Linux 复习

主要内容

- Linux 常用命令
- Linux 文件系统
- bash 的基本功能
- Linux 进程管理
- 文件编辑器 vim
- 正则表达式
- 文本过滤（grep、sed、awk）
- Shell 脚本编程

基本概念

- 命令、命令行、提示符、shell、终端
- root、用户、用户名、用户ID、组、用户主目录
- 通配符、根目录、绝对路径、相对路径

Linux 文件命名规则

可以使用几乎任何字符，但通常使用字母（大小写）、数字、下划线、句点等

文件名中尽量少用特殊字符

linux 应用程序和文件区分大小写！

Linux 常用命令

Linux 命令一般格式

```
command [options] [arguments]
```

man、info、which

passwd（本地）、yppasswd（NIS）

输入密码时，屏幕上不会有任何输出或提示！

更多命令，参见课程主页

Linux 常用命令

ls、ln、cp、mv、rm

```
ls -l -a -F -d -S -t -r
ln -s
cp -r -f -p -i
mv -f -i -u
rm -r -f -i
```

touch、file

```
touch -t
```


Linux 常用命令

cat、more、less、head、tail、cut、wc

```
cat -n
head -num
tail -num -f
cut -c num1-num2
wc -c -w -l
```

cd、pwd、mkdir、rmdir、rm -r

```
cd ~ / ./ -
```

alias、unalias、history

Linux 常用命令

gzip、gunzip、zip、unzip

```
gzip -d -r -num
zip -r
```

chown、chmod、umask

```
chmod [who] [+|-|=] [mode] files/dirs
chmod num files/dirs
chmod -R
```

mount、umount、df、du

```
df -h
du -sh
```


Linux 常用命令

tar：打包、解包

```
tar -cvf  /  tar -xvf    --> .tar
tar -czvf /  tar -xzvf   --> .tgz
tar -cjvf /  tar -xjvf   --> .tbz
```

! tar 的选项前的符号 - 可以省略

echo、read、tee

```
echo -n -e
read -p
tee -a      # 追加方式
```

Find 命令

q find 搜索满足条件的文件，并执行指定的操作

```
find  [起始目录]  条件  操作
```

n 以文件名或文件属性查找

-name '字符串'	-empty
-gid	-size n[bckw]
-uid	-type f/d/l
-group '字符串'	
-user '字符串'	

l 字符串内可以使用通配符 *、?、[ ]
l 大小可以使用 + 、 -

Find 命令

n 以时间为条件查找

```
-amin / -atime / -anewer   被访问
-cmin / -ctime / -cnewer   文件状态被修改
-mmin / -mtime / -newer    内容被修改
```

n 可以执行的操作

```
-exec 命令名称 {}  \;
-ok 命令名称 {}  \;
-ls
-print / -printf 格式 / -fprint 文件名
```


文件系统

u 文件系统　文件的组织、存储和操作
u 常见的文件系统类型：
FAT、NTFS、ext3、swap、 NFS、vfat
u Linux 目录结构：目录树结构、父子结构
u 路径：绝对路径、相对路径
u 当前工作目录：pwd
u 特殊目录：/　~　.　..　-
u Linux 常见文件类型：
普通文件、目录文件、链接文件、设备文件

文件的访问权限

u 每个文件或目录都可以有 读、写 和 执行 的权限
u 三种不同类型的用户：所有者、同组用户和其它用户
u 查看文件和目录的属性： ls –l / ls -ld
u 修改文件和目录的访问权限： chmod

```
chmod [who] [+|-|=] [mode] files/dirs
chmod num files/dirs
```


Bash 基本功能

u Shell： 命令语言解释器、解释型程序设计语言
u 常见的 Shell
sh、csh、ksh、bash、tcsh、pdksh
u shell 命令行特征

```
ls -F; cp -i mydata newdata
```

```
find /dev -name '[Ii]???' –type f \
     -exec ls -l {} \;
```

u 通配符： *　?　[]

Bash 基本功能

u 命令行自动补齐功能：Tab
u 别名：alias / unalias
u 管道： |
u 命令历史记录：history、上下箭头键
u 引用（屏蔽特殊字符的特殊含义）
转义字符 \ 、单引号 ' 、双引号 "
u 定制 bash： bash 配置文件

```
~/.bashrc
```


输入输出重定向

comd > filename	comd >> filename
comd 2 > filename	comd 2 >> filename
comd > filename 2>&1	comd >> filename 2>&1
comd < filename	
comd < filename >filename2	
comd << delimiter 从标准输入中读入，直至遇到 delimiter 分界符	
comd <&m　把文件描述符 m 作为标准输入	
comd >&m　把标准输出重定向到文件描述符 m 中	
comd <&-　关闭标准输入	

命令的执行顺序

q 使用 &&

```
comd1 && comd2
```

&& 左边的命令（comd1）返回真（即返回 0，成功被执行）后，&& 右边的命令（comd2）才能够被执行。

q 使用 ||

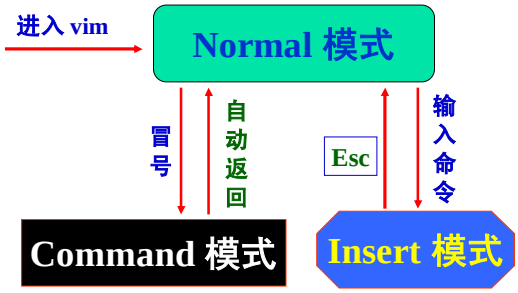
```
comd1 || comd2
```

如果 || 左边的命令（comd1）未执行成功，那么就执行 || 右边的命令（comd2）。

Shell 进程

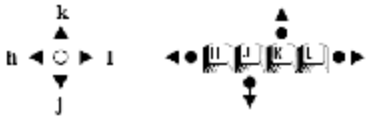
- 进程
  - 正在运行的程序叫做进程（process）
  - 进程号（process ID）：每个进程的进程号是唯一的
  - 查看当前运行的程序及其进程号：ps
- 多进程：分时技术
- 前台/后台：&、fg、bg
- 进程的优先级及其调整：nice、renice
- 终止进程：kill

vim 编辑器



vim 编辑器

- 打开或新建文件：vi fname / vi –r fname
- 从 Normal 模式切换到 Insert 模式：
  i / I / a / A / o / O
- 任何模式下按 ESC键 便可返回到 Normal 模式
- 保存文件并退出 vim
  :w / :wq / :w newfile / :x / ZZ
- 光标移动：方向键、hjkl、重复因子、...



vim 编辑器

- 删除文本：x/X，dd/d0/d$/dw、重复因子
- 取消命令：u/U、ctrl+r、重复命令：.
- 复制与粘贴：yy/y0/y$、p/P、重复因子
- 文本替换：r/R、s/S
- 文本行合并：J
- 搜索字符串：?、/
- 替换字符串：[address]s/old/new[/gc]
- 与shell的交互：:!cmd、:sh
- 在线帮助：:help、vimtutor(命令行方式)

正则表达式元字符

^	只匹配行首（可以看成是行首的标志）
$	只匹配行尾（可以看成是行尾的标志）
*	一个单字符后紧跟 *，匹配 0 个或多个此单字符
[ ]	匹配 [] 内的任意一个字符 （[^] 反向匹配）
\	用来屏蔽一个元字符的特殊含义
.	匹配任意单个字符
x\{n\}	匹配 字符 x 连续出现 n 次的情形
x\{n,\}	匹配 字符 x 至少连续出现 n 次的情形
x\{n,m\}	匹配 字符 x 连续出现次数在 n 与 m 之间的情形

注：字符 x 可以使用 []，\或 . 来指定。
如：[a-z]\{5\}，\$\{2,\}，.\{2,5\}

常用的正则表达式举例

[Ss]igna[lL]	匹配 signal、signaL、Signal、SignaL
[Ss]igna[lL]\.	同上，但后面加一句点
^USER$	只包含 USER 的行
\.	带句点的行
^d..x..x..x	对对用户、用户组及其他用户、组成员有可执行权限的目录
^[^l]	不以 l 开始的行
[yYnN]	大写或小写的 y 或 n
.*	匹配任意多个字符
^.*$	匹配任意行
^......$	只包含 6 个字符的行

常用的正则表达式举例

[a-zA-Z]	任意单个字母
[^a-zA-Z0-9]	非字母或数字
[^0-9\$]	非数字或美元符号
[123]	1 到 3 中一个数字
\^q	包含 ^q 的行
^.$	仅有一个字符的行
^\.[0-9][0-9]	以一个句点和两个数字开始的行

[0-9]\{2\}-[0-9]\{2\}-[0-9]\{4\} 日期格式 dd-mm-yyyy
[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\} 类 IP 地址格式 nnn.nnn.nnn.nnn

文本过滤

grep、 sed、 awk

grep、sed、awk 不会修改源文件的内容

grep

```
grep [选项]  pattern  filelist
```

- pattern：可以是正则表达式（用单引号括起来）、字符串（加双引号）、或一个单词。

-c	只输出匹配的行的总数
-i	不区分大小写
-h	查询多个文件时，不显示文件名
-l	查询多个文件时，只输出包含匹配模式的文件名
-n	显示匹配的行及行号
-v	反向查找，即只显示不包含匹配模式的行
-r	递归查找

grep 命令

- 精确匹配单词： \< 和 \>
- 递归搜索：-r

```
grep -rl "eth0" /etc
```

- 关于某个字符连续出现次数的匹配

```
grep 'o\{2,\}'  helloworld
```

- grep 与管道

```
ls –l | grep '^d'
```

- egrep 与 fgrep

sed

```
sed [-n][-e] 'sed_cmd' input_file
```

- sed_cmd 的格式：'[address]sed_edit_cmd'
  其中 address 为行定位模式，sed_edit_cmd 为编辑操作

- sed 逐行处理输入 (文件)，并将输出结果发送到屏幕

sed 定位方式

n	表示第 n 行
$	表示最后一行
m,n	表示从第 m 行到第 n 行
/pattern/	查询包含指定模式的行。如 /disk/、/[a-z]/
/pattern/,n	表示从包含指定模式的行 到 第 n 行
n,/pattern/	表示从第 n 行 到 包含指定模式的行
/模式1/,/模式2/	表示从包含模式1 到 包含模式2的行
!	反向选择， 如 m,n! 的结果与 m,n 相反

常见 sed 编辑命令

p	打印匹配行	s	替换命令
=	显示匹配行的行号	l	显示指定行中所有字符
d	删除匹配的行	r	读文件
a\	在指定行后面追加文本	w	写文件
i\	在指定行前面追加文本	n	读取指定行的下面一行
c\	用新文本替换指定的行	q	退出 sed

一些 sed 行命令集

'/north/p'	打印所有包含 north 的行
'/north/!p'	打印所有不包含 north 的行
's/\.$//g'	删除以句点结尾的行中末尾的句点
's/^ *//g'	删除行首空格（命令中 ^ * 之间有两个空格）
's/ */ /g'	将连续多个空格替换为一个空格 命令中 */ 前有三个空格，后面是一个空格
'/^$/d'	删除空行
's/^.//g'	删除每行的第一个字符，同 's/.//'
's/^/%/g'	在每行的最前面添加百分号 %
'3,5s/d/D/'	把第 3 行到第 5 行中每行的 第一个 d 改成 D

awk

```
awk [-F 字段分隔符] 'awk_script' input_file
```

- awk_script 可以由一条或多条 awk_cmd 组成，每条 awk_cmd 各占一行。
- 每个 awk_cmd 由两部分组成：/pattern/{actions}
  /pattern/ 省略时表示对所有记录执行指定的 actions
  {actions} 省略时表示打印匹配行

```
awk 'BEGIN {actions}
     /pattern1/{actions}
      ......
     /patternN/{actions}
     END {actions}'  input_file
```

注意 BEGIN 和 END都是大写字母。

其中 BEGIN {actions} 和 END {actions} 是可选的

awk 的执行过程

① 如果存在 BEGIN，awk 首先执行它指定的 actions

② awk 从输入中读取一行，称为一条输入记录

③ awk 将读入的记录分割成数个字段，并将第一个字段放入变量 $1 中，第二个放入变量 $2 中，以此类推；$0 表示整条记录；字段分隔符可以通过选项 -F 指定，否则使用缺省的分隔符。

④ 把当前输入记录依次与每一个 awk_cmd 中 pattern 比较：
如果相匹配，就执行对应的 actions；
如果不匹配，就跳过对应的 actions，直到完成所有的 awk_cmd

⑤ 当一条输入记录处理完毕后，awk 读取输入的下一行，重复上面的处理过程，直到所有输入全部处理完毕。

⑥ 如果输入是文件列表，awk 将按顺序处理列表中的每个文件。

⑦ awk 处理完所有的输入后，若存在 END，执行相应的 actions。

awk 中的模式匹配

u 使用正则表达式：/rexp/，如 /^A/、/A[0-9]*/

awk 中正则表达式中常用到的元字符有：

^	只匹配行首（可以看成是行首的标志）
$	只匹配行尾（可以看成是行尾的标志）
*	一个单字符后紧跟 *，匹配 0个或多个此字符
[ ]	匹配 [] 内的任意一个字符（[^] 反向匹配）
\	用来屏蔽一个元字符的特殊含义
.	匹配任意单个字符
str1|str2	匹配 str1 或 str2
+	匹配一个或多个前一字符
?	匹配零个或一个前一字符
( )	字符组

awk 中的模式匹配

② 使用布尔（比较）表达式，表达式的值为真时执行相应的操作 (actions)

l 表达式中可以使用变量（如字段变量 $1,$2 等）和 /rexp/

l 表达式中的运算符有

n 关系运算符: < > <= >= == !=

n 匹配运算符: ~ !~

x ~ /rexp/ 如果 x 匹配 /rexp/，则返回真；

x!~ /rexp/ 如果 x 不匹配 /rexp/，则返回真。

```
awk '$2 > 20 {print $0}' shipped
```

```
awk '$4 ~ /^6/ {print $0}' shipped
```


awk 中的模式匹配

l 复合表达式：&&（逻辑与）、||（逻辑或）、!（逻辑非）

```
expr1 && expr2   两个表达式的值都为真时，返回真
expr1 || expr2   两个表达式中有一个的值为真时，返回真
!expr    表达式的值为假时，返回真
```

```
awk '($2<20)&&($4~/^6/){print $0}' shipped
```

```
awk '($2<20)||($4~/^6/){print $0}' shipped
```

```
awk '!($4~/^6/){print $0}' shipped
```

```
awk '/^A/ && /0$/{print}' shipped
```

注：表达式中有比较运算时，一般用圆括号括起来

awk 中的字段分隔符

q 字段分隔符

awk 中的字段分隔符可以用 -F 选项指定，缺省是空格。

```
awk '{print $1}' datafile2
```

```
awk –F: '{print $1}' datafile2
```

```
awk -F'[:/]' '{print $1}' datafile2
```


actions 和内置变量

q 操作由一条或多条语句或者命令组成，语句、命令之间用分号“;”隔开。操作中还可以使用流程控制结构的语句

q awk 命令

l print 输出列表：各输出参数之间用逗号或空格隔开

l printf（[格式控制符], 输出列表）：格式化打印

l next：停止处理当前记录, 开始读取和处理下一条记录

l nextfile：停止处理当前文件而处理下一个文件

l exit：使 awk 停止执行而退出。若存在 END 语句，则执行 END 指定的 actions

q 字段变量：$0, $1, $2, ...

q 内置变量： FILENAME、NR、FNR、NF 等

shell 编程——变量

q 变量命名

l 以字母或下划线开头，后面可以跟字母、数字 或下划线

l 变量名关于大小写敏感

q 变量类型：局部变量、环境变量

q 变量赋值：variable=value

l 赋值号两边不能有空格

q 显示变量的值：echo

q 清除变量：unset

q 只读变量：readonly

shell 编程——变量

q 环境变量： export / declare -x

q 整型变量：declare -i

q 变量置换： ${var:-word}、${var:=word}
${var:?word}、${var:+word}

q 相关命令

```
export / export -n / export -p
```

```
declare –r / declare –x / declare –i
```

```
basename 文件或目录
```


shell 编程——变量

q 位置参量/命令行参数

$0、$1-$9、${10}、$#、$*、$@、$$、$!、$?

q 变量的间接引用

```
eval newstr=\$$str2; echo $newstr
```

```
newstr=${!str2}
```

q 命令替换 `hostname` 、$(hostname)

q 整数运算

declare 定义的整型变量可以直接进行运算

否则需用 let 命令或 $[...]、$((...)) 进行整数运算

变量的输入输出

q 输入：read

```
read / read var1 ... / read -p "提示"
```

q 输出：printf

输出参数用空格隔开

```
printf "%-12.5f \t %d \n" 123.45 8
```

format 以%开头

flag
- -:左对齐
- +:输出符号
- 0:空白处添0
- 空格:前面加一空格

field width 字段宽度

precision 小数点后输出位数

格式符
- cdefgsox
- \b \n \r \t \v \\ \" %%

字符串操作

q 字符串操作

m 的取值从 0 到 ${#var}-1

${#var}	返回字符串变量 var 的长度
${var:m}	返回${var}中从第m个字符到最后的部分
${var:m:len}	返回${var}中从第m个字符开始，长度为len的部分
${var#pattern}	删除${var}中开头部分与pattern匹配的最小部分
${var##pattern}	删除${var}中开头部分与pattern匹配的最大部分
${var%pattern}	删除${var}中结尾部分与pattern匹配的最小部分
${var%%pattern}	删除${var}中结尾部分与pattern匹配的最大部分
${var/old/new}	用new替换${var}中第一次出现的old
${var//old/new}	用new替换${var}中所有的old(全局替换)

注：pattern，old 中可以使用通配符。

条件测试

q 字符串测试

操作符两边必须留空格！

[-z string]	如果字符串string长度为0，返回真
[-n string]	如果字符串string长度不为0，返回真
[str1 = str2]	两字符串相等（也可以使用 == ）
[str1 != str2]	两字符串不等

如果使用双方括号，可以使用 通配符 进行模式匹配。

[[str1 = str2]]	两字符串相等（也可以使用 == ）
[[str1 != str2]]	两字符串不等
[[str1 > str2]]	str1大于str2,按ASCII码比较
[[str1 < str2]]	str1小于str2,按ASCII码比较

例：name=Tom; [[$name > Tom]]; echo $?

条件测试

q 整数测试

注意这两种方法的区别！

[int1 -eq int2]	int1 等于 int2
[int1 -ne int2]	int1 不等于 int2
[int1 -gt int2]	int1 大于 int2
[int1 -ge int2]	int1 大于或等于 int2
[int1 -lt int2]	int1 小于 int2
[int1 -le int2]	int1 小于或等于 int2

((int1 == int2))	int1 等于 int2
((int1 != int2))	int1 不等于 int2
((int1 > int2))	int1 大于 int2
((int1 >= int2))	int1 大于或等于 int2
((int1 < int2))	int1 小于 int2
((int1 <= int2))	int1 小于或等于 int2

条件测试

q 逻辑测试

[expr1 -a expr2]	逻辑与，都为真时，结果为真
[expr1 -o expr2]	逻辑或，有一个为真时，结果为真
[! expr]	逻辑非

如果使用双方括号，可以使用 通配符 进行模式匹配。

[[pattern1 && pattern2]]	逻辑与
[[pattern1 || pattern2]]	逻辑或
[[! pattern]]	逻辑非

条件测试

q 文件测试

-f fname	fname 存在且是普通文件时，返回真（即返回 0）
-L fname	fname 存在且是链接文件时，返回真
-d fname	fname 存在且是一个目录时，返回真
-e fname	fname（文件或目录）存在时，返回真
-s fname	fname 存在且大小大于 0 时，返回真
-r fname	fname（文件或目录）存在且可读时，返回真
-w fname	fname（文件或目录）存在且可写时，返回真
-x fname	fname（文件或目录）存在且可执行时，返回真

条件语句

```
if expr1        # 如果expr1 为真(返回值为0)
then            # 那么
   commands1    # 执行语句块 commands1
elif expr2      # 若expr1 不真，而expr2 为真
then            # 那么
   commands2    # 执行语句块 commands2
elif expr3      # 若前面都不真，而 expr3 为真
then            # 那么
   commands3    # 执行语句块 commands3
else            # 若前面的条件都不成立
   commands4    # 执行语句块 commands4
fi              # if 语句必须以单词 fi 终止
```


选择语句

```
case expr in # expr 为表达式，关键词 in 不要忘!
  pattern1)  # 若expr 与pattern1 匹配，注意括号
   commands1 # 执行语句块 commands1
   ;;        # 跳出case 结构
  pattern2)  # 若expr 与pattern2 匹配
   commands2 # 执行语句块 commands2
   ;;        # 跳出case 结构
  ... ...    # 可以有任意多个模式匹配
  *)         # 若expr 与上面的模式都不匹配
   commands  # 执行语句块 commands
   ;;        # 跳出case 结构
esac         # case 语句必须以单词 esac 终止
```

循环语句

```
for variable in list
# 每一次循环，依次把列表list 中的一个值赋给循环变量
do            # 循环开始的标志
  commands    # 循环变量每取一次值，循环体就执行一遍
done          # 循环结束的标志
```

```
while expr   # 执行 expr
do # 若expr的退出状态为0，进入循环，否则退出while
  commands    # 循环体
done          # 循环结束标志，返回循环顶部
```

```
until expr   # 执行 expr
do # 若expr的退出状态非0，进入循环，否则退出until
  commands    # 循环体
done          # 循环结束标志，返回循环顶部
```


循环和菜单

```
select variable in list
do            # 循环开始的标志
  commands    # 循环变量每取一次值，循环体就执行一遍
done          # 循环结束的标志
```

- break、continue、
- sleep、exit
- shift