下载文章

> Linux | 菜鸟教程 (runoob.com) 推荐这个作为入门
>
> linux命令行与shell脚本编程大全

linux 基础

一、基本操作

1 ls list

`command [-options] parameter ...`
对于选项通常会带'-'号，-h，有时也会用'--'使用完整全名，--help

`ls -l` 显示详情信息
`ls -a` 显示隐藏文件
`ls -R` 显示子目录文件
`ls -F` 显示文件类型
-F选项会在目录名之后添加正斜线（/），以方便用户在输出中分辨。类似地，它还会在可执行文件（比如上面的my_script文件）之后添加星号（*），以帮助用户找出可在系统中运行的文件
`ls -i` 显示文件inode编码

通配符

`ls -l t?st` 替代一个字符
`ls -l t*st` 替代多个字符
`ls -l t[abc]st` 用abc替代
`ls -l t[a-c]st` 用a到c替代
`ls -l t[!d]st` 用除了d替代
pwd print working directory
`pwd -P` 如果你在一个链接文件夹内，单纯的pwd会出现错误的位置，加个-P选项就会获得正确的路径了

2 处理文件

`touch test` 创建一个空文件
`rm -f` 强制删除
`cp -R Document/ NewDocument/` 在执行cp -R命令之前，目录NewDocuments并不存在。它是随着cp -R命令被创建的，整个Documents目录中的内容都被复制到其中。
`ln -s file slink_file` 创建软链接，相当于快捷方式
`cp -i file1 file2` **-i**的选项是在执行操作时问询
例如：`rm -i`
`mv -i`

`mkdir -p /path/to/new/dir` 根据需要创建缺失的父目录 即如果你只有/path，这个命令会自动创建/path/to，/path/to/new，path/to/new/dir文件夹

3 查看文件内容

`file .bashrc`查看文件类型甚至是软链接的位置
`cat -n file`查看文本并加上行数
`less file` 查看大段文本
`tail -n 5 file`or`tail -5 file`查看最后5行
`head -n 5 file`or`head -5 file`查看开始5行
`tail -f file` 该选项允许你在其他进程使用此文件时查看文件的内容。tail命令会保持活动状态并持续地显示添加到文件中的内容。这是实时监测系统日志的绝佳方式。

二、数据管理

1 数据排序

`sort file`默认情况下，sort命令会依据会话所指定的默认语言的排序规则来对文本文件中的数据行进行排序
`sort -n file` 将数字按值排序升序
`sort -nr file` 将数字按值排序逆序
`sort -M file` 含有时间戳日期的文件按默认的排序方法来排序

2数据搜索

`grep [options] pattern [file]`grep命令输出了匹配该模式的整行
`grep -v t file` 输出不包含t的行
`grep -n t file` 输出包含t的行及行号
`grep -c t file` 输出包含t的行数
`grep -e t -e r file` 输出包含t和r的行

3命令别名

`alias -p` 查看能使用的别名
`alias li='ls -i --color=auto'` 创建别名并使用色彩自动编码
`unalias li` 删除别名 如果被删除的别名不是你设置的，那么等下次重新登录系统的时候，该别名就会再次出现。可以通过修改环境文件永久地删除某个别名。

这个在.bashrc文件中修改后，可以在每次启动终端的时候就运行命令，从而不用每次都要重新命名别名，在输入登录命令这种很长而且不用修改的命令的时候，指定一个别名会是一个优雅的操作

三、环境变量

设定变量

全局环境变量对于shell会话和所有生成的子shell都是可见的。局部环境变量则只对创建它的shell可见。

- 设置局部变量
  `my_variable="Hello World"` 在变量名、等号和值之间没有空格

- 设置全局变量
  `export my_variable="Hello World"` export命令使其变成了全局环境变量。
  这个可以作为在多个脚本之间变量的交换，在大型脚本中或许很有用

-删除环境变量
`unset my_variable`

四、构建基础脚本

让两个命令一起运行，可以将其放在同一行用分号隔开
例如：

```
#!/bin/bash
for i in {0..8}; do cd condor_prime1v2_${i}; source SplitAndRun.sh; cd ..; done
```

1 echo命令

- 在默认情况下，无须使用引号将要显示的字符串划定出来
  `echo this is a test`
- echo命令可用单引号或双引号来划定字符串 `echo "this's a test"`
  `echo -n` 字符串和命令输出显示在同一行中，即不换行

2 自定义变量

- 变量名区分大小写
- 使用等号为变量赋值。**在变量、等号和值之间不能出现空格**`a=1; str="test"`

3 命令替换

从命令输出中提取信息并将其赋给变量

- 反引号(`) *这个可以显示自己是个高手，但更推荐用下面那个，因为与其他变量命名格式类似*
- $()格式

```
today=$(date +%y%m%d) #会以两位数输出日期240116
cat log > log.$today
```

Tips
`yy` 复制一行
`p` 粘贴一行
`u` 撤销上次操作
`.` 重复上次操作
`/test` 搜索test
`:` 末行模式，可以进行更多的操作
`:1` 定位到第一行

`:$`定位到最后一行
`:.,.+5w test.tst`当前行到之后的五行输出为test.txt

4 数学运算

用方括号将数学运算的结果赋予变量`a=$[(1+1)*2]` 但bash shell只支持整数运算
浮点计算

- bc计算器
  通过`bc`访问bash计算器，通过`quit`退出计算器
- 在脚本中使用bc

```
a=$(echo "scale=4; 3.4 / 5" | bc) #scale设置为4为小数
```

- **双括号(())** 双括号里面的语句可以使用高级的数学表达式，并且不用转义大于号之类的 >
  - *我觉得一般情况下可以无脑用双括号，浮点数用bc就行*

退出状态码
`echo $?` 保存最后一个已执行命令的退出状态码,无报错状态码为0

五、结构化命令

1 if-then

```
if command #如果if语句行命令的退出状态值为0
then
commands #那么代码块中的命会被执行
else
commands #否则，会执行else部分
fi
#嵌套结构
if command1 then
commands elif command2 then
commands fi
```

- **数值比较**
  用到方括号的语句不多，基本上除了条件判断时用到，其他都用圆括号

```
if [ $n1 -gt $n2 ]
then echo $n1
else echo $n2
fi
```

condition	describe
n1 -eq n2	n1 = n2
n1 -ge n2	n1 >= n2
n1 -gt n2	n1 > n2
n1 -le n2	n1 <= n2
n1 -lt n2	n1 < n2
n1 -ne n2	n1 != n2

- **字符串比较**

condition	describe
str1 = str2	str1与str2是否一致
str1 != str2	str1与str2是否不一致
str1 < str2	**
str1 > str2	**
-n str1	检查str1的长度是否不为0
-z str1	检查str1的长度是否为0

- *一个字符串是否大于或小于另一个字符串*
  - 大于号和小于号必须**转义**`str1 \> str2`，否则shell会将其视为重定向符
  - 大小比较的是字符的Unicode的编码值
  - 大于和小于顺序与sort命令所采用的不同
- **布尔运算符号**
  && and
  || or

高级应用

- **双括号(())** 双括号里面的语句可以使用高级的数学表达式，并且不用转义大于号之类的 >
- **case命令**

```
read choice
case $choice in
    a)
    command
    ;;
    b)
    command
    ;;
    *)
    echo "no choice"
    ;;
esac
```

2 for

- for a in *list*

```
list=abc "ab'c" "ab c"
for test in abc "ab'c" "ab c"
do
  command
done
file="list.txt"
for test in $(cat $file) #会依据特定的字段分隔符遍历，分隔符：空格、制表符、换行符
IFS=$'\n' #可以指定换行符为分隔符，这样就能跳过字段中的空格和制表符
IFS=$'\n:' #这样就指定换行符与冒号作为字段分隔符
```

- for a in *range*

```
for ((a=1,b=10;a<=10;a++.b--)) #仿c语言的shell命令
```

---

3 `while`与`until`

```
a=3
while echo $a #这里只是展示可以放在while里面
  [ $a -ge 0 ] #这里需要与方括号空一格
do
  a=$[$a-1] #这里赋值时不能空格，但可以与方括号空一格，或者按照之前说的惯例用圆括号赋值，这样方便记忆一点 a=$(($a-1))
done
until echo $a
  [ $a -eq 0 ]
do
  a=$[$a-1]
done
```

---

4 `break` and `continue`

```
for ((a=1;a<4;a++))
do
  if [ $a -eq 2 ]; then
    continue #continue会中止这次循环，从头开始
  fi
  for ((b=1;b<4;b++))
  do
    if [ $b -eq 2 ]; then
      break 2 #默认下break只会终止一层循环，后面加数字2意味着终止向外的第二层循环
    fi
  done
done > test.txt #你可以将循环结果输出到一个文件内，或用管道传给其他命令如：done | sort 即将输出进行排序
```

5 `read`

`echo -n` 字符串和命令输出显示在同一行中，即不换行
`read -p` 可以指定提示字符
`read -t 5` 当时间超过5s没有输入就会返回非0退出状态码
`read -n 1` 统计输入的字符数为1时自动退出并赋值
`read -s` 不会显示输入的数据，用于密码

```
echo -n "enter your name:"
read name
read -p "enter your name:" first last #两个方式一样，但这种方式能分配多个变量
if read -t 5 -p "enter your name:" first last
then
else
  echo "time out"
fi
read -n 1 -p "enter y/n" chose
```

六.输出与输入

一个命令会将输出显示在终端显示器上，正确的输出会指向STDOUT文件，错误的输出会指向STDERR文件

- 标准文件描述符
  0 标准输入 1 标准输出 2 标准错误
  `ls file1 file2 file3 1>out 2>err` 1指向正确的输出，2指向错误的输出 不指定的话正确与错误都会输出到终端上
  `ls file1 file2 file3 &>both` &会输出二者

- 重定向输出
  如果你有意在脚本中生成错误消息，可以将单独的一行输出重定向到STDERR。这只需要使用输出重定向符号将输出重定向到STDERR文件描述符。在重定向到文件描述符时，必须在文件描述符索引值之前加一个&：
  `echo "This is an error message" >&2`
- exec命令
  exec命令会启动一个新shell并将STDOUT文件描述符重定向到指定文件，例如 `exec 2>error` `exec 0<inputfile` 这让脚本可以在文件testfile中而不是键盘上获取输入，之前的read命令就是从STDIN读取数据0，即通过键盘输入的数据，这里将0重定向为inputfile就意味着会从这里读取数据。
  `exec 3>file3out` 你可以自定义一个描述符 这样在之后就能让你想要输出的输出到3这个符号内`echo "This should be stored in a file" >&3`可以维持显示器的正常输出，并将特定信息重定向到指定文件（比如日志文件）
- 神奇的小功能
  - `/dev/null`就像是一个垃圾桶
    `echo "This is an error message" > /dev/null`这样就能丢弃这个数据
    `cat /dev/null > testfile` 这样就能直接清除文件内容，而不需要重新创建
  - `mktemp testfile.XXXXXX`会任意填入字符并创建*文件*
    `mktemp -t testfile.XXXXXX` 会在系统的临时目录/tmp/test.XXXXXX创建文件
    `mktemp -d testdir.XXXXXX`会任意填入字符并创建*文件夹*
  - `tee test`像是一个T转接口
    `data | tee test`这样会同时输出到屏幕上和test文件内(>test)
    `data | tee -a test`不会覆盖test(>>test)

七、脚本运行

1 系统信号与产生

sig	value	mean
1	SIGHUP	挂起
2	SIGINT	中断 -> Ctrl+C
3	SIGQUIT	停止
9	SIGKILL	无条件终止 -> kill -9 PID
20	SIGTSTP	停止或暂停但不终止 -> Ctrl+Z

- trap命令 trap *command signals*
  `trap "echo 'trapped Ctrl+C'" SIGINT` 输入Ctrl+C并不会停止脚本而是执行echo命令，这样可以防止命令被打断

2 脚本后台运行

`&`在一条命令后面加&就能使命令后台运行，并通过对1，2的重定向，使终端不显示信息而达到完全静默
`ps`可以查看目前的作业信息
`jobs`更推荐jobs命令查看作业情况并有一些命令行选项
`-l PID` | `-r running` | `-s stopped`
Tips: jobs命令输出中的加号和减号。带有加号的作业为默认作业。如果作业控制命令没有指定作业号，则引用的就是该作业。
带有减号的作业会在默认作业结束之后成为下一个默认作业。任何时候，不管shell中运行着多少作业，带加号的作业只能有一个，带减号的作业也只能有一个。

八、高级脚本

- 函数

```
function func1 { #定义一个函数
  read -p "enter your number:" a
  echo "$a *2" |bc
}
func1 #直接调用func1
result=$(func1) #将函数结果输出给$result
echo -e "\nyour result is $result" #-e 为转义选项，转义\n

function func2 {
  #local result=$(echo "$1 * $2" |bc) #local保证了变量只在函数中有效，函数之外的同名变量互不影响
  echo "$1 * $2" |bc
}
if [ $# -eq 2 ]; then   #`$#` 用于获取参数的个数
  result=$(func2 $1 $2) #你需要手动输入 例如：source test.sh 1.1 2
  echo "your result is $result"
else
  echo "error"
fi
```

- 数组

```
#输入数组
function func3 {
  local myarray=($(echo "$@")) #`$@` 用于展开参数列表
  echo ${myarray[*]} #表示将数组的所有元素作为单个参数展开，元素之间用第一个字符IFS的值分隔
}
array=(1 2 3 4 5)
#array=("apple" "banana" "cat")
func3 ${array[*]}

#输出数组
```

```
function func4 {
  local n=($#)
  local myarray=($(echo "$@"))
  local newarray=($(echo "$@")) #这里可能是想产生一个与myarray同样多数组的newarray，这样在之后的填入中才能做到依次替换
  for (( i=1; i<=$n; i++)){
    newarray[$i]=$((${myarray[$i]} * 2))
  }
  echo ${newarray[*]}
}
array=(1 2 3 4 5)
func4 ${array[*]}
```

- 递归

```
function func5 {
  if [ $1 -eq 1 ]; then
    echo $1
  else
    local x=$(( $1 - 1 ))  #得到n-1
    local result=$(func5 $x)
    echo $(($result * $1)) #
  fi
}
read a
echo $(func5 $a)
```

九、sed 命令

需要注意的是，sed命令不会修改原始文件，只是在终端中的输出发生变化，你可以用重定向来输出sed修改后的文件或者用 -i 选项直接修改文件内容

- 用于批量修改文本
  `sed 's/old/new/' test.txt` s为替换命令
  `sed -e 's/old1/new1/; s/old2/new2/' test.txt` 多个修改，注意分号和单引号
  `sed -f txt.sed test.txt` 将`s/old1/new1/`放进txt.sed中，用-f选项可以进行大量修改
  `sed 'd' test.txt` d为删除命令 *慎用*
  `sed 'i\指定文本'`与`sed 'a\指定文本'`分别在指定行前和指定行后增加一行内容
- 命令组 *如果需要在单行中执行多条命令，可以用花括号将其组合在一起*

```
sed -n '/test/{
> =
> p
> }' test.txt
```

- 替换标志
  `sed 's/pattern/replacement/flags` flag为替换标志 有四种 可以写在一起
  - 数字 指定文中的第几处
    `echo -e "test1 test1 test2 \ntest3 test4"|sed 's/test1/new/2'`
    ->test1 new test2
    ->test3 test4
  - g 替换所有
    `echo -e "test1 test1 test2 \ntest3 test4"|sed 's/test1/new/g'`
    ->new new test2
    ->test3 test4
  - p 打印出替换后的行 *-n的意思是只打印匹配的行*
    `echo -e "test1 test1 test2 \ntest3 test4"|sed -n 's/test1/new/gp'`
    ->new new test2
  - w *file* 将替换结果写入文件
  - *`sed 's|//ccc0|\nchain->Add(\"${FILE_PATH}\")|g'` 当/会引起混乱时，可以使用别的一些符号分隔*
- 指定行地址 *对i和a同样适用*
  - `sed '2s/test1/new/'` 指定第2行
  - `sed '2,4s/test1/new/'` 指定2到4行
  - `sed '2,$s/test1/new/'` 指定2行之后 *$代表最后一行*
- 打印
  - p *打印文本*
  - = *打印行号*
  - | *列出行，可以打印出不可打印字符，例如\t*
- `sed -n '/test/p'` 这个和 `grep test` 类似 前者更高级一点，可以使用正则表达式，见十一章

十、gawk 命令

是awk的增强版本
`gawk '{print "hello"}'` 设hello为显示一行固定的文本字符串，因此不管在数据流中输入什么文本，你都会得到同样的文本输出
`echo "me:hello world" |awk  '{print $1}'` ->me:hello 默认以空格作为分隔符
`echo "me:hello world" |awk -F: '{print $1}'`->me **-F: 可以指定:为分隔符** -F'[:/]' 可以指定多个字符

```
---for test.gawk---
BEGIN {
  print "title:character table"
  print "character \t number"
  print "---- \t ----"
  }
{
```

```
  print $1 "\t" $2
}
END {
  print "end"
 }
---
gawk -f test.gawk test.txt #与sed的-f选项类似，你可以将命令写入文件
```

一些常见处理

- 数学表达式
  `gawk '$1 == 0 {print $0}'`意味着第一列为0则输出整行
- if 语句
  `gawk '{if ($1 > 2) print $1}'`
  `gawk '{if ($1 > 2) {a = $1 *2 ; print a} else {print $2}}' test.txt`
- while,for 语句 *类似if*

printf 格式化打印

需要添加换行符，否则会在同一行输出

> printf "*format string*", *var1, var2*
> 其中*format string*格式为 *%[modifier]control-letter*
> %d或%i显示整数
> %e科学计数
> %f显示浮点 %2.1f保留一位小数
> %s显示字符串

`gawk 'BEGIN{FS=","} {printf "%s ", $1} END{printf "\n"}'`设定,为分隔符，打印第一列字符，由于没有换行，在最后输出一个换行符，这样第一列的字符都会输出在同一行内

`gawk 'BEGIN{FS="\n"; RS=""} {printf "%-16s %s\n", $1, $4}' data2` %16s 即输出宽度为16字符右对齐 %-16s 左对齐

> Ima Test
> 123 Main Street
> Chicago, IL 60601
> (312)555-1234
>
> Frank Tester
> 456 Oak Street
> Indianapolis, IN 46201
> (317)555-9876

默认情况下，RS是换行符 \n，表示记录是一行。这里将RS设为空白意味着，出现空白行之前看为一行，以换行符为分隔符，第4列即为电话列

十一、正则表达式

> 能用在sed与gawk

- 锚点字符
  - `^`锚定行首
    `sed -n '/^this/p'`这个只会匹配每行开头为this的行
  - `$`锚定行尾
    `sed -n '/end$/p'`这个只会匹配每行结尾为this的行
  - 组合锚点 删除空白行
    `sed '/^$/d'`强烈推荐
- 点号字符
  点号字符可以匹配除换行符之外的任意单个字符。点号字符必须匹配一个字符
- 字符组 []
  `echo "Yes" | sed -n '/[Yy]es/p'`
  `sed -n '^[0-9][0-9]$/p'` 这样可以匹配两位数
  `sed -n '[^ch]at$/p'` ^放到方括号内代表将其排除在外
- 星号 *星号表明前面的字符可以出现0次或多次*
  `echo "ik" | sed -n '/ie*k/p'` 这样依然可以匹配到

> 不能用在sed

- 问号 *问号表明前面的字符可以出现0次或1次*
- 加号 *加号表明前面的字符可以出现1次或多次*
- 花括号 {m,n} *表明前面的字符可以出现m次到n次*
  在默认情况下，gawk不识别正则表达式区间，必须指定gawk的命令行选项--re-interval才行。
  `echo "beet" | gawk --re-interval '/be{1}t/{print $0}'`
- 竖线 *竖线代表or*
  `gawk '/cat|dog/{print $0}'`
- 表达式分组 ()
  `gawk '/(c|b)a(b|t)/{print $0}'`

后记

一些乱七八糟的笔记，不保证正确性

du -sh .
sed -i “s/forward0/forward/g” crab3_forward0_Ntuple.py
`grep -o "$FIELD" "$FILE" | wc -l` 出现次数
`grep -A 5 "$FIELD"  "$FILE"` 出现后5行内容
kill $(jobs -p)
condor_rm -all
condor_rm -all -name lpcschedd6.fnal.gov lpc上删除condor jobs

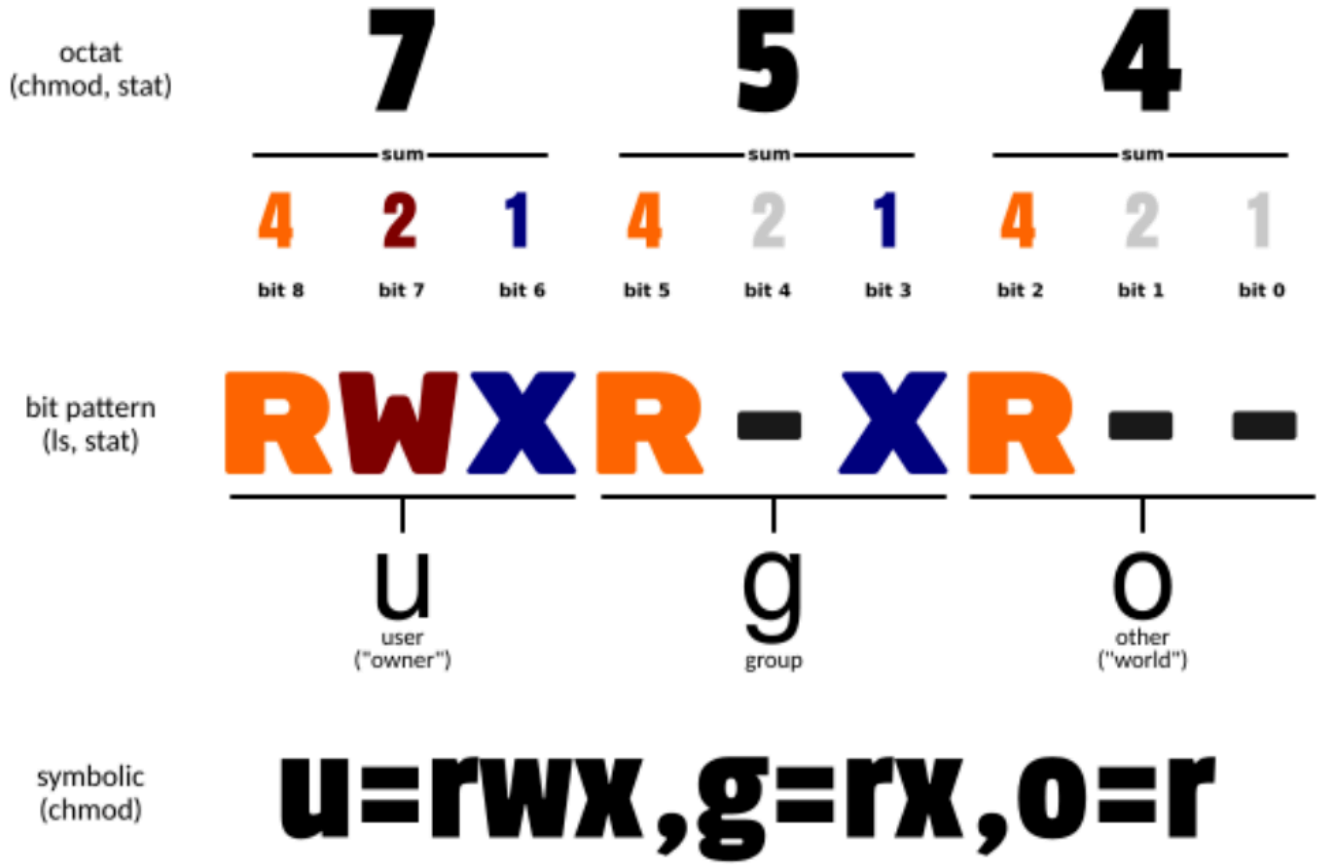
ls -lhtr 命令会以人类可读的方式，按照修改时间的倒序，列出当前工作目录中的所有文件和子目录的详细信息。这样可以方便地查看最近修改过的文件。
cat ion.C |wc 确定ioc的行数

s/old/new/gc 替换整个文件中出现的所有old，并在每次替换时提示。

在终端中编写shell脚本.sh文件时，可以使用以下方法来实现多行同时缩进：

1. 使用vim打开你的shell脚本文件，然后按ESC键进入命令模式。
2. 按下ctrl+v键进入可视块模式，然后使用上下键选中你想要缩进的多行代码。
3. 按下shift+i键进入插入模式，然后连续敲击空格键来添加缩进（注意这个时候只有一行会显示缩进）。
4. 最后按下esc键，你会发现所有选中的行都被缩进了1。
   如果你想要删除多行前面的空格，可以按照以下步骤操作：
5. 同样先按下ctrl+v键进入可视块模式，然后使用上下左右键选中你想要删除空格的多行代码以及前面的空格。
6. 然后按下del键即可删除选中的空格1。

chmod -R 755 public/ 设置文件夹及子文件夹的权限为rwxr-xr-x
r：read，可读取此文件的内容，即可以打开文件；
w：write，可编辑此文件的内容，如可以增加、删除、更改文件内容；
x：execute，可以执行此文件。



将cds上的root数据文件拷到服务器上
xrdcp -d 1 -f root://xrootd-cms.infn.it//

.h 头文件可以用crab那一步的cmsRun 产生的mymultilep.root文件获取
root /path/***.root
.ls //mkcands
mkcands->cd()
.ls //X_data
X_data->MakeClass("myntuple")

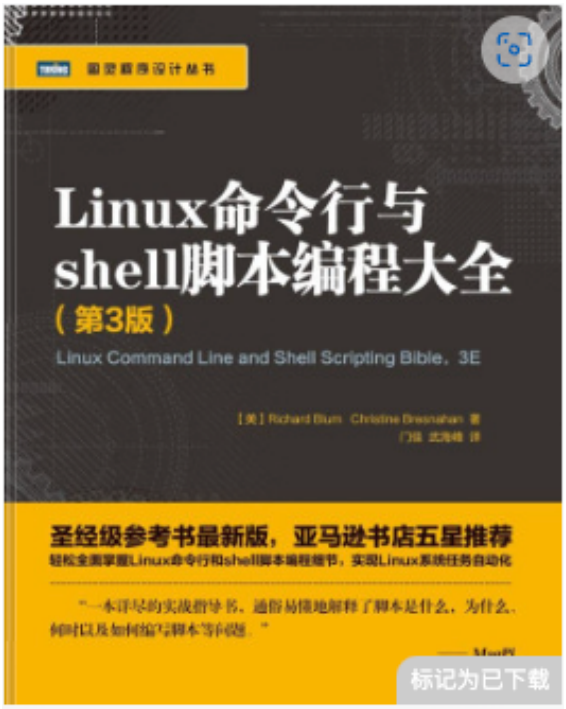
```
root [0]
Attaching file mymultilep.root as _file0...
(TFile *) 0x3e47710
root [1] .ls
TFile**         mymultilep.root
 TFile*         mymultilep.root
  KEY: TDirectoryFile   mkcands;1       mkcands
root [2] mkcands->cd()
(bool) true
root [3] .ls
TDirectoryFile*         mkcands mkcands
 KEY: TTree     X_data;8        X(3872) Data [current cycle]
 KEY: TTree     X_data;7        X(3872) Data [backup cycle]
 KEY: TTree     X_data;6        X(3872) Data [backup cycle]
root [4] X_data->MakeClass("myntuple")
```

Lpc会有时用不了有些命令
source /cvmfs/cms.cern.ch/cmsset_default.sh
source /cvmfs/cms.cern.ch/crab3/crab.sh

修改路径；清除；
scramv1 b ProjectRename
scramv1 b clean
scramv1 b

删除重复行
cat hh.txt | sort -g -k7 | uniq

disown -h %1 无敌的代码，我愿称之为神！后台挂起序号1的job，并且断线不发送停止代码，之前后台挂着，断线就会结束job



本文参考这本书，如果有疑问请自行下载查阅
linux命令行与shell脚本编程大全

END

written by zhuf

---

linux 进阶与服务器管理

零、一些小知识对之前的补充

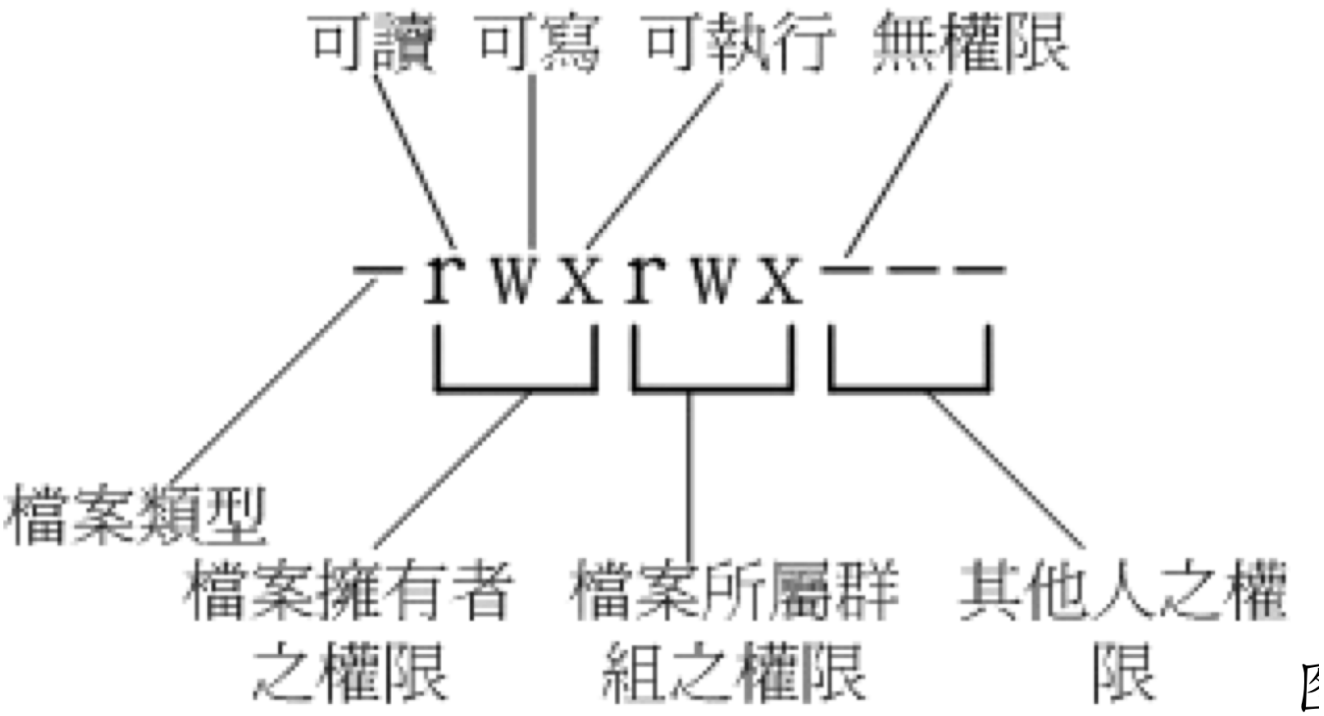
- 语言
  `locale` 显示当前的语系
  LANG="zh_CN.UTF-8"
  LC_COLLATE="zh_CN.UTF-8"
  LC_CTYPE="zh_CN.UTF-8"
  LC_MESSAGES="zh_CN.UTF-8"
  LC_MONETARY="zh_CN.UTF-8"
  LC_NUMERIC="zh_CN.UTF-8"
  LC_TIME="zh_CN.UTF-8"
  LC_ALL=

  `LANG=en_US.utf8` 修改输出信息为英文
  `export LC_ALL=en_US.utf8` 同步更新LC_ALL
- man指令
  当你不知道一个命令的用法，“男人”一下，你就知道，显示的手册（manual）通常会分为以下的几个部分

代号	内容
NAME	简短的指令说明，左上角除了指令外括号里面还会有一些数字，这几个比较重要：1，使用者可执行指令或文件；5，配置文件或某文件的格式；8，系统管理员可用的指令
SYNOPSIS	简短的指令语法
DESCRIPTION	完整说明
COMMANDS	当程序执行时，可以在程序中下达这个指令

- 文件权限
  linux的每个文件都有User,Group,Others三种身份的权限，`ls -al`查看文件权限

图5.2.1、文件属性的示意图



文件类型 [d] 目录 [-]文件 [l]链接文件

- 改变文件属性
  - chgrp 改变文件所属群组
    在`/etc/group`中是存在的群组名，用`chgrp [group name] [file name]`就可以将对应文件

    - chown 改变文件拥有者
      在`/etc/passwd`中是存在的使用者，用`chown -R [user name] [dirname/filename]`或者`chown -R [user name]:[group name] [dirname/filename]`可以修改群组和使用者所属，-R代表递归
    - chmod 改变文件的权限
      文件权限用数字加和的方式表示 r:4 w:2 x:1
      一些常用的权限数字
      755 将文件变为可执行文件，但不让其他人修改
      740 将文件不被别人看到

- 一些指令
  `who`能看到谁在偷偷工作
  `df` 系统中设备的使用情况

一、linux 文件与目录

1.1 / (root)根目录

目录	放置文件内容
/bin	bin下放置的是一些操作指令，比如cat, mv,cp
/boot	放置开机配置文件
/etc	系统的配置文件，包括人员账号密码
/lib	放置函数库
/tmp	正如其名，是临时文件
/usr	(Unix Software Resource)一些系统默认的软件，类似window的C盘
/dev	linux中每个设备都被当成一个文件，这个就是设备文件

软件源有很多，可以使用一些国内源
`/etc/apt/sources.list` or `/etc/apt/sources.list.d/` 放的是软件源链接
`sudo apt-get update` 通过这个命令更新软件源

1.2 $PATH 可执行文件路径的变量

当执行一个指令的时候"ls"，系统会依照PATH的设置去每个定义的目录下寻找文件名为ls的可执行文件进行指令执行。
以$PATH`/usr/local/bin:/usr/sbin:/usr/bin:/root/bin`为例，如果用root权限将/bin/ls移动到/root文件夹下，这样的情况下就不能执行`ls`命令了，只有当你用这样的命令`PATH="${PATH}:/root"`，将/root下的ls路径加入到$PATH中之后，才能正确的执行ls命令
如果有两个ls，则在$PATH中的目录先搜索到的ls会被执行

未完成。。。