1.日志查询类

按关键字不分页查询  
grep "applyId” app.log //applyId为要查询的关键字，app.log为日志名

按关键字从前往后分页查询  
cat -n app.log | grep "applyId" | more  
以上命令会分页输出，使用空格键翻页，使用回车键显示更多

按关键字只显示最后M行查询  
tail -n M app.log | grep “applyId” //M为行数

按关键字动态显示最新M行查询  
tail -f M app.log | grep “applyId" //M为行数

日志太多，将相关日志输出到文件后查看  
cat -n app.log | grep "applyId" >app-new.log //app-new.log为文件名

按日期范围查询  
sed -n '/2017-10-17 10:10:10/,/2017-10-17 10:15:00/p' app.log  
注意：  
2017-10-17 10:10:10 和 2017-10-17 10:15:00必须是日志中真实打印输出过的

动态输出最新M行日志  
tail -100f app.log

2.端口查询类

查看端口使用情况  
netstat -tln

查看指定端口使用情况  
netstat -ano | grep 2181 //查看2181端口使用情况

查看占用指定端口的进程  
sudo lsof -i :2181 //查看2181端口被哪个进程占用

3.查看网络连接

列出所有打开的网络连接(tcp和udp)  
lsof -i

列出所有打开的tcp网络连接  
lsof -i tcp

列出所有打开的udp网络连接  
lsof -i udp

列出指定的tcp端口情况  
lsof -i tcp:80

列出指定的udp端口情况  
lsof -i udp:55

4.查看资源使用情况

查看CPU使用率、内存、虚拟内存交换情况、IO读写情况  
vmstat 5 //每5秒刷新一次

查看系统中各个进程的资源占用状况  
top //使用top命令后，按Shift+M, 显示结果将按照进程内存占用率排序

查看内存使用情况  
free -m //以MB为单位

查看磁盘空间使用情况  
df -h

5.进程操作

根据名称查询进程  
ps -ef | grep java //使用标准格式显示  
ps aux | grep java //使用BSD的格式显示

杀掉进程  
kill -9 pid //pid为相应的进程号

6.文件复制

scp是一种安全方式的文件复制，基于ssh

复制本地文件到远程机器  
复制本地文件app.config到远程192.168.10.190服务器/config/目录下  
scp /config/app.config root@192.168.10.190:/config/ //root是用户名

复制远程文件到本地

scp root@192.168.10.190:/config/app.config /config/ //root是用户名

# 初识Linux

**什么是Linux**

1. Linux可以划分为四个部分：
   1. Linux内核
   2. GNU工具
   3. 图形化桌面环境
   4. 应用软件

**深入探究Linux内核**

1. Linux系统的核心是内核。内核控制着计算机系统上的所有硬件和软件，在必要时分配硬件，并根据需要执行软件。内核主要负责四种功能
2. **系统内存管理**
   1. 内核不仅管理服务器上的可用物理内存，还可以创建和管理虚拟内存
   2. 内核通过硬盘上的存储空间实现虚拟内存，这块区域成为交换空间：内核不断地在交换空间和物理内存间反复交换虚拟内存中的内容，使得可用内存更多
   3. 内存存储单元按组划分成很多页（page）。每个页存放在物理内存或交换空间
   4. 内核维护一个**内存页面表**，指明哪些页面位于物理内存中，哪些在磁盘上
   5. 内核会记录哪些页面**正在使用**，并**自动**把一段时间**未访问**的内存页面复制到交换空间（**即使还有可用内存**）。
   6. 当程序要访问一个已被换出的内存页面时，内核必须从物理内存换出另外一个页面给它**让出空间**，然后从交换空间换入请求的内存页面。这个过程就**缺页中断**，需要**花费时间**，拖慢运行中的进程。
3. **软件程序管理**
   1. 运行中的程序称为进程，进程可以在前台运行，将输出显示在屏幕，也可以在后台运行，屏蔽用户。内核控制Linux系统如何**管理**运行在系统上的**所有进程**。
   2. 内核创建了**第一个进程**（init进程）**启动**系统上的所有其他进程。当内核启动时，它会将init进程加载到虚拟内存中。内核在启动其他任何进程时，都会在虚拟内存中给新进程分配一块**专有区域**来存储该进程用到的数据和代码。
   3. Ubuntu采用/etc/init.d目录，将开机时启动或停止某个应用的**脚本**放在这个目录下
   4. 这些脚本通过/etc/rcX.d目录下的**入口**启动，X代表运行级，运行级决定了init进程运行**目录中定义好的**某些特定类型的进程。Linux有五个启动运行级：
      1. 运行级为1，只启动基本的系统进程和一个控制台中断进程，成为**单用户模式**。通常用来进行**紧急的文件系统维护**，仅有**管理员**能登陆。
      2. **标准**的启动运行级是3。该运行级下，大多数应用软件（如网络支持程序）都会启动，是**基于控制台**的系统。
      3. 运行级为5时，会启动**图形化**的X Window系统，允许用户通过图形化桌面窗口登录系统。
4. **硬件设备管理**
   1. 任何Linux系统需要与之通信的设备，都需要在内核代码中加入其**驱动程序**代码。驱动程序代码相当于应用程序和硬件设备的中介，允许内核和设备**交换数据。**
   2. Linux内核中有两种方法用于插入设备驱动代码：
      1. **编译**进内核的设备驱动**代码（每次都要重新编译一次内核代码）**
      2. 可**插入**内核的设备驱动**模块**
         1. **无需重新编译**内核，更高效
         2. 设备不再使用时，也可将模块从内核中**移走**，更方便
   3. Linux将硬件设备当成了特殊的文件，成为**设备文件**。设备文件分三类：
      1. **字符型**设备文件：处理数据时**每次只能处理一个字符**的设备（如大多数类型的调制解调器和终端）
      2. **块**设备文件：每次能处理**大块数据**的设备（如**硬盘**）
      3. **网络**设备文件：采用**数据包**发送和接收数据的设备
         1. 网卡
         2. 回环设备，允许Linux系统使用常见的网络编程协议**同自身通信**。
   4. Linux为系统上每个设备都创建一种称为**节点**的特殊文件。
      1. **与设备的所有通信都通过节点完成**。
      2. 每个节点都有**唯一的数值对**供Linux内核**标识**它。
      3. 数值对包括一个**主设备号**和一个**次设备号**，**类似的设备**被划分到同样的主设备号下，次设备号用于标识主设备组下的某个**特定设备**。
5. **文件管理系统**，看第6页
   1. 有别于其他操作系统，Linux内核支持通过**不同类型的文件系统**从磁盘读写数据。
   2. 除了自有文件系统外，Linux还支持从**其他操作系统采用的文件系统**读写数据。
   3. Linux内核采用虚拟文件系统(VFS)作为和每个文件系统**交互的接口**，为Linux内核同**任何类型文件系统**通信提供了一个标准接口。当某个文件系统都被挂载和使用时，VFS将信息**缓存在内存**中。

**GNU工具**

1. 开源软件是指：任何人都可以使用、修改或集成进自己系统的软件，无需任何授权费用
2. coreutils软件包**移植**了很多Unix系统常见的**命令行工具**供Linux系统使用。这组GNU核心工具包由三部分组成：
   1. 用以处理**文件**的工具
   2. 用以操作**文本**的工具
   3. 用以管理**进程**的工具
3. Shell是一种特殊的**交互式**工具，为用户提供了**启动程序**、**管理**文件系统中的**文件**以及运行在Linux系统上的**进程**的途径。
   1. Shell的核心是**命令行提示符**：允许用户**输入**文本命令，然后**解释**命令，并在内核**执行**。
   2. Shell包含了一组内部命令，用这些命令可以完成诸如**复制文件**、**移动文件**、**重命名文件**、**显示和终止**系统**进程**等操作。
   3. Shell允许在命令行提示符中**输入程序的名称**，它将程序名传递给内核以**启动**它。
   4. 可以将**多个shell命令**放在文件中作为程序执行，这些程序称为**shell脚本**。在命令行执行的任何命令都可以放入shell脚本。
   5. 在Linux系统上，有多种Linux Shell可用。不同shell有不同的特性，有些更利于**创建脚本**，有些更利于**管理进程**
   6. **Bourne shell**是标准的Unix Shell
   7. **Bash shell**是所有Linux发行版**默认**的shell，由GNU项目开发**。**
   8. **Ash shell**是运行在**内存受限**环境中简单的轻量级shell，但与bash shell**完全兼容**
   9. **Korn shell**是一种与Bourne shell兼容的shell，支持如关联数组和浮点运算等高级的编程特性。
   10. **Tcsh shell**是一种将C语言中的一些元素引入shell脚本中的shell
   11. **Zsh shell**结合了bash、tcsh和korn的特性，同时提供了高级编程特性、共享历史文件和主题化提示符，是一种高级shell

**Linux桌面环境**

1. W Window系统
   1. 有两个基本要素决定了**视频环境**：显卡和显示器。
   2. X Window软件是**图形显示**的**核心**部分，是**直接**和PC上的显卡和显示器打交道的**底层**程序。控制Linux程序如何在电脑上显示出窗口和图形。
   3. 核心的X Window软件只能够提供图形化**显示环境**，但是并没有提供**桌面环境**供用户**操作文件**或者**开启程序**。
2. KDE桌面（K Desktop Enviroment）
   1. 能够生成一个**类似于windows操作系统的图形化桌面环境**
   2. 允许把应用程序图标和文件图标放在桌面特定位置上，单机图标就能运行程序
   3. 底部的横条为面板，有四个部分组成，类似于Windows：
      1. KDE菜单和Windows的开始菜单类似，包含启动已安装程序的链接。
      2. 程序快捷方式，一些常使用的程序的快速链接
      3. 任务栏：显示当前正在运行的程序。
      4. 小应用程序：一些根据状态发生变化的图片，相当于右下角
   4. 除了桌面功能，KDE项目还开发了大量可以在KDE环境下运行的应用程序
3. GNOME桌面（GNU Network Object Model Enviroment，GNU网络对象模型环境）
   1. 现在已成为许多Linux发行版**默认的桌面环境**
   2. 不再沿用Windows的标准感官，但还是继承了一些习惯的功能：
      1. 一块防止图标的桌面区域
      2. 两个面板区域
      3. 拖放功能
   3. 同样开发了一批集成进GNOME桌面的图形化程序
4. Unity桌面
   1. 这是**Ubuntu**采用的桌面环境
   2. 电脑、手机、平板的桌面使用方式相同
5. 其他桌面
   1. 图形化桌面环境的弊端在于：占用相当一部分**系统资源**保证正常运行。
   2. Linux开发人员开发了一些**低内存开销**的图形化桌面应用，提供了能在老旧PC上完美运行的**基本功能**。
   3. 没有大量专门为这些桌面设计的应用
   4. 仍然能运行许多**基本的图形化程序**，支持文字处理、电子表格、数据库、绘图、多媒体等功能

**Linux发行版**

1. 发行版就是完整的Linux系统包，有很多不同的Linux发行版满足可能存在的各种运算需求。大多数发行版是**为某个特定用户群定制**的，如商业用户、软件开发人员、普通用户
2. 每个定制的发行版都包含了**支持特定功能**所需的各种软件包，比如为软件开发人员准备的编译器和IDE（集成开发环境）
3. 核心Linux发行版（典型的有红帽）
   1. 包含内核、一个或多个图形化桌面环境、预编译好的几乎所有能见到的Linux应用
   2. 发行版的大量**可选配置**可能会导致加载过多或没有加载够
   3. 从高端的数据库服务器到常见的游戏等应用软件应有尽有。
4. 特殊用途的发行版（CentOS基于红帽、Ubuntu基于学校、Mint基于家庭娱乐）
   1. 通常基于某个主流发行版，但仅包含其中**一小部分**用于特殊用途的应用程序
   2. 定制化发行版还尝试通过**自动检测**和**自动配置**常见硬件帮助新手安装。
   3. 特殊用途的发行版足有上百款，并且还在不断增多。
   4. 许多特殊用途的Linux发行版都是基于**Debian Linux**。

**Linux LiveCD（Ubuntu,PCLinuxOS）**

1. 包含了Linux样本系统的可引导CD，不是必须从标准硬盘启动，直接同过CD启动PC。
2. 由于单张CD容量限制，**并非完整**的Linux系统，但可以自己**加入**各种软件。
3. 许多特定用途的发行版都有对应的Linux LiveCD版本
4. 缺点是：
   1. 由于要从CD上访问所有东西，应用程序运行会很慢
   2. 由于无法向CD写入数据，对Linux系统的任何修改都会在重启后失效
5. 一些Linux LiveCD进行了改进：
   1. 能将CD上的Linux系统文件复制到内存中
   2. 能将系统文件复制到硬盘上
   3. 能在U盘上存储系统设置
   4. 能在U盘上存储用户设置

# 走进Shell

**进入命令行**

1. 在没有图形化桌面时，与Unix系统交互的唯一方式就是文本命令行界面（Command line interface，CLI），只能接受文本输入，也只能显示文本和基本的图形输出。
2. 如今，想要输入shell命令仍旧需要使用CLI。

**控制台终端**

1. 进入CLI的一种方法是让Linux退出图形化桌面模式，进入文本模式
2. 这种模式成为Linux控制台，和图形化桌面出现之前一样。
3. Linux系统启动后会创建5~6个甚至更多虚拟控制台，通过显示器和键盘就能访问。

**通过Linux控制台终端访问CLI**

1. 很多Linux发行版启动多个虚拟控制台后会切换到图形化环境，为用户提供了图形化登录，但要访问虚拟控制台就只能通过**手动**的方式
2. 大多数Linux发行版，可以用Ctrl+Alt+F1~F7，FX对应虚拟控制台X，Ubuntu用F7
3. Login提示符后输入用户ID，password提示符后输入密码就可以进入控制台终端了
4. 在Linux虚拟控制台**无法运行任何**图形化程序。
5. 登录后可以保持**活动状态**，可以在多个虚拟控制台**切换**，拥有多个活动会话。
6. Setterm -inversescreen on可以将背景设置为白色，文本设置为黑色。off就关闭该模式
7. Setterm -background white可以设置背景，setterm -foreground black可以设置字体
8. 有black、red、green、yellow、blue、magenta、cyan、white八种颜色可选
9. Setterm -store恢复外观为默认设置，setterm -reset恢复成默认设置后清屏

**图形化终端**

1. 可以使用Linux图形化桌面环境中的终端仿真包，但只负责Linux图形化体验部分
2. 完整的体验效果需要借助多个组件实现，包括图形化终端仿真软件（客户端）
3. Linux图形化桌面环境包含四个部分：
   1. 客户端：用于请求图形化服务，如桌面环境、网络浏览器、图形化终端仿真器
   2. 显示服务器：负责显示和输入设备
   3. 窗口管理器：为窗口加入边框，提供移动和管理窗口等功能
   4. 部件库：为桌面环境中的客户端添加菜单以及外观项
4. 要在桌面使用命令行关键在于图形化终端仿真器，可以看做GUI中的CLI终端

**通过图形化终端仿真访问CLI**

1. 相比较于虚拟化控制台终端，图形化桌面环境提供了更多访问CLI的方式
2. 有大量仿真器，各有各的特性，看P20
3. GNOME Terminal、Konsole Terminal、xterm通常都会默认安装在Linux发行版中

**使用GNOME Terminal仿真器，P21**

1. 是GNOME桌面环境的默认终端仿真器，Ubuntu Unity也采用其作为默认终端仿真器
2. Ubuntu的Unity可以采用快捷键Ctrl+Alt+T快速访问GNOME终端
3. 一些GNOME终端的菜单项和快捷键，P22

**使用Konsole Terminal仿真器，P25**

1. KDE桌面项目的终端仿真软件包，具备基本的终端仿真特性，还有一些更高级的图形应用程序功能。
2. 大多数桌面环境中，可以创建启动器访问Konsole Terminal这样的应用程序
3. 如果已经安装Konsole Terminal，可以在其他终端模拟器输入konsole命令启动
4. 一些Konsole终端的菜单项和快捷键，P26

**使用xterm仿真器，P29**

1. 是最古老也最基础的终端仿真软件包，在X Window出现之前就有了，通常默认包含在发行版中。
2. 功能完善，并且不需要太多资源来运行，在专门为老旧硬件设计的Linux发行版很流行
3. 能够仿真旧式终端，能够仿真VT色彩序列控制码，可以在脚本中使用色彩。
4. 图形化界面非常朴素，常常没有包含在桌面环境的菜单中
5. 在Ubutu的Unity界面是默认安装的，在GNOME和KDE并没有默认安装
6. Xterm的命令行参数，众多的配置选项，通过-help参数确定哪些支持的，P31

# 基本bash shell命令

**Shell提示符**

1. 在ubuntu Linux系统上shell提示符是：username@server:~$
2. Shell提示符除了作为shell的入口，还能够提供如用户名、系统名等辅助信息

**Bash手册**

1. 大多数Linux发行版自带查找shell命令的手册
2. man命令用来访问存储在Linux系统上的手册页面。
3. Man xterm可以进入查找xterm命令的手册页面。
4. 一开始可能没什么用，但养成习惯阅读decription部分的前两段可以学到各种技术行话
5. 手册是由**分页程序**显示的，空格或上下键翻页，回车键逐行查看，q键退出
6. Man man指令可以查看与手册相关的手册页，手册页相关P36

**浏览文件系统**

1. 等登录系统并获得shell命令提示符后，通常位于自己的**主目录**中。
2. Linux将文件存储都在单个目录结构中，称为**虚拟目录**。
3. Linux虚拟目录结构只包含了一个称为**根目录**的基础目录，根目录下的目录和文件会按照它们的路径一一列出。
4. Linux用正斜线/**划分目录**，而反斜线\是**转义字符**。
5. 路径本身并没有提供任何有关文件究竟存放在哪个**物理磁盘**上的信息。
6. Linux上的第一块硬盘称为根驱动器，**根驱动器**包含了虚拟目录的**核心**，其他目录从这里开始构建。
7. Linux会在根目录上创建一些叫**挂载点**的特殊目录，是虚拟目录中用于**分配额外存储设备**的目录，虚拟目录会让文件和目录出现在这些挂载点目录中，但**实际**存储在别的存储器
8. 通常**系统文件**都存在根驱动器中，**用户文件**则存储在其他驱动器中。
9. 比如在根目录下创建一个挂在目录/HOME，那么用户目录就在这个目录下
10. 常见Linux目录名称：
    1. /：虚拟目录的根目录，通常不存储文件
    2. /bin：二进制目录，存放许多**用户级**的GNU工具
    3. /boot：启动目录，存放启动文件
    4. /dev：设备目录，Linux在这创建设备节点
    5. /etc：系统配置文件目录
    6. /home：主目录，Linux在这里创建用户目录
    7. /lib：库目录，存放系统和应用程序的库文件
    8. /media：媒体目录，可移动媒体设备的常用挂载点
    9. /mnt：挂载目录，另一个可移动媒体设备的常用挂载点
    10. /opt：可选目录，常用于存放第三方软件包和数据文件
    11. /proc：进程目录，存放现有硬件和当前进程的相关信息
    12. /root：root用户的主目录
    13. /sbin：系统的二进制目录，存放许多GNU**管理员级**工具
    14. /run：运行目录，存放系统运作时的**运行时数据**
    15. /srv：服务目录，存放**本地服务**的相关文件
    16. /sys：系统目录，存放**系统硬件信息**的相关文件
    17. /tmp：临时目录，可以在此目录下创建和删除临时工作文件
    18. /usr：用户二进制目录，大量**用户级的GNU工具**和**数据文件**都存储在这里
    19. /var：可变目录，用于存放**经常变化**的文件，比如日志文件。
11. 常见的目录名均基于文件系统层级标准（filesystem hierarchy standard，FHS）很多Linux发行版都遵循了FHS。FHS偶尔也会更新。
12. 获得shell CLI提示符后，会话从**主目录**开始，是分配各用户账户的一个特有目录。
13. 在CLI提示符下**使用cd命令切换虚拟目录**，可用相对路径或绝对路径
14. **绝对文件路径**定义了在**虚拟目录结构**中该目录的确切位置，以虚拟目录的**根目录**开始，相当于是目录的**全名**。
    1. Shell命令提示符中，~表示主目录，如果到了其他目录则替换~为其他路径
    2. 使用**pwd指令常看当前目录**。切换目录时使用pwd可以确保处在正确目录中。
    3. **单独使用cd可以直接跳回主目录**
15. **相对文件路径**允许用户指定一个**基于当前位置**的目标文件路径。
    1. 不以代表根目录的反斜线/开头，而是以**目录名**或一个特殊字符开始。
    2. 优点是如果只在特定目录中工作，可以**减少输入内容**
    3. 建议在熟悉了目录布局后在使用相对路径
    4. 特殊字符：
       1. 单字符（.）表示当前目录，不过对cd命令来说没什么意义
       2. 双点符（..）表示当前目录的**父目录**

**文件和目录列表**

1. 列表命令**ls可以列出当前目录下的文件和目录。**按列进行字母排序，通过颜色区分不同类型文件。
2. ls -F命令在目录名后加了正斜线/，可执行后加了星号\*，用以区分目录和文件。
3. ls -a命令显示以.号开头的隐藏文件。
4. ls -R命令可以**递归**地显示目录下的所有目录和文件，输出可能会很长
5. ls -i命令可以看一个文件的**inode编号**，是用于**标识**文件的**唯一**编号
6. 注意：
   1. 选项是**区分大小写**的
   2. 选项是可以**合并**的，比如-F -R可以合并为-FR
7. ls -l命令显示长列表（**包含每个文件更多相关信息**）
   1. 第一行显示了目录中包含的**总块数**。之后每一行包含了每一个文件的信息
   2. 文件类型，如目录（d），文件（-），字符型文件（c），块设备（b）
   3. 文件的权限
   4. 文件的硬链接总数
   5. 文件属主的用户名
   6. 文件属组的组名
   7. 文件的大小
   8. 文件的上次修改时间
   9. 文件名或目录名
8. 可以输入man ls查看ls命令参数
9. ls filename命令直接过滤匹配，还有更多的文件扩展匹配：
   1. ?用于匹配任意单个字符
   2. \*用于匹配任意0个或多个字符
   3. [a-z]可以匹配任意小写字母，[ab]可以匹配a和b这两个字符
   4. [!a]可以匹配a以外的任意字符

**处理文件**

1. touch命令可以**创建空文件**，可以改变文件的**修改时间**，将创建者的**用户名**作为文件的属主。touch -a命令可以仅仅更改**访问时间**。
2. cp source destination可以将源文件复制成新文件，并以destination命名
   1. cp -i可以强制shell询问是否覆盖已有文件，y覆盖，n则不处理
   2. 可以将文件复制到一个**路径**下，但要记得路径最后**加正斜线/**，否则会在最后一级目录的父目录下创建一个新文件，名字和最后一级目录相同
   3. 也可以将其他路径的文件复制到某个路径下或者当期路径下（.）
   4. cp -R可以递归地复制整个目录中的内容
   5. 同样可以使用通配符
3. 制表键自动补全允许在输入文件名或目录名时按Table键让shell帮忙将内容补充完整
   1. 没有匹配项时无法补全
   2. 只有一个匹配项时自动补全
   3. 有多个匹配项时再按一次Table键列出相关前缀的文件
4. 链接文件：可以不保存物理文件的实际副本而是**虚拟副本**。链接是目录中指向文件真实位置的占位符，Linux中有两种不同类型的文件链接：
   1. 符号链接：是一个实实在在的文件，指向存放在虚拟目录结构中某个地方的另一个文件，两个文件彼此内容并不相同。原始文件必须事先存在。
   2. 通过ln -s realfile linkfile创建符号链接
   3. 硬链接：会创建**独立**的**虚拟**文件，包含了原始文件的**信息**及**位置**，但从根本上来说是**同一个文件**。引用了硬链接文件等于引用源文件。原始文件必须事先存在。
   4. 通过ln realfile linkfile创建硬链接
   5. 只能对处于相同存储介质的文件创建硬链接，不同存储介质间只能创建符号链接
   6. 注意，不要创建符号链接的符号链接，会造成混乱的连接链。可以直接创建源文件的另一个链接，最好不用复制符号链接文件。
5. **mv**命令可以将文件和目录**移动**到另一个位置或**重命名**。
   1. Inode和时间戳都不变，mv命令只影响文件名
   2. mv -i命令强制shell对覆盖进行询问
   3. 移动目录时将把目录下所有内容移动。
6. **rm**命令可以将文件**删除，无法删除目录**
   1. 由于rm指令删除后不进回收站，加上**-i**进行提示就更重要
   2. 也可以用-f参数强制删除而不提示。

**处理目录**

1. 有些命令对文件和目录都有效（cp），有些只对文件有效（rm），有些只对目录有效
2. **mkdir**可以用来**创建目录**
   1. 默认情况只能在**当前目录**下创建单独的一个目录。
   2. 加上**-p**参数可以同时创建多个目录和子目录，对一个路径创建**缺失**的目录
3. **rmdir**可以用来**删除目录**
   1. 默认情况下，只删除空目录，目录下不能有目录或文件
   2. rmdir并没有-i进行确认，因为它只删除空目录所以并不需要
   3. 可以使用rm -ri指令对目录**递归**地询问删除，但是很繁琐
   4. rm -rf可以直接删除整个目录包括下面的所有文件，但是没有提示，很危险

**查看文件内容**

1. **file**命令能够确定文件的**字符编码**和包含的**文本信息**，如ascll text
   1. 可以用来区分文件和目录
   2. 可以查看符号链接文件链接到哪个文件上
   3. 能够确认二进制文件编译时面向的平台和需要的库
2. **cat**命令用于显示文本文件中所有数据
   1. cat -n会给所有行加上行号
   2. cat -b可以只给有文本的行加上行号
   3. cat -T会用^I替换文本中所有制表符
   4. 缺点是对大型文件，文本会在显示器上一闪而过无法返回
3. **more**命令用于显示文本文件内容，会在显示**每页**数据后停下来
   1. more和bash手册一样，空格翻译，回车翻行，上下翻页
   2. 只支持文本的基本移动
4. **less**命令能够实现文本的前后翻动和一些高级的搜索功能，是more的升级版
5. **tail**命令默认会显示文件末尾10行的内容
   1. tail -n 数字 修改显示的行数
   2. tail -f可以在**其他进程使用该文件时查看**文件内容，并保持活动状态，可以用来实时监测系统日志。
6. **head**命令默认会显示文件开头10行的内容
   1. 类似于tail，head -n 数字 修改显示的行数
   2. 文件的开头通常不会改变，因此不会像tail那样支持-f参数特性

# 更多的bash shell指令

**监测程序**

1. 图形化桌面继承了大量的应用来生成一个完整的桌面环境，因此系统中总是运行着大量的程序。一些工具能对进程进行管理。
2. **ps**指令在默认情况下只会显示运行在**当前控制台**下的属于**当前用户**的进程
   1. Linux系统使用的GNU ps命令支持三种不同类型的命令行参数
      1. Unix风格的参数，前面加单破折线-，在P66页
      2. BSD风格的参数，前面不加破折线，在P67页
      3. GNU风格的长参数，前面加双破折线--，在P69页
   2. ps -ef可以查看系统上运行的所有进程，e指定所有进程，f扩展输出：
      1. UID：启动这些进程的用户
      2. PID：进程的进程ID
      3. PPID：父进程的进程ID
      4. C：进程生命周期中的CPU利用率
      5. STIME：进程启动时的系统时间
      6. TTY：进程启动时的终端设备
      7. TIME：运行进程需要的累计CPU时间
      8. CMD：启动的进程的名称
   3. ps -l输出了更多的信息，产生一个长格式输出：
      1. F：内核分配给进程的系统标记
      2. S：进程的状态（O：正在运行；S：正在休眠；R：可运行；Z：进程已结束但父进程已不存在；T：停止）
      3. PRI:进程的优先级（数字越大优先级越低）
      4. NI：参与决定优先级
      5. ADDR：进程的内存地址
      6. SZ：假如进程被换出，所需交换空间的大致大小
      7. WCHAN：进程休眠的内核函数的地址。
   4. Unix和BSD类型的参数有很多**重叠**的地方，选择自己喜欢的格式就行了。
   5. 不同的部分参数输出：
      1. VSZ：进程在内存中的大小，以KB为单位
      2. RSS：进程未换出时占用的物理内存
      3. STAT：代表当前进程状态的双字符状态码，能比单字符更清楚表示进程状态
   6. 双字符进程状态码第一个字符和Unix风格的S列相同的值，第二个参数：
      1. <：表示该进程运行在**高**优先级上
      2. N：表示该进程运行在**低**优先级上
      3. L：表示该进程有页面**锁定**在内存中
      4. s：该进程是**控制**进程
      5. l：该进程是**多线程**的
      6. +：该进程运行在前台
   7. GNU在现有的Unix和BSD上提供了一些新的功能，比如--forest参数显示进程的层级信息，并用ASCLL码字符绘制图表，让跟踪父进程和子进程更容易。
3. ps命令的缺点是只能显示某个**特定时间点**的信息，无法查看进程**趋势**
4. **top**命令能够**实时**显示进程信息
   1. 第一行显示当前时间、系统运行时间、登录用户数、系统平均负载
   2. **平均负载**有三个值，最近1分钟的、最近5分钟的、最近15分钟的，值越大说明系统负载越高。由于进程短期**突发**活动，最近1分钟高负载很常见。
   3. Linux系统管理的要点在于定义什么程度才算高负载，这个值取决于系统的硬件配置和系统通常运行的程序、
   4. 第二行显示进程概要信息，top命令将进程叫做task：有多少进程巡行、休眠、停止或僵化（进程完成但父进程没有响应）
   5. 第三行显示了CPU的概要信息：进程属主（用户还是系统）、进程的状态（运行、空闲还是等待）和CPU利用率分成几类输出
   6. 第四行说明了系统的物理内存：总共有多少内存、当前用了多少、还有多少空闲
   7. 第五行针对系统的交换空间：总共有多少空间、当前用了多少、还有多少空闲
   8. 之后就显示了当前运行中的进程的详细列表：
      1. PID：进程的ID
      2. USER：进程的属主名
      3. PR：进程的优先级
      4. NI：进程的谦让度值
      5. VIRT：进程占用的虚拟内存总量
      6. RES：进程占用的物理内存总量
      7. SHR：进程和其他程序共享的内寸总量
      8. S：进程的状态（D代表可中断的休眠模式，R代表运行态，S代表休眠态，T代表跟踪状态，Z代表僵化状态
      9. %CPU：进程占用的CPU时间比例
      10. %MEM：进程使用的内存占可用内存的比例
      11. TIME：内存启动到目前为止的CPU时间总量
      12. COMMAND：进程所对应的命令行名称，即启动的进程名
      13. 默认情况下，top命令按照%CPU对进城进行**排序**
      14. 单字符f命令可以选择对输出排序的字段
      15. 单字符d允许修改轮询间隔
      16. 单字符q可以退出top
5. 在Linux中，进程间通过**信号**进行通信，信号就是预定好的一个消息，进程能够识别它并决定忽略还是做出反应。Linux进程信号描述在P72
6. **kill pid**命令可以通过进程ID（PID）给进程发送TERM信号（尽可能终止）
   1. kill命令只能用进程的PID，不能用命令名，所以有时不好用
   2. 要发送进程信号，必须是进程的属主或者root用户
   3. kill -s其他信号（信号名或信号值） pid可以添加其他信号
7. **killall 进程名**命令支持通过进程名结束进程
   1. 进程名支持通配符
   2. 以root身份使用该命令要小心，容易误用通配符结束重要的系统进程

**监测磁盘空间**

1. 现在的大部分Linux发行版都能自动挂载特定类型的可移动存储媒体（U盘、CD等）
2. **mount命令**默认会**输出当前系统挂载的设备列表**，命令参数在P74
   1. mount命令提供四部分信息：
      1. 媒体的设备文件名
      2. 媒体挂载到虚拟目录的挂载点
      3. 文件系统类型
      4. 已挂载媒体的访问状态
   2. 要手动在虚拟目录中挂载设备，需要以root用户身份登录或者以root用户身份运行sudo命令，mount -t type device directory
      1. type指定了磁盘被格式化的文件系统类型
         1. Vfat：Windows的长文件系统
         2. Ntfs：Windows广泛使用的高级文件系统
         3. Iso9660：标准的CD-ROM文件系统
         4. 大多数U盘会被格式化为vfat文件系统，CD必须使用ISO9660文件系统
      2. device定义了存储设备的设备文件的位置
      3. directory定义了挂载点在虚拟目录中的位置
      4. 例如，手动将U盘/dev/sdb1挂载到/media/disk：
         1. mount -t vfat /dev/sdb1 /media/disk
   3. 媒体设备挂载到虚拟目录后，root用户就有了对该设备的所有访问权限
   4. **mount -o 额外选项**，允许添加一些以逗号分隔的额外选项：
      1. ro：以只读方式挂载
      2. rw：以读写方式挂载
      3. user：允许普通用户挂载文件系统
      4. check=none：挂载文件系统时不进行完整性校验
      5. loop：挂载一个文件
3. **Umount [directory|device]命令**用于在移出设备前先**卸载设备**:
   1. 如果有任何程序在使用设备上的文件，系统不会允许卸载
4. **df命令**可以查看每个**已挂载**的**有数据**的**磁盘**的使用情况
   1. 命令输出如下：
      1. 设备的文件位置
      2. 能容纳多少个1024字节大小的块
      3. 已用了多少个1024字节大小的块
      4. 剩余多少个1024字节大小的块
      5. 已用空间所占的比例
      6. 设备挂载到了哪个挂载点上
   2. df -h命令把输出按照用户易读的形式显示，比如用M和G，不直接输出Byte
   3. df命令的输出值显示的是Linux系统认为的当前值，可能有的进程已经删除文件但还未释放，是不会计算进闲置空间的
5. **du命令**显示某个**特定目录**的磁盘使用情况，默认目录是当前目录：
   1. 每行输出左边的数值是占用的磁盘块数，右边是目录
   2. 目录时从底层往上显示的
   3. **du -c**可以显示已列出文件总的大小
   4. **du -h**可以按照用户易读的个数输出，如K、M、G
   5. **du -s**可以显示每个输出参数的总计

**处理数据文件**

1. 当有大量数据时，通常很难处理这些信息及提取有用信息。比如du命令
2. **sort**可以对数据进行**排序，sort参数表P80**
   1. 默认情况sort会把数字当做字符来执行标准的排序
   2. **sort -n**可以按照数字的标准进行排序
   3. **sort -M**用三字符月份名(如Aug)按月份排序
   4. **sort -t 字符**可以指定一个用来区分键的位置的字符
   5. **sort - k arg1 [arg2]**可以从arg1开始，到arg2结束
   6. **sort -r**将结果按照降序输出
3. **grep [options] pattern [file]**命令会在输入或指定文件中查找匹配模式的行
   1. **参数-v**可以输出所有不匹配的行
   2. **参数-n**可以显示行号
   3. **参数-c**可以只输出有多少行匹配
   4. **参数-e**可以指定多个匹配模式，在每个-e后面键入模式即可
   5. 匹配模式可以使用基本的Unix风格的正则表达式
4. **egrep**命令支持POSIX扩展正则表达式
5. **fgrep**命令支持将匹配模式指定为用换行符分割的一列固定长度的字符串，可用于文件

**压缩数据**

1. Linux文件压缩工具
   1. bzip2----.bz2----用块排序文本压缩算法和霍夫曼编码
   2. compress----.Z----最初的Unix文本压缩工具，没人用了
   3. gzip----.gz----GNU压缩工具，GNU项目的产物，最流行。
   4. Zip----.zip----windows的zip压缩的Unix实现
2. **gzip**命令用来压缩文件成gzip
   1. 可以用通配符一次性压缩多个文件
3. **gzcat**命令用来查看压缩成gzip的文本文件的内容
4. **gunzip**命令用来解压缩gzip文件

**归档数据**

1. **tar function [options] object1 object2**命令可以把输出写到文件里用来**归档**数据
   1. tar的function参数和options参数在P84
   2. tar -cvf source.tar test/ test2/可以创建source.tar归档文件，含有test和test2目录
   3. tar -tf source.tar可列出tar文件的内容
   4. tar -xvf source.tar可以提取出tar文件的内容
   5. tar -zxvf filename.tar可以对下载的.tgz结尾的用gzip压缩的开源软件进行解压。

# 理解shell

**Shell的类型**

1. cat /etc/passwd可以看到用户fenyu使用/bin/bash作为默认的shell程序
2. bash位于/bin目录内，是一个可执行程序
3. /bin/sh是默认的系统shell，用于需要在启动时使用的系统shell脚本
4. ls -l /bin/sh发现ubuntu发行版中，sh指向dash，也就是dash是系统默认shell
5. 其实用户不是必须使用默认的shell，比如bash shell，可以使用发行版中的所有可用shell，只需要输入文件名就可以启动对应的shell，比如**/bin/dash**，可以键入exit退出

**Shell的父子关系**

1. 直接启动虚拟控制器终端或运行终端仿真器时启动的shell都是**父shell**
2. 当运行**子shell**的时候是没有提示的，通常就需要ps --forest -f查看进程
3. 在生成子shell时，只有部分父进程的环境被复制到子shell环境中
4. 子shell可以在父shell中创建，也可以在另一个shell中创建
5. 可以用参数设置shell的启动方式：
   1. bash -c string：从string中读取命令并处理
   2. bash -i：启动一个能够接受用户输入的交互shell
   3. bash -l：以登录shell的形式启动
   4. bash -r：启动一个受限shell，用户会被限制在默认目录中
   5. bash -s：从标准输入中读取命令
   6. 可以用bash --help查看更多命令
6. **exit**命令可以有序的退出shell
   1. 不仅可以退出子shell，还可以退出虚拟控制台或终端仿真器

**进程列表**

1. 可以在一行中输入**一串**的shell命令，用分号；分割即可
   1. 比如pwd；ls；cd /etc；pwd；cd；pwd；ls
   2. 所有命令必须依次执行
2. 若要成为进程列表，只需加上括号，如（pwd；ls；cd /etc；pwd；cd；pwd；ls）
   1. 括号的加入使**命令列表**成为了**进程列表**，生成一个**子shell**执行对应命令
   2. 可以加上环境变量命令echo $BASH\_SHELL知道是否生成了子shell，1是0否
3. 还可以在进程列表中嵌套括号（）创建子shell的子shell
4. 在shell脚本中，经常使用子shell进行**多进程**处理，但是子shell**成本高**，会明显拖慢处理速度。而且并非真正的多进程，因为中断控制着子shell的I/O

**别出心裁的子shell用法**

1. **sleep num**命令可以让进程等待（睡眠）指定的秒数
2. 在命令末尾加入&可以让命令进入后台模式
   1. 置入后台后会输出后台作业号和该进程号
3. **jobs**命令可以显示当前运行在后台模式的所有用户的进程
   1. jobs -l可以额外显示后台进程的PID
4. 将进程列表置入后台模式后，既可以在子shell中进行繁重的处理工作，同时可以让子shell的I/O**不会受制**于终端
5. **coproc subshell**命令表示协程，可以在后台生成子shell，并在子shell中执行命令
   1. 协程名默认为COPROC
   2. 可以通过coproc name { subshell; }更改协程名，注意花括号前后都要有空格
6. 生成子shell成本不低，嵌套子shell成本将更高

**理解shell的内建命令**

1. shell的内建命令和外键命令操作方式大不相同

**外部命令**

1. 有时也被称为文件系统命令，是存在于bash shell之外的程序，通常位于/bin，/usr/bin，/sbin或/usr/sbin中
2. 比如ps就是一个外部命令，有两种方式找到它
   1. **which shell**命令只显示外部命令文件
   2. **type -a shell**命令可以查看命令的不同实现
3. 当外部命令执行时，会创建一个**子进程**。这个过程称为衍生（forking）
4. 外部命令是有**代价**的，需要花费时间和资源设置子进程的环境。

**内建命令**

1. 内建命令和外部命令的区别在于内建命令**不需要子进程**来执行。内建命令和shell编译成了**一体**，作为shell工具的**组成部分**存在，**不需要**借助外部程序文件来运行。
2. 因为不需要衍生进程，也不需要打开程序文件，内建命令的执行速度要更快，**效率也更高**。---------------------------附录A有GNU内建命令列表。
3. 有些命令多**多种实现**，既有外部命令，也有内建命令，如echo和pwd
   1. 多于有多种实现的命令，如果要用外部命令，直接指明对应文件即可
   2. 比如pwd，直接输入/bin/pwd
4. **history**命令会**跟踪**用过的命令，可以唤回这些命令并重新使用
   1. 通常历史记录会保存最近的1000条命令。
   2. 可以设置HISTSIZE环境变量更改保存在历史记录中的命令数
   3. 命令历史记录保存在隐藏文件.bash\_history中，位于用户的主目录中。
   4. Bash的历史记录是先存放在内存中的，然后才被写入历史文件中。
   5. **history -a**命令可以强制直接将内存中的历史记录写入.bash\_history中
   6. **!!**命令可以唤出最近使用的一条命令并使用。
   7. ! **num**命令可以唤出第num行的命令并使用
5. **alias simple=’shell -args’**命令可以为shell命令创建别名
   1. 允许为常用的命令（包括参数）创建一个别名，可以将输入量大大减少
   2. **alias -p**命令查看当前已经设置的别名
   3. 定义好别名后，随时可以在shell或shell脚本中用它
   4. 由于命令别名属于内建命令，一个别名只在它被定义的shell进程中才有效，创建了一个bash子进程，在该bash中别名无效。

# 使用Linux环境变量

**什么是环境变量**

1. bash shell用一个叫环境变量的特性**存储有关shell会话和工作环境的信息**。
2. 这项特性允许你**在内存中存储数据**，以便程序或shell中运行的脚本能够更轻松地访问
3. 这也是**存储持久数据**的一种简便方法
4. 在bash shell中，环境变量分为两类：
   1. 全局变量
   2. 局部变量
5. Bash shell使用一致的专有环境变量，但不同Linux发行版经常会添加自有环境变量。

**全局环境变量**

1. 全局环境变量对于shell会话和所有生成的子shell都是可见的，这对于子shell需要获取父shell信息的情况非常有用。也就是说对父子进程来说，环境变量值都是相同的。
2. 系统环境变量基本都是用**大写字母**，区别于普通用户的环境变量。
3. **printenv 环境变量名**命令可以显示环境变量的值
4. **echo $环境变量名**命令也可以显示环境变量的值。引用环境变量时，前面要加美元$号
5. 用环境变量名做命令行参数时，前面也要加美元符$

**局部环境变量**

1. 局部环境变量只能在定义他们的进程中可见。
2. 尽管是局部的，但是和全局环境变量一样重要
3. Linux系统默认定义了标准的局部环境变量，用户也可以定义自己的局部变量，叫做用户定义局部变量。
4. **set**命令会显示为某个特定进程设置的所有环境变量，包括局部变量
   1. 但是没有专门只显示局部环境变量的命令

**设置局部用户自定义变量**

1. 一旦启动了bash shell（或执行一个shell脚本），就能创建在这个shell进程内可见的局部变量了。可以通过等号=给环境变量赋值，只可以是数字或字符串。
2. 如果要给变量赋值一个含有空格的字符串值，必须用单或双引号界定字符串的收尾。而如果没有单引号，bash shell会以为下一个词是另一个要执行的命令
3. 注意，**变量名区分大小写**
   1. 所有系统变量名均使用大写字母，这是bash shell的标准惯例
   2. 自己定义的局部环境变量应使用小写字母，避免重新定义系统环境变量
4. 局部环境变量能在进程中任何地方使用，但是如果生成子shell，在子shell中不可用
5. 类似的，如果在子shell中设置了一个局部变量，一旦退出子进程，局部变量将不可用

**设置全局环境变量**

1. 全局环境变量在不管是父进程还是子进程中都是可见的
2. 创建全局环境变量，首先要创建一个局部环境变量，然后把它导出到全局环境。
3. 导出过程通过**export命令**完成，变量名前面不需要加$
4. 修改子shell中的全局环境变量，只能改变该变量在子shell中的值，无法改变该变量本身，同时子shell也无法使用export命令设置全局环境变量。

**删除环境变量**

1. 使用**unset命令**可以删除已经存在的环境变量，变量名前面不需要加$
2. 通常来说，如果要用到环境变量，使用$；如果要操作变量，不使用$。一个例外是printenv
3. 注意，在子shell中删除一个全局环境变量，只对子shell有效。

**默认的shell环境变量**

1. bash shell与Unix Bourne shell兼容的系统环境变量在P110
2. Bash shell自有的系统环境变量在P111
3. 不是所有的默认环境变量都会在**set命令**中列出，因为并不是每一个都必须有值

**设置Path环境变量**

1. PATH环境变量定了用于进行命令和程序查找的目录，如果命令或者程序的位置没有包含在PATH变量中，那么如果不使用绝对路径，shell是没法找到的。
2. 可以将新的搜索目录添加到原来的PATH值中，通过**PATH=$PATH:<newpath>命令**，引用原来的PATH值，分号用来分割各个路径。
3. 如果希望子shell也能找到程序的位置，那么需要把修改后的PATH通过export导出
4. 常常将当前目录也加入环境变量：**PATH=$PATH:.**
5. 对PATH变量的修改只能持续到退出或者重启系统。

**定位系统环境变量**

1. 在Linux系统启动一个bash shell时，默认情况下bash会在启动文件（或叫环境文件）中查找命令，bash检查的文件取决于启动bash shell的方式

**登录shell**

1. 当登录Linux系统时，bash shell会作为登录shell启动
2. /etc/profile文件时系统上默认的bash shell主启动文件，每个用户登录时都会执行
   1. 不同的Linux发行版在这个文件里放了不同的命令
   2. Ubuntu发行版中设计了/etc/bash.bashrc的文件，包含了系统环境变量
   3. CentOS并没有/etc/bash.bashrc，但在内部导出了一些环境变量
   4. 以上两个发行版都用for语句迭代了/etc/profile.d目录下的所有文件，为Linux系统提供了一个防止特定应用程序启动文件的地方，用户登录时，shell会执行。
   5. 大部分应用都会创建两个启动文件，一个供bash shell用（.sh扩展名），一个供c shell用（.csh扩展名）
3. $HOME/.bash\_profile、$HOME/.bashrc、$HOME/.bash\_login、$HOME/.profile是针对用户的，可根据个人需求订制。
   1. 大多数Linux发行版只用到这四个启动文件中的一到两个
   2. 它们都起一个作用：提供一个用户专属的启动文件来定义该用户用到的环境变量
   3. 这四个文件都以.开头，说明是隐藏文件（不会在通常ls命令中出现），为与HOME目录下。
   4. 每个用户都可以编辑这些文件并添加自己的环境变量，这些环境变量会在每次启动bash shell会话时生效。
4. $HOME表示的是某个用户的主目录，和~的作用是一样的
5. Shell会按照以下顺序，运行第一个被找到的文件，余下的则被忽略
   1. $HOME/.bash\_profile
   2. $HOME/.bash\_login、
   3. $HOME/.profile
   4. 并没有$HOME/.bashrc，它通常是用过其他文件运行的

**交互式shell进程**

1. 如果bash shell不是登陆系统时启动的（比如在命令行提示符输入bash启动），那么启动的shell叫交互式shell
2. 交互式shell不会像登录shell一样运行，但依然提供了命令提示符来输入命令
3. 交互式shell不会访问/etc/profile文件，只会检查用户HOME目录的.bashrc文件，.bashrc文件的左右有两个：
   1. 查看/etc目录下的通用bashrc文件
   2. 为用户提供一个定制自己的命令别名和脚本函数的地方

**非交互式shell**

1. 系统执行shell脚本时用的就是这种shell
2. 它没有命令行提示符。
3. Bash shell提供了BASH\_ENV环境变量，用于查看启动一个非交互式shell进程时要执行的启动文件，如果有指定文件，shell会执行文件中的命令，通常包括shell脚本变量设置。
4. Ubuntu和CentOS发行版中，BASH\_ENV变量没有设置。
5. 有些shell脚本是通过启动一个子shell执行的，子shell可以继承父shell导出过的变量
6. 对于不启动子shell的脚本，变量已经存在于当前shell中，就算没有设置BASH\_ENV，依然可以使用当前的局部变量和全局变量。

**环境变量持久化**

1. 对全局环境变量来说，如果将新的或修改过的变量设置放在/etc/profile文件中，那么升级了所用的发行版，这个文件也会升级，原来定制过的变量设置就没有了。
2. 最好的方法时在/etc/profile.d目录中创建一个以.sh结尾的文件，把所有新的或修改过的全局环境变量设置放在这个文件中。
3. 在大多数发行版，存储个人用户永久性bash shell变量的地方是$HOME/.bashrc文件。
4. 如果设置了BASH\_ENV变量，除非它指向的是$HOME/.bashrc，否则应该将非交互式shell的用户变量放在别的地方。
5. 另外，图形化界面组成部分的环境变量可能需要在另外一些配置文件中设置。

**数组变量**

1. 环境变量可以作为**数组**使用
2. 要给环境变量设置多个值，可以把值放在括号里，值与值之间用空格分割
   1. 如variable=(arg1 arg2 arg3)
3. 数组的索引值从0开始
4. 要引用单独一个数组元素，可以通过索引，索引值用[]括起来，变量用{}括起来
   1. 如echo ${variable[2]}
5. 要显示整个数组变量，可以用星号（\*）作为索引
   1. 如echo ${variable[\*]}
6. 可以通过**unset命令**删除数组元素
   1. unset va[i]可以删除索引为i的值
   2. 但是删除后，后续的元素的索引不变，同时该索引处的输出将为空
   3. unset va可以删除整个数组
7. 由于数组会让事情很麻烦，可移植性也不好，所以在shell脚本编程中不常用

# 理解Linux系统文件权限

**Linux的安全性**

1. 完整的系统中必有一套能够保护文件免遭非授权用户浏览或修改的权限
2. Linux沿用的Unix文件权限的办法，允许用户和组根据每个文件和目录的安全性设置访问文件
3. Linux安全系统的**核心**是用户账户。每个能进入Linux系统的用户都会被分配唯一的用户账户，用户对系统中各个对象的访问权限取决于他们登录系统时用的账户。
4. 用户权限是通过创建用户时分配的UID（User ID）跟踪的。UID是数值，每个用户都有唯一的UID，但登录系统时用的不是UID，而是用户名（关联着一个密码）

**/etc/passwd文件**

1. Linux系统使用一个专门的文件/etc/passwd将用户的登录名匹配到对应的UID值
2. Root账户Linux系统的管理员，UID为0
3. Linux系统会为各种各样的功能创建不同的用户账户，这些账户叫系统账户，是系统上运行的各种服务进程访问资源用的特殊账户，所有运行在后台的服务都需要用一个系统账户登录到Linux系统上
4. 运行在Linux服务器后台的几乎所有的服务都是用自己的账户登录，即使有人攻入了某个服务，也无法访问整个系统。
5. Linux为系统账户预留了500以下的UID值，有些服务甚至要用特定的UID才能正常工作
6. 为普通用户创建账户时，大多数Linux系统会从500开始，将第一个可用UID分配
7. /etc/passwd文件字段包含的信息：
   1. 登录用户名
   2. 登录密码，都被设置成了x
   3. 用户账户的UID（用户ID，数字形式）
   4. 用户账户的GID（组ID，数字形式）
   5. 用户账户的文本描述（成为备注字段）
   6. 用户HOME目录的位置
   7. 用户的默认shell
8. 绝大多数Linux系统将用户密码保存在了/etc/shadow文件中，只用特定程序才能访问
9. 直接在文本编辑器手动管理/etc/passwd文件比较危险，可以用标准用户管理工具。

**/etc/shadow文件**

1. 只有root用户才能访问/etc/shadow文件，比/etc/passwd安全许多
2. 该文件为系统上每个账户都保存了一条记录，9个字段如下：
   1. 登录名
   2. 加密后的密码
   3. 自上次修改密码后过去的天数（自1970年1月1日起）
   4. 多少天后才能更改密码
   5. 多少天后必须更改密码
   6. 密码过期前多少天提醒用户更改密码
   7. 密码过期后多少天禁用用户账户
   8. 用户账户被禁用日期（自1970年1月1日起到当天天数）
   9. 保留字段

**添加新用户**

1. **useradd命令**可以添加新用户，一次性创建新用户账户以及设置用户HOME目录结构
2. 系统默认值被设置在/etc/default/useradd文件中，使用-D参数查看所有默认值
   1. GROUP：组ID
   2. HOME：HOME目录路径
   3. INACTIVE：过期后是否被禁用
   4. EXPIRE：是否设置过期日期
   5. SHELL：默认shell
   6. SKEL：新的HOME目录的模板，将SKEL目录下内容复制
   7. CREATE\_MAIL\_SPOOL：是否为该用户账户在mail目录下创建用于接收邮件的文件
3. 一些Linux发行版会把Linux用户和组工具放在/user/sbin目录
4. 创建新用户时，如果不在命令行指定具体的值，就会使用默认值
5. 注意，用户管理命令需要以root用户账户登录或通过sudo命令以root用户账户身份。
6. userdd命令的参数在P128页，useradd更改默认值参数在P129页
7. 可以在-D选项后面跟一个参数在加值来修改系统默认的新用户设置
   1. 如useradd -D -g 101

**删除用户**

1. **userdel命令**可以删除/etc/passwd文件中的用户信息，不会删除该账户任何文件
2. **userdel -r命令**还会删除用户的HOME目录以及邮件目录
   1. 删除HOME要小心，因为不知道是否在其中存放了其他用户的重要文件

**修改用户**

1. usermod：修改用户账户的字段，还可以指定主要组和附加组的所属关系
   1. 是账户修改工具最强大的一个，能修改/etc/passwd中大部分字段
      1. -c修改备注字段
      2. -e修改过期日期
      3. -g修改默认登录组
      4. -l修改用户账户的登录名
      5. -L锁定账户，使账户无法登录，无需删除账户和用户的数据。
      6. -p修改账户密码
      7. -U解除锁定，使用户能够登录
2. passwd：修改已有用户的密码
   1. 系统上任何用户都可以更改自己的密码
   2. 只有root用户才有权限修改别人的密码
   3. passwd -e能够强制用户下次登录时修改密码
3. chpasswd：从标准输入自动读取登录名和密码对列表，给密码加密，然后为账户设置
   1. 登录名和密码对用冒号（：）分割
   2. 可以用重定向命令将含有userid：passwd对的文件重定向给该命令
   3. 如chpasswd < users.txt
4. chsh：修改用户账户的默认登录shell
   1. 使用时必须用shell的全路径名，不能只用shell名
   2. 如chsh -s /bin/csh test
5. chfn：修改用户账户的备注信息
   1. 将用于Unix的finger命令的信息存进备注字段，或将备注字段流空
   2. finger命令可以查看Linux系统上的用户信息
      1. 出于安全性考虑，很多Linux系统管理员在系统禁用finger命令，不少Linux发行版甚至没有默认安装该命令
   3. 如果使用chfn命令时没有命令，它将询问将哪些合适的内容加进备注字段
6. chage：用来管理用户账户的有效期
   1. 参数：
      1. -d：设置上次修改密码到现在的天数
      2. -E：设置密码过期的日期
      3. -I：设置密码过期到锁定账户的天数
      4. -m：设置修改密码之间最少要多少天
      5. -W：设置密码过期前多久开始出现提醒信息
   2. 命令的日期值可以是：
      1. YYYY-MM-DD格式
      2. 代表从1970年1月1日起到该天的天数
   3. 过期的账户和锁定的账户相似，账户仍然存在，但用户无法登陆

**使用Linux组**

1. 组权限允许多个用户对系统中的对象（比如文件、目录、设备）共享一组共用权限。
2. 有的Linux发行版会创建一个组，把所有的用户都当做这个组的成员。
3. 有的Linux发行版如Ubuntu会为每个用户创建一个单独的组，更安全一些。
4. 每个组都有唯一的GID和唯一的组名

**/etc/group文件**

1. /etc/group文件包含系统上用到的每个组的信息
2. 系统账户用的组通常会分配低于500的GID值，而用户组的GID则从500开始分配
3. /etc/group有4个字段：
   1. 组名
   2. 组密码，允许非组内成员通过它临时成为该组成员。
   3. GID
   4. 属于改组的用户列表
4. 不能通过直接修改/etc/group文件来添加用户到一个组，要用usermod命令
5. 在添加用户到不同的组之前，首先得创建组
6. 在组列表中，有些组并没有列出用户，这并不是说这些组没有成员。当一个用户在/etc/passwd文件中指定某个组作为默认组时，用户账户不会再作为该组成员出现在组列表

**创建新组**

1. groupadd命令可以在系统上创建新组。
2. 在创建新组时，默认没有用户被分配到该组。groupadd命令无法将用户添加到组中，但可以用**usermod -G <groupname> <username>命令**将用户添加到组中。
3. 如果更改了已登录系统账户所属的用户组，该用户必须登出系统后再登录，组关系更改才能生效。
4. -g选项会将指定组名替换默认组。-G选项将该组添加到用户的组列表，不会影响默认组

**修改组**

1. **groupmod -g命令**可以修改已有组的GID
2. **groupmod -n命令**可以修改已有组的组名，由于所有的安全权限都是基于GID的，可以随意修改组名而不影响文件的安全性。

**使用文件权限符**

1. **ls -l命令**可以用来查看Linux系统上的文件、目录和设备的权限
2. 第一个字段表示：
   1. -代表文件
   2. d（directory）代表目录
   3. l（link）代表链接
   4. c（char）代表字符型设备
   5. b（block）代表块设备
   6. n（net）代表网络设备
3. 之后有三组三字符的编码，每一组定义了三种访问权限：
   1. r：可读
   2. w：可写
   3. x：可执行
   4. -：表示没有该种权限
4. 三组权限分别对应对象的3个安全级别：
   1. 对象的属主
   2. 对象的属组
   3. 系统的其他用户

**默认文件权限**

1. **umask命令**可以设置所创建文件和目录的默认权限
2. 八进制模式权限：第一位代表特别的安全特性，叫粘着位，后面的可以转成二进制：
   1. ---：000：0
   2. --x：001：1
   3. -w-：010：2
   4. -wx：011：3
   5. r--：100：4
   6. r-x：101：5
   7. rw-：110：6
   8. rwx：111：7
3. umask只是掩码，要把umask值从对象的全权限值减掉才是真正的权限：
   1. 对文件来说，全权限是666（都有读写的权限）
   2. 对目录来说，全权限是777（都有读写执行的权限）
4. 在大多数Linux发行版，umask值通常会设置在/etc/profile启动文件中，不过如Ubuntu等是在/etc/login.defs中设置的

**改变权限**

1. **chmod <options> <mode> <file>命令**用来改变文件或目录的安全性设置
2. mode参数可以使用八进制模式或符号模式进行安全性设置
3. 符号模式指定权限的格式为[ugoa] [+-=] [options]:
   1. 其中[ugoa]:
      1. u代表用户
      2. g代表组
      3. o代表其他
      4. a代表所有
   2. 其中[+-=]:
      1. +代表增加权限
      2. -代表移出权限
      3. =代表设置权限
   3. 其中[options]:
      1. rwx
      2. X：如果对象是目录或者已有执行权限，赋予执行权限
      3. s：运行时重新设置UID或GID
      4. t：保留文件或目录
      5. u：将权限设置为属主一样
      6. g：将权限设置为属组一样
      7. o：将权限设置为其他用户一样
4. chmod -R可以让权限的改变递归地作用到文件和子目录

**改变所属关系**

1. **chown [options] owner[.group] file命令**改变文件的属主
   1. 支持同时改变文件的属主和属组
   2. 只改变属主不需要.
   3. 同时改变属组和属主则owner.group
   4. 只改变属组则.group
   5. 如果属组和属主名相同，可直接owner.同时改变
2. **chown -R命令**可以递归地改变子目录和文件的所属关系
3. **chown -h命令**可以改变该文件的所有符号链接文件的所属关系
4. 只有root用户能改变文件的属主
5. 文件的原属组或目标属组的成员可以改变文件的数组
6. **chgrp <newGroup> <fileName>命令**可以更改文件或目录的默认属组

**共享文件**

1. **Linux上共享文件的的方法是创建组**
2. Linux为每个文件和目录存储了3个额外的信息位：
   1. 设置用户ID（SUID）：当文件被用户使用时，程序会以文件属主的权限运行
   2. 设置组ID（SGID）：对文件来说，程序会以文件属组的权限运行；对目录来说，目录中创建的新文件会以目录的默认属组作为默认属组
   3. 粘着位：进程结束后文件还驻留(黏着)在内存中
3. 四个八进制模式权限的第一个：
   1. 000：0：所有位都清零
   2. 001：1：粘着位置位
   3. 010：2：SGID位置位
   4. 011：3：SGID和粘着位置位
   5. 100：4：SUID位置位
   6. 101：5：SUID位和粘着位都置位
   7. 110：6：SUID位和SGID位都置位
   8. 111：7：所有位都置位
4. SGID也可以在符号模式下用符号s，即chmod g+s fileName
5. 要创建一个共享目录，是目录里的新文件都能沿用目录的属组：
   1. 用mkdir创建希望共享的目录
   2. 通过chgrp命令将目录的默认属组改为包含所有需要共享文件的用户的组（本人必须是组的成员）
   3. 将目录的SGID位置位，保证目录中新建文件都用共享组作为默认属组
   4. 为了让这个环境能正常工作，所有组成员都需要把umask设置为属组文件可写

# 管理文件系统

**一、探索Linux文件系统**

**基本的Linux文件系统**

1. Linux支持多种类型的文件系统管理文件和目录。每种文件系统都在存储设备上实现了**虚拟目录结构**，进特性略有不同。
2. Ext文件系统：即扩展（extended）文件系统是Linux系统引入的最早的文件系统
   1. 它为Linux提供了一个基本的类Unix文件系统：使用虚拟目录来操作硬件设备，在物理设备上按定长的块来存储数据。
   2. Ext采用名为**索引节点**的系统来存放虚拟目录中所存储文件的信息。索引节点系统在每个物理设备中创建一个单独的表（成为索引节点表）来存储这些文件的信息
   3. 存储在虚拟目录的每一个索引节点表中都有一个条目。
   4. Ext文件系统名称中的extended部分来自其跟踪的每个文件的额外数据，包括：
      1. 文件名
      2. 文件大小
      3. 文件的属主
      4. 文件的属组
      5. 文件的访问权限
      6. 指向存有文件数据的每个硬盘块的指针
   5. Linux通过唯一的数值（索引节点号）来引用索引节点表中的每个索引节点，这个值是创建文件时由文件系统分配的。文件系统通过索引节点号而不是文件全名或路径来标识文件。
3. Ext2文件系统：第二代扩展文件系统
   1. Ext2文件系统扩展了索引节点表的格式来保存系统上每个文件的更多信息
   2. Ext2的索引节点表为文件添加了创建时间值、修改时间值和最后访问时间值来帮助系统管理员追踪文件的访问情况
   3. Ext2还允许最大文件大小增加到了2TB，以容纳数据库服务器中常见的大文件
   4. Ext2文件系统还改变了文件在数据块中的存储方式：
      1. ext文件系统常见的问题是在文件写入物理设备时，存储数据用的块很容易分散在整个设备中（即出现碎片）。数据块的碎片化会降低文件系统性能，需要更长时间在存储设备查找特定文件的所有块
      2. 保存文件时，ext2文件系统通过按组分配磁盘块来减轻碎片化。通过**将数据块分组**，文件系统在读取文件时不需要为了数据块查找整个物理设备。
   5. Ext2的重大缺陷是容易在系统崩溃或断电时损坏。由于文件系统每次存储或更新文件，都要用新信息更新索引节点表。即使文件数据正常保存到了物理设备上，如果索引节点表没完成更新，ext2文件系统将不知道那个文件存在。

**日志文件系统**

1. 日志文件系统为Linux系统增加了一层安全性。它的做法是先将文件的更改写入到临时文件（日志）中，在数据成功写到存储设备和索引节点表之后，再删除对应的日志条目。
2. 如果系统在数据写入存储设备之前崩溃或断电了，日志系统下次会读取日志文件并处理上次留下的未写入的数据。
3. Linux有3中广泛使用的日志方法，每种的保护等级都不相同：
   1. 数据模式：索引节点和文件都会写入日志，丢失数据风险低，但性能差。所有写到存储设备上的数据都要写两次：第一次写入日志，第二次写入真正的存储设备
   2. 有序模式：只有索引节点数据会写入日志，但只有数据成功写入后才删除；在性能和安全之间取得了良好的折中。
   3. 会写模式：只有索引节点数据会被写入日志，但不控制文件数据何时写入；丢失数据风险高，但仍比不用日志好。
4. Ext3文件系统：直到2008年都几乎所有Linux发行版默认的文件系统
   1. 采用和ext2相同的索引节点表结构，但给每个存储设备增加了一个日志文件，以将准备写入存储设备的数据先记入日志。
   2. 默认情况，ext3文件系统用有序模式的日志功能----只将索引节点信息写入日志文件，直到数据块都被成功写入存储设备才删除。
   3. 仍缺少一些功能：如无法恢复误删的文件，没有任何内建的数据压缩功能，也不支持加密文件。
5. Ext4文件系统：现在已经是大多数流行的Linux发行版的默认文件系统，如Ubuntu
   1. 除了支持数据压缩和加密，ext4文件系统还支持一个称为区段的特性：区段在存储设备上按块分配空间，但在索引节点表中只保存起始块的位置。由于无需列出所有存储文件数据的数据块，它可以在索引节点表中节省一些空间。
   2. 还引入了块预分配技术：如果想在存储设备给一个已经知道要变大的文件预留空间，ext4文件系统可以分配所有需要用到的块，而不仅仅是目前用到的块。ext4文件系统用0填满预留的数据块，不会将它们分配给其他文件。
6. Reiser文件系统：是Reiser在2001年为Linux创建的文件系统
   1. 只支持会写日志模式----只把索引节点表数据写到日志文件，因此也是最快的日志文件系统之一。
   2. 可以在线调整已有文件系统的大小
   3. 利用**尾部压缩**技术：将一个文件的数据填进另一个文件的数据块中空白空间
7. JFS文件系统：IBM开发的最老的日志文件系统之一
   1. 采用有序日志方法----只在日志中保存索引节点表数据，直到真正的文件数据被写进存储设备时才删除它
   2. 采用基于区段的文件分配----为每个写入存储设备的文件分配一组块，减少碎片。
8. XFS文件系统：最初用于商业Unix系统，2002年被发布到了使用Linux环境的版本
   1. 采用回写模式的日志，在提供了高性能的同时也引入了一定的风险
   2. 允许在线调整文件系统的大小，类似于ReiserFS，但是只能扩大不能缩小。

**写时复制（copy-on-write）文件系统**

1. 兼顾了安全性和性能，如果要修改数据，或使用克隆和可写快照，修改过的数据并不会直接覆盖当前的数据，而是被放入文件系统的另一个位置上。即使是数据修改已经完成，之前的旧数据也不会被重写，COW文件系统渐渐流行。
2. ZFS文件系统：最早在2012年由SUN公司投入LInux产品的使用
   1. 是一个稳定的文件系统，与Resier4、Btrfs、ext4势均力敌
   2. 最大的弱项就是没有使用GPL许可，暂时无缘Linux默认的文件系统
3. Btrfs文件系统：也被称为B树文件系统，由Oracle在2007年开始研发
   1. 在Reiser4的诸多特性的基础上改进了可靠性
   2. 具有稳定性、易用性以及能够动态调整已挂载的文件系统的大小

**二、操作文件系统**

**创建分区**

1. **fdisk命令**可以帮助管理安装在系统上的任何存储设备上的分区。
   1. 它是一个交互式程序，允许输入命令逐步完成硬盘分区的操作。
   2. 要启动该命令，必须**指定要分区的存储设备的设备名**，以及**超级用户权限**。
   3. Linux采用了一种标准格式为硬盘分配设备名称
      1. 老式的IDE驱动器，使用/dev/hdx
      2. 较新的SATA驱动器和SCSI驱动器，Linux使用/dev/sdx
      3. x表示需要，从a开始类推
      4. 在格式化分区之前，最好检查是否指定了驱动器
   4. fdisk交互式命令提示符使用单字母，在P148页
2. 分区可以按主分区和扩展分区创建。
   1. 主分区可以被文件系统直接格式化
   2. 扩展分区出现的原因是每个存储设备上只能有四个分区。可以通过创建一个扩展分区，然后再扩展分区内创建逻辑分区进行扩展

文件系统剩余的看P150页

# 安装软件程序

**包管理基础**

1. Linux开发人员通过把软件打包成更易于安装的预编译包，使软件安装更方便
2. 各主流Linux发行版都采用了某种形式的包管理系统（PMS）控制软件和库的安装。
3. 包管理系统利用一个数据库纪录各种相关的内容：
   1. Linux系统上已安装了什么软件包
   2. 每个包安装了什么文件
   3. 每个已安装软件包的版本
4. 软件包存储在服务器上，可以利用本地Linux系统上的PMS工具通过互联网访问，这些服务器成为仓库（repository），可以用PMS搜索新的软件包，或者更新已安装的软件包
5. PMS的不足在于目前还没有统一的标准工具，PMS工具及相关命令在不同的Linux发行版上有很大的不同。Linux广泛使用的两种主要的PMS基础工具是dpkg和rpm。
6. 基于Debian发行版（如Ubuntu）使用的是dpkg命令，这些发行版的PMS工具也是以dpkg命令为基础的，dpkg会直接和Linux系统上的PMS交互，用来安装、管理、删除包。
7. 基于Red Hat的发行版使用的是rpm命令，能够列出已安装包，安装新包和删除已有包
8. **基于Debian的系统**

**用aptitude管理软件包**

1. 目前最常用的命令行工具为aptitude，本质上是apt工具和dpkg的前端。dpkg是软件包管理系统工具，而aptitude则是完整的软件包管理系统。
2. 使用aptitude会进入全屏模式：
   1. 使用方向键在菜单上移动
   2. Installed Packages查看已安装了什么软件包，可以看到几组软件包，每组后面括号里的数字表示这个组有多少软件包。
   3. 选择一个组后，可以看到软件包名称和版本号的列表
   4. 选中软件包后，可以看到更详细的信息，如描述、主页、大小和维护人员等。
   5. 按q可以退出全屏显示
3. 如果已经知道了系统上软件包的名称，可以用**aptitude show <package-name>**显示详情
4. 若要得到所有跟某个软件包相关的所有文件的列表，必须用**dpkg -L <packagename>**
5. 也可以反向操作，通过**dpkg --search file-name命令**查找某个决定路径文件属于哪个包
   1. 注意，使用时必须用绝对文件路径

**用aptitude安装软件包**

1. aptitude search <packge-name>命令可以找到特定软件包
   1. Search选项可以无需在packge-name周围加通配符，会隐式添加
   2. 包名字之前如果有i，说明已经安装到系统上了
   3. 包名字之前如果有p或v，说明包可用，但是还没安装
2. **aptitude install <package-name>命令**可以安装包

**用aptitude更新软件**

1. 使用**aptitude safe-grade命令**可以更新系统上所有软件包
   1. 不需要软件包名作为参数
   2. 可以妥善解决包依赖关系的问题。
2. 而**aptitude full-upgrade和aptitude dist-upgrade**不会检查包与包之间的依赖关系。

**用aptitude卸载软件**

1. 要想只删除软件包，而不删除数据和配置文件，可以用**aptitude remove命令**
2. 要删除软件包并删除相关的数据和配置文件，用**aptitude purge命令**
3. 要看软件包是否已删除，可以再用**aptitude search命令**
   1. 如果软件包名称前面是c，意味着软件删除但配置文件还在
   2. 如果软件包名称前面是p，意味着软件删除，配置文件也删除

**aptitude仓库**

1. aptitude默认的仓库位置是在安装Linux发行版时设置的，具体位置在/etc/apt/sources.list
2. Aptitude只会从这些仓库中下载文件，如果需要为PMS添加额外的软件仓库，就在这个文件中设置。通常即使其他地方有更新的版本，也应该等到出现在仓库中再安装。
3. 结构deb (deb-src) address distribution\_name packge\_type\_list
   1. deb值说明是已编译程序员
   2. deb-src值说明是源代码的源
   3. address条目是仓库的web地址
   4. distribution\_name是发行版版本的名称
   5. package\_type\_list表明仓库有什么类型的包
4. 通常软件仓库网站或各种开发包开发人员网站上都会有一行文本，可以直接复制到sources.list文件中
5. 基于RedHat的系统，在**P171页**
6. 从源码安装
7. 首先到程序的官方网站下载tarball包
8. 用**tar -zxvf命令**解包
9. 使用**cd命令**跳到那么目录中
10. 其中README文件包含了软件安装需要的操作
11. 使用**./configure命令**检查Linux系统，确保有合适的编译器，确保正确的库依赖关系。
12. 使用**make命令**构建各种二进制文件。
    1. **make命令**会编译源码，然后链接器会为这个包创建最终的可执行文件
    2. 会在编译和链接所有源码文件的过程中产生大量的输出。
    3. 从安装的目录下运行程序有些不便，会想将它安装到常用的位置上，那么可以用**sudo make install命令**，就可以直接运行了

# 使用编辑器

1. **vim编辑器**
2. vi编辑器是Unix系统最初的编辑器，使用控制台图形模式来模拟文本编辑窗口，允许查看文件中的行、在文件中移动、插入、编辑和替换文本。比较复杂，但拥有大量特性，成为Unix管理员多年来的支柱性工具。
3. GNU项目将vi编辑器移植到开源世界，并做一些改进，因此叫vi improved，即vim

**检查vim软件包**

1. readlink -f命令能够找出链接文件的最后一环
2. 在Ubuntu中，可以通过**sudo apt-get install vim命令**安装基础版的vim包
3. /user/bin/vi在基础版之前链接向tiny版，只提供少量的vim功能，之后就指向基础版了

**vim基础**

1. 只要输入**vim <fileName>命令**就可启动vim编辑器
2. 如果未指定文件名，或者文件不存在，就会开辟一块新的缓冲区来编辑；如果文件存在，就会将文件内容整个读到缓冲区来编辑
3. 如果文件内容未占据整个屏幕，非文件内容的行会放置~
4. 窗口底部显示了文件名，总行数和总字符数，如果是新文件则为New File
5. Vim编辑器有两种模式：
   1. 普通模式：初始模式，会将按键解释为命令
   2. 插入模式：在当前光标输入的每个键都插入到缓冲区
6. Vim的命令行模式可以输入额外的命令控制vim的行为，在普通模式按下冒号：可进入
7. 使用**vimtutor命令**可以学习vim的操作方法，P184是一些基本的vim操作
8. **nano编辑器，在P187页，是较简单的编辑器**
9. **emacs编辑器，在P189页**
10. **KDE系编辑器，KDE桌面使用的编辑器，在P196页**
11. **GNOME系编辑器，GNOME或Unity桌面环境使用的编辑器，在P202页**

**启动gedit**

1. gedit主编辑器窗口左侧框显示当前正在编辑的文档，可以按F9或View-Side Pane启动

**基本的gedit功能**

1. gedit采用菜单栏和工具栏来设置功能和配置设置

**设定偏好设置**

1. Edit菜单的Preferences菜单项可以设置偏好
2. 设置view偏好：
   1. Text Wrapping：决定如何处理编辑器中的长行
   2. Line Numbers：在编辑器窗口左边显示行号
   3. Current Line：高亮显示当前行
   4. Right Margin：启用右边界，可以设置显示多少列，默认80
   5. Bracket Matching：高亮显示括号对
3. 设置Editor偏好：
   1. Tab Stops：设定按下制表符时跳过的空白数，默认是8
   2. Automatic Indentation：让gedit在文本中自动为段落和代码元素缩进
   3. File Saving：打开文件时是否创建备份文件，是否一定时间间隔自动保存
4. 设置Font&Color偏好
   1. Font：选用字体和字体大小
   2. Color Scheme：选择文本、北京、选定文本、选定内容的色彩
5. 管理插件：
   1. 插件列表**P206页**
   2. 已启用的插件会打钩
   3. 一些插件会出现在Tools菜单栏，一些插件会出现在Edit菜单栏
6. **Linux还有很多编辑器如Eclipse等**

# 构建基本脚本

1. **使用多个命令**
2. shell脚本的关键在于输入多个命令并处理每个命令的结果，甚至将一个命令的结果传给另一个命令。
3. shell允许将多个命令串起来，一次执行完成
4. 如果要两个命令一起运行，可以将它们放在同一行中，彼此用分号隔开。
   1. 使用这种办法能将任意多个命令串连在一起使用
   2. 缺陷是不能超过最大命令行字符数255
   3. 缺陷是每次运行之前，都必须在命令提示符下键入整个命令

**创建shell脚本文件**

1. 必须在文件的第一行指定要是用的shell，格式为#!/bin/bash
   1. 在通常shell脚本中，井号#用作注释行，shell并不会处理shell脚本中的注释行，但是**第一行是个例外**，#后面的惊叹号！会告诉shell用哪个shell来运行脚本
   2. 指定了shell后，就可以在文件的每一行输入一个命令，然后加一个回车符
   3. 如果需要，可以使用分号将两个命令放在一行上
   4. Shell会根据命令在文件中出现的顺序进行处理
2. 要让shell找到脚本，有两种方法：
   1. 将shell脚本文件所在目录添加到PATH环境变量中
   2. 在提示符中用绝对或相对文件路径来引用shell脚本文件
3. 为了引用当前目录下的文件，可以用**./fileName命令**
4. 由于在创建脚本文件时，umask的值决定了新文件的权限，默认只有属主和属组才有读写权限，因此用**chmod u+x fileName命令**赋予属主执行文件的权限

**显示消息**

1. echo命令是shell脚本与用户交互的重要工具：
   1. 默认情况不需要使用引号将要显示的文本划定出来
   2. 如果在文本中使用了其中一种引号，那么就需要使用另一种引号将文本划定出来
2. 可以将echo语句添加到shell脚本中任何需要显示额外信息的地方
3. **echo -n命令**可以将文本输出和命令输出显示在同一行，此时文本输出需要用引号界定
4. **使用变量**

**环境变量**

1. **set命令**可以显示一份完整的当前环境变量列表
2. 在脚本中，美元符$加上变量名可以使用这些环境变量
3. echo命令中的环境变量会在脚本运行时替换成当前值，即使放在双引号中也替换，但是如果要直接显示美元符$，可以使用转义字符/$
4. 还可以通过${variable}形式引用变量，花括号通常用来帮助识别变量名

**用户变量**

1. 定义变量允许临时存储数据并在整个脚本中使用
2. 用户变量可以由字母、数字、下划线\_**组成**，**长度**不超过20个字符。
3. 用户变量**区分大小写**。
4. 使用等号将**赋值**给用户变量。
5. Shell脚本会**自动**决定变量值的数据类型。
6. 在脚本的整个生命周期中，shell脚本中定义的变量会一直保持它们的值，脚本运行结束就被删除。
7. 和系统变量类似，用户变量也可以通过美元符$引用
8. 注意，引用一个变量赋值给另一个变量时，需要使用美元符，如var1=$var2

**命令替换**

1. shell脚本可以从命令输出中提取信息，并将其赋值给变量
2. 可以用反引号`，注意不是单引号’
   1. 比如test=`date`
3. 也可以用美元号，使用括号将命令包起来
   1. 比如test=$(date)
4. date +%y%m%d表示将日期以两位数的年月日组合的格式输出
5. 命令替换会创建一个子shell来运行对应的命令，因此由该子shell执行的命令无法使用脚本中创建的变量。如果使用内建的shell命令，并不会涉及子shell；如果使用路径./运行命令，也会创建出子shell
6. **重定向输入和输出**

**输出重定向**

1. 最基本的重定向是将命令的输出发送到一个文件中，**command > outputfile命令**
   1. 如果文件已经重载，会用新的文件覆盖已有文件
2. **comman >> outputfile命令**可以追加数据而不是覆盖数据

**输入重定向**

1. 输入重定向将文件的内容重定向到命令，**command < outputfile命令**
2. **wc命令**可以对数据中的文本进行计数，会输出三个值：
   1. 文本的行数
   2. 文本的词数
   3. 文本的字节数
3. 内联输入重定向，command << marker命令
   1. marker用来标记数据的开始和借位
   2. 数据的开始和结尾文本标记必须一致
   3. 任何字符串都可作为文本标记
   4. 次提示符会持续提示，直到输入了作为文本标记的那个字符串
4. **管道**
5. 管道连接可以将命令直接重定向到另一个命令，command1 | command2
   1. Linux系统会同时运行这两个命令，在系统内部将它们连接起来，在第一个命令产生输出的同时，输出会立即送给第二个命令
   2. 数据传输不会用到任何中间文件或缓冲区
   3. 比如set | more
   4. 管道最流行的方法之一是将命令产生的大量输出通过管道传送给**more命令**。
6. **执行数学运算**

**expr命令**

1. **expr命令**允许在命令行上处理数学表达式，但特别笨拙
2. expr命令操作符在**P223页**
3. 一些操作符会有另外的含义，比如星号\*，需要用转移\\*才能代表乘

**使用方括号**

1. 可以用美元符$和方括号将数学表达式围起来，比expr命令方便，也适用于脚本
   1. 比如var=$[1 + 5]
   2. 不用担心乘号\*等符号被误解为其他符号
   3. 可以直接在方括号内用$+变量引用其他变量

**浮点解决方案**

1. bash计算器，叫做bc，实际是一种编程语言，能够识别数字（整数和浮点数）、变量（简单变量和数组）、注释（#或/\*\*/）、表达式、编程语句、函数
   1. 使用**bc命令**访问，之后就可以进行计算了
   2. 要退出计算器，输入quit
   3. **bc -q命令**可以不显式冗长的欢迎信息进入计算器
   4. 可以用等号=对变量进行赋值
   5. 可以使用print语句打印变量和数字
2. 在脚本中使用bc：
   1. 格式为var=$(echo “options; expression” | bc)
   2. 可以在options设置变量，如果变量不止1个，用分号;分开
   3. 例如var3=$(echo “var1=2;var2=3;var1 \* var2 | bc)
   4. scale变量表示显示多少位小数，可以设置
   5. 可以使用内联重定向，在一个命令行列出多个表达式，比如：
      1. var5=$(bc << haha
      2. scale=4
      3. a1=($var1 \* $var2)
      4. b1=($var3 \* $var4)
      5. a1+b1
      6. haha
      7. )
   6. 注意，在bash计算器中创建的变量只在bash计算器有效，不能在shell脚本用
3. **退出脚本**
4. 一般运行完最后一条命令时，脚本就自然而然结束了
5. Shell中每个命令都使用退出状态码（0~255的整数）告诉shell已经运行完毕，在命令结束运行时由命令传给shell，可以捕获这个值并在脚本中使用

**查看退出状态码**

1. 专门的变量**$?**用来保存上个已执行命令的退出状态码
2. 对于需要检查的命令，必须在其运行完立即查看，否则会被下一条命令覆盖
3. 一个成功结束的命令的退出状态码是0，如果有错误，就是一个正整数
4. 退出状态码在**P229页**

**exit命令**

1. 默认情况下，shell脚本会以脚本中最后一个命令的退出状态码退出
2. exit命令允许在脚本结束时指定一个退出状态码
   1. 比如exit 5，指定退出状态码为5
   2. 也可指定参数，exit $var1
3. 超出255的状态码会对256取模

# 使用结构化命令

1. **使用if-then语句**
2. if-then语句的格式：if command \n then \n commands \n fi
   1. if语句会运行if后面的命令，如果退出状态码是0，就会执行then，如果退出状态码不是0，就不会执行then，而是执行下一条命令
   2. 运行if命令后，错误信息仍会显示在脚本的输出中
   3. 可以通过分号，将then与if放在同一行：if command; then \n commands \n fi
   4. then部分的语句可能不止一条，bash会将这些命令当成一个块，要么全部执行，要么全部跳过。
3. **使用if-then-else语句**
4. 格式为：if command \n then \n commands \n else \n commands \n fi
5. 当退出状态码为0时，执行then部分；不为0时，执行else部分
6. else部分和then部分一样，可以包含多条命令
7. **嵌套if**
8. 使用嵌套if-then语句可能使代码不易阅读，很难理清逻辑流程。
9. 可以用elif command \n then替代else，增加条件判断。
10. 紧跟elif语句的else语句，属于整个结构，而不是之前的if-then代码块
11. **test命令**
12. if-then语句不能测出命令退出状态码之外的条件
13. 如果**test condition命令**列出的条件成立，test命令就会退出并返回为0的退出状态码；如果条件不成立，test命令就会退出并返回非0的退出状态码，使得if-then语句不再被执行
14. 如果test $var，表示测试变量var是否有内容（即不为””，且已赋值）
15. 在if-then语句中也可以不用test，直接if [ condition ]定义测试条件（condition前后要有空格），效果和用test一样

**数值比较**

1. n1 -eq n2：检查n1和n2是否相等
2. n1 -ge n2：检查n1是否大于等于n2
3. n1 -gt n2：检查n1是否大于n2
4. n1 -le n2：检查n1是否小于等于n2
5. n1 -lt n2：检查n1是否小于n2
6. n1 -ne n2：检查n1是否不等于n2
7. Bash shell只能处理整数，不能处理浮点数。

**字符串比较**

1. 字符串比较测试在**P242页**
2. 字符串相等性：
   1. str1 = str2：检查str1和str2是否相同
   2. str1 != str2：检查str1和str2是否不同
   3. 比较测试会将所有的标点和大小写情况都考虑在内
3. 字符串顺序：
   1. str1 < str2：检查str1是否比str2小
   2. str1 > str2：检查str1是否比str2大
   3. 大于号和小于号必须用\转义，否则会看成重定向
   4. 大于和小于的顺序和sort命令所采用的不同
   5. 在比较测试中，使用标准的ASCII数值，大写字母被认为小于小写字母；在sort命令中，使用本地化语言设置，正好相反。
4. 字符串大小：
   1. -n str1：检查str1是否非0
   2. -z str1：检查str1是否为0
   3. 可以检查变量中是否含有数据，未定义和””都表示不含数据。

**文件比较**

1. test命令的文件比较功能在**P246页**
2. -d file：检查指定目录是否存在。打算将文件写入目录或者切换到目录，可以先进行检测。
3. -e file：检查file是否存在，可以检查文件和目录
4. -f file：检查file是否是文件
5. -r file：检查file是否可读
6. -s file：检查文件是否是非空文件，可以用在不想删除非空文件的时候
7. -w file：检查文件是否可写
8. -x file：检查文件检查文件是否可执行
9. -O file：检查所属关系，可以判断是否是文件的属主
10. -G file：检查默认属组关系，只会检查默认组
11. file -nt file2：检查file1是否比file2新
12. File -ot file2：检查file1是否比file2旧
13. **复合条件测试**
14. if-then语句允许使用布尔逻辑来组合测试，有两种可用
    1. &&，两个条件都满足时才成立
    2. ||，有一个条件满足时成立
15. **if-then的高级特性**

**使用双括号**

1. 双括号格式：(( expresion ))，expression可以是任意的数学赋值或比较表达式
   1. val++：后增
   2. ++val：先增
   3. val--：后减
   4. --val：先减
   5. !：逻辑求反
   6. ~：位求反
   7. \*\*：幂运算
   8. <<：左移
   9. >>：右移
   10. &：位布尔和
   11. |：位布尔或
   12. &&：逻辑和
   13. ||：逻辑或
2. 可以在if的判断语句中使用双括号命令，也可以在脚本的普通命令中使用来赋值

**使用双方括号**

1. 格式为：[[ expression ]]，提供了字符串比较的高级特性，有模式匹配
   1. 不是所有shell都支持双方括号
   2. 在模式匹配中，可以定义一个正则表达式
2. **case命令**
3. 格式为：

case vaiable in

pattern1 | pattern2) commands1;;

pattern3) commands2;;

\*) commands;;

esac

1. 如果variable与模式匹配，就会执行功能指定的命令块
2. 可以通过竖线|在一行中分割出多个模式
3. 星号\*会捕获所有与已知模式不匹配的值

# 更多的结构化命令

1. **for命令**
2. for命令格式：

for var in list

do

commands

done

1. 在list参数中，需要提供迭代中要用到的一系列值。第一次迭代使用列表第一个值，第二次迭代使用列表第二个值，依次类推，直到所有值都迭代完
2. 可以使用格式：for var in list; do将do语句和for语句放到同一行

**读取列表中的值**

1. 在最后一次迭代后，迭代变量的值会在shell脚本的剩余部分一直保持有效，在可以在for循环命令之外跟其他变量一样使用。
2. 它会一直保持最后一次迭代的值，除非修改了它。

**读取列表中的复杂值**

1. 如果for循环列表用到了单引号’，那么有两种解决办法
   1. 将单引号通过反斜线\转义，即\’
   2. 使用双引号””界定单引号出现的单词，即”test’s”
2. for循环假定每个值都是用空格分割的，如果单独的数据有空格，需要用双引号界定
   1. 比如”New York”
   2. 在某个值两边使用双引号时，shell不会将双引号当成值的一部分

**从变量读取列表**

1. 变量可以作为for循环的列表，只需要在for循环的列表处引用变量即可
2. 向已有列表**拼接**一个值可以直接：list=$list” add”

**从命令读取值**

1. 可以使用命令的输出来替代for循环的列表，通过$()包围命令
   1. 比如$(cat /bin/bash)

**更改字段分隔符**

1. IFS环境变量叫做内部字段分隔符，定义了bash shell用作字段分隔符的一系列字符。
2. 默认情况下，bash shell将下列字符当做字段分隔符：
   1. 空格
   2. 制表符
   3. 换行符
3. 如果bash shell在数据中看到这些字符任意一个，都会假定表明了新数据字段的开始
4. 可以在脚本中直接改变IFS的值
   1. 比如，IFS=$’\n’，只将换行符作为分隔符
5. 如果同一个脚本中只在一个地方改变分隔符，后续任用原来分分隔符，可以：
   1. 设置临时变量保存原来的分隔符：IFS.OLD=$IFS
   2. 更改分隔符给某个地方使用：IFS=’\n’
   3. 将分隔符复原：IFS=$IFS.OLD
6. 比如IFS=$’\n’:;”可以将换行符、冒号、分号、双引号作为分隔符

**用通配符读取目录**

1. 可以用for命令自动遍历目录中的文件，此时需要在文件名或路径名中使用通配符，会强制shell使用文件扩展匹配
2. 比如/test/\*，将匹配test目录下的每个对象
3. 在Linux中，目录名和文件名包含空格是合法的，因此应该将for循环的迭代变量用双引号括起来：”$var”，防止错误产生
4. 在for命令中可以列出多个目录通配符，比如：
   1. for file in /test/\* /home/\*
5. **C语言风格的for命令**

**C语言的for命令**

1. 格式为for(( variable assignment ; condition ; iteration process ))
2. 比如：for (( a = 1; a < 10; a++ ))
3. 它使用了C语言风格的变量引用方式而不是shell风格的变量引用方式
   1. 变量赋值可以有空格
   2. 条件中的变量不以美元符$开头
   3. 迭代过程中的算式未用expr命令格式

**使用多个变量**

1. C语言风格的for命令也允许为迭代使用多个变量。循环会单独处理每个变量，可以为每个变量定义不同的迭代过程。但是只能在for循环定义一种条件。
2. 例如：for((a=1,b=10;a<=10;a++,b--))
3. **while命令**

**While的基本格式**

1. while命令的格式：

while testCommand

do

commands

done

1. testCommand可以使用普通的bash shell命令，或者test命令进行条件测试
2. while命令的关键在于指定的testCommand状态码必须**随着循环中运行的命令而改变**，只要条件成立，就会不停循环执行定义好的命令

**使用多个测试命令**

1. 可以在while语句定义多个测试命令，只有最后一个测试命令的退出状态码决定什么时候退出循环，前面的测试命令可以是普通的bash shell命令
2. 测试命令每次迭代都会执行，包括测试命令失败的最后一次迭代
3. **until命令**
4. until命令的格式：

until testCommands

do

commands

done

1. 和while相反，只有测试命令的退出状态码不为0，才会执行循环
2. 和while相似，可以在until语句中放入多个测试命令，只有最后一个决定什么时候执行定义的命令。
3. **嵌套循环**
4. 嵌套循环指可以在循环内使用任意类型的命令，包括其他循环命令
5. shell能够区分开内部循环和外部循环的各自的do和done命令
6. **循环处理文件数据**
7. 通常必须遍历存储在文件中的数据，就必须用两种技术：
   1. 嵌套循环
   2. 修改IFS环境变量
8. **控制循环**

**break命令**

1. 可以跳出任意类型的循环，包括until、while、for
2. 可以直接跳出单个循环或者内部循环
3. 可以使用**break n命令**跳出外部循环，其中n指定了要跳出的循环层级
   1. 默认情况下为1，表明跳出当前循环

**continue命令**

1. continue命令可以提前终止某次循环中的命令，跳过此次循环剩余命令，但整个循环还会继续。
2. 可以使用continue n命令指定要继续执行哪一级循环，其中n执行了要跳出的循环层级
3. **处理循环的输出**
4. 可以对循环的输出使用管道或者进行重定向
5. 比如done > output.txt会将输出重定向到output.txt文件中，而不是显示在屏幕上
6. 比如done | sort使用管道将输出结果进行排序

**查找可执行文件在P284**

**创建多个用户账户在P285**

# 处理用户输入

1. **命令行参数**
2. 向shell脚本传递数据的最基本方法使用命令行参数，允许在运行脚本时添加数据
   1. 比如./test 10 30

**读取参数**

1. bash shell会将一些称为位置参数的特殊变量分配给输入到命令行中的所有参数。
2. $0是程序名，$1是第一个参数，以此类推，直到$9是第9个参数
3. 可以在shell脚本像使用其他变量一样使用$0~9的值，shell脚本会自动分配
4. 如果要输入更多的命令行参数，每个参数用空格隔开
5. 如果要在参数值中加入空格，必须用引号（单引号或双引号），引号并非数据的一部分，只是表明数据的起止位置
6. 第九个参数之后，需要使用花括号，比如${10}

**读取脚本名**

1. 如果使用另一个命令来运行脚本，命令和脚本名会混在一起
2. 如果使用完整路径运行脚本，就会显示完整路径
3. **$(basename $0)命令**会返回不包含路径的脚本名

**测试参数**

1. 在shell脚本中使用命令行参数需要注意，如果脚本不加参数运行可能会出问题
2. 使用参数前应该先检测其中是否有数据，比如if[ -n $1 ]
3. **特殊参数变量**

**参数统计**

1. bash shell提供了一个特殊变量**$#**表示脚本运行时携带的命令行参数的个数
2. 有两种方法获取命令行最后一个参数：
   1. 通过var=$#，之后引用$var表示命令行最后一个参数
   2. 通过${!#}，表示命令行最后一个参数。但是如果没有任何参数
   3. 没有任何参数时，var和$#都返回0，但${!#}返回脚本名

**抓取所有的数据**

1. **$\***变量会将命令行上提供的所有参数当做一个单词保存，会将这些参数视为一个整体
2. **$@**变量会将命令行上的所有参数当做多个独立的单词。通过遍历得到每个参数
3. **移动变量**
4. **shift命令**能够用来移动命令行参数，轻松跳过不需要的参数。
5. 默认情况会将每个参数变量向左移动一个位置。比如$3会移到$2,$1会被删除
   1. $0因为是脚本名，是不会改变的
6. 可以用来遍历命令行参数，尤其是**不知道有多少参数的时候**。只操作第一个参数，然后移动参数，继续操作第一个数，直到为空
7. 需要注意的是，如果某个参数被移除，它的值就被丢弃了，无法恢复
8. 也可以用**shift <number>命令**移动多个位置
9. **处理选项**

**查找选项**

1. 可以像处理命令行参数一样处理命令行选项
2. 处理简单选项
   1. 可以通过case来判断某个参数是否为选项
3. 分离参数和选项
   1. shell会用双破折线--来表明选项列表结束，可以在命令行输入--，然后通过case判断，如果遇到双破折线--，则退出遍历
4. 处理带值的选项
   1. 看书**P299页**

**使用getopt命令**

1. **getopt命令**可以接受一系列任意形式的命令行选项和参数，并转成适当的格式
2. 格式为：**getopt optstring parameters**
   1. optstring定义了有效的选项字母，还定义了哪些选项字母需要参数值
   2. 首先，在optstring中列出在脚本中用到的每个命令行选项字母
   3. 然后，在每个需要参数值的选项字母后面加一个冒号
   4. 在命令运行时，会自动将复合选项，比如-cd分成两个选项-c和-d；同时插入双破折线分割选项和参数
   5. 如果指定了不再optstring中的选项，会产生错误信息
   6. **getopt -q命令**可以忽略错误信息
3. 在脚本中使用getopt
   1. 通过**set命令的双破折线--选项**将命令行参数替换成set命令的命令行值
   2. 比如set -- $(getopt -q ab:cd “$@”)
   3. 原始的命令行参数变量的值会被getopt的输出替换，通过getopt**格式化**
4. tetopt并不擅长处理带空格和引号的参数值，它会将空格当做参数分隔符。

**使用更高级的getopts**

1. 内建于bash shell，和近亲getopt很像，但多了些扩展功能。
2. 每次调用时，**只处理命令行上检测到的一个参数**。处理完所有参数，会退出并返回一个大于0的退出状态码
3. 格式为：**getopts optstring variable**
   1. 有效地选项字母都列在optstring中，如果选项有参数，就加冒号：
   2. 当前参数会保存在命令行定义的variable中
4. getopts会用到两个环境变量
   1. OPTARG保存选项后面跟的参数值
   2. OPTIND保存参数列表中getopts正在处理的参数位置
5. getopts会移除命令行选项开头的单破折线
6. 可以在参数值中包含空格，用引号包围起来就行
7. 可以将选项字母和参数值放在一起，不用加空格
8. 将命令行上未定义的选项同一输出成问号？

**五、将选项标准化，常用的命令行选项的常用含义在P305**

**六、获得用户输入**

**基本的读取**

1. **read <varlist>命令**从标准输入（键盘）或另一个文件描述符中接受输入。在收到输入后，read命令会将数据放进变量var
2. **echo -n命令**不会在字符串末尾添加换行符，允许紧跟其后输入数据
3. **read -p <”string”> var命令**可以直接在read命令中指定提示字符串
4. Read命令可以指定多个变量，输入的每个数据会顺序分配到变量中，如果变量数量不够，剩余的数据会全部分给最后的变量
5. 可以不指定任何变量，最后收到的数据会保存在特殊环境变量**REPLY**中

**超时**

1. read -t <number>命令指定了计时器，number指定等待输入的秒数。
2. 计数器过期后，read命令会返回一个非零退出状态码；否则如果有输入，就表示成功
3. **read -n<number>命令**指定输入多少个字符后自动完成并退出，比如read -n1接收单字符

**隐藏方式读取**

1. **read -s命令**将输入隐藏，使数据不会显示到屏幕上，但会赋给变量。一般用于密码

**从文件中读取**

1. 可以每次调用read命令，从文件读取一行文本。当文件没有内容时，退出并返回非零状态码
2. 常见的方法是对文件使用cat命令，将结果通过管道直接传送给含有read命令的while
   1. 如cat test | while read line

# 呈现数据

1. **理解输入和输出**
2. 有两种显示脚本输出的方式，要么全部显示，要么什么都不显示：
   1. 在显示器屏幕上显示输出
   2. 将输出重定向到文件中

**标准文件描述符**

1. Linux系统将每个对象当做文件处理，包括输入和输出进程。
2. Linux用文件描述符来标识每个文件对象，文件描述符是一个非负整数，可以**唯一标识**会话中打开的文件。
3. 每个进程一次最多可以有9个文件描述符。bash shell保留前三个文件描述符（0、1、2）
   1. 0：STDIN，标准输入
   2. 1：STDOUT，标准输出
   3. 2：STDERR，标准错误
4. STDIN：对终端界面来说，标准输入是键盘
   1. Shell从STDIN文件描述符对应的键盘获得输入，在用户输入时处理每个字符
   2. 使用重定向<时，Linux会用重定向指定的文件来替换标准输入文件描述符
   3. 当在命令行只输入**cat命令**时，会从STDIN接受输入。输入一行，就显示一行
   4. 也可以使用STDIN重定向符号强制**cat命令**接收来自另一个文件的输入
5. STDOUT：在终端界面上，标准输出就是显示器
   1. Shell的所有输出会被定向到标准输出中，也就是显示器
   2. 可以用>输出重定向到文件中，比如ls -l > test
   3. 也可以用>>将数据追加到文件中
   4. 当命令生成错误消息时，shell未将错误消息重定向到输出文件，shell对于错误消息的处理是跟普通输出分开的
6. STDERR：shell或shell运行的程序和脚本出错时生成的错误消息都会发送到这个位置
   1. 默认情况下，STDERR文件描述符会和STDOUT文件描述符指向同样的地方，也就是默认情况，错误消息输出到显示器输出中
   2. STDERR并不会随着STDOUT的重定向而发生改变

**重定向错误**

1. 可以将错误文件描述符2放到重定向符号前，不能有空格：
   1. ls -l kk 2> test
   2. 用这种方法，就只会重定向错误信息，不会重定向普通数据
2. 同时重定向错误和数据必须用两个重定向符号
   1. ls -l test kk dd 1> test1 2> test2
   2. 可以用这种方法将脚本的正常输出和脚本生成的错误信息分离开
3. 也可以将STDERR和STDOUT的输出重定向到同一个输出文件，bash shell提供了&>
   1. ls -l test kk dd &> test
   2. 当使用&>符号时，错误消息**优先**于标准输出显示，以便于集中浏览错误信息。
4. **在脚本中重定向输出**

**临时重定向**

1. 可以在脚本中将单独的一行输出重定向到STDERR，必须在文件描述符数字前加&
   1. 比如，echo “this is error” >&2
   2. 脚本的STDERR文件描述符所指向的位置将显示错误文本

**永久重定向**

1. 可以使用**exec命令**告诉shell脚本在执行期间重定向某个特定文件描述符
   1. 比如，exec 1> testerr
2. exec命令会启动一个新的shell并将文件描述符重定向到文件
3. 永久重定向后，同样还可以使用临时重定向
4. 不过一旦重定向了STDOUT或STDERR，就很难将它们重定向**回原来的位置**
5. **在脚本中重定向输出**
6. **exec命令**允许将STDIN重定向到Linux系统上的文件上
   1. 如 exec 0< testfile
7. **创建自己的重定向**

**创建输出文件描述符**

1. 在脚本中使用exec 3> testout，然后对某些命令使用>& 3重定向到文件描述符3
2. 也可以使用exec 3>> testout，将输出追加

**重定向文件描述符**

1. 文件描述符可以相互分配，因此，要恢复已重定向的文件描述符：
   1. 将原来描述符分配给临时文件描述符：exec 1>& 3
   2. 重定向：exec 1> testout
   3. 恢复：exec 3>& 1

**创建输入文件描述符**

1. 和输出文件描述符一样，只是重定向符号反过来：
   1. 将原来描述符分配给临时文件描述符：exec 6<& 0
   2. 重定向：exec 0< testout
   3. 恢复：exec 0<& 6

**创建读写文件描述符**

1. 可以打开单个文件描述符作为输入和输出
   1. exec 3<> testfile
2. 但是使用时要小心：shell会维护一个内部指针，指明在文件中的当前位置，任何读或写都会从文件指针上次的位置开始。

**关闭文件描述符**

1. 如果创建了新的输入或输出文件描述符，shell会在脚本退出时自动关闭它们
2. 有些特殊情况，仍需手动关闭文件描述符，将它重定向到特殊符号&-：
   1. exec 3>&-
3. 一旦关闭了文件描述符，就不能在脚本中向它写入任何数据，否则会生成错误信息
4. 注意，如果随后在脚本中打开了同一个输出文件，shell会用一个新文件替代已有文件，如果输出数据，就会覆盖已有文件。
5. **列出打开的文件描述符**
6. lsof命令会列出整个Linux系统打开的所有文件描述符
7. 这个功能有争议，因为它会向非系统管理员用户提供Linux系统的信息。因此许多Linux系统隐藏了该命令。
8. 该命令会产生大量输出，会显示当前Linux系统上打开的每个文件的有关信息，包括后台运行的所有进程以及登录到系统的任何用户
9. **lsof -p <PID> -d <HID>命令**可以通过-p指定进程编号，通过-d指定文件描述符编号
   1. 其中，可以用特殊的环境变量$$表示当前的PID
   2. 文件描述符编号用逗号分开
10. lsof默认输出七列信息
    1. COMMAND：正在运行的命令的前9个字符
    2. PID：进程的ID
    3. USER：进程属主的登录名
    4. FD：文件描述符及访问类型
    5. TYPE：文件的类型（CHR字符型，BLK块型，DIR目录，REG常规文件）
    6. DEVICE：设备的设备号
    7. SIZE：文件大小
    8. NODE：本地文件的节点号
    9. NAME：文件名
11. **阻止命令输出**
12. 有时候**不想显示脚本的输出**，这在将脚本作为后台进程运行时很常见
13. 如果不想显示错误信息，可以将STDERR重定向代一个叫做**null**的特殊文件，shell输出到null文件的任何数据都不会保存，全部被丢掉了
14. 在Linux系统上null的标准位置是/dev/null，重定向到该位置的任何数据都会被丢弃
15. 可以在输入重定向将/dev/null作为输入文件，快速清除现有文件的数据
    1. 如：cat /dev/null > test
16. **创建临时文件**
17. Linux使用/tmp目录存放不需要永久保留的文件。大多数Linux发行版配置了在启动时自动删除/tmp目录的所有文件。这个特性允许创建临时文件而不用操心清理工作
18. 系统上的任何用户账户都有权限在读写/tmp目录中的文件。
19. **mktemp命令**可以在/tmp目录创建一个唯一的临时文件，shell来创建，但不用默认的umask。它会将文件的读写权限分配给文件的属主，并将你设置为属主，一旦创建该文件，你有完整的读写权限，而其他用户无法访问（root除外）

**创建本地临时文件**

1. 默认情况下，mktemp命令会在本地目录创建一个文件。只要指定一个文件名模板就行了，文件名任意，末尾加上六个X

**在/tmp目录创建临时文件**

1. **mktemp -t命令**会强制在系统临时目录创建临时文件
2. 使用这个特性，会返回全路径，而不是只有文件名

**创建临时目录**

1. **mktemp -d命令**创建临时目录而不是临时文件
2. 这样就能在目录进行任何需要的操作，比如创建其他临时文件
3. **记录消息**
4. **tee <fileName>命令**相当于管道的一个T型接头，将STDIN过来的数据同时发往两处，一处是STDOUT，另一处是tee命令指定的文件名
5. 注意，tee命令会在每次使用时覆盖输出文件内容；如果想追加，使用**tee -a <fillName>**
6. **实例见P329**

# 控制脚本

1. **处理信号**
2. Linux利用信号与运行在系统中的进程进行通信，可以通过对脚本进行编程，使其在收到特定信号时执行某些命令，从而控制shell脚本的操作

**重温Linux信号**

1. Linux系统和应用程序可以生成超过30个信号
2. 常见信号有：
   1. 信号：1；值：SIGHUP；描述：挂起进程
   2. 信号：2；值：SIGINT；描述：终止进程
   3. 信号：3；值：SIGQUIT；描述：停止进程，停止为shell分配CPU处理时间
   4. 信号：9；值：SIGKILL；描述：无条件终止进程
   5. 信号：15；值：SIGTERM；描述：尽可能终止进程
   6. 信号：17；值：SIGSTOP；描述：无条件停止进程，但不是终止
   7. 信号：18；值：SIGTSTP；描述：停止或暂停进程，但不是终止
   8. 信号：19；值：SIGCONT；描述：继续运行停止的进程
3. 默认情况下，bash shell会忽略收到的任何SIGQUIT(3)和SIGTERM(15)信号，防止意外终止。但是会受到SIGHUP(1)和SIGINT(2)信号
4. 当收到了SIGHUP或SIGINT信号时，shell会将同样的信号传给所有由它启动的进程
5. Shell会将这些信号传给shell脚本程序处理。而shell脚本的默认行为是忽略这些信号，因为它们可能不利于脚本的运行。可以在脚本中加入识别信号的代码并处理。

**生成信号**

1. 中断进程
   1. **Ctrl+C组合键**会生成SIGINT信号，并将其发送给当前在shell中运行的所有进程。
   2. 可以运行sleep 100，然后用Ctrl+C组合键测试它，可以提前终止sleep命令
2. 暂停进程
   1. **Ctrl+Z组合键**会生成一个SIGTSTP信号，停止shell中运行的任何进程
   2. 停止和终止进程不同：
      1. 停止会让程序继续保留在内存中，并能从上次停止的位置继续执行
   3. Shell将shell中运行的每个进程称为作业，并为每个作业分配唯一的作业号，从1开始分配，依次类推。
   4. 如果shell会话中有一个已停止的作业，在退出shell时，bash会提醒你
   5. 可以用**ps -l命令**查看已停止的作业
      1. 在S（进程状态）列中，已停止作业的状态显示为T
   6. 可是用kill -9 PID命令终止已停止作业或者再输入一遍exit命令
      1. 终止作业后，下次输入命令行后shell会显示作业被终止
3. 捕获信号
   1. **trap命令**可以指定shell脚本要监视并从shell中拦截的Linux信号
   2. 如果脚本收到了trap命令中列出的信号，该信号不再由shell处理，而是本地处理
   3. 格式为**trap commands signals**
4. 捕获脚本退出
   1. **trap <commands> EXIT命令**可以捕获shell脚本的退出
   2. 如果提前退出脚本，同样能捕获
5. 修改或移出捕获
   1. 使用带有新选项的trap命令可以在脚本不同位置进行不同的捕获处理
   2. **trap -- signals命令**可以删除已设置好的捕获，单破折号-和双破折号--都可以
6. **以后台模式运行脚本**
7. 后台进程并不运行在终端显示器上，在后台运行不占用终端会话

**后台运行脚本**

1. 在运行脚本的命令行后加一个&符就可以让脚本在后台运行
   1. 比如./test &
   2. 运行后会出现如[1] 2222的字段，分别为作业号和PID
2. 当后台进程结束时，会出现如[1] Done ./test的信息表示作业号、作业号和启动命令
3. 当后台进程运行时，仍然使用终端显示器显示STDOUT和STDERR信息
4. 后台进程在显示输出的同时，仍然可以在终端运行命令。但是脚本输出和命令输出会混杂在一起，最好进行重定向避免杂乱。

**运行多个后台作业**

1. 可以在命令行提示符下同时启动多个后台作业
2. 每次启动新作业时，Linux系统会分配一个新的作业号和PID
3. 每一个后台进程都和终端会话联系在一起，如果终端会话退出，后台进程也随之退出
4. **在非控制台下运行脚本**
5. 如果想在退出终端会话后仍然让脚本在后台运行，可以用**nohup命令**
   1. 比如nohup ./test &
6. nohup命令运行了另一个命令阻止所有发送给该进程的SIGHUP信号。这会在退出终端的时候阻止进程退出，忽略发送过来的SIGUP信号
7. 为了保存nohup命令产生的输出，会自动将STDOUT和STDERR的消息重定向到nohup.out文件中
8. 如果使用nohup运行了另一个命令，该命令的所有输出都会追加到已有的nohup.out文件中。同一目录下只有一个nohup.out文件。
9. **作业控制**

**查看作业**

1. jobs命令可以查看shell当前正在处理的作业
2. jobs会显示作业号，加减号，作业状态（停止还是运行），作业命令
3. jobs命令的一些选项
   1. -l：列出进程PID
   2. -n：列出上次shell发出的通知后改变了状态的作业
   3. -p：只列出作业的PID
   4. -r：只列出运行中的作业
   5. -s：只列出已停止的作业
4. 带加号的作业会被当做默认作业。在使用作业控制命令时，如果未指定任何作业号，默认作业会被当成作业控制命令的操作对象
5. 当默认作业完成后，带减号的作业会成为下一个默认作业。任何时候都只有一个带加号的作业和一个带减号的作业

**重启停止的作业**

1. **bg <jobId>命令**可以以后台模式重启一个作业，如果不加作业后就重启默认作业
2. **fg <jobId>命令**可以以前台模式重启作业，前台模式会接管当前的工作终端
3. **调整谦让度**
4. 在多任务操作系统（如Linux），内核负责将CPU时间分配给系统运行的每个进程
5. 调度优先级是内核分配给进程的CPU时间（相对于其他进程）
6. 在Linux系统中，由shell启动的所有进程的调度优先级默认都是相同的
7. 调度优先级是整数，从-20（最高优先级）到+19（最低优先级）。默认优先级为0

**nice命令**

1. **nice命令**可以设置能够命令启动时的调度优先级
2. **nice -n <number>命令或nice -<number>命令**可以指定优先级
3. 必须将nice命令和要启动的命令放在同一行
   1. 比如：nice -n 10 ./test或nice -10 ./test
4. 使用ps命令可以查看进程，NI列列出了调度优先级
5. nice命令阻止普通系统用户提高命令的优先级

**renice命令**

1. **renice -n <number> -p <pid>命令**可以改变已运行命令的优先级
   1. 比如renice -n 10 -p 5055
2. 只能对属于自己的进程执行renice
3. 只能降低进程的优先级
4. Root用户可以任意调整进程优先级
5. **定时运行作业**

**用at命令计划执行作业**

1. at命令会将作业提交到队列中，指定shell何时运行该作业
2. at的守护进程atd会以后台模式运行，检查作业队列来运行作业，大多数Linux发行版会在启动时运行该守护进程
3. atd守护进程会检查系统上的特殊目录（通常在/var/spool/at）来获取at命令提交的作业。默认情况下，atd守护进程会每60秒检查一下这个目录，有作业时，检查设置的运行时间，如果时间和当前时间匹配，就运行此作业

**at命令的格式**

1. 格式为：**at [-f filename] time**
2. 默认情况将STDIN的输入放到队列中，可以用-f参数指定用于读取命令（脚本文件）的文件名
3. Time参数指定Linux系统何时运行该作业，如果指定的时间错过，会在第二天的这个时间运行指定的作业
4. at命令能够识别多种不同的时间格式：
   1. 标准的小时分钟格式，比如10:15
   2. AM/PM提示符，比如10:15PM
   3. 特殊可命名时间，比如now、midnight、noon、teatime等
5. 可以通过不同的日期格式指定特定的日期：
   1. 标准日期格式，比如MMDDYY、MM/DD/YY、DD.MM.YY
   2. 文本日期，比如Jul 4或Dec 25或Jul或Dec
6. 针对不同的优先级，存在26种不同的作业队列，通常由小写字母或大写字母指代，字母排序越高，优先级就越低。默认情况下会提交到a队列
7. 可以用-q参数指定不同的队列字母

**获取作业的输出**

1. 当作业在Linux系统上运行时，显示器不会关联到该作业，取而代之，Linux系统会将提交改作业的用户的电子邮箱地址作为STDOUT和STDERR
2. at命令还显示分配给作业的作业号以及为作业安排的运行时间
3. 使用e-mail作为at命令的输出极其不便。at命令利用sendmail应用程序发送邮件，如果没有安装，将无法得到任何输出。
4. 因此，使用at命令时最好在脚本中对STDOUT和STDERR进行重定向
5. 可以使用-M选项屏蔽作业产生的输出信息

**列出等待的作业**

1. **atq命令**可以查看系统中有哪些作业在等待
2. 作业列表显示了作业号、系统运行该作业日期、所在的作业队列、运行账户

**删除作业**

1. **atrm <jobId>命令**可以删除队列中指定作业号的作业

**安排需要定期执行的脚本**

1. Linux系统使用cron程序安排要定期执行的作业。cron会在后台运行并检查一个特殊表，以获知已经安排执行的作业。

**Cron时间表**

1. cron时间表采用特别的格式指定作业何时运行，格式为：
   1. min hour daofmonth month dayofweek command
2. cron时间表运行使用特定值、取值范围或者通配符来指定条目
3. 比如每天的10:15运行一个命令：15 10 \* \* \* command
4. 比如每周一的4:15PM运行一个命令：15 16 \* \* 1 command
5. 比如每个月第一天中午12点执行命令：00 12 1 \* \* command
   1. dayofmonth表项指定的月份的日期值为1~31
6. 命令列表必须指明运行的命令或脚本的全路径名
7. 由于cron程序会用提交作业的用户账户运行该脚本，因此必须有访问该命令和指定文件的权限。

**构建cron时间表**

1. 每个系统用户都可以有自己的cron时间表来运行安排好的任务
2. **crontab -l命令**可以列出已有的cron时间表
3. 默认情况下，用户的cron时间表并不存在
4. **crontab -e命令**为cron时间表添加条目，会启动一个文本编辑器。

**浏览cron目录**

1. 如果对执行时间的精度要求不高，可以直接把脚本复制到一些特殊目录下
2. /etc/cron.hourly目录下的脚本会每天执行一次
3. daily、monthly、weekly同理

**anacron程序**

1. 如果某个作业在cron时间表中安排运行的时间已到，但Linux系统处于关机状态，那么这个作业就不会被运行。当系统开机时，cron程序不会再运行那些错过的作业。
2. 如果anacron直到某个作业错过了执行时间，它会尽快运行该作业。这个功能常用于进行常规日志维护的脚本
3. anacron只会处理位于cron目录的程序，如/etc/cron.monthly
4. 使用**时间戳**决定作业是否在正确的计划间隔内运行了，每个cron目录都有一个时间戳文件，该文件位于/var/spool/anacron。
5. anacron程序使用自己的时间表（通常位于/etc/anacrontab）检查作业目录
   1. 基本格式为period delay identifier command
   2. Period条目定义多久运行一次，以天为单位。以此条目检查作业的时间戳文件
   3. Delay条目指定等待多少分钟运行错过的脚本
   4. Command条目包含了run-parts程序（负责运行目录中传给它的任何脚本）和cron脚本目录名
   5. 注意，anacron不会运行/etc/cron.hourly的脚本。
   6. Identifier条目是特别的非空字符串，如cron-weekly，用于唯一标识作业

**使用新shell启动脚本**

1. 可以将需要在shell登录时运行的脚本放在$HOME/.bash\_profile文件中
2. 每次启动一个新shell时，bash shell都会运行.bashrc文件。如果需要一个脚本在登入bash shell和启动bash shell时都运行，可以把脚本放入.bashrc文件中

# 创建函数

1. **基本的脚本函数**
2. 函数是一个脚本代码块，可以为其命名并在代码中任何位置**重用**。
3. 要在脚本中使用该代码块时，只要使用所起的函数名就行了（该过程成为调用函数）

**创建函数**

1. 第一种创建函数的格式，采用关键字function，后跟函数名：

function name {

commands

}

* 1. name属性定义了函数的唯一名称。脚本中定义的每个函数都必须有唯一的名称
  2. commands是构成函数的一条或多条bash shell命令。在调用该函数时，bash shell会按命令在函数中出现的顺序依次执行，就像在普通脚本一样

1. 第二种创建函数的格式更接近于其他编程语言：

name() {

commands

}

**使用函数**

1. 只需在脚本中像使用其他命令一样直接输入函数名即可，每次引用函数名，bash shell会找到函数的定义并执行在定义中的命令
2. 函数定义并不一定是shell脚本首先要做的事，但一定要在被调用前定义，否则报错
3. 函数名必须是唯一的，如果重定义了函数，新定义会覆盖原来函数的定义，不会报错
4. **返回值**
5. bash shell把函数当做一个小型脚本，运行结束时会返回一个退出状态码

**默认退出状态码**

1. **默认**情况下，函数的退出状态码是函数中**最后一条命令**返回的退出状态码。
2. 在函数执行结束后，可以用标准变量**$?**确定函数的退出状态码
3. 如果使用默认退出状态码，即使有指令没有正常运行，也可能因为最后一条正常返回0

**使用return命令**

1. bash shell使用**return <code>命令**退出并返回特定的退出状态码，由code指定状态码
2. 要注意退出状态码必须是0~255。
3. 如果在使用$?提取返回值前执行其他命令，函数的返回值就会丢失。

**使用函数输出**

1. 可以将函数的输出保存在shell变量中，通过$()引用函数名
   1. 比如，var=$(funName)
2. **read -p <string>命令**输出的提示字符串并不会作为STDOUT输出的一部分，会忽略掉
3. **在函数中使用变量**

**向函数传递参数**

1. 可以像命令行参数一样给函数传递参数，在函数中可以使用标准的参数环境变量如$0、$1、$#
2. 传递参数时，必须将参数和函数放在同一行，如$(fun 10 10)，向fun函数传递了10和10两个参数
3. 由于函数使用特殊参数环境变量作为自己的参数，无法直接获取脚本在命令行中的参数