Linux基础和shell编程

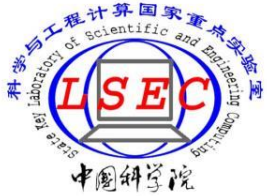
钱莹

qianying@lsec.cc.ac.cn

中国科学院数学与系统科学研究院

计算数学与科学工程计算研究所

科学与工程计算国家重点实验室



Outline

- Linux 基础
  - Linux常用命令
  - 编译环境
    - 编辑器vi
    - 编译调试环境
    - Make
- shell 编程
  - shell定义及访问方式
  - shell中的变量
  - shell流程控制
  - Shell数组及数组遍历
  - Shell中的各种运算符

Linux 基础

- 操作系统：Windows 系统 / Linux 或 Unix 系统
  - 原生系统
  - Win10+Ubuntu子系统
  - Windows+虚拟机
  - 远程登录：ssh
    - ✓ Xshell（家庭教育免费版）：https://www.netsarang.com/zh/free-for-home-school/
    - ✓ Putty：https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Linux 常用命令

- 系统维护及管理命令

  date——显示和设置系统日期和时间

  setenv——查询或设置环境变量(set environment variable)

  kill——发送一个 signal 给某一个 process

  at——在指定的时间执行指令

- 文件操作及管理命令

  ls——显示文件及目录

  find——查找文件

Linux 常用命令

- 磁盘及设备管理命令

  df——检查文件系统的磁盘空间占用情况(disk free)

  du——显示磁盘空间的使用情况(disk usage)

  mount——挂载设备

- 用户管理命令

  adduser——新增用户帐户

  userdel——删除用户帐号

- 文档操作命令

  csplit——分割文件(Split a file into context-determined pieces)

  sort——对文件中的各行进行排序

Linux 常用命令

- 网络通信命令

  netstat——显示网络连接、路由表和网络接口信息

  ifconfig——显示或设置网络设备

- 程序开发命令

  cc——c编译

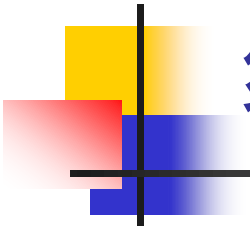
  link——链接

- X Window管理命令

  startx——启动X Window

Linux 常用命令

- 切换目录：cd
- 列出当前目录：ls
- 删除文件：rm –f (filename)
- 创建目录：mkdir (dirname)
- 删除目录：rm –rf (dirname)
- 查看文件内容：more, cat
- 文件拷贝：cp (src) (dest)
- 目录拷贝：cp –r (src) (dest)
- 文本编辑器：nano（类似于记事本），(g)vim，(x)emacs
- 帮助：man （命令名称）；info（命令名称）



编辑器 vi

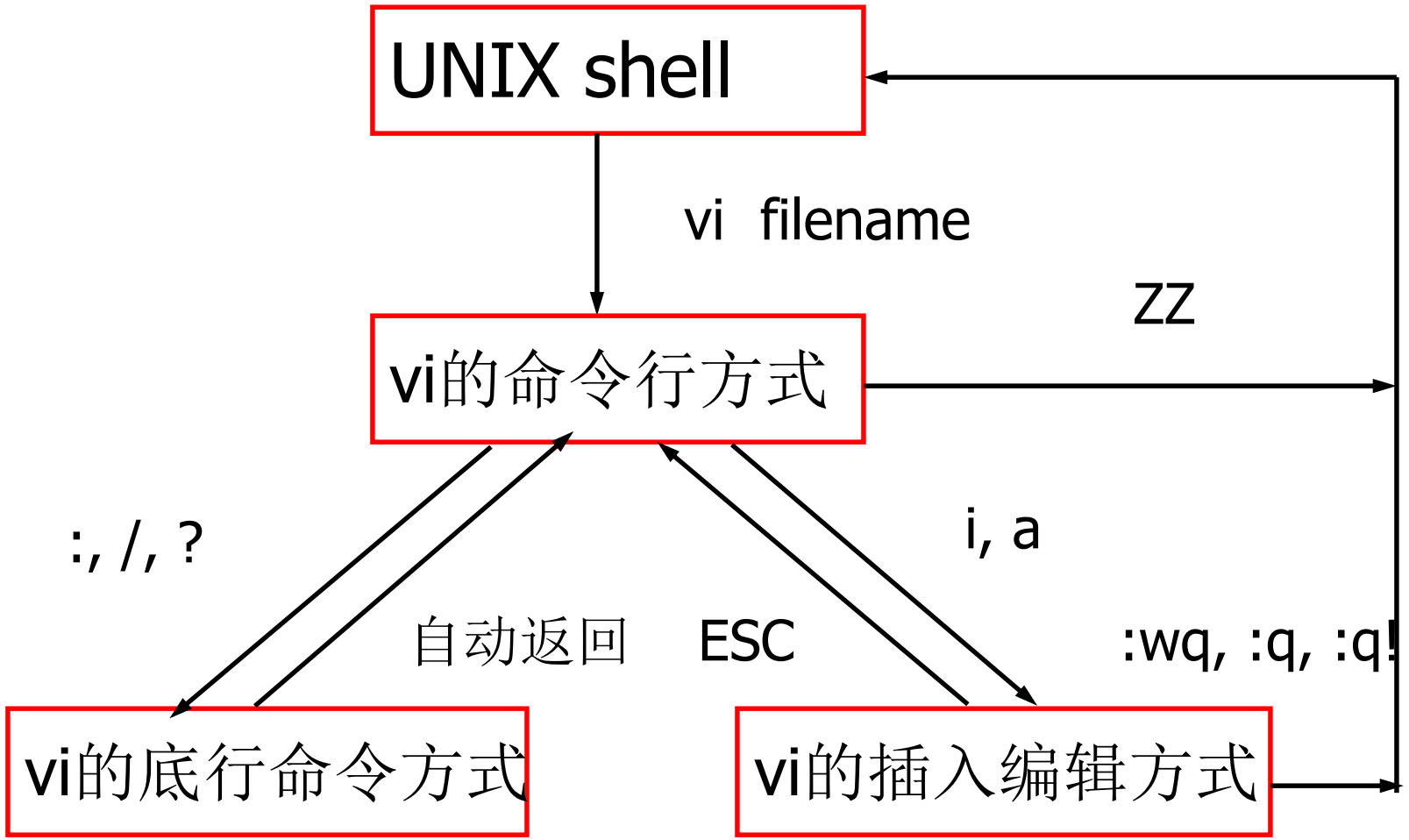
vi 基本概念

- 命令行模式(command mode): 控制屏幕光标的移动，字符的删除。
- 底行模式(last line mode): 将文件保存或退出vi,.....等。
- 插入模式(insert mode): 文字输入，按Esc键回到命令行模式。
- 可视模式(visual mode)

vi 基本操作

- 进入vi：vi myfile
- 切换至插入模式编辑文件：在命令行模式下按一下字母i
- 退出vi及保存文件：在命令行模式下，按一下冒号键进入底行模式
  - :w filename 将文章以指定的文件名(filename)保存
  - :wq (存盘并退出vi)
  - :q! (不存盘强制退出vi)

vi中多种工作方式的转换关系



命令行方式下常用命令

- h(←)——左移 l(→)——右移 k(↑)——上移 j(↓)——下移
- 0——光标移至行首
- $——光标移至行尾
- H——光标移至屏幕的最上行
- M——光标移至屏幕的中部
- L——光标移至屏幕的最下行
- G——光标移至文件最后一行行首
- nG——光标移至文件第n行行首
- dd——删除光标所在行
- 4dd——删除从光标所在行开始4行
- d0——删除光标至行首
- D——删除光标至行尾
- u——取消上一次操作
- .——重复上一次操作

底行命令方式下常用命令

- /string——从光标处向下寻找字符串string
- ?string——从光标处向上寻找字符串string
- :w——保存
- :w filename——另存为
- :w! filename——强行写盘
- :q——退出vi
- :q!——强行退出vi
- :wq——写盘后退出vi
- :r filename——将文件读入编辑缓冲区
- :e filename——打开并编辑文件
- :! shellcmd——在vi中执行shell命令

Linux常用编译系统

- 编译器由前端和后端组成。通常用户只需使用前端命令即可完成编译、链接。
- C编译器：cc，gcc（GNU C）等。
- Fortran编译器：f77，fc，g77（GNU Fortran），f90（Fortran 90）等。
- 可用man查看使用手册，如：man cc，man f77等。
- 命令行形式：

  $cc [options] files [options]

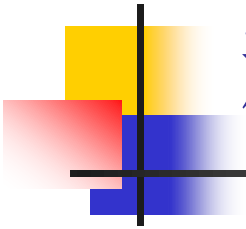
  $f77 [options] files [options]

Linux常用编译系统

- 文件类型由文件扩展名决定：
  - C源代码：.c;
  - Fortran 77源代码：.f;
  - 带预处理的Fortran源代码：.F;
  - C++源代码：.c++, .c, .cpp, .cc, .cxx;
  - 汇编代码：.s, .S;
  - 目标文件：.o;
  - 库文件：.a;
  - 共享库：.so;

Linux常用编译系统

- 命令行选项：
  - -c: 只编译，不链接，即只生成.o文件。
  - -o filename: 指定输出文件名，缺省为*.o, a.out等。
  - -Ipath: 指定（增加）包含文件（如*.h）的搜索目录。
  - -Lpath: 指定（增加）库文件的搜索目录。
  - -lname: 与库文件libname.a(.so)链接。
  - 优化开关：-O，-O1，-O2，-O3，等等。
  - -g：目标码中包含源文件名、行号等信息（用于程序调试）
- 例：
  - f77 –O2 –o prog file1.f file2.c file3.o file4.a
  - f77 –c file.f
    f77 –o out file.o
  - f77 –c –I/usr/local/mpi/include file.f
  - f77 –o prog –L/usr/local/mpi/lib file.o –lmpi (等价于
  - f77 –o prog file.o /usr/local/mpi/lib/libmpi.a)



程序调试

GDB

GDB是GNU开源组织发布的强大的UNIX下的程序调试工具。一般来说，GDB主要帮忙你完成下面四个方面的功能：

- 启动你的程序，可以按照你的自定义的要求随心所欲的运行程序。
- 可让被调试的程序在你所指定的断点处停住。
- 当程序被停住时，可以检查此时你的程序中所发生的事。
- 动态的改变你程序的执行环境。

valgrind

valgrind是在linux系统下开发应用程序时用于调试内存问题的工具。

```
valgrind –tool=memcheck –leak-check=yes  \
        --show-reachable=yes –run-libc-freeres=yes  \
        ./yourprogram
```

实用工具Make

- 命令形式

```
make [-f Makefile] [options] [target [target …] ]
```

其中-f选项给出定义规则的文件名（简称Makefile文件），缺省使用当前目录下的Makefile或makefile文件。Target指明要求生成的目标（在Makefile中定义），当命令行中不给出target时make只生成Makefile中定义的第一个目标。比较有用的命令行选项有下面这些：

-f 文件名：指定Makefile文件名。

-n：只显示将要执行的命令而并不执行它们。

-p：显示定义的全部规则及宏，用于对Makefile的调试。

- 通过Makefile文件定义一组文件之间的依赖关系及处理命令，方便程序开发过程中的编译与维护。
- 处理规则的建立以特定的文件扩展名及文件修改时间为基础。缺省支持常用的程序扩展名：.c，.f，.F，.o，.a，.h等等。用户可以通过.SUFFIXES: 目标定义新的文件扩展名。

实用工具Make

- 基本规则：

  目标：依赖对象

  <tab> 处理命令

  <tab>… …

例：

```
prog：file1.f file2.f file3.o
        f77 –O2 –o prog file1.f file2.f file3.o
```

其含义为：如果目标（prog）不存在，或者任何一个依赖对象（file1.f，file2.f，file3.o）比目标新，则执行指定的命令。

实用工具Make

- 宏定义：

```
SRC=file1.f file2.f file3.c
prog: $(SRC) |
        f77 –O2 –o prog $(SRC)
```

环境变量可以在Makefile中作为宏使用，如$(HOME)。

- 常用预定义的宏：
  - $@: 代表目标名（上例中为prog）
  - $<: 第一个依赖对象名（上例中为file1.f）
  - $^: 全部依赖对象名（上例中为file1.f file2.f file3.c）
  - $?: 全部比目标新的依赖对象名
  - $*: 用在隐式规则中，代表不含扩展名的依赖对象

实用工具Make

- 隐式规则：

```
prog: file1.o file2.o file3.o
        f77 –O2 –o prog $?
.c.o:
        cc –O2 –c $*.c
 .f.o:
        f77 –O2 –c $*.f
```

Makefile 示例

```
CC        = mpicc
FFLAG  = -g
CFLAG  =-g

.f.o:
        $(MPIF&&) $(FFLAG) –c $*.f

.c.o:
        $(CC) $(CFLAG) –c $*.c

sendrecv:  sendrecv.c
        $(CC) $(CFLAG) –o $@ $<

clean:
        rm –f *~ *.o a.out
```

什么是shell

shell是一个命令解释器，它在操作系统的最外层，负责直接与用户对话，把用户的输入解释给操作系统，并处理各种各样的操作系统的输出结果，输出屏幕返回给用户。

shell对话方式

- 交互的方式：从键盘输入命令，通过/bin/bash的解析，可以立即得到Shell的回应，一问一答的方式
- 非交互式：shell脚本

  shell脚本的编写规范：开头第一行为：#!/bin/bash 或者 #!/bin/sh。这是因为#!又称为幻数，在执行bash脚本的时候，内核会根据它来确定该用哪一个程序来解释脚本中的内容，这一行必须在脚本的顶端的第一行。

shell预定义变量

- **$#**：位置参数的数量
- **$***：所有位置参数的内容
- **$?**：命令执行后返回的状态
- **$$**：当前进程的进程号
- **$!**：后台运行的最后一个进程号
- **$0**：当前执行的进程名

其中，“**$?**”用于检查上一个命令执行是否正确(在**Linux**中，命令退出状态为**0**表示该命令正确执行，任何非**0**值表示命令出错)。

shell中的变量

- 环境变量

  用于定义shell的运行环境，保证shell的正确执行。所有的环境变量都是系统的全局变量可以用于所有子进程中，包括编辑器、shell脚本和各类应用。

  - $HOME　　用户的家目录
  - $USER　　当前用户
  - $UID　　当前用户的uid
  - $SHELL　　当前用户使用的shell
  - $HISTSIZE　　记录在命令行历史文件中的命令行数
  - $PATH　　执行命令时寻找的目录
  - $PWD　　当前用户的家目录

shell中的变量

- 局部变量

  局部变量也称为本地变量，在用户当前的shell中使用，如果退出shell，则失效。

  例：

  变量名=value（数字、字母、下划线组成）

  - **单引号**：所见即所得，单引号内的内容原样输出。
  - **双引号**：如果内容中有变量等，会将变量解析出来，然后将最终的结果打印
  - **不加引号**：把内容输出出来，如果有连续的空格会将空格合并为一个空格，然后将变量等解析出来再输出。
  - **反引号**：把字符串当做命令去执行。

```
#!/bin/sh
your_name="milly"
# 使用双引号拼接
greeting="hello, "$your_name" !"
greeting_1="hello, ${your_name} !"
echo $greeting  $greeting_1
# 使用单引号拼接
greeting_2='hello, '$your_name' !'
greeting_3='hello, ${your_name} !'
echo $greeting_2  $greeting_3
```

输出结果：

```
hello, milly ! hello, milly !
hello, milly ! hello, ${your_name} !
```

shell中的变量

- 特殊变量

  $0：表示获取当前脚本的脚本名

  $n：获取当前执行的shell脚本的第n个参数，n=1..9,如果n大于10，用${10}

  $#：获取当前shell命令行中参数的个数

  $*：表示参数列表

  $@：表示参数列表

  $$：获取当前shell的进程号

  $!：执行上一个指令的PID

  $?：获取执行的上一个指令的返回值（0表示成功，非零表示失败）

  $_：在此之前执行的命令或脚本的最后一个参数

注：当“$*”和“$@”都加双引号时，两者有区别，都不加双引号时，两者无区别。

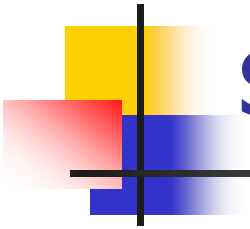
```
#!/bin/absh
echo "执行的文件名：$0";  # $0 为执行的文件名
echo "参数个数：$#";   # 传递到脚本的参数个数
echo "第一个参数为：$1";  # 第1个输入参数
echo "第二个参数为：$2";  # 第2个输入参数

for i in "$*"; do       # $*以一个单字符串显示所有向脚本
    传递的参数。$@可以获取所有传入参数
   echo $i
done
```

输出结果：

```
[qianying@ln02 shell-script]$ sh teshu.sh 1 3
执行的文件名：teshu.sh
参数个数：2
第一个参数为：1
第二个参数为：3
1 3
```

变量子串的常见操作

变量子串	说明
${#string}	返回string这个变量的长度
${strting:position}	在变量string中从position开始提取子串（从0开始，取到结尾）
${string:pos:len}	在变量string中从pos开始提取子串，提取长度为len
${string#sub}	在变量string中从头开始，删除sub匹配的子串
${string%sub}	在变量string中从末尾开始，删除sub匹配的子串
${string/old/new}	在变量string中将old内容，替换成new（只替换第一个找到的）
${string//old/new}	在变量string中将old内容，替换成new（替换所有找到的内容）



shell流程控制

- if-else
- for循环
- while语句
- shell case语句

shell数组

数组中可以存放多个值。Bash Shell 只支持一维数组（不支持多维数组），初始化时不需要定义数组大小，数组元素的下标由0开始

```
数组名=(值1 值2 ... 值n)

my_array=(A B "C" D)
echo "第一个元素为: ${my_array[0]}"
echo "第二个元素为: ${my_array[1]}"
echo "第三个元素为: ${my_array[2]}"
echo "第四个元素为: ${my_array[3]}"
echo "数组的元素为: ${my_array[*]}"
echo "数组的元素为: ${my_array[@]}"
echo "数组元素个数为: ${#my_array[*]}"
echo "数组元素个数为: ${#my_array[@]}"

还可以单独定义数组的各个分量：

my_array[0]=value0
```

数组遍历的方法

- **标准的for循环**

```
for(( i=0;i<${#array[@]};i++)) do
#${#array[@]}获取数组长度用于循环
echo ${array[i]};
done;
```

- **for … in**遍历（不带数组下标）：

```
for element in ${array[@]}
#也可以写成for element in ${array[*]}
do
echo $element
done
```

数组遍历的方法

- 遍历（带数组下标）：

```
for i in "${!arr[@]}";
do
    printf "%s\t%s\n" "$i" "${arr[$i]}"
done
```

- **While循环法：**

```
i=0
while [ $i -lt ${#array[@]} ]
#当变量（下标）小于数组长度时进入循环体
do
    echo ${ array[$i] }
    #按下标打印数组元素
    let i++
done
```

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。

  假定变量 a 为 10，变量 b 为 20：

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

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。

逻辑运算符

变量 a 为 10，变量 b 为 20:

运算符	说明	举例
&&	逻辑的 AND	[[$a -lt 100 && $b -gt 100]] 返回 false
||	逻辑的 OR	[[$a -lt 100 || $b -gt 100]] 返回 true

字符串运算符

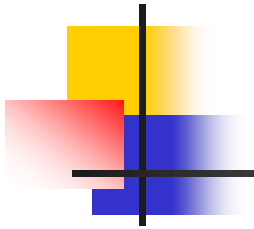
假定变量 a 为 “abc”，变量 b 为 “efg”：

运算符	说明	举例
=	检测两个字符串是否相等，相等返回 true。	[$a = $b] 返回 false。
!=	检测两个字符串是否相等，不相等返回 true。	[$a != $b] 返回 true。
-z	检测字符串长度是否为0，为0返回 true。	[-z $a] 返回 false。
-n	检测字符串长度是否为0，不为0返回 true。	[-n $a] 返回 true。
str	检测字符串是否为空，不为空返回 true。	[$a] 返回 true。

文件测试运算符

假设file="/share/home/qianying/shell-script/cal.sh"

运算符	说明	举例
-b file	检测文件是否是块设备文件，如果是，则返回 true。	[-b $file] 返回 false。
-c file	检测文件是否是字符设备文件，如果是，则返回 true。	[-c $file] 返回 false。
-d file	检测文件是否是目录，如果是，则返回 true。	[-d $file] 返回 false。
-f file	检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。	[-f $file] 返回 true。
-g file	检测文件是否设置了 SGID 位，如果是，则返回 true。	[-g $file] 返回 false。
-k file	检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。	[-k $file] 返回 false。
-p file	检测文件是否是有名管道，如果是，则返回 true。	[-p $file] 返回 false。
-u file	检测文件是否设置了 SUID 位，如果是，则返回 true。	[-u $file] 返回 false。
-r file	检测文件是否可读，如果是，则返回 true。	[-r $file] 返回 true。
-w file	检测文件是否可写，如果是，则返回 true。	[-w $file] 返回 true。
-x file	检测文件是否可执行，如果是，则返回 true。	[-x $file] 返回 true。
-s file	检测文件是否为空（文件大小是否大于0），不为空返回 true。	[-s $file] 返回 true。
-e file	检测文件（包括目录）是否存在，如果是，则返回 true。	[-e $file] 返回 true。



谢谢！