



# Linux知识手册

## 目录

### awk

- [1.语法](#)
- [2.基本用法](#)
- [3.运算符](#)
- [4.使用正则，字符串匹配](#)
- [5.忽略大小写](#)
- [6.模式取反](#)
- [7.awk脚本](#)
- [8.计算文件大小](#)
- [9.从文件中找出长度大于 80 的行](#)
- [10.打印九九乘法表](#)
- [11.创建数组](#)
- [12.删除数组元素](#)
- [13.多维数组](#)
- [14.查找字符串 \(index 使用\)](#)
- [15.正则表达式匹配查找\(match 使用\)](#)
- [16.截取字符串\(substr使用\)](#)
- [17.字符串分割 \(split使用\)](#)
- [参考资料](#)

公众号:程序员百科全书

## grep

- 1.在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。
- 2.以递归的方式查找符合条件的文件。例如，查找指定目录/etc/acpi 及其子目录（如果存在子目录的话）下所有文件中包含字符串"update"的文件。
- 3.反向查找。查找文件名中包含 test 的文件中不包含test 的行。
- 4.系统报警显示了时间，但是日志文件太大无法直接 cat 查看。（查询含有特定文本的文件，并拿到这些文本所在的行）
- 5.在当前目录里第一级文件夹中寻找包含指定字符串的 .in 文件
- 6.从文件内容查找与正则表达式匹配的行
- 7.查找时不区分大小写
- 8.查找匹配的行数
- 9.从文件内容查找不匹配指定字符串的行
- 10.从根目录开始查找所有扩展名为 .log 的文本文件，并找出包含 "ERROR" 的行
- 11.从当前目录开始查找所有扩展名为 .in 的文本文件，并找出包含 "thermcontact" 的行
- 12.查找指定进程
- 13.查找指定进程个数
- 参考资料

## sed

- 1.删除centos7系统/etc/grub2.cfg文件中所有以空白开头的行行首的空白字符
- 2.在/tmp/file.txt文件中不以#开头的行的行首增加#号
- 3.用命令行更改/tmp/file.txt文件，把里面所有的“name”更改为“address”
- 4.使用sed命令打印出/tmp/file.txt文件的第一行到第三行
- 5.删除/etc/fstab文件中所有以#开头，后面至少跟一个空白字符的行的行首的# 和空白字符
- 6.在/etc/fstab文件中不以#开头的行的行首增加#号
- 7.处理/etc/fstab路径,使用sed命令取出其目录名和基名
- 8.利用sed 取出ifconfig命令中本机的IPv4地址
- 9.将文本文件的n和n+1行合并为一行， n为奇数行
- 10.在每一行后增加一空行？
- 11.将/tmp/file.txt文件中第2到第8行之间所有大写字母替换成小写字母
- 12.使用sed找出/tmp/file.txt文件中包含oldboy的行
- 13.将/tmp/file.txt文件中以; 结尾的行，行首插入#
- 14.删除file.txt文件中的空行
- 15.使用sed将selinux彻底关闭
- 参考链接

## vim

- 1.进入vim编辑器
- 2.vi / vim 的三种工作方式
- 3.模式之间的转换
- 4.vim如何插入字符
- 5.vim如何快速定位到某行

- 6.如何删除字符
- 7.用vi命令编辑text.txt，如何跳转到末行、首行、行首、行末，如何在光标下一行插入，如何复制5行、删除10行、查找jingfeng的字符、把jingfeng替换为jfedu.NET
- 8.vim操作，将第9行至第15行的数据、复制到第16行 !!!!!!!
- 9.有如下文本，保存在文件a.txt中，要求将所有ghi替换为xyz。请写出linux系统下所有可能的方法（不限制语言） !!!!!!!
- 10.退出 vi / vim 命令
- 参考链接

## 备份压缩

- 1.打包跟压缩的区别：
- 2.Linux主要压缩方式
- 3.文件后缀名.tar.gz 是用哪种方式压缩的
- 4..文件后缀名.tar.bz2是用哪种方式压缩的
- 5.压缩多个文件
- 6.zip格式
- 7.zip压缩文件
- 8.将所有.jpg 的文件打成一个名为 all.tar 的包
- 9.将所有.gif 的文件增加到 all.tar 的包里面去
- 10.将目录里所有jpg文件打包成 tar.jpg
- 11. gzip 中 将压缩数据输出到标准输出中，并保留源文件的选项是
- 12.如果一个系统没有备份策略，请写出一个较为全面合理的备份方案
- 13.完全备份
- 14.累计增量备份
- 15.差异增量备份
- 16.如果网站服务器每天产生的日志数量较大，请问如何备份？
- 参考链接

## 变量与环境变量

- 1.变量
- 2.环境变量
- 3.Bash 解释器
- 4.为什么不能将当前目录(.) 添加到PATH中呢？
- 5.修改环境变量PATH的三种方法
- 参考链接

## 查找和文件搜索

- 1.列出当前目录和子目录下的所有文件
- 2.查找特殊的目录或路径
- 3.查找/etc目录中文件名为passwd的文件
- 4.按文件所在的深度（层次） 查找
- 5.反向查找
- 6. 结合多个查找条件
- 7.只查找文件或目录

- [8.同时在多个目录下查找](#)
- [9.查找隐藏文件](#)
- [10.查找指定权限的文件](#)
- [11.locate](#)
- [参考链接](#)

## 磁盘管理

- [1.将系统内所有的文件系统列出来](#)
- [2.将容量结果以易读的容量格式显示出来](#)
- [3.将系统内的所有特殊文件格式及名称都列出来](#)
- [4.将 /etc 底下的可用的磁盘容量以易读的容量格式显示](#)
- [5.只列出当前目录下的所有文件夹容量（包括隐藏文件夹）](#)
- [6.将文件的容量也列出来](#)
- [7.检查根目录下每个目录所占用的容量](#)
- [8.列出所有分区信息](#)
- [9.找出你系统中的根目录所在磁盘，并查阅该硬盘内的相关信息](#)
- [10.将分区 /dev/hdc6（可指定你自己的分区）格式化为 ext3 文件系统](#)
- [11.强制检测 /dev/hdc6 分区](#)
- [12.用默认的方式，将刚刚创建的 /dev/hdc6 挂载到 /mnt/hdc6 上面](#)
- [13.卸载 /dev/hdc6](#)
- [参考资料](#)

## 计划任务

- [1.什么是crond](#)
- [2.为什么要使用crond](#)
- [3.计划任务有哪几种?](#)
- [4.crontab配置文件记录了时间周期的含义](#)
- [5.了解crontab的时间编写规范](#)
- [6.使用crontab编写cron定时任务](#)
- [7.使用root用户每5分钟执行一次时间同步](#)
- [8.每天的下午3,5点，每隔半小时执行一次sync命令](#)
- [9.crond如何备份](#)
- [10.crond如何拒绝某个用户使用](#)
- [11.crond调试](#)
- [12.crond编写思路](#)
- [参考资料](#)

## 软件包管理

- [1.rpm](#)
- [2.如何安装软件包](#)
- [3.测试安装软件包，不做真实的安装](#)
- [4.如何使用 rpm 初始化数据库?](#)
- [5.升级软件包](#)
- [6.怎样查询一个已安装软件包的信息?](#)

- [7.yum](#)
- [8.列出所有可更新的软件清单命令\(yum\)](#)
- [9.更新所有软件命令](#)
- [10.删除软件包命令](#)
- [参考链接](#)

## 网络通信

- [1.为“ssh”生成、管理和转换认证密钥的命令是](#)
- [2.如何指定登录用户](#)
- [3.指定端口登录](#)
- [4.ssh创建密钥](#)
- [5.删除主机密钥](#)
- [6.列出所有的端口](#)
- [7.列出TCP协议的端口](#)
- [8.UDP协议的端口](#)
- [9.列出处于监听状态的socket](#)
- [10.列出监听的TCP端口](#)
- [11.网络连接状态有哪些](#)
- [12.ping](#)
- [13.如何使用ping](#)
- [14.ifconfig](#)
- [15.配置网卡的IP地址,在eth0上配置上192.168.0.1的IP地址及24位掩码](#)
- [16.若想再在eth0上在配置一个192.168.1.1/24的IP地址怎么办?](#)
- [17.配置网卡的硬件地址](#)
- [18.将网卡禁用](#)
- [19.将网卡启用](#)
- [20.在指定网络接口上发出DHCP请求](#)
- [参考链接](#)

## 文件传输

- [1.scp](#)
- [2.rcp](#)
- [3.wget](#)
- [4.rsync](#)
- [参考资料](#)

## 文件管理

- [1.显示当前工作目录的绝对路径](#)
- [2.查看当前目录的所有内容信息](#)
- [3.显示当前目录所有的文件和目录,包括隐藏的](#)
- [4.以行的形式显示当前目录所有的文件和目录,包括隐藏的](#)
- [5.显示/etc目录下,所有.conf结尾,且以m,n,r,p开头的文件或目录](#)
- [6.显示 /var 目录下所有以l开头,以一个小写字母结尾,且中间出现至少一位数的文件或目录](#)

- 7.显示/etc目录下以任意一位数字开头，且以非数字结尾的文件或目录
- 8.显示/etc/目录下以非字母开头，后面跟了一个字母及其它任意长度任意字符的文件或目录
- 9、显示/etc/目录下所有文件名以rc开头，并且后面是 0 到 6 中的数字，其它为任意字符的文件或目录
- 10、显示 /etc 目录下，所有以 .d 结尾的文件或目录
- 11.回到自己的家目录
- 12.回到当前目录的上一级目录
- 13.回到当前目录的上上一级目录
- 14.在home下创建dog目录，只能创建一级目录
- 15.创建空文件 test.txt
- 16.同时创建子目录dir1, dir2, dir3
- 17.递归创建目录
- 18.复制1234.txt文件到新文件2345.txt
- 19.如果复制后的新文件名已存在会怎样呢？
- 20.为避免不知道有没有同名文件被覆盖,需要添加哪个选项
- 21.将file 1.txt文件从当前目录移动到其它目录，以/home/pungki/为例
- 22.将file1.txt重命名为file2.txt
- 23.除了mv，你还知道其他的修改文件名方式吗
- 参考链接

## 系统信息

- 1.查看Linux内核版本命令
- 2.查看Linux系统版本的命令
- 3.Linux查看版本当前操作系统发行版信息
- 4.查看服务器名称
- 5.查看网络信息
- 6.查看CPU信息
- 7.列出所有PCI设备
- 8.查看环境变量
- 9.查看活动用户
- 10.列出所有系统服务
- 11.内存管理的必要性
- 12.虚拟地址
- 13.虚拟地址的好处
- 14.物理地址
- 15.页的概念
- 16.swap对换空间
- 17.缺页中断定义
- 18.缺页中断的次数
- 19.页面置换算法
- 20.如何杀死某个进程
- 21.显示内存使用情况
- 22.解释free -m的输出
- 23.进程
- 24.列出所有运行中/激活进程
- 25.列出需要进程
- 26.以树状结构显示进程关系

- [27.杀进程的命令](#)
- [28.进程的状态](#)
- [29.僵尸进程](#)
- [30.查看环境变量](#)
- [31.孤儿进程](#)
- [32.如何杀死某个进程](#)
- [参考链接](#)

## 用户管理

- [1.添加新用户](#)
- [2.删除帐号](#)
- [3.修改帐号](#)
- [4.用户密码管理](#)
- [5.添加新用户组](#)
- [6.删除用户组](#)
- [7.修改用户组属性](#)
- [8.切换到其他用户组](#)
- [9.批量添加用户](#)
- [10.普通用户切换到root](#)
- [11.root用户切换到普通用户](#)
- [参考资料](#)

## Shell echo 命令

- [1.语法](#)
- [2.显示普通字符串:](#)
- [3.显示转义字符](#)
- [4.显示变量](#)
- [5.显示换行](#)
- [6.显示不换行](#)
- [7.显示结果定向至文件](#)
- [8.原样输出字符串, 不进行转义或取变量\(用单引号\)](#)
- [9.显示命令执行结果](#)
- [参考链接](#)

## Shell printf命令

- [1.语法](#)
- [2.printf 的转义序列](#)
- [参考链接](#)

# Shell test命令

- [1.test命令](#)
- [2.数值测试](#)
- [3.字符串测试](#)
- [4.文件测试](#)
- [参考链接](#)

# Shell变量

- [1.定义变量](#)
- [2.使用变量](#)
- [3.只读变量](#)
- [4.删除变量](#)
- [5.变量类型](#)
- [Shell 字符串](#)
- [6.单引号](#)
- [7.双引号](#)
- [8.拼接字符串](#)
- [9.获取字符串长度](#)
- [10.提取子字符串](#)
- [11.查找子字符串](#)
- [Shell 数组](#)
- [12.定义数组](#)
- [13.读取数组](#)
- [14.获取数组的长度](#)
- [Shell 注释](#)
- [15.多行注释](#)
- [参考链接](#)

公众号:程序员百科全书

# Shell 传递参数

- [实例](#)
- [参考链接](#)

# Shell 函数

- [1.语法](#)
- [2.函数参数](#)
- [参考链接](#)



# Shell 基本运算符

- [1.概念](#)
- [2.算术运算符](#)
- [3.关系运算符](#)
- [4.布尔运算符](#)
- [5.逻辑运算符](#)
- [6.字符串运算符](#)
- [7.文件测试运算符](#)

# Shell 流程控制

- [1.流程控制](#)
- [2.if else](#)
- [3.if else](#)
- [4.if else-if else](#)
- [5.for 循环](#)
- [6.while 语句](#)
- [7.无限循环](#)
- [8.until 循环](#)
- [9.case ... esac](#)
- [10.跳出循环](#)
- [11.break 命令](#)
- [12.continue](#)
- [参考链接](#)

# Shell 输入 / 输出重定向

- [1.定义](#)
- [2.输出重定向](#)
- [3.输入重定向](#)
- [4.重定向深入讲解](#)
- [5.Here Document](#)
- [6./dev/null 文件](#)
- [参考链接](#)

# Shell 数组

- [1.语法](#)
- [2.读取数组](#)
- [3.获取数组中的所有元素](#)
- [4.获取数组的长度](#)
- [参考链接](#)

# Shell文件包含

- [1.语法](#)
- [2.创建shell脚本](#)
- [参考链接](#)

## awk

### 1.语法

```
awk [选项参数] 'script' var=value file(s)
或
awk [选项参数] -f scriptfile var=value file(s)
```

选项参数说明：

**-F fs or --field-separator fs**

指定输入文件折分隔符，**fs**是一个字符串或者是一个正则表达式，如**-F:**。

**-v var=value or --assign var=value**

赋值一个用户定义变量。

**-f scripfile or --file scriptfile**

从脚本文件中读取**awk**命令。

**-mf nnn and -mr nnn**

对**nnn**值设置内在限制，**-mf**选项限制分配给**nnn**的最大块数目；**-mr**选项限制记录的最大数目。这两个功能是Bell实验室版**awk**的扩展功能，在标准**awk**中不适用。

**-W compact or --compat, -W traditional or --traditional**

在兼容模式下运行**awk**。所以**gawk**的行为和标准的**awk**完全一样，所有的**awk**扩展都被忽略。

**-W copleft or --copleft, -W copyright or --copyright**

打印简短的版权信息。

**-W help or --help, -W usage or --usage**

打印全部**awk**选项和每个选项的简短说明。

**-W lint or --lint**

打印不能向传统**unix**平台移植的结构的警告。

**-W lint-old or --lint-old**

打印关于不能向传统**unix**平台移植的结构的警告。

**-W posix**

打开兼容模式。但有以下限制，不识别：**/x**、函数关键字、**func**、换码序列以及当**fs**是一个空格时，将新行作为一个域分隔符；操作符**=**不能代替**^**和**^=**；**fflush**无效。

**-W re-interval or --re-interval**

允许间隔正则表达式的使用，参考(**grep**中的Posix字符类)，如括号表达式**[[alpha:]]**。

**-W source program-text or --source program-text**

使用program-text作为源代码，可与-f命令混用。

-W version or --version

打印bug报告信息的版本。

## 2.基本用法

log.txt文本内容如下：

```
2 this is a test
3 Are you like awk
This's a test
10 There are orange,apple,mongo
```

用法一：

```
awk '{[pattern] action}' {filenames}    # 行匹配语句 awk '' 只能用单引号
```

示例：

```
# 每行按空格或TAB分割，输出文本中的1、4项
$ awk '{print $1,$4}' log.txt
-----
2 a
3 like
This's
10 orange,apple,mongo
# 格式化输出
$ awk '{printf "%-8s %-10s\n",$1,$4}' log.txt
-----
2          a
3          like
This's
10         orange,apple,mongo
```

用法二：

```
awk -F    #-F相当于内置变量FS，指定分割字符
```

示例：

```
# 使用","分割
$ awk -F, '{print $1,$2}' log.txt
-----
2 this is a test
3 Are you like awk
This's a test
10 There are orange apple
```

```
# 或者使用内建变量
$ awk 'BEGIN{FS=","} {print $1,$2}' log.txt
-----
2 this is a test
3 Are you like awk
This's a test
10 There are orange apple
# 使用多个分隔符.先使用空格分割, 然后对分割结果再使用","分割
$ awk -F '[ ,]' '{print $1,$2,$5}' log.txt
-----
2 this test
3 Are awk
This's a
10 There apple
```

用法三:

```
awk -v # 设置变量
```

示例:

```
$ awk -va=1 '{print $1,$1+a}' log.txt
-----
2 3
3 4
This's 1
10 11
$ awk -va=1 -vb=s '{print $1,$1+a,$1b}' log.txt
-----
2 3 2s
3 4 3s
This's 1 This'ss
10 11 10s
```

用法四:

```
awk -f {awk脚本} {文件名}
```

示例:

```
$ awk -f cal.awk log.txt
```

### 3.运算符

过滤第一列大于2的行

```
$ awk '$1>2' log.txt      #命令
#输出
3 Are you like awk
This's a test
10 There are orange,apple,mongo
```

过滤第一列等于2的行

```
$ awk '$1==2 {print $1,$3}' log.txt      #命令
#输出
2 is
```

过滤第一列大于2并且第二列等于'Are'的行

```
$ awk '$1>2 && $2=="Are" {print $1,$2,$3}' log.txt      #命令
#输出
3 Are you
```

### 4.使用正则，字符串匹配

~ 表示模式开始。// 中是模式。

```
# 输出第二列包含 "th"，并打印第二列与第四列
$ awk '$2 ~ /th/ {print $2,$4}' log.txt
-----
this a
```

### 5.忽略大小写

```
$ awk 'BEGIN{IGNORECASE=1} /this/' log.txt
-----
2 this is a test
This's a test
```

### 6.模式取反

```
$ awk '$2 !~ /th/ {print $2,$4}' log.txt
```

```
-----  
Are like
```

```
a
```

```
There orange,apple,mongo
```

```
$ awk '!/th/ {print $2,$4}' log.txt
```

```
-----  
Are like
```

```
a
```

```
There orange,apple,mongo
```

## 7.awk脚本

关于 awk 脚本，我们需要注意两个关键词 BEGIN 和 END。

BEGIN{ 这里面放的是执行前的语句 }

END {这里面放的是处理完所有的行后要执行的语句 }

{这里面放的是处理每一行时要执行的语句}

假设有这么一个文件（学生成绩表）：

```
$ cat score.txt
```

```
Marry    2143  78  84  77
```

```
Jack     2321  66  78  45
```

```
Tom       2122  48  77  71
```

```
Mike     2537  87  97  95
```

```
Bob       2415  40  57  62
```

我们的 awk 脚本如下：

```
$ cat cal.awk
```

```
#!/bin/awk -f
```

```
#运行前
```

```
BEGIN {
```

```
    math = 0
```

```
    english = 0
```

```
    computer = 0
```

```
    printf "NAME      NO.      MATH  ENGLISH  COMPUTER  TOTAL\n"
```

```
    printf "-----\n"
```

```
}
```

```
#运行中
```

```
{
```

```
    math+=$3
```

```
    english+=$4
```

```
    computer+=$5
```

```
    printf "%-6s %-6s %4d %8d %8d %8d\n", $1, $2, $3,$4,$5, $3+$4+$5
```

```
}
```

```
#运行后
```

```
END {
```

```

printf "-----\n"
printf "  TOTAL:%10d %8d %8d \n", math, english, computer
printf "AVERAGE:%10.2f %8.2f %8.2f\n", math/NR, english/NR,
computer/NR
}

```

我们来看一下执行结果：

```

$ awk -f cal.awk score.txt
NAME      NO.      MATH    ENGLISH  COMPUTER  TOTAL
-----
Marry     2143      78      84       77        239
Jack      2321      66      78       45        189
Tom       2122      48      77       71        196
Mike      2537      87      97       95        279
Bob       2415      40      57       62        159
-----
TOTAL:           319      393      350
AVERAGE:        63.80    78.60    70.00

```

## 8.计算文件大小

```

$ ls -l *.txt | awk '{sum+=$5} END {print sum}'
-----
666581

```

## 9.从文件中找出长度大于 80 的行

```
awk 'length>80' log.txt
```

## 10.打印九九乘法表

```

seq 9 | sed 'H;g' | awk -v RS=''
' {for (i=1;i<=NF;i++) printf ("%dx%d=%d%s", i, NR, i*NR, i==NR?"\n":"\t") } '

```

## 11.创建数组

```

$ awk 'BEGIN {
sites["runoob"]="www.runoob.com";
sites["google"]="www.google.com"
print sites["runoob"] "\n" sites["google"]
}'

```

执行以上命令，输出结果为：

```

www.runoob.com
www.google.com

```

## 12.删除数组元素

下面的例子中，数组中的 `google` 元素被删除（删除命令没有输出）：

```
$ awk 'BEGIN {
  sites["runoob"]="www.runoob.com";
  sites["google"]="www.google.com"
  delete sites["google"];
  print fruits["google"]
}'
```

## 13.多维数组

下面是模拟二维数组的例子：

```
$ awk 'BEGIN {
  array["0,0"] = 100;
  array["0,1"] = 200;
  array["0,2"] = 300;
  array["1,0"] = 400;
  array["1,1"] = 500;
  array["1,2"] = 600;
  # 输出数组元素
  print "array[0,0] = " array["0,0"];
  print "array[0,1] = " array["0,1"];
  print "array[0,2] = " array["0,2"];
  print "array[1,0] = " array["1,0"];
  print "array[1,1] = " array["1,1"];
  print "array[1,2] = " array["1,2"];
}'
```

执行上面的命令可以得到如下结果：

```
array[0,0] = 100
array[0,1] = 200
array[0,2] = 300
array[1,0] = 400
array[1,1] = 500
array[1,2] = 600
```

## 14.查找字符串 (index 使用)

使用了三元运算符: 表达式 ? 动作1 : 动作2



```
$ awk 'BEGIN{info="this is a test2012test!";print  
index(info,"11111")?"ok":"no found";}'  
no found  
$ awk 'BEGIN{info="this is a test2012test!";print  
index(info,"is")?"ok":"no found";}'  
ok  
$ awk 'BEGIN{info="this is a test2012test!";print  
index(info,"test")?"ok":"no found";}'  
ok
```

## 15.正则表达式匹配查找(match 使用)

```
$ awk 'BEGIN{info="this is a test2012test!";print match(info,/ [0-  
9]+/)? "ok": "no found";}'  
ok
```

## 16.截取字符串(substr使用)

从第 4 个 字符开始，截取 10 个长度字符串。

```
$ awk 'BEGIN{info="this is a test2012test!";print substr(info,4,10);}'  
s is a tes
```

## 17.字符串分割 (split使用)

```
$ awk 'BEGIN{info="this is a test";split(info,tA," ");print  
length(tA);for(k in tA){print k,tA[k];}}'  
4  
2 is  
3 a  
4 test  
1 this
```

## 参考资料

<https://www.runoob.com/linux/linux-comm-awk.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# grep

1.在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。

```
grep test *file
```

2.以递归的方式查找符合条件的文件。例如，查找指定目录/etc/acpi 及其子目录（如果存在子目录的话）下所有文件中包含字符串"update"的文件。

```
grep -r update /etc/acpi
```

3.反向查找。查找文件名中包含 test 的文件中不包含test 的行。

```
grep -v test *test*
```

4.系统报警显示了时间，但是日志文件太大无法直接 cat 查看。(查询含有特定文本的文件，并拿到这些文本所在的行)

```
grep -n '2019-10-24 00:01:11' *.log
```

5.在当前目录里第一级文件夹中寻找包含指定字符串的.in 文件

```
grep "thermcontact" /.in
```

6.从文件内容查找与正则表达式匹配的行

```
grep -e "正则表达式" 文件名
```

7.查找时不区分大小写

```
grep -i "被查找的字符串" 文件名
```

8.查找匹配的行数

```
grep -c "被查找的字符串" 文件名
```

9.从文件内容查找不匹配指定字符串的行

```
grep -v "被查找的字符串" 文件名
```

## 10.从根目录开始查找所有扩展名为.log的文本文件，并找出包含"ERROR"的行

```
find / -type f -name "*.log" | xargs grep "ERROR"
```

## 11.从当前目录开始查找所有扩展名为.in的文本文件，并找出包含"thermcontact"的行

```
find . -name "*.in" | xargs grep "thermcontact"
```

## 12.查找指定进程

```
ps -ef|grep java
```

## 13.查找指定进程个数

```
ps -ef|grep -c java
```

## 参考资料

<https://www.runoob.com/linux/linux-comm-grep.html>

<https://blog.csdn.net/llljllj/article/details/89810340>



扫码关注，收获知识

程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## sed

### 1.删除centos7系统/etc/grub2.cfg文件中所有以空白开头的行行首的空白字符

```
sed -r 's/^[[:blank:]]+//' /etc/grub2.cfg
```

### 2.在/tmp/file.txt文件中不以#开头的行的行首增加#号

```
[root@web01 shell]# sed -n '/^[ a-zA-Z]/p' /tmp/file.txt | sed 's/^/#/g'
```

### 3.用命令行更改/tmp/file.txt文件，把里面所有的“name”更改为“address”

```
[root@web01 shell]# sed 's/name/address/g' /tmp/file.txt
```

### 4.使用sed命令打印出/tmp/file.txt文件的第一行到第三行

```
[root@web01 sed]# sed -n '2,3p' /tmp/file.txt
```

### 5.删除/etc/fstab文件中所有以#开头，后面至少跟一个空白字符的行的行首的# 和空白字符

```
[root@centos7 ~]# sed -r 's/^[[:space:]]+/#/g' /etc/fstab
```

### 6.在/etc/fstab文件中不以#开头的行的行首增加#号

```
[root@centos7 ~]# sed -r 's/^[^#]/#&/g' /etc/fstab
```

```
[root@centos7 ~]# sed -r '/^[^#]/s@^#@#' /etc/fstab
```

### 7.处理/etc/fstab路径,使用sed命令取出其目录名和基名

```
[root@centos7 ~]# echo "etc/fstab/dd/" | sed -r 's@^(.*)/(.+)${@1@'
[root@centos7 ~]# echo "etc/fstab/dd/" | sed -r 's@^(.*)/(.+)${@2@'
dd/
```

### 8.利用sed 取出ifconfig命令中本机的IPv4地址

```
[root@centos7 ~]# ifconfig eth0 | sed -rn '/netmask/s#.*net (.*)
net.*#\1#p'
192.168.38.128
```

### 9.将文本文件的n和n+1行合并为一行， n为奇数行

```
[root@centos7 ~]# seq 10 | sed "1~2N;s/\n/ /"
1 2
3 4
5 6
7 8
9 10
```

### 10.在每一行后增加一空行?

```
[root@centos7 ~]# sed G /etc/fstab
```

```
[root@qqq tmp]# sed -r 's/$/\n/' /etc/passwd
```

## 11.将/tmp/file.txt文件中第2到第8行之间所有大写字母替换成小写字母

```
[root@web01 sed]# sed 's#[a-z]#\u&#g' /tmp/file.txt | sed '2,8s/[A-Z]/\l&/g'
```

## 12.使用sed找出/tmp/file.txt文件中包含oldboy的行

```
[root@web01 sed]# sed -n '/oldboy/p' /tmp/file.txt
```

## 13.将/tmp/file.txt文件中以; 结尾的行，行首插入#

```
[root@web01 sed]# sed -n '/;$/p' /tmp/file.txt | sed 's@^@#@g'
#i like linux;
```

## 14.删除file.txt文件中的空行

```
[root@web01 sed]# sed -r '/^$/d' /tmp/file.txt
```

## 15.使用sed将selinux彻底关闭

```
[root@web01 sed]# sed '/^SELINUX=/c SELINUX=disabled' /etc/linux/config
disabled enforcing
```

## 参考链接

<http://www.bubuko.com/infodetail-3275744.html>

<https://blog.51cto.com/14012942/2427099>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# vim

## 1.进入vim编辑器

1.进入vim编辑器

## 2.vi / vim 的三种工作方式

命令模式：在这种模式下，可以通过输入vi的命令对文件的内容就行处理(复制，删除，移动等)，也可以通过按光标来移动光标

编辑模式：在这种模式下，可以在光标处输入内容

命令项模式：在命令模式下，用户输入冒号后，光标会跳到底行，然后输入命令

## 3.模式之间的转换

命令模式：输入a A i l o O等命令 -> 编辑模式

编辑模式：按ESC键 -> 命令模式

命令项模式：输入冒号 -> 命令项模式

## 4.vim如何插入字符

i 在光标前插入

a 当前位置编辑

A 快速到达行尾并进入编辑模式

O 在当行上面插入一个空行并进入编辑模式

o 在当行下面插入一个空行并进入编辑模式

## 5.vim如何快速定位到某行

第一行 gg

最后一行 G

移动到100行 100gg

## 6.如何删除字符

x: 删除光标处的字符 X: 删除光标前的一个字符

do: 删除光标所在行的第一个字符到当前光标的前一个字符的一串字符

D: 删除从当前光标所在字符到当前光标所在行的最后一个字符的一串字符

dd: 删除光标所在行的所有字符/剪切当前行

dw: 删除从光标处字符开始的第一个单词

u: 撤销命令,一步一步撤销

## 7.用vi命令编辑text.txt，如何跳转到末行，首行，行首、行末，如何在光标下一行插入，如何复制5行，删除10行，查找jingfeng的字符，把jingfeng替换为jfedu.NET

跳转末行 G

跳转首行 gg 跳转到第一行

跳转行首 I (大写字母)

跳转行末 快速到达行尾并进入编辑模式

下一行插入 o(小写字母o)

复制五行 5yy

删除10行 10dd

## 8.vim操作，将第9行至第15行的数据，复制到第16行 !!!!!!!

进入命令模式

: 9, 15 copy 16 或 : 9, 15 co 16

由此可有:

: 9, 15 move 16 或 :9,15 m 16 将第9行到第15行的文本内容移动到第16行的后面

## 9.有如下文本，保存在文件a.txt中，要求将所有ghi替换为xyz。请写出linux系统下所有可能的方法（不限制语言） !!!!!!!

```
vim a.txt
sed -i s#ghi#xyz#g
grep a.txt 'ghi' | xargs sed s#.*#xyz#g
tr a.txt 'ghi' 'xyz'
```

## 10.退出 vi / vim 命令

: w 保存文件 不退出vi : wq 保存文件,退出vi : q! 不保存文件,退出vi

### 参考链接

<https://blog.csdn.net/asd136912/article/details/79551540>

<https://www.jianshu.com/p/981fco3f229>



扫码关注，收获知识

程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## 备份压缩

### 1.打包跟压缩的区别:

打包是指将多个文件或者目录放在一起，形成一个总的包，这样便于保存和传输，但是大小是没有变化的，压缩是指将一个或者多个大文件或者目录通过压缩算法使文件的体积变小以达到压缩的目的，可以节省存储空间，在压缩的时候通常是先打包再压缩；

## 2.Linux主要压缩方式

linux主要有三种压缩方式：

- 1.gzip：是公认的压缩这速度最快，压缩大文件的时候与其他的压缩方式相比更加明显，历史最久，应用最广泛的压缩方式
- 2.bzip：压缩形成的文件小，但是可用性不如gzip
- 3.xz：是最新的压缩方式，可以自动提供最佳的压缩率

## 3.文件后缀名 .tar.gz 是用哪种方式压缩的

gzip

## 4.文件后缀名.tar.bz2是用哪种方式压缩的

用于bzip2压缩方式

## 5.压缩多个文件

tar zcvf FileName.tar.gz DirName1 DirName2 DirName3

## 6.zip格式

zip是Windows中最常见的压缩格式，Linux也可以正确识别，zip命令所在目录为/usr/bin/zip，所有用户可以执行，用来压缩文件或目录

## 7.zip压缩文件

```
# 压缩文件
zip -r test.zip file
```

## 8.将所有 .jpg 的文件打成一个名为 all.tar 的包

```
# tar -cf all.tar *.jpg
```

-c 是表示产生新的包，-f 指定包的文件名。

## 9.将所有 .gif 的文件增加到 all.tar 的包里面去

```
# tar -rf all.tar *.gif
```

-r 是表示增加文件的意思。

## 10.将目录里所有jpg文件打包成 tar.jpg

```
tar -cvf jpg.tar *.jpg
```



## 11. gzip 中将压缩数据输出到标准输出中，并保留源文件的选项是

-C

## 12. 如果一个系统没有备份策略，请写出一个较为全面合理的备份方案

需要备份的内容：

1. 重要的系统目录

/etc/

/home/

/root/

/var/spool/mail/

/var/spool/cron/

/var/spool/at/

2. mysql 数据库

RAM 包安装的mysql: /var/lib/mysql/

源码安装的mysql: /usr/local/mysql/data/

3. apache 服务

网站内容: /var/www/html/ /usr/local/apache2/htdocs/

配置文件: /etc/httpd/conf/httpd.conf /usr/local/apache2/conf/httpd.conf

日志文件: /var/log/httpd/ /usr/local/apache2/logs/

4. 如果有其他服务，也需要备份重要数据

备份策略：

1. 完整备份：实现命令：cp, tar, dump, xfsdump

2. 增量备份：每次备份以前一次备份作为参照

实现命令：centos6: dump 工具; centos7: xfsdump 工具

3. 差异备份：每次备份以第一次备份作为参照

实现命令：centos6: dump 工具

centos7: xfsdump 工具

备份频率：

实时备份：如MySQL主从同步

定时备份：如每天，每周备份，一般通过“脚本+定时任务”实现

备份存储位置：

基本原则：不要把鸡蛋放在同一个篮子中

本地备份

异地备份

常见服务器的备份方案：

1. 每日备份的数据（异地备份）

mysql 数据库（主从备份之外，增量备份一次）

2. 每周备份的数据（异地备份）

mysql 数据库（完整备份）

重要的系统数据

### 13.完全备份

完全备份是指把所有需要备份的数据全部备份。当然，完全备份可以备份整块硬盘、整个分区或某个具体的目录。对于 Linux 操作系统来说，完全备份指的就是将根目录下的所有文件进行备份。

完全备份的好处是，所有数据都进行了备份，系统中任何数据丢失都能恢复，且恢复效率较高。如果完全备份备份的是整块硬盘，那么甚至不需要数据恢复，只要把备份硬盘安装上，服务器就会恢复正常。

完全备份的缺点也很明显，那就是需要备份的数据量较大，备份时间较长，备份了很多无用数据，占用的空间较大，所以完全备份不可能每天执行。

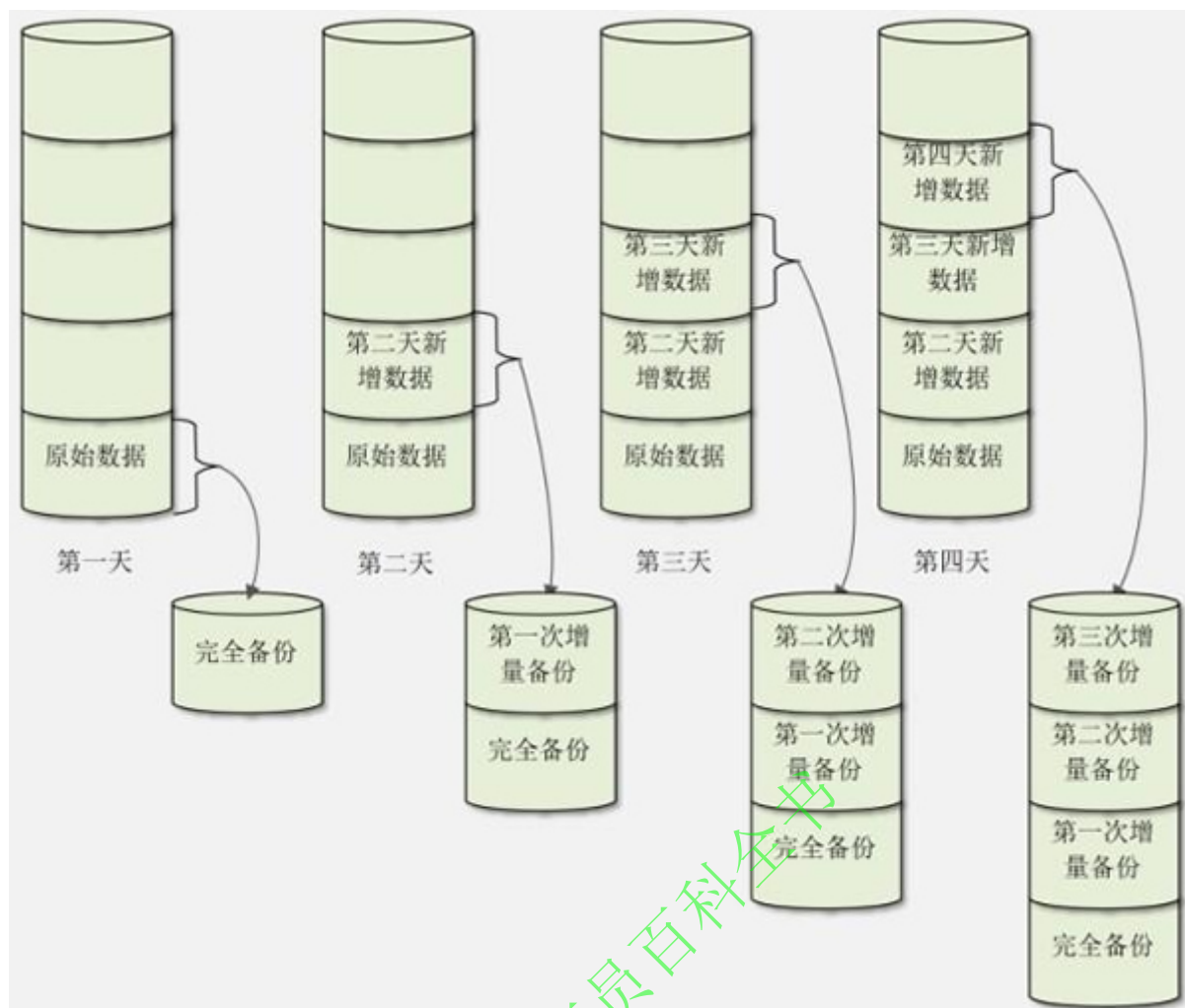
我们一般会对关键服务器进行整盘完全备份，如果出现问题，则可以很快地使用备份硬盘进行替换，从而减少损失。我们甚至会对关键服务器搭建一台一模一样的服务器，这样只要远程几个命令（或使用 Shell 脚本自动检测，自动进行服务器替换），备份服务器就会接替原本的服务器，使故障响应时间大大缩短。

### 14.累计增量备份

在一个数据量很大的业务应用中，每天对 Linux 系统进行完全备份是不现实的，这就需要用到增量备份策略。

累计增量备份是指先进行一次完全备份，服务器运行一段时间之后，比较当前系统和完全备份的备份数据之间的差异，只备份有差异的数据。服务器继续运行，再经过一段时间，进行第二次增量备份。在进行第二次增量备份时，当前系统和第一次增量备份的数据进行比较，也是只备份有差异的数据。第三次增量备份是和第二次增量备份的数据进行比较，以此类推。

因此，累计增量备份就是只备份每天增加或者变化的数据，而不备份系统中没有变动的数据。我们画一张示意图，如图 1 所示。

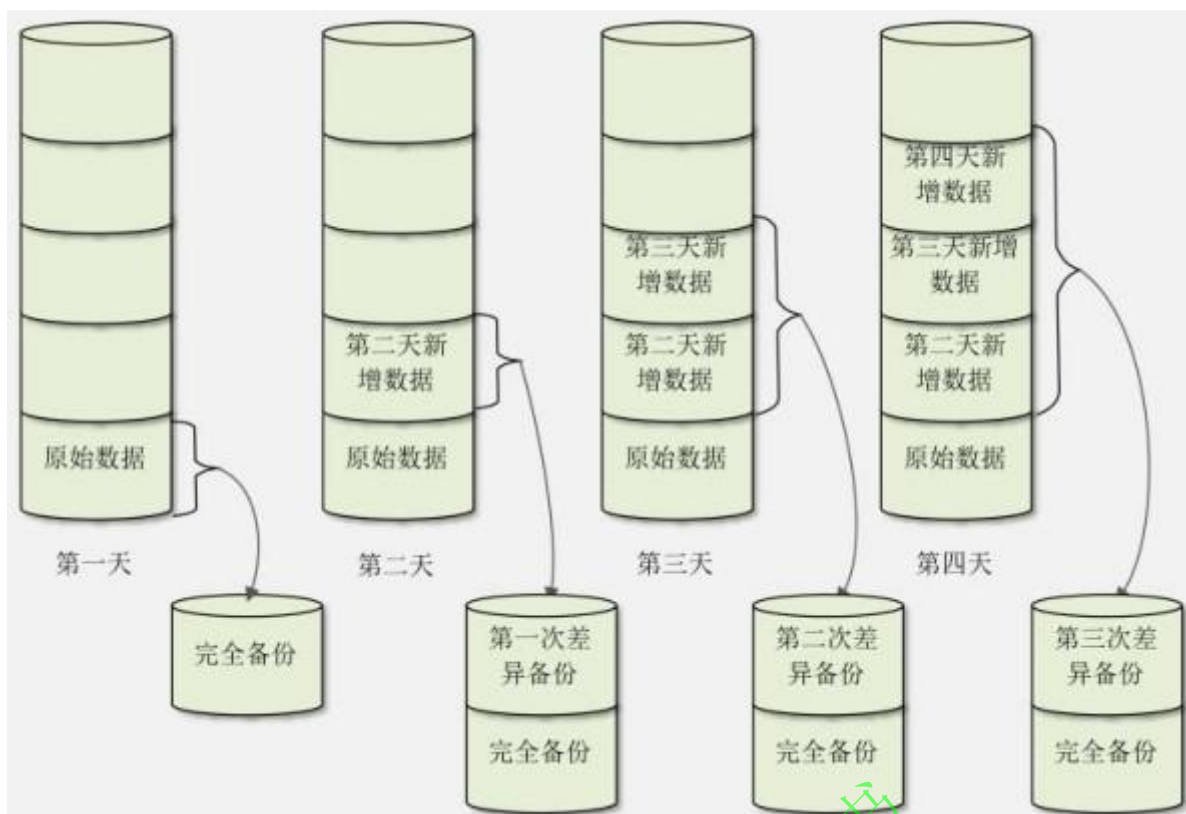


假设我们在第一天进行一次完全备份。第二天增量备份时，只会备份第二天和第一天之间的差异数据，但是第二天的总备份数据是完全备份加第一次增量备份的数据。第三天增量备份时，只会备份第三天和第二天之间的差异数据，但是第三天的总备份数据是完全备份加第一次增量备份的数据，再加第二次增量备份的数据。当然，第四天增量备份时，只会备份第四天和第三天的差异数据，但是第四天的总备份数据是完全备份加第一次增量备份的数据，加第二次增量备份的数据，再加第三次增量备份的数据。

采用累计增量备份的好处是，每次备份需要备份的数据较少，耗时较短，占用的空间较小；坏处是数据恢复比较麻烦，如果是图 1 的例子，那么当进行数据恢复时，就要先恢复完全备份的数据，再依次恢复第一次增量备份的数据、第二次增量备份的数据和第三次增量备份的数据，最终才能恢复所有的数据。

## 15. 差异增量备份

差异增量备份（后续简称差异备份）也要先进行一次完全备份，但是和累计增量备份不同的是，每次差异备份都备份和原始的完全备份不同的数据。也就是说，差异备份每次备份的参照物都是原始的完全备份，而不是上一次的差异备份。我们也画一张示意图，如图 2 所示。



假设我们在第一天也进行一次完全备份。第二天差异备份时，会备份第二天和第一天之间的差异数据，而第二天的备份数据是完全备份加第一次差异备份的数据。第三天进行差异备份时，仍和第一天的原始数据进行对比，把第二天和第三天所有的数据都备份在第二次差异备份中，第三天的备份数据是完全备份加第二次差异备份的数据。第四天进行差异备份时，仍和第一天的原始数据进行对比，把第二天、第三天和第四天所有的不同数据都备份到第三次差异备份中，第四天的备份数据是完全备份加第三次差异备份的数据。

相比较而言，差异备份既不像完全备份一样把所有数据都进行备份，也不像增量备份在进行数据恢复时那么麻烦，只要先恢复完全备份的数据，再恢复差异备份的数据即可。不过，随着时间的增加，和完全备份相比，变动的数据越来越多，那么差异备份也可能会变得数据量庞大、备份速度缓慢、占用空间较大。

一个比较的备份策略是，对于数据量不大，并且每天数据量增加不多的系统，优先选择完全备份；对于数据量巨大，每天新增数据也很多的系统，视情况选择差异备份或者增量备份。

## 16.如果网站服务器每天产生的日志数量较大，请问如何备份？

- 系统日志管理工具：logrotate（日志切割，日志轮替）
- apache服务配置文件自带日志切割功能，但是需要通过脚本进行轮替

### 参考链接

[https://blog.csdn.net/weixin\\_44901564/article/details/99682926](https://blog.csdn.net/weixin_44901564/article/details/99682926)

<https://blog.csdn.net/u012060395/article/details/104541101>

<https://blog.csdn.net/rubulai/article/details/90814639>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# 变量与环境变量

- [1.变量](#)
- [2.环境变量](#)
- [3.Bash 解释器](#)
- [4.为什么不能将当前目录 \(.\) 添加到PATH中呢?](#)
- [5.修改环境变量PATH的三种方法](#)
- [参考链接](#)

## 1.变量

变量是计算机系统用于保存可变值的数据类型。在Linux系统中，变量名称一般都是大写的，这是一种约定俗成的规范。我们可以直接通过变量名称来提取到对应的变量值。

## 2.环境变量

Linux系统中的环境变量是用来定义系统运行环境的一些参数，比如每个用户不同的家目录、邮件存放位置等。

## 3.Bash 解释器

判断用户输入的是内部命令还是外部命令。内部命令是解释器内部的指令，会被直接执行；而用户在绝大部分时间输入的是外部命令，这些命令交由步骤4 继续处理。可以使用type 命令名称来判断用户输入的命令是内部命令还是外部命令。

## 4.为什么不能将当前目录 (.) 添加到PATH中呢?

原因是，尽管可以将当前目录 (.) 添加到PATH 变量中，从而在某些情况下可以让用户免去输入命令所在路径的麻烦。但是，如果黑客在比较常用的公共目录/tmp 中存放了一个与ls 或cd 命令同名的木马文件，而用户又恰巧在公共目录中执行了这些命令，那么就极有可能中招了。所以，作为一名态度谨慎、有经验的运维人员，在接手了一台Linux 系统后一定会在执行命令前先检查PATH 变量中是否有可疑的目录。

## 5.修改环境变量PATH的三种方法

<1>.临时环境变量，重启后消失

直接用export命令：

```
export PATH=$PATH:/opt/au1200_rm/build_tools/bin
```

<2>.修改/etc/profile文件：对所有用户都有效。

```
#vi /etc/profile
```

在最后添加：export PATH="\$PATH:/opt/au1200\_rm/build\_tools/bin"

<3>.修改 ~/.bashrc 文件，仅对当前用户有效。


```
#vi ~/.bashrc
```

在最后添加：export PATH="\$PATH:/opt/au1200\_rm/build\_tools/bin"

方法<2><3>可以永久改变环境变量，一般需要重新注销系统才能生效。也可以运行#source /etc/profile命令，使修改立即生效。该方法的原理是再执行一次/etc/profile shell脚本。

## 参考链接

<https://www.kanzhun.com/mianshiti/542958.html>



扫码关注，收获知识

### 程序员百科全书

关注公众号 回复 "技术" 加入技术交流群  
关注公众号 回复 "linux" 获取Linux面试题

## 查找和文件搜索

### 1. 列出当前目录和子目录下的所有文件

这个命令会列出当前目录以及子目录下的所有文件。

```
$  
`find` `` `` /abc ``.txt `` `` /subdir `` `` /subdir/how ``.php `` `` /cool ``.p  
hp
```

该命令与以下命令效果相同

```
$ ``find` `` `` $ ``find` ``. -print
```

### 2. 查找特殊的目录或路径

下面的命令会查找当前目录下 test 文件夹中的文件，默认列出所有文件。

```
$ ``find`  
`` `` /test `` `` /test `` `` /test/abc ``.txt `` `` /test/subdir `` `` /test/subdi  
r/how ``.php `` `` /test/cool ``.php
```



下面的命令用于查找指定名称的文件。

```
$ ``find` ``.``/test` ``-name ``"abc.txt"``.``/test/abc``.txt
```

也可以使用通配符

```
$ ``find` ``.``/test` ``-name  
``"*.*.php"``.``/test/subdir/how``.php``.``/test/cool``.php
```

请注意，所有的文件夹都会被递归地查找。所以，这是用于查找指定扩展名文件的一种非常强大的方式。

如果我们尝试搜索 / 文件夹，也就是根目录，就会搜索整个文件系统，包括挂载的设备以及网络存储设备。所以请小心使用。当然，你随时可以通过按 **Ctrl + C** 来终止命令。

注意：当指定文件夹的时候（例如示例中的 `"/test"` 文件夹），忽略末尾的斜杠是没有问题的。但是，如果文件夹是一个指向其它位置的链接（**symlink**）时，你必须在末尾写上斜杠才能使 **find** 命令正常工作（`find ./test/`）。

忽略大小写

在查找文件名时，忽略大小写往往非常有用。要忽略大小写，只需要使用 **iname** 选项，而不是 **name** 选项。

```
$ ``find` ``.``/test` ``-iname  
``"*.*.Php"``.``/test/subdir/how``.php``.``/test/cool``.php
```

总是用双引号或单引号来包围匹配模式（文件名参数），这非常有用。不这样做的话有时也能正常工作，有时也可能产生奇怪的结果。

### 3.查找/etc目录中文件名为passwd的文件

```
[root@localhost ~]# find /etc/ -name passwd  
/etc/passwd  
/etc/pam.d/passwd  
##查找/etc目录中文件名以.conf文件结尾的文件  
[root@localhost mnt]# find /etc/ -name *.conf
```

### 4.按文件所在的深度（层次）查找

```
##-maxdepth表示最大深度，即最多层次  
[root@localhost ~]# find /etc/ -maxdepth 1 -name passwd  
/etc/passwd  
[root@localhost ~]# find /etc/ -maxdepth 2 -name passwd  
/etc/passwd  
/etc/pam.d/passwd  
##-mindepth表示最小深度，即最少层次  
[root@localhost ~]# find /etc/ -mindepth 2 -name passwd  
/etc/pam.d/passwd
```

```
[root@localhost ~]# find /etc/ -mindepth 1 -name passwd
/etc/passwd
/etc/pam.d/passwd
##查找/etc目录下最少层次为1最多层次为2的以.conf结尾的文件
[root@localhost ~]# find /etc/ -mindepth 1 -maxdepth 2 -name *.conf
```

## 5.反向查找

除了查找满足条件的文件之外，我们还可以查找不满足条件的所有文件。当我们知道要在查找中排除哪些文件时，这个选项就能发挥作用了。

```
$ ``find` ``.``/test` ``-not -name
``"*.*.php"``.``/test``.``/test/abc``.txt``.``/test/subdir
```

在上面的示例中我们找到了所有扩展名不是 **php** 的文件和文件夹。我们也可以使用感叹号 **!** 来代替 **-not**。

```
find` ``.``/test` ``! -name ``"*.*.php"
```

## 6. 结合多个查找条件

我们可以同时使用多个查找条件来指定文件名并排除某些文件。

```
$ ``find` ``.``/test` ``-name ``'abc*``' ``! -name
``"*.*.php"``.``/test/abc``.txt``.``/test/abc
```

上面的命令查找所有以 **abc** 开头并且不含 **.php** 扩展名的文件。这个示例展现了 **find** 命令自带的查找表达式是多么的强大。

OR 操作符

当我们使用多个查找条件时，**find** 命令会将它们通过 **AND** 操作符结合起来，也就是说，只有满足所有条件的文件才会被列出。不过，如果我们需要进行基于 **OR** 运算的查找时，可以加上 **-o** 开关。

```
$ ``find` ``-name ``'*.*.php'`` -o -name
``'*.*.txt'``.``/abc``.txt``.``/subdir/how``.php``.``/abc``.php``.``/cool`
`.php
```

上面的命令查找所有以 **.php** 结尾或者以 **.txt** 结尾的文件。

## 7.只查找文件或目录

有时我们只想通过某个名字查找对应的文件或对应的目录，我们可以很容易实现这个要求。

```
$ ``find` ``.``/test` ``-name abc*``.``/test/abc``.txt``.``/test/abc
```

只查找文件



```
$ ``find` ``.``/test` ``-``type` `f -name ``"abc*""``.``/test/abc``.txt
```

只查找目录

```
$ ``find` ``.``/test` ``-``type` `d -name ``"abc*""``.``/test/abc
```

## 8.同时在多个目录下查找

如果你想要在两个不同的目录内进行查找，命令非常简单。

```
$ ``find` ``.``/test` ``.``/dir2` ``-``type` `f -name  
``"abc*""``.``/test/abc``.txt``.``/dir2/abcdefg``.txt
```

检查一下，它确实列出了来自给定的两个目录的文件。

## 9.查找隐藏文件

在Linux系统中，隐藏文件的名字以英文的句号开头，即.。所以要列出隐藏文件，只需加上简单的文件名过滤条件就行了。

```
$ find ~ -type f -name ".*"
```

## 10.查找指定权限的文件

通过指定 `perm` 选项，我们可以查找具有特定权限的文件。下面的示例中查找了所有具有 `0664` 权限的文件。

```
$ ``find` ``. -``type` `f -perm  
0664``.``/abc``.txt``.``/subdir/how``.php``.``/abc``.php``.``/cool``.php
```

我们可以用这个命令来查找带有错误权限的文件，这些文件可能会产生安全问题。

可以结合 反向查找 来进行权限检查。

```
$ ``find` ``. -``type` `f ! -perm  
0777``.``/abc``.txt``.``/subdir/how``.php``.``/abc``.php``.``/cool``.php
```

## 11.locate

`locate`命令用于查找文件，它比`find`命令的搜索速度快，它需要一个数据库，这个数据库由每天的例行工作（`crontab`）程序来建立。当我们建立好这个数据库后，就可以方便地来搜寻所需文件了。

即先运行：`updatedb`（无论在那个目录中均可，可以放在`crontab`中）后在 `/var/lib/slocate/` 下生成 `slocate.db` 数据库即可快速查找。在命令提示符下直接执行 `#updatedb` 命令即可：

例如：查找相关字issue

```
$ locate issue
```

```
/etc/issue
```

/etc/issue.net  
/usr/man/man5/issue.5  
/usr/man/man5/issue.net.5

## 参考链接

<https://www.jb51.net/article/108198.htm>

<https://blog.csdn.net/lilygg/article/details/84076757>

<https://blog.csdn.net/nyist327/article/details/42557439>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## 磁盘管理

### 1.将系统内所有的文件系统列出来

```
[root@www ~]# df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/hdc2        9920624    3823112   5585444   41% /
/dev/hdc3        4956316    141376   4559108    4% /home
/dev/hdc1        101086      11126    84741   12% /boot
tmpfs            371332         0    371332    0% /dev/shm
```

### 2.将容量结果以易读的容量格式显示出来

```
[root@www ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/hdc2       9.5G  3.7G  5.4G   41% /
/dev/hdc3       4.8G  139M  4.4G    4% /home
/dev/hdc1       99M   11M   83M   12% /boot
tmpfs           363M     0  363M    0% /dev/shm
```

### 3.将系统内的所有特殊文件格式及名称都列出来

```
[root@www ~]# df -aT
```

Filesystem	Type	1K-blocks	Used	Available	Use%	Mounted on
/dev/hdc2	ext3	9920624	3823112	5585444	41%	/
proc	proc	0	0	0	-	/proc
sysfs	sysfs	0	0	0	-	/sys
devpts	devpts	0	0	0	-	/dev/pts
/dev/hdc3	ext3	4956316	141376	4559108	4%	/home
/dev/hdc1	ext3	101086	11126	84741	12%	/boot
tmpfs	tmpfs	371332	0	371332	0%	/dev/shm
none	binfmt_misc	0	0	0	-	
/proc/sys/fs/binfmt_misc						
sunrpc	rpc_pipefs	0	0	0	-	
/var/lib/nfs/rpc_pipefs						

#### 4.将 /etc 底下的可用的磁盘容量以易读的容量格式显示

```
[root@www ~]# df -h /etc
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hdc2	9.5G	3.7G	5.4G	41%	/

#### 5.只列出当前目录下的所有文件夹容量（包括隐藏文件夹）

```
[root@www ~]# du
```

8	./test4	<==每个目录都会列出来
8	./test2	
....中间省略....		
12	./gconfd	<==包括隐藏文件的目录
220	.	<==这个目录(.)所占用的总量

#### 6.将文件的容量也列出来

```
[root@www ~]# du -a
```

12	./install.log.syslog	<==有文件的列表了
8	./.bash_logout	
8	./test4	
8	./test2	
....中间省略....		
12	./gconfd	
220	.	

#### 7.检查根目录底下每个目录所占用的容量

```
[root@www ~]# du -sm /*
7          /bin
6          /boot
.....中间省略.....
0          /proc
.....中间省略.....
1          /tmp
3859       /usr      <==系统初期最大就是他了啦!
77         /var
```

## 8.列出所有分区信息

```
[root@AY120919111755c246621 tmp]# fdisk -l

Disk /dev/xvda: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00000000

   Device Boot      Start         End      Blocks   Id  System
/dev/xvda1    *           1         2550      2048000    83  Linux
/dev/xvda2             2550         2610         490496    82  Linux swap /
Solaris

Disk /dev/xvdb: 21.5 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x56f40944

   Device Boot      Start         End      Blocks   Id  System
/dev/xvdb2             1         2610      20964793+    83  Linux
```

## 9.找出你系统中的根目录所在磁盘，并查阅该硬盘内的相关信息

```
[root@www ~]# df /
Filesystem            1K-blocks      Used Available Use% Mounted on
/dev/hdc2              9920624    3823168   5585388   41% /
```

<==注意：重点在找出磁盘文件名而已

```
[root@www ~]# fdisk /dev/hdc <==仔细看，不要加上数字喔！
The number of cylinders for this disk is set to 5005.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): <==等待你的输入！
```

## 10.将分区 /dev/hdc6（可指定你自己的分区） 格式化为 ext3 文件系统

```
[root@www ~]# mkfs -t ext3 /dev/hdc6
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
251392 inodes, 502023 blocks
25101 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=515899392
16 block groups
32768 blocks per group, 32768 fragments per group
15712 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 34 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
# 这样就创建起来我们所需要的 Ext3 文件系统了！简单明了！
```

<==这里指的是分割槽的名称 (label)

<==block 的大小配置为 4K

<==由此配置决定的inode/block数量

<==有日志记录

## 11.强制检测 /dev/hdc6 分区

```
[root@www ~]# fsck -C -f -t ext3 /dev/hdc6
fsck 1.39 (29-May-2006)
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
vbird_logical: 11/251968 files (9.1% non-contiguous), 36926/1004046
blocks
```

## 12.用默认的方式，将刚刚创建的 /dev/hdc6 挂载到 /mnt/hdc6 上面

```
[root@www ~]# mkdir /mnt/hdc6
[root@www ~]# mount /dev/hdc6 /mnt/hdc6
[root@www ~]# df
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
.....中间省略.....					
/dev/hdc6	1976312	42072	1833836	3%	/mnt/hdc6

## 13.卸载/dev/hdc6

```
[root@www ~]# umount /dev/hdc6
```

## 参考资料

<https://www.runoob.com/linux/linux-file-system.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# 计划任务

## 1.什么是crond

crond就是计划任务，类似于我们平时生活中的闹钟。定点执行。

## 2.为什么要使用crond

crond主要是做一些周期性的任务，比如：凌晨3点定时备份数据。比如：11点开启网站抢购接口，12点关闭网站抢购接口。

## 3.计划任务有哪几种？

系统级别的定时任务：临时文件清理、系统信息采集、日志文件切割

用户级别的定时任务：定时向互联网同步时间、定时备份系统配置文件、定时备份数据库的数据

## 4.crontab配置文件记录了时间周期的含义

```
[root@lqz ~]# vim /etc/crontab
SHELL=/bin/bash                #执行命令的解释器
PATH=/sbin:/bin:/usr/sbin:/usr/bin #环境变量
MAILTO=root                    #邮件发给谁
# Example of job definition:
# .----- minute (0 - 59) #分钟
# | .----- hour (0 - 23) #小时
# | | .----- day of month (1 - 31) #日期
# | | | .----- month (1 - 12) OR jan,feb,mar,apr #月份
# | | | | .---- day of week (0 - 6) (Sunday=0 or 7) OR
sun,mon,tue,wed,thu,fri,sat #星期
# | | | | |
#
# command to be executed

# 表示任意的(分、时、日、月、周)时间都执行
# - 表示一个时间范围段，如5-7点
# , 表示分隔时段，如6,0,4表示周六、日、四
# /1 表示每隔n单位时间，如*/10 每10分钟
```

## 5.了解crontab的时间编写规范

```
00 02    1s      #每天的凌晨2点整执行
00 02 1    1s      #每月的1日的凌晨2点整执行
00 02 14 2    1s    #每年的2月14日凌晨2点执行
00 02    7 1s     #每周天的凌晨2点整执行
00 02    6 5 1s    #每年的6月周五凌晨2点执行
00 02 14    7 1s    #每月14日或每周日的凌晨2点都执行
00 02 14 2 7 1s    #每年的2月14日或每年2月的周天的凌晨2点执行
/10 02    1s      #每天凌晨2点，每隔10分钟执行一次
        1s        #每分钟都执行
00 00 14 2    1s    #每年2月14日的凌晨执行命令
/5         1s      #每隔5分钟执行一次
00 02    1,5,8 1s   #每年的1月5月8日凌晨2点执行
00 02 1-8    1s     #每月1号到8号凌晨2点执行
0 21        1s      #每天晚上21:00执行
45 4 1,10,22 1s    #每月1、10、22日的4:45执行
```

```
45 4 1-10 1 #每月1到10日的4:45执行
3,15 8-11 /2 1s #每隔两天的上午8点到11点的第3和第15分钟执行
0 23-7/1 1s #晚上11点到早上7点之间, 每隔一小时执行
15 21 1-5 1s #周一到周五每天晚上21:15执行
```

## 6.使用crontab编写cron定时任务

参数 含义

- e 编辑定时任务
- l 查看定时任务
- r 删除定时任务
- u 指定其他用户

## 7.使用root用户每5分钟执行一次时间同步

```
#1.如何同步时间
[root@lqz ~]# ntpdate time.windows.com &>/dev/null
#2.配置定时任务
[root@lqz ~]# crontab -e -u root
[root@lqz ~]# crontab -l -u root
/5 ntpdate time.windows.com &>/dev/null
```

## 8.每天的下午3,5点, 每隔半小时执行一次sync命令

```
[root@lqz ~]# crontab -l
/30 15,17 sync &>/dev/null
```

## 9.cron如何备份

通过查找/var/log/cron中执行的记录, 去推算任务执行的时间  
定时的备份/var/spool/cron/{username}

## 10.cron如何拒绝某个用户使用

```
#1.使用root将需要拒绝的用户加入/etc/cron.deny
[root@lqz ~]# echo "lqz" >> /etc/cron.deny

#2.登陆该普通用户, 测试是否能编写定时任务
[oldboy@lqz ~]$ crontab -e
You (lqz) are not allowed to use this program (crontab)
See crontab(1) for more information
```

## 11.cron调试

调整任务每分钟执行的频率, 以便做后续的调试。

如果使用cron运行脚本, 请将脚本执行的结果写入指定日志文件, 观察日志内容是否正常。

命令使用绝对路径, 防止无法找到命令导致定时任务执行产生故障。

通过查看/var/log/cron日志, 以便检查我们执行的结果, 方便进行调试。



## 12.crontd编写思路

1.手动执行命令，然后保留执行成功的结果。

2.编写脚本

脚本需要统一路径/scripts

脚本内容复制执行成功的命令(减少每个环节出错几率)

脚本内容尽可能的优化,使用一些变量或使用简单的判断语句

脚本执行的输出信息可以重定向至其他位置保留或写入/dev/null

3.执行脚本

使用bash命令执行,防止脚本没有增加执行权限(/usr/bin/bash)

执行脚本成功后,复制该执行的命令,以便写入cron

4.编写计划任务

加上必要的注释信息,人、时间、任务

设定计划任务执行的周期

粘贴执行脚本的命令(不要手敲)

5.调试计划任务

增加任务频率测试

检查环境变量问题

检查crond服务日志

### 参考资料

<http://blog.iis7.com/article/4692.html>



扫码关注，收获知识

程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## 软件包管理

### 1.rpm

Linux rpm 命令用于管理套件。

rpm（英文全拼：redhat package manager）原本是 Red Hat Linux 发行版专门用来管理 Linux 各项套件的程序，由于它遵循 GPL 规则且功能强大方便，因而广受欢迎。逐渐受到其他发行版的采用。RPM 套件管理方式的出现，让 Linux 易于安装，升级，间接提升了 Linux 的适用度。

## 2.如何安装软件包

```
[root@localhost ~]# rpm-ivh PACKAGE_NAME-VERSION.rpm
```

## 3.测试安装软件包，不做真实的安装

```
[root@localhost ~]# rpm-ivh--relocate /=usr/local/PACKAGE_NAME  
PACKAGE_NAME-VERSION.rpm
```

## 4.如何使用 rpm 初始化数据库?

通过rpm 命令查询一个rpm 包是否安装了，也是要通过rpm 数据库来完成的；所以我们要经常用下面的两个命令来初始化rpm 数据库；

```
<code>[root@feiyu ~]# rpm --initdb[root@feiyu ~]# rpm --rebuilddb      注：  
这个要花好长时间；</code>
```

注：这两个参数是极为有用，有时rpm 系统出了问题，不能安装和查询，大多是这里出了问题。

## 5.升级软件包

```
[root@localhost ~]# rpm-Uvh PACKAGE_NAME-VERSION.rpm
```

## 6.怎样查询一个已安装软件包的信息?

语法格式： rpm -qi 软件名

```
[root@feiyu ~]# rpm -qi nmon
```

## 7.yum

yum ( Yellow dog Updater, Modified) 是一个在 Fedora 和 RedHat 以及 SUSE 中的 Shell 前端软件包管理器。

基于 RPM 包管理，能够从指定的服务器自动下载 RPM 包并且安装，可以自动处理依赖性关系，并且一次安装所有依赖的软件包，无须繁琐地一次次下载、安装。

yum 提供了查找、安装、删除某一个、一组甚至全部软件包的命令，而且命令简洁而又好记。

## 8.列出所有可更新的软件清单命令(yum)

```
yum check-update
```

## 9.更新所有软件命令

```
yum update
```

## 10.删除软件包命令

```
yum remove <package_name>
```

## 参考链接

<https://www.runoob.com/linux/linux-comm-rpm.html>

<http://www.tianfeiyu.com/?p=1271>

[https://www.sohu.com/a/294989082\\_100145576](https://www.sohu.com/a/294989082_100145576)



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术" 加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## 网络通信

### 1.为“ssh”生成、管理和转换认证密钥的命令是

```
ssh-keygen (选项)
```

### 2.如何指定登录用户

```
ssh root@192.168.0.102
```

### 3.指定端口登录

```
$ ssh 192.168.0.102 -p 222
```

### 4.ssh创建秘钥

```
$ ssh-keygen -t dsa/rsa
```

## 5.删除主机秘钥

```
$ ssh-keygen -R 192.168.0.102
```

## 6.列出所有的端口

```
netstat -a
```

## 7.列出TCP协议的端口

```
netstat -at
```

## 8.UDP协议的端口

```
netstat -au
```

## 9.列出处于监听状态的socket

```
netstat -l
```

## 10.列出监听的TCP端口

```
netstat -lt
```

## 11.网络连接状态有哪些

1)、LISTEN:首先服务端需要打开一个socket进行监听,状态为LISTEN.

```
/* The socket is listening for incoming connections. 侦听来自远方TCP端口的连接请求 */
```

2)、SYN\_SENT:客户端通过应用程序调用connect进行active open.于是客户端tcp发送一个SYN以请求建立一个连接.之后状态置为SYN\_SENT.

```
/*The socket is actively attempting to establish a connection. 在发送连接请求后等待匹配的连接请求 */
```

3)、SYN\_RECV:服务端应发出ACK确认客户端的 SYN,同时自己向客户端发送一个SYN. 之后状态置为SYN\_RECV

```
/* A connection request has been received from the network. 在收到和发送一个连接请求后等待对连接请求的确认 */
```

4)、ESTABLISHED: 代表一个打开的连接,双方可以进行或已经在数据交互了。

```
/* The socket has an established connection. 代表一个打开的连接,数据可以传送给用户 */
```

5)、FIN\_WAIT1:主动关闭(active close)端应用程序调用close,于是其TCP发出FIN请求主动关闭连接,之后进入FIN\_WAIT1状态。

```
/* The socket is closed, and the connection is shutting down. 等待远程TCP的连接中断请求, 或先前的连接中断请求的确认 */
```

6)、CLOSE\_WAIT:被动关闭(passive close)端TCP接到FIN后, 就发出ACK以回应FIN请求(它的接收也作为文件结束符传递给上层应用程序), 并进入CLOSE\_WAIT.

```
/* The remote end has shut down, waiting for the socket to close. 等待从本地用户发来的连接中断请求 */
```

7)、FIN\_WAIT2:主动关闭端接到ACK后, 就进入了 FIN-WAIT-2 .

```
/* Connection is closed, and the socket is waiting for a shutdown from the remote end. 从远程TCP等待连接中断请求 */
```

8)、LAST\_ACK:被动关闭端一段时间后, 接收到文件结束符的应用程序将调用CLOSE关闭连接。这导致它的TCP也发送一个 FIN, 等待对方的ACK. 就进入了LAST-ACK .

```
/* The remote end has shut down, and the socket is closed. Waiting for acknowledgement. 等待原来发向远程TCP的连接中断请求的确认 */
```

9)、TIME\_WAIT:在主动关闭端接收到FIN后, TCP 就发送ACK包, 并进入TIME-WAIT状态。

```
/* The socket is waiting after close to handle packets still in the network. 等待足够的时间以确保远程TCP接收到连接中断请求的确认 */
```

10)、CLOSING: 比较少见.

```
/* Both sockets are shut down but we still don't have all our data sent. 等待远程TCP对连接中断的确认 */
```

11)、CLOSED: 被动关闭端在接受到ACK包后, 就进入了closed的状态。连接结束.

```
/* The socket is not being used. 没有任何连接状态 */
```

12)、UNKNOWN: 未知的Socket状态。

```
/* The state of the socket is unknown. */
```

## 12.ping

在网络维护过程中, Ping命令是一个经常使用的DOS命令, 它是用来检查网络是否畅通或者网络连接速度的命令。可用于诊断连接性、可访问性和名称解析, 可以探测对方计算机的活动情况, 还可以通过数据返回时间简单推测对方的操作系统。

## 13.如何使用ping

举例

```
ping www.baidu.com
```

## 14.ifconfig

在Linux系统中主要用于显示配置网络设备, 通常需要以root身份登录或使用sudo以便在Linux机器上使用ifconfig工具。依赖于ifconfig命令中使用一些选项属性, ifconfig工具不仅可以被用来简单地获取网络接口配置信息, 还可以修改这些配置。

## 15.配置网卡的IP地址,在eth0上配置上192.168.0.1 的IP地址及24位掩码

```
ifconfig eth0 192.168.0.1 netmask 255.255.255.0
```

## 16.若想再在eth0上在配置一个192.168.1.1/24 的IP地址怎么办?

```
ifconfig eth0:0 192.168.1.1 netmask 255.255.255.0
```

## 17.配置网卡的硬件地址

```
ifconfig eth0 hw ether xx: xx: xx: xx: xx: xx
```

## 18.将网卡禁用

```
ifconfig eth0 down
```

## 19.将网卡启用

```
ifconfig eth0 up
```

## 20.在指定网络接口上发出DHCP请求

```
[root@linuxcool ~]# dhclient eth0
```

## 参考链接

<https://www.jianshu.com/p/8e88d4b11dec>

<https://jingyan.baidu.com/article/2f9b480db035e141cb6cc23e.html>

<https://blog.csdn.net/freeking101/article/details/53520974>

<https://www.jb51.net/network/546149.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# 文件传输

## 1.scp

【优点】简单方便，安全可靠；支持限速参数

【缺点】不支持排除目录

【用法】

scp就是secure copy，是用来进行远程文件拷贝的。数据传输使用ssh，并且和ssh使用相同的认证方式，提供相同的安全保证。

命令格式：

```
scp [参数] <源地址（用户名@IP地址或主机名）>:<文件路径> <目的地址（用户名 @IP 地址或主机名）>:<文件路径>
```

举例：

```
scp /home/work/source.txt work@192.168.0.10:/home/work/ #把本地的source.txt文件拷贝到192.168.0.10机器上的/home/work目录下
```

```
scp work@192.168.0.10:/home/work/source.txt /home/work/ #把192.168.0.10机器上的source.txt文件拷贝到本地的/home/work目录下
```

```
scp work@192.168.0.10:/home/work/source.txt work@192.168.0.11:/home/work/ #把192.168.0.10机器上的source.txt文件拷贝到192.168.0.11机器的/home/work目录下
```

```
scp -r /home/work/sourcedir work@192.168.0.10:/home/work/ #拷贝文件夹，加-r参数
```

```
scp -r /home/work/sourcedir work@www.myhost.com:/home/work/ #使用主机名
```

```
scp -r -v /home/work/sourcedir work@www.myhost.com:/home/work/ #显示详情，加-v参数
```

## 2.rcp

【概述】

目标主机需要事先打开rcp功能，并设置好rcp的权限：把源主机加入到可信任主机列表中，否则无法在源主机上使用rcp远程复制文件到目标主机。

## 3.wget

【优点】简单方便，支持排除目录，支持限速参数

【缺点】只能从远程机器将文件或文件夹下载到本地，并且远程机器需要支持ftp服务（例如启动proftpd）；参数较多，使用上比scp复杂

【用法】

wget是一个从网络上自动下载文件的自由工具，支持通过HTTP、HTTPS、FTP三个最常见的TCP/IP协议下载，并可以使用HTTP代理。

命令格式：

```
wget [参数] ftp://<目标机器ip或主机名>/<文件的绝对路径> #proftpd格式
```

举例：

```
wget ftp://192.168.0.10//home/work/source.txt #从192.168.0.10上拷贝文件夹
source.txt

wget ftp://www.myhost.com//home/work/source.txt #使用主机名
wget -nH -P /home/work/ ftp://www.myhost.com//home/work/source.txt #指定
本地保存路径, 使用参数"-P 路径"或者"--directory-prefix=路径"; -nH, --no-host-
directories 不创建主机目录

wget -r -l 0 -nH -P /home/work/
ftp://www.myhost.com//home/work/sourcedir #递归下载sourcedir目录, 使用参数-
r; 参数-l, --level=NUMBER 最大递归深度 (inf 或 0 代表无穷).
wget --cut-dirs=3 -r -l 0 -nH -P /home/work/
ftp://www.myhost.com//home/work/sourcedir #-参数-cut-dirs=NUMBER 忽略
NUMBER层远程目录, 本例中将myhost上的sourcedir目录保存到本地的work目录下。
wget --limit-rate=200k --cut-dirs=3 -r -l 0 -nH -P /home/work/
ftp://www.myhost.com//home/work/sourcedir #-参数--limit-rate=RATE 限定下
载输率
wget --limit-rate=200k --cut-dirs=3 -r -l 0 -nH -P /home/work/ -X
/home/work/sourcedir/notincludedir
ftp://www.myhost.com//home/work/sourcedir #排除路径使用-x参数
wget -q --limit-rate=200k --cut-dirs=3 -r -l 0 -nH -P /home/work/ -X
/home/work/sourcedir/notincludedir
ftp://www.myhost.com//home/work/sourcedir #参数-q表示安静模式, 无输出; 默认
是-v, 冗余模式
```

## 4.rsync

【优点】功能强大, 操作类似scp, 支持排除目录, 支持限速参数; 还支持本地复制。

【缺点】暂无

【用法】

rsync是类unix系统下的数据镜像备份工具, 从软件的命名上就可以看出来——remote sync。它的操作方式和scp和相似, 但是比scp强大很多。使用双冒号分割主机名和文件路径时, 是使用rsync服务器, 这里不做介绍。

命令格式:

rsync [参数] 源地址 (用户名@IP地址或主机名):<文件路径> <目的地址 (用户名 @IP 地址或主机名) >:<文件路径>

举例:



```
rsync /home/work/source.txt work@192.168.0.10:/home/work/ #把本地的
source.txt文件拷贝到192.168.0.10机器上的/home/work目录下

rsync work@192.168.0.10:/home/work/source.txt /home/work/ #把
192.168.0.10机器上的source.txt文件拷贝到本地的/home/work目录下

rsync work@192.168.0.10:/home/work/source.txt
work@192.168.0.11:/home/work/ #把192.168.0.10机器上的source.txt文件拷贝到
192.168.0.11机器的/home/work目录下

rsync -r /home/work/sourcedir work@192.168.0.10:/home/work/ #拷贝文件夹，
加-r参数

rsync -r /home/work/sourcedir work@www.myhost.com:/home/work/ #使用主机名
rsync -r -v /home/work/sourcedir work@www.myhost.com:/home/work/ #显示详
情，加-v参数

rsync -r -v --exclude sourcedir/notinclude /home/work/sourcedir
work@www.myhost.com:/home/work/ #排除子目录，注意：--exclude后面的路径不能为
绝对路径，必须为相对路径才可以，否则匹配不上，就不会被排除掉。
```

## 参考资料

[https://blog.csdn.net/qw\\_xingzhe/article/details/80167888](https://blog.csdn.net/qw_xingzhe/article/details/80167888)



扫码关注，收获知识

程序员百科全书

关注公众号 回复 "技术" 加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## 文件管理

### 1. 显示当前工作目录的绝对路径

pwd

### 2. 查看当前目录的所有内容信息

ls

### 3.显示当前目录所有的文件和目录，包括隐藏的

```
ls -a
```

### 4.以行的形式显示当前目录所有的文件和目录，包括隐藏的

```
ls -al
```

### 5.显示/etc目录下，所有.conf结尾，且以m,n,r,p开头的文件或目录

```
[19:23:32 root@centos82[ data]#ls /etc/{m,n,r,p}*.conf
ls: cannot access '/etc/p*.conf': No such file or directory
/etc/man_db.conf    /etc/mke2fs.conf    /etc/nsswitch.conf
/etc/resolv.conf
```

### 6.显示 /var 目录下所有以l开头，以一个小写字母结尾，且中间出现至少一位数的文件或目录

```
[19:28:22 root@centos82[ data]#ls /var/l*[0-9]*[[:lower:]]
ls: cannot access '/var/l*[0-9]*[[:lower:]]': No such file or directory
```

### 7.显示/etc目录下以任意一位数字开头，且以非数字结尾的文件或目录

```
[19:29:30 root@centos82[ data]#ls /etc/[0-9]*[^0-9]
ls: cannot access '/etc/[0-9]*[^0-9]': No such file or directory
```

### 8.显示/etc/目录下以非字母开头，后面跟了一个字母及其它任意长度任意字符的文件或目录

```
[19:31:31 root@centos82[ data]#ls /etc/[^a-Z][a-Z]*
ls: cannot access '/etc/[^a-Z][a-Z]*': No such file or directory
```

### 9、显示/etc/目录下所有文件名以rc开头，并且后面是0到6中的数字，其它为任意字符的文件或目录

```
[19:33:32 root@centos82[ data]#ls /etc/rc[0-6]*
/etc/rc0.d:

/etc/rc1.d:

/etc/rc2.d:

/etc/rc3.d:

/etc/rc4.d:

/etc/rc5.d:

/etc/rc6.d:
```

## 10. 显示 /etc 目录下，所有以 .d 结尾的文件或目录

```
[19:34:35 root@centos82[ data]#ls /etc/*.d
/etc/bash_completion.d:
authselect-completion.sh  iprconfig  redefine_filedir

/etc/binfmt.d:

/etc/chkconfig.d:

/etc/cron.d:
.....
```

## 11.回到自己的家目录

cd ~或者cd

## 12.回到当前目录的上一级目录

cd ..

## 13.回到当前目录的上级的上一级目录

cd ../../

## 14.在home下创建dog目录，只能创建一级目录

mkdir /home/dog

## 15.创建空文件 test.txt

touch test.txt

## 16.同时创建子目录dir1, dir2, dir3

```
[root@linuxcool ~]# mkdir dir1 dir2 dir3
```

## 17.递归创建目录

```
[root@linuxcool ~]# mkdir -p linuxcool/dir
```

## 18.复制1234.txt文件到新文件2345.txt

copy 1234.txt 2345.txt

## 19.如果复制后的新文件名已存在会怎样呢?

文件会被覆盖并不会提醒直接就执行成功了。

## 20.为避免不知道有没有同名文件被覆盖,需要添加哪个选项

cp -i

## 21.将file\_1.txt文件从当前目录移动到其它目录,以/home/pungki/为例

```
$ mv file_1.txt /home/pungki/office
```

## 22.将file1.txt重命名为file2.txt

```
$ mv file_1.txt file_2.txt
```

## 23.除了mv,你还知道其他的修改文件名方式吗

rename

### 参考链接

<https://www.jianshu.com/p/86ad6eeb1a51>

<https://www.runoob.com/>

<https://jingyan.baidu.com/article/14bd256e1ca9defb6c261259.html>

<https://www.jianshu.com/p/3172011c1aeb>



扫码关注,收获知识

### 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## 系统信息

### 1.查看Linux内核版本命令

```
1. uname -a (Linux查看版本当前操作系统内核信息)
```

```
2. cat /proc/version (Linux查看当前操作系统版本信息)
```

## 2.查看Linux系统版本的命令

1、lsb\_release -a, 即可列出所有版本信息:  
这个命令适用于所有的Linux发行版, 包括RedHat、SUSE、Debian...等发行版。

2、cat /etc/redhat-release, 这种方法只适合Redhat系的Linux:

```
[root@S-CentOS home]# cat /etc/redhat-release
```

```
CentOS release 6.5 (Final)
```

3、cat /etc/issue, 此命令也适用于所有的Linux发行版。

## 3.Linux查看版本当前操作系统发行版信息

1.cat /etc/issue 或cat /etc/redhat-release

## 4.查看服务器名称

```
hostname
```

## 5.查看网络信息

```
ifconfig
```

## 6.查看CPU信息

```
cat /proc/cpuinfo
```

## 7.列出所有PCI设备

```
lspci -tv
```

## 8.查看环境变量

```
env
```

## 9.查看活动用户

```
w
```

## 10.列出所有系统服务

```
chkconfig --list
```

## 11.内存管理的必要性

出现在早期的计算机系统当中，程序是直接运行在物理内存中，每一个程序都能直接访问物理地址。如果这个系统只运行一个程序的话，并且这个程序所需的内存不要超过该机器的物理内存就不会出现问题。但是如今的系统都是支持多任务和多进程的，那么这时我们就要考虑如何将系统内有限的物理内存及时有效的分配给多个程序了，这就是内存管理的基本概念。顾名思义就是记录哪些内存是正在使用的，哪些内存是空闲的；在进程需要时为其分配内存，在进程使用完后释放内存。

## 12.虚拟地址

即使是现代操作系统中，内存依然是计算机中很宝贵的资源，看看你电脑几个T固态硬盘，再看看内存大小就知道了。为了充分利用和管理系统内存资源，Linux采用虚拟内存管理技术，利用虚拟内存技术让每个进程都有4GB互不干涉的虚拟地址空间。

进程初始化分配和操作的都是基于这个「虚拟地址」，只有当进程需要实际访问内存资源的时候才会建立虚拟地址和物理地址的映射，调入物理内存页。

打个不是很恰当的比方。这个原理其实和现在的某某网盘一样，假如你的网盘空间是1TB，真以为就一口气给了你这么大空间吗？那还是太年轻，都是在你往里面放东西的时候才给你分配空间，你放多少就分多少实际空间给你，但你和你朋友看起来就像大家都拥有1TB空间一样。

## 13.虚拟地址的好处

避免用户直接访问物理内存地址，防止一些破坏性操作，保护操作系统每个进程都被分配了4GB的虚拟内存，用户程序可使用比实际物理内存更大的地址空间4GB的进程虚拟地址空间被分成两部分：「用户空间」和「内核空间」。



## 14.物理地址

当需进程要实际访问内存的时候，会由内核的「请求分页机制」产生「缺页异常」调入物理内存页。

把虚拟地址转换成内存的物理地址，这中间涉及利用MMU 内存管理单元（Memory Management Unit）对虚拟地址分段和分页（段页式）地址转换，关于分段和分页的具体流程，这里不再赘述，可以参考任何一本计算机组成原理教材描述。



Linux 内核会将物理内存分为3个管理区，分别是：

### ZONE\_DMA

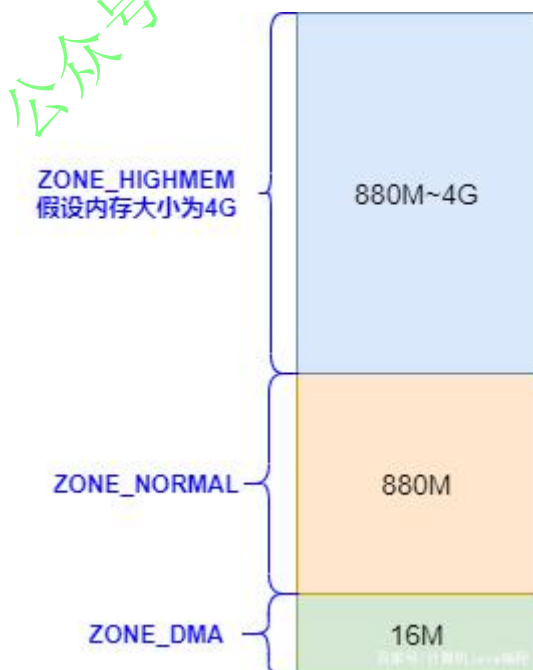
DMA内存区域。包含0MB~16MB之间的内存页框，可以由老式基于ISA的设备通过DMA使用，直接映射到内核的地址空间。

### ZONE\_NORMAL

普通内存区域。包含16MB~896MB之间的内存页框，常规页框，直接映射到内核的地址空间。

### ZONE\_HIGHMEM

高端内存区域。包含896MB以上的内存页框，不进行直接映射，可以通过永久映射和临时映射进行这部分内存页框的访问。



### 用户空间

用户进程能访问的是「用户空间」，每个进程都有自己独立的用户空间，虚拟地址范围从从0x00000000 至 0xBFFFFFFF 总容量3G。

用户进程通常只能访问用户空间的虚拟地址，只有在执行内陷操作或系统调用时才能访问内核空间。

## 15.页的概念

linux 内核中把物理页作为内存分配的最小单位，32位CPU 页的大小通常为4K，64位的CPU 通常支持8K的也。内存管理单元MMU 同样以页为大小分配内存。

## 16.swap对换空间

32位Linux系统的每个进程可以有4 GB的虚拟 内存空间。而且系统中还要同时存在多个进程，但是，事实上大多数计算机都没有这么多物理内存空间，当系统中的物理内存紧缺时，就需要利用对换空间把一部分未来可能不用的页面从物理内存中移到对换设备或对换文件中。

Linux采用两种方式保存换出的页面：

一种是利用整个块设备，如硬盘的一个分区，即对换设备，

另一种是利用文件系统中固定长度的文件，即对换文件。它们统称为对换空间。

这两种方式的相同之处是它们的内部格式一致。但是在执行效率方面，对换设备要好一些。这是因为对换设备上同一页面的数据块是连续存放的，故而可以顺序存取，而在对换文件中。同一页面的数据块实际的物理位置可能是不连续的，需要通过对换文件的 inode检索，这就降低了存取效率。

每个对换文件或对换设备由 `struct swap_info` 结构来描述。有关对换设备的函数主要是 `get_swap_page(...)`。当内存中的页面需要被换出时，调用 `get_swap_page`函数 申请得到一个对换空间中的物理页面。如果成功，就返回一个非零代码，否则返回0。

## 17.缺页中断定义

当一个进程退出时，他并不是完全消失，而是等到它的符进程发出wait系统调用才会完全消失，除非父进程发出wait系统调用终止进程，否则该进程一直处于僵死状态，等待父进程终止它。

实际上，僵尸进程已经死了，它要向父进程返回一个退出状态，报告死讯。而父进程一直不进行wait系统调用的话，子进程就变成了僵尸进程。要消灭僵尸进程的话，先找到僵尸进程的父进程，杀死父进程，然后由init（进程号为1）消灭僵尸进程。init是所有进程的父进程。

## 18.缺页中断的次数

中断次数=进程的物理块数+页面置换次数。

缺页中断率=（缺页中断次数 / 总访问页数）



## 19.页面置换算法

如果父进程比子进程先死的话，子进程就变成孤儿进程了，这时候孤儿进程就被init（进程号为1）进程领养。并由init进程对他们完成状态收集工作。

## 20.如何杀死某个进程

当发生缺页中断时，如果操作系统内存中没有空闲页面，则操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间。而用来选择淘汰哪一页的规则叫做页面置换算法。

几种缺页中断算法（FIFO，LRU与LFU）的实现过程：

在分页虚拟存储管理的系统中，有一用户进程，它依次要访问的页面序列是序列7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1。假定分配给该进程的页数（物理块）为3且进程初始时未装载页面（意思就是进程只能使用三块内存）。计算缺页次数和缺页率？那么采用FIFO、LRU、OPT调度算法产生的缺页中断数各为多少？缺页中断率各为多少？

### 1)、先进先出（FIFO）

优先淘汰最早进入内存的页面，亦即在内存中驻留时间最久的页面。该算法实现简单，只需把调入内存的页面根据先后次序链接成队列，设置一个指针总指向最早的页面。但该算法与进程实际运行时的规律不适应，因为在进程中，有的页面经常被访问。

利用FIFO置换算法时的置换图：

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块1	7	7	7	2		2	2	4	4	4	0		0	0			7	7	7	
物理块2		0	0	0		3	3	3	2	2	2		1	1			1	0	0	
物理块3			1	1		1	0	0	0	3	3		3	2			2	2	1	
缺页否	√	√	√	√		√	√	√	√	√	√		√	√			√	√		
√																				

进程访问页面2时，把最早进入内存的页面7换出。然后访问页面3时，再把2, 0, 1中最先进入内存的页换出。由图可以看出，利用FIFO算法时进行了12次页面置换。发生缺页中断的次数为15。

### 2)、最近最久未使用(LRU)置换算法

选择最近最长时间未访问过的页面予以淘汰，它认为过去一段时间内未访问过的页面，在最近的将来可能也不会被访问。该算法为每个页面设置一个访问字段，来记录页面自上次被访问以来所经历的时间，淘汰页面时选择现有页面中值最大的予以淘汰。

对上面的实例采用LRU算法进行页面置换，如图所示。进程第一次对页面2访问时，将最近最久未被访问的页面7置换出去。然后访问页面3时，将最近最久未使用的页面1换出。

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块1	7	7	7	2		2		4	4	4	0		1	1	1					
物理块2		0	0	0		0		0	0	3	3		3	0	0					
物理块3			1	1		3		3	2	2	2		2	2	7					
缺页否	√	√	√	√		√		√	√	√	√		√	√			√			

### 3). 最佳置换算法(OPT)

最佳(Optimal, OPT)置换算法所选择的被淘汰页面将是以后永不使用的,或者是在最长时间内不再被访问的页面,这样可以保证获得最低的缺页率。但由于人们目前无法预知进程在内存下的若干页面中哪个是未来最长时间内不再被访问的,因而该算法无法实现。

对上面的实例采用OPT算法进行页面置换,如图所示:

访问页面	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
物理块1	7	7	7	2		2		2		2								7		
物理块2		0	0	0		0		4		0		0						0		
物理块3			1	1		3		3		3		1						1		
缺页否	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√

进程运行时,先将7,0,1三个页面依次装入内存。进程要访问页面2时,产生缺页中断,根据最佳置换算法,选择第18次访问才需调入的页面7予以淘汰。然后,访问页面0时,因为已在内存中所以不必产生缺页中断。访问页面3时又会根据最佳置换算法将页面1淘汰.....依此类推,如图所示。从图中可以看出采用最佳置换算法时的情况。

可以看到,发生缺页中断的次数为9,页面置换的次数为6。

## 21. 显示内存使用情况

**Linux free**命令(free)用来显示Linux系统中空闲的、已用的物理内存及交换分区(swap)内存,及被内核使用的buffer。

其数据取自 /proc/meminfo文件。在这个文件中,数据的存储就是以KB为单位,所以free的默认值也是KB。

## 22. 解释free -m的输出

-m: 以MB为单位显示内存使用情况;

```
free -m
              total        used         free       shared    buffers
cached
Mem:          2016         1973           42            0          163
1497
-/+ buffers/cache:          312        1703
Swap:          4094            0        4094
```

第一部分Mem行解释:

total: 内存总数;  
used: 已经使用的内存数;  
free: 空闲的内存数;  
shared: 当前已经废弃不用;  
buffers Buffer: 缓存内存数;  
cached Page: 缓存内存数。

关系:  $total = used + free$

## 第二部分(-/+ buffers/cache)解释:

```
(-buffers/cache) used内存数: 第一部分Mem行中的 used - buffers - cached  
(+buffers/cache) free内存数: 第一部分Mem行中的 free + buffers + cached
```

可见-buffers/cache反映的是被程序实实在在吃掉的内存,而+buffers/cache反映的是可以挪用的内存总数。

第三部分是指交换分区。

## 23.进程

执行中的程序称作进程。当程序可以执行文件存放在存储中,并且运行的时候,每个进程会被动态得分配系统资源、内存、安全属性和与之相关的状态。可以有多个进程关联到同一个程序,并同时执行不会互相干扰。操作系统会有效地管理和追踪所有运行着的进程。

## 24.列出所有运行中/激活进程

```
ps -a
```

## 25.列出需要进程

```
ps -ef |grep
```

## 26.以树状结构显示进程关系

```
pstree
```

## 27.杀进程的命令

- 1、杀死 - 通过ID杀死一个进程
- 2、killall - 按名称杀死一个进程

## 28.进程的状态

进程描述符(task\_struct)中的state变量描述进程的当前状态。常见的状态如下:

- 1、TASK\_RUNNING(运行状态): 表示进程是可执行的,它正在执行,或者在运行队列中等待执行。
- 2、TASK\_INTERRUPTIBLE(可中断,也就是睡眠状态): 进程正在睡眠(也就是进程被阻塞)等待某些条件满足后才运行。
- 3、TASK\_UNINTERRUPTIBLE(不可中断,磁盘休眠状态):除了不会因为接收到信号而被唤醒运行之外,他与可中断状态相同。在这个状态的进程通常会等待IO的结束。可以通过发送SIGSTOP信号给进程来停止进程,这个被暂停的进程可以通过发送SIGCONT信号让进程继续执行。

例:

停止进程: kill -SIGSTOP

继续运行: kill -SIGCONT

4、TASK\_ZOMBIE(僵死状态): 该进程已经结束了, 但是父进程还没有使用wait()系统调用。因为父进程不能读取到子进程退出的返回代码, 所以就会产生僵死进程。为了父进程能够获知它的消息, 子进程的进程描述符仍然被保留, 一旦父进程调用wait(), 进程描述符就会被释放。

5、TASK\_STOPPED(停止状态): 进程停止执行, 进程不能投入运行。通常这种状态发生在接收到SIGSTOP、SIGTSTP、SIGTTIN、SIGOUT等信号的时候。此外, 在调试期间接受到的任何信号, 都会使进程进入这种状态。

## 29.僵尸进程

当一个进程退出时, 他并不是完全消失, 而是等到它的父进程发出wait系统调用才会完全消失, 除非父进程发出wait系统调用终止进程, 否则该进程一直处于僵死状态, 等待父进程终止它。

实际上, 僵尸进程已经死了, 它要向父进程返回一个退出状态, 报告死讯。而父进程一直不进行wait系统调用的话, 子进程就变成了僵尸进程。要消灭僵尸进程的话, 先找到僵尸进程的父进程, 杀死父进程, 然后由init (进程号为1) 消灭僵尸进程。init是所有进程的父进程。

## 30.查看环境变量

### 1) 检查当前僵尸进程信息

```
ps -ef | grep defunct | grep -v grep | wc -l

175

top | head -2

top - 15:05:54 up 97 days, 23:49, 4 users, load average: 0.66, 0.45, 0.39
Tasks: 829 total, 1 running, 479 sleeping, 174 stopped, 175 zombie

ps -ef | grep defunct | grep -v grep
```

### 2) 获得杀僵尸进程语句

```
ps -ef | grep defunct | grep -v grep | awk '{print "kill -9 " $2,$3}'
```

执行上面获得的语句即可, 使用信号量9, 僵尸进程数会大大减少.

### 3) 过一会儿检查当前僵尸进程信息

```
ps -ef | grep defunct | grep -v grep | wc -l

125
top | head -2

top - 15:29:26 up 98 days, 12 min, 7 users, load average: 0.27, 0.54,
0.56
Tasks: 632 total, 1 running, 381 sleeping, 125 stopped, 125 zombie
```

发现僵尸进程数减少了一些,但还有不少啊.

#### 4) 再次获得杀僵尸进程语句

```
ps -ef | grep defunct | grep -v grep | awk '{print "kill -18 " $3}'
```

执行上面获得的语句即可,这次使用信号量18杀其父进程,僵尸进程应该会全部消失.

#### 5) 过一会儿再检查当前僵尸进程信息

```
ps -ef | grep defunct | grep -v grep | wc -l

0
top | head -2

top - 15:39:46 up 98 days, 23 min, 7 users, load average: 5.46, 2.20,
1.12
Tasks: 134 total, 1 running, 133 sleeping, 0 stopped, 0 zombie
```

#### 6) 清除ZOMBIE(僵尸)进程原理

##### kill -18 PPID

PPID是其父进程,这个信号是告诉父进程,该子进程已经死亡了,请收回分配给他的资源.

如果还不行则先看其父进程又无其他子进程,如果有,可能需要先kill其他子进程,也就是兄弟进程.

方法是:

##### kill -15 PID1 PID2

PID1,PID2是僵尸进程的父进程的其它子进程.

然后再kill父进程:

##### kill -15 PPID

# 31.孤儿进程

如果父进程比子进程先死的话，子进程就变成孤儿进程了，这时候孤儿进程就被init（进程号为1）进程领养。并由init进程对他们完成状态收集工作。

# 32.如何杀死某个进程

在做项目的时候经常会出现程序死机、锁死、无响应等情况，这时候就需要找到程序相应的进程将其杀掉即可。步骤如下：

## 1. 定位进程

top 命令：可以实时动态地查看系统的整体运行情况，是一个综合了多方信息监测系统性能和运行信息的实用工具。通过 top 命令所提供的交互式界面，用热键可以管理。

输入 top 后可以看到如下的界面，实时显示进程情况。

```
ubuntu@VM-0-15-ubuntu: ~
top - 13:35:29 up 27 days, 3:10, 3 users, load average: 0.00, 0.00, 0.00
Tasks: 135 total, 1 running, 131 sleeping, 3 stopped, 0 zombie
%Cpu(s): 1.3 us, 0.7 sy, 0.0 ni, 97.7 id, 0.3 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 1917272 total, 448640 free, 277044 used, 1191588 buff/cache
KiB Swap: 0 total, 0 free, 0 used, 1399832 avail Mem

  PID USER      PR  NI   VIRT   RES   SHR  S  %CPU  %MEM     TIME+ COMMAND
 29537 root        20   0 439732 14344 2972  S   0.7   0.7 42:32.39 barad_agent
    3 root        20   0     0     0     0  S   0.3   0.0  1:08.26 ksoftirqd/0
 5420 root        20   0 339992 13528 10892  S   0.3   0.7 26:47.72 smbd
29880 ubuntu      20   0 43640   3576 2992  R   0.3   0.2  0:00.04 top
    1 root        20   0 37948   5480 3452  S   0.0   0.3  1:11.48 systemd
    2 root        20   0     0     0     0  S   0.0   0.0  0:00.09 kthreadd
    5 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 kworker/0:0H
    7 root        20   0     0     0     0  S   0.0   0.0  4:12.84 rcu_sched
    8 root        20   0     0     0     0  S   0.0   0.0  0:00.00 rcu_bh
    9 root        rt   0     0     0     0  S   0.0   0.0  0:00.00 migration/0
   10 root        rt   0     0     0     0  S   0.0   0.0  0:10.76 watchdog/0
   11 root        20   0     0     0     0  S   0.0   0.0  0:00.00 kdevtmpfs
   12 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 netns
   13 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 perf
   14 root        20   0     0     0     0  S   0.0   0.0  0:00.71 khungtaskd
   15 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 writeback
   16 root        25   5     0     0     0  S   0.0   0.0  0:00.00 ksmd
   17 root        39  19     0     0     0  S   0.0   0.0  0:05.24 khugepaged
   18 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 crypto
   19 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 kintegrityd
   20 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 bioset
   21 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 kblockd
   22 root        0 -20     0     0     0  S   0.0   0.0  0:00.00 ata_sff
```

ps 命令：process status 的简称，用于报告当前系统的进程状态。此命令长配合 grep 过滤输出结果，常用的结构：

```
ps -aux | grep ***
```

aux 选项如下所示：

a - 显示所有用户的进程

u - 显示进程的用户和拥有者

x - 显示不依附于终端的进程

举个例子，查看 python 相关的进程

```
ubuntu@VM-0-15-ubuntu:~$ ps aux|grep python
ubuntu 14992 0.0 0.4 37416 9520 pts/0 T 11:04 0:00 python
ubuntu 30579 0.0 0.0 16064 972 pts/2 S+ 13:41 0:00 grep --color=auto python
```

第一行数据解释从左到右）：

“ubuntu” 是用户；“14992” 是 PID；“0.0” 是 %CPU - 占用 CPU 的百分比；“0.4” 是 %MEM - 占用内存的百分比；

PID 就是我们要的

## 2. 杀死进程

我们可以通过 进程的名字和进程的 ID（PID）来结束进程。

结束命令：

kill：通过进程 ID 来结束进程

killall：通过进程名字结束进程

最长使用的结束进程的信号是：

Signal Name	Single Value	Effect
SIGHUP	1	挂起
SIGINT	2	键盘的中断信号
SIGKILL	9	发出杀死信号
SIGTERM	15	发出终止信号
SIGSTOP	17, 19, 23	停止进程

我们可以通过 Single Value 的值来代替信号的名字。所以我们现在来杀死 python 进程：

```
kill SIGNAL PID
```

SIGNAL 是要发送的信号，PID 是进程号。

```
kill -9 14992
```

上面的命令就是杀死 python 进程的。如果有多个 python 程序在运行，想要全部结束的话，可以

```
killall -9 python
```

## 参考链接

<https://www.jianshu.com/p/2b2a7955a8ef>

[https://blog.csdn.net/qq\\_31278903/article/details/83146031](https://blog.csdn.net/qq_31278903/article/details/83146031)

<https://blog.csdn.net/u013616945/article/details/77435607>

<https://blog.51cto.com/ixdba/541355>

<https://blog.csdn.net/hguisu/article/details/6152921>

<https://blog.csdn.net/boenxia/article/details/89642006>

<https://www.linuxprobe.com/12linux-process-commands.html>

[https://blog.csdn.net/qq\\_31278903/article/details/83146031](https://blog.csdn.net/qq_31278903/article/details/83146031)

<https://www.linuxprobe.com/linux-kill-manager.html>

[https://blog.csdn.net/LF\\_2016/article/details/54577707](https://blog.csdn.net/LF_2016/article/details/54577707)

<https://blog.51cto.com/petervip/956479>

[https://blog.csdn.net/qq\\_29303759/article/details/81942356](https://blog.csdn.net/qq_29303759/article/details/81942356)

## 用户管理

### 1.添加新用户

useradd 选项 用户名

参数说明：

选项：

-c comment 指定一段注释性描述。

-d 目录 指定用户主目录，如果此目录不存在，则同时使用-m选项，可以创建主目录。

-g 用户组 指定用户所属的用户组。

-G 用户组，用户组 指定用户所属的附加组。

-s Shell文件 指定用户的登录Shell。

-u 用户号 指定用户的用户号，如果同时有-o选项，则可以重复使用其他用户的标识号。

用户名：

指定新账号的登录名。



## 2.删除帐号

userdel 选项 用户名

常用的选项是 **-r**，它的作用是把用户的主目录一起删除。

## 3.修改帐号

usermod 选项 用户名

常用的选项包括**-c**、**-d**、**-m**、**-g**、**-G**、**-s**、**-u**以及**-o**等，这些选项的意义与**useradd**命令中的选项一样，可以为用户指定新的资源值。

另外，有些系统可以使用选项：**-l** 新用户名

这个选项指定一个新的账号，即将原来的用户名改为新的用户名。

## 4.用户密码管理

passwd 选项 用户名

可使用的选项：

**-l** 锁定口令，即禁用账号。

**-u** 口令解锁。

**-d** 使账号无口令。

**-f** 强迫用户下次登录时修改口令。

如果默认用户名，则修改当前用户的口令。

普通用户修改自己的口令时，**passwd**命令会先询问原口令，验证后再要求用户输入两遍新口令，如果两次输入的口令一致，则将这个口令指定给用户；而超级用户为用户指定口令时，就不需要知道原口令。

## 5.添加新用户组

groupadd 选项 用户组

可以使用的选项有：

**-g** GID 指定新用户组的组标识号（GID）。

**-o** 一般与**-g**选项同时使用，表示新用户组的GID可以与系统已有用户组的GID相同。

## 6.删除用户组

groupdel 用户组

## 7.修改用户组属性

groupmod 选项 用户组

常用的选项有：

**-g** GID 为用户组指定新的组标识号。

**-o** 与**-g**选项同时使用，用户组的新GID可以与系统已有用户组的GID相同。

**-n**新用户组 将用户组的名字改为新名字

## 8.切换到其他用户组

如果一个用户同时属于多个用户组，那么用户可以在用户组之间切换，以便具有其他用户组的权限。

用户可以在登录后，使用命令`newgrp`切换到其他用户组，这个命令的参数就是目的用户组。例如：

```
$ newgrp root
```

这条命令将当前用户切换到`root`用户组，前提条件是`root`用户组确实是该用户的主组或附加组。类似于用户账号的管理，用户组的管理也可以通过集成的系统管理工具来完成。

## 9.批量添加用户

添加和删除用户对每位Linux系统管理员都是轻而易举的事，比较棘手的是如果要添加几十个、上百个甚至上千个用户时，我们不太可能还使用`useradd`一个一个地添加，必然要找一种简便的创建大量用户的方法。Linux系统提供了创建大量用户的工具，可以让您立即创建大量用户，方法如下：

(1) 先编辑一个文本用户文件。

每一列按照`/etc/passwd`密码文件的格式书写，要注意每个用户的用户名、UID、宿主目录都不可以相同，其中密码栏可以留做空白或输入`x`号。一个范例文件`user.txt`内容如下：

```
user001::600:100:user:/home/user001:/bin/bash
user002::601:100:user:/home/user002:/bin/bash
user003::602:100:user:/home/user003:/bin/bash
user004::603:100:user:/home/user004:/bin/bash
user005::604:100:user:/home/user005:/bin/bash
user006::605:100:user:/home/user006:/bin/bash
```

(2) 以`root`身份执行命令 `/usr/sbin/newusers`，从刚创建的用户文件`user.txt`中导入数据，创建用户：

```
# newusers < user.txt
```

然后可以执行命令 `vipw` 或 `vi /etc/passwd` 检查 `/etc/passwd` 文件是否已经出现这些用户的数据，并且用户的宿主目录是否已经创建。

(3) 执行命令`/usr/sbin/pwunconv`。

将 `/etc/shadow` 产生的 `shadow` 密码解码，然后回写到 `/etc/passwd` 中，并将`/etc/shadow`的`shadow`密码栏删掉。这是为了方便下一步的密码转换工作，即先取消 `shadow password` 功能。

```
# pwunconv
```

(4) 编辑每个用户的密码对照文件。

格式为：

```
用户名:密码
```

实例文件 passwd.txt 内容如下:

```
user001:123456
user002:123456
user003:123456
user004:123456
user005:123456
user006:123456
```

(5) 以 root 身份执行命令 /usr/sbin/chpasswd。

创建用户密码，chpasswd 会将经过 /usr/bin/passwd 命令编码过的密码写入 /etc/passwd 的密码栏。

```
# chpasswd < passwd.txt
```

(6) 确定密码经编码写入/etc/passwd的密码栏后。

执行命令 /usr/sbin/pwconv 将密码编码为 shadow password，并将结果写入 /etc/shadow。

```
# pwconv
```

## 10.普通用户切换到root

```
$su -
```

注意：单纯地使用su而不是su-，来切换成为root身份，这种方式下很多原本的变量不会被改变，如PATH。因此：切换成root最好用su -

## 11.root用户切换到普通用户

```
$su - wangsy
或者
$su -l wangsy
```

## 参考资料

<https://www.runoob.com/linux/linux-user-manage.html>

<https://blog.csdn.net/diyingqian/article/details/80012802>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# Shell echo 命令

## 1.语法

Shell echo 命令 Shell 的 echo 指令与 PHP 的 echo 指令类似，都是用于字符串的输出。

Shell 的 echo 指令与 PHP 的 echo 指令类似，都是用于字符串的输出。命令格式：

```
echo string
```

您可以使用 echo 实现更复杂的输出格式控制。

## 2.显示普通字符串:

```
echo "It is a test"
```

这里的双引号完全可以省略，以下命令与上面实例效果一致：

```
echo It is a test
```

## 3.显示转义字符

```
echo "\"It is a test\""
```

结果将是：

```
"It is a test"
```

同样，双引号也可以省略

## 4.显示变量

read 命令从标准输入中读取一行，并把输入行的每个字段的值指定给 shell 变量

```
#!/bin/sh
read name
echo "$name It is a test"
```

以上代码保存为 `test.sh`, `name` 接收标准输入的变量, 结果将是:

```
[root@www ~]# sh test.sh
OK                               #标准输入
OK It is a test                 #输出
```

## 5 显示换行

```
echo -e "OK! \n" # -e 开启转义
echo "It is a test"
```

输出结果:

```
OK!

It is a test
```

## 6. 显示不换行

```
#!/bin/sh
echo -e "OK! \c" # -e 开启转义 \c 不换行
echo "It is a test"
```

输出结果:

```
OK! It is a test
```

## 7. 显示结果定向至文件

```
echo "It is a test" > myfile
```

## 8. 原样输出字符串, 不进行转义或取变量 (用单引号)

```
echo '$name\''
```

输出结果:

```
$name\''
```

## 9. 显示命令执行结果

```
echo `date`
```

注意: 这里使用的是反引号 ```, 而不是单引号 `'`。

结果将显示当前日期

## 参考链接

<https://www.runoob.com/linux/linux-shell-printf.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术" 加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# Shell printf命令

## 1.语法

printf 命令模仿 C 程序库 (library) 里的 printf() 程序。

printf 由 POSIX 标准所定义，因此使用 printf 的脚本比使用 echo 移植性好。

printf 使用引用文本或空格分隔的参数，外面可以在 printf 中使用格式化字符串，还可以制定字符串的宽度、左右对齐方式等。默认 printf 不会像 echo 自动添加换行符，我们可以手动添加 \n。

printf 命令的语法：

```
printf format-string [arguments...]
```

参数说明：

- **format-string:** 为格式控制字符串
- **arguments:** 为参数列表。

## 实例

```
$ **echo** "Hello, Shell"
Hello, Shell
$ **printf** "Hello, Shell**\n**"
Hello, Shell
$
```

接下来,我来用一个脚本来体现 printf 的强大功能:

```

*#!/bin/bash*
*# author:菜鸟教程*
*# url:www.runoob.com*

**printf** "%-10s %-8s %-4s**\n**" 姓名 性别 体重kg
**printf** "%-10s %-8s %-4.2f**\n**" 郭靖 男 66.1234
**printf** "%-10s %-8s %-4.2f**\n**" 杨过 男 48.6543
**printf** "%-10s %-8s %-4.2f**\n**" 郭芙 女 47.9876

```

执行脚本，输出结果如下所示：

姓名	性别	体重kg
郭靖	男	66.12
杨过	男	48.65
郭芙	女	47.99

**%s %c %d %f** 都是格式替代符，**%s** 输出一个字符串，**%d** 整型输出，**%c** 输出一个字符，**%f** 输出实数，以小数形式输出。

**%-10s** 指一个宽度为 10 个字符（- 表示左对齐，没有则表示右对齐），任何字符都会被显示在 10 个字符宽的字符内，如果不足则自动以空格填充，超过也会将内容全部显示出来。

**%-4.2f** 指格式化为小数，其中 **.2** 指保留 2 位小数。

```

*#!/bin/bash*
*# author:菜鸟教程*
*# url:www.runoob.com*

*# format-string为双引号*
**printf** "%d %s**\n**" 1 "abc"

*# 单引号与双引号效果一样*
**printf** '%d %s\n' 1 "abc"

*# 没有引号也可以输出*
**printf** **%**s abcdef

*# 格式只指定了一个参数，但多出的参数仍然会按照该格式输出，format-string 被重用*
**printf** **%**s abc def

**printf** "%s**\n**" abc def

**printf** "%s %s %s**\n**" a b c d e f g h i j

*# 如果没有 arguments, 那么 %s 用NULL代替，%d 用 0 代替*
**printf** "%s and %d **\n**"

```

执行脚本，输出结果如下所示：

```
1 abc
1 abc
abcdefabcdefabc
def
a b c
d e f
g h i
j
and 0
```

## 2.printf 的转义序列

序列	说明
\a	警告字符，通常为ASCII的BEL字符
\b	后退
\c	抑制（不显示）输出结果中任何结尾的换行字符（只在 <b>%b</b> 格式指示符控制下的参数字符串中有效），而且，任何留在参数里的字符、任何接下来的参数以及任何留在格式字符串中的字符，都被忽略
\f	换页（formfeed）
\n	换行
\r	回车（Carriage return）
\t	水平制表符
\v	垂直制表符
\	一个字面上的反斜杠字符
\ddd	表示1到3位数八进制值的字符。仅在格式字符串中有效
\oddd	表示1到3位的八进制值字符

### 实例



```
$ **printf** "a string, no processing:<%s>**\n**" "A**\n**B"
a string, no processing:**<**A\nB**>**

$ **printf** "a string, no processing:<%b>**\n**" "A**\n**B"
a string, no processing:**<**A
B**>**

$ **printf** "www.runoob.com \a"
www.runoob.com $          *#不换行*
```

## 参考链接

<https://www.runoob.com/linux/linux-shell-printf.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# Shell test命令

## 1.test命令

Shell中的 test 命令用于检查某个条件是否成立，它可以进行数值、字符和文件三个方面的测试。

## 2.数值测试

参数	说明
-eq	等于则为真
-ne	不等于则为真
-gt	大于则为真
-ge	大于等于则为真
-lt	小于则为真
-le	小于等于则为真

实例

```
num1=100
num2=100
if test ${num1} -eq ${num2}
then
    echo '两个数相等！'
else
    echo '两个数不相等！'
fi
```

输出结果：

```
两个数相等！
```

代码中的 `[]` 执行基本的算数运算，如：

实例

```
#!/bin/bash

a=5
b=6

result=$((a+b)) # 注意等号两边不能有空格
echo "result 为: $result"
```

结果为：

```
result 为: 11
```

3.字符串测试

参数	说明
=	等于则为真
!=	不相等则为真
-z 字符串	字符串的长度为零则为真
-n 字符串	字符串的长度不为零则为真

实例

```
num1="rulnoob"
num2="runoob"
if test $num1 = $num2
then
    echo '两个字符串相等!'
else
    echo '两个字符串不相等!'
fi
```

输出结果：

两个字符串不相等！

## 4.文件测试

参数	说明
-e 文件名	如果文件存在则为真
-r 文件名	如果文件存在且可读则为真
-w 文件名	如果文件存在且可写则为真
-x 文件名	如果文件存在且可执行则为真
-s 文件名	如果文件存在且至少有一个字符则为真
-d 文件名	如果文件存在且为目录则为真
-f 文件名	如果文件存在且为普通文件则为真
-c 文件名	如果文件存在且为字符型特殊文件则为真
-b 文件名	如果文件存在且为块特殊文件则为真

实例

```
cd /bin
if test -e ./bash
then
    echo '文件已存在!'
else
    echo '文件不存在!'
fi
```

输出结果：

```
文件已存在！
```

另外，Shell 还提供了与( **-a** )、或( **-o** )、非( **!** )三个逻辑操作符用于将测试条件连接起来，其优先级为：**!**最高，**-a**次之，**-o**最低。例如：

实例

```
cd /bin
if test -e ./notFile -o -e ./bash
then
    echo '至少有一个文件存在！'
else
    echo '两个文件都不存在'
fi
```

输出结果：

```
至少有一个文件存在！
```

## 参考链接

<https://www.runoob.com/linux/linux-shell-test.html>



扫码关注，收获知识

程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## Shell变量

### 1.定义变量

定义变量时，变量名不加美元符号（\$，PHP 语言中变量需要），如：

```
your_name="runoob.com"
```

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

- 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。
- 中间不能有空格，可以使用下划线（`_`）。

- 不能使用标点符号。
- 不能使用 **bash** 里的关键字（可用 **help** 命令查看保留关键字）。

有效的 **Shell** 变量名示例如下：

```
RUNOOB
LD_LIBRARY_PATH
_var
var2
```

无效的变量命名：

```
?var=123
user*name=runoob
```

除了显式地直接赋值，还可以用语句给变量赋值，如：

```
for file in `ls /etc`
或
for file in $(ls /etc)
```

以上语句将 **/etc** 下目录的文件名循环出来。

## 2.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：

```
your_name="qinjx"
echo $your_name
echo ${your_name}
```

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

```
for skill in Ada Coffe Action Java; do
    echo "I am good at ${skill}Script"
done
```

如果不给 **skill** 变量加花括号，写成 `echo "I am good at $skillScript"`，解释器就会把 `$skillScript` 当成一个变量（其值为空），代码执行结果就不是我们期望的样子了。

推荐给所有变量加上花括号，这是个好的编程习惯。

已定义的变量，可以被重新定义，如：

```
your_name="tom"
echo $your_name
your_name="alibaba"
echo $your_name
```

这样写是合法的，但注意，第二次赋值的时候不能写 `$your_name="alibaba"`，使用变量的时候才加美元符（\$）。

### 3.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：

```
#!/bin/bash
myUrl="https://www.google.com"
readonly myUrl
myUrl="https://www.runoob.com"
```

运行脚本，结果如下：

```
/bin/sh: NAME: This variable is read only.
```

### 4.删除变量

使用 `unset` 命令可以删除变量。语法：

```
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

```
#!/bin/sh
myUrl="https://www.runoob.com"
unset myUrl
echo $myUrl
```

以上实例执行将没有任何输出。

### 5.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。

### 6.单引号

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

单引号字符串的限制：

- 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；
- 单引号字符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。

### 7.双引号

```
your_name='runoob'
str="Hello, I know you are \"$your_name\"! \n"
echo -e $str
```

输出结果为：

```
Hello, I know you are "runoob"!
```

双引号的优点：

- 双引号里可以有变量
- 双引号里可以出现转义字符

### 8.拼接字符串

```
your_name="runoob"
# 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, runoob ! hello, runoob !
hello, runoob ! hello, ${your_name} !
```

## 9.获取字符串长度

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

## 10.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="runoob is a great site"
echo ${string:1:4} # 输出 unoo
```

注意：第一个字符的索引值为 0。

## 11.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="runoob is a great site"
echo `expr index "$string" io` # 输出 4
```

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。

## 12.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：

```
数组名=(值1 值2 ... 值n)
```

例如：

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

或者



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

还可以单独定义数组的各个分量：

```
array_name[0]=value0  
array_name[1]=value1  
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。

### 13.读取数组

读取数组元素值的一般格式是：

```
${数组名[下标]}
```

例如：

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

使用 @ 符号可以获取数组中的所有元素，例如：

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

### 14.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：

```
# 取得数组元素的个数  
length=${#array_name[@]}  
# 或者  
length=${#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：

```
#-----  
# 这是一个注释  
# author: 菜鸟教程  
# site: www.runoob.com  
# slogan: 学的不仅是技术，更是梦想！  
#-----  
##### 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。

## 15.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...  
注释内容...  
!
```

## 参考链接

<https://www.runoob.com/linux/linux-shell-variable.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## Shell 传递参数

我们可以在执行 Shell 脚本时，向脚本传递参数，脚本内获取参数的格式为：**\$n**。**n** 代表一个数字，1 为执行脚本的第一个参数，2 为执行脚本的第二个参数，以此类推.....

### 实例

以下实例我们向脚本传递三个参数，并分别输出，其中 **\$0** 为执行的文件名（包含文件路径）：

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

echo "Shell 传递参数实例！";
echo "执行的文件名: $0";
echo "第一个参数为: $1";
echo "第二个参数为: $2";
echo "第三个参数为: $3";
```

为脚本设置可执行权限，并执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh 1 2 3
Shell 传递参数实例！
执行的文件名: ./test.sh
第一个参数为: 1
第二个参数为: 2
第三个参数为: 3
```

另外，还有几个特殊字符用来处理参数：

参  
数  
处  
理

说明

参数处理	说明
\$#	传递到脚本的参数个数
\$*	以一个单字符串显示所有向脚本传递的参数。如"\$*"用「"」括起来的情况、以"\$1 \$2 ... \$n"的形式输出所有参数。
\$\$	脚本运行的当前进程ID号
\$_	后台运行的最后一个进程的ID号
\$@	与\$*相同，但是使用时加引号，并在引号中返回每个参数。如"\$@"用「"」括起来的情况、以"\$1" "\$2" ... "\$n" 的形式输出所有参数。
\$-	显示Shell使用的当前选项，与set命令功能相同。
\$?	显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误。

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

echo "Shell 传递参数实例！";
echo "第一个参数为: $1";

echo "参数个数为: $#";
echo "传递的参数作为一个字符串显示: $*";
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh 1 2 3
Shell 传递参数实例！
第一个参数为: 1
参数个数为: 3
传递的参数作为一个字符串显示: 1 2 3
```

\$\* 与 \$@ 区别：

- 相同点：都是引用所有参数。
- 不同点：只有在双引号中体现出来。假设在脚本运行时写了三个参数 1、2、3，，则 "\$\*" 等价于 "1 2 3"（传递了一个参数），而 "\$@" 等价于 "1" "2" "3"（传递了三个参数）。

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

echo "-- \$* 演示 ---"
for i in "$*"; do
    echo $i
done

echo "-- \$@ 演示 ---"
for i in "$@"; do
    echo $i
done
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh 1 2 3
-- $* 演示 ---
1 2 3
-- @$ 演示 ---
1
2
3
```

## 参考链接

<https://www.runoob.com/linux/linux-shell-passing-arguments.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# Shell 函数

## 1.语法

Shell 函数 linux shell 可以用用户定义函数，然后在 shell 脚本中可以随便调用。

linux shell 可以用用户定义函数，然后在 shell 脚本中可以随便调用。

shell 中函数的定义格式如下：

```
[ function ] funname [()]\n\n{\n\n    action;\n\n    [return int;]\n\n}
```

说明:

- 1、可以带 **function fun()** 定义,也可以直接 **fun()** 定义,不带任何参数。
- 2、参数返回,可以显示加: **return** 返回,如果不加,将以最后一条命令运行结果,作为返回值。**return** 后跟数值 **n(0-255)**

下面的例子定义了一个函数并进行调用:

```
#!/bin/bash\n# author:菜鸟教程\n# url:www.runoob.com\n\ndemoFun() {\n    echo "这是我的第一个 shell 函数!"\n}\n\necho "-----函数开始执行-----"\ndemoFun\n\necho "-----函数执行完毕-----"
```

输出结果:

```
-----函数开始执行-----\n这是我的第一个 shell 函数!\n-----函数执行完毕-----
```

下面定义一个带有 **return** 语句的函数:

```
#!/bin/bash\n# author:菜鸟教程\n# url:www.runoob.com\n\nfunWithReturn() {\n    echo "这个函数会对输入的两个数字进行相加运算..." \n    echo "输入第一个数字: "\n    read aNum\n    echo "输入第二个数字: "\n    read anotherNum\n    echo "两个数字分别为 $aNum 和 $anotherNum !"\n    return $(( $aNum+$anotherNum ))\n}
```

```
}  
funWithReturn  
echo "输入的两个数字之和为 $? !"
```

输出类似下面：

```
这个函数会对输入的两个数字进行相加运算...  
输入第一个数字：  
1  
输入第二个数字：  
2  
两个数字分别为 1 和 2 !  
输入的两个数字之和为 3 !
```

函数返回值在调用该函数后通过 `$?` 来获得。

注意：所有函数在使用前必须定义。这意味着必须将函数放在脚本开始部分，直至 `shell` 解释器首次发现它时，才可以使用。调用函数仅使用其函数名即可。

## 2.函数参数

在 `Shell` 中，调用函数时可以向其传递参数。在函数体内部，通过 `$n` 的形式来获取参数的值，例如，`$1` 表示第一个参数，`$2` 表示第二个参数...

带参数的函数示例：

```
#!/bin/bash  
# author:菜鸟教程  
# url:www.runoob.com  
  
funWithParam() {  
    echo "第一个参数为 $1 !"  
    echo "第二个参数为 $2 !"  
    echo "第十个参数为 $10 !"  
    echo "第十个参数为 ${10} !"  
    echo "第十一个参数为 ${11} !"  
    echo "参数总数有 $# 个!"  
    echo "作为一个字符串输出所有参数 $* !"  
}  
  
funWithParam 1 2 3 4 5 6 7 8 9 34 73
```

输出结果：

```
第一个参数为 1 !
第二个参数为 2 !
第十个参数为 10 !
第十个参数为 34 !
第十一个参数为 73 !
参数总数有 11 个!
作为一个字符串输出所有参数 1 2 3 4 5 6 7 8 9 34 73 !
```

注意，\$10 不能获取第十个参数，获取第十个参数需要 \${10}。当  $n \geq 10$  时，需要使用 \${n} 来获取参数。

另外，还有几个特殊字符用来处理参数：

参数处理	说明
\$#	传递到脚本或函数的参数个数
\$*	以一个单字符串显示所有向脚本传递的参数
\$\$	脚本运行的当前进程 ID 号
#!	后台运行的最后一个进程的 ID 号
\$@	与 \$* 相同，但是使用时加引号，并在引号中返回每个参数。
\$-	显示 Shell 使用的当前选项，与 set 命令功能相同。
\$?	显示最后命令的退出状态。0 表示没有错误，其他任何值表明有错误。

## 参考链接

<https://www.runoob.com/linux/linux-shell-func.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术" 加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题



# Shell 基本运算符

## 1.概念

Shell 基本运算符 Shell 和其他编程语言一样，支持多种运算符，包括：算数运算符关系运算符布尔运算符字符串运算符文件测试运算符 原生 `bash` 不支持简单的数学运算，但是可以通过其他命令来实现，例如 `awk` 和 `expr`，`expr` 最常用。

Shell 和其他编程语言一样，支持多种运算符，包括：

- 算数运算符
- 关系运算符
- 布尔运算符
- 字符串运算符
- 文件测试运算符

原生 `bash` 不支持简单的数学运算，但是可以通过其他命令来实现，例如 `awk` 和 `expr`，`expr` 最常用。

`expr` 是一款表达式计算工具，使用它能完成表达式的求值操作。

例如，两个数相加（注意使用的是反引号 ``` 而不是单引号 `'`）：

实例

```
\#!/bin/bash

val=`expr 2 + 2`
echo " 两数之和为 : $val"
```

执行脚本，输出结果如下所示：

```
两数之和为 : 4
```

两点注意：

- 表达式和运算符之间要有空格，例如 `2+2` 是不对的，必须写成 `2 + 2`，这与我们熟悉的大多数编程语言不一样。
- 完整的表达式要被 ``` 包含，注意这个字符不是常用的单引号，在 `Esc` 键下边。

## 2.算术运算符

下表列出了常用的算术运算符，假定变量 `a` 为 10，变量 `b` 为 20：

运算符	说明	举例

运算符	说明	举例
+	加法	<code>expr \$a + \$b</code> 结果为 30。
-	减法	<code>expr \$a - \$b</code> 结果为 -10。
*	乘法	<code>expr \$a \* \$b</code> 结果为 200。
/	除法	<code>expr \$b / \$a</code> 结果为 2。
%	取余	<code>expr \$b % \$a</code> 结果为 0。
=	赋值	<code>a=\$b</code> 将把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	<code>[\$a == \$b]</code> 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	<code>[\$a != \$b]</code> 返回 true。

注意：条件表达式要放在方括号之间，并且要有空格，例如：**`[$a==$b]`** 是错误的，必须写成 **`[$a == $b]`**。

## 实例

算术运算符实例如下：

```
\#!/bin/bash
\# author: 菜鸟教程
\# url:www.runoob.com

a=10
b=20

val=`expr $a + $b`
echo "a + b : $val"

val=`expr $a - $b`
echo "a - b : $val"

val=`expr $a \* $b`
echo "a * b : $val"

val=`expr $b / $a`
echo "b / a : $val"
```

```
val=`expr $b % $a`  
echo "b % a : $val"  
  
if [ $a == $b ]  
then  
    echo "a 等于 b"  
fi  
if [ $a != $b ]  
then  
    echo "a 不等于 b"  
fi
```

执行脚本，输出结果如下所示：

```
a + b : 30  
a - b : -10  
a * b : 200  
b / a : 2  
b % a : 0  
a 不等于 b
```

注意：

- 乘号 (\*) 前边必须加反斜杠 ( \ ) 才能实现乘法运算；
- if...then...fi 是条件语句，后续将会讲解。
- 在 MAC 中 shell 的 expr 语法是：\$((表达式))，此处表达式中的 "\*" 不需要转义符号 ""。

### 3.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。

下表列出了常用的关系运算符，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。

运算符	说明	举例
-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。

## 实例

关系运算符实例如下：

```
\#!/bin/bash
\# author: 菜鸟教程
\# url:www.runoob.com

a=10
b=20

if [ $a -eq $b ]
then
    echo "$a -eq $b : a 等于 b"
else
    echo "$a -eq $b: a 不等于 b"
fi
if [ $a -ne $b ]
then
    echo "$a -ne $b: a 不等于 b"
else
    echo "$a -ne $b : a 等于 b"
fi
if [ $a -gt $b ]
then
    echo "$a -gt $b: a 大于 b"
else
    echo "$a -gt $b: a 不大于 b"
fi
if [ $a -lt $b ]
then
    echo "$a -lt $b: a 小于 b"
else
    echo "$a -lt $b: a 不小于 b"
fi
if [ $a -ge $b ]
then
    echo "$a -ge $b: a 大于或等于 b"
else
```

```
    echo "$a -ge $b: a 小于 b"
fi
if [ $a -le $b ]
then
    echo "$a -le $b: a 小于或等于 b"
else
    echo "$a -le $b: a 大于 b"
fi
```

执行脚本，输出结果如下所示：

```
10 -eq 20: a 不等于 b
10 -ne 20: a 不等于 b
10 -gt 20: a 不大于 b
10 -lt 20: a 小于 b
10 -ge 20: a 小于 b
10 -le 20: a 小于或等于 b
```

## 4.布尔运算符

下表列出了常用的布尔运算符，假定变量 **a** 为 10，变量 **b** 为 20：

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。

实例

布尔运算符实例如下：

```
\#!/bin/bash
\# author: 菜鸟教程
\# url:www.runoob.com

a=10
b=20

if [ $a != $b ]
then
    echo "$a != $b : a 不等于 b"
```

```

else
    echo "$a == $b: a 等于 b"
fi
if [ $a -lt 100 -a $b -gt 15 ]
then
    echo "$a 小于 100 且 $b 大于 15 : 返回 true"
else
    echo "$a 小于 100 且 $b 大于 15 : 返回 false"
fi
if [ $a -lt 100 -o $b -gt 100 ]
then
    echo "$a 小于 100 或 $b 大于 100 : 返回 true"
else
    echo "$a 小于 100 或 $b 大于 100 : 返回 false"
fi
if [ $a -lt 5 -o $b -gt 100 ]
then
    echo "$a 小于 5 或 $b 大于 100 : 返回 true"
else
    echo "$a 小于 5 或 $b 大于 100 : 返回 false"
fi

```

执行脚本，输出结果如下所示：

```

10 != 20 : a 不等于 b
10 小于 100 且 20 大于 15 : 返回 true
10 小于 100 或 20 大于 100 : 返回 true
10 小于 5 或 20 大于 100 : 返回 false

```

## 5.逻辑运算符

以下介绍 Shell 的逻辑运算符，假定变量 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

实例

逻辑运算符实例如下：

```

\#!/bin/bash
\# author: 菜鸟教程
\# url:www.runoob.com

a=10

```

```
b=20

if [[ $a -lt 100 && $b -gt 100 ]]
then
    echo "返回 true"
else
    echo "返回 false"
fi

if [[ $a -lt 100 || $b -gt 100 ]]
then
    echo "返回 true"
else
    echo "返回 false"
fi
```

执行脚本，输出结果如下所示：

```
返回 false
返回 true
```

## 6.字符串运算符

下表列出了常用的字符串运算符，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。	[ <code>\$a = \$b</code> ] 返回 false。
!=	检测两个字符串是否不相等，不相等返回 true。	[ <code>\$a != \$b</code> ] 返回 true。
-z	检测字符串长度是否为 0，为 0 返回 true。	[ <code>-z \$a</code> ] 返回 false。
-n	检测字符串长度是否不为 0，不为 0 返回 true。	[ <code>-n "\$a"</code> ] 返回 true。
\$	检测字符串是否为空，不为空返回 true。	[ <code>\$a</code> ] 返回 true。

实例

字符串运算符实例如下：

```
#!/bin/bash
\# author: 菜鸟教程
\# url:www.runoob.com

a="abc"
b="efg"
```

```

if [ $a = $b ]
then
    echo "$a = $b : a 等于 b"
else
    echo "$a = $b: a 不等于 b"
fi
if [ $a != $b ]
then
    echo "$a != $b : a 不等于 b"
else
    echo "$a != $b: a 等于 b"
fi
if [ -z $a ]
then
    echo "-z $a : 字符串长度为 0"
else
    echo "-z $a : 字符串长度不为 0"
fi
if [ -n "$a" ]
then
    echo "-n $a : 字符串长度不为 0"
else
    echo "-n $a : 字符串长度为 0"
fi
if [ $a ]
then
    echo "$a : 字符串不为空 "
else
    echo "$a : 字符串为空 "
fi

```

执行脚本，输出结果如下所示：

```

abc = efg: a 不等于 b
abc != efg : a 不等于 b
-z abc : 字符串长度不为 0
-n abc : 字符串长度不为 0
abc : 字符串不为空

```

## 7.文件测试运算符

文件测试运算符用于检测 Unix 文件的各种属性。

属性检测描述如下：

操  
作  
符

说明

举例



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

其他检查符：

- **-S**: 判断某文件是否 `socket`。
- **-L**: 检测文件是否存在并且是一个符号链接。

实例

变量 **file** 表示文件 **/var/www/runoob/test.sh**，它的大小为 100 字节，具有 **rwX** 权限。下面的代码，将检测该文件的各种属性：

```
#!/bin/bash
\# author: 菜鸟教程
\# url:www.runoob.com

file="/var/www/runoob/test.sh"
if [ -r $file ]
then
    echo "文件可读"
else
    echo "文件不可读"
fi
if [ -w $file ]
then
    echo "文件可写"
else
    echo "文件不可写"
fi
if [ -x $file ]
then
    echo "文件可执行"
else
    echo "文件不可执行"
fi
if [ -f $file ]
then
    echo "文件为普通文件"
else
    echo "文件为特殊文件"
fi
if [ -d $file ]
then
    echo "文件是个目录"
else
    echo "文件不是个目录"
fi
if [ -s $file ]
then
    echo "文件不为空"
else
    echo "文件为空"
fi
if [ -e $file ]
then
    echo "文件存在"
else
    echo "文件不存在"
```

执行脚本，输出结果如下所示：

```
文件可读
文件可写
文件可执行
文件为普通文件
文件不是个目录
文件不为空
文件存在
```

## 参考链接

<https://www.runoob.com/linux/linux-shell-basic-operators.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术" 加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## Shell 流程控制

### 1. 流程控制

Shell 流程控制 和 Java、PHP 等语言不一样，sh 的流程控制不可为空，如（以下为 PHP 流程控制写法）：实例 [mycode4 type='php']..

和 Java、PHP 等语言不一样，sh 的流程控制不可为空，如（以下为 PHP 流程控制写法）：

实例

```
<?php
if (isset($_GET["q"])) {
    search(q);
}
else {
    // 不做任何事情
}
```

在 sh/bash 里可不能这么写，如果 else 分支没有语句执行，就不要写这个 else。

## 2.if else

fi

if 语句语法格式：

```
if condition
then
    command1
    command2
    ...
    commandN
fi
```

写成一行（适用于终端命令提示符）：

```
if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi
```

末尾的 **fi** 就是 **if** 倒过来拼写，后面还会遇到类似的。

## 3.if else

if else 语法格式：

```
if condition
then
    command1
    command2
    ...
    commandN
else
    command
fi
```

## 4.if else-if else

if else-if else 语法格式：

```
if condition1
then
    command1
elif condition2
then
    command2
else
    commandN
fi
```

以下实例判断两个变量是否相等：

## 实例

```
a=10
b=20
if [ $a == $b ]
then
    echo "a 等于 b"
elif [ $a -gt $b ]
then
    echo "a 大于 b"
elif [ $a -lt $b ]
then
    echo "a 小于 b"
else
    echo "没有符合条件的"
fi
```

输出结果：

```
a 小于 b
```

if else 语句经常与 test 命令结合使用，如下所示：

## 实例

```
num1=$((2*3))
num2=$((1+5))
if test ${num1} -eq ${num2}
then
    echo '两个数字相等！'
else
    echo '两个数字不相等！'
fi
```

输出结果：

```
两个数字相等！
```

## 5.for 循环

与其他编程语言类似，Shell 支持 for 循环。

for 循环一般格式为：

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

写成一行：

```
for var in item1 item2 ... itemN; do command1; command2... done;
```

当变量值在列表里，**for** 循环即执行一次所有命令，使用变量名获取列表中的当前取值。命令可为任何有效的 **shell** 命令和语句。**in** 列表可以包含替换、字符串和文件名。

**in** 列表是可选的，如果不用它，**for** 循环使用命令行的位置参数。

例如，顺序输出当前列表中的数字：

实例

```
for loop in 1 2 3 4 5
do
    echo "The value is: $loop"
done
```

输出结果：

```
The value is: 1
The value is: 2
The value is: 3
The value is: 4
The value is: 5
```

顺序输出字符串中的字符：

```
#!/bin/bash

for str in This is a string
do
    echo $str
done
```

输出结果：

```
This  
is  
a  
string
```

## 6.while 语句

**while** 循环用于不断执行一系列命令，也用于从输入文件中读取数据。其语法格式为：

```
while condition  
do  
    command  
done
```

以下是一个基本的 **while** 循环，测试条件是：如果 **int** 小于等于 5，那么条件返回真。**int** 从 1 开始，每次循环处理时，**int** 加 1。运行上述脚本，返回数字 1 到 5，然后终止。

实例

```
\#!/bin/bash  
int=1  
while(( $int<=5 ))  
do  
    echo $int  
    let "int++"  
done
```

运行脚本，输出：

```
1  
2  
3  
4  
5
```

以上实例使用了 **Bash let** 命令，它用于执行一个或多个表达式，变量计算中不需要加上 **\$** 来表示变量，具体可查阅：[Bash let 命令](#)

。

**while** 循环可用于读取键盘信息。下面的例子中，输入信息被设置为变量 **FILM**，按 **Ctrl-D** 结束循环。

实例

```
echo '按下 <CTRL-D> 退出'
echo -n '输入你最喜欢的网站名:'
while read FILM
do
    echo " 是的! $FILM 是一个好网站 "
done
```

运行脚本，输出类似下面：

```
按下 <CTRL-D> 退出
输入你最喜欢的网站名:菜鸟教程
是的! 菜鸟教程 是一个好网站
```

## 7.无限循环

无限循环语法格式：

```
while :
do
    command
done
```

或者

```
while true
do
    command
done
```

或者

```
for (( ; ; ))
```

## 8.until 循环

until 循环执行一系列命令直至条件为 true 时停止。

until 循环与 while 循环在处理方式上刚好相反。

一般 while 循环优于 until 循环，但在某些时候——也只是极少数情况下，until 循环更加有用。

until 语法格式：

```
until condition
do
    command
done
```



**condition** 一般为条件表达式，如果返回值为 **false**，则继续执行循环体内的语句，否则跳出循环。

以下实例我们使用 **until** 命令来输出 0 ~ 9 的数字：

实例

```
\#!/bin/bash

a=0

until [ ! $a -lt 10 ]
do
    echo $a
    a=`expr $a + 1`
done
```

运行结果：

输出结果为：

```
0
1
2
3
4
5
6
7
8
9
```

## 9.case ... esac

**case ... esac** 为多选择语句，与其他语言中的 **switch ... case** 语句类似，是一种多分枝选择结构，每个 **case** 分支用右圆括号开始，用两个分号 **;;** 表示 **break**，即执行结束，跳出整个 **case ... esac** 语句，**esac**（就是 **case** 反过来）作为结束标记。

可以用 **case** 语句匹配一个值与一个模式，如果匹配成功，执行相匹配的命令。

**case ... esac** 语法格式如下：

```
case 值 in
模式1)
    command1
    command2
    ...
    commandN
;;
模式2)
```

```
command1
command2
...
commandN
;;
esac
```

**case** 工作方式如上所示，取值后面必须为单词 **in**，每一模式必须以右括号结束。取值可以为变量或常数，匹配发现取值符合某一模式后，其间所有命令开始执行直至 **;;**。

取值将检测匹配的每一个模式。一旦模式匹配，则执行完匹配模式相应命令后不再继续其他模式。如果无一匹配模式，使用星号 **\*** 捕获该值，再执行后面的命令。

下面的脚本提示输入 1 到 4，与每一种模式进行匹配：

### 实例

```
echo '输入 1 到 4 之间的数字:'
echo '你输入的数字为:'
read aNum
case $aNum in
    1) echo '你选择了 1'
        ;;
    2) echo '你选择了 2'
        ;;
    3) echo '你选择了 3'
        ;;
    4) echo '你选择了 4'
        ;;
    *) echo '你没有输入 1 到 4 之间的数字'
        ;;
esac
```

输入不同的内容，会有不同的结果，例如：

```
输入 1 到 4 之间的数字:
你输入的数字为:
3
你选择了 3
```

下面的脚本匹配字符串：

### 实例

```
\#!/bin/sh

site="runoob"

case "$site" in
    "runoob") echo "菜鸟教程"
        ;;
    "google") echo "Google 搜索"
        ;;
    "taobao") echo "淘宝网"
        ;;
    esac
```

输出结果为：

菜鸟教程

## 10.跳出循环

在循环过程中，有时候需要在未达到循环结束条件时强制跳出循环，Shell 使用两个命令来实现该功能：**break** 和 **continue**。

## 11.break 命令

**break** 命令允许跳出所有循环（终止执行后面的所有循环）。

下面的例子中，脚本进入死循环直至用户输入数字大于 5。要跳出这个循环，返回到 **shell** 提示符下，需要使用 **break** 命令。

实例

```
\#!/bin/bash
while :
do
    echo -n "输入 1 到 5 之间的数字："
    read aNum
    case $aNum in
        1|2|3|4|5) echo " 你输入的数字为 $aNum!"
            ;;
        *) echo "你输入的数字不是 1 到 5 之间的！游戏结束"
            break
            ;;
    esac
done
```

执行以上代码，输出结果为：

```
输入 1 到 5 之间的数字:3
你输入的数字为 3!
输入 1 到 5 之间的数字:7
你输入的数字不是 1 到 5 之间的! 游戏结束
```

## 12.continue

`continue` 命令与 `break` 命令类似，只有一点差别，它不会跳出所有循环，仅仅跳出当前循环。

对上面的例子进行修改：

实例

```
\#!/bin/bash
while :
do
    echo -n "输入 1 到 5 之间的数字:"
    read aNum
    case $aNum in
        1|2|3|4|5) echo " 你输入的数字为 $aNum!"
            ;;
        *) echo "你输入的数字不是 1 到 5 之间的!"
            continue
            echo "游戏结束"
            ;;
    esac
done
```

运行代码发现，当输入大于 5 的数字时，该例中的循环不会结束，语句 `echo "游戏结束"` 永远不会被执行。

## 参考链接

<https://www.runoob.com/linux/linux-shell-process-control.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# Shell 输入 / 输出重定向

## 1.定义

Shell 输入 / 输出重定向 大多数 UNIX 系统命令从你的终端接受输入并将所产生的输出发送回到您的终端。

大多数 UNIX 系统命令从你的终端接受输入并将所产生的输出发送回到您的终端。一个命令通常从一个叫标准输入的地方读取输入，默认情况下，这恰好是你的终端。同样，一个命令通常将其输出写入到标准输出，默认情况下，这也是你的终端。

重定向命令列表如下：

命令	说明
<code>command &gt; file</code>	将输出重定向到 <code>file</code> 。
<code>command &lt; file</code>	将输入重定向到 <code>file</code> 。
<code>command &gt;&gt; file</code>	将输出以追加的方式重定向到 <code>file</code> 。
<code>n &gt; file</code>	将文件描述符为 <code>n</code> 的文件重定向到 <code>file</code> 。
<code>n &gt;&gt; file</code>	将文件描述符为 <code>n</code> 的文件以追加的方式重定向到 <code>file</code> 。
<code>n &gt;&amp; m</code>	将输出文件 <code>m</code> 和 <code>n</code> 合并。
<code>n &lt;&amp; m</code>	将输入文件 <code>m</code> 和 <code>n</code> 合并。
<code>&lt;&lt; tag</code>	将开始标记 <code>tag</code> 和结束标记 <code>tag</code> 之间的内容作为输入。

需要注意的是文件描述符 0 通常是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）。

## 2.输出重定向

重定向一般通过在命令间插入特定的符号来实现。特别的，这些符号的语法如下所示：

```
command1 > file1
```

上面这个命令执行 `command1` 然后将输出的内容存入 `file1`。

注意任何 `file1` 内的已经存在的内容将被新内容替代。如果要将新内容添加在文件末尾，请使用 `>>` 操作符。

实例

执行下面的 **who** 命令，它将命令的完整的输出重定向在用户文件中 (**users**):

```
$ who > users
```

执行后，并没有在终端输出信息，这是因为输出已被从默认的标准输出设备（终端）重定向到指定的文件。

你可以使用 **cat** 命令查看文件内容：

```
$ cat users
_mbsetupuser console Oct 31 17:35
tianqixin console Oct 31 17:35
tianqixin ttys000 Dec 1 11:33
```

输出重定向会覆盖文件内容，请看下面的例子：

```
$ echo "菜鸟教程: www.runoob.com" > users
$ cat users
菜鸟教程: www.runoob.com
$
```

如果不希望文件内容被覆盖，可以使用 **>>** 追加到文件末尾，例如：

```
$ echo "菜鸟教程: www.runoob.com" >> users
$ cat users
菜鸟教程: www.runoob.com
菜鸟教程: www.runoob.com
$
```

### 3.输入重定向

和输出重定向一样，**Unix** 命令也可以从文件获取输入，语法为：

```
command1 < file1
```

这样，本来需要从键盘获取输入的命令会转移到文件读取内容。

注意：输出重定向是大于号 (**>**)，输入重定向是小于号 (**<**)。

实例

接着以上实例，我们需要统计 **users** 文件的行数, 执行以下命令：

```
$ wc -l users
2 users
```

也可以将输入重定向到 **users** 文件：

```
$ wc -l < users
2
```

注意：上面两个例子的结果不同：第一个例子，会输出文件名；第二个不会，因为它仅仅知道从标准输入读取内容。

```
command1 < infile > outfile
```

同时替换输入和输出，执行 `command1`，从文件 `infile` 读取内容，然后将输出写入到 `outfile` 中。

## 4. 重定向深入讲解

一般情况下，每个 Unix/Linux 命令运行时都会打开三个文件：

- 标准输入文件 (`stdin`)： `stdin` 的文件描述符为 0，Unix 程序默认从 `stdin` 读取数据。
- 标准输出文件 (`stdout`)： `stdout` 的文件描述符为 1，Unix 程序默认向 `stdout` 输出数据。
- 标准错误文件 (`stderr`)： `stderr` 的文件描述符为 2，Unix 程序会向 `stderr` 流中写入错误信息。

默认情况下，`command > file` 将 `stdout` 重定向到 `file`，`command < file` 将 `stdin` 重定向到 `file`。

如果希望 `stderr` 重定向到 `file`，可以这样写：

```
$ command 2>file
```

如果希望 `stderr` 追加到 `file` 文件末尾，可以这样写：

```
$ command 2>>file
```

**2** 表示标准错误文件 (`stderr`)。

如果希望将 `stdout` 和 `stderr` 合并后重定向到 `file`，可以这样写：

```
$ command > file 2>&1
```

或者

```
$ command >> file 2>&1
```

如果希望对 `stdin` 和 `stdout` 都重定向，可以这样写：

```
$ command < file1 >file2
```

`command` 命令将 `stdin` 重定向到 `file1`，将 `stdout` 重定向到 `file2`。

## 5.Here Document

Here Document 是 Shell 中的一种特殊的重定向方式，用来将输入重定向到一个交互式 Shell 脚本或程序。

它的基本的形式如下：

```
command << delimiter
    document
delimiter
```

它的作用是将两个 **delimiter** 之间的内容 (**document**) 作为输入传递给 **command**。

注意：

- 结尾的 **delimiter** 一定要顶格写，前面不能有任何字符，后面也不能有任何字符，包括空格和 **tab** 缩进。
- 开始的 **delimiter** 前后的空格会被忽略掉。

实例

在命令行中通过 **wc -l** 命令计算 Here Document 的行数：

```
$ wc -l << EOF
    欢迎来到
    菜鸟教程
    www.runoob.com
EOF
3          # 输出结果为 3 行
$
```

我们也可以将 Here Document 用在脚本中，例如：

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

cat << EOF
欢迎来到
菜鸟教程
www.runoob.com
EOF
```

执行以上脚本，输出结果：

```
欢迎来到
菜鸟教程
www.runoob.com
```



## 6./dev/null 文件

如果希望执行某个命令，但又不希望在屏幕上显示输出结果，那么可以将输出重定向到 /dev/null：

```
$ command > /dev/null
```

/dev/null 是一个特殊的文件，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。但是 /dev/null 文件非常有用，将命令的输出重定向到它，会起到 "禁止输出" 的效果。

如果希望屏蔽 stdout 和 stderr，可以这样写：

```
$ command > /dev/null 2>&1
```

注意：0 是标准输入（STDIN），1 是标准输出（STDOUT），2 是标准错误输出（STDERR）。

这里的 2 和 > 之间不可以有空格，2> 是一体的时候才表示错误输出。

### 参考链接

<https://www.runoob.com/linux/linux-shell-io-redirections.html>



扫码关注，收获知识

程序员百科全书

关注公众号 回复 "技术" 加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

## 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（与 PHP 类似）。

与大部分编程语言类似，数组元素的下标由 0 开始。

Shell 数组用括号来表示，元素用 "空格" 符号分割开，语法格式如下：

```
array_name=(value1 value2 ... valuen)
```

## 1.语法

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

my_array=(A B "C" D)
```

我们也可以使用下标来定义数组：

```
array_name[0]=value0
array_name[1]=value1
array_name[2]=value2
```

## 2.读取数组

读取数组元素值的一般格式是：

```
${array_name[index]}
```

实例

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

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]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh
第一个元素为: A
第二个元素为: B
第三个元素为: C
第四个元素为: D
```

## 3.获取数组中的所有元素

使用@ 或 \* 可以获取数组中的所有元素，例如：

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

my_array[0]=A
my_array[1]=B
my_array[2]=C
my_array[3]=D

echo "数组的元素为: ${my_array[*]}"
echo "数组的元素为: ${my_array[@]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh
数组的元素为: A B C D
数组的元素为: A B C D
```

## 4.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：

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

my_array[0]=A
my_array[1]=B
my_array[2]=C
my_array[3]=D

echo "数组元素个数为: ${#my_array[*]}"
echo "数组元素个数为: ${#my_array[@]}"
```

执行脚本，输出结果如下所示：

```
$ chmod +x test.sh
$ ./test.sh
数组元素个数为: 4
数组元素个数为: 4
```

## 参考链接

<https://www.runoob.com/linux/linux-shell-array.html>



扫码关注，收获知识

## 程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题

# Shell文件包含

## 1.语法

和其他语言一样，Shell 也可以包含外部脚本。这样可以很方便的封装一些公用的代码作为一个独立的文件。

Shell 文件包含的语法格式如下：

```
. filename    # 注意点号(.)和文件名中间有一空格
```

或

```
source filename
```

## 2.创建shell脚本

创建两个 shell 脚本文件。

test1.sh 代码如下：

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

url="http://www.runoob.com"
```

test2.sh 代码如下：

```
#!/bin/bash
# author:菜鸟教程
# url:www.runoob.com

#使用 . 号来引用test1.sh 文件
. ./test1.sh

# 或者使用以下包含文件代码
# source ./test1.sh

echo "菜鸟教程官网地址: $url"
```

接下来，我们为 `test2.sh` 添加可执行权限并执行：

```
$ chmod +x test2.sh
$ ./test2.sh
菜鸟教程官网地址: http://www.runoob.com
```

**注：**被包含的文件 `test1.sh` 不需要可执行权限。

## 参考链接

<https://www.runoob.com/linux/linux-shell-include-file.html>



扫码关注，收获知识

程序员百科全书

关注公众号 回复 "技术"加入技术交流群

关注公众号 回复 "linux" 获取Linux面试题