



Plain Text | 复制代码

```
1 # groupadd -g 344 runoob
```

4.3.4. 修改用户

`groupmod` 命令用来修改 `group` 相关的参数，例如群组识别码或者名称



LaTeX | 复制代码

```
1 groupmod [options] [groupname]
```

常见的参数有：

- `-g <群组识别码>` 设置欲使用的群组识别码
- `-o` 重复使用群组识别码
- `-n <新群组名称>` 设置欲使用的群组名

例如修改组名：



Plain Text | 复制代码

```
1 # groupmod -n linux linuxso
```

4.3.5. 删除用户组

`groupdel` 用于删除用户组，如果该群组中仍包括某些用户，则必须先删除这些用户后，方能删除群组



LaTeX | 复制代码

```
1 groupdel [groupname]
```

日常工作通常会碰到只有 `root` 用户才有权限执行的操作，这就需要使用用户身份切换的命令：

4.3.6. su

用于变更为其他使用者的身份，除 `root` 外，需要键入该使用者的密码

4.4. sudo

`sudo` 命令以系统管理者的身份执行指令，也就是说，经由 `sudo` 所执行的指令就好像是 `root` 亲自执行

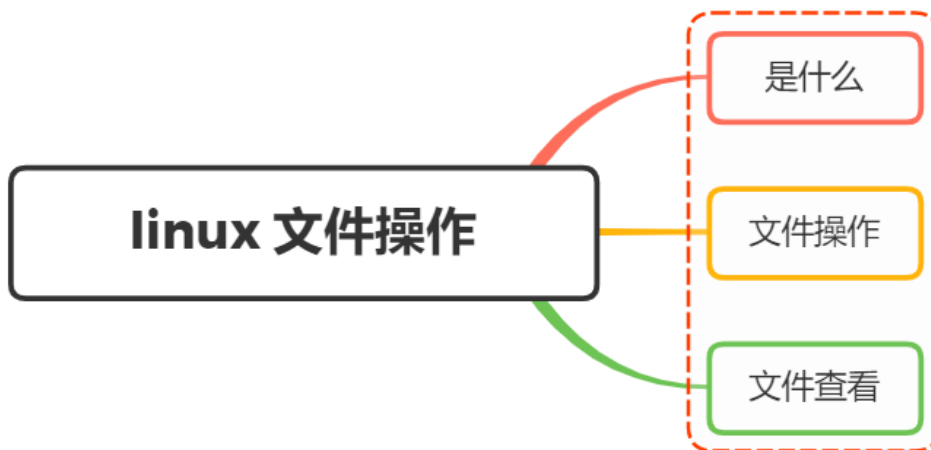
不是所有的用户都能执行 `sudo` 命令的，而是在 `/etc/sudoers` 文件内的用户才能执行这个命令

例如 `sudo` 命令使用 `ls`：

▼ Plain Text | 复制代码

```
1 $ sudo ls
```

5. 说说 linux系统下 文件操作常用的命令有哪些？



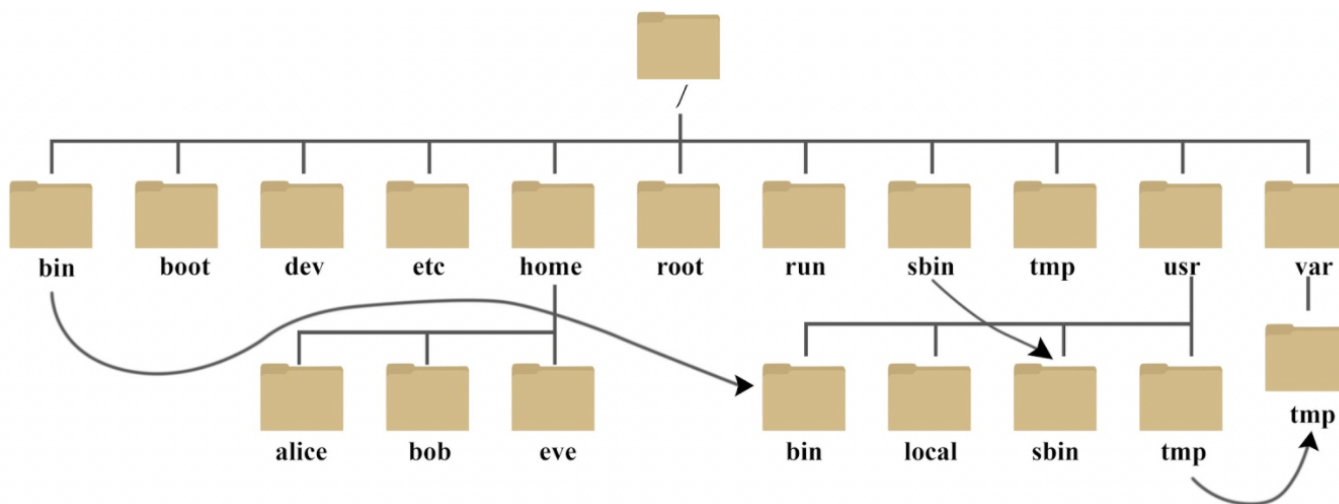
5.1. 是什么

`Linux` 是一个开源的操作系统（OS），是一系列Linux内核基础上开发的操作系统的总称（常见的有Ubuntu、centos）

系统通常会包含以下4个主要部分

- 内核
- shell
- 文件系统
- 应用程序

文件系统是一个**目录树的结构**，文件系统结构从一个根目录开始，根目录下可以有任意多个文件和子目录，子目录中又可以有任意多个文件和子目录



5.2. 文件操作

常见处理目录的命令如下：

- ls（英文全拼：list files）：列出目录及文件名
- cd（英文全拼：change directory）：切换目录
- pwd（英文全拼：print work directory）：显示目前的目录
- mkdir（英文全拼：make directory）：创建一个新的目录
- rmdir（英文全拼：remove directory）：删除一个空的目录
- cp（英文全拼：copy file）：复制文件或目录
- rm（英文全拼：remove）：删除文件或目录
- mv（英文全拼：move file）：移动文件与目录，或修改文件与目录的名称


5.2.1. ls

列出目录文件，选项与参数：

- -a：全部的文件，连同隐藏文件(开头为 . 的文件) 一起列出来(常用)
- -d：仅列出目录本身，而不是列出目录内的文件数据(常用)
- -l：长数据串列出，包含文件的属性与权限等等数据；(常用)

例如将家目录下的所有文件列出来(含属性与隐藏档)



Plain Text |  复制代码

```
1 [root@www ~]# ls -al ~
```

5.2.2. cd

切换工作目录

语法：

▼ Plain Text 复制代码

```
1 cd [相对路径或绝对路径]
```

▼ Plain Text 复制代码

```
1 # 表示回到自己的家目录，亦即是 /root 这个目录
2 [root@www runoob]# cd ~
3
4 # 表示去到目前的上一级目录，亦即是 /root 的上一级目录的意思；
5 [root@www ~]# cd ..
```

5.2.3. pwd

`pwd` 是 `Print Working Directory` 的缩写，也就是显示目前所在目录的命令。

▼ Plain Text 复制代码

```
1 [root@www ~]# pwd [-P]
```

选项与参数：

- `-P`：显示出确实的路径，而非使用连结 (link) 路径

5.2.4. mkdir

创建新目录

语法：

▼ Plain Text 复制代码

```
1 mkdir [-mp] 目录名称
```

选项与参数：

- `-m`：配置文件的权限
- `-p`：帮助你直接将所需要的目录(包含上一级目录)递归创建起来

5.2.5. rmdir (删除空的目录)

语法:

▼ Plain Text | 复制代码

```
1 rmdir [-p] 目录名称
```

选项与参数:

- `-p` : 连同上一级『空的』目录也一起删除

5.2.6. cp

即拷贝文件和目录

语法:

▼ Plain Text | 复制代码

```
1 cp 目标文件 拷贝文件
```

用法如下:

▼ Plain Text | 复制代码

```
1 cp file file_copy --> file 是目标文件, file_copy 是拷贝出来的文件
2 cp file one --> 把 file 文件拷贝到 one 目录下, 并且文件名依然为 file
3 cp file one/file_copy --> 把 file 文件拷贝到 one 目录下, 文件名为file_copy
4 cp *.txt folder --> 把当前目录下所有 txt 文件拷贝到 folder 目录下
5 复制代码
```

常用参数如下:

- `-r` 递归的拷贝, 常用来拷贝一整个目录

5.2.7. rm (移除文件或目录)

语法:

▼ Plain Text | 复制代码

```
1 rm [-fir] 文件或目录
```

选项与参数:

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

5.2.8. mv (移动文件与目录，或修改名称)

语法：

```
1 [root@www ~]# mv [-fiu] source destination
2 [root@www ~]# mv [options] source1 source2 source3 .... directory
```

选项与参数：

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

5.2.9. ln

Linux 文件的存储方式分为3个部分，文件名、文件内容以及权限，其中文件名的列表是存储在硬盘的其它地方和文件内容是分开存放的，每个文件名通过 **inode** 标识绑定到文件内容

Linux 下有两种链接类型：硬链接和软链接

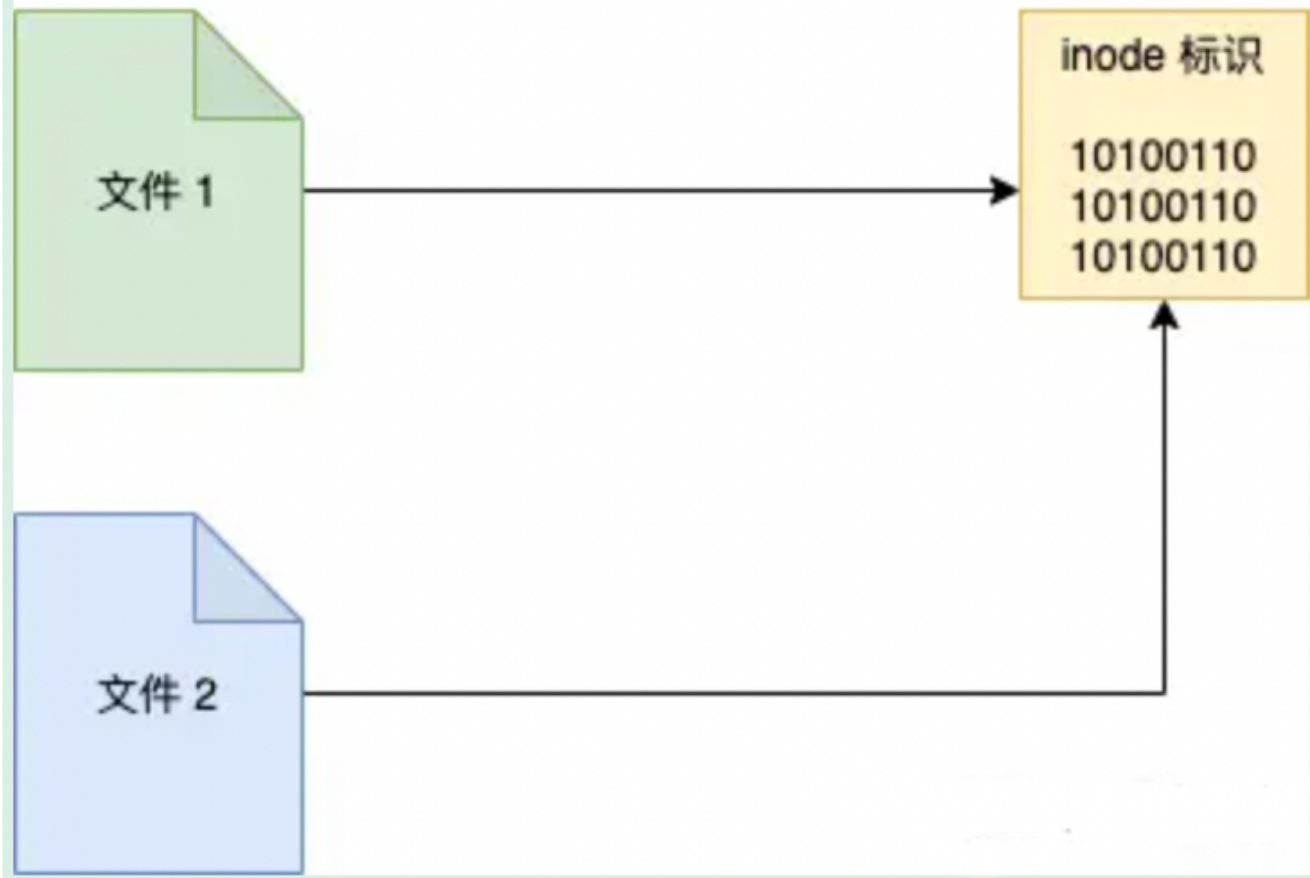
5.2.9.1. 硬链接

使链接的两个文件共享同样文件内容，就是同样的 **inode**，一旦文件1和文件2之间有了硬链接，那么修改任何一个文件，修改的都是同一块内容

语法：

```
1 # 创建 file2 为 file1 的硬链接
2 ln file1 file2
```

硬链接



删除文件1不会影响删除文件2，对于硬链接来说，删除任意一方的文件，共同指向的文件内容并不会从硬盘上删除

只有同时删除了两个文件后，它们共同指向的文件内容才会消失。

5.2.9.2. 软链接

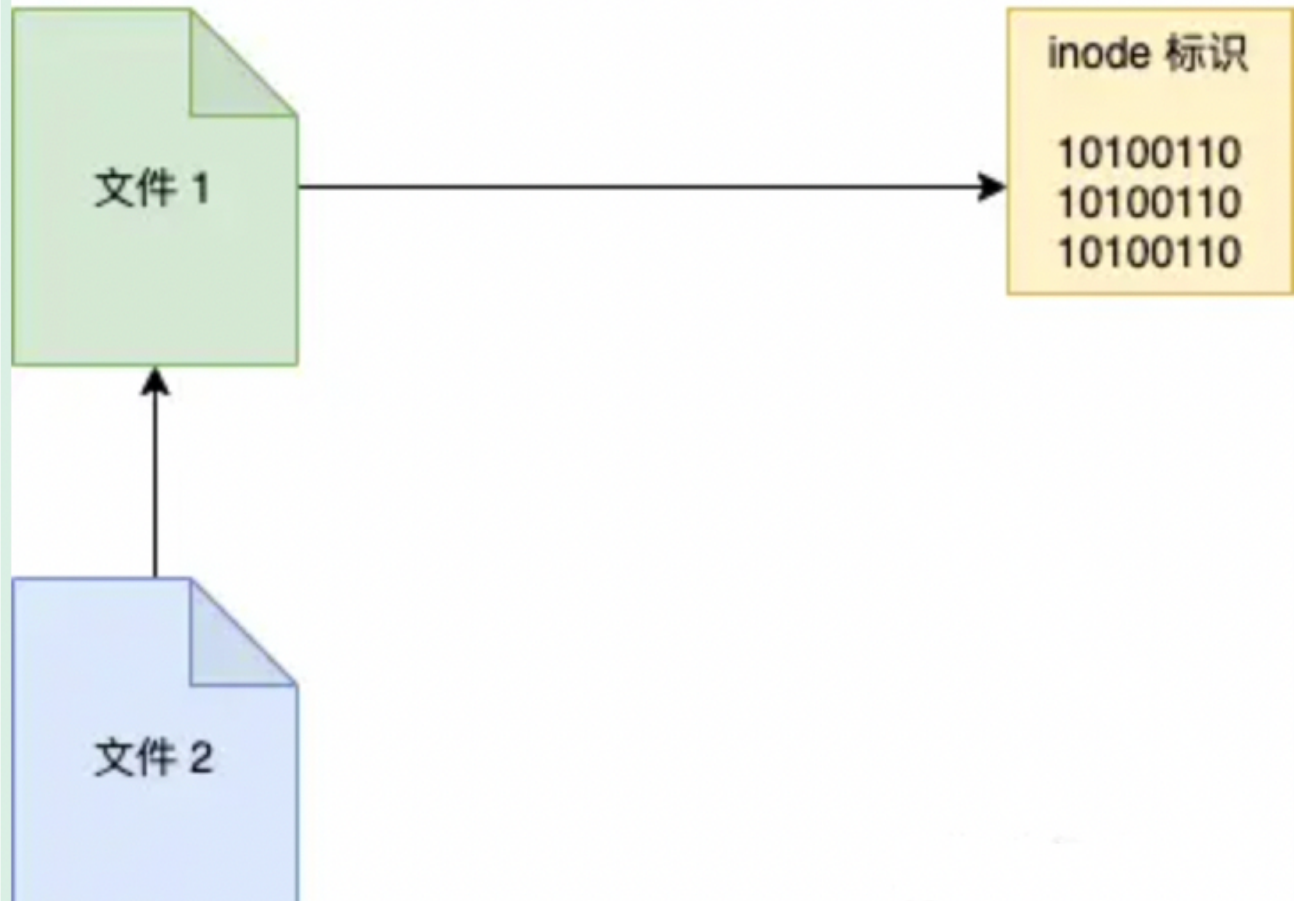
类似 window 系统的快捷方式

使用方式：

```
1 ln -s file1 file2
```

Plain Text | 复制代码

软链接



其实 `file2` 只是 `file1` 的一个快捷方式，它指向的是 `file1`，所以显示的是 `file1` 的内容，但其实 `file2` 的 `inode` 与 `file1` 并不相同

删除了 `file2` 的话，`file1` 是不会受影响的，但如果删除 `file1` 的话，`file2` 就会变成死链接，因为指向的文件不见了

5.3. 文件查看

常见的文件内容查看有如下：

- `cat` 由第一行开始显示文件内容
- `less` 一页一页的显示文件内容
- `head` 只看头几行
- `tail` 只看尾巴几行

5.3.1. cat

由第一行开始显示文件内容

语法：

▼ Plain Text | 复制代码

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

常见的选项与参数如下：

- -b：列出行号，仅针对非空白行做行号显示，空白行不标行号！
- -n：列印出行号，连同空白行也会有行号，与 -b 的选项不同

5.3.2. less

一页一页翻动，以下实例输出/etc/man.config文件的内容：

▼ Plain Text | 复制代码

```
1 [root@www ~]# less /etc/man.config
2 #
3 # Generated automatically from man.conf.in by the
4 # configure script.
5 #
6 # man.conf from man-1.6d
7 ....(中间省略)....
8 :    <== 这里可以等待你输入命令！
```

less运行时可以输入的命令有：

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

5.3.3. head

取出文件前面几行

语法：

▼ Plain Text 复制代码

```
1 head [-n number] 文件
```

选项与参数：

- -n：后面接数字，代表显示几行的意思

▼ Plain Text 复制代码

```
1 [root@www ~]# head /etc/man.config
```

5.3.4. tail

取出文件后面几行

语法：

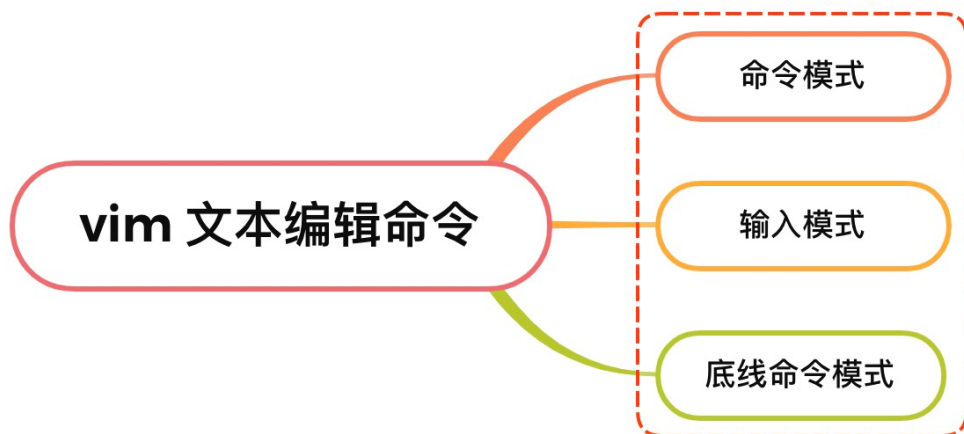
▼ Plain Text 复制代码

```
1 tail [-n number] 文件
```

选项与参数：

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

6. 说说 linux 系统下 文本编辑常用的命令有哪些？



6.1. 是什么

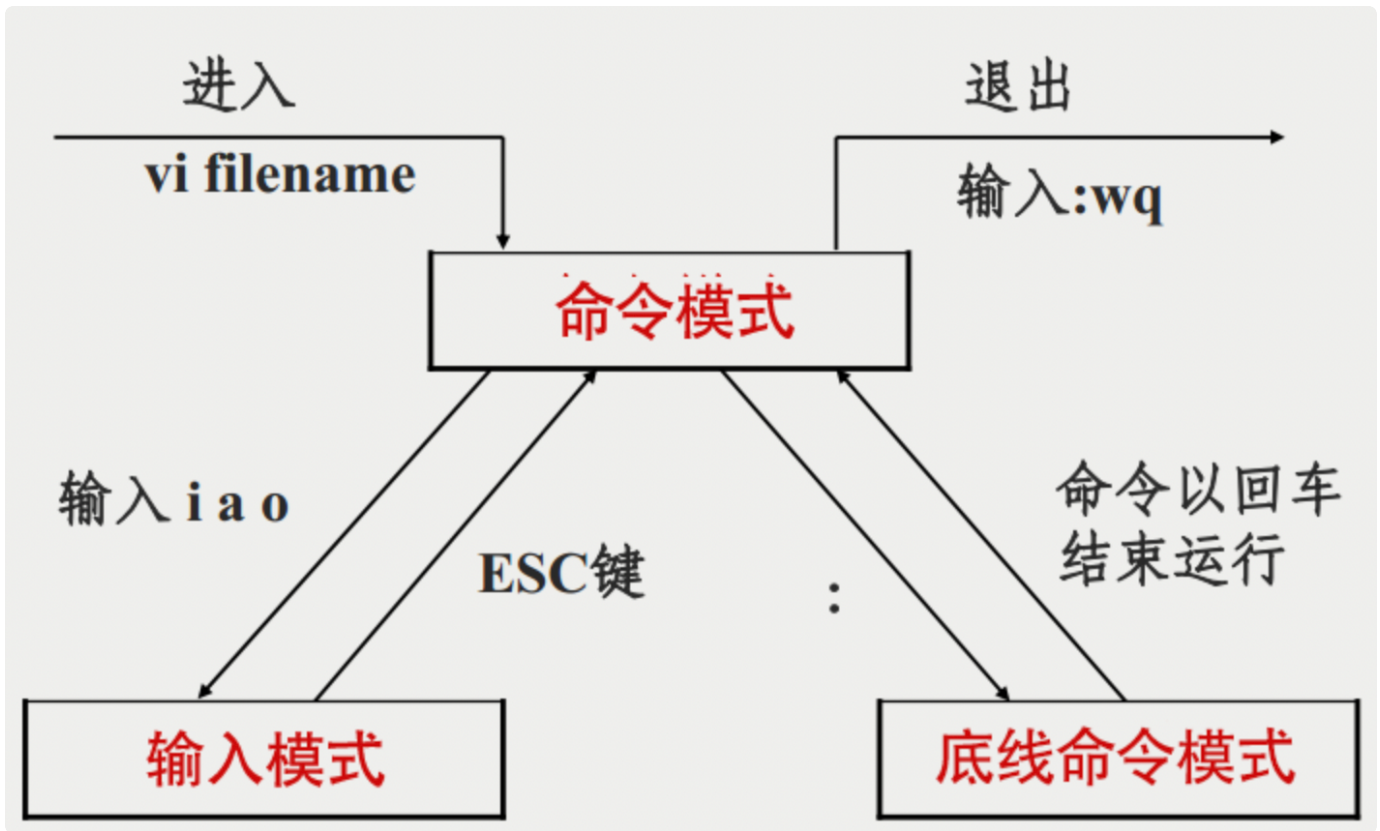
`Vim` 是从 `vi` 发展出来的一个文本编辑器，代码补全、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用。

简单的来说，`vi` 是老式的字处理器，不过功能已经很齐全了，但是还是有可以进步的地方而 `vim` 可以说是程序开发者的一项很好用的工具

6.2. 使用

基本上 `vi/vim` 共分为三种模式，分别是：

- 命令模式 (Command mode)
- 输入模式 (Insert mode)
- 底线命令模式 (Last line mode)



6.2.1. 命令模式

Vim 的默认模式，在这个模式下，你不能输入文本，但是可以让我们在文本间移动，删除一行文本，复制黏贴文本，跳转到指定行，撤销操作，等等

6.2.1.1. 移动光标

常用的命令如下：

- h 向左移动一个字符
- j 向下移动一个字符
- k 向上移动一个字符
- i 向右移动一个字符

或者使用方向键进行控制

如果想要向下移动 **n** 行，可通过使用 "nj" 或 "n↓" 的组合按键

6.2.1.2. 搜索

常见的命令如下：

- /word：向光标之下寻找一个名称为 word 的字符

- ?word: 向光标之上寻找一个字符串名称为 word 的字符串
- n: 代表重复前一个搜寻的动作, 即再次执行上一次的操作
- N: 反向进行前一个搜索动作

6.2.1.3. 删除、复制、粘贴

常用的命令如下:

- x: 向后删除一个字符
- X: 向前删除一个字符
- nc: n 为数字, 连续向后删除 n 个字符
- dd: 删除光标所在的那一整行
- d0: 删除光标所在处, 到该行的最前面一个字符
- d\$删除光标所在处, 到该行的最后一个字符
- ndd: 除光标所在的向下 n 行
- yy: 复制光标所在的那一行
- y0: 复制光标所在的那个字符到该行行首的所有数据
- y\$: 复制光标所在的那个字符到该行行尾的所有数据
- p: 已复制的数据在光标下一行贴上
- P: 已复制的数据在光标上一行贴上
- nc: 重复删除n行数据

6.2.2. 输入模式

命令模式通过输入大小写 **i**、**a**、**o** 可以切换到输入模式, 如下:

- i: 从目前光标所在处输入
- I: 在目前所在行的第一个非空格符处开始输入
- a: 从目前光标所在的下一个字符处开始输入
- A: 从光标所在行的最后一个字符处开始输入
- o: 在目前光标所在的下一行处输入新的一行
- O: 目前光标所在的上一行处输入新的一行

输入模式我们熟悉的文本编辑器的模式, 就是可以输入任何你想输入的内容

如果想从插入模式回到命令模式, 使用按下键盘左上角的 **ESC** 键

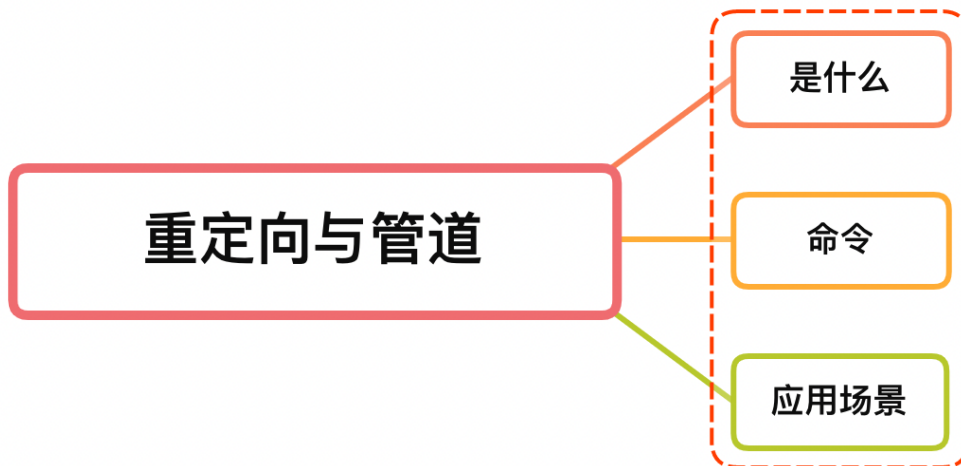
6.2.3. 底线命令模式

这个模式下可以运行一些命令例如“退出”，“保存”，等动作，为了进入底线命令模式，首先要进入命令模式，再按下冒号键：

常见的命令如下：

- w: 将编辑的数据写入硬盘档案中
- w!: 若文件属性为『只读』时，强制写入该档案
- q: 未修改，直接退出
- q!: 修改过但不存储
- wq: 储存后离开

7. 说说你对输入输出重定向和管道的理解？ 应用场景？

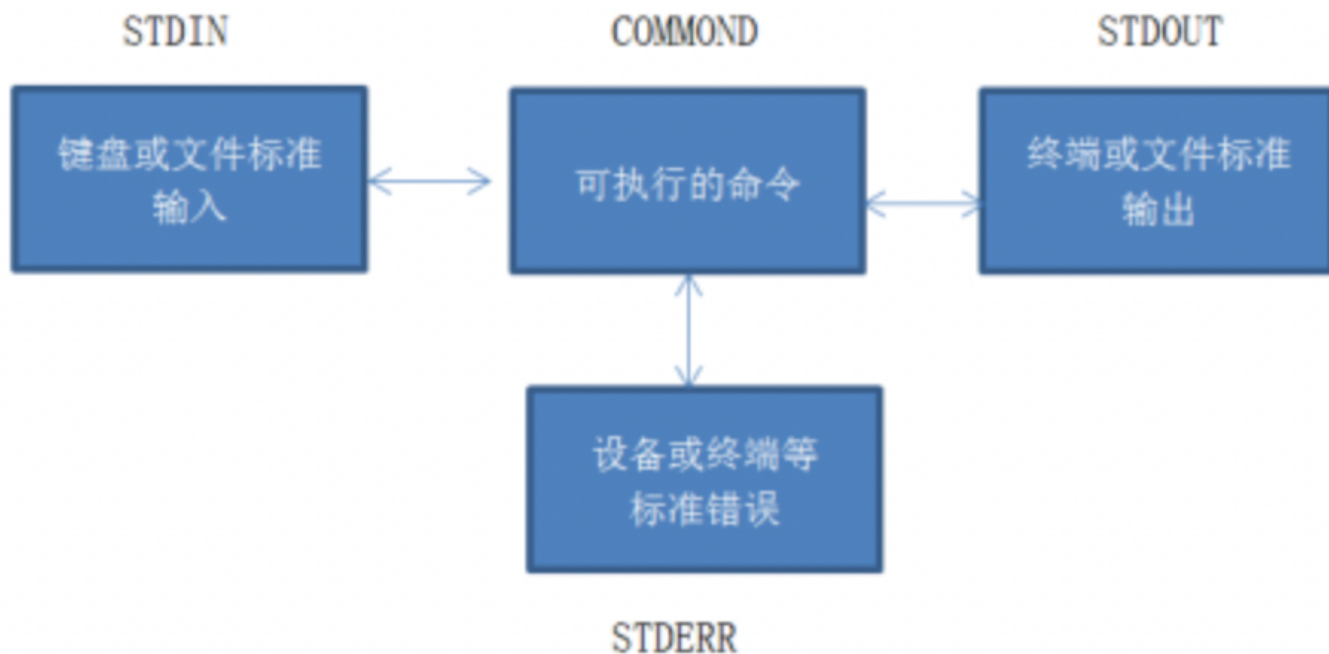


7.1. 是什么

linux 中有三种标准输入输出，分别是 `STDIN`，`STDOUT`，`STDERR`，对应的数字是0、1、2：

- `STDIN` 是标准输入，默认从键盘读取信息
- `STDOUT` 是标准输出，默认将输出结果输出至终端
- `STDERR` 是标准错误，默认将输出结果输出至终端

对于任何 linux 命令的执行会有下面的过程：



一条命令的执行需要键盘等的标准输入，命令的执行和正确或错误，其中的每一个双向箭头就是一个通道，所以数据流可以流入到文件端（**重定向或管道**）

简单来讲，重定向就是把本来要显示在终端的命令结果，输送到别的地方，分成：

- 输入重定向：流出到屏幕如果命令所需的输入不是来自键盘，而是来自指定的文件
- 输出重定向：命令的输出可以不显示在屏幕，而是写在指定的文件中

管道就是把两个命令连接起来使用，一个命令的输出作为另一个命令的输入

两者的区别在于：

- 管道触发两个子进程，执行 | 两边的程序；而重定向是在一个进程内执行。
- 管道两边都是shell命令
- 重定向符号的右边只能是Linux文件
- 重定向符号的优先级大于管道

7.2. 命令

重定向常见的命令符号有：

- >： 输出重定向到一个文件或设备 覆盖原来的文件

如果该文件不存在，则新建一个文件

如果该文件已经存在，会把文件内容覆盖

这些操纵不会征用用户的确认

- `>>`：输出重定向到一个文件或设备，但是是追加原来的文件的末尾
- `<`：用于制定命令的输入
- `<<`：从键盘的输入重定向为某个命令的输入

以逐行输入的模式（回车键进行换行）

所有输入的行都将在输入结束字符串之后发送给命令

- `2>` 将一个标准错误输出重定向到一个文件或设备，会覆盖原来的文件
- `2>>` 将一个标准错误输出重定向到一个文件或设备，是追加到原来的文件
- `2>&1`：组合符号，将标准错误输出重定向到标准输出相同的地方

1就是代表标准输出

- `>&` 将一个标准错误输出重定向到一个文件或设备覆盖原来的文件
- `|&` 将一个标准错误管道输出到另一个命令作为输入

7.3. 应用场景

将当前目录的文件输出重定向到 `1.txt` 文件中，并且会清空原有的 `1.txt` 的内容



Plain Text

复制代码

```
1  ls -a > 1.txt
```

或者以追加的形式，重定向输入到 `1.txt` 中



Plain Text

复制代码

```
1  ls -a >> 1.txt
```

将标准错误输出到某个文件，可以如下：

▼ Plain Text | 复制代码

```
1 $ touch 2> 2.txt
2 $ cat 2.txt
3 touch: 缺少了文件操作数
4 请尝试执行 "touch --help" 来获取更多信息。
```

通过组合符号将两者结合在一起，无论进程输出的信息是正确还是错误的信息，都会重定向到指定的文件里

▼ Plain Text | 复制代码

```
1 [root@linguanghai home]# abc &> file.txt
2 [root@linguanghai home]# cat file.txt
3 -bash: abc: command not found
```

再者通过管道查询文件内容是否包含想要的信息：

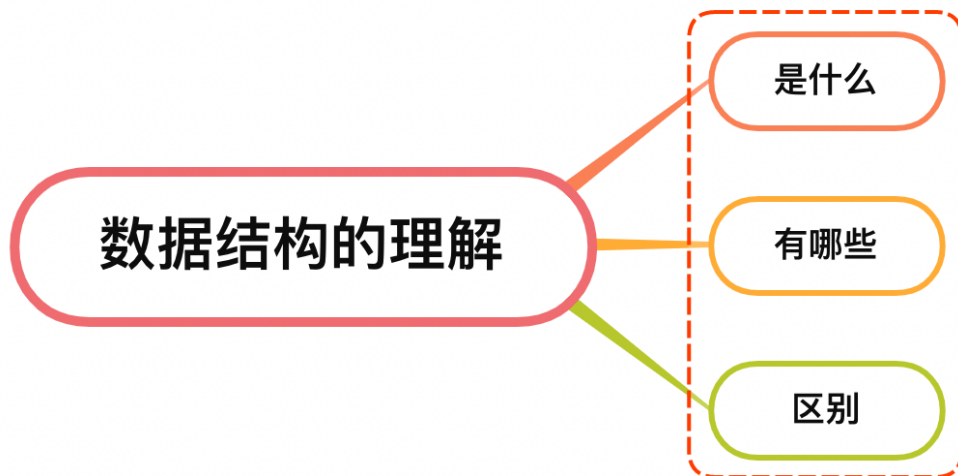
▼ Plain Text | 复制代码

```
1 cat test.txt | grep -n 'xxx'
```

上述 `cat test.txt` 会将 `test.txt` 的内容作为标准输出，然后利用管道，将其作为 `grep -n 'xxx'` 命令的标准输入。

算法面试真题（18题）

1. 说说你对数据结构的理解？有哪些？区别？



1.1. 是什么

数据结构是计算机存储、组织数据的方式，是指相互之间存在一种或多种特定关系的数据元素的集合

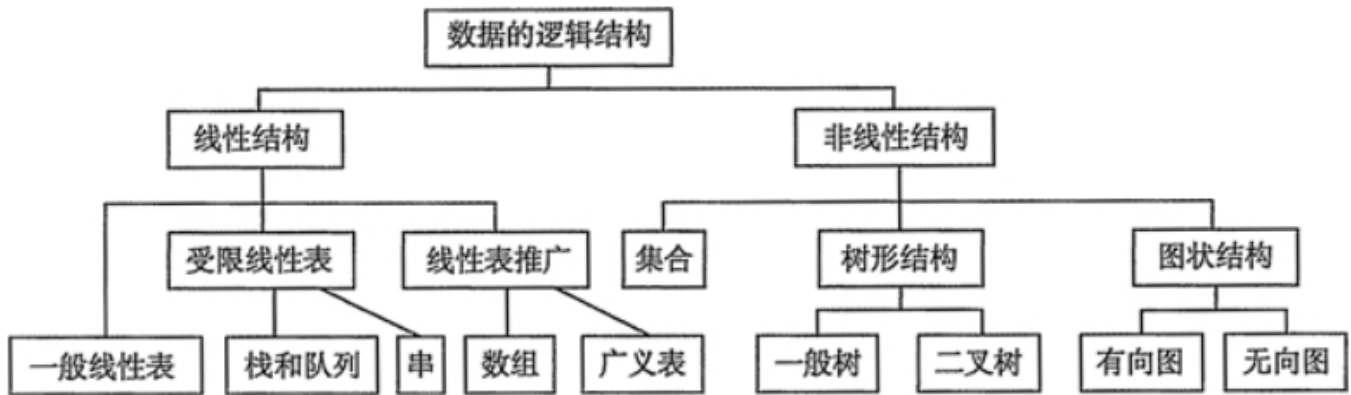
前面讲到，一个程序 = 算法 + 数据结构，数据结构是实现算法的基础，选择合适的数据结构可以带来更高的运行或者存储效率

数据元素相互之间的关系称为结构，根据数据元素之间关系的不同特性，通常有如下四类基本的结构：

- 集合结构：该结构的数据元素间的关系是“属于同一个集合”
- 线性结构：该结构的数据元素之间存在着一对一的关系
- 树型结构：该结构的数据元素之间存在着一对多的关系
- 图形结构：该结构的数据元素之间存在着多对多的关系，也称网状结构

由于数据结构种类太多，逻辑结构可以再分成为：

- 线性结构：有序数据元素的集合，其中数据元素之间的关系是一对一的关系，除了第一个和最后一个数据元素之外，其它数据元素都是首尾相接的
- 非线性结构：各个数据元素不再保持在一个线性序列中，每个数据元素可能与零个或者多个其他数据元素发生关联



1.2. 有哪些

常见的数据结构有如下：

- 数组
- 栈
- 队列
- 链表
- 树
- 图
- 堆
- 散列表

1.2.1. 数组

在程序设计中，为了处理方便，一般情况把具有相同类型的若干变量按有序的形式组织起来，这些按序排列的同类数据元素的集合称为数组

1.2.2. 栈

一种特殊的线性表，只能在某一端插入和删除的特殊线性表，按照先进后出的特性存储数据
先进入的数据被压入栈底，最后的数据在栈顶，需要读数据的时候从栈顶开始弹出数据

1.2.3. 队列

跟栈基本一致，也是一种特殊的线性表，其特性是先进先出，只允许在表的前端进行删除操作，而在表的后端进行插入操作

1.2.4. 链表

是一种物理存储单元上非连续、非顺序的存储结构，数据元素的逻辑顺序是通过链表中的指针链接次序实现的

链表由一系列结点（链表中每一个元素称为结点）组成，结点可以在运行时动态生成

一般情况，每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域

1.2.5. 树

树是典型的非线性结构，在树的结构中，有且仅有一个根结点，该结点没有前驱结点。在树结构中的其他结点都有且仅有一个前驱结点，而且可以有两个以上的后继结点

1.2.6. 图

一种非线性结构。在图结结构中，数据结点一般称为顶点，而边是顶点的有序偶对。如果两个顶点之间存在一条边，那么就表示这两个顶点具有相邻关系

1.2.7. 堆

堆是一种特殊的树形数据结构，每个结点都有一个值，特点是根结点的值最小（或最大），且根结点的两个子树也是一个堆

1.2.8. 散列表

若结构中存在关键字和 K 相等的记录，则必定在 $f(K)$ 的存储位置上，不需比较便可直接取得所查记录

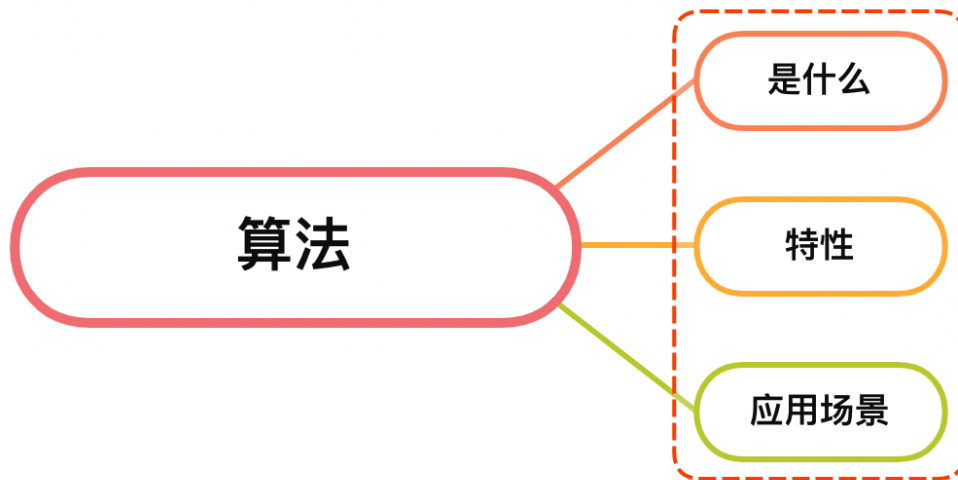
1.3. 区别

上述的数据结构，之前的区别可以分成线性结构和非线性结构：

- 线性结构有：数组、栈、队列、链表等

- 非线性结构有：树、图、堆等

2. 说说你对算法的理解？ 应用场景？



2.1. 是什么

算法（Algorithm）是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制

也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出

如果一个算法有缺陷，或不适合于某个问题，执行这个算法将不会解决这个问题

一个程序=算法+数据结构，数据结构是算法实现的基础，算法总是要依赖于某种数据结构来实现的，两者不可分割

因此，算法的设计和选择要同时结合数据结构，简单地说数据结构的设计就是选择存储方式，如确定问题中的信息是用数组存储还是用普通的变量存储或其他更加复杂的数据结构

针对上述，可以得出一个总结：不同的算法可能用不同的时间、空间或效率来完成同样的任务

2.2. 特性

关于算法的五大特性，有如下：

- 有限性（Finiteness）：一个算法必须保证执行有限步之后结束
- 确切性（Definiteness）：一个算法的每一步骤必须有确切的定义
- 输入（Input）：一个算法有零个或多个输入，以刻画运算对象的初始情况，所谓零个输入是指算法

本身给定了初始条件

- 输出（Output）：一个算法有一个或多个输出。没有输出的算法毫无意义
- 可行性（Effectiveness）：算法中执行的任何计算步骤都是可以被分解为基本的可执行的操作步骤，即每个计算步骤都可以在有限时间内完成（也称之为有效性）

2.3. 应用场景

在前端领域中，数据结构与算法无法不在，例如现在的 `vue` 或者 `react` 项目，实现虚拟 `DOM` 或者 `Fiber` 结构，本质就是一种数据结构，如下一个简单的虚拟 `DOM`：

```
JavaScript | 复制代码
1 {
2   type: 'div',
3   props: {
4     name: 'lucifer'
5   },
6   children: [{
7     type: 'span',
8     props: {},
9     children: []
10  }]
11 }
```

`vue` 与 `react` 都能基于对应的数据结构实现 `diff` 算法，提高了整个框架的性能以及拓展性包括在前端 `javascript` 编译的时候，都会生成对应的抽象语法树 `AST`，其本身不涉及到任何语法，因此你只要编写相应的转义规则，就可以将任何语法转义到任何语法，也是 `babel`，`PostCSS`，`prettier`，`typescript`

除了这些框架或者工具底层用到算法与数据结构之外，日常业务也无处不在，例如实现一个输入框携带联想功能，如下：



如果我们要实现这个功能，则可以使用前缀树，如下：