

# Linux面试真题（7题）

## 1. 说说你对操作系统的理解？核心概念有哪些？



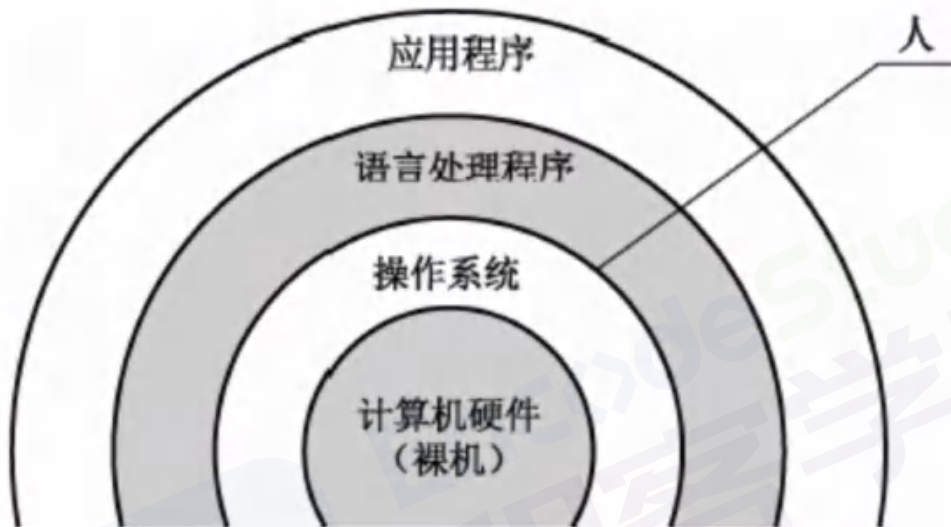
### 1.1. 是什么

操作系统（Operating System，缩写：OS）是一组主管并控制计算机操作、运用和运行硬件、软件资源和提供公共服务来组织用户交互的相互关联的系统软件程序，同时也是计算机系统的内核与基石

简单来讲，操作系统就是一种复杂的软件，相当于软件管家

操作系统需要处理如管理与配置内存、决定系统资源供需的优先次序、控制输入与输出设备、操作网络与管理文件系统等基本事务，

操作系统的类型非常多样，不同机器安装的操作系统可从简单到复杂，可从移动电话的嵌入式系统到超级电脑的大型操作系统，在计算机与用户之间起接口的作用，如下图：



许多操作系统制造者对它涵盖范畴的定义也不尽一致，例如有些操作系统集成了图形用户界面，而有些仅使用命令行界面，将图形用户界面视为一种非必要的应用程序

## 1.2. 核心概念

操作系统的核心概念都是对具体物理硬件的抽象，主要有如下：

- 进程（线程）：进程（线程）是操作系统对CPU的抽象
- 虚拟内存（地址空间）：虚拟内存是操作系统对物理内存的抽象
- 文件：文件是操作系统对物理磁盘的抽象
- shell：它是一个程序，可从键盘获取命令并将其提供给操作系统以执行。
- GUI：是一种用户界面，允许用户通过图形图标和音频指示符与电子设备进行交互
- 计算机架构(computer architecture)：在计算机工程中，计算机体系结构是描述计算机系统功能，组织和实现的一组规则和方法。它主要包括指令集、内存管理、I/O 和总线结构
- 多处理系统(Computer multitasking)：是指计算机同时运行多个程序的能力
- 程序计数器(Program counter)：程序计数器 是一个 CPU 中的寄存器，用于指示计算机在其程序序列中的位置
- 多线程(multithreading)：是指从软件或者硬件上实现多个线程并发执行的技术
- CPU 核心(core)：它是 CPU 的大脑，它接收指令，并执行计算或运算以满足这些指令。一个 CPU 可以有多个内核
- 图形处理器(Graphics Processing Unit)：又称显示核心、视觉处理器、显示芯片或绘图芯片
- 缓存命中(cache hit)：当应用程序或软件请求数据时，会首先发生缓存命中
- RAM(Random Access Memory)：随机存取存储器，也叫主存，是与 CPU 直接交换数据的内部存

## 存储器

- ROM (Read Only Memory): 只读存储器是一种半导体存储器，其特性是一旦存储数据就无法改变或删除
- 虚拟地址(virtual memory): 虚拟内存是计算机系统内存管理的一种机制
- 驱动程序(device driver): 设备驱动程序，简称驱动程序 (driver)，是一个允许高级别电脑软件与硬件交互的程序
- USB(Universal Serial Bus): 是连接计算机系统与外部设备的一种串口总线标准，也是一种输入输出接口的技术规范
- 地址空间(address space): 地址空间是内存中可供程序或进程使用的有效地址范围
- 进程间通信(interprocess communication): 指至少两个进程或线程间传送数据或信号的一些技术或方法
- 目录(directory): 在计算机或相关设备中，一个目录或文件夹就是一个装有数字文件系统的虚拟容器
- 路径(path name): 路径是一种电脑文件或目录的名称的通用表现形式，它指向文件系统上的一个唯一位置。
- 根目录(root directory): 根目录指的就是计算机系统中的顶层目录，比如 Windows 中的 C 盘和 D 盘，Linux 中的 /
- 工作目录(Working directory): 它是一个计算机用语。用户在操作系统内所在的目录，用户可在此目录之下，用相对文件名访问文件。
- 文件描述符(file descriptor): 文件描述符是计算机科学中的一个术语，是一个用于表述指向文件的引用的抽象化概念
- 客户端(clients): 客户端是访问服务器提供的服务的计算机硬件或软件。
- 服务端(servers): 在计算中，服务器是为其他程序或设备提供功能的计算机程序或设备

## 1.3. 总结

- 操作系统是管理计算机硬件与软件资源的程序，是计算机的基石
- 操作系统本质上是一个运行在计算机上的软件程序，用于管理计算机硬件和软件资源
- 操作系统存在屏蔽了硬件层的复杂性。操作系统就像是硬件使用的负责人，统筹着各种相关事项
- 操作系统的内核 (Kernel) 是操作系统的核心部分，它负责系统的内存管理，硬件设备的管理，文件系统的管理以及应用程序的管理。内核是连接应用程序和硬件的桥梁，决定着系统的性能和稳定性

## 2. 说说什么是进程？什么是线程？区别？



### 2.1. 进程

操作系统中最核心的概念就是进程，进程是对正在运行中的程序的一个抽象，是系统进行资源分配和调度的基本单位

操作系统的其他所有内容都是围绕着进程展开的，负责执行这些任务的是 CPU

活动监视器 过去 12 小时内使用过的应用程序						
CPU 内存 能耗 磁盘 网络 搜索						
App 名称	对能耗的影响	12 小时...	App 小憩	防止睡眠	用户	
> Photoshop CC	12.4	63.08	是	否	linguanghai	
> Google Chrome	0.8	8.30	否	否	linguanghai	
> Code	0.1	6.17	否	否	linguanghai	
> 微信	1.2	2.48	否	否	linguanghai	
聚焦	0.5	0.75	-	-	-	
> Typora	0.2	0.16	是	否	linguanghai	
> 邮件	0.1	0.12	是	否	linguanghai	
> 访达	0.1	0.08	否	否	linguanghai	
> CCXProcess	0.0	0.03	否	否	linguanghai	
> Scroll Reverser	0.5	0.03	否	否	linguanghai	
终端	0.0	0.01	是	否	linguanghai	

进程是一种抽象的概念，从来没有统一的标准定义看，一般由程序、数据集合和进程控制块三部分组成：

- 程序用于描述进程要完成的功能，是控制进程执行的指令集
- 数据集合是程序在执行时所需要的数据和工作区
- 程序控制块，包含进程的描述信息和控制信息，是进程存在的唯一标志

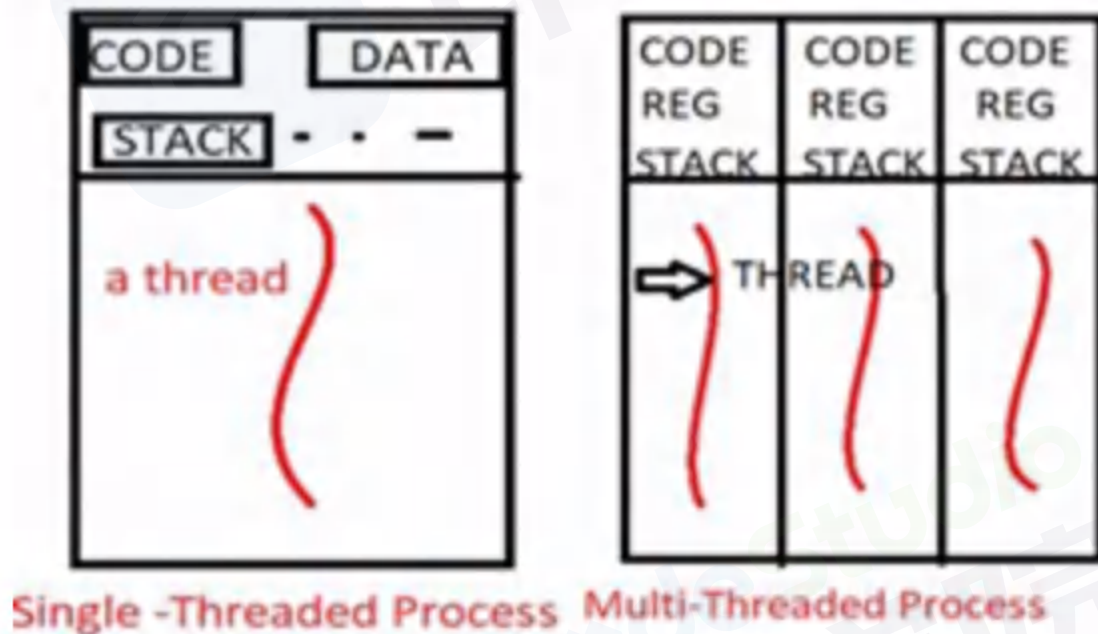
## 2.2. 线程

**线程**（thread）是操作系统能够进行**运算调度**的最小单位，其是进程中的一个执行任务（控制单元），负责当前进程中程序的执行

一个进程至少有一个线程，一个进程可以运行多个线程，这些线程共享同一块内存，线程之间可以共享对象、资源，如果有冲突或需要协同，还可以随时沟通以解决冲突或保持同步

举个例子，假设你经营着一家物业管理公司。最初，业务量很小，事事都需要你亲力亲为。给老张家修完暖气管道，立马再去老李家换电灯泡——这叫单线程，所有的工作都得顺序执行

后来业务拓展了，你雇佣了几个工人，这样，你的物业公司就可以同时为多户人家提供服务了——这叫多线程，你是主线程



但实际上，并不是线程越多，进程的工作效率越高，这是因为在一个进程内，不管你创建了多少线程，它们总是被限定在一颗 CPU 内，或者多核 CPU 的一个核内

这意味着，多线程在宏观上是并行的，在微观上则是分时切换串行的，多线程编程无法充分发挥多核计算资源的优势

这导致使用多线程做任务并行处理时，线程数量超过一定数值后，线程越多速度反倒越慢的原因

## 2.3. 区别

- **本质区别：**进程是操作系统资源分配的基本单位，而线程是任务调度和执行的基本单位
- **在开销方面：**每个进程都有独立的代码和数据空间（程序上下文），程序之间的切换会有较大的开

销；线程可以看做轻量级的进程，同一类线程共享代码和数据空间，每个线程都有自己独立的运行栈和程序计数器（PC），线程之间切换的开销小

- **所处环境**：在操作系统中能同时运行多个进程（程序）；而在同一个进程（程序）中有多个线程同时执行（通过CPU调度，在每个时间片中只有一个线程执行）
- **内存分配方面**：系统在运行的时候会为每个进程分配不同的内存空间；而对线程而言，除了CPU外，系统不会为线程分配内存（线程所使用的资源来自其所属进程的资源），线程组之间只能共享资源
- **包含关系**：没有线程的进程可以看做是单线程的，如果一个进程内有多个线程，则执行过程不是一条线的，而是多条线（线程）共同完成的；线程是进程的一部分，所以线程也被称为轻权进程或者轻量级进程

举个例子：进程=火车，线程=车厢

- 线程在进程下行进（单纯的车厢无法运行）
- 一个进程可以包含多个线程（一辆火车可以有多个车厢）
- 不同进程间数据很难共享（一辆火车上的乘客很难换到另外一辆火车，比如站点换乘）
- 同一进程下不同线程间数据很易共享（A车厢换到B车厢很容易）
- 进程要比线程消耗更多的计算机资源（采用多列火车相比多个车厢更耗资源）
- 进程间不会相互影响，一个线程挂掉将导致整个进程挂掉（一列火车不会影响到另外一列火车，但是如果一列火车上中间的一节车厢着火了，将影响到所有车厢）

### 3. 说说你对 shell 的理解？常见的命令？





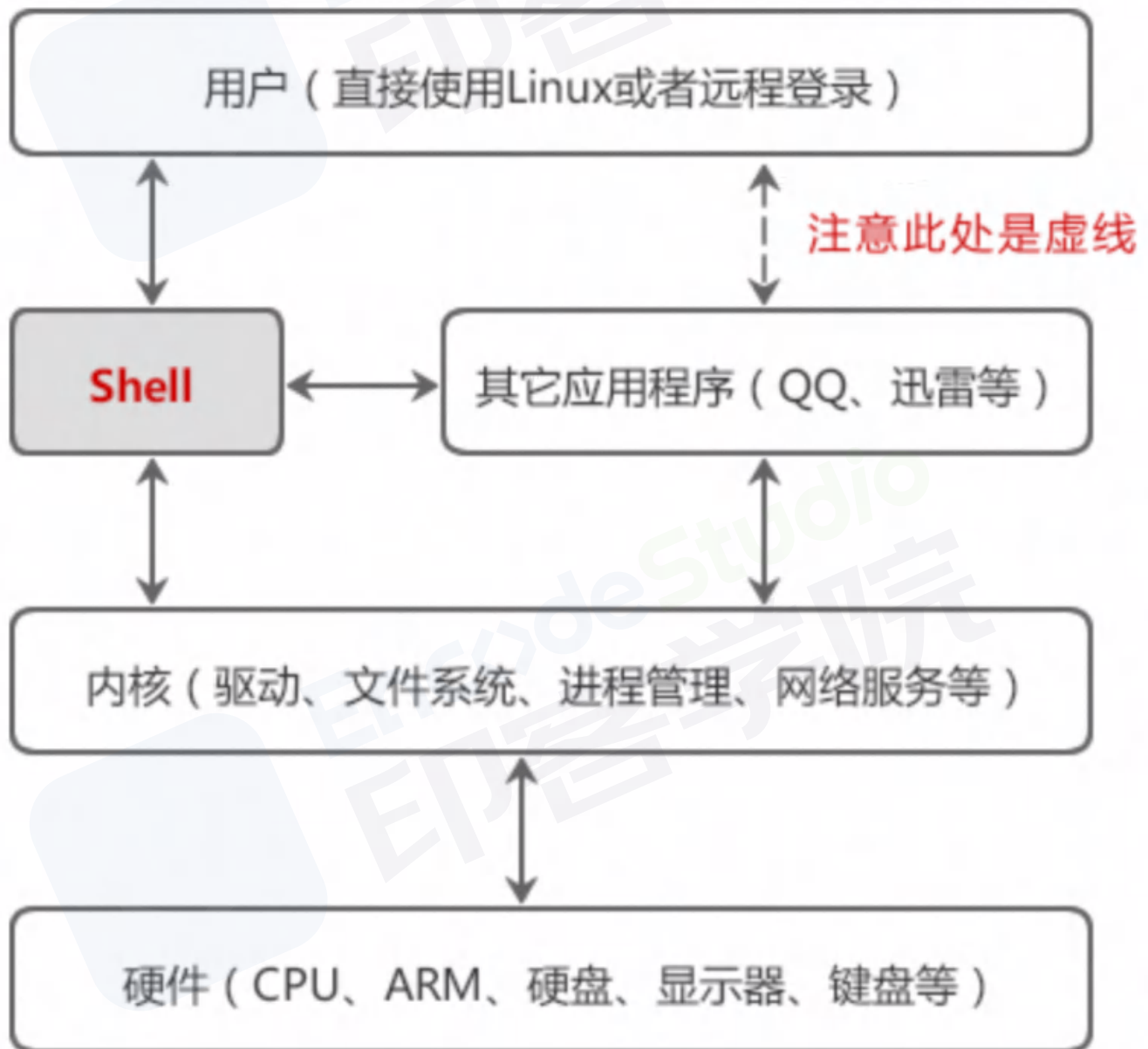
## 3.1. 是什么

`Shell` 是一个由 `c` 语言编写的应用程序，它是用户使用 Linux 的桥梁。`Shell` 既是一种命令语言，又是一种程序设计语言

它连接了用户和 `Linux` 内核，让用户能够更加高效、安全、低成本地使用 `Linux` 内核

其本身并不是内核的一部分，它只是站在内核的基础上编写的一个应用程序，它和 QQ、微信等其它软件没有什么区别，特殊的地方就是开机立马启动，并呈现在用户面前

主要作用是接收用户输入的命令，并对命令进行处理，处理完毕后再将结果反馈给用户，比如输出到显示器、写入到文件等，同样能够调用和组织其他的应用程序，相当于一个领导者的身份，如下图：



那么 `shell` 脚本就是多个 `Shell` 命令的组合并通过 `if` 条件分支控制或循环来组合运算，实现一些复杂功能，文件后缀名为 `.sh`

常用的 `ls` 命令，它本身也是一个 `Shell` 脚本，通过执行这个 `Shell` 脚本可以列举当前目录下的文件列表，如下创建一个 `hello.sh` 脚本

▼ Shell 复制代码

```
1  #!/bin/bash
2
3  # 执行的命令主体
4  ls
5  echo "hello world"
```

- `#!/bin/bash`：指定脚本要使用的 Shell 类型为 Bash
- `ls`、`echo`：脚本文件的内容，表明我们执行 `hello.sh` 脚本时会列举出当前目录的文件列表并且会向控制台打印 `hello world`

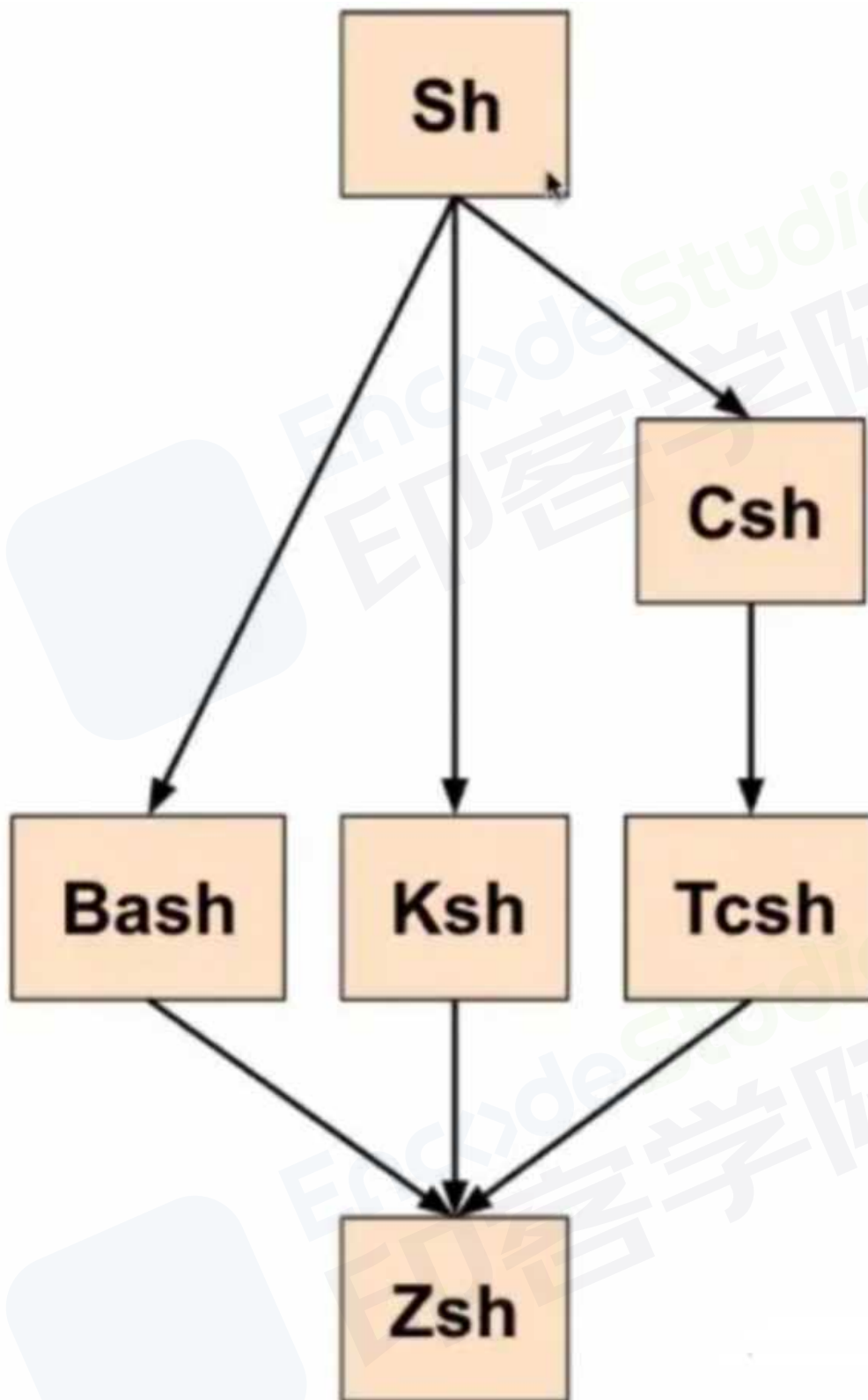
执行方式为 `.hello.zsh`

## 3.2. 种类

`Linux` 的 `Shell` 种类众多，只要能给用户提​​供命令行环境的程序，常见的有：

- Bourne Shell (`sh`)，是目前所有 Shell 的祖先，被安装在几乎所有发源于 Unix 的操作系统上
- Bourne Again shell (`bash`)，是 `sh` 的一个进阶版本，比 `sh` 更优秀，`bash` 是目前大多数 Linux 发行版以及 macOS 操作系统的默认 Shell
- C Shell (`csh`)，它的语法类似 C 语言
- TENEX C Shell (`tcsh`)，它是 `csh` 的优化版本
- Korn shell (`ksh`)，一般在收费的 Unix 版本上比较多见
- Z Shell (`zsh`)，它是一种比较新近的 Shell，集 `bash`、`ksh` 和 `tcsh` 各家之大成





关于 `Shell` 的几个常见命令：

- `ls`：查看文件
- `cd`：切换工作目录
- `pwd`：显示用户当前目录

- mkdir: 创建目录
- cp: 拷贝
- rm: 删除
- mv: 移动
- du: 显示目录所占用的磁盘空间

### 3.3. 命令

Shell 并不是简单的堆砌命令，我们还可以在 Shell 中编程，这和使用 C++、C#、Java、Python 等常见的编程语言并没有什么两样。

Shell 虽然没有 C++、Java、Python 等强大，但也支持了基本的编程元素，例如：

- if...else 选择结构，case...in 开关语句，for、while、until 循环；
- 变量、数组、字符串、注释、加减乘除、逻辑运算等概念；
- 函数，包括用户自定义的函数和内置函数（例如 printf、export、eval 等）

下面以 bash 为例简单了解一下 shell 的基本使用

#### 3.3.1. 变量

Bash 没有数据类型的概念，所有的变量值都是字符串，可以保存一个数字、一个字符、一个字符串等等

同时无需提前声明变量，给变量赋值会直接创建变量

访问变量的语法形式为：\${var} 和 \$var 。

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，所以推荐加花括号。

▼

Bash | 复制代码

```
1 word="hello"
2 echo ${word}
3 # Output: hello
```

#### 3.3.2. 条件控制

跟其它程序设计语言一样，Bash 中的条件语句让我们可以决定一个操作是否被执行。结果取决于一个包在 [[ ]] 里的表达式

跟其他语言一样，使用 `if...else` 进行表达，如果中括号里的表达式为真，那么 `then` 和 `fi` 之间的代码会被执行，如果则 `else` 和 `fi` 之间的代码会被执行

```
1 if [[ 2 -ne 1 ]]; then
2     echo "true"
3 else
4     echo "false"
5 fi
6 # Output: true
```

`fi` 标志着条件代码块的结束

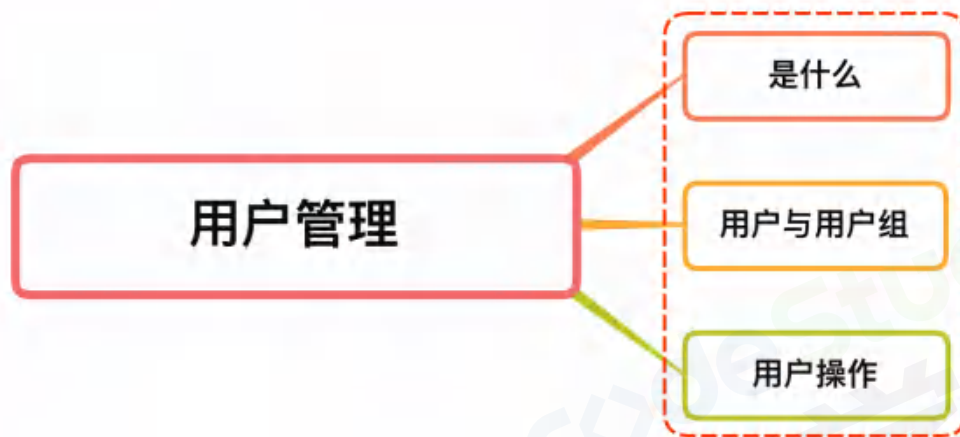
### 3.3.3. 函数

bash 函数定义语法如下：

```
1 [ function ] funname [()] {
2     action;
3     [return int;]
4 }
```

- 函数定义时，`function` 关键字可有可无
- 函数返回值 – `return` 返回函数返回值，返回值类型只能为整数（0–255）。如果不加 `return` 语句，`shell` 默认将以最后一条命令的运行结果，作为函数返回值
- 函数返回值在调用该函数后通过 `$?` 来获得
- 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

## 4. 说说你对 linux 用户管理的理解？相关的命令有哪些？



## 4.1. 是什么

Linux是一个多用户的系统，允许使用者在系统上通过规划不同类型、不同层级的用户，并公平地分配系统资源与工作环境

而与 Windows 系统最大的不同，Linux 允许不同的用户同时登录主机，同时使用主机的资源

既然是多用户的系统，那么最常见的问题就是权限，不同的用户对于不同的文件都应该有各自的权限

例如，小 A 希望个人文件不被其他用户读取，而如果不对文件进行权限设置，共享了主机资源的小 B 也可以读取小 A 的个人文件，这是不合理的

这里面涉及到用户与用户组的概念

## 4.2. 用户与用户组

Linux 以“用户与用户组”的概念，建立用户与文件权限之间的联系，保证系统能够充分考虑每个用户的隐私保护，很大程度上保障了 Linux 作为多用户系统的可行性

从文件权限的角度出发，“用户与用户组”引申为三个具体的对象：

- 文件所有者
- 用户组成员
- 其他人

每一个对象对某一个文件的持有权限是不同的

### 4.2.1. 文件所有者

当一个用户创建了一个文件，这个用户就是这个文件的文件所有者。文件所有者对文件拥有最高权限，同时排他性地拥有该文件

除非文件所有者开放权限，否则其他人无法对文件执行查看、修改等操作

### 4.2.2. 用户组

将“其他用户”区分为用户组成员和其他人后，若文件所有者希望对部分用户开放权限，而对其他人继续保持私有，则只需要将这部分用户与文件所有者划入一个用户组

这样，这部分用户就成了与文件所有者同组的用户组成员。用户可以对用户组成员开放文件权限，用户组成员则具备了查看、修改文件的权限，而对其他无关用户保持私有

例如，团队成员之间保持文件资源共享，但对非团队成员保持私有，这就需要将文件所有者与团队成员用户划分为同一个用户组，再对用户组成员开放权限即可

### 4.2.3. 其他人

既与文件所有者没有任何联系的其他用户

### 4.2.4. 小结

户和用户组的对应关系是：一对一、多对一、一对多或多对多：

- 一对一：某个用户可以是某个组的唯一成员
- 多对一：多个用户可以是某个唯一的组的成员，不归属其它用户组
- 一对多：某个用户可以是多个用户组的成员
- 多对多：多个用户对应多个用户组，并且几个用户可以是归属相同的组

### 4.2.5. 拓展

当我们使用 `ls -l` 的时候，会列出当前目录的文件信息，如下：

Plain Text   复制代码							
1	drwxr-xr-x	3	osmond	osmond	4096	05-16 13:32	nobp

- d：文件类型
- rwxr-xr-x：文件权限
- 3 硬链接数或目录包含的文件数

- osmond：文件所有者
- 4096：文件长度
- 05-16 13:32：文件上次修改的事件和日期
- nobp：文件名

下面主要看看文件权限分析，实际上是由9个字符组成，每3个一组：

- 第一组控制文件**所有者**的访问权限
- 第二组控制所有者**所在用户组**的其他成员的访问权限
- 第三组控制**系统其他用户**的访问权限

- 代表当前没有， `rwX` 对应代表的意思如下：

权限	描述字符	对文件的含义	对目录的含义
读权限	<code>r</code>	可以读取文件的内容	可以列出目录中的文件列表
写权限	<code>w</code>	可以修改或删除文件	可以在该目录中创建或删除文件或子目录
执行权限	<code>x</code>	可以执行该文件	可以使用 <code>cd</code> 命令进入该目录

#### 4.2.6. 用户操作

用户相关的操作有如下：

#### 4.2.7. 新增用户

`useradd` 可以用来创建新用户，简要语法为：

```
1 useradd [options] [username]
```

例如：



添加一个一般用户

▼ Plain Text 复制代码

```
1 # useradd kk //添加用户kk
```

为添加的用户指定相应的用户组

▼ Plain Text 复制代码

```
1 # useradd -g root kk //添加用户kk，并指定用户所在的组为root用户组
```

创建一个系统用户

▼ Plain Text 复制代码

```
1 # useradd -r kk //创建一个系统用户kk
```

为新添加的用户指定/home目录

▼ Plain Text 复制代码

```
1 # useradd-d /home/myf kk //新添加用户kk，其home目录为/home/myf
2 //当用户名kk登录主机时，系统进入的默认目录为/home/myf
```

## 4.3. 设置密码

创建的用户还没有设置登录密码，需要利用 `passwd` 进行密码设置

▼ LaTeX 复制代码

```
1 passwd [options] [username]
```

**option** 参数有如下：

- `-d` 删除密码
- `-f` 强迫用户下次登录时必须修改口令
- `-w` 口令要到期提前警告的天数
- `-k` 更新只能发送在过期之后
- `-l` 停止账号使用
- `-S` 显示密码信息
- `-u` 启用已被停止的账户

- -x 指定口令最长存活期
- -g 修改群组密码
- 指定口令最短存活期
- -i 口令过期后多少天停用账户

例如，修改用户密码

▼ Plain Text 复制代码

```
1 # passwd runoob //设置runoob用户的密码
2 Enter new UNIX password: //输入新密码，输入的密码无回显
3 Retype new UNIX password: //确认密码
4 passwd: password updated successfully
5 #
```

显示账号密码信息

▼ Plain Text 复制代码

```
1 # passwd -S runoob
2 runoob P 05/13/2010 0 99999 7 -1
```

删除用户密码

▼ Plain Text 复制代码

```
1 # passwd -d lx138
2 passwd: password expiry information changed.
```

### 4.3.1. 修改用户

**chage** 命令用来修改与用户密码相关的过期信息，如密码失效日、密码最短保留天数、失效前警告天数等

▼ LaTeX 复制代码

```
1 chage [option] [username]
```

常见的参数有：

- -d: 指定密码最后修改日期
- -E: 密码到期的日期
- -l: 列出用户以及密码的有效期

- -m: 密码能够更改的最小天数
- -M: 密码保持有效的最大天数

### 4.3.2. 删除用户

userdel 命令用来删除用户的相关的所有数据。

```
1 userdel [options] [username]
```

常见的参数有：

- -r: 删除用户登入目录以及目录中所有文件

例如删除用户账号

```
1 # userdel hnlinux
```

用户组相关的操作如下：

### 4.3.3. 新增用户组

**groupadd** 用于创建一个新的工作组，新工作组的信息将被添加到系统文件中

```
1 groupadd [options] [groupname]
```

常见的参数有如下：

- -g: 指定新建工作组的 id;
- -r: 创建系统工作组，系统工作组的组ID小于 500
- -K: 覆盖配置文件 "/ect/login.defs"
- -o: 允许添加组 ID 号不唯一的工作组
- -f,--force: 如果指定的组已经存在，此选项将失明了仅以成功状态退出

例如创建一个新的组，并添加组 ID。



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，这就需要用户使用身份切换的命令：

#### 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`：

```
1 $ sudo ls
```

## 5. 说说 linux系统下 文件操作常用的命令有哪些？



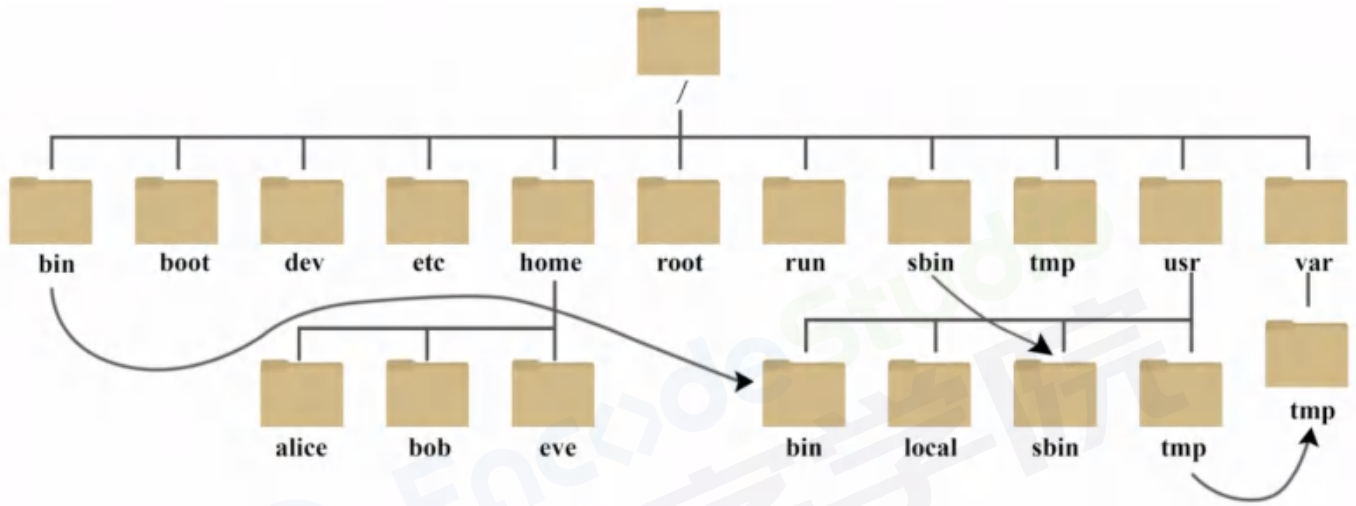
### 5.1. 是什么

`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

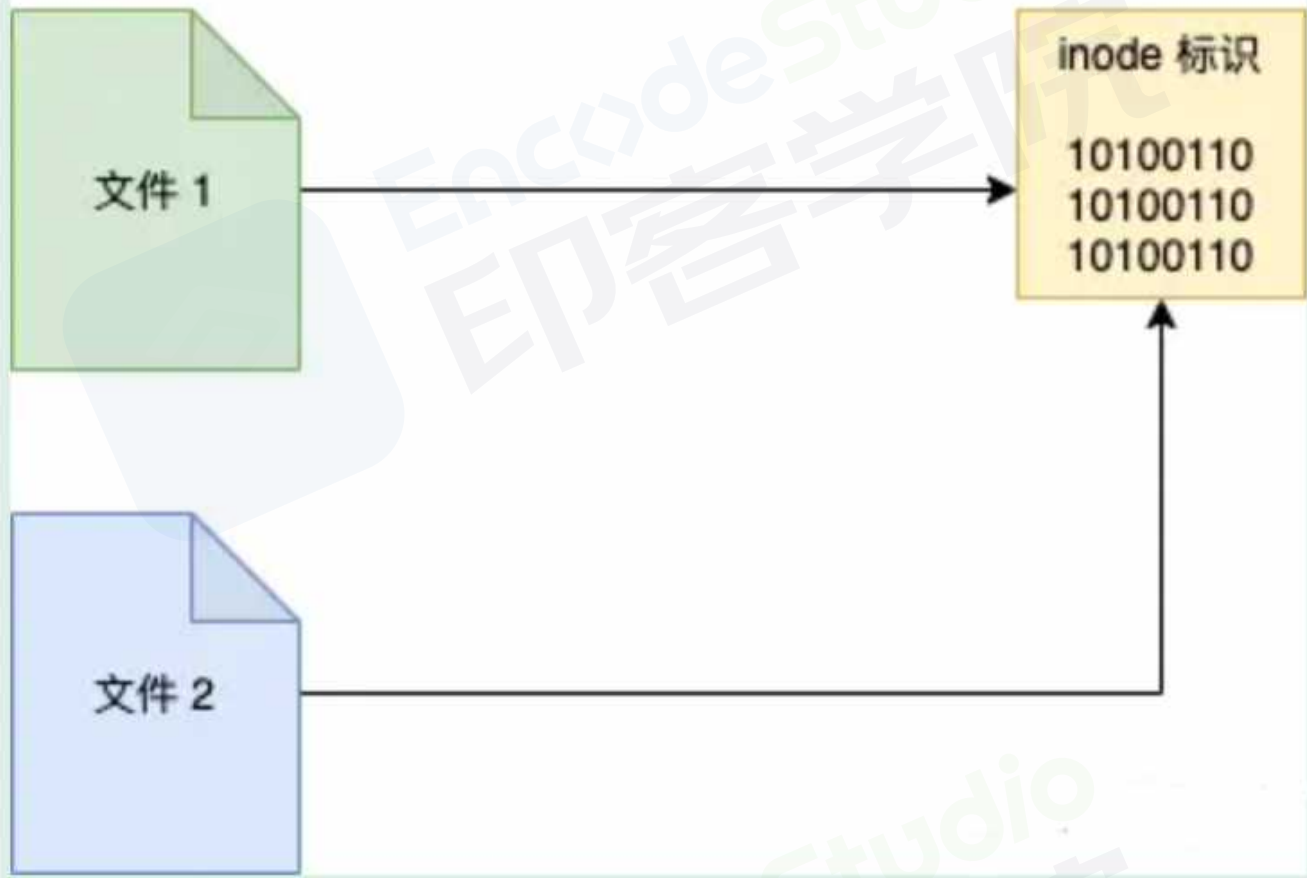
### 5.2.9.1. 硬链接

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。

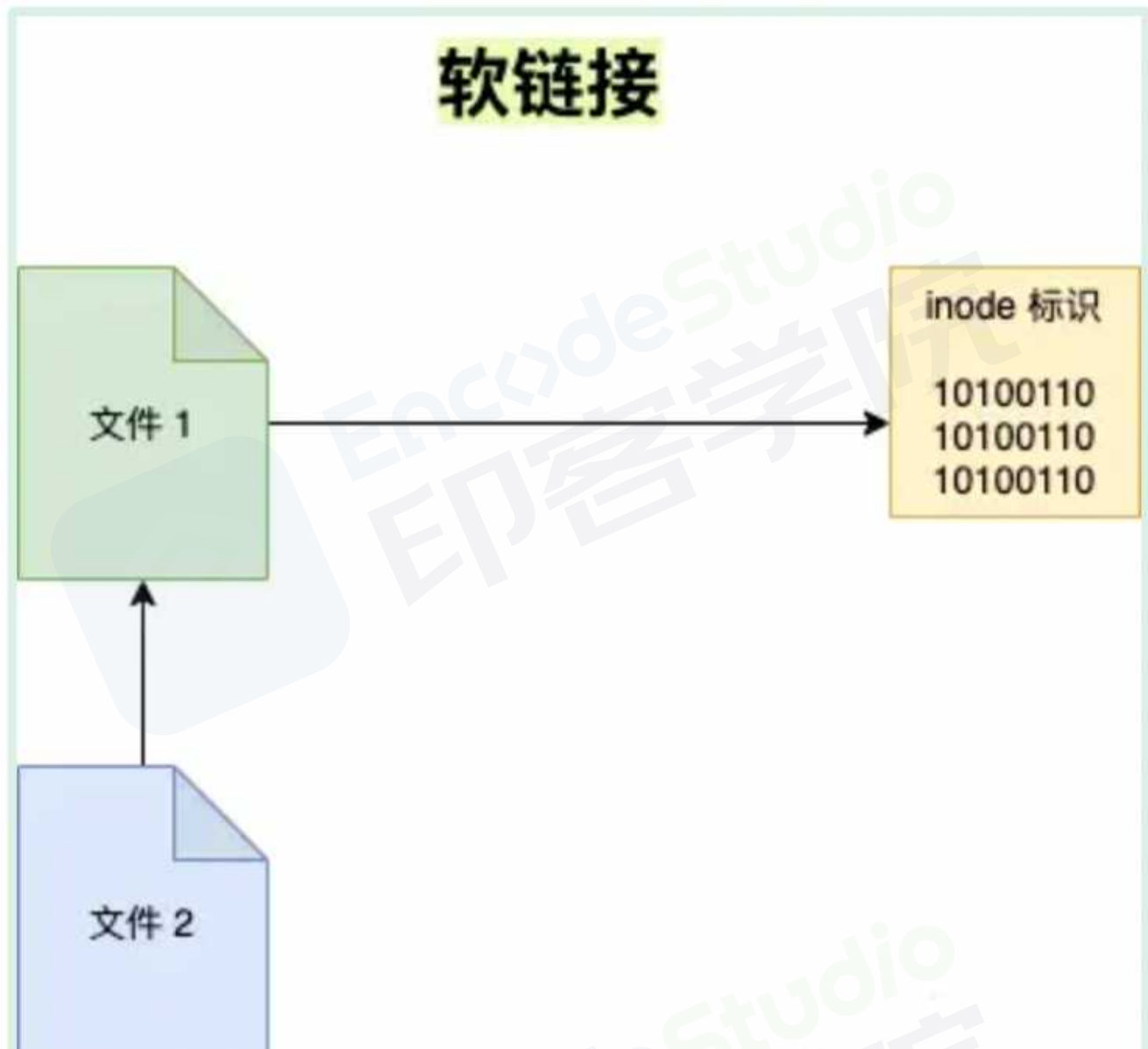
## 5.2.9.2. 软链接

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



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：

```
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：

```
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：

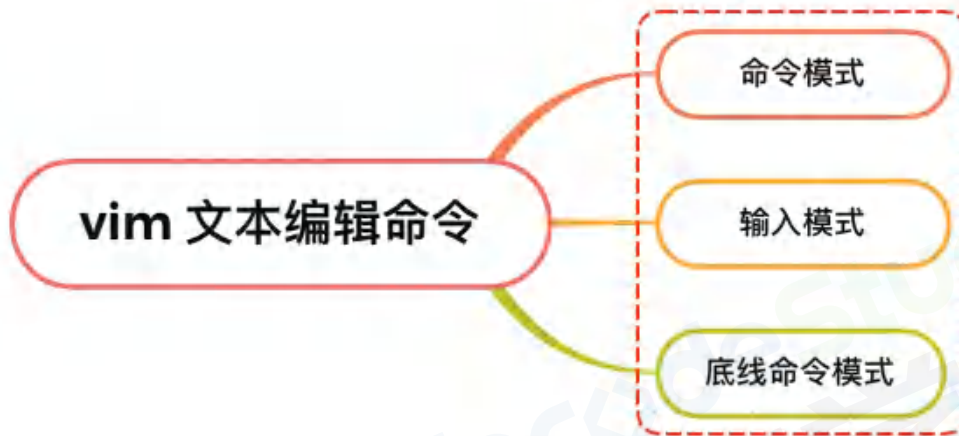
▼ Plain Text | 复制代码

```
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

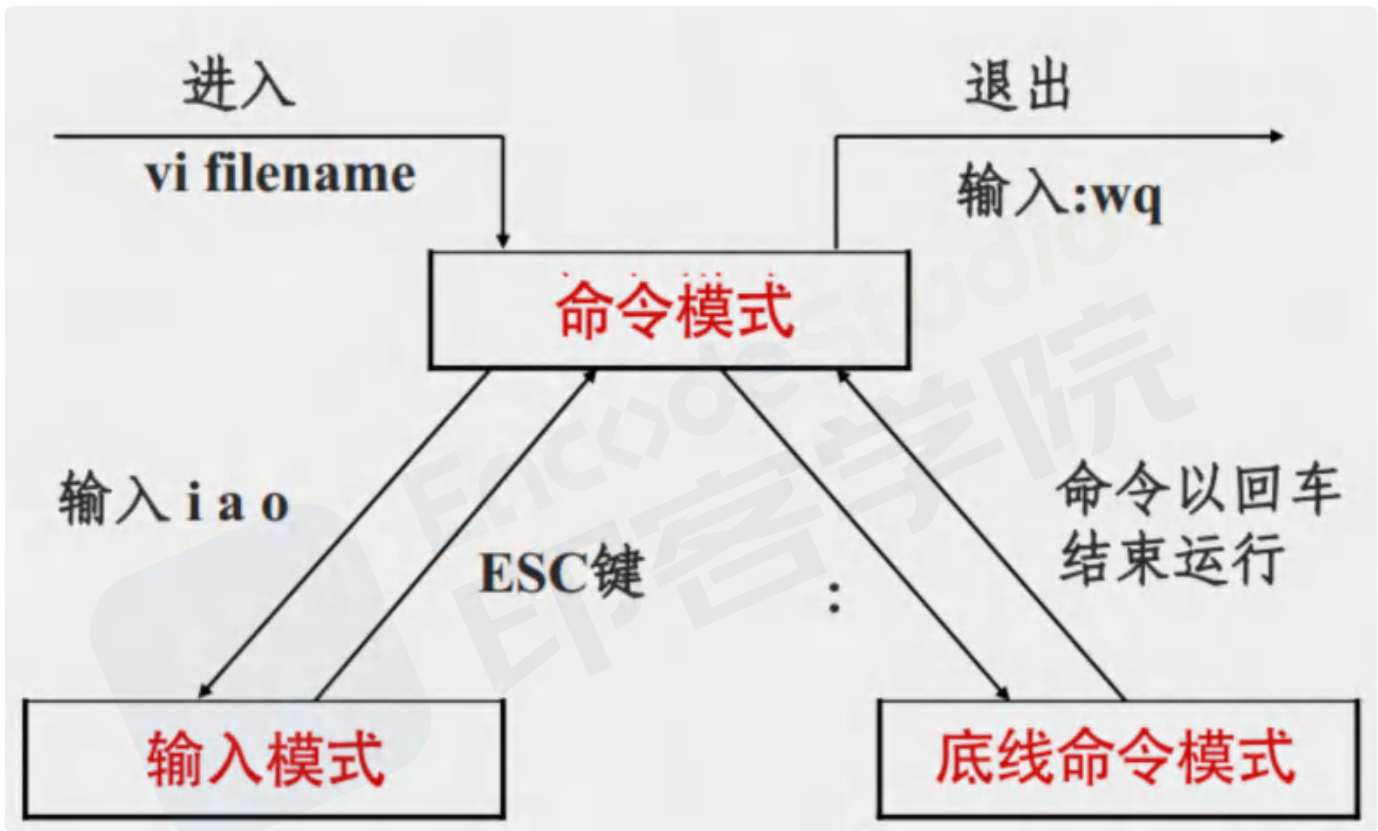
Vim 是从 vi 发展出来的一个文本编辑器，代码补全、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用。

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: 删除游标所在的那一整行
- d0: 删除游标所在处，到该行的最前面一个字符
- d\$删除游标所在处，到该行的最后一个字符
- ndd: 除光标所在的向下 n 行
- yy: 复制游标所在的那一行
- 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

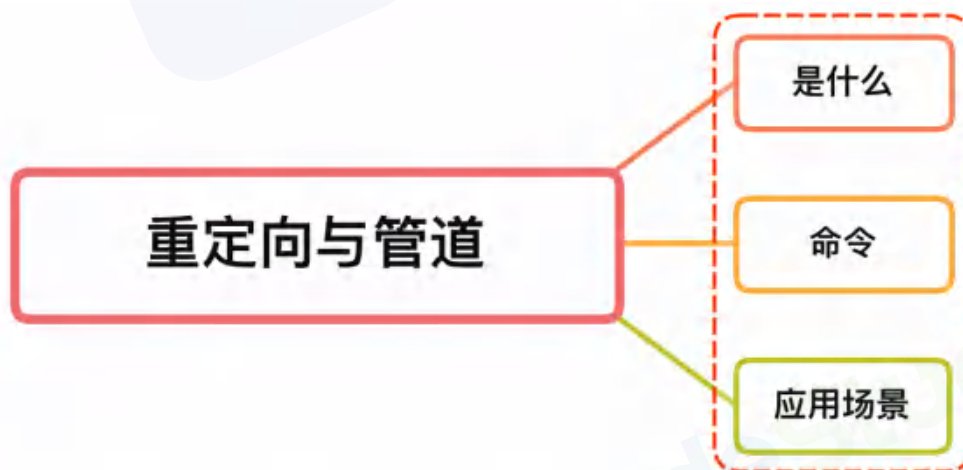
### 6.2.3. 底线命令模式

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？

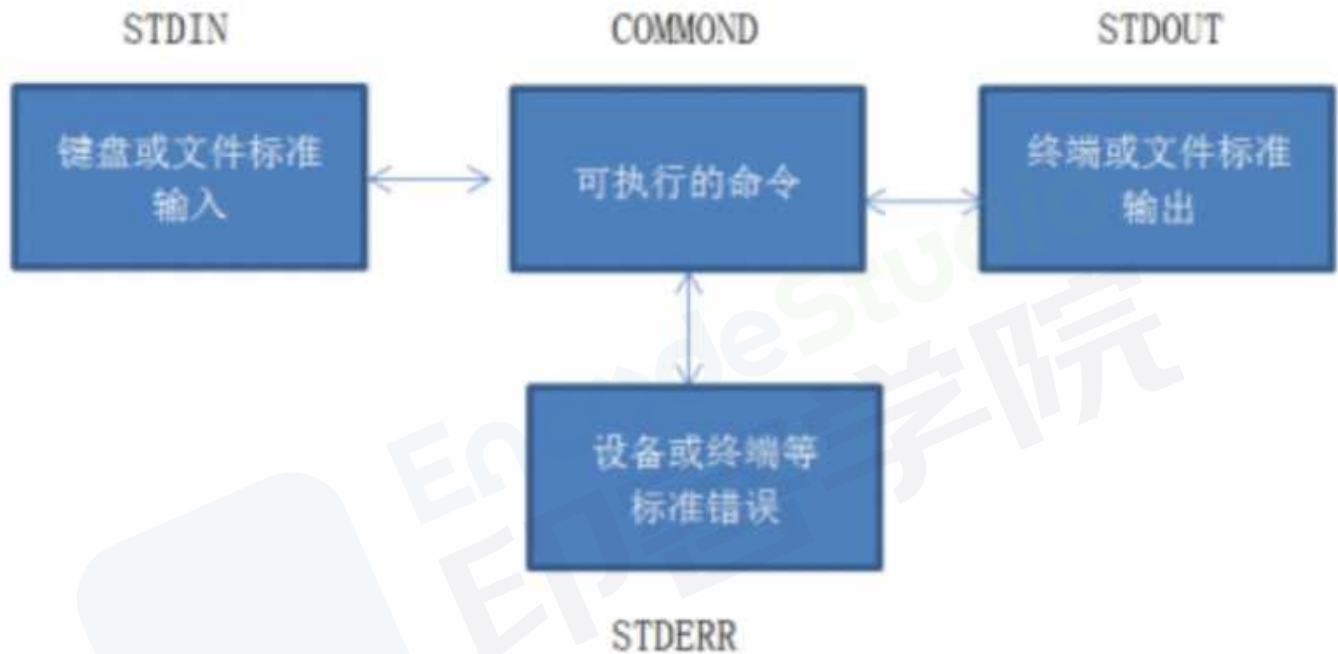


### 7.1. 是什么

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。