本教程针对 Linux 后端服务器方面的知识

Linux 常用命令教程

处理目录的常用命令

接下来我们就来看几个常见的处理目录的命令吧：
ls：列出目录
cd：切换目录
pwd：显示目前的目录
mkdir：创建一个新的目录
rmdir：删除一个空的目录
cp：复制文件或目录
rm：移除文件或目录
mv：移动文件与目录，或修改文件与目录的名称
你可以使用 *man [命令]* 来查看各个命令的使用文档，如 ：man cp。

ls（列出目录）

在 Linux 系统当中， ls 命令可能是最常被运行的。语法：

```
1. 0voice@ubuntu:~$ ls [-aAdfFhilnrRSt] 目录名称
2. 0voice@ubuntu:~$ ls [--color={never,auto,always}] 目录名称
3. 0voice@ubuntu:~$ ls [--full-time] 目录名称
```

选项与参数：
-a ：全部的文件，连同隐藏档(开头为 . 的文件) 一起列出来(常用)
-d ：仅列出目录本身，而不是列出目录内的文件数据(常用)
-l ：长数据串列出，包含文件的属性与权限等等数据；(常用)
将家目录下的所有文件列出来(含属性与隐藏档)

```
1. 0voice@ubuntu:~$ ls -al
```

cd（切换目录）

cd 是 Change Directory 的缩写，这是用来变换工作目录的命令。语法：

```
1. cd [相对路径或绝对路径]
```

案例

```
# 表示回到自己的目录，亦即是 /0voice 这个目录
0voice@ubuntu:~$ cd ~

# 表示去到目前的上一级目录，亦即是 /0voice 的上一级目录的意思；
0voice@ubuntu:~$ cd ..
```

接下来大家多操作几次应该就可以很好的理解 cd 命令的。

pwd（显示目前所在的目录）

pwd 是 Print Working Directory 的缩写，也就是显示目前所在目录的命令。
选项与参数：
-P ：显示出确实的路径，而非使用连结 (link) 路径。
实例：单纯显示出目前的工作目录：

```
0voice@ubuntu:~$ pwd
/home/0voice   <== 显示出目录啦～
```

实例显示出实际的工作目录，而非连结档本身的目录名而已。

```
0voice@ubuntu:~$ cd /var/mail   <==注意，/var/mail 是一个连结档
0voice@ubuntu:/var/mail$ pwd
/var/mail         <==列出目前的工作目录
0voice@ubuntu:/var/mail$ pwd -P
/var/spool/mail   <==怎么回事？有没有加 -P 差很多～
0voice@ubuntu:/var/mail$ ls -ld /var/mail
lrwxrwxrwx 1 0voice 0voice 10 AUG 4 17:54 /var/mail -> spool/mail
# 看到这里应该知道为啥了吧？因为 /var/mail 是连结档，连结到 /var/spool/mail
# 所以，加上 pwd -P 的选项后，会不以连结档的数据显示，而是显示正确的完整路径啊！
```

mkdir（创建新目录）

如果想要创建新的目录的话，那么就使用 mkdir (make directory)吧。语法：

```
mkdir [-mp] 目录名称
```

选项与参数：
-m ：配置文件的权限喔！直接配置，不需要看默认权限 (umask) 的脸色～
-p ：帮助你直接将所需要的目录(包含上一级目录)递归创建起来！
实例：请到/tmp 底下尝试创建数个新目录看看：

```
0voice@ubuntu:~$ cd /tmp
0voice@ubuntu:tmp$ mkdir test    <==创建一名为 test 的新目录
0voice@ubuntu:tmp$ mkdir test1/test2/test3/test4
mkdir: cannot create directory `test1/test2/test3/test4':
No such file or directory       <== 没办法直接创建此目录啊!
0voice@ubuntu:tmp$ mkdir -p test1/test2/test3/test4
```

加了这个 -p 的选项，可以自行帮你创建多层目录！实例：创建权限为 rwx--x--x 的目录。

```
0voice@ubuntu:/tmp$ mkdir -m 711 test2
0voice@ubuntu:/tmp$ ls -l
drwxr-xr-x  3 0voice  0voice  4096 Jul 18 12:50 test
drwxr-xr-x  3 0voice  0voice  4096 Jul 18 12:53 test1
drwx--x--x  2 0voice  0voice  4096 Jul 18 12:54 test2
```

上面的权限部分，如果没有加上 -m 来强制配置属性，系统会使用默认属性。
如果我们使用 -m ，如上例我们给予 -m 711 来给予新的目录 drwx--x--x 的权限。

rmdir（删除空的目录）

语法：

```
rmdir [-p] 目录名称
```

选项与参数：
-p ：连同上一级『空的』目录也一起删除

删除 0voice 目录

```
0voice@ubuntu:/tmp$ rmdir 0voice/
```

将 mkdir 实例中创建的目录(/tmp 底下)删除掉！

```
0voice@ubuntu:/tmp$ ls -l   <==看看有多少目录存在?
drwxr-xr-x  3 0voice  0voice 4096 Jul 18 12:50 test
drwxr-xr-x  3 0voice  0voice 4096 Jul 18 12:53 test1
drwx--x--x  2 0voice  0voice 4096 Jul 18 12:54 test2
0voice@ubuntu:/tmp$ rmdir test   <==可直接删除掉，没问题
0voice@ubuntu:/tmp$ rmdir test1  <==因为尚有内容，所以无法删除!
rmdir: `test1': Directory not empty
0voice@ubuntu:/tmp$ rmdir -p test1/test2/test3/test4
0voice@ubuntu:/tmp$ ls -l        <==您看看，底下的输出中 test 与 test1 不见了!
drwx--x--x  2 0voice  0voice 4096 Jul 18 12:54 test2
```

利用 -p 这个选项，立刻就可以将 test1/test2/test3/test4 一次删除。不过要注意的是，这个 rmdir 仅能删除空的目录，你可以使用 rm 命令来删除非空目录。

cp（复制文件或目录）

cp 即拷贝文件和目录。语法:

```
1.  0voice@ubuntu:~$ cp [-adfilprsu] 来源档(source) 目标档(destination)
2.  0voice@ubuntu:~$ cp [options] source1 source2 source3 .... directory
```

选项与参数：
-a：相当於 -pdr 的意思，至於 pdr 请参考下列说明；(常用)
-d：若来源档为连结档的属性(link file)，则复制连结档属性而非文件本身；
-f：为强制(force)的意思，若目标文件已经存在且无法开启，则移除后再尝试一次；
-i：若目标档(destination)已经存在时，在覆盖时会先询问动作的进行(常用)
-l：进行硬式连结(hard link)的连结档创建，而非复制文件本身；
-p：连同文件的属性一起复制过去，而非使用默认属性(备份常用)；
-r：递归持续复制，用於目录的复制行为；(常用)
-s：复制成为符号连结档 (symbolic link)，亦即『捷径』文件；
-u：若 destination 比 source 旧才升级 destination ！
用 0voice 身份，将 0voice 目录下的 .bashrc 复制到 /tmp 下，并命名为 bashrc

```
1.  0voice@ubuntu:~$ cp ~/.bashrc /tmp/bashrc
2.  0voice@ubuntu:~$ cp -i ~/.bashrc /tmp/bashrc
3.  cp: overwrite `/tmp/bashrc'? n  <==n 不覆盖，y 为覆盖
```

rm（移除文件或目录）

语法：

```
1.  rm [-fir] 文件或目录
```

选项与参数：
-f ：就是 force 的意思，忽略不存在的文件，不会出现警告信息；
-i ：互动模式，在删除前会询问使用者是否动作
-r ：递归删除啊！最常用在目录的删除了！这是非常危险的选项!!!
将刚刚在 cp 的实例中创建的 bashrc 删除掉！

```
1.  0voice@ubuntu:~$ rm -i bashrc
2.  rm: remove regular file `bashrc'? y
```

如果加上 -i 的选项就会主动询问喔，避免你删除到错误的档名！

mv（移动文件与目录，或修改名称）

移动文件与目录，或者修改名称，语法：

```
1. 0voice@ubuntu:~$ mv [-fiu] source destination
2. 0voice@ubuntu:~$ mv [options] source1 source2 source3 .... directory
```

选项与参数：
-f ：force 强制的意思，如果目标文件已经存在，不会询问而直接覆盖；
-i ：若目标文件 (destination) 已经存在时，就会询问是否覆盖！
-u ：若目标文件已经存在，且 source 比较新，才会升级 (update)

复制一文件，创建一目录，将文件移动到目录中

```
1. 0voice@ubuntu:~$ cd /tmp
2. 0voice@ubuntu:/tmp$ cp ~/.bashrc bashrc
3. 0voice@ubuntu:/tmp$ mkdir mvtest
4. 0voice@ubuntu:/tmp$ mv bashrc mvtest
```

将某个文件移动到某个目录去，就是这样做！
将刚刚的目录名称更名为 mvtest2

```
1. 0voice@ubuntu:/tmp$ mv mvtest mvtest2
```

Linux 文件内容查看

Linux 系统中使用以下命令来查看文件的内容：
cat: 由第一行开始显示文件内容
tac: 从最后一行开始显示，可以看出 tac 是 cat 的倒著写！
nl: 显示的时候，顺道输出行号！
more: 一页一页的显示文件内容
less: 与 more 类似，但是比 more 更好的是，他可以往前翻页！
head: 只看头几行
tail: 只看尾巴几行

你可以使用 man [命令]来查看各个命令的使用文档，如 ：man cp。

cat(文本输出)

由第一行开始显示文件内容

语法：

```
1. cat [-AbEnTv]
```

选项与参数：
-A ：相当於 -vET 的整合选项，可列出一些特殊字符而不是空白而已；
-b ：列出行号，仅针对非空白行做行号显示，空白行不标行号！
-E ：将结尾的断行字节 $ 显示出来；
-n ：列印出行号，连同空白行也会有行号，与 -b 的选项不同；
-T ：将 [tab] 按键以 ^I 显示出来；
-v ：列出一些看不出来的特殊字符

检看 /etc/issue 这个文件的内容：

```
1. 0voice@ubuntu:~$ cat /etc/magic
2. # Magic local data for file(1) command.
3. # Insert here your local magic data. Format is described in magic(5).
```

tac（与 cat 相反）

tac 与 cat 命令刚好相反，文件内容从最后一行开始显示，可以看出 tac 是 cat 的倒着写！如：

```
1. 0voice@ubuntu:~$ tac /etc/magic
2. # Insert here your local magic data. Format is described in magic(5).
3. # Magic local data for file(1) command.
```

nl(显示行号)

显示行号，语法：

```
1. nl [-bnw] 文件
```

选项与参数：
-b ：指定行号指定的方式，主要有两种：
-b a ：表示不论是否为空行，也同样列出行号(类似 cat -n)；
-b t ：如果有空行，空的那一行不要列出行号(默认值)；
-n ：列出行号表示的方法，主要有三种：
-n ln ：行号在荧幕的最左方显示；
-n rn ：行号在自己栏位的最右方显示，且不加 0 ；
-n rz ：行号在自己栏位的最右方显示，且加 0 ；
-w ：行号栏位的占用的位数。

实例一：用 nl 列出 /etc/issue 的内容

```
0voice@ubuntu:~$ nl /etc/magic
     1  # Magic local data for file(1) command.
     2  # Insert here your local magic data. Format is described in magic(5)
```

more（一页一页翻动）

一页一页翻动

```
0voice@ubuntu:~$ more /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi
--More--(70%)
```

在 more 这个程序的运行过程中，你有几个按键可以按的：

空白键 (space)：代表向下翻一页；

Enter ：代表向下翻『一行』；

/字串：代表在这个显示的内容当中，向下搜寻『字串』这个关键字；

:f ：立刻显示出档名以及目前显示的行数；

q ：代表立刻离开 more ，不再显示该文件内容。

b 或 [ctrl]-b ：代表往回翻页，不过这动作只对文件有用，对管线无用。

less

一页一页翻动，以下实例输出/etc/man.config 文件的内容：

```
0voice@ubuntu:~$ less /etc/profile
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi
```

less 运行时可以输入的命令有：
空白键 ： 向下翻动一页；
[pagedown] ： 向下翻动一页；
[pageup] ： 向上翻动一页；
/字串 ： 向下搜寻『字串』的功能；
?字串 ： 向上搜寻『字串』的功能；
n ： 重复前一个搜寻 (与 / 或 ? 有关！)
N ： 反向的重复前一个搜寻 (与 / 或 ? 有关！)
q ： 离开 less 这个程序；

head

取出文件前面几行，语法：

```
head [-n number] 文件
```

选项与参数：

-n ：后面接数字，代表显示几行的意思

```
1.  0voice@ubuntu:~$ head -n 10  /etc/profile
```

默认的情况中，显示前面 10 行！若要显示前 20 行，就得要这样：

```
1.  0voice@ubuntu:~$ head -n 20  /etc/profile
```

tail（取出文件后面几行）

取出文件后面几行，语法：

```
1.  tail [-n number] 文件
```

选项与参数：
-n ：后面接数字，代表显示几行的意思
-f ：表示持续侦测后面所接的档名，要等到按下[ctrl]-c 才会结束 tail 的侦测

```
1.  0voice@ubuntu:~$ tail -n 10 /etc/profile
2.  0voice@ubuntu:~$ tail -n 20 /etc/profile
```

Shell 教程

Shell 是一个用 c 语言编写的程序，它是用户使用 Linux 的桥梁。Shell 既是命令语言，又是一种程序设计语言。
Shell 脚本（shell script），是一种为 shell 编写的脚本程序。业界所说的 shell 通常指的是 shell 脚本。Shell 与 Shell 脚本是两个不同的概念。
Linux 中的 shell 有很多种类，常用的几种：
1> Bourne Shell（/usr/bin/sh 或/bin/sh）
2> Bourne Again Shell（/bin/bash）
3> C Shell（/usr/bin/csh）
4> K Shell（/usr/bin/ksh）
5> Shell for Root（/sbin/sh）
本教程使用的是 Bash，也就是 Bourne Again Shell，由于易用和免费，Bash 在日常工作中被广泛使用。同时，Bash 也是大多数 Linux 系统默认的 Shell。

第一个 Shell 脚本

打开文本编辑器（vi/vim），新建一个文件 first.sh，扩展名为 sh（sh 代表 shell）。扩展名并不

影响脚本执行。
实例：

```
1. #!/bin/bash
2. echo “Hello World!”
```

#! 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 shell。
echo 命令用于向窗口输出文本。

运行 Shell 脚本方式
1. 作为可执行程序
   将上面的代码保存为 first.sh，并 cd 到相应目录：
   $ chmod +x first.sh
   $ ./first.sh
2. 作为解释器参数
   /bin/bash ./first.sh

Shell 变量

定义变量时，变量名不加美元符号（$），如：

```
1. domain="www.0voice.com"
```

注意，变量名与等号之间不能有空格，变量名的命名需遵循如下规则：
1> 命名只能使用英文字母，数字和下划线，首个字符不能以数字开头。
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）

例如，有效的 shell 变量名

```
1. zerovoice
2. ZERO_VOICE
3. _ZERO_VOICE
4. Zerovoice0
```

例如，无效的 shell 变量名

```
1. 0voice
2. ?voice
3. zero*voice
```

使用变量

使用一个定义过的变量，只要在变量名前面加美元符号即可，如：

```
1.  domain="www.0voice.com"
2.  echo $domain
3.  echo ${domain}
```

变量外的大括号，$domain 与${domain}效果一样。也是为了帮助解释器识别变量边界。

```
1.  for skill in C CPP Linux Shell; do
2.      echo "I am good at ${skill}Code"
3.  done
```

如果不给 skill 变量加上大括号，写成了 echo "I am good at ${skill}Code"，解释器就会把$skillCode 当成一个变量。

只读变量

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变。下面的例子尝试更改只读变量，结果报错：

```
1.  #!/bin/bash
2.  url="http://www.google.com"
3.  readonly url
4.  url="http://www.0voice.com"
```

运行脚本，结果如下：

```
./var.sh: line 4: url: readonly variable
```

删除变量

使用 unset 命令可以删除变量。语法：

```
1.  unset variable_name
```

变量被删除后不能再次使用。unset 命令不能删除只读变量。

实例

```
1.  #!/bin/bash
2.  url="http://www.0voice.com"
```

```
3. unset url
4. echo $url
```

以上实例执行将没有任何输出。

变量类型

运行 shell 时，会同时存在三种变量：
1）局部变量 局部变量在脚本或命令中定义，仅在当前 shell 实例中有效，其他 shell 启动的程序不能访问局部变量。
2）环境变量 所有的程序，包括 shell 启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候 shell 脚本也可以定义环境变量。
3）shell 变量 shell 变量是由 shell 程序设置的特殊变量。shell 变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了 shell 的正常运行

Shell 字符串

字符串是 shell 编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。

单引号

单引号字符串的限制：

```
1. str='this is a string'
```

单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；
单引号字串中不能出现单独一个的单引号（对单引号使用转义符后也不行），但可成对出现，作为字符串拼接使用。

双引号

```
1. your_name='0voice'
2. str="Hello, I know you are \"$your_name\"! \n"
3. echo -e $str
```

输出结果为：

```
1. Hello, I know you are "0voice"!
```

双引号的优点：

1. 双引号里可以有变量
2. 双引号里可以出现转义字符

拼接字符串

```
your_name="0voice"
# 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
```

输出结果为：

```
hello, 0voice ! hello, 0voice !
```

获取字符串长度

```
string="abcd"
echo ${#string} #输出 4
```

提取子字符串

以下实例从字符串第 2 个字符开始截取 4 个字符：

```
string="0voice is a great college"
echo ${string:1:4} # 输出 voic
```

查找子字符串

查找字符 i 或 o 的位置(哪个字母先出现就计算哪个)：

```
string="0voice is a great college "
echo `expr index "$string" io`  # 输出 3
```

注意： 以上脚本中 ` 是反引号，而不是单引号 '，不要看错了哦。

Shell 数组

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。

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。

定义数组

在 Shell 中，用括号来表示数组，数组元素用"空格"符号分割开。定义数组的一般形式为：

```
1. 数组名=(值 1 值 2 ... 值 n)
```

例如：

```
1. array_name=(value0 value1 value2 value3)
```

还可以单独定义数组的各个分量：

```
1. array_name[0]=value0
2. array_name[1]=value1
3. array_name[n]=valuen
```

可以不使用连续的下标，而且下标的范围没有限制。

读取数组

读取数组元素值的一般格式是：

```
1. ${数组名[下标]}
```

例如：

```
1. valuen=${array_name[n]}
```

使用@符号可以获取数组中的所有元素，例如：

```
1. echo ${array_name[@]}
```

获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同，例如：

```
1. # 取得数组元素的个数
2. length=${#array_name[@]}
3. # 或者
4. length=${#array_name[*]}
```

```
# 取得数组单个元素的长度
lengthn=${#array_name[n]}
```

Shell 注释

以 # 开头的行就是注释，会被解释器忽略。通过每一行加一个 # 号设置多行注释，像这样：

```
#--------------------------------------------
# slogan：一切只为渴望更优秀的你
#--------------------------------------------
##### 用户配置区 开始 #####
#
#
# 这里可以添加脚本描述信息
#
#
##### 用户配置区 结束  #####
```

如果在开发过程中，遇到大段的代码需要临时注释起来，过一会儿又取消注释，怎么办呢？每一行加个#符号太费力了，可以把这一段要注释的代码用一对花括号括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行，达到了和注释一样的效果。

多行注释

多行注释还可以使用以下格式：

```
:<<EOF
注释内容...
注释内容...
注释内容...
EOF
```

EOF 也可以使用其他符号:

```
:<<'
注释内容...
注释内容...
注释内容...
'

:<<!
注释内容...
注释内容...
```

```
10. 注释内容...
11. !
```