**shell脚本**

**1.#！**

    shell脚本通常以#！起始，当脚本作为一个可执行文件运行时，它指明了这个脚本文件的解释程序。

    如果脚本文件中没有#!这一行，那么它执行时会默认用当前Shell去解释这个脚本（$SHELL环境变量/bin/bash）。

    如果以将脚本作为sh命令行参数，则会忽略#！行。

**2.进程相关**

**常用命令**: ps(收集进程信息,-f参数表示列出详细信息,-e表示获取运行在系统中的每一个进程信息)

                top(输出一个占用CPU最多的进程列表)

**信号机制**: kill -l列出了所有可用信号

                通过kill命令向进程发送制定信号 kill -s SIGNAL PID ,以下是常用信号

                SIGHUP 1: 对控制进程或终端进行挂起检测,常因终端掉线或用户退出而引发

                SIGINT  2: 当按下Ctrl+C时发送此信号

                SIGKILL 9: 用于强制杀死进程  kill -s SIGKILL PID | kill -s 9  PID

                SIGTERM 15: 默认用于终止进程,与SIGKILL不同的是该信号可以被阻塞和处理. 通常用来要求程序自己正常退出,正常来讲如果退出正常的话,进程的子进程也可以被终止

                SIGTSTP 20:当按下Ctrl+Z时发送此信号

**捕捉并响应信号:** trap命令在脚本中用于捕获信号,一旦脚本收到这个信号时,对应于这个信号的函数就会开始执行

                       trap 'handler' SIGINT 表示脚本运行时,按下Ctrl+C,脚本就会执行handler方法;通过无限循环while来保持捕获状态,这样就可以响应另一个进程发送来的信号.如果进程要忽略某个信号,可以把commands设为空,trap '' 2表示忽略Ctrl+C信号.

**进程前后台切换:** 进程启动时,在参数后面加上一个&可以让该进程在后台启动,或者使用bg PID也可以达到同样的目的(jobs查看后台运行进程)

                         fg jobnumber 让后台运行的进程到前台来,jobnumber不是pid

                         bg jobnumber 让进程到后台去,jobnumber不是pid

                         ctrl+c是强制中断程序的执行,而ctrl+z的是将进程中断,但是此进程并没有结束,只是维持挂起的状态,用户可以使

                         用fg/bg操作继续前台或后台的任务,fg命令重新启动前台被中断的任务,bg命令把被中断的任务放在后台执行.

**kill进程**: kill PID 杀死进程,把资源释放出来,但是不能清理子进程

                  kill -s SIGKILL PID 强制杀死进程,比较危险,可能导致系统资源无法释放

                  kill -TERM PID 试图正常终止进程以及子进程

                  kill -HUP PID 重启进程    ＃linux PCB

**3.重定向**

文件描述符: 文件描述符是与文件输入输出相关联的非负整数,文件描述符0,1,2是给系统预留的

                       0    stdin(标准输入)

                       1    stdout(标准输出)

2    stderr(错误输出)

    exec命令:  系统调用exec是以新的进程去代替原来的进程，但进程的PID保持不变。原进程的代码段，数据段，堆栈段被新的进程所代替。

                        exec 在对文件描述符进行操作的时候，也只有在这时，exec 不会覆盖当前的 shell 环境。

                      进程包含内容(一个可执行程序; 与进程相关联的数据等)

                      exec I/O重定向:

                      shell的文件描述符为0~9

                      exec 1>outfile  打开文件outfile作为stdout

                      exec 2>errfile   打开文件 errfile作为 stderr

                      exec n>file   将文件file和n号描述符关联，文件作为输出用途

                      date >&n   将date输出定向到n描述符，即文件file

                      exec n<file   将文件file和n号描述符关联，文件只作为输入用途

                       exec n<>file  将文件file和n号描述符关联，文件作为输入输出用途

                       exec 1>&-  关闭 文件描述符1

**4.shell函数**

function fname(){

        echo $1,$2;

        echo "$@";

        echo "$\*";      ＃main函数返回值？ 返回值范围？

        return 0;　　＃返回值范围为0～255

      }

如果需要定义自己变量，可以在函数中定义：local 变量=值 ，这时变量就是内部变量，它的修改，不会影响函数外部相同变量的值

      通过$?获得函数返回值,范围为0~255; 函数执行正常默认返回0,错误返回127

      调用函数直接使用函数名即可，在函数名后加参数可以传递参数

　　fname arg1 arg2;

  注：无符类型：原码取反为反码，反码加1为补码，负数以补码表示

　　  有符类型：最高位0正1负

示例：

#!/bin/bash

set -e

function param(){

echo $1 ":" $2

}

param $1 $2