或目录
[root@localhost /]# alias nic='cd /mnt/tools/vmware-tools-distrib'
[root@localhost /]# nic
[root@localhost vmware-tools-distrib]# alias copy='cp'
[root@localhost vmware-tools-distrib]# alias dir='ls'
[root@localhost vmware-tools-distrib]# alias ls ='ls -l'
alias ls='ls --color=tty'
bash: alias: =ls -l: not found
[root@localhost vmware-tools-distrib]# alias ls='ls -l'
[root@localhost vmware-tools-distrib]# alias
alias copy='cp'
alias cp='cp -i'
alias dir='ls'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls -l'
alias mv='mv -i'
alias nic='cd /mnt/tools/vmware-tools-distrib'
alias rm='rm -i'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[root@localhost vmware-tools-distrib]#
```



(3) Bash有两级提示符。第一级提示符是经常见到的Bash在等待命令输入时的情况。第一级提示符的默认值是\$/#符号。如果用户不喜欢这个符号，或者愿意自己定义提示符，只需修改PS1变量的值，注意PS1和PS2大写，参数如表所示。

特殊字符	说明
\!	显示该命令的历史编号
\#	显示shell激活后，当前命令的历史编号
\\$	显示一个\$符号，如果当前用户是root则显示#符号
\\	显示一个反斜杠
\d	显示当前日期
\h	显示运行该shell的计算机主机名
\n	打印一个换行符，这将导致提示符跨行
\s	显示正在运行的Shell的名称
\t	显示当前时间
\u	显示当前用户的用户名
\W	显示当前工作目录基准名
\w	显示当前工作目录



格式: PS1=”输入一个命令: ”

例1: PS1='\d', 将提示符改为日期

```
lex@lex-vm:~/lesson/lesson8$ PS1='\d'  
二 5月 04
```

例 2: 加参数\u@\h\w\\$, 显示系统原来的提示符号

```
lex@lex-vm:~/lesson/lesson8$ PS1='\d'  
二 5月 04PS1="\u@\h:\w\$"  
lex@lex-vm:~/lesson/lesson8$
```

第二级提示符是当Bash为执行某条命令需要用户输入更多信息时显示的。第二级提示符默认为“>”。如果需要自己定义该提示符，只需改变PS2变量的值。例如将其改为：

PS2=”please input: ”，如图所示，将二级提示符修改为please input。

```
7 7 14  
lex@lex-vm:~/lesson/lesson8$wc<<end  
please input1  
please inputalsdt  
please inputend  
2 2 8  
lex@lex-vm:~/lesson/lesson8$
```




9.3.3 自定义变量

Shell编程中，使用变量无需事先声明，同时变量名的命名须遵循如下规则：

- (1) 首个字符必须为字母（a-z, A-Z）
- (2) 中间不能有空格，可以使用下划线（_）
- (3) 不能使用标点符号
- (4) 不能使用bash里的关键字（可用help命令查看保留关键字）

例1：下面变量名是合法的

book123 My_12

例2：下面变量名是不合法的

123abd bc&123

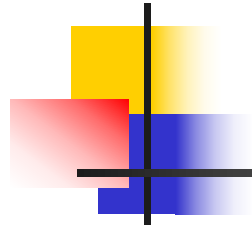


9.3.4 自定义变量的使用

1. 变量值的引用与输出

- (1) 引用变量时在变量名前面加上 \$ 符号。
- (2) 输出变量时用 echo。
- (3) 如果变量恰巧包含在其他字符串中，为了区分变量和其他字符串，**需要用 {} 将变量名括起来，也可以直接引用变量**，如图所示。

```
lex@lex-vm:~/lesson/lesson8$ day=monday
lex@lex-vm:~/lesson/lesson8$ echo day
day
lex@lex-vm:~/lesson/lesson8$ echo $day
monday
lex@lex-vm:~/lesson/lesson8$ echo "today is $day"
today is monday
lex@lex-vm:~/lesson/lesson8$ echo "today is ${day}"
today is monday
```



2. 变量的赋值和替换

(1) 变量赋值的方式：变量名=值

例：

day=monday

string=welcome!

注意：给变量赋值的时候，不能在“=”两边留空格，如图所示。

```
=monday
lex@lex-vm:~/lesson/lesson8$ day = monday
Command 'day' not found, did you mean:
  command 'dat' from deb liballegro4-dev (2:4.4.3.1-1)
  command 'dan' from deb emboss (6.6.0+dfsg-7ubuntu2)
  command 'dab' from deb bsdgames (2.17-28build1)
  command 'say' from deb gnustep-gui-runtime (0.28.0-3build1)
  command 'dav' from deb dav-text (0.9.0-1)
  command 'dar' from deb dar (2.6.10-1)
  command 'dad' from deb debian-dad (1)
Try: sudo apt install <deb name>
```



(2) 重置就相当于赋给这个变量另外一个值

(3) 清空某一变量的值可以使用unset命令

```
lex@lex-vm:~/lesson/lesson8$ day=monday
lex@lex-vm:~/lesson/lesson8$ echo $day
monday
lex@lex-vm:~/lesson/lesson8$ day=sunday
lex@lex-vm:~/lesson/lesson8$ echo $day
sunday
lex@lex-vm:~/lesson/lesson8$ unset day
lex@lex-vm:~/lesson/lesson8$ echo $day
lex@lex-vm:~/lesson/lesson8$
```



(4) 变量可以有条件的替换, **替换**条件放在一对大括号{}中。

①当变量未定义或者值为空时,返回值为value的内容,否则, 返回变量的值。
其格式为: `${variable:-value}` 。

```
lex@lex-vm:~/lesson/lesson8$ echo $th
lex@lex-vm:~/lesson/lesson8$ echo ${th:-world}
world
lex@lex-vm:~/lesson/lesson8$ th=China
lex@lex-vm:~/lesson/lesson8$ echo ${th:-world}
China
```

②若变量未定义或者值为空时,在返回value的值的的同时value**赋值给**variable。
其格式为: `${variable:=value}` 。

```
lex@lex-vm:~/lesson/lesson8$ echo $th
China
lex@lex-vm:~/lesson/lesson8$ echo ${th:=world}
China
lex@lex-vm:~/lesson/lesson8$ unset th
lex@lex-vm:~/lesson/lesson8$ echo ${th:=world}
world
lex@lex-vm:~/lesson/lesson8$ echo $th
world
```

③若变量已赋值的话,其值才用value替换,否则不进行任何替换。
其格式为: `${variable:+value}` value替换variable ,

```
lex@lex-vm:~/lesson/lesson8$ unset th
lex@lex-vm:~/lesson/lesson8$ echo $th

lex@lex-vm:~/lesson/lesson8$ echo ${th:+world}

lex@lex-vm:~/lesson/lesson8$ echo $th

lex@lex-vm:~/lesson/lesson8$ th=China
lex@lex-vm:~/lesson/lesson8$ echo ${th:+world}
world
lex@lex-vm:~/lesson/lesson8$ echo $th
China
```



Linux操作系统

单位：杭州电子科技大学
通信工程学院



9.4 数组

- (1) **bash支持一维数组（不支持多维数组）**，并且没有限定数组的大小；
- (2) 类似于C语言，数组元素的下标由0开始编号。获取数组中的元素要利用下标，下标可以是整数或算术表达式，其值应大于或等于0；
- (3) 数组的使用可以**先声明，再赋值，也可以直接赋值。**

9.4.1 数组的声明

➤ 对数组进行声明，使用declare命令，

格式：declare [+/-] [选项] variable

✓ -/+：指定/关闭变量的属性

- **a**:定义后面名为variable的变量为数组（array）类型
- **i**:定义后面名为variable的变量为整数数字（integer）类型
- **x**: 将后面的variable变成环境变量
- **r**: 将变量设置成readonly类型
- **f**: 将后面的变量定义为函数或显示函数
- **p**:查看变量被声明的类型

注意：

(1)declare既可以声明变量(-i),也可以声明数组(-a)

(2)直接引用数组名names,相当于引用names[0]

(3)即echo \${#names}输出names[0]的字符个数

(4) @ 或 * 可以获取数组中的全部元素

```
lex@lex-vm:~/lesson/lesson9$ declare -i a=4.3
bash: declare: 4.3: syntax error: invalid arithmetic operator
(error token is ".3")
lex@lex-vm:~/lesson/lesson9$ echo $a

lex@lex-vm:~/lesson/lesson9$ declare +i a=4.3
lex@lex-vm:~/lesson/lesson9$ echo $a
4.3
lex@lex-vm:~/lesson/lesson9$ declare -a names
lex@lex-vm:~/lesson/lesson9$ names=lex0
lex@lex-vm:~/lesson/lesson9$ echo $names
lex0
lex@lex-vm:~/lesson/lesson9$ names[1]=lex1
lex@lex-vm:~/lesson/lesson9$ names[2]=lex2
lex@lex-vm:~/lesson/lesson9$ echo ${names}
lex0
lex@lex-vm:~/lesson/lesson9$ echo ${names[*]}
lex0 lex1 lex2
lex@lex-vm:~/lesson/lesson9$ echo ${#names[*]}
3
lex@lex-vm:~/lesson/lesson9$ echo ${#names}
4
lex@lex-vm:~/lesson/lesson9$ echo ${#names[@]}
3
lex@lex-vm:~/lesson/lesson9$ echo ${names[@]}
lex0 lex1 lex2
lex@lex-vm:~/lesson/lesson9$ declare -p names
declare -a names=([0]="lex0" [1]="lex1" [2]="lex2")
```

9.4.2 数组的赋值

在Shell中，用括号来表示数组，数组元素用“空格”符号分割开。

(1) 定义数组的一般形式为：

`array_name=(value0 ... valuen)` **连续赋值**

例如：`array_name=(value0 value1 value2 value3)` //此时下标从0开始，**中间用空格分开，不能用逗号，用逗号虽不出错，但是不是数组。**

(2) 还可以单独定义数组的各个分量，可以不使用连续的下标，而且下标的范围没有限制。

`array_name[0]=value0`

`array_name[2]=value2`

`array_name[4]=value4`

```
lex@lex-vm:~/lesson/lesson9$ array1=(1,2,3)
lex@lex-vm:~/lesson/lesson9$ echo $array1
1,2,3
lex@lex-vm:~/lesson/lesson9$ echo ${array1[0]}
1,2,3
lex@lex-vm:~/lesson/lesson9$ echo ${array1[1]}
lex@lex-vm:~/lesson/lesson9$ echo ${array1[2]}
```

(3) 对数组进行声明并赋值

`declare -a name=(a b c d e f)` //此时数组下标从0开始

`name[0]=A` //将第一个元素a修改为A

`name[9]=j` //将第10个元素赋值为j



9.4.3 数组的读取

读取数组元素值的一般格式是：

`${array_name[index]}`

(1) 取数组中的元素的时候，语法形式为：`echo ${array_name[index]}`

(2) 如果想要取数组的全部元素，则要使用：`echo ${array_name[@]}`
`echo ${array_name[*]}`

```
1,2,3
lex@lex-vm:~/lesson/lesson9$ echo ${array2[0]}
1
lex@lex-vm:~/lesson/lesson9$ echo ${array2[1]}
2
lex@lex-vm:~/lesson/lesson9$ echo ${array2[2]}
3
lex@lex-vm:~/lesson/lesson9$
```



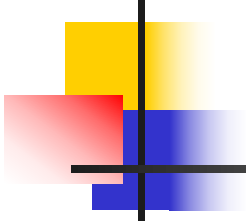
例1：给数组赋值，

```
#!/bin/sh  
NAME[0]="Zara"  
NAME[1]="Qadir"  
NAME[2]="Mahnaz"  
NAME[3]="Ayan"  
NAME[4]="Daisy"  
echo "First Index: ${NAME[0]}"  
echo "Second Index: ${NAME[1]}"
```

\$. /test.sh （结果）

First Index: Zara

Second Index: Qadir



例2：使用@ 或 * 可以获取数组中的所有元素，程序如下所示：

```
#!/bin/sh
NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
echo "First Method: ${NAME[*]}"
echo "Second Method: ${NAME[@]}"
```

输出结果如下。

```
$/test.sh
```

```
First Method: Zara Qadir Mahnaz Ayan Daisy
```

```
Second Method: Zara Qadir Mahnaz Ayan Daisy
```



9.4.4数组的长度

1、用`${#数组名[@]}` 或 `${#数组名[*]}`可以得到**数组长度**

格式: `length=${#array_name[@]}` 或 `length=${#array_name[*]}`

例如: 数组 `a=(1 2 3 4 5)` 输出长度的结果如图所示。

```
lex@lex-vm:~/lesson/lesson9$ a=(1 2 3 4 5)
lex@lex-vm:~/lesson/lesson9$ echo ${#a[*]}
5
```



2、用`${#数组名[n]}` 取得数组单个元素的长度

格式: `lengthn=${#array_name[n]}`

例如: 数组 `b=(two three)` 有两个元素, 分别输出2个元素的长度, 结果:

```
lex@lex-vm:~/lesson/lesson9$ b=(two three)
lex@lex-vm:~/lesson/lesson9$ echo ${#b}
3
lex@lex-vm:~/lesson/lesson9$ echo ${#b[0]}
3
lex@lex-vm:~/lesson/lesson9$ echo ${#b[1]}
5
```

思考题: `names=(lex wq lzh Rui wqiao)`, 写出以下语句输出结果

(1) `echo names`

(2) `echo $names`

(3) `echo ${#names}`

(4) `echo ${#names[*]}`

(5) `echo ${#names[3]}`

(6) 如果 `names=(lex,wq,lzh Rui,wqiao)`, 求以上输出结果。

9.5 shell的输入/输出

9.5.1 输入命令read

使用read语句从键盘或文件的某一行文本中读入信息，并将其赋给一个变量，**如果只指定了一个变量，那么read将会把所有的输入赋给该变量**，直到遇到第一个文件结束符或回车。一般形式为：

read variable1 variable2.....

- (1) shell用空格作为多个变量之间的分隔符
- (2) shell将输入文本域超长部分赋予最后一个变量

```
lex@lex-vm:~/lesson/lesson9$ read name sex age
lex male 35
lex@lex-vm:~/lesson/lesson9$ echo $name
lex
lex@lex-vm:~/lesson/lesson9$ echo $sex
male
lex@lex-vm:~/lesson/lesson9$ echo $age
35
lex@lex-vm:~/lesson/lesson9$ read name sex age
lex male 35 test
lex@lex-vm:~/lesson/lesson9$ echo $name
lex
lex@lex-vm:~/lesson/lesson9$ echo $sex
male
lex@lex-vm:~/lesson/lesson9$ echo $age
35 test
```

9.5 shell的输入/输出

9.5.1 输入命令read

选项:

-p 在等待read输入时, 输出提示信息;

-t 指定等用户输入的时间(单位为秒);

-n 指定输入的字符数后便执行;

-s 隐藏输入的数据, 适用于加密信息的输入;

```
lex@lex-vm:~/lesson/lesson9$ bash -v readptsn.sh
#!/bin/bash
read -p "please input something to test -p:" sth
please input something to test -p:lex
echo ${sth} "is what you input"
lex is what you input

read -t 10 -p "please input something to test -t:\n" passwd
please input something to test -t:\nlex
echo $passwd
lex

read -s -p "please input something to test -s:\n" passwd
please input something to test -s:\nlex echo $passwd
lex

read -n 2 -p "please input your age to test -n:\n" age
please input your age to test -n:\n23echo $age
23
```

9.5.2 输出命令echo

使用echo可以输出文本或变量到标准输出，或者把字符串输入到文件中，它的一般形式为：echo [选项] 字符串。

选项：**-n**：输出后不自动换行；

-e：启用“\”字符的转换。

- \a 发出警告声
- \b 删除前一个字符
- \c 最后不加上换行符号，并忽略后面的字符。
- \f 换行但光标仍旧停留在原来的位置
- **\n 换行且光标移至行首**
- \r 光标移至行首，但不换行
- \t 插入tab
- \v 与\f相同
- **\\ 插入\字符**
- \x 插入十六进制数所代表的ASCII字符

```
lex@lex-vm:~/lesson/lesson9$ echo -e "hello\aworld"
helloworld
lex@lex-vm:~/lesson/lesson9$ echo -e "hello\bworld"
hellworld
lex@lex-vm:~/lesson/lesson9$ echo -e "hello\cworld"
hellolex@lex-vm:~/lesson/lesson9$ echo -e "hello\fworld"
hello
    world
lex@lex-vm:~/lesson/lesson9$ echo -e "hello\nworld"
hello
world
lex@lex-vm:~/lesson/lesson9$ echo -e "hello\rworld"
world
lex@lex-vm:~/lesson/lesson9$ echo -e "hello\\world"
hello\world
```



例1: \t（插入tab）和\n（换行且光标到行首）的应用

```
lex@lex-vm:~/lesson/lesson9$ echo -e "a\tb\tc\nd\tf\ng\ti"
```

a	b	c
d	e	f
g	h	i

例2: \x的应用，（插入十六进制数所代表的ASCII字符）

```
lex@lex-vm:~/lesson/lesson9$ echo -e "\x61\x09\x62"
```

> "

a	b
---	---

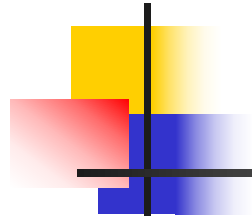
9.6运算符和特殊字符

9.6.1运算符

•Shell 拥有自己的运算符，shell 的运算符及优先级的结合方式如表所示

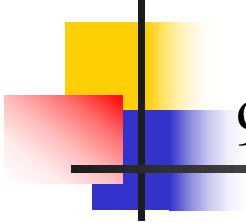
运算符	解释	结合方式
()	括号(函数等)	→
[]	数组	→
! ~	取反 按位取反	→
++ --	增量 减量	→
+ -	正号 负号	→
* / %	乘法 除法 取模	→

+ -	加法 减法	→
<< >>	左移 右移	→
< <=	小于 小于等于	→
>= >	大于 大于等于	→
== !=	等于 不等于	→
&	按位与	→
^	按位异或	→
	按位或	→
&&	逻辑与	→
	逻辑或	→
?:	条件	←
= += *= /=	赋值	←
^= = <<= >>=	赋值	←



例：创建/home/lex/lesson/lesson9/cal 目录后，在此目录下建立文件test_cal

```
lex@lex-vm:~/lesson/lesson9$ mkdir ./cal && touch ./cal/test_cal
lex@lex-vm:~/lesson/lesson9$ ls
cal
lex@lex-vm:~/lesson/lesson9$ cd cal
lex@lex-vm:~/lesson/lesson9/cal$ ls
test_cal
```



9.6.2. 特殊字符

(1) 反斜线 (\)

- 反斜线是转义字符，它告诉Shell不要对其后面的那个字符进行特殊处理，只当做普通字符即可。

(2) 双引号 (“”)

- 由双引号括起来的字符，\$、反斜线和反引号几个字符仍是特殊字符并保留其特殊功能外，其余字符仍视为普通字符。

(3) 单引号 (‘)

- 由单引号括起来的字符都作为普通字符出现。

(4) 反引号 (`)

- Shell把反引号括起来的字符串解释为命令行后首先执行，并以它的标准输出结果取代整个反引号部分

(5) 注释符

- 在shell中以字符#开头的正文行表示注释行。



9.7 Shell语句

使用shell脚本编程时，可以使用if语句、case语句、for语句、while语句和until语句等，对程序的流程进行控制。

9.7.1 test命令

test命令用于检查某个条件是否成立，如果条件为真，则返回一个0值。如果表达式不为真，则返回一个大于0的值，也可以将其称为假值。

格式为：**test expression**或者 **[expression]**

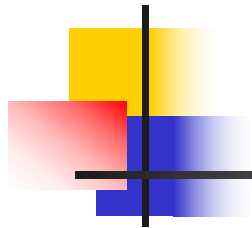
表达式一般是字符串、整数或文件和目录属性，并且可以包含相关的运算符。运算符可以是字符串运算符、整数运算符、文件运算符或布尔运算符。

1. 整数运算符

■ test命令中，用于比较整数的关系运算符如表所示：

运算符	解释
-eq	两数值相等（equal）
-ne	两数值不等（not equal）
-gt	n1大于n2（greater than）
-lt	n1小于n2（less than）
-ge	n1大于等于n2（greater than or equal）
-le	n1小于等于n2（less than or equal）

```
lex@lex-vm:~/lesson/lesson9/cal$ test 10 -eq 10
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
0
```



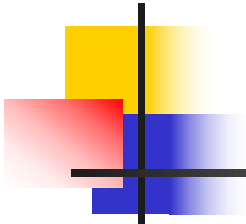
2. 字符串运算符

■ 用于字符串判断时，test的关系运算符如表所示

运算符	解释
<code>-z string</code>	判断字符串string是否为0，若string为空字符串，则为true
<code>-n string</code>	判断字符串string是否为非0，若string为非空字符串，则为true
<code>str1 = str2</code>	判断两个字符串str1和 str2是否相等，若相等，则为true
<code>str1 != str2</code>	判断两个字符串str1和 str2是否不相等，若不相等，则为true

如果条件为真，返回一个0值。如果表达式不为真，则返回一个大于0的值，也可以将其称为假值。

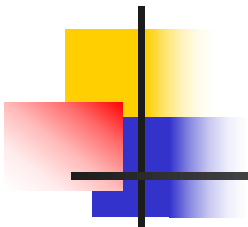
```
1
lex@lex-vm:~/lesson/lesson9/cal$ n1=tom
lex@lex-vm:~/lesson/lesson9/cal$ n2=lucy
lex@lex-vm:~/lesson/lesson9/cal$ test $n1 = $n2
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
1
lex@lex-vm:~/lesson/lesson9/cal$ test $n1 != $n2
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
0
```



3. 文件运算符

■用于文件和目录属性比较时，`test` 的运算符如表所示

运算符	解 释
<code>-e file</code>	判断file文件名是否存在
<code>-f file</code>	判断file文件名是否存在且为文件
<code>-d file</code>	判断file文件名是否存在且为目录（directory）
<code>-b file</code>	判断file文件名是否存在且为一个block device
<code>-c file</code>	判断file文件名是否存在且为一个character device
<code>-S file</code>	判断file文件名是否存在且为一个Socket
<code>-P file</code>	判断file文件名是否存在且为一个FIFO（pipe）管道



-L file	判断file文件名是否存在且为一个连接文件
-r file	判断file文件名是否存在且具有“可读”权限
-w file	判断file文件名是否存在且具有“可写”权限
-x file	判断file文件名是否存在且具有“可执行”权限
-u file	判断file文件名是否存在且具有“SUID”属性
-g file	判断file文件名是否存在且具有“SGID”属性
-k file	判断file文件名是否存在且具有“Sticky bit”属性
-s file	判断file文件名是否存在且为“非空白文件”
file1 -nt file2	判断file1是否比file2新(newer than)
file1 -ot file2	判断file1是否比file2旧(older than)
file1 -ef file2	判断file1与file2是否为同一文件



例：判断文件是否存在，并查看返回值情况，如图所示。

```
lex@lex-vm:~/lesson/lesson9/cal$ ls
test_cal
lex@lex-vm:~/lesson/lesson9/cal$ test -e abc
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
1
lex@lex-vm:~/lesson/lesson9/cal$ touch abc
lex@lex-vm:~/lesson/lesson9/cal$ ls
abc  test_cal
lex@lex-vm:~/lesson/lesson9/cal$ test -e abc
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
0
```

4. 逻辑运算符

- test 命令的逻辑运算符如表

运算符	解释
-a	逻辑与
-o	逻辑或
!	逻辑非

例：判断\$num的值是否在10和20之间

```
0
lex@lex-vm:~/lesson/lesson9/cal$ num=9
lex@lex-vm:~/lesson/lesson9/cal$ test "$num" -gt 10
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
1
lex@lex-vm:~/lesson/lesson9/cal$ test "$num" -lt 10
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
0
lex@lex-vm:~/lesson/lesson9/cal$ num=19
lex@lex-vm:~/lesson/lesson9/cal$ test "$num" -gt 10 -a "$num" -lt 20
lex@lex-vm:~/lesson/lesson9/cal$ echo $?
0
```




Linux操作系统

单位：杭州电子科技大学
通信工程学院



9.7 Shell语句

Shell编程流程控制语句

■ 条件判断语句：

- if语句
- case语句

■ 循环语句：

- for语句
- while语句
- until语句

9.7.2 if语句

if语句的结构分为：单分支if语句、双分支if语句和多分支if语句。

1.单分支if语句

只判断指定的条件，当条件成立时执行相应的操作，否则不做任何操作。

格式为：

if [command]

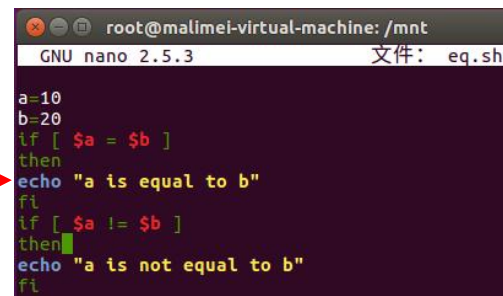
then

命令序列

fi

必须在左括号的右侧和右括号的左侧各加一个空格.否则会报错。

例：用单分支if语句判断两个数a和b是否相等，如果相等输出a is equal to b,如果不等，输出“a is not equal to b”。



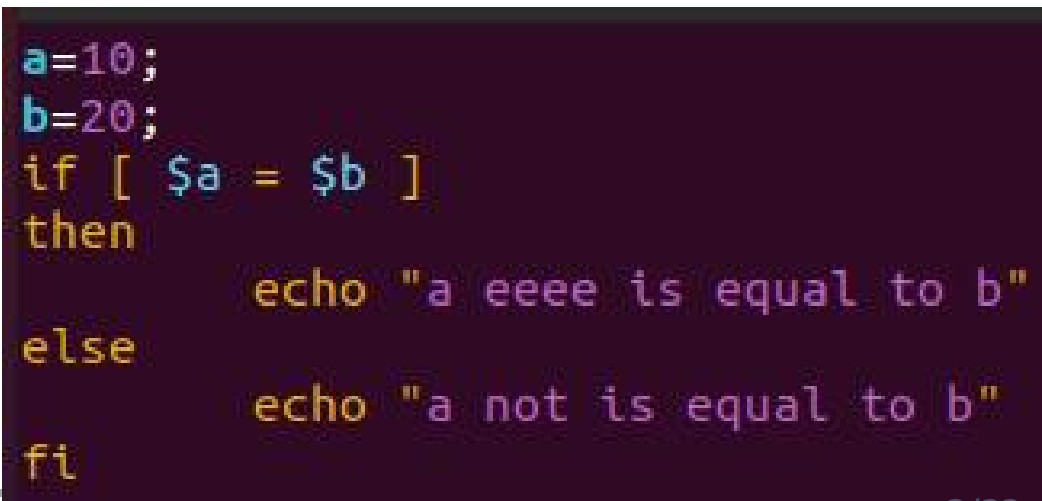
```
root@malimei-virtual-machine: /mnt
GNU nano 2.5.3 文件: eq.sh

a=10
b=20
if [ $a = $b ]
then
echo "a is equal to b"
fi
if [ $a != $b ]
then
echo "a is not equal to b"
fi
```

注意：格式

shell脚本对空格有严格的规定：

- 赋值语句等号两边不能有空格
- 字符串比较的等号两边必须有空格。



```
a=10;
b=20;
if [ $a = $b ]
then
echo "a is equal to b"
else
echo "a is not equal to b"
fi
```

2.双分支if语句

双分支的if语句在条件成立或不成立的时候分别执行不同的命令序列。格式为：

```
if [ command ]
```

```
then
```

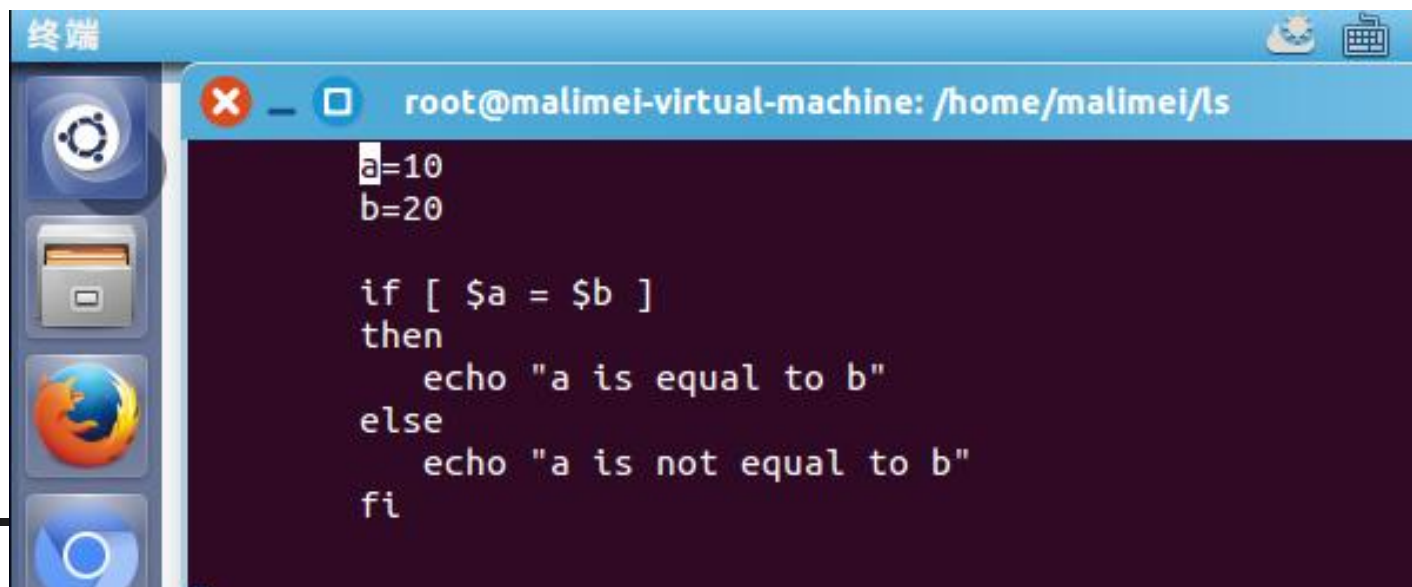
```
命令序列1
```

```
else
```

```
命令序列2
```

```
fi
```

例：双分支if语句判断两个数a和b是否相等，如果相等输出a is equal to b,如果不等，输出“a is not equal to b”

A terminal window titled "终端" (Terminal) with a blue header bar. The header bar contains a red close button, a blue maximize button, and a white icon. The terminal text shows a root user at a machine named "malimei-virtual-machine" in the directory "/home/malimei/ls". The script sets a=10 and b=20, then uses an if statement to check if a equals b. Since they are not equal, it prints "a is not equal to b".

```
root@malimei-virtual-machine: /home/malimei/ls
a=10
b=20

if [ $a = $b ]
then
    echo "a is equal to b"
else
    echo "a is not equal to b"
fi
```

3. 多分支if语句

在shell脚本中，if语句能够嵌套使用，进行多次判断。格式为：

```
if [ command1 ]
```

```
then
```

```
    命令序列1
```

```
elif [ command2 ]
```

```
then
```

```
    命令序列2
```

```
else
```

```
    命令序列3
```

```
fi
```

elif=else if

例：多分支if语句判断两个数a和b是否相等，如果相等输出a is equal to b,如果不等，判断大小。

```
a=10;
b=20;
if [ $a = $b ]
then
    echo "a is equal to b"
elif [ $a -gt $b ]
then
    echo "a is greater than b"
elif [ $a -lt $b ]
then
    echo "a is less than b"
else
    echo "none of the conditon met"
fi
```



9.7.3 case语句

case ... esac 与其他语言中的 switch ... case 语句类似，是一种多分支选择结构，case 语句匹配一个值或一个模式，如果匹配成功，执行相匹配的命令。case语句格式如下：

```
case $变量名 in
```

```
模式1)
```

```
命令序列1
```

```
;;
```

```
模式2)
```

```
命令序列2
```

```
;;
```

```
*)
```

```
默认执行的命令序列
```

```
esac
```

说明:

- (1) case行尾必须为“in”
- (2) 每一个模式必须以右括号“)”结束
- (3) 两个分号“;;”表示命令序列结束
- (4) 匹配模式中可使用方括号表示一个连续的范围，如[0-9]
- (5) 使用竖杠符号“|”表示或。
- (6) 最后的“*)”表示默认模式，当使用前面的各种模式均无法匹配该变量时，将执行“*)”

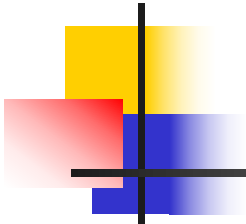
```
echo 'Input a student number between 1 to 4'
echo 'Your number is:\c'
read num
case $num in
    1) echo 'you select 1'
        display girl1.jpeg
        close girl1.jpeg
        ;;
    2) echo 'you select 2'
        display girl2.jpeg
        ;;
    3) echo 'you select 3'
        display girl3.jpeg
        ;;
    4) echo 'you select 4'
        display girl4.jpeg
        ;;
    *) echo 'you do not select a number between 1 to 4'
        ;;
esac
```


例:输入大小写的yes, 都输出YES;输入大小写的no, 都输出no.

```
#!/bin/sh
echo "please input \"yes\" or \"no\":"
read var
case "$var" in
    [yY][eE][sS]) echo "your input is YES";;
    [nN][oO])      echo "your input is no";;
    *)             echo "input error!";;
esac
exit 0
```

```
lex@lex-vm:~/lesson/lesson10$ sh yesno.sh
please input "yes" or "no":
yEs
your input is YES
lex@lex-vm:~/lesson/lesson10$ sh yesno.sh
please input "yes" or "no":
nO
your input is no
lex@lex-vm:~/lesson/lesson10$ sh yesno.sh
please input "yes" or "no":
yESS
input error!
```

注意:\和不加\的区别



方括号表示一个连续的范围，如[0-9]
这是[a-d]、[A-D]

```
#!/bin/sh
echo "please input "a-d" or " A-D""
read var
case "$var" in
    [a-d]) echo "your input is $var";;
    [A-D]) echo "your input is $var";;
    *)      echo "Input Error!";;
esac
exit 0
```



使用竖杠符号 “|”表示或。

范围[]加竖线”|”，表示在范围中或的条件。

```
#!/bin/bash
echo "please input a|b|A|B"
read var
case "$var" in
    a|b|A|B) echo "a or b have been input ...";;
    *) echo "others have been input ...";;
esac
```

```
lex@lex-vm:~/lesson/lesson10$ sh huotest.sh
please input a|b|A|B
A
a or b have been input ...
lex@lex-vm:~/lesson/lesson10$
```



9.7.4 while 语句

while 语句是shell提供的一种循环机制，当条件为真的时候它允许循环体中的命令继续执行，否则退出循环。

语句格式：

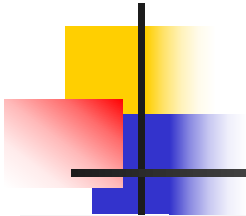
```
while [ 条件测试命令 ]  
do  
    命令序列  
done
```



例：编写脚本，输入整数n，计算1到n的和。

```
#!/bin/bash
read -p "please input a number:" n
sum=0;
i=1;
echo "i=$i" "n=$n";
echo $i+$n
while [ $i -le $n ]
do
    #sum=${sum+$i};
    sum=$((sum+$i));
    echo "i= $i", "sum=$sum";
    #i=${i+1};
    #i=$((i+1))
    #i=`expr $i + 1`
    #let i++
    let i+=1
done
echo "the sum of ' 1+2+3+..._+n' is $sum "
```

- `$[]` `$()` :它们是一样的，都是进行数学运算的。
- 支持`+` `-` `*` `/` `%` : 分别为“加、减、乘、除、取模”。
- 自增`i++`方法：图中5种.



`$(())`可以将其他进制转成十进制数显示出来。

用法如下：

`echo $((N#xx))`

其中，`N`为进制，`xx`为该进制下某个数值，命令执行后可以得到该进制数转成十进制后的值。

```
lex@lex-vm:~/lesson/lesson10$ echo $((2#111))
7
lex@lex-vm:~/lesson/lesson10$ echo $((2#111))
7
lex@lex-vm:~/lesson/lesson10$ echo $((16#2a))
42
lex@lex-vm:~/lesson/lesson10$ echo $((8#11))
9
```

利用while循环计算1到100之间所有偶数之和

```
#!/bin/bash
i=2
sum=0
while [ $i -le 100 ]
do
    #let sum=sum+i
    let sum=$sum+$i
    let i+=2
done
echo $sum
```

加上\$也可以

let 命令是 bash 中用于计算的工具，用于执行一个或多个表达式，变量计算中可不需要加上 \$ 来表示变量。如果表达式中包含了空格或其他特殊字符，则必须引起来。

```
lex@lex-vm:~/lesson/lesson10$ bash while2.sh
2550
```


利用循环，让用户做选择，根据客户的选择显示相应结果。

```
while true
do
  echo "*****"
  echo "    menu    "
  echo "1.time and date"
  echo "2.system info"
  echo "3.uesrs are doing"
  echo "4.exit"
  echo "*****"
  read -p "enter you choice [1-4]:" choice
#根据客户的选择做相应的操作
  case $choice in
    1)
      echo "today is `date +%Y-%m-%d`"
      echo "time is `date +%H:%M:%S`"
      read -p "press [enter] key to continue..." Key #暂停循环，提示客户按enter键
      ;;
    2)
      uname -r
      read -p "press [enter] key to continue..." Key
      ;;
    3)
      w
      read -p "press [enter] key to continue..." Key
      ;;
    4)
      echo "Bye!"
  esac
done
```

w命令是一个在linux系统中用来显示当前登录用户及这些用户正在做什么的命令。它同时也能显示系统运行时长，当前系统时间和系统平均负载情况。



9.7.5 until语句

until语句是当条件满足时退出循环，否则执行循环，语句格式为：

```
until [ 条件测试命令 ]
```

```
do
```

```
    命令序列
```

```
done
```

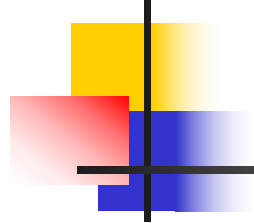
例：循环输出1到10的数字。

Until”语句提供了与 “while” 语句相反的功能：只要特定条件为假，就重复执行语句。

输出1到10，条件是大于10时，就退出循环，不大于10，就输出。

```
var=1
until [ $var -gt 10 ]
do
    echo $var
    var=$(( $var +1 ))
done
```

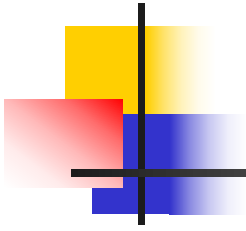
```
lex@lex-vm:~/lesson/lesson10$ sh until1.sh
1
2
3
4
5
6
7
8
9
10
```



不同符号的异同

➤ `$()`: 用于给变量赋值。（必须是命令，不能是普通的变量值）

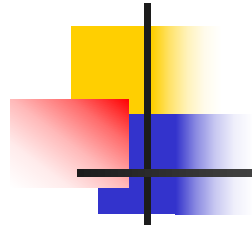
```
lex@lex-vm:~/lesson/lesson10$ hello=$(hostname)
lex@lex-vm:~/lesson/lesson10$ echo $hello
lex-vm
lex@lex-vm:~/lesson/lesson10$ hello=$(ls)
lex@lex-vm:~/lesson/lesson10$ echo $hello
az.sh case_test.sh girl1.jpeg girl2.jpeg girl3.jpeg girl4.
jpeg huotest.sh if.sh lsix-master master.zip mulif.sh unti
l1.sh while1.sh while2.sh while3.sh yesno.sh
lex@lex-vm:~/lesson/lesson10$ hello=$(2)
2: command not found
```

- 
- ``符号也是用于命令交换的，和\$() 的操作是一样的。

```
lex@lex-vm:~/lesson/lesson10$ hello=`hostname`  
lex@lex-vm:~/lesson/lesson10$ echo $hello  
lex-vm
```

- \${变量名}: 用于变量替换 和\$变量名一样

```
lex@lex-vm:~/lesson/lesson10$ hello=hi  
lex@lex-vm:~/lesson/lesson10$ echo ${hello}  
hi  
lex@lex-vm:~/lesson/lesson10$ echo $hello  
hi
```



➤ `$[]` (1) 用于算术计算，注意里面出现的只有数字

```
lex@lex-vm:~/lesson/lesson10$ echo $[RANDOM]
10361
lex@lex-vm:~/lesson/lesson10$ echo $[RANDOM]
14441
lex@lex-vm:~/lesson/lesson10$ echo $[RANDOM%100+1]
99
lex@lex-vm:~/lesson/lesson10$ echo $[RANDOM%100+1]
1
lex@lex-vm:~/lesson/lesson10$ echo $[RANDOM%100+1]
56
```



9.7.6 for语句

for语句的格式:

for 变量名 in 取值列表

do

命令序列

done

使用for 循环时，可以为变量设置一个取值列表，每次读取列表中不同的变量值并执行命令序列，变量值用完后退出循环。



例：使用for语句创建命令行上所有整数之和。

```
#!/bin/bash
sum=0
for INT in $*
do
    sum=$(( $sum + $INT ))
    echo "sum= $sum"
    #let sum+=INT
done
echo $sum
```

```
lex@lex-vm:~/lesson/lesson10$ bash fortest.sh 1 3 4 5
sum= 1
sum= 4
sum= 8
sum= 13
13
```


9.7.7 循环控制语句

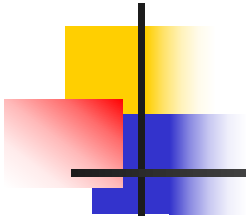
1. break语句

break语句用于for、while和until循环语句中，忽略循环体中任何其他语句和循环条件的限制，强行退出循环。

例：输入整数n，我们最多只计算1到10的和。

```
#!/bin/bash
read -p "please input a number:" n
sum=0
i=1
for i in `seq 1 $n`
do
    if [ $i -gt 10 ]
    then
        break
    fi
    sum=$((sum+i))
    i=$((i+1))
done
echo "the sum of '1+2+3+...n' is $sum"
```

```
lex@lex-vm:~/lesson/lesson10$ sh breaktest.sh
please input a number:13
the sum of '1+2+3+...n' is 55
```



seq命令用于以指定增量从首数开始显示数字到尾数，即产生从某个数到另外一个数之间的所有整数，并且可以对整数的格式、宽度、分割符号进行控制。

语法：

- [1] seq [选项] 尾数
- [2] seq [选项] 首数 尾数
- [3] seq [选项] 首数 增量 尾数

```
lex@lex-vm:~/lesson/lesson10$ seq 3
1
2
3
lex@lex-vm:~/lesson/lesson10$ seq -2 3
-2
1
4
lex@lex-vm:~/lesson/lesson10$ seq 10 -2 0
10
8
6
4
2
0
```

```
#!/bin/bash
read -p "please input a number:" n
sum=0
i=1
for i in `seq 1 $n`
do
    if [ $i -gt 10 ]
    then
        break
    fi
    sum=$((sum+$i))
    i=$((i+1))
done
echo "the sum of '1+2+3+...n' is $sum"
```

注意：in 后面的表达方式。

```
#!/bin/bash
read -p "please input a number:" n
sum=0
i=1
#for i in `seq 1 $n`
for i in $(seq 1 $n)
do
    if [ $i -gt 10 ]
    then
        break
    fi
    sum=$((sum+i))
    i=$((i+1))
done
echo "the sum of '1+2+3+...n' is $sum"
```



2. continue语句

continue语句应用在for、while和until语句中，用于让脚本跳过其后面的语句，执行下一次循环。

例：编写脚本，输入整数n，计算1到n的奇数和。

```
read -p "please input a number:" n
sum=0
i=1
for i in `seq 1 $n`
do
    if [ ${i%2} -eq 0 ]
    then
        i=$((i+1)) #如果是偶数，加1
        continue  #//跳出本次循环，执行下一次循环
    fi
    sum=$((sum+i)) #如果是奇数，就累加
    i=$((i+1))    #//i再加1
done
echo "the sum of odd number between 1 and $n is $sum"
```

```
lex@lex-vm:~/lesson/lesson10$ bash continuetest.sh
please input a number:30
the sum of odd number between 1 and 30 is 225
```

9.8 综合应用

9.8.1 综合应用一

例：编写shell脚本，执行后，显示一行提示“Please input a number:”，逐次打印用户输入的数值，直到用户输入“end”为止。

```
#!/bin/sh
unset var
while [ "$var" != "end" ]
do
    echo -n "please input a number: "
    read var
    if [ "$var" = "end" ]
    then
        break;
    fi
    echo "var is $var"
done
```

```
lex@lex-vm:~/lesson/lesson10$ sh input.sh
please input a number: 23l
var is 23l
please input a number: fads
var is fads
please input a number: 43
var is 43
please input a number: t
var is t
please input a number:
var is
please input a number: r
var is r
please input a number: t
var is t
please input a number: ewqtr
var is ewqtr
please input a number: end
```

9.8.2综合应用二

例：编写shell脚本，使用ping命令检测192.168.0.100~192.168.0.105的主机目前哪些主机开机，哪些主机关机

```
#!/bin/bash
network="192.168.0"
for sitenu in $(seq 100 105)
do
    ping -c 1 -w 1 ${network}.${sitenu} && result=0 || result=1
    if [ "$result" == 0 ]
    then
        echo "Server ${network}.${sitenu} is UP."
    else
        echo "Server ${network}.${sitenu} is DOWN."
    fi
done
```

说明：

- (1) -c count是数量，即发ping包的数量。
- (2) -w timeout——指定超时间隔，单位为毫秒。
- (3) **&>把标准错误及标准输出中的信息都重定向，这里面是warninginfo**，表示标准输出重定向到空设备文件，执行完输出的东西都放入warninginfo。不管出错了，还是正常ping通了，只要是输出到屏幕的信息都放入warninginfo里。

9.8.3综合应用三

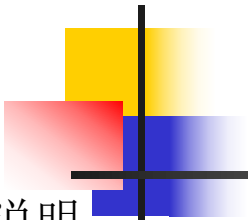
例：编写shell脚本，提示输入某个目录文件名，然后输出此目录内所有文件的权限，若可读输出readable，若可写输出writable、若可执行输出executable。

```
#!/bin/bash
read -p "please input a directory:" dir
if [ "$dir" == " " -o ! -d "$dir" ]
then
    echo "The $dir does not exist in your system"
    exit 0
fi
filelist=$(ls $dir)
for filename in $filelist
do
    perm=""
    test -r "$dir/$filename" && perm="$perm readable"
    test -w "$dir/$filename" && perm="$perm writable"
    test -x "$dir/$filename" && perm="$perm executable"
    echo "The file $dir/$filename's permission is $perm"
done
```

exit命令用于退出当前shell，
在shell脚本中可以终止当前脚本执行。

状态值0代表执行成功，其他
值代表执行失败退出。

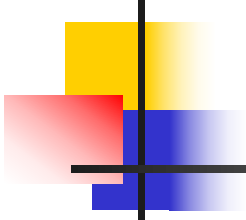
&& 表示前一条命令执行成功时，才执行后一条命令，如 echo '1' && echo '2'



说明:

if ["\$dir" == " "] || [! -d "\$dir"] 或者if ["\$dir" == " " -o ! -d "\$dir"], [! -d "\$dir"]是个条件判断, 其中[! -d]参数是使用 **test**指令的参数, 其中-d是测试目录是否存在, ! 返回一个结果为真的值。

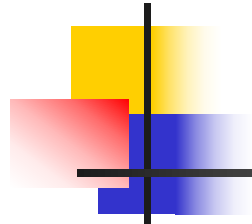
```
lex@lex-vm:~/lesson$ ls -l
total 24
drwxrwxr-x 3 lex lex 4096 5月 16 11:21 lesson10
drwxrwxr-x 2 lex lex 4096 5月 15 11:06 lesson11
drwxrwxr-x 2 lex lex 4096 5月 12 09:21 lesson12
drwxrwxr-x 2 lex lex 4096 5月 12 09:21 lesson13
drwxrwxr-x 2 lex lex 4096 5月 12 09:21 lesson14
-rw-rw-r-- 1 lex lex 132 5月 16 11:13 tmp
lex@lex-vm:~/lesson$ bash ./lesson10/rwxtest.sh
please input a directory:/home/lex/lesson
The file /home/lex/lesson/lesson10's permission is readable writable executable
The file /home/lex/lesson/lesson11's permission is readable writable executable
The file /home/lex/lesson/lesson12's permission is readable writable executable
The file /home/lex/lesson/lesson13's permission is readable writable executable
The file /home/lex/lesson/lesson14's permission is readable writable executable
The file /home/lex/lesson/tmp's permission is readable writable
```



例：编写脚本查看你当前目录下的文件属性(是普通文件还是目录)

```
#!/bin/bash
for i in *
do
    if [ -f $i ]
    then
        echo "$i is a file"
    else
        if [ -d $i ]
        then
            echo "$i is a directory"
        fi
    fi
done
```

```
lex@lex-vm:~/lesson/lesson10$ sh fd.sh
az.sh is a file
breaktest.sh is a file
breaktest1.sh is a file
case_test.sh is a file
continuetest.sh is a file
fd.sh is a file
fortest.sh is a file
girl1.jpeg is a file
girl2.jpeg is a file
girl3.jpeg is a file
girl4.jpeg is a file
huotest.sh is a file
if.sh is a file
input.sh is a file
lsix-master is a directory
master.zip is a file
mulif.sh is a file
pingtest.sh is a file
rxwtest.sh is a file
until1.sh is a file
warninginfo is a file
while1.sh is a file
while2.sh is a file
while3.sh is a file
yesno.sh is a file
```



根据文件名的前四位创建二级目录，前两位创建一级目录，后两位创建二级目录

```
#!/bin/sh
for dir in `cat jian`
do
    dir1=`echo $dir | cut -c1-2`
    dir2=`echo $dir | cut -c3-4`
    if [ ! -d "$dir1/$dir2" ] ;
    then
        mkdir -p "$dir1/$dir2"
    fi
done
```

```
lex@lex-vm:~/lesson/lesson10/ji$ ls -l
total 4
drwxrwxr-x 2 lex lex 4096 5月 16 11:36 an
```

cut命令用于显示每行从开头算起 num1 到 num2 的文字。

-c : 以字符为单位进行分割。



例：创建三个用户lex101,lex102,lex103；密码都是red

```
for name in lex101 lex102 lex103
do
    sudo useradd $name
    echo $name:red | chpasswd
done
```

chpasswd 命令：批量更新用户口令的工具，是把一个文件内容重新定向添加到/etc/shadow中。