

1.switch 能作用在 byte 上吗？

答：switch(expr1)中，expr1 只能是一个整数表达式或者枚举常量。Expr1 只能是 byte、short、char、int 四个整数类型和枚举类型。Java7 以后增加了 String 类型。

二分查找：

```
// 二分查找递归实现
public static int binSearch(int srcArray[], int start, int end, int key) {
    int mid = (end - start) / 2 + start;
    if (srcArray[mid] == key) {
        return mid;
    }
    if (start >= end) {
        return -1;
    } else if (key > srcArray[mid]) {
        return binSearch(srcArray, mid + 1, end, key);
    } else if (key < srcArray[mid]) {
        return binSearch(srcArray, start, mid - 1, key);
    }
    return -1;
}
```

```
// 二分查找普通循环实现
public static int binSearch(int srcArray[], int key) {
    int mid = srcArray.length / 2;
    if (key == srcArray[mid]) {
        return mid;
    }

    int start = 0;
    int end = srcArray.length - 1;
    while (start <= end) {
        mid = (end - start) / 2 + start;
        if (key < srcArray[mid]) {
            end = mid - 1;
        } else if (key > srcArray[mid]) {
            start = mid + 1;
        } else {
            return mid;
        }
    }
    return -1;
}
```

单例模式：

```
1. public class Singleton{
2.     private static class SingletonHolder{
3.         public static Singleton instance = new Singleton();
4.     }
5.     private Singleton(){}
6.     public static Singleton newInstance(){
7.         return SingletonHolder.instance;
8.     }
9. }
```

工厂模式：

快速排序：平均时间复杂度（ $n \log n$ ）

```
[html]
01. public int getMiddle(Integer[] list, int low, int high) {
02.     int tmp = list[low];    //数组的第一个作为中轴
03.     while (low < high) {
04.         while (low < high && list[high] > tmp) {
05.             high--;
06.         }
07.         list[low] = list[high];    //比中轴小的记录移到低端
08.         while (low < high && list[low] < tmp) {
09.             low++;
10.         }
11.         list[high] = list[low];    //比中轴大的记录移到高端
12.     }
13.     list[low] = tmp;              //中轴记录到尾
14.     return low;                  //返回中轴的位置
15. }
```

递归形式的分治排序算法：

```
[html]
01. public void _quickSort(Integer[] list, int low, int high) {
02.     if (low < high) {
03.         int middle = getMiddle(list, low, high);    //将list数组进行一分为二
04.         _quickSort(list, low, middle - 1);        //对低字表进行递归排序
05.         _quickSort(list, middle + 1, high);        //对高字表进行递归排序
06.     }
07. }
```

玩过 Linux 的人都会知道，Linux 中的命令的确是非常多，但是玩过 Linux 的人也从来不会因为 Linux 的命令如此之多而烦恼，因为我们只需要掌握我们最常用的命令就可以了。当然你也可以在使用时去找一下 man，他会帮你解决不少的问题。然而每个人玩 Linux 的目的都不同，所以他们常用的命令也就差异非常大，而我主要是用 Linux 进行 C/C++ 和 shell 程序编写的，所以常用到的命令就会跟一个管理 Linux 系统的人有所不同。因为不想在使用是总是东查西找，所以在此总结一下，方便一下以后的查看。不多说，下面就说说我最常用的 Linux 命令。

## 1、cd 命令

这是一个非常基本，也是大家经常需要使用的命令，它用于切换当前目录，它的参数是要切换到的目录的路径，可以是绝对路径，也可以是相对路径。如：

```
[plain] view plain copy
print?
```

1. `cd /root/Documents` # 切换到目录/root/Documents
2. `cd ./path` # 切换到当前目录下的 `path` 目录中，“.”表示当前目录
3. `cd ../path` # 切换到上层目录中的 `path` 目录中，“..”表示上一层目录

## 2、ls 命令

这是一个非常有用的查看文件与目录的命令，list 之意，它的参数非常多，下面就列出一些我常用的参数吧，如下：

```
[plain] view plain copy
print?
```

1. `-l` : 列出长数据串，包含文件的属性与权限数据等
2. `-a` : 列出全部的文件，连同隐藏文件（开头为.的文件）一起列出来（常用）
3. `-d` : 仅列出目录本身，而不是列出目录的文件数据
4. `-h` : 将文件容量以较易读的方式（GB, kB 等）列出来
5. `-R` : 连同子目录的内容一起列出（递归列出），等于该目录下的所有文件都会显示出来

注：这些参数也可以组合使用，下面举两个例子：

```
[plain] view plain copy
print?
```

1. `ls -l` #以长数据串的形式列出当前目录下的数据文件和目录
2. `ls -lR` #以长数据串的形式列出当前目录下的所有文件

## 3、grep 命令

该命令常用于分析一行的信息，若当中有我们所需要的信息，就将该行显示出来，该命令通常与管道命令一起使用，用于对一些命令的输出进行筛选加工等等，它的简单语法为

```
[plain] view plain copy
print?
```

1. `grep [-acinv] [--color=auto] '查找字符串' filename`

它的常用参数如下：

```
[plain] view plain copy
print?
```

1. `-a` : 将 `binary` 文件以 `text` 文件的方式查找数据
2. `-c` : 计算找到‘查找字符串’的次数

3. **-i** : 忽略大小写的区别, 即把大小写视为相同
4. **-v** : 反向选择, 即显示出没有‘查找字符串’内容的那一行
5. # 例如:
6. # 取出文件/etc/man.config 中包含 MANPATH 的行, 并把找到的关键字加上颜色
7. `grep --color=auto 'MANPATH' /etc/man.config`
8. # 把 `ls -l` 的输出中包含字母 **file** (不区分大小写) 的内容输出
9. `ls -l | grep -i file`

#### 4、find 命令

find 是一个基于查找的功能非常强大的命令, 相对而言, 它的使用也相对较为复杂, 参数也比较多, 所以在这里将给把它们分类列出, 它的基本语法如下:

[plain] view plain copy

print?

1. `find [PATH] [option] [action]`
- 2.
3. # 与时间有关的参数:
4. **-mtime n** : n 为数字, 意思为在 n 天之前的“一天内”被更改过的文件;
5. **-mtime +n** : 列出在 n 天之前 (不含 n 天本身) 被更改过的文件名;
6. **-mtime -n** : 列出在 n 天之内 (含 n 天本身) 被更改过的文件名;
7. **-newer file** : 列出比 **file** 还要新的文件名
8. # 例如:
9. `find /root -mtime 0` # 在当前目录下查找今天之内有改动的文件
- 10.
11. # 与用户或用户组名有关的参数:
12. **-user name** : 列出文件所有者为 **name** 的文件
13. **-group name** : 列出文件所属用户组为 **name** 的文件
14. **-uid n** : 列出文件所有者为用户 ID 为 n 的文件
15. **-gid n** : 列出文件所属用户组为用户组 ID 为 n 的文件
16. # 例如:
17. `find /home/ljianhui -user ljianhui` # 在目录/home/ljianhui 中找出所有者为 ljianhui 的文件
- 18.
19. # 与文件权限及名称有关的参数:
20. **-name filename** : 找出文件名为 **filename** 的文件
21. **-size [+ -]SIZE** : 找出比 **SIZE** 还要大 (+) 或小 (-) 的文件
22. **-type TYPE** : 查找文件的类型为 **TYPE** 的文件, **TYPE** 的值主要有: 一般文件 (**f**)、设备文件 (**b**、**c**)、
23. 目录 (**d**)、连接文件 (**l**)、socket (**s**)、FIFO 管道文件 (**p**);
24. **-perm mode** : 查找文件权限刚好等于 **mode** 的文件, **mode** 用数字表示, 如 0755;
25. **-perm -mode** : 查找文件权限必须要全部包括 **mode** 权限的文件, **mode** 用数字表示
26. **-perm +mode** : 查找文件权限包含任一 **mode** 的权限的文件, **mode** 用数字表示
27. # 例如:
28. `find / -name passwd` # 查找文件名为 **passwd** 的文件

```
29. find . -perm 0755 # 查找当前目录中文件权限的 0755 的文件
30. find . -size +12k # 查找当前目录中大于 12KB 的文件，注意 c 表示 byte
```

## 5、cp 命令

该命令用于复制文件，copy 之意，它还可以把多个文件一次性地复制到一个目录下，它的常用参数如下：

```
[plain] view plain copy
print?
```

1. **-a** : 将文件的特性一起复制
2. **-p** : 连同文件的属性一起复制，而非使用默认方式，与 **-a** 相似，常用于备份
3. **-i** : 若目标文件已经存在时，在覆盖时会先询问操作的进行
4. **-r** : 递归持续复制，用于目录的复制行为
5. **-u** : 目标文件与源文件有差异时才会复制

例如：

```
[plain] view plain copy
print?
```

1. **cp -a file1 file2** # 连同文件的所有特性把文件 **file1** 复制成文件 **file2**
2. **cp file1 file2 file3 dir** # 把文件 **file1**、**file2**、**file3** 复制到目录 **dir** 中

## 6、mv 命令

该命令用于移动文件、目录或更名，move 之意，它的常用参数如下：

```
[plain] view plain copy
print?
```

1. **-f** : **force** 强制的意思，如果目标文件已经存在，不会询问而直接覆盖
2. **-i** : 若目标文件已经存在，就会询问是否覆盖
3. **-u** : 若目标文件已经存在，且比目标文件新，才会更新

注：该命令可以把一个文件或多个文件一次移动一个文件夹中，但是最后一个目标文件一定要是“目录”。

例如：

```
[plain] view plain copy
print?
```

1. **mv file1 file2 file3 dir** # 把文件 **file1**、**file2**、**file3** 移动到目录 **dir** 中
2. **mv file1 file2** # 把文件 **file1** 重命名为 **file2**

## 7、rm 命令

该命令用于删除文件或目录，`remove` 之间，它的常用参数如下：

[plain] view plain copy

print?

1. `-f` : 就是 **force** 的意思，忽略不存在的文件，不会出现警告消息
2. `-i` : 互动模式，在删除前会询问用户是否操作
3. `-r` : 递归删除，最常用于目录删除，它是一个非常危险的参数

例如：

[plain] view plain copy

print?

1. `rm -i file` # 删除文件 **file**，在删除之前会询问是否进行该操作
2. `rm -fr dir` # 强制删除目录 **dir** 中的所有文件

## 8、ps 命令

该命令用于将某个时间点的进程运行情况选取下来并输出，`process` 之意，它的常用参数如下：

[plain] view plain copy

print?

1. `-A` : 所有的进程均显示出来
2. `-a` : 不与 **terminal** 有关的所有进程
3. `-u` : 有效用户的相关进程
4. `-x` : 一般与 **a** 参数一起使用，可列出较完整的信息
5. `-l` : 较长，较详细地将 **PID** 的信息列出

其实我们只要记住 `ps` 一般使用的命令参数搭配即可，它们并不多，如下：

[plain] view plain copy

print?

1. `ps aux` # 查看系统所有的进程数据
2. `ps ax` # 查看不与 **terminal** 有关的所有进程
3. `ps -lA` # 查看系统所有的进程数据
4. `ps axjf` # 查看连同一部分进程树状态

## 9、kill 命令

该命令用于向某个工作（%jobnumber）或者是某个 **PID**（数字）传送一个信号，它通常与 `ps` 和 `jobs` 命令一起使用，它的基本语法如下：

[plain] view plain copy

print?

1. `kill -signal PID`

signal 的常用参数如下：

注：最前面的数字为信号的代号，使用时可以用代号代替相应的信号。

```
[plain] view plain copy  
print?
```

1. 1: SIGHUP, 启动被终止的进程
2. 2: SIGINT, 相当于输入 **ctrl+c**, 中断一个程序的进行
3. 9: SIGKILL, 强制中断一个进程的进程
4. 15: SIGTERM, 以正常的结束进程方式来终止进程
5. 17: SIGSTOP, 相当于输入 **ctrl+z**, 暂停一个进程的进程

例如：

```
[plain] view plain copy  
print?
```

1. # 以正常的结束进程方式来终于第一个后台工作，可用 **jobs** 命令查看后台中的第一个工作进程
2. **kill -SIGTERM %1**
3. # 重新改动进程 ID 为 PID 的进程, PID 可用 **ps** 命令通过管道命令加上 **grep** 命令进行筛选获得
4. **kill -SIGHUP PID**

## 10、killall 命令

该命令用于向一个命令启动的进程发送一个信号，它的一般语法如下：

```
[plain] view plain copy  
print?
```

1. **killall [-iIe] [command name]**

它的参数如下：

```
[plain] view plain copy  
print?
```

1. **-i** : 交互式的意思，若需要删除时，会询问用户
2. **-e** : 表示后面接的 **command name** 要一致，但 **command name** 不能超过 15 个字符
3. **-I** : 命令名称忽略大小写
4. # 例如：
5. **killall -SIGHUP syslogd # 重新启动 syslogd**

## 11、file 命令

该命令用于判断接在 file 命令后的文件的基本数据，因为在 Linux 下文件的类型并不是以后缀为分的，所以这个命令对我们来说就很有用了，它的用法非常简单，基本语法如下：

```
[plain] view plain copy
```



```
print?
```

1. `file filename`
2. #例如:
3. `file ./test`

## 12、tar 命令

该命令用于对文件进行打包，默认情况并不会压缩，如果指定了相应的参数，它还会调用相应的压缩程序（如 `gzip` 和 `bzip` 等）进行压缩和解压。它的常用参数如下：

```
[plain] view plain copy
```

```
print?
```

1. `-c` : 新建打包文件
2. `-t` : 查看打包文件的内容含有哪些文件名
3. `-x` : 解打包或解压缩的功能，可以搭配 `-C`（大写）指定解压的目录，注意 `-c`, `-t`, `-x` 不能同时出现在同一条命令中
4. `-j` : 通过 `bzip2` 的支持进行压缩/解压缩
5. `-z` : 通过 `gzip` 的支持进行压缩/解压缩
6. `-v` : 在压缩/解压缩过程中，将正在处理的文件名显示出来
7. `-f filename` : `filename` 为要处理的文件
8. `-C dir` : 指定压缩/解压缩的目录 `dir`

上面的解说可以已经让你晕过去了，但是通常我们只需要记住下面三条命令即可：

```
[plain] view plain copy
```

```
print?
```

1. 压缩: `tar -jcv -f filename.tar.bz2` 要被处理的文件或目录名称
2. 查询: `tar -jtv -f filename.tar.bz2`
3. 解压: `tar -jxv -f filename.tar.bz2 -C` 欲解压缩的目录

注：文件名并不定要以后缀 `tar.bz2` 结尾，这里主要是为了说明使用的压缩程序为 `bzip2`

## 13、cat 命令

该命令用于查看文本文件的内容，后接要查看的文件名，通常可用管道与 `more` 和 `less` 一起使用，从而可以一页页地查看数据。例如：

```
[plain] view plain copy
```

```
print?
```

1. `cat text | less #` 查看 `text` 文件中的内容
2. # 注：这条命令也可以使用 `less text` 来代替

## 14、chgrp 命令

该命令用于改变文件所属用户组，它的使用非常简单，它的基本用法如下：

```
[plain] view plain copy
print?
```

1. `chgrp [-R] dirname/filename`
2. `-R` : 进行递归的持续对所有文件和子目录更改
3. # 例如:
4. `chgrp users -R ./dir` # 递归地把 `dir` 目录下中的所有文件和子目录下所有文件的用户组修改为 `users`

## 15、chown 命令

该命令用于改变文件的所有者，与 `chgrp` 命令的使用方法相同，只是修改的文件属性不同，不再详述。

## 16、chmod 命令

该命令用于改变文件的权限，一般的用法如下：

```
[plain] view plain copy
print?
```

1. `chmod [-R] xyz` 文件或目录
2. `-R`: 进行递归的持续更改，即连同子目录下的所有文件都会更改

同时，`chmod` 还可以使用 `u` (user)、`g` (group)、`o` (other)、`a` (all) 和 `+` (加入)、`-` (删除)、`=` (设置) 跟 `rxw` 搭配来对文件的权限进行更改。

```
[plain] view plain copy
print?
```

1. # 例如:
2. `chmod 0755 file` # 把 `file` 的文件权限改变为 `-rwxr-xr-x`
3. `chmod g+w file` # 向 `file` 的文件权限中加入用户组可写权限

## 18、vim 命令

该命令主要用于文本编辑，它接一个或多个文件名作为参数，如果文件存在就打开，如果文件不存在就以该文件名创建一个文件。`vim` 是一个非常好用的文本编辑器，它里面有很多非常好用的命令，在这里不再多说。你可以从这里下载 [vim 常用操作](#) 的详细说明。

## 19、gcc 命令

对于一个用 Linux 开发 C 程序的人来说，这个命令就非常重要了，它用于把 C 语言的源程序文件，编译成可执行程序，由于 g++ 的很多参数跟它非常相似，所以这里只介绍 gcc 的参数，它的常用参数如下：

[plain] view plain copy

print?

1. **-o** : **output** 之意，用于指定生成一个可执行文件的文件名
2. **-c** : 用于把源文件生成目标文件（.o），并阻止编译器创建一个完整的程序
3. **-I** : 增加编译时搜索头文件的路径
4. **-L** : 增加编译时搜索静态连接库的路径
5. **-S** : 把源文件生成汇编代码文件
6. **-lm**: 表示标准库的目录中名为 **libm.a** 的函数库
7. **-lpthread** : 连接 NPTL 实现的线程库
8. **-std=** : 用于指定使用的 C 语言的版本
- 9.
10. # 例如:
11. # 把源文件 **test.c** 按照 **c99** 标准编译成可执行程序 **test**
12. **gcc -o test test.c -lm -std=c99**
13. #把源文件 **test.c** 转换为相应的汇编程序源文件 **test.s**
14. **gcc -S test.c**

## 20、time 命令

该命令用于测算一个命令（即程序）的执行时间。它的使用非常简单，就像平时输入命令一样，不过在命令的前面加入一个 time 即可，例如：

[plain] view plain copy

print?

1. **time ./process**
2. **time ps aux**

在程序或命令运行结束后，在最后输出了三个时间，它们分别是：

user: 用户 CPU 时间，命令执行完成花费的用户 CPU 时间，即命令在用户态中执行时间总和；

system: 系统 CPU 时间，命令执行完成花费的系统 CPU 时间，即命令在核心态中执行时间总和；




real: 实际时间，从 command 命令行开始执行到运行终止的消逝时间；

## 3.TCP 与 UDP 的区别

## TCP 与 UDP 区别总结:

- 1、TCP 面向连接（如打电话要先拨号建立连接）;UDP 是无连接的，即发送数据之前不需要建立连接
- 2、TCP 提供可靠的服务。也就是说，通过 TCP 连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP 尽最大努力交付，即不保证可靠交付
- 3、TCP 面向字节流，实际上是 TCP 把数据看成一连串无结构的字节流;UDP 是面向报文的
- UDP 没有拥塞控制，因此网络出现拥塞不会使源主机的发送速率降低（对实时应用很有用，如 IP 电话，实时视频会议等）
- 4、每一条 TCP 连接只能是点到点的;UDP 支持一对一，一对多，多对一和多对多的交互通信
- 5、TCP 首部开销 20 字节;UDP 的首部开销小，只有 8 个字节
- 6、TCP 的逻辑通信信道是全双工的可靠信道，UDP 则是不可靠信道

链表反转

```
[cpp]   
01. Node * ReverseList(Node *head)
02. {
03.     Node *p1,*p2,*p3;
04.     if(head==NULL || *head==NULL)
05.         return head;
06.     p1=head;
07.     p2=p1->next;
08.     while(p2)           //注意条件
09.     {
10.         p3=p2->next;     //要改变p2->next的指针，所以必须先保留p2->next
11.         p2->next=p1;
12.         p1=p2;           //循环往后
13.         p2=p3;
14.     }
15.     head->next=NULL;     //原先的head已经变成tail，别忘了置空，只有到这步才能置空
16.     *head=p1;
17.     return head;
18. }
```

## 多线程同步方法

**1 临界区:**通过对多线程的串行化来访问公共资源或一段代码，速度快，适合控制数据访问。

**2 互斥量:**为协调共同对一个共享资源的单独访问而设计的。

**3 信号量:**为控制一个具有有限数量用户资源而设计。

**4 事件:**用来通知线程有一些事件已发生，从而启动后继任务的开始。