9.3. 区别

撤销（revert）被设计为撤销公开的提交（比如已经push）的安全方式，`git reset`被设计为重设本地更改

因为两个命令的目的不同，它们的实现也不一样：重设完全地移除了一堆更改，而撤销保留了原来的更改，用一个新的提交来实现撤销

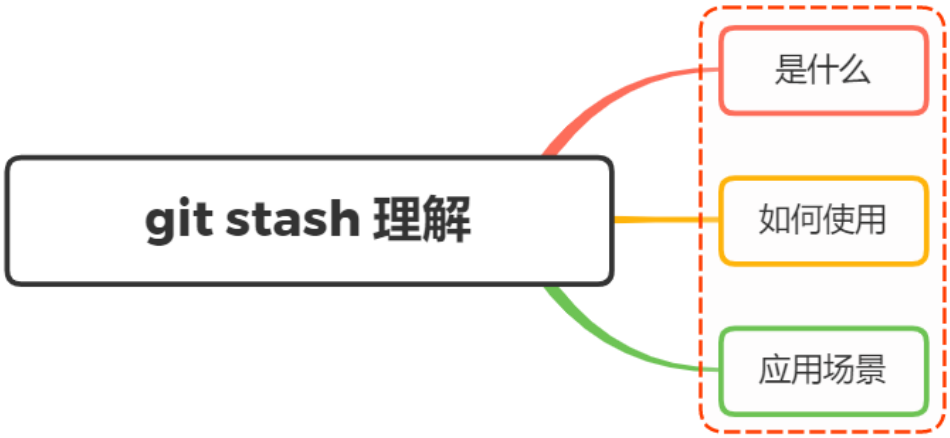
两者主要区别如下：

- git revert是用一次新的commit来回滚之前的commit，git reset是直接删除指定的commit
- git reset 是把HEAD向后移动了一下，而git revert是HEAD继续前进，只是新的commit的内容和要revert的内容正好相反，能够抵消要被revert的内容
- 在回滚这一操作上看，效果差不多。但是在日后继续 merge 以前的老版本时有区别

> git revert是用一次逆向的commit“中和”之前的提交，因此日后合并老的branch时，之前提交合并的代码仍然存在，导致不能够重新合并
>
> 但是git reset是之间把某些commit在某个branch上删除，因而和老的branch再次merge时，这些被回滚的commit应该还会被引入

- 如果回退分支的代码以后还需要的情况则使用`git revert`， 如果分支是提错了没用的并且不想让别人发现这些错误代码，则使用`git reset`

10. 说说你对git stash 的理解？应用场景？



10.1. 是什么

stash，译为存放，在 git 中，可以理解为保存当前工作进度，会把暂存区和工作区的改动进行保存，这些修改会保存在一个栈上

后续你可以在任何时候任何分支重新将某次的修改推出来，重新应用这些更改的代码

默认情况下，`git stash` 会缓存下列状态的文件：

- 添加到暂存区的修改（staged changes）
- Git跟踪的但并未添加到暂存区的修改（unstaged changes）

但以下状态的文件不会缓存：

- 在工作目录中新的文件（untracked files）
- 被忽略的文件（ignored files）

如果想要上述的文件都被缓存，可以使用 `-u` 或者 `--include-untracked` 可以工作目录新的文件，使用 `-a` 或者 `--all` 命令可以当前目录下的所有修改

10.2. 如何使用

关于 `git stash` 常见的命令如下：

- git stash
- git stash save
- git stash list
- git stash pop
- git stash apply
- git stash show
- git stash drop
- git stash clear

10.2.1. git stash

保存当前工作进度，会把暂存区和工作区的改动保存起来

10.2.2. git stash save

`git stash save` 可以用于存储修改.并且将 `git` 的工作状态切回到 `HEAD` 也就是上一次合法提交上

如果给定具体的文件路径，`git stash` 只会处理路径下的文件.其他的文件不会被存储，其存在一些参数：

- --keep-index 或者 -k 只会存储为加入 git 管理的文件
- --include-untracked 为追踪的文件也会被缓存,当前的工作空间会被恢复为完全清空的状态
- -a 或者 --all 命令可以当前目录下的所有修改，包括被 git 忽略的文件

10.2.3. git stash list

显示保存进度的列表。也就意味着， `git stash` 命令可以多次执行，当多次使用 `git stash` 命令后，栈里会充满未提交的代码，如下：

```
linguanghui@macdeMacBook-Pro git-test % git stash list
stash@{0}: WIP on test: 8431f5e first commit
stash@{1}: WIP on test: 8431f5e first commit
```

其中， `stash@{0}` 、 `stash@{1}` 就是当前 `stash` 的名称

10.2.4. git stash pop

`git stash pop` 从栈中读取最近一次保存的内容，也就是栈顶的 `stash` 会恢复到工作区

也可以通过 `git stash pop` + `stash` 名字执行恢复哪个 `stash` 恢复到当前目录

如果从 `stash` 中恢复的内容和当前目录中的内容发生了冲突，则需要手动修复冲突或者创建新的分支来解决冲突

10.2.5. git stash apply

将堆栈中的内容应用到当前目录，不同于 `git stash pop` ，该命令不会将内容从堆栈中删除

也就说该命令能够将堆栈的内容多次应用到工作目录中，适应于多个分支的情况

同样，可以通过 `git stash apply` + `stash` 名字执行恢复哪个 `stash` 恢复到当前目录

10.2.6. git stash show

查看堆栈中最新保存的 `stash` 和当前目录的差异

通过使用 `git stash show -p` 查看详细的不同

通过使用 `git stash show stash@{1}` 查看指定的 `stash` 和当前目录差异

```
linguanghui@macdeMacBook-Pro git-test % git stash show
 a.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
linguanghui@macdeMacBook-Pro git-test % git stash show -p
diff --git a/a.txt b/a.txt
index 72943a1..e6d83f1 100644
--- a/a.txt
+++ b/a.txt
@@ -1 +1,2 @@
-aaa
+aaaa
+1
linguanghui@macdeMacBook-Pro git-test % git stash show stash@{1}
 c.txt | 1 +
 1 file changed, 1 insertion(+)
```

10.2.7. git stash drop

`git stash drop` + `stash` 名称表示从堆栈中移除某个指定的stash

10.2.8. git stash clear

删除所有存储的进度

10.3. 应用场景

当你在项目的一部分上已经工作一段时间后，所有东西都进入了混乱的状态， 而这时你想要切换到另一个分支或者拉下远端的代码去做一点别的事情

但是你创建一次未完成的代码的 `commit` 提交，这时候就可以使用 `git stash`

例如以下场景：

当你的开发进行到一半,但是代码还不想进行提交 ,然后需要同步去关联远端代码时.如果你本地的代码和远端代码没有冲突时,可以直接通过 `git pull` 解决

但是如果可能发生冲突怎么办.直接 `git pull` 会拒绝覆盖当前的修改，这时候就可以依次使用下述的命令：

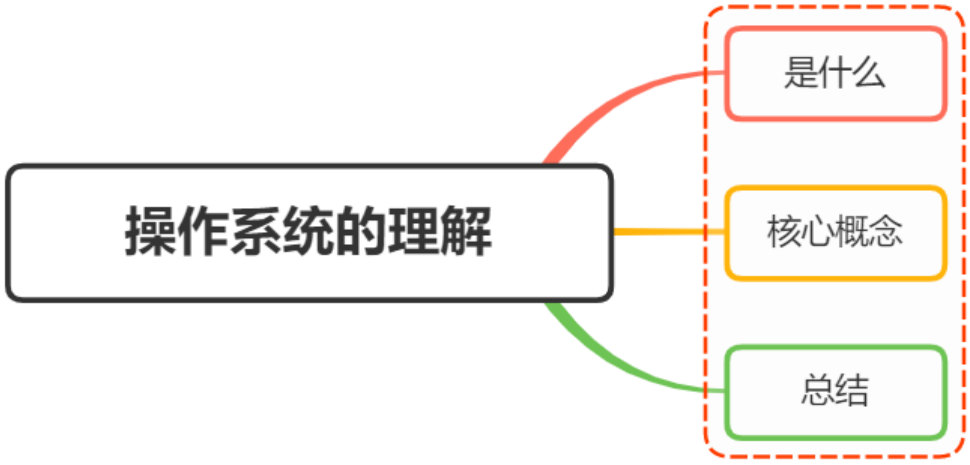
- git stash
- git pull
- git stash pop

或者当你开发到一半，现在要修改别的分支问题的时候，你也可以使用 `git stash` 缓存当前区域的代码

- git stash：保存开发到一半的代码
- git commit –m '修改问题'
- git stash pop：将代码追加到最新的提交之后

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？



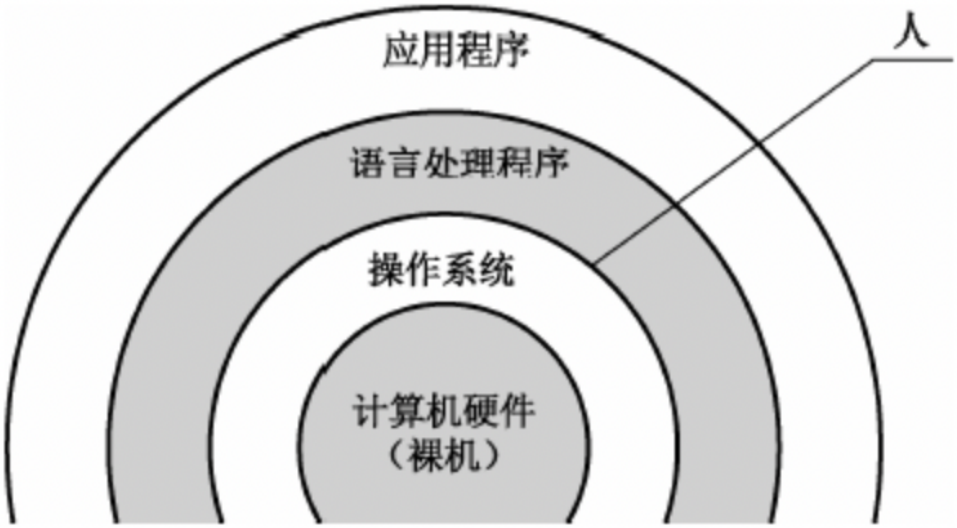
1.1. 是什么

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

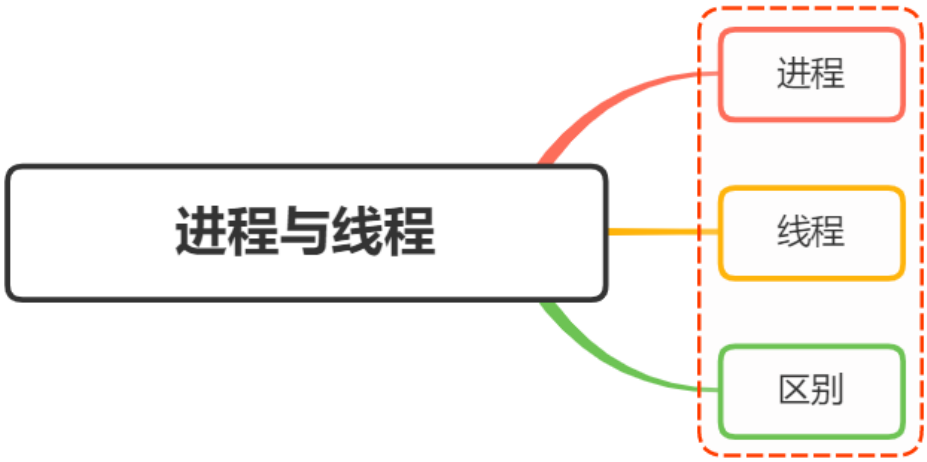
储器

- 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
- 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`

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	linguanghui
Google Chrome	0.8	8.30	否	否	linguanghui
Code	0.1	6.17	否	否	linguanghui
微信	1.2	2.48	否	否	linguanghui
聚焦	0.5	0.75	-	-	-
Typora	0.2	0.16	是	否	linguanghui
邮件	0.1	0.12	是	否	linguanghui
访达	0.1	0.08	否	否	linguanghui
CCXProcess	0.0	0.03	否	否	linguanghui
Scroll Reverser	0.5	0.03	否	否	linguanghui
终端	0.0	0.01	是	否	linguanghui

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

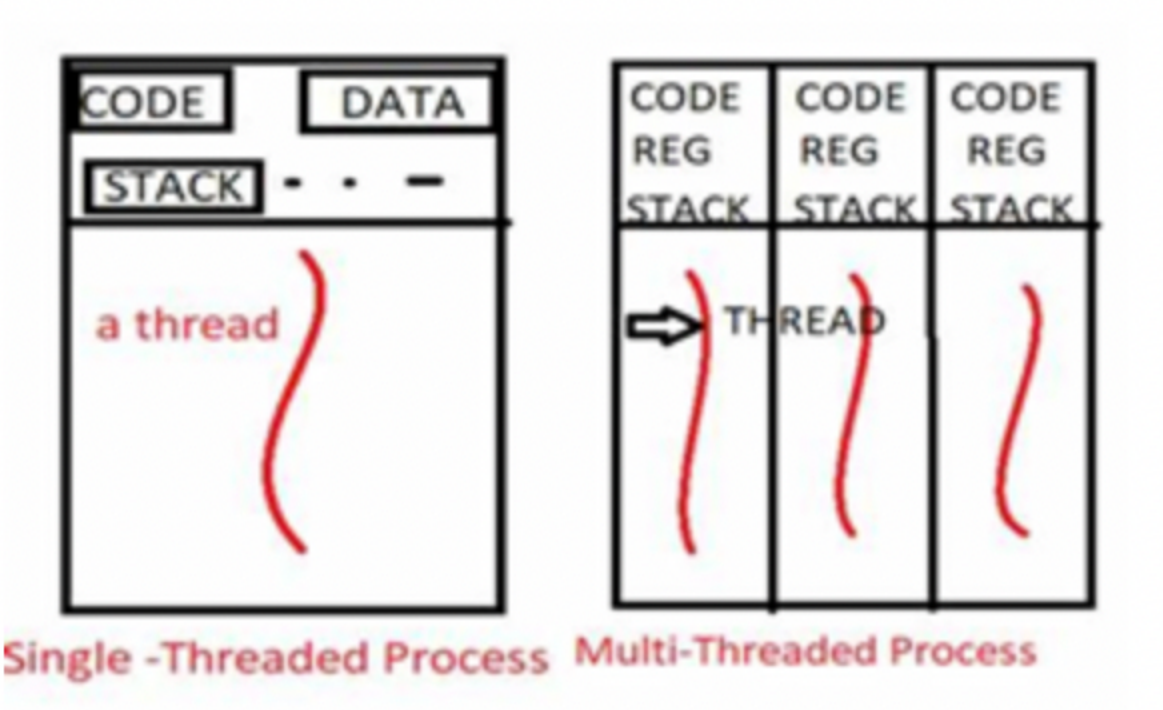
2.2. 线程

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

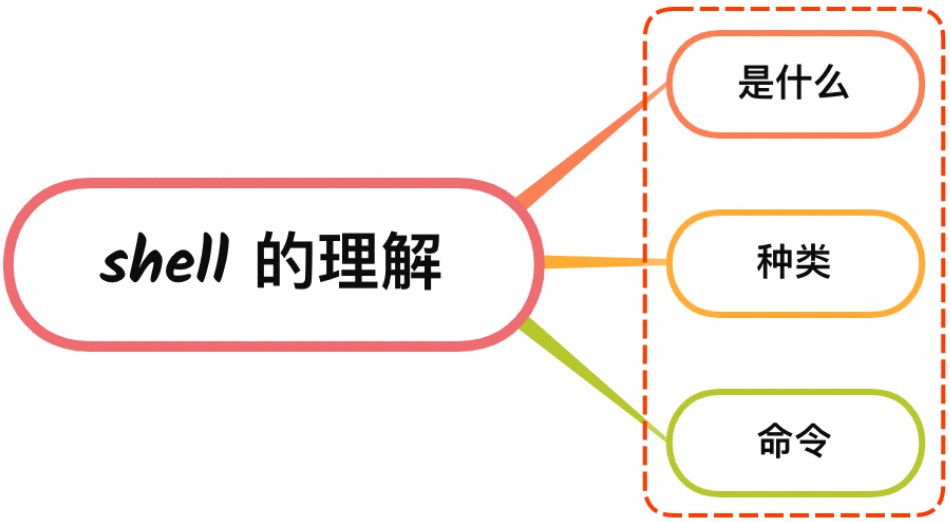
销；线程可以看做轻量级的进程，同一类线程共享代码和数据空间，每个线程都有自己独立的运行栈和程序计数器（PC），线程之间切换的开销小

- **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？



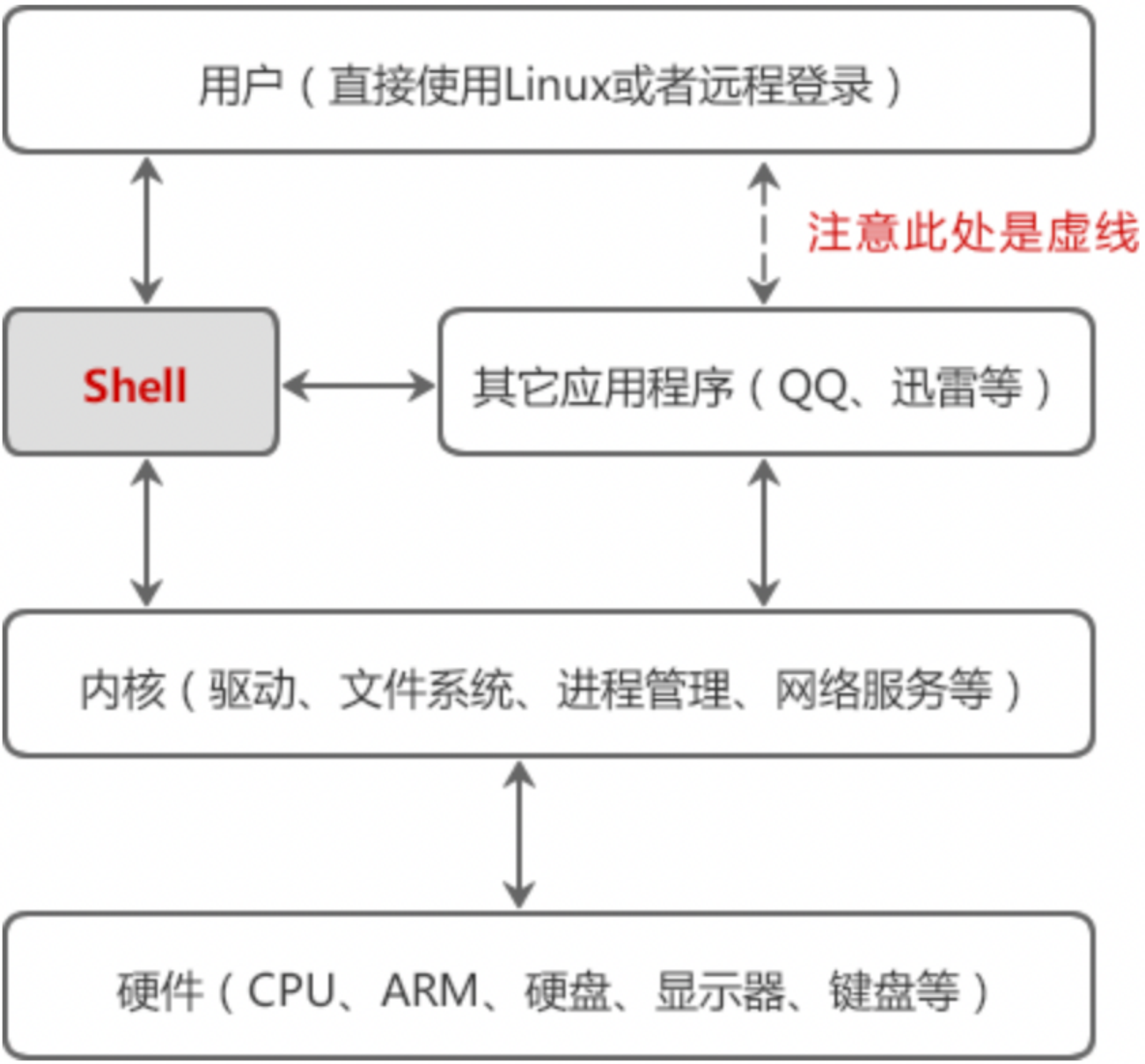
3.1. 是什么

`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
#!/bin/bash

# 执行的命令主体
ls
echo "hello world"
```

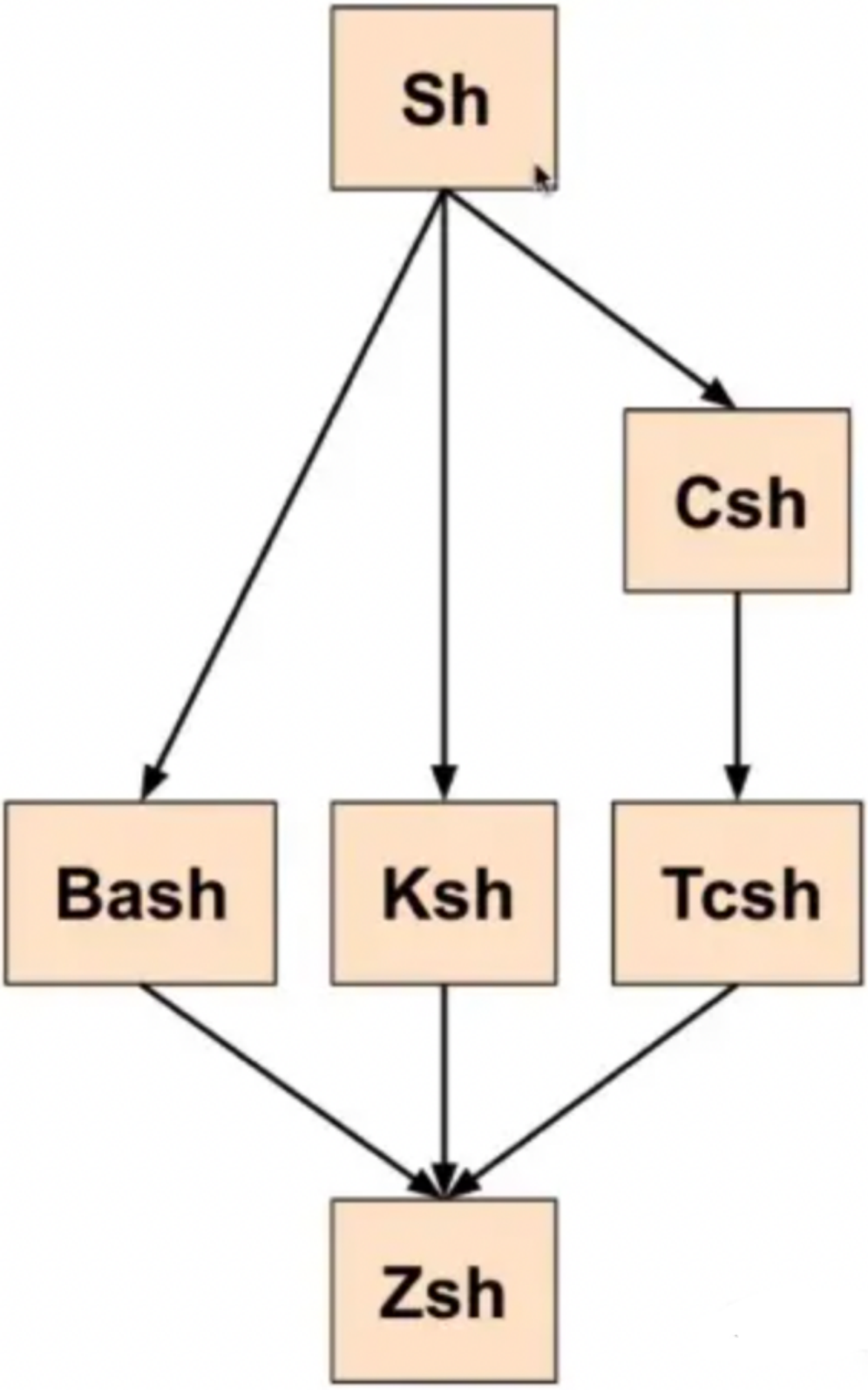
- #!/bin/bash ： 指定脚本要使用的 Shell 类型为 Bash
- ls、echo： 脚本文件的内容，表明我们执行 hello.sh 脚本时会列举出当前目录的文件列表并且会向控制台打印 `hello world

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

```bash
word="hello"
echo ${word}
# 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

Shell | 复制代码

```shell
if [[ 2 -ne 1 ]]; then
  echo "true"
else
  echo "false"
fi
# Output: true
```

`fi` 标志着条件代码块的结束

3.3.3. 函数

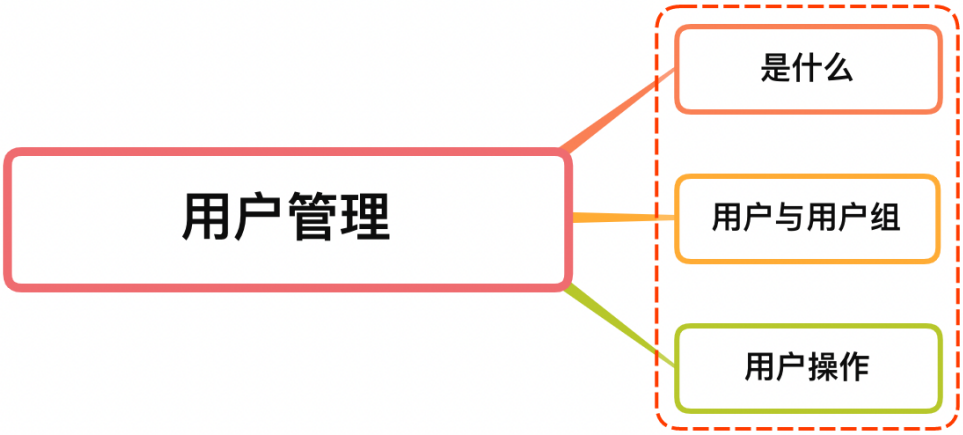
bash 函数定义语法如下：

Bash | 复制代码

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

- 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：


```
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	r	可以读取文件的内容	可以列出目录中的文件列表
写权限	w	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	x	可以执行该文件	可以使用cd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：

```LaTeX
useradd [options] [username]
```

例如：

添加一个一般用户

```
# useradd kk //添加用户kk
```

为添加的用户指定相应的用户组

```
# 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

```
# useradd -r kk //创建一个系统用户kk
```

为新添加的用户指定/home目录

```
# useradd-d /home/myf kk //新添加用户kk，其home目录为/home/myf
//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

```
asswd [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

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

显示账号密码信息

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

删除用户密码

```
# passwd -d lx138
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

```
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。

```LaTeX
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

```Plain Text
# 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

```LaTeX
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。