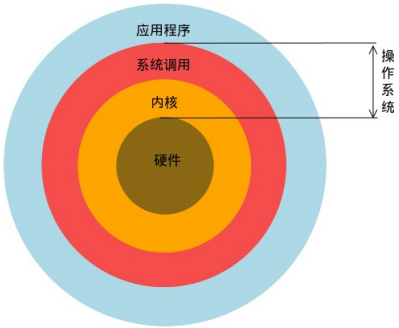
Posix
GNU
Makefile
Cmake
Bash
Shell
Powershell
BSD
Linux 包括内核、系统调用、应用程序
GNU GPL V2V3
开源软件
Tty 终端设备
Pty 虚拟设备
Pts ptm
Linux 接口
终端用户 GUI/TUICLI
程序员 system call

# Linux——操作系统介绍




# 历史背景

贝尔实验室开发 Multics （multiplexed information and compuing service）系统，但失败
Ken Thompson（C 语言之父）利用汇编语言开发了 File Server System（Unics，即 UNIX 的原型）；
发明了 C 语言，而后写出了 UNIX 的内核；

Dennis Rirchie 和 Ken Thompson 发明了 C 语言，而后贝尔实验室开发了他们自己的一个操作系统 —— UNIX 。

Bill Joy 修改了 UNIX 源码称为，BSD

UNIX 最初免费发布并因此在大学里受到欢迎。后来，UNIX 实现了 TCP/IP 协议栈。

UNIX 开始变得商业化，它的价格也变得非常昂贵。而唯一低廉的选择就是 MINIX，这是一个功能有限的类似 UNIX 的操作系统。

计划开发一个比 MINIX 性能更好的操作系统，即 LInux。于是开发出了 Linux。

Linux 是类 Unix 系统（POSIX）

包含：

用户级接口 各种管理器

程序员级接口 系统调用

适用于多种硬件平台、分布式系统和嵌入式系统的应用

Linux 的内核结构是模块结构，可以动态加载，适于嵌入式系统

Kernel.org

内核版本 双树系统（稳定树、开发树）

x.y.z y-偶数-稳定树 y-奇数-开发树

Linux 不是一个完整的操作系统

负责控制硬件设备、文件系统、进程调度等

不包括应用程序如编辑器/多媒体/网络

Linux 的发行版是完整的操作系统

DistroWatch.com: Put the fun back into computing. Use Linux, BSD.

如 Ubuntu

GNU（GNU'S NOT UNIX）项目：

产品：GCC、Emacs、Bash Shell、GLIBC；

倡导"自由软件"；GNU 的软件缺乏一个开放的平台运行，只能在 UNIX 上运行；

自由软件指用户可以对软件做任何修改，甚至再发行，但是始终要挂着 GPL (GENERAL PUBLIC LINCENSE)的版权；自由软件是可以卖的，但是不能只卖软件，而是卖服务、手册等；

GNU 开发一个完全的类 Unix 操作系统

GNU 的内核是 Linux，GNU/Linux

自由软件 Free Sofrware

基金会 FSF

自由和免费

自由软件关乎权利，≠免费软件

自由软件可以是商业化软件

为了避免 GNU 开发的自由软件被其它人用作专利软件，因此创建 GPL（general public lincense）版权声明

开源软件 Open Source Software

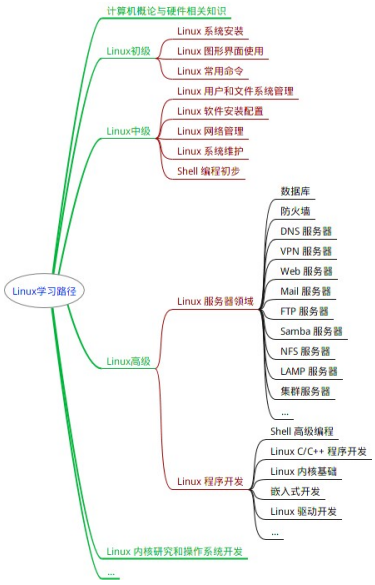
是一种软件开发方法

完全开放源代码、接受各种测试

Linux 也是开源软件

MIT 开发了 GUI,成立了研发 Xfree86

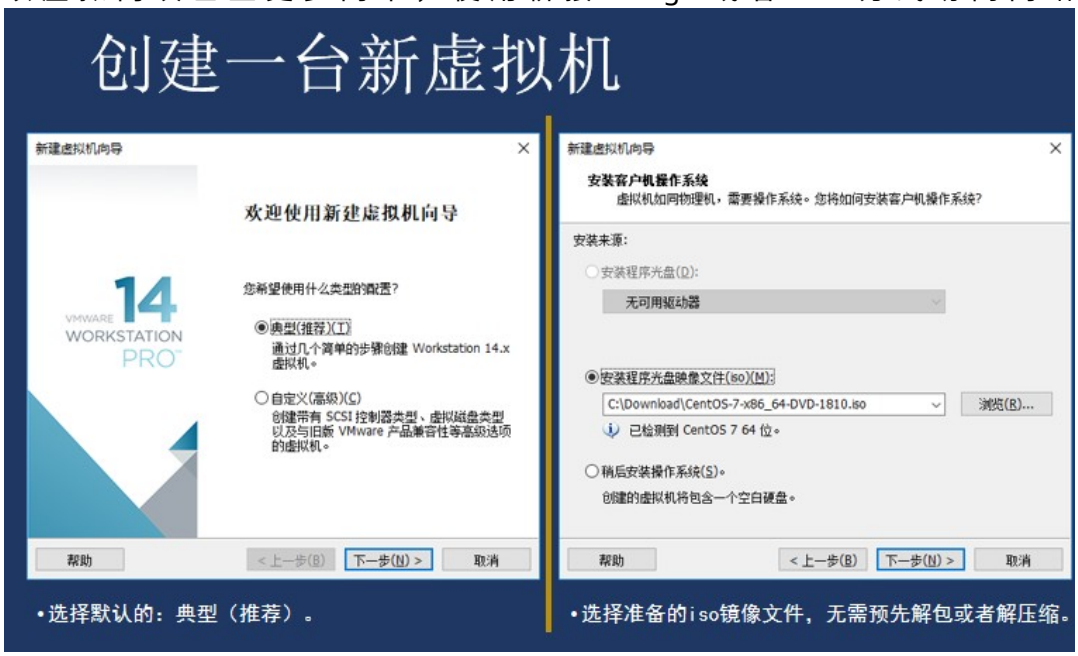
Linux 本身只是操作系统的内核。内核是使其它程序能够运行的基础。它实现了多任务和硬件管理，用户或者系统管理员交互运行的所有程序实际上都运行在内核之上。

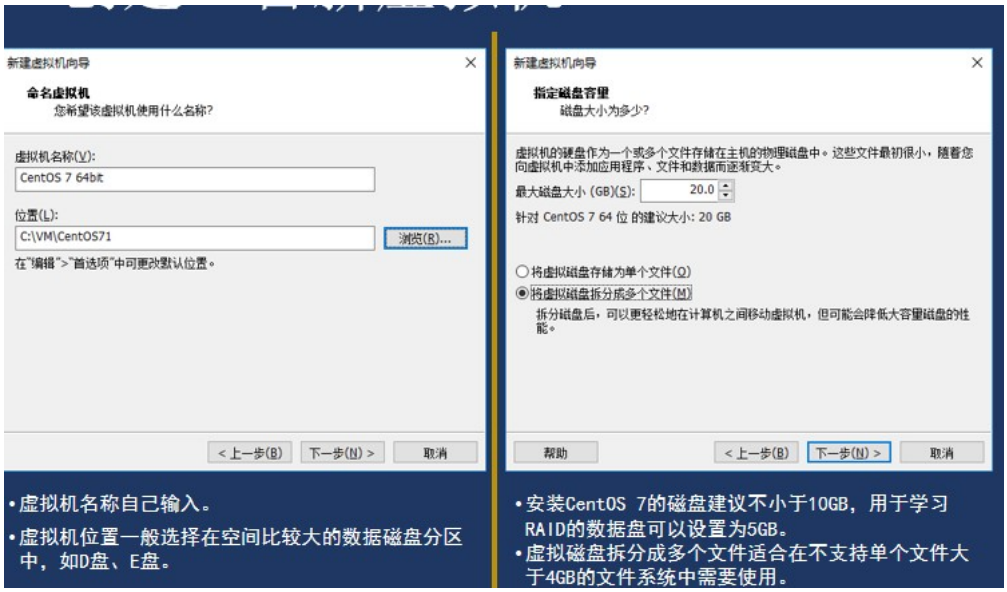



Unix 哲学强调构建简单、紧凑、清晰、模块化且可扩展的代码，便于开发者（非原始创建者）维护和复用。该哲学推崇可组合性，而非整体式设计。

# 虚拟机软件

虚拟机软件
Virtual Machine
通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统。
virtual box：是开源的轻量级虚拟机平台。完整安装包很小，功能相对也很精简。不能通过文件拖拽的方式文件共享，而是通过“映射网络驱动器”的方式与主机通过共享文件夹共享文件。
virtual PC 适合做 Windows 虚拟机。作为微软自己的产品，在 Windows 平台下使用非常方便，占用内存小，启动快，联网方面。无需桥接、NAT 等，直接作为同一子网的一台普通计算机使用，无需任何网络设置即可上网。
VMWare 软件兼容性好，快照功能很快捷方便，适合调试极易产生蓝屏的软件和工具。可以虚拟两块甚至更多网卡，使用桥接 Bridge 或者 NAT 方式访问网络。

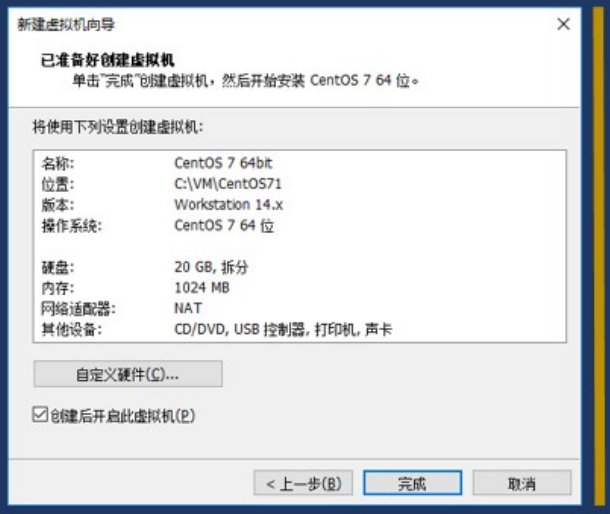





- 虚拟机名称自己输入。
- 虚拟机位置一般选择在空间比较大的数据磁盘分区中，如D盘、E盘。

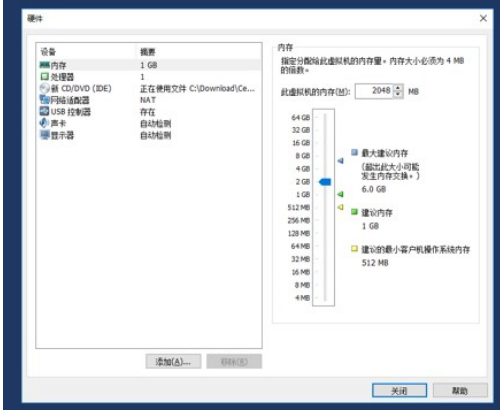
- 安装CentOS 7的磁盘建议不小于10GB，用于学习RAID的数据盘可以设置为5GB。
- 虚拟磁盘拆分成多个文件适合在不支持单个文件大于4GB的文件系统中需要使用。

# 创建一台新虚拟机



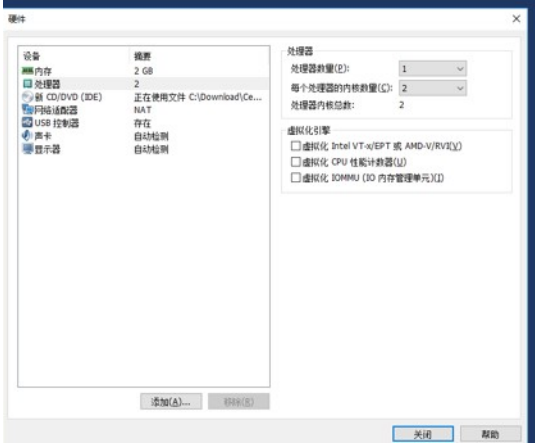

- 虚拟机的主要配置
- 通过自定义硬件修改配置
  - 内存：调整大小
  - 处理器：调整数量
  - 网络：调整连接模式
  - 其他：不需要的可以移除

# 自定义硬件——内存



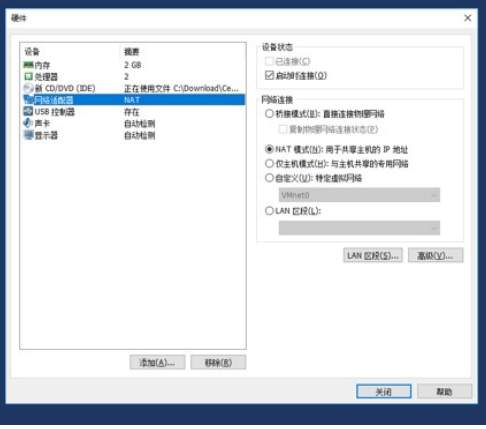

- 虚拟机配置的内存就是实际占用的内存容量
- 根据原始硬件系统配置调整
- 建议不少于2GB，1GB也可

# 自定义硬件——处理器



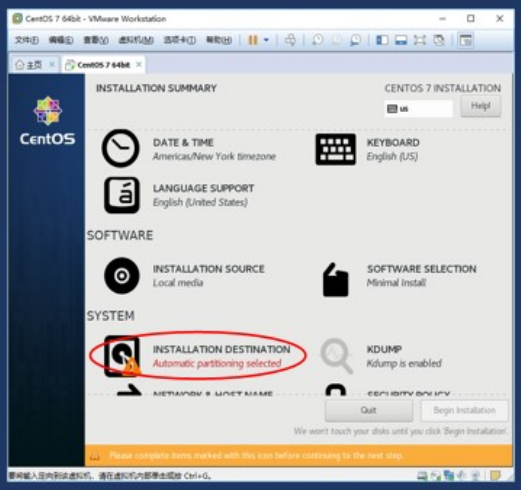

- 根据硬件配置设置
- 1CPU/2内核

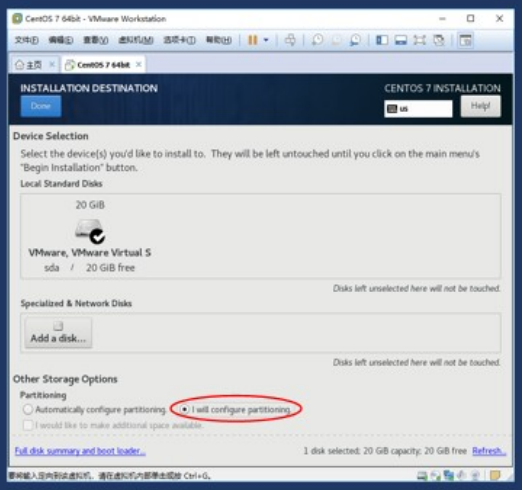
# 自定义硬件——网络适配器




- 勾选启动时连接
- 网络连接选择桥接模式或者NAT模式

# 安装目标和分区设置

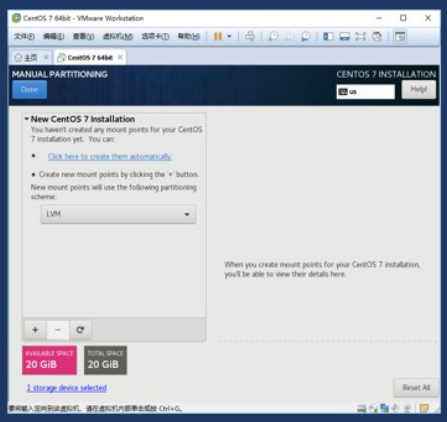


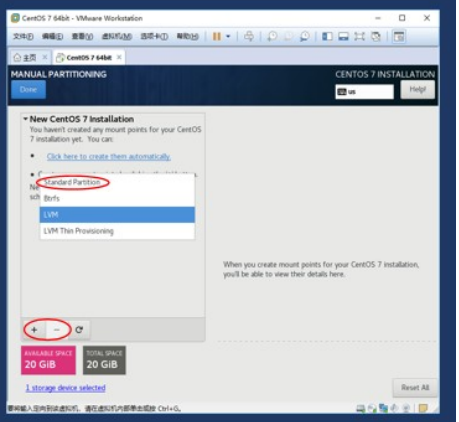




- 有黄色和红色提示的是需要进一步设置的
- 图中所示是安装目标和分区的设置

- 有自动分区和用户自定义分区两种
- 选择I will configure partitioning

# 设置分区模式和挂载点

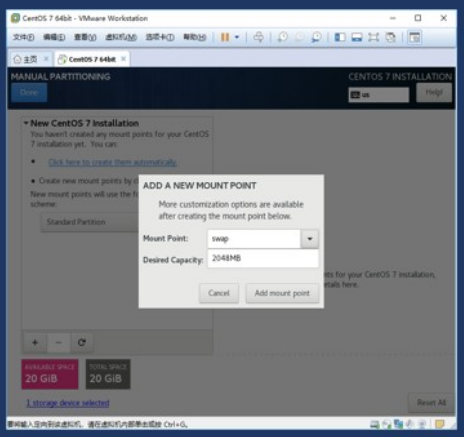


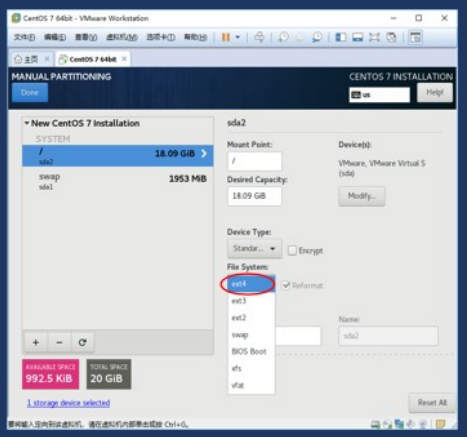




- 默认选择LVM分区模式
- 我们先选择使用标准分区模式（standard）

- 点击“+”按钮新建挂载点（mount point）
- 这就是在进行分区设置了

# 设置分区和文件系统



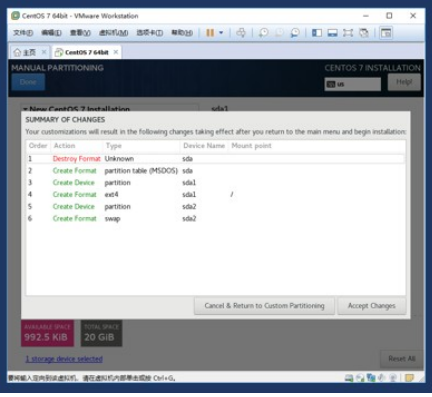




- 设置挂载点mount point
- 设置分区大小capacity
- 容量空着不填写就是分配剩余所有容量

- 至少设置两个分区：swap(交换)和/(根)
- Swap大小与内存容量相近
- 根分区选择ext4文件系统

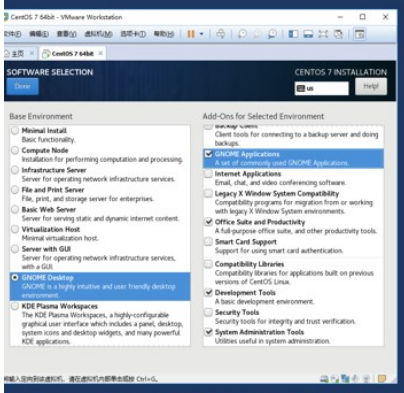


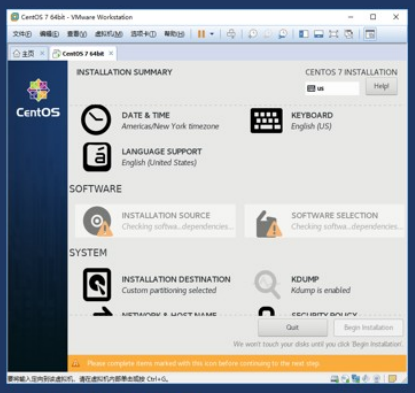
# 分区设置方案确认




- 如果有多块磁盘，可以考虑把用户/home挂载点划分到单独磁盘上
- 还可以把启动/boot单独划分分区
- 左图所示是两个分区:/和swap

# 选择服务器软件包



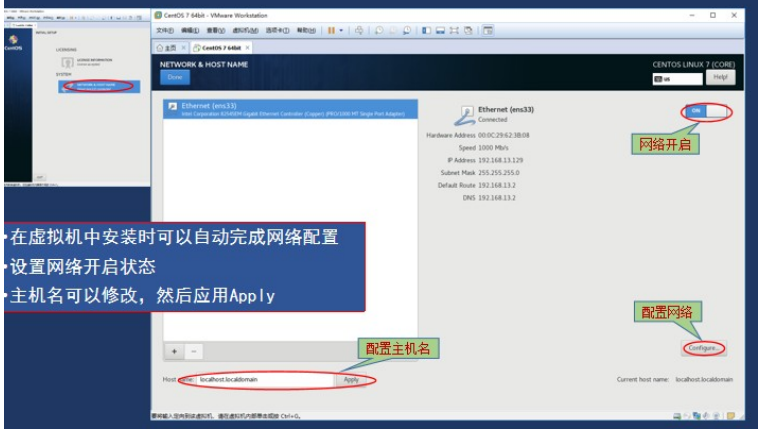

- 生产用服务器可以考虑Minimal Install
- 学习建议选用带有桌面管理器的环境
- Server with GUI 或者 GNOME Desktop

- 进行软件包依赖检查

  checking software dependencies

29

# 配置网络和主机名

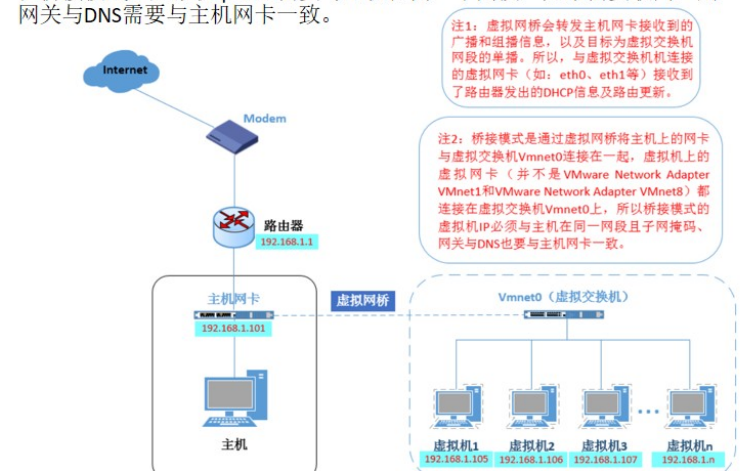


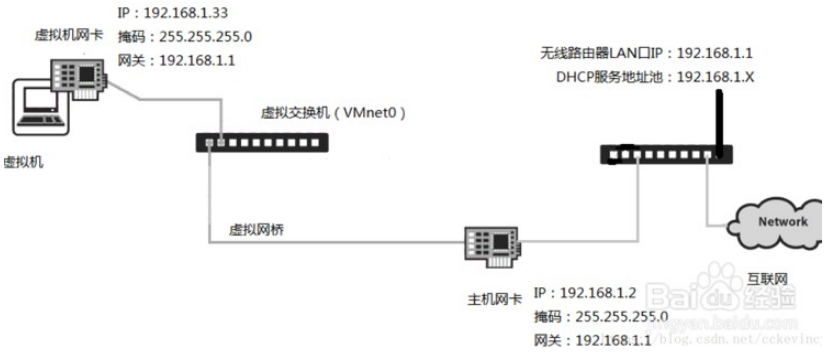

- 在虚拟机中安装时可以自动完成网络配置
- 设置网络开启状态
- 主机名可以修改，然后应用Apply

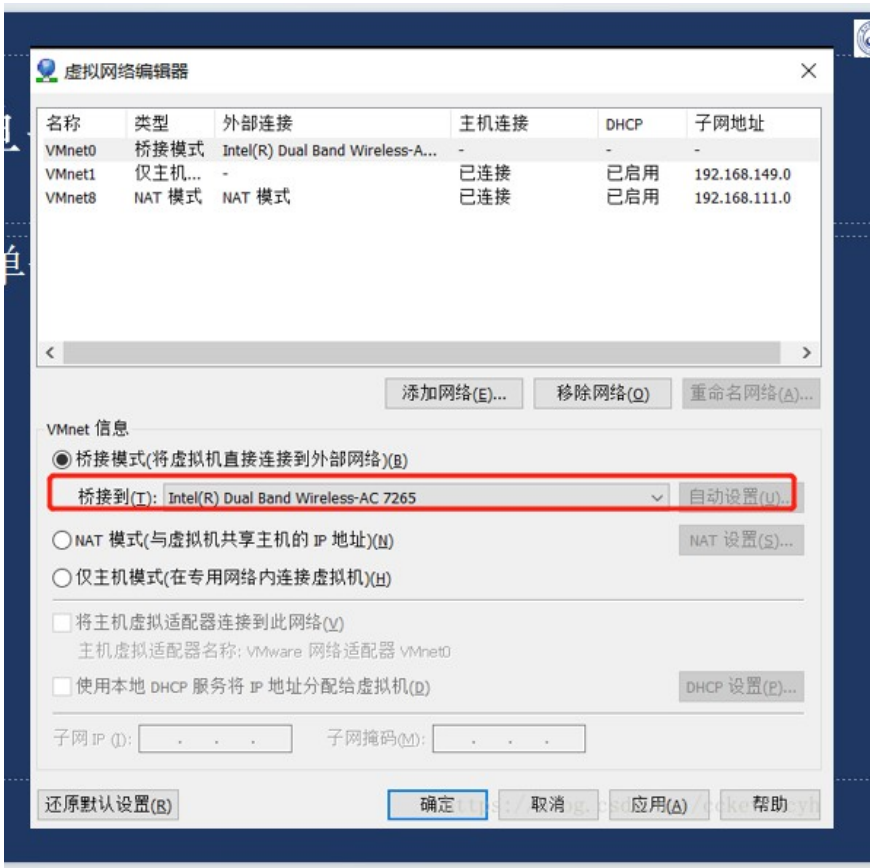
三种网络模式

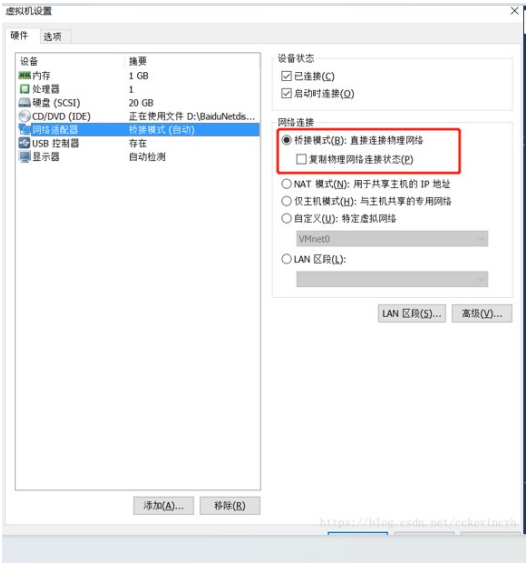
- 桥接模式（Bridged）
  - 桥接模式就是将主机网卡与虚拟机虚拟的网卡利用虚拟网桥进行通信。在桥接的作用下，类似于把物理主机虚拟为一个交换机，所有桥接设置的虚拟机连接到这个交换机的一个接口上，物理主机也同样插在这个交换机当中。
  - 桥接模式使用VMnet0。虚拟机和主机就如同插在同一台交换机上的两台主机。如果路由器开启了DHCP服务，主机和虚拟机能够自动获得IP地址。否则，需要仿照主机网卡手动配置，设置一个同网段的不同的IP地址即可。

在桥接模式，虚拟机ip地址需要与主机在同一个网段，如果需要联网，则网关与DNS需要与主机网卡一致。



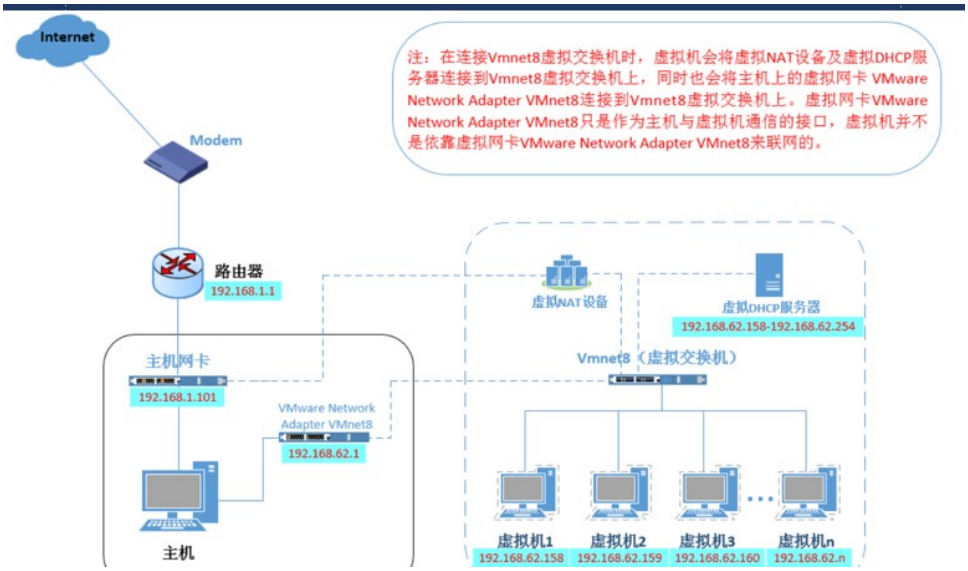






网络地址转换模式（NAT）

桥接模式配置简单，但如果网络环境是 ip 资源很缺少或对 ip 管理比较严格的话，那桥接模式就不适用了。要使用 vmware 的 NAT 模式。




在 NAT 模式中，主机网卡直接与虚拟 NAT 设备相连，然后虚拟 NAT 设备与虚拟 DHCP 服务器一起连接在虚拟交换机 VMnet8 上，这样就实现了虚拟机联网。

VMware Network Adapter VMnet8 虚拟网卡主要是为了实现主机与虚拟机之间的通信。

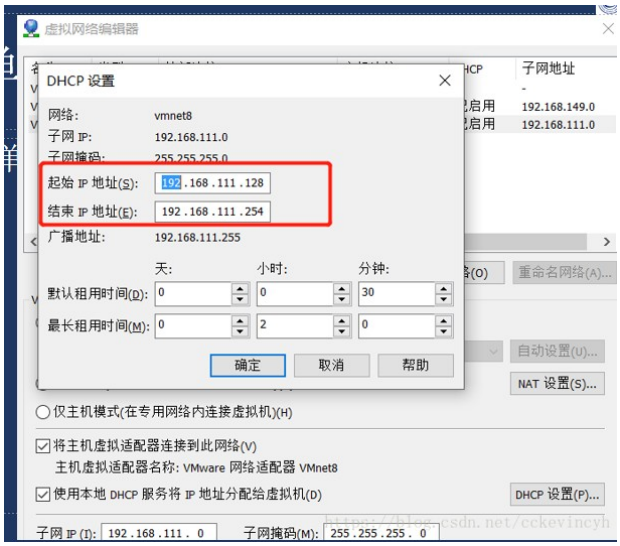
设置虚拟机中 NAT 模式的选项

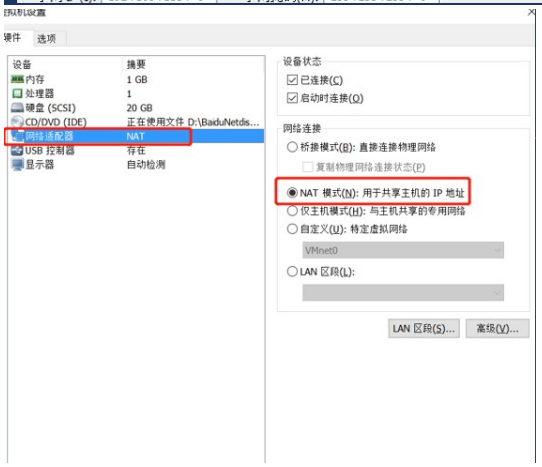
打开 vmware，点击“编辑”下的“虚拟网络编辑器”

设置 NAT 参数及 DHCP 参数。


虚拟网络编辑器
名称 类型 外部连接 主机连接 DHCP 子网地址
VMnet0 桥接模式 自动桥接 - - -
VMnet1 仅主机... - 已连接 已启用 192.168.149.0
VMnet8 NAT 模式 NAT 模式 已连接 已启用 192.168.111.0
1
添加网络(E)... 移除网络(O) 重命名网络(A)...
VMnet 信息
桥接模式(将虚拟机直接连接到外部网络)(B)
桥接到(T): 自动 自动设置(U)...
NAT 模式(与虚拟机共享主机的 IP 地址)(N)
NAT 设置(S)...
2
仅主机模式(在专用网络内连接虚拟机)(H)
将主机虚拟适配器连接到此网络(V)
主机虚拟适配器名称: VMware 网络适配器 VMnet8
使用本地 DHCP 服务将 IP 地址分配给虚拟机(D)
3
DHCP 设置(P)...
子网 IP (I): 192 . 168 . 111 . 0 子网掩码(M): 255 . 255 . 255 . 0
还原默认设置(R) 确定 取消 应用(A) 帮助


NAT 设置
网络: vmnet8
子网 IP: 192.168.111.0
子网掩码: 255.255.255.0
网关 IP(G): 192 .168 .111 . 2
端口转发(F)
主机端口 类型 虚拟机 IP 地址 描述
添加(A)... 移除(R) 属性(P)
高级
允许活动的 FTP(T)
允许任何组织唯一标识符(O)
UDP 超时(以秒为单位)(U): 30
配置端口(C): 0




# 学习工具

在线查看 Linux 内核代码
http://https://elixir.bootlin.com/linux/latest/source
http://lxr.linux.no
下载某版本内核源码文件
Kernel.org
解压缩到本地文件系统目录
使用工具阅读源码
Source insight

## Vim+ctags

在 vim 中安装插件 ctags 后，就可以在终端方便地使用 vim 命令查看 Linux 源码内容了。
安装 ctags 插件
方法一：ctags 源码安装
方法二：安装工具 apt-get、yum、pacman 等
修改 vim 配置文件

编辑 vim 配置文件 sudo vim /etc/vim/vimrc
添加如下内容：
set  tags=/home/stu01/src/linux-4.12/tags
set  autochdir
利用 ctags 文件查看 Linux 源码信息



Ctrl+]
当光标处于要查看的变量名、数据结构名、函数名处时，Ctrl+]可以跳转到相应的定义处
Ctrl+T
返回前一次的位置处
ta
在命令模式下使用，可以显示变量等的定义
:ta  normal_prio  // 显示 normal_prio()的源码
其他用法
man  ctags
Vim 中 help ctags

Linux 中的软件发布
二进制包
rmp
deb
源码包
tar
tar.gz
zip

rar

# Rpm包

- 红帽包管理工具（Red hat Package Manager）
- 安装

```
rpm -ivh /home/user01/zsh-5.0.2-31.el7.x86_64.rpm
```

- RPM安装常用选项：
  - -i：安装（Install）软件。
  - -U：升级（Upgrade）旧版本的软件。
  - -e：移除/删除（Erase）软件。
  - -v：显示详细的处理信息。
  - -h：显示安装进度。卸载不能用

如何查看安装的shell:

```
#cat /etc/shells
/bin/sh
/bin/bash
/usr/bin/sh
/usr/bin/bash
/bin/zsh
```

- rpm -q 查询

```
rpm -q zsh
```

  - 常与下面参数组合使用

-a 查询所有已经安装的软件包
-f 查询 文件所属哪个软件包,反向查询
-i 显示已经安装的rpm软件包信息,后面直接跟包名
-l 查询软件包中,文件安装的位置
-p 查询未安装软件包的相关信息,后面要跟软件的命名
-R 查询软件包的依赖性

```
rpm -qa | grep vi   #不需要后缀
rpm -qf /usr/sbin/ifconfig   #查询ifconfig属于哪个安装包（配合which）
```

- 卸载包

```
rpm -e  dhcp   #卸载包（包名）
```

# Deb包

- Debian-based系统的安装包(.exe)
- 一般是软件源repository中安装
- 有时需要下载deb包手动安装

# Deb包

- dpkg工具安装
  - 安装：sudo dpkg -i deb文件名

    sudo dpkg --install deb文件名

```
sudo dpkg -i /home/user01/teamviewer_14.5.1691_amd64.deb
```

  - 查看安装的应用：sudo dpkg -l
  - 卸载：sudo dpkg -r 软件包

    sudo dpkg -p 软件包

- apt/apt-get工具安装
  - 安装：sudo apt install 软件名

    sudo apt --fix-broken install 软件名

    GUI下可以直接双击deb文件开始安装
  - 卸载：sudo apt autoremove #卸载无用的依赖

    sudo apt remove 软件名 #软件包

# 配置yum软件源

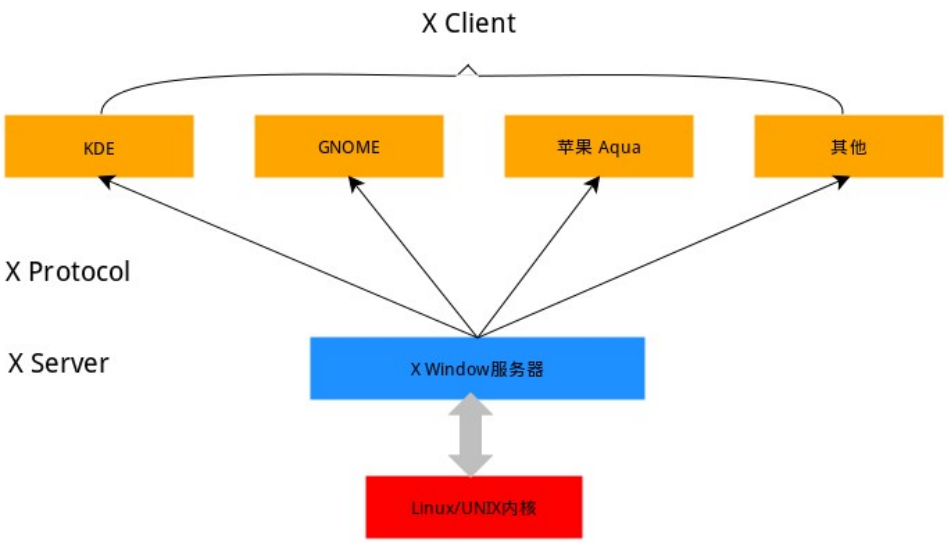
- yum (yellowdog updater modified)
- vim /etc/yum.repos.d/Local.repo #新建源
- yum clean all #清理缓存
- yum -y install dhclient

## 源码安装

- ./configure  #也可能是其他
- make  #编译
- make install  #安装
- make clean  #清理

## 终端

UNIX/Linux 本身是没有图形界面的，我们通常在 UNIX/Linux 发行版上看到的图形界面实际都只是运行在 Linux 系统之上的一套软件，
而 Linux 上的这套软件以前是 XFree86，现在则是 xorg（X.Org），而这套软件又是通过 X 窗口系统（X Window System，也常被称为 X11 或 X）实现的，X 本身只是工具包及架构协议，而 xorg 便是 X 架构规范的一个实现体，也就是说它是实现了 X 协议规范的一个提供图形界面服务的服务器，就像实现了 http 协议提供 web 服务的 Apache 。如果只有服务器也是不能实现一个完整的桌面环境的，当然还需要一个客户端，我们称为 X Client，像如下几个大家熟知也最流行的实现了客户端功能的桌面环境 **KDE**，**GNOME**，**XFCE**，**LXDE** 。




Linux/UNIX内核的X Window图形架构

# 终端

我们并不是直接利用操作系统，而是利用 shell 的中间程序来完成。在图形界面下为了实现让我们在一个窗口中完成用户输入和显示输出。

Linux 系统还提供了一个叫做终端模拟器的程序（Terminal）。 下面是几个比较常见的终端模拟器： gnome-terminal，Konsole，xterm，rxvt，kvt，nxterm 和 eterm 。

不过要注意的是这里所说的终端（Terminal）和控制台（Console）是有区别的。

Tty 终端设备的统称 teletypewriters 电传打字机
现在叫终端（terminal）
终端是一种字符型设备
Tty1-tty6 表示文字界面，可以用 alt+ctrl+F1~F6 切换，F7 是返回图形界面
Pty 虚拟终端
远程 telnet 连接到主机或使用 xterm 时，需要一个终端交互，这就是虚拟终端
pty（pseudo-tty）
Pts/ptmx
终端本质上是对应着 Linux 上的 /dev/tty 设备，Linux 的多用户登录就是通过不同的 /dev/tty 设备完成的，Linux 默认提供了 6 个纯命令行界面的 “terminal”（准确的说这里应该是 6 个 virtual consoles）来让用户登录。在物理机系统上你可以通过使用`[Ctrl]`+`[Alt]`+`[F1]`～`[F6]`进行切换。当你切换到其中一个终端后想要切换回图形界面，你可以按下`[Ctrl]`+`[Alt]`+`[F7]`来完成。

Linux 接口

终端用户 GUI

TUI/CLI

程序员

System call

## 在进行目录切换的过程中请多使用 Tab 键自动补全，可避免输入错误，连续按两次 Tab 可以显示全部候选结果

| | |
|---|---|
| **按键** | **作用** |
| `Ctrl+d` | 键盘输入结束或退出终端 |
| Ctrl+c | 终止当前程序 |

| | |
|---|---|
| **按键** | **作用** |
| Tab 键 | 补全命令、目录、参数等 |
| `Ctrl+s` | 暂停当前程序，暂停后按下任意键恢复运行 |
| `Ctrl+z` | 将当前程序放到后台运行，恢复到前台为命令 `fg` |
| `Ctrl+a` | 将光标移至输入行头，相当于 `Home` 键 |
| `Ctrl+e` | 将光标移至输入行末，相当于 `End` 键 |
| `Ctrl+k` | 删除从光标所在位置到行末 |
| `Alt+Backspace` | 向前删除一个单词 |
| `Shift+PgUp` | 将终端显示向上滚动 |
| `Shift+PgDn` | 将终端显示向下滚动 |
| `Arrow up` 向上键 | 恢复之前输入过的命令 |
| | |

## 命令程序行

这个 Shell（壳）。有壳就有核，这里的核就是指 UNIX/Linux 内核，Shell 是指“提供给使用者使用界面”的软件（命令解析器），类似于 DOS （磁盘操作系统）下的 command（命令行）和后来的 cmd.exe 。普通意义上的 Shell 就是可以接受用户输入命令的程序。它之所以被称作 Shell 是因为它隐藏了操作系统底层的细节。同样的 UNIX/Linux 下的图形用户界面 GNOME 和 KDE，有时也被叫做“虚拟 shell”或“图形 shell”。
UNIX/Linux 操作系统下的 Shell 既是用户交互的界面，也是控制系统的脚本语言。当然这一点也有别于 Windows 下的命令行，虽然该命令行也提供了很简单的控制语句。在 Windows 操作系统下，有些用户从来都不会直接使用 Shell，然而在 UNIX 系列操作系统下，Shell 仍然是控制系统启动、X11 启动和很多其它实用工具的脚本解释程序。

在 UNIX/Linux 中比较流行的常见的 Shell 有 bash、zsh、ksh、csh 等等，Ubuntu 终端默认使用的是 bash，默认的桌面环境是 GNOME 或者 Unity（基于 GNOME）。

在 linux 中，最最重要的就是命令，这就包含了 2 个过程，输入和输出

### linux 的哲学就是：没有结果就是最好的结果

## 通配符

通配符是一种特殊语句，主要有星号（*）和问号（?），用来对字符串进行模糊匹配（比如文件名、参数名）
终端里面输入的通配符是由 Shell 处理的，不是由所涉及的命令语句处理的，它只会出现在命令的“参数值”里（它不能出现在命令名称里， 命令不记得，那就用 `Tab` 补全）.总之，通配符实际上就是一种 Shell 实现的路径扩展功能。在通配符被处理后， Shell 会先完成该命令的重组，然后继续处理重组后的命令，直至执行该命令。

Shell 常用通配符：

| 字符 | 含义 |
| --- | --- |
| `*` | 匹配 0 或多个字符 |
| `?` | 匹配任意一个字符 |
| `[list]` | 匹配 list 中的任意单一字符 |
| `[^list]` | 匹配 除 list 中的任意单一字符以外的字符 |
| `[c1-c2]` | 匹配 c1-c2 中的任意单一字符 如：[0-9][a-z] |
| `{string1,string2,...}` | 匹配 string1 或 string2 (或更多) 其一字符串 |
| `{c1..c2}` | 匹配 c1-c2 中全部字符 如 {1..10} |

# 帮助命令

在 Linux 环境中，如果你遇到困难，可以使用 `man` 命令，它是 `Manual pages` 的缩写。

## Man command

Man [options] command
为了便于查找，man 手册被进行了分册（分区段）处理，在 Research UNIX、BSD、OS X 和 Linux 中，手册通常被分为 8 个区段，安排如下：

| 区段 | 说明 |
| --- | --- |
| 1 | 一般命令 |
| 2 | 系统调用 |
| 3 | 库函数，涵盖了 C 标准函数库 |
| 4 | 特殊文件（通常是/dev 中的设备）和驱动程序 |
| 5 | 文件格式和约定 |
| 6 | 游戏和屏保 |
| 7 | 杂项 |
| 8 | 系统管理命令和守护进程 |
| 9 | 其他（Linux 特定的）， 用来存放内核例行程序的文档。 |

要查看相应区段的内容，就在 man 后面加上相应区段的数字即可，如：

```
$ man 1 ls
```

所有的手册页遵循一个常见的布局，为了通过简单的 ASCII 文本展示而被优化，而这种情况下可能没有任何形式的高亮或字体控制。一般包括以下部分内容：

**NAME（名称）**

该命令或函数的名称，接着是一行简介。

**SYNOPSIS（概要）**

对于命令，正式的描述它如何运行，以及需要什么样的命令行参数。对于函数，介绍函数所需的参数，以及哪个头文件包含该函数的定义。

**DESCRIPTION（说明）**

命令或函数功能的文本描述。

OPTIONS

针对 SYNOPSIS 部分中，有列举的所有可用的选项说明

COMMANDS

当这个程序在执行的时候，可以在此程序中下达的指令

FILES

这个程序或数据所使用或参考连接到的某些文件

**EXAMPLES（示例）**

常用的一些示例。

**SEE ALSO（参见）**

相关命令或函数的列表。

也可能存在其它部分内容，但这些部分没有得到跨手册页的标准化。常见的例子包括：OPTIONS（选项），EXIT STATUS（退出状态），ENVIRONMENT（环境），BUGS（程序漏洞），FILES（文件），AUTHOR（作者），REPORTING BUGS（已知漏洞），HISTORY（历史）和 COPYRIGHT（版权）。

在 man 中使用搜索`/<你要搜索的关键字>`，查找完毕后你可以使用`n`键切换到下一个关键字所在处，`shift+n`为上一个关键字所在处。使用`Space`（空格键）翻页，`Enter`（回车键）向下滚动一行，或者使用`k`,`j`（vim 编辑器的移动键）进行向前向后滚动一行。按下`h`键为显示使用帮助（因为 man 使用 less 作为阅读器，实为`less`工具的帮助），按下`q`退出。

想要获得更详细的帮助，你还可以使用`info`命令，不过通常使用`man`就足够了。如果你知道某个命令的作用，只是想快速查看一些它的某个具体参数的作用，那么你可以使用`--help`参数，大部分命令都会带有这个参数，如：

```
$ ls --help
```

## 查看版本内核号

第一个种方法使用`uname`命令。

常用的选项是`-a`，也可以使用`-r`。使用`-a`显示信息很长，包含不仅仅内核版本，所以也可以使用`-r`就是只是显示内核版本号。可以再配以`-s`和`-m`选项，显示稍微完整一些的信息，包括操作系统和硬件系统信息。

第二种方法是查看`/proc`下的`version`文件，该文件中包含内核版本信息。因为只是一个文件，所以使用任何可以查看文件内容的方法都可以，例如使用 cat 命令。

## 命令格式一般格式：

cmd [options] [arguments]
说明：
最简单的 Shell 命令只有命令名，复杂的 Shell 命令可以有多个选项和参数。
选项和参数都作为 Shell 命令执行时的输入，它们之间用空格分隔开。
Linux 区分大小写

单字符参数前使用一个减号（-）
单词参数前使用两个减号（--）。
多个单字符参数前可以只使用一个减号。
操作对象（arguments）可以是文件也可以是目录，有些命令必须使用多个操作对象， 如

cp 命令必须指定源操作对象和目标操作对象。
并非所有命令的格式都遵从以上规则，例如 dd、find 等
命令在正常执行结果后返回一个 0 值，如果命令出错，则返回一个非零值 (在 shell 中可用变量 $? 查看)。

## 命令区别

> **内建命令**实际上是 shell 程序的一部分，其中包含的是一些比较简单的 Linux 系统命令，这些命令是写在 bash 源码的 builtins 里面的，由 shell 程序识别并在 shell 程序内部完成运行，通常在 Linux 系统加载运行时 shell 就被加载并驻留在系统内存中。而且解析内部命令 shell 不需要创建子进程，因此其执行速度比外部命令快。比如：history、cd、exit 等等。
> **外部命令**是 Linux 系统中的实用程序部分，因为实用程序的功能通常都比较强大，所以其包含的程序量也会很大，在系统加载时并不随系统一起被加载到内存中，而是在需要时才将其调入内存。虽然其不包含在 shell 中，但是其命令执行过程是由 shell 程序控制的。外部命令是在 Bash 之外额外安装的，通常放在/bin，/usr/bin，/sbin，/usr/sbin 等等。比如：ls、vi 等。
> 我们可以使用 type 命令来区分命令是内建的还是外部的。

Info
与 man 功能相似的帮助，分页显示
Info [optinons] command
help 命令是用于显示 shell 内建命令的简要帮助信息。帮助信息中显示有该命令的简要说明以及一些参数的使用以及说明，一定记住 help 命令只能用于显示内建命令的帮助信息，不然就会得到你刚刚得到的结果。其实外部命令基本上都有一个参数 command —help。

## Whatis

查找一个存储所有相关命令信息的数据库，根据命令名返回相关结果

- Usage: whatis [*options*] *command*
  - *options:* 选项（常用-d|-v|-r|-w|-l)
  - *command:* 命令名

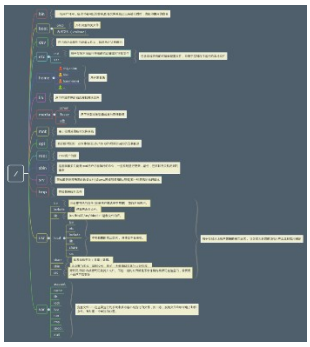
# 文件系统

一种不同是体现在目录与存储介质（磁盘，内存，DVD 等）的关系上，以往的 Windows 一直是以存储介质为主的，主要以盘符（C 盘，D 盘...）及分区来实现文件管理，然后之下才是目录。

然而 UNIX/Linux 恰好相反，UNIX 是以目录为主的，Linux 也继承了这一优良特性。 Linux 是以树形目录结构的形式来构建整个系统的，可以理解为树形目录是一个用户可操作系统的骨架。虽然本质上无论是目录结构还是操作系统内核都是存储在磁盘上的，但从逻辑上来说 Linux 的磁盘是“挂在”（挂载在）目录上的，每一个目录不仅能使用本地磁盘分区的文件系统，也可以使用网络上的文件系统。举例来说，可以利用网络文件系统（Network File System，NFS）服务器载入某特定目录等。
Tip：目录是文件系统中的一个概念，因为目录本身也是文件，叫做目录文件，简称目录。在图形用户界面中，目录被表示为文件夹。主要区别在于，文件夹是一个逻辑概念，不一定映射到物理目录。目录是文件系统对象，而文件夹是图形用户界面对象。
若指代文档的容器，使用“文件夹”一词更为贴切。而“目录”一词则指代计算机上存储文档文件和文件夹的结构化列表方式，类似于电话簿包含姓名、号码和地址的列表，但并不包含实际文档本身。
在 Windows 系统中，文件夹=虚拟文件夹+文件系统目录。

## FHS 定义

FHS 定义了两层规范，第一层是，/下面的各个目录应该要放什么文件数据，例如 /etc 应该放置设置文件，/bin 与 /sbin 则应该放置可执行文件等等。
第二层则是针对 /usr 及 /var 这两个目录的子目录来定义。例如 /var/log 放置系统日志文件，/usr/share 放置共享数据等等。



将目录定义为四种交互作用的形态，如下表所示：

| | 可分享的(shareable) | 不可分享 |
|---|---|---|
| 不可变的(static) | /usr(软件放置处) | /etc(配置 |
| | /opt(第三方软件) | /boot(开 |
| 可变动的(variable) | /var/mail(用户邮件信箱) | /var/run |
| | /var/news(新闻组) | /var/loc |

# 路径

如果你想进入某个具体的目录或者想获得某个目录的文件（目录本身也是文件）那就得用路径来找到了。
使用 cd 命令可以切换目录，在 Linux 里面使用 . 表示当前目录，.. 表示上一级目录（**注意，我们上一节介绍过的，以 . 开头的文件都是隐藏文件，所以这两个目录必然也是隐藏的，你可以使用 ls -a 命令查看隐藏文件**），- 表示上一次所在目录，～ 通常表示当前用户的 home 目录。使用 pwd 命令可以获取当前所在路径（绝对路径）。

**绝对路径**

关于绝对路径，简单地说就是以根" / "目录为起点的完整路径，以你所要到的目录为终点，表现形式如： /usr/local/bin，表示根目录下的 usr 目录中的 local 目录中的 bin 目录。

**相对路径**

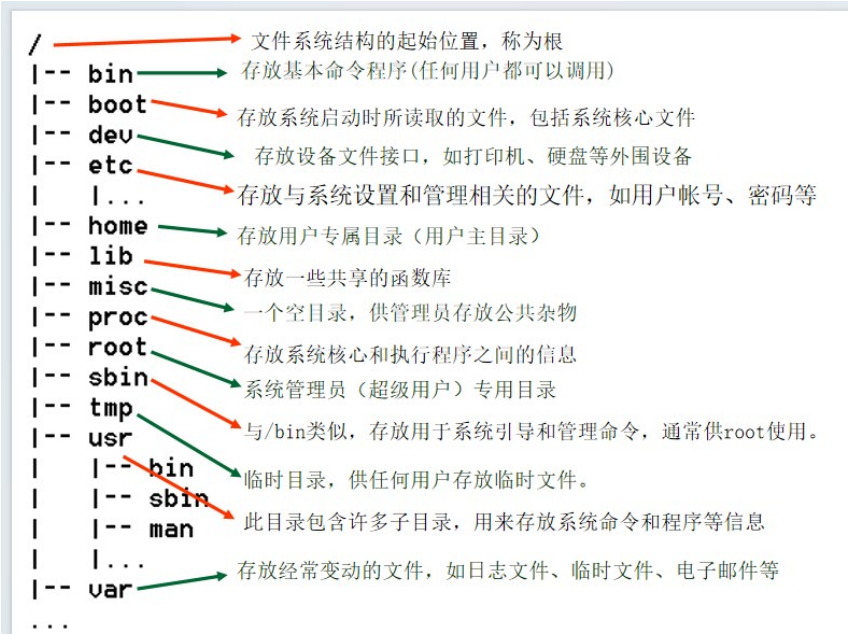
相对路径，也就是相对于你当前的目录的路径，相对路径是以当前目录 . 为起点，以你所要到的目录为终点，表现形式如： usr/local/bin （这里假设你当前目录为根目录）。你可能注意到，我们表示相对路径实际并没有加上表示当前目录的那个 . ，而是直接以目录名开头，因为这个 usr 目录为 / 目录下的子目录，是可以省略这个 . 的（以后会讲到一个类似不能省略的情况）；如果是当前目录的上一级目录，则需要使用 .. ，比如你当前目录为 /home/shiyanlou 目录下，根目录就应该表示为 ../../ ，表示上一级目录（ home 目录）的上一级目录（ / 目录）。

# 文件类型

普通文件 ( - )
目录 ( d )
符号链接 ( l )
字符设备文件 ( c )
块设备文件 ( b )
套接字 ( s )
命名管道 ( p )

普通文件仅仅就是字节序列，Linux 并没有对其内容规定任何的结构。
普通文件可以是程序源代码（c、c++、python、perl 等）、可执行文件（文件编辑器、数据库系统、出版工具、绘图工具等）、图片、声音、图像等。
Linux 不会区别对待这些普通文件，只有处理这些文件的应用程序才会根据文件的内容赋予相应的含义。
在 Linux 环境下，只要是可执行的文件并具有可执行属性它就能执行，不管其文件名后缀是什么。但是对一些数据文件一般也遵循一些文件名后缀规则。
目录文件是由一组目录项组成，目录项可以是对其他文件的指向也可以是其下的子目录指向。

一个文件的名称是存储在他的父目录中的，而并非同文件内容本身存储在一起。
硬连接文件实际上就是在某目录中创建目录项，从而使不止一个目录可以引用到同一个文件。这种链接关系由 ln 命令行来建立。
硬链接并不是一种特殊类型的文件，只是因为在文件系统中允许不止一个目录项指向同一个文件。
用户登录后，将会进入一个系统指定的专属目录，即用户的主目录，该目录名通常为用户的登录账号。如
用户 bigdata 的主目录为：/home/bigdata
在创建用户时，系统管理员会给每个用户建立一个主目录，通常在 /home/ 目录下。
用户对自己主目录的文件拥有所有权，可以在自己的主目录下进行相关操作。
每个用户名对应一个用户 ID 号（一个数字）；每个用户都被分配到一个指定的组 (group) 中。
默认情况下 RHEL/CentOS 在创建用户的同时会创建一个和用户同名的私有组。
符号链接又称软链接，是指将一个文件指向另外一个文件的文件名。
这种符号链接的关系由 ln -s 命令行来建立。
硬链接
链接文件和被链接文件必须位于同一个文件系统内
不能建立指向目录的硬链接
软链接
链接文件和被链接文件可以位于不同文件系统
可以建立指向目录的软链接
设备文件
设备是指计算机中的外围硬件装置，即除了 CPU 和内存以外的所有设备。通常，设备中含有数据寄存器或数据缓存器、设备控制器，它们用于完成设备同 CPU 或内存的数据交换。
在 Linux 下，为了屏蔽用户对设备访问的复杂性，采用了设备文件，即可以通过象访问普通文件一样的方式来对设备进行访问读写。
设备文件用来访问硬件设备，包括硬盘、光驱、打印机等。每个硬件设备至少与一个设备文件相关联。
设备文件分为：字符设备（如：键盘）和块设备（如：磁盘）。
设备的使用方法
用户可以用设备名来使用设备
用户可以用访问文件的方法来使用设备
设备名以文件系统中的设备文件的形式存在
所有的设备文件存放在/dev 目录下
几个特殊的设备
/dev/null　　－空设备
/dev/zero　　－零设备
Linux 的目录结构
Linux 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。
Linux 的这种文件系统结构使得一个目录和它包含的文件/子目录之间形成一种层次关系。
大小写敏感
以“.”开头的文件或目录是隐含的




## 文件操作

| | | | |
|---|---|---|---|
| cat | 查看文件内容 | more/less | 查看文件内容 |
| cd | 切换工作目录 | touch | 改变文件的时间属性 |
| chown | 改变文件属权 | mv | 改名或移动文件 |
| chmod | 改变文件权限 | pwd | 显示当前所在的目录 |
| clear | 清除屏幕 | rm | 删除文件或目录 |
| cp | 拷贝文件 | find | 查找文件 |
| ln | 创建文件链接 | which | 寻找命令 |
| ls | 显示目录内容 | tar | 文件打包 |
| mkdir<br>rmdir | 创建/删除空目录 | [g]zip/<br>unzip<br>7za | 文件压缩和解压 |
| Tree | 显示目录树 | | |

1．新建文件

使用 touch 命令生成新的空文件或更改现有文件的时间，关于 touch 命令，其主要作用是来更改已有文件的时间戳的（比如，最近访问时间，最近修改时间），但其在不加任何参数的情况下，只指定一个文件名，则可以创建一个指定文件名的空白文件（不会覆盖已有同名文件）
格式：touch [参数] <文件> ...
参数
-a : 只更改访问时间。
-m : 只更改修改时间。
-t <STAMP> : 使用[[CC]YY]MMDDhhmm[.ss]格式的时间而非当前时间。
-r <参考文件或目录> : 使用指定文件的时间属性而非当前时间。
GNU/Linux 文件的 3 种类型的时间戳：
mtime: 最后修改时间 (ls -lt)
ctime: 状态改变时间 (ls -lc)
atime: 最后访问时间 (ls -lu)
说明
ctime 并非文件创建时间。
覆盖一个文件会改变 mtime、ctime 和 atime 三类时间。
改变文件的访问权限或拥有者会改变文件的 ctime 和 atime。
读文件会改变文件的 atime。

**2. 新建目录**
使用 mkdir（make directories）命令可以创建一个空目录，也可同时指定创建目录的权限属性。
使用 -p 参数，同时创建父目录（如果不存在该父目录）
Rmdir 删除目录（空目录）
**1. 复制文件**
使用 cp 命令（copy）复制一个文件到指定目录。
cp test father/son/grandson
格式：cp [参数] <源> <目标>
说明
若复制的目标文件已存在，则被覆盖。
可以将多个源文件复制到目标目录中。
可以将源目录复制为指定的目标目录（目标目录不存在）。
可以将源目录复制到指定的目标目录中。

| 参数 | 说明 |
| --- | --- |
| -a | 等价于 –dpR |
| -R，-r | 递归地复制目录及目录内的所有项目 |
| -p | 在复制文件过程中保留文件属性，包括属主、组、权限与时间戳 |
| -d | 当复制符号链接的源文件时，目标文件也将创建符号链接且指向源文件所链接的原始文件 |

| | |
|---|---|
| -f | 强制复制，不管目标是否存在 |
| -i | 交互式复制，覆盖文件前需要确认 |
| -u | 只有当源文件的状态改变时间（ctime）比目标文件更新时或目标尚不存在时才进行复制 |

**2. 复制目录**
如果直接使用 cp 命令复制一个目录的话，要成功复制目录需要加上 -r 或者 -R 参数，表示递归复制，就是说有点目录及其下面的子目录层层进入复制的意思。
**1. 删除文件**
使用 rm（remove files or directories）命令删除一个文件
你如果想忽略这提示，直接删除文件，可以使用 -f 参数强制删除：
**2. 删除目录**
跟复制目录一样，要删除一个目录，也需要加上 -r 或 -R 参数：
**1. 移动文件**
使用 mv（move or rename files）命令移动文件（剪切）。命令格式是 mv 源目录文件 目的目录。
**2. 重命名文件**
mv 命令除了能移动文件外，还能给文件重命名。命令格式为 mv 旧的文件名 新的文件名。
**3. 批量重命名**
要实现批量重命名，mv 命令就有点力不从心了，我们可以使用一个看起来更专业的命令 rename 来实现。
Tree
显示目录树
判断文件类型
文件可以包含许多类型的数据
在打开前检查文件的类型来决定要使用的恰当命令或程序
命令
file [选项] <文件名>...
stat [选项] <文件名>...
ln 命令
功能：创建链接文件。
格式：ln [参数] <被链接的文件> <链接文件名>
参数：
-s : 创建符号链接，而非硬链接。
-f : 强行创建链接，不论其是否存在。
-i : 覆盖原有文件之前先询问用户。
举例：
```
$ ln somefile hardlinkfile
$ ln -s somefile softlinkfile
$ ln -s somedir softlinkfile
```

# 标准输入输出

当我们执行一个 shell 命令行时通常会自动打开三个标准文件，即标准输入文件（stdin），默认对应终端的键盘、标准输出文件（stdout）和标准错误输出文件（stderr），后两个文件都对应被重定向到终端的屏幕，以便我们能直接看到输出内容。
进程将从标准输入文件中得到输入数据，将正常输出数据输出到标准输出文件，而将错误信息送到标准错误文件中。
前两个命令都是用来打印文件内容到标准输出（终端），其中 cat 为正序显示，tac 为倒序显示。可以加上 -n 参数显示行号

用户登录后进入的目录通常是自己的主目录
可用 pwd 命令查看用户的当前目录
可用 cd 命令来切换目录
一些特殊字符的特殊含义：
“ .” 表示当前目录
“..” 表示当前目录的上一级目录（父目录）
“-” 表示用 cd 命令切换目录前所在的目录
“~” 表示用户主目录的绝对路径名
绝对路径
以斜线（/）开头
描述到文件位置的完整说明
任何时候你想指定文件名的时候都可以使用
相对路径
不以斜线（/）开头
指定相对于你的当前工作目录而言的位置
可以被用作指定文件名的简捷方式
常用的文本文件提取命令

| 命令 | 功能 |
| --- | --- |
| cat、tac | 滚屏显示文本文件内容 |
| more、less | 分屏显示文本文件内容 |
| head、tail | 显示文本文件的前或后若干行<br>（横向截取文本文件内容） |

| | |
|---|---|
| cut | 纵向切割出文本指定的部分<br>（纵向截取文本文件内容） |
| grep | 在文本文件中查找指定的字符串<br>（按关键字提取文本文件中匹配的行） |

## cat

**真实功能**命令的核心功能其实是 **读取输入并原样输出**（concatenate 的缩写），它既能：

- **输入侧：可以接受文件参数 或 stdin**
- **输出侧：总是输出到 stdout（除非用 > 重定向）**

cat -n passwd
nl 命令，添加行号并打印

## 1.使用 more 和 less 命令分页查看文件

。其中 more 命令比较简单，只能向一个方向滚动，而 less 为基于 more 和 vi （一个强大的编辑器，我们有单独的课程来让你学习）开发，功能更强大。less 的使用基本和 more 一致，具体使用请查看 man 手册，这里只介绍 more 命令的使用。
打开后默认只显示一屏内容，终端底部显示当前阅读的进度。可以使用 Enter 键向下滚动一行，使用 Space 键向下滚动一屏，按下 h 显示帮助，q 退出。
zless file1  分页显示压缩文本文件

## 2. 使用 head 和 tail 命令查看文件

这两个命令，那些性子比较急的人应该会喜欢，因为它们一个是只查看文件的头几行（默认为 10 行，不足 10 行则显示全部）和尾几行。还是拿 passwd 文件举例，比如当我们想要查看最近新增加的用户，那么我们可以查看这个 /etc/passwd 文件，不过我们前面也看到了，这个文件里面一大堆乱糟糟的东西，看起来实在费神啊。因为系统新增加一个用户，会将用户的信息添加到 passwd 文件的最后，那么这时候我们就可以使用 tail 命令了：
$ tail /etc/passwd
甚至更直接的只看一行， 加上 -n 参数，后面紧跟行数：
$ tail -n 1 /etc/passwd
关于 tail 命令，不得不提的还有它一个很牛的参数 -f，这个参数可以实现不停地读取某个文件的内容并显示。这可以让我们动态查看日志，达到实时监视的目的。
下一步

我们可以使用 file 命令查看文件的类型：
说明这是一个可执行文件，运行在 64 位平台，并使用了动态链接文件（共享库）。
与 Windows 不同的是，如果你新建了一个 shiyanlou.txt 文件，Windows 会自动把它识别为文本文件，而 file 命令会识别为一个空文件。这个前面我提到过，在 Linux 中文件的类型不是根据文件后缀来判断的。当你在文件里输入内容后才会显示文件类型。
与搜索相关的命令常用的有 whereis，which，find 和 locate。
whereis 只能搜索二进制文件(-b)，man 帮助文件(-m)和源代码文件(-s)。
使用 locate 命令查找文件也不会遍历硬盘，它通过查询 /var/lib/mlocate/mlocate.db 数据库来检索信息。不过这个数据库也不是实时更新的，系统会使用定时任务每天自动执行 updatedb 命令来更新数据库。所以有时候你刚添加的文件，它可能会找不到，需要手动执行一次 updatedb 命令（在我们的环境中必须先执行一次该命令）**注意，它不只是在 /etc 目录下查找，还会自动递归子目录进行查找。**

## 3. which

which 本身是 Shell 内建的一个命令，我们通常使用 which 来确定是否安装了某个指定的程序，因为它只从 PATH 环境变量指定的路径中去搜索命令并且返回第一个搜索到的结果。也就是说，我们可以看到某个系统命令是否存在以及执行的到底是哪一个地方的命令。

## 4. find 查找命令

find 应该是这几个命令中最强大的了，它不但可以通过文件类型、文件名进行查找而且可以根据文件的属性（如文件的时间戳，文件的权限等）进行搜索
find 命令语法：
find [path][option] [action]
find [<起始目录> ...] [<选项表达式>] [<条件匹配表达式>] [<动作表达式>]
<起始目录>：对每个指定的 <起始目录> 递归搜索目录树
省略<起始目录>时表示当前目录
<选项表达式>：控制 find 命令的行为
<条件匹配表达式>：根据匹配条件查找文件
<动作表达式>：指定对查找结果的操作，默认为显示在标准输出（-print）
不带任何参数的 find 命令将在屏幕上递归显示当前目录下的文件列表。

| 选项 | 说明 |
| --- | --- |
| -L | 如果遇到符号链接文件，就跟踪链接所指的文件 |

| | |
|---|---|
| -regextype TYPE | 指定 -regex 和 -iregex 使用的正则表达式类型，默认为 emacs |
| -depth/-d | 查找目录自身之前先处理目录中的文件（即深度优先） |
| -mount/-xdev | 查找文件时不跨越文件系统 |
| -maxdepth LEVELS | 设置最大的查找深度 |
| --help | 显示 find 命令帮助信息 |
| --version | 显示 find 的版本 |

用户和组

| 条件 | 说明 |
|---|---|
| -uid N | 用户 ID 为 N 的所有文件 |
| -gid N | 组 ID 为 N 的所有文件 |
| -user USERNAME | 用户名为 USERNAME 的所有文件 |
| -group GROUPNAME | 组名为 GROUPNAME 的所有文件 |
| -nouser | 文件属于不在 /etc/passwd 文件中的用户 |
| -nogroup | 文件属于不在 /etc/group 文件中的组 |

N 可以使用 N,+N,-N

文件权限

| 条件 | 说明 |
|---|---|
| -perm MODE | 精确匹配权限模式为 MODE 的文件 |
| -perm -MODE | 匹配权限模式至少为 MODE 的文件<br>（用户、组和其他人这三种权限必须都匹配） |
| -perm /MODE 或<br>-perm +MODE | 匹配权限模式至少为 MODE 的文件<br>（用户、组和其他人这三种权限中有一种匹配即可） |

-name 选项

2. 与时间相关的命令参数：

| 参数 | 说明 |
| --- | --- |
| -atime | 最后访问时间 |
| -ctime | 最后修改文件内容的时间 |
| -mtime | 最后修改文件属性的时间 |

3. -size 选项

如果要根据文件的大小进行搜索，则使用-size 选项

4. -exec 和-ok 选项

这两个选项可以对搜索到的文件执行命令，例如重命名、删除、移动等操作。

注意，对于应用与集合的命令，是不适合使用这两个选项的，应该使用管道通信。

-exec 选项会对每个文件单独执行命令。如果你使用 wc -l 计算每个文件的行数，你会得到每个文件的行数结果，而不是这些文件行数的总和。并且，执行 wc -l 会多次调用 wc 命令，每次处理一个文件，这对于计算所有文件的行数总和并不高效。

要使用 kill 命令来杀死这个进程。

今天就教你怎么召唤一双眼睛出来监督你：

$ xeyes

你可以使用如下命令将它放到后台运行：

$ nohup xeyes &

# 6、ls

ls 命令

ls [选项] [目录或是文件]

ls －l 按完整格式显示目录及文件信息（权限、所有者、文件大小、修改时间、文件名）

r－读 w－写 x－执行

owner/group/others

ls －a

ls －i

ls －al

| 选项 | 说明 |
| --- | --- |
| -a | 列出目录下的所有文件，包括以 . 开头的隐含文件。 |
| -l | 列出文件的详细信息，通常称为“长格式”。 |

| | |
|---|---|
| -d | 输入参数是目录时，只显示该目录本身。 |
| -A | 显示除 “.” 和 “..” 外的所有文件。 |
| -R | 递归地列出所有子目录下的文件。 |
| -h | 以人类易读的单位显示文件大小。 |
| -S | 以文件大小排序输出。 |
| -t | 以时间排序输出。 |

## 文件打包与压缩

Linux tar（英文全拼：tape archive ）命令用于备份文件。
tar 是 Linux 和 Unix 系统中用于归档文件和目录的强大命令行工具。
tar 名字来自 "tape archive"（磁带归档），最初用于将文件打包到磁带设备中，但现在广泛用于在文件系统中打包和压缩文件。
tar 通常用于将多个文件和目录打包成一个归档文件，称为 "tarball"（通常带有 .tar 扩展名）。
tar 本身不压缩文件，但可以与压缩工具（如 gzip 或 bzip2）结合使用，创建压缩的归档文件（如 .tar.gz 或 .tar.bz2）。

| 命令 | 功能 |
|---|---|
| xz | 使用 LZMA 算法的高性能压缩/解压工具 |
| gzip | 流行的 GNU gzip 数据压缩/解压程序 |
| bzip2 | 免费的，无专利的高性能数据压缩工具 |
| zip/unzip | 与 WinZIP 兼容的压缩/解压工具 |
| rar | 与 WinRAR 兼容的压缩/解压工具 |

| | |
|---|---|
| 7za | 使用 LZMA 算法的高性能压缩/解压工具 |
| tar | 文件打包、归档工具 |

| 文件后缀名 | 说明 |
|---|---|
| *.zip | zip 程序打包压缩的文件 |
| *.rar | rar 程序压缩的文件 |
| *.7z | 7zip 程序压缩的文件 |
| *.tar | tar 程序打包，未压缩的文件 |
| *.gz | gzip 程序（GNU zip）压缩的文件 |
| *.xz | xz 程序压缩的文件 |
| *.bz2 | bzip2 程序压缩的文件 |
| *.tar.gz(.tgz) | tar 打包，gzip 程序压缩的文件 |
| *.tar.xz | tar 打包，xz 程序压缩的文件 |
| *tar.bz2(.tbz) | tar 打包，bzip2 程序压缩的文件 |
| *.tar.7z | tar 打包，7z 程序压缩的文件 |

tar [options] -f archive.tar [files...]

- -f archive.tar：指定归档文件的名称。
- [files...]：要打包的文件和目录。

基本功能：打包和解包
格式： tar [选项] 文件或者目录
常用选项
-c：创建新的打包文件。
-t：列出打包文件的内容，查看已经打包了哪些文件。
-x：从打包文件中释放文件。
-f：指定打包文件名。
-v：详细列出 tar 处理的文件信息。
-z：用 gzip 来压缩/解压缩打包文件。
-j：用 bzip2 来压缩/解压缩打包文件。
-J：用 xz 来压缩/解压缩打包文件。

命令中，-r 参数表示递归打包包含子目录的全部内容，-q 参数表示为安静模式，即不向屏幕输出信

息，-o，表示输出文件，需在其后紧跟打包输出文件名。后面使用 du 命令查看打包后文件的大小（后面会具体说明该命令）。

- 设置压缩级别为 9 和 1（9 最大，1 最小），重新打包：

这里添加了一个参数用于设置压缩级别 -[1-9]，1 表示最快压缩但体积大，9 表示体积最小但耗时最久。最后那个 -x 是为了排除我们上一次创建的 zip 文件，否则又会被打包进这一次的压缩文件中，**注意：这里只能使用绝对路径，否则不起作用**。

使用 -e 参数可以创建加密压缩包：

关于 zip 命令，因为 Windows 系统与 Linux/Unix 在文本文件格式上的一些兼容问题，比如换行符（为不可见字符），在 Windows 为 CR+LF（Carriage-Return+Line-Feed：回车加换行），而在 Linux/Unix 上为 LF（换行），所以如果在不加处理的情况下，在 Linux 上编辑的文本，在 Windows 系统上打开可能看起来是没有换行的。如果你想让你在 Linux 创建的 zip 压缩文件在 Windows 上解压后没有任何问题，那么你还需要对命令做一些修改：

$ zip -r -l -o shiyanlou.zip /home/shiyanlou/Desktop

需要加上 -l 参数将 LF 转换为 CR+LF 来达到以上目的。

使用 -O（英文字母，大写 o）参数指定编码类型：

unzip -O GBK

将 shiyanlou.zip 解压到当前目录：

$ unzip shiyanlou.zip

使用安静模式，将文件解压到指定目录：

$ unzip -q shiyanlou.zip -d ziptest

上述指定目录不存在，将会自动创建。如果你不想解压只想查看压缩包的内容你可以使用 -l 参数：

$ unzip -l shiyanlou.zip

在 Linux 上面更常用的是 tar 工具，tar 原本只是一个打包工具，只是同时还是实现了对 7z、gzip、xz、bzip2 等工具的支持，这些压缩工具本身只能实现对文件或目录（单独压缩目录中的文件）的压缩，没有实现对文件的打包压缩，所以我们也无需再单独去学习其他几个工具，tar 的解压和压缩都是同一个命令，只需参数不同，使用比较方便。

下面先掌握 tar 命令一些基本的使用方式，即不进行压缩只是进行打包（创建归档文件）和解包的操作。

行打包（创建归档文件）和解包的操作。

- 创建一个 tar 包：

$ cd /home/shiyanlou

$ tar -P -cf shiyanlou.tar /home/shiyanlou/Desktop

上面命令中，-P 保留绝对路径符，-c 表示创建一个 tar 包文件，-f 用于指定创建的文件名，注意文件名必须紧跟在 -f 参数之后，比如不能写成 tar -fc shiyanlou.tar，可以写成 tar -f shiyanlou.tar -c ~。你还可以加上 -v 参数以可视的的方式输出打包的文件。

- 解包一个文件（-x 参数）到指定路径的**已存在**目录（-C 参数）：

$ mkdir tardir

$ tar -xf shiyanlou.tar -C tardir

- 只查看不解包文件 -t 参数：

$ tar -tf shiyanlou.tar

- 保留文件属性和跟随链接（符号链接或软链接），有时候我们使用 tar 备份文件当你在其他主机还原时希望保留文件的属性（-p 参数）和备份链接指向的源文件而不是链接本身（-h 参数）：

```
$ tar -cphf etc.tar /etc
```

- 我们只需要在创建 tar 文件的基础上添加 -z 参数，使用 gzip 来压缩文件：

```
$ tar -czf shiyanlou.tar.gz /home/shiyanlou/Desktop
```

- 解压 *.tar.gz 文件：

```
$ tar -xzf shiyanlou.tar.gz
```

| 压缩文件格式 | 参数 |
| --- | --- |
| *.tar.gz | -z |
| *.tar.xz | -J |
| *tar.bz2 | -j |

- zip：
- 打包： zip something.zip something （目录请加 -r 参数）
- 解包：unzip something.zip
- 指定路径：-d 参数
- tar：
- 打包：tar -cf something.tar something
- 解包：tar -xf something.tar
- 指定路径：-C 参数

Gzip
压缩后 gzip 会在每个文件的后面添加扩展名 .gz。
压缩后原文件会被自动删除。
在 windows 下可以用 winzip 或 winrar 或 7-zip 解压。
用法：gzip [选项] 文件列表
选项：
-d: 解开压缩文件。
-f: 强行压缩文件，不理会文件名称或硬链接是否存在以及该文件是否为符号链接。
-l: 列出压缩文件的相关信息（压缩文件的大小；未压缩文件的大小；压缩比；未压缩文件的名字）。
-n: 压缩文件时，不保存原来的文件名称及时间戳（默认为保存，即-N）。
-r : 递归处理，将指定目录下的所有文件及子目录一同处理。
-t : 测试压缩文件是否正确无误。
-v : 显示指令执行过程。
-<压缩率> : 压缩率是一个介于 1～9 的数值，默认值为“6”，数值越大压缩率越高。
--best 参数等价于-9；--fast 参数等价于-1。
信息显示命令

| 命令 | 功能 |
| --- | --- |

| | |
|---|---|
| hostname | 显示主机名称 |
| uname | 显示操作系统信息 |
| dmesg | 显示系统启动信息 |
| lsmod | 显示系统加载的内核模块 |
| date | 显示系统时间（cal 可以显示系统时间的日历） |
| env | 显示系统环境变量 |
| locale | 显示当前语言环境（cat /etc/sysconfig/i18n） |
| cat /etc/redhat-release | 显示操作系统版本（head -1 /etc/issue） |
| cat /proc/cpuinfo | 显示 CPU 信息 |
| lspci/lsusb | 显示 PCI/USB 接口信息 |
| rpm -qa | 显示系统已安装的所有软件包 |

| 命令 | 功能 |
|---|---|
| top | 显示当前系统中耗费资源最多的进程 |
| free | 显示当前内存的使用情况（cat /proc/meminfo） |
| du -h | 显示指定的文件（目录）已使用的磁盘空间的总量 |
| df -h | 显示文件系统磁盘空间的使用情况 |
| uptime | 显示系统运行时间、用户数、负载 |
| fdisk -l | 查看所有分区 |
| mount | 查看已经挂装的分区 |
| swapon -s | 查看所有交换分区 |
| ps -ef | 查看所有进程 |
| pstree | 显示进程树 |
| chkconfig --list | 列出所有系统服务 |

| 命令 | 功能 |
| --- | --- |
| who、w | 显示在线登录用户 |
| whoami | 显示用户自己的身份 |
| tty | 显示用户当前使用的终端 |
| id | 显示当前用户的 id 信息 |
| groups | 显示当前用户属于哪些组 |
| last | 查看用户登录日志 |
| crontab -l | 查看当前用户的计划任务 |

| 命令 | 功能 |
| --- | --- |
| ifconfig | 显示网络接口信息 |
| route | 显示系统路由表 |
| iptables -nL | 显示包过滤防火墙的规则设置 |
| netstat | 显示网络状态信息 |
| cat /etc/resolv.conf | 显示 DNS 配置 |
| cat /etc/hosts | 显示静态主机解析表 |

## 数据流重定向

你可能对重定向这个概念感到些许陌生，但你应该在前面的课程中多次见过>或>>操作了，并知道他们分别是将标准输出导向一个文件或追加到一个文件中。这其实就是（输出）重定向，将原本输出到标准输出的数据重定向到一个文件中，因为标准输出(/dev/stdout)本身也是一个文件，我们将命令输出导向另一个文件自然也是没有任何问题的。
上述两个重定向是输出重定向。相应的还有 < 和 << 两种操作，是输入重定向。
在 Linux 系统中默认提供了三个逻辑设备（特殊文件），用于终端的显示和输出，分别为 stdin （标准输入,默认对应于终端作为输入），stdout （标准输出，默认对应于终端作为输出），stderr （标准错误输出，默认对应于终端作为输出）。

| 文件描述符 | 设备文件 | 说明 | 常见指向目标 |
| --- | --- | --- | --- |
| 0 | /dev/stdin | 标准输入 | 终端设备/dev/pts/1 |
| 1 | /dev/stdout | 标准输出 | 终端设备 |
| 2 | /dev/stderr | 标准错误 | 终端设备 |

文件描述符：在形式上是一个非负整数。实际上，它是一个索引值，指向内核为每一个进程所维护的记录表，表中记录该进程打开文件的信息。当进程打开一个现有文件或者创建一个新文件时，内核就会向进程返回一个文件描述符。在程序设计中，一些涉及底层的程序编写往往会围绕着文件描述符展开。但是文件描述符这一概念往往只适用于 UNIX、Linux 这样的操作系统。

| 操作符 | 文件存在时的行为 | 文件不存在时的行为 |
| --- | --- | --- |
| > | 覆盖原内容 | 创建新文件 |
| >> | 追加到原内容末尾 | 创建新文件 |

- 默认使用终端的标准输入 stdin 作为命令的输入和标准输出作为命令的输出：

```
cat
（按 Ctrl+C 退出）
```

- 将 cat 接收的输入又输出（heredoc 的方式）重定向到一个文件：

```
mkdir Documents
cat > Documents/test.c <<EOF
...内容...
EOF
```

1. **cat > Documents/test.c** 表示将 **cat** 命令的输出重定向到 **Documents/test.c** 文件（如果文件不存在则创建）
1. **<<EOF（输入重定向）** 是 "here document" 语法，它表示：
    - 开始一个多行输入
    - 将后续所有内容作为 **cat** 的输入
    - 直到遇到单独一行的 **EOF** 为止（这个标记词可以自定义，常用 EOF 或 END）

- 将一个文件作为命令的输入，标准输出作为命令的输出：

```
cat Documents/test.c
```

上述命令也可以作为查看短文件内容的方法，但是如果文件内容很长就会滚屏到最后一屏。

- 将 echo 命令通过管道传过来的数据作为 cat 命令的输入，将标准输出作为命令的输出：

```
echo 'hi' | cat
```

1. **echo 'hi' 输出字符串 hi\n 到标准输出**
2. **管道 | 将前一个命令的标准输出重定向为下一个命令的标准输入**
3. **cat 检测到自己没有文件名参数，于是开始读取标准输入**
4. **将读取到的内容（hi\n）原样输出到标准输出（终端）**

- 将 echo 命令的输出从默认的标准输出重定向到一个普通文件：

echo 'hello shiyanlou' > redirect
cat redirect
也就是说，cat 和 echo 都可以重定向输出
**管道默认是连接前一个命令的输出到下一个命令的输入**，而重定向通常是需要一个文件来建立两个命令的连接。
重定向标准输出到文件，这是一个很实用的操作，另一个很实用的操作是将标准错误重定向，标准输出和标准错误都被指向伪终端的屏幕显示，所以我们经常看到的一个命令的输出通常是同时包含了标准输出和标准错误的结果的。比如下面的操作：
# 使用 cat 命令同时读取两个文件，其中一个存在，另一个不存在
cat Documents/test.c hello.c
# 你可以看到除了正确输出了前一个文件的内容，还在末尾出现了一条错误信息
# 下面我们将输出重定向到一个文件
cat Documents/test.c hello.c > somefile

遗憾的是，这里依然出现了那条错误信息，这正是因为如我上面说的那样，标准输出和标准错误虽然都指向终端屏幕，实际它们并不一样。那有的时候我们就是要隐藏某些错误或者警告，那又该怎么做呢。这就需要用到我们前面讲的文件描述符了：
# 将标准错误重定向到标准输出，再将标准输出重定向到文件，注意要将重定向到文件写到前面
cat Documents/test.c hello.c >somefile 2>&1
# 或者只用 bash 提供的特殊的重定向符号"&"将标准错误和标准输出同时重定向到文件
cat Documents/test.c hello.c &>somefilehell
**注意你应该在输出重定向文件描述符前加上&,否则 shell 会当做重定向到一个文件名为 1 的文件中**
**方法 1：仅重定向标准错误（2>）**
cat Documents/test.c hello.c 2> error.log
**方法 2：丢弃标准错误（2> /dev/null）**
如果不想保存错误信息，而是直接丢弃：
cat Documents/test.c hello.c 2> /dev/null

- **/dev/null** 是一个特殊设备，写入它的数据会被丢弃。

你可能还有这样的需求，除了需要将输出重定向到文件,也需要将信息打印在终端。那么你可以使用 tee 命令来实现：
echo 'hello shiyanlou' | tee hello
你应该可以看出我们前面的重定向操作都只是临时性的，即只对当前命令有效，那如何做到“永久”有效呢，比如在一个脚本中，你需要某一部分的命令的输出全部进行重定向，难道要让你在每个命令上面加上临时重定向的操作嘛，当然不需要，我们可以使用 exec 命令实现“永久”重定向。exec 命令的作用是使用指定的命令替换当前的 Shell，即使用一个进程替换当前进程，或者指定新的重定向：

```
# 先开启一个子 Shell
zsh
# 使用 exec 替换当前进程的重定向，将标准输出重定向到一个文件
exec 1>somefile
 # 后面你执行的命令的输出都将被重定向到文件中,直到你退出当前子 shell，或取消 exec 的重定向（后面将告诉你怎么做）
ls
exit
cat somefile
```

**什么没有新开终端窗口?**

- **zsh 命令的本质**：
  它只是在当前终端中**新建一个 Zsh 子进程**，而不是像图形化终端（如 GNOME Terminal）那样创建一个新窗口。
- **输入/输出继承**：
  子 Shell 默认绑定到父 Shell 的同一个终端设备（如 **/dev/pts/1**），所以输入/输出仍在原窗口

**方法 1：检查进程层级**

在子 Shell 中运行：

```
pstree -ps $$
```

在 Shell 中有 9 个文件描述符。上面我们使用了也是它默认提供的 0,1,2 号文件描述符。另外我们还可以使用 3-8 的文件描述符，只是它们默认没有打开而已。你可以使用下面命令查看当前 Shell 进程中打开的文件描述符：

```
cd /dev/fd/;ls -Al
```

同样使用 exec 命令可以创建新的文件描述符：

```
zsh
exec 3>somefile
# 先进入目录，再查看，否则你可能不能得到正确的结果，然后再回到上一次的目录
cd /dev/fd/;ls -Al;cd -
# 注意下面的命令>与&之间不应该有空格，如果有空格则会出错
echo "this is test" >&3
cat somefile
exit
```

如上面我们打开的 3 号文件描述符，可以使用如下操作将它关闭：

```
exec 3>&-
cd /dev/fd;ls -Al;cd -
```

在 Linux 中有一个被称为“黑洞”的设备文件,所有导入它的数据都将被“吞噬”。

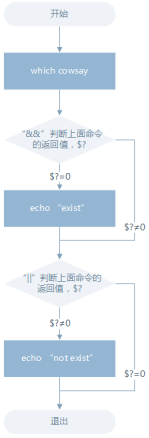
在类 UNIX 系统中，/dev/null，或称空设备，是一个特殊的设备文件，它通常被用于丢弃不需要的输出流，或作为用于输入流的空文件，这些操作通常由重定向完成。读取它则会立即得到一个 EOF。

我们可以利用/dev/null 屏蔽命令的输出：

```
cat Documents/test.c 1>/dev/null 2>&1
```

当我们需要使用 apt-get 安装一个软件，然后安装完成后立即运行安装的软件（或命令工

具），
这时你可能就会想：要是我可以一次性输入完，让它自己去依次执行各命令就好了，这就是这一小节要解决的问题。
简单的顺序执行可以使用;，比如上述操作可以写为：
# 命令顺序执行
sudo apt-get update;sudo apt-get install some-tool;some-tool
但是有时候这样的错误并不是直观可以判断出来的。因此需要能够有选择性的执行命令，比如上一条命令执行成功才继续下一条，或者不成功又该做出其它什么处理,比如我们使用 which 来查找是否安装了某个命令，如果找到就执行该命令，否则什么也不做（虽然这个操作没有什么实际意义，但可帮你更好的理解一些概念）：
which cowsay > /dev/null && cowsay -f head-in ohch~
cowsay 没有是 1 有是 0
上面命令中的 && 就是用来实现选择性执行的，它表示如果前面的命令执行结果（不是表示终端输出的内容，而是表示命令执行状态的结果）返回 0 则执行后面的命令，否则不继续执行。上一次命令的返回结果从 $? 环境变量获取.
而且还有一个 || 表示逻辑或，同样 Shell 也有一个 ||，它们的区别就在于，shell 中的这两个符号除了也可用于表示逻辑与和或之外，就是可以实现这里的命令执行顺序的简单控制。 || 在这里就是与 && 相反的控制效果，当上一条命令执行结果为 ≠0($?≠0)时则执行它后面的命令：




which cowsay>/dev/null && echo "exist" || echo "not exist"
如果存在 ——echo “exists”成立，返回 0——后面那一句不成立
如果不存在——echo “exists”不成立，返回 1——后面那一句成立
如果二者互换
which cowsay>/dev/null || echo "not exist" && echo "exist"
如果存在——echo “not exist”不成立,返回 1——后面也不成立
如果不存在—echo “not exist”成立，返回 0——后面成立
如果二者互换
which cowsay>/dev/null || echo "exist" && echo "not exist"
那完全就反了
如果存在——echo “exists”不成立，返回 1——后面也不成立
管道是什么？管道是一种通信机制，通常用于进程间的通信（也可通过 socket 进行网络通信），它表现出来的形式就是将前面每一个进程的输出(stdout)直接作为下一个进程的输入(stdin)。

管道又分为匿名管道和具名管道（这里将不会讨论在源程序中使用系统调用创建并使用管道的情况，它与命令行的管道在内核中实际都是采用相同的机制）。我们在使用一些过滤程序时经常会用到的就是匿名管道，在命令行中由 | 分隔符表示， | 在前面的内容中我们已经多次使用到了。具名管道简单的说就是有名字的管道，通常只会在源程序中用到具名管道。下面我们就将通过一些常用的可以使用管道的"过滤程序"来帮助你熟练管道的使用。
先试用一下管道，比如查看 /etc 目录下有哪些文件和目录，使用 ls 命令来查看：
ls -al /etc

有太多内容，屏幕不能完全显示，这时候可以使用滚动条或快捷键滚动窗口来查看。不过这时候可以使用管道：
ls -al /etc | less

通过管道将前一个命令(ls)的输出作为下一个命令(less)的输入，然后就可以一行一行地看。
cut 命令
打印每一行的某一字段
打印 /etc/passwd 文件中以 : 为分隔符的第 1 个字段和第 6 个字段分别表示用户名和其家目录：
cut /etc/passwd -d ':' -f 1,6
打印 /etc/passwd 文件中每一行的前 N 个字符：
# 前五个（包含第五个）
cut /etc/passwd -c -5
# 前五个之后的（包含第五个）
   cut /etc/passwd -c 5-
# 第五个
cut /etc/passwd -c 5
# 2 到 5 之间的（包含第五个）
cut /etc/passwd -c 2-5

**输出时，cut 直接拼接 "one" 和 "three"，不添加任何字符，**所以结果是 **"onethree"**
但你的测试显示输出是 **"one:three"**，这说明：你的 **cut** 版本（可能是某些定制版或不同环境）可能行为略有不同
**-d, --delimiter=DELIM**
**作用**：指定字段的分隔符（默认是 **TAB** 制表符）。
**-f, --fields=LIST**
**作用**：选择要提取的字段（列），可以指定单个字段、多个字段或范围。

## grep 命令

grep（global search regular expression）是一个强大的文本搜索工具。grep 使用正则表达式搜索文本，并把匹配的行打印出来。

UNIX 的 grep 家族包括 grep、egrep 和 fgrep：
grep 使用 Basic regular expression (BRE) 书写匹配模式，等效于 grep -G
egrep 使用 Extended regular expression (ERE) 书写匹配模式，等效于 grep -E
fgrep 不使用任何正则表达式书写匹配模式（以固定字符串对待），执行快速搜索，等效于 grep -F

在文本中或 stdin 中查找匹配字符串 grep 命令的一般形式为：
grep [命令选项]... 用于匹配的表达式 [文件]...
grep [options] PATTERN [FILE...]
PATTERN 是查找条件
可以是普通字符串
可以是正则表达式，通常用单引号将 RE 括起来。
FILE 是要查找的文件，可以是用空格间隔的多个文件，也可是使用 Shell 的通配符在多个文件中查找 PATTERN，省略时表示在标准输入中查找。
grep 命令不会对输入文件进行任何修改或影响，可以使用输出重定向将结果存为文件。

| | |
|---|---|
| -c | 只显示匹配行的次数 |
| -i | 搜索时不区分大小写 |
| -n | 输出匹配行的行号 |
| -v | 输出不匹配的行（反向选择） |
| -r | 对目录（子目录）的所有文件递归地进行 |
| -l | 列出匹配 PATTERN 的文件名 |
| --color=auto | 对匹配内容高亮显示 |
| -A NUM | 同时输出匹配行的后 NUM 行 |
| -B NUM | 同时输出匹配行的前 NUM 行 |
| -C NUM | 同时输出匹配行的前、后各 NUM 行 |

grep 在文件中查找字符串
grep  表达式  目标文件
grep “abc” file1
ls –l | grep “^-r.x”

xargs 是一条 UNIX 和类 UNIX 操作系统的常用命令。它的作用是将参数列表转换成小块分段传递给其他命令，以避免参数列表过长的问题。
这个命令在有些时候十分有用，特别是当用来处理产生大量输出结果的命令如 fin

d，locate 和 grep 的结果。
cut -d: -f1 < /etc/passwd | sort | xargs echo
上面这个命令用于将 /etc/passwd 文件按 : 分割取第一个字段排序后，使用 echo 命令生成一个列表。
文本分析命令

| 命令 | 功能 |
| --- | --- |
| wc | 统计文本 |
| sort | 以行为单位对文本文件排序 |
| uniq | 删除文本文件中连续的重复的行 |
| diff | 比较两个文本文件的差异 |
| diff3 | 比较三个文本文件的差异 |
| patch | 为文本文件打补丁 |
| aspell | 为文本文件做拼写检查（西文） |

## wc 命令

用于统计并输出一个文件中行、单词和字节的数目，比如输出 /etc/passwd 文件的统计信息：
wc /etc/passwd
功能：统计文本文件的行数、字数、字符数
格式：wc [选项] [<文件> ...]
举例
$ wc file
$ wc -l file　　# 统计行数
$ wc -w file　　# 统计字数
$ wc -c file　　# 统计字符数
$ wc -L file　　# 统计最长一行的长度

## Sort 排序

功能：以行为单位对文件进行排序
格式：sort [选项] [<文件> ...]
选项

| -r | 逆向排序 |
| --- | --- |
| -f | 忽略字母的大小写 |
| -n | 根据字符串的数值进行排序 |
| -u | 对相同的行只输出一行 |
| -t c | 选项使用 c 做为列的间隔符 |
| -b | 忽略前导的空格 |
| -i | 只考虑可打印字符 |
| -k N | 以第 N 列进行排序（默认以空格或制表符作为列的间隔符） |

默认为字典排序：
cat /etc/passwd | sort

反转排序：
cat /etc/passwd | sort -r
按特定字段排序：
cat /etc/passwd | sort -t':' -k 3

上面的 -t 参数用于指定字段的分隔符，这里是以":"作为分隔符；-k 字段号 用于指定对哪一个字段进行排序。这里 /etc/passwd 文件的第三个字段为数字，默认情况下是以字典序排序的，如果要按照数字排序就要加上 -n 参数：
cat /etc/passwd | sort -t':' -k 3 -n
uniq 命令可以用于过滤或者输出重复行。

- 过滤重复行

## History 命令

我们可以使用 history 命令查看最近执行过的命令（实际为读取${SHELL}_history 文件,如我们环境中的~/.zsh_history 文件），不过你可能只想查看使用了哪个命令而不需要知道具体干了什么，那么你可能就会要想去掉命令后面的参数然后去掉重复的命令：
history | cut -c 8- | cut -d ' ' -f 1 | uniq

然后经过层层过滤，你会发现确是只输出了执行的命令那一列，不过去重效果好像不明显仔细看你会发现它确实去重了，只是不那么明显，之所以不明显是**因为 uniq 命令只能去连续重复的行，不是全文去重**，所以要达到预期效果，我们先排序：
history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq
# 或者
history | cut -c 8- | cut -d ' ' -f 1 | sort -u

这就是 Linux/UNIX 哲学吸引人的地方，大繁至简，一个命令只干一件事却能干到最好。

- 输出重复行

# 输出重复过的行（重复的只输出一个）及重复次数
history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq -dc
# 输出所有重复的行
history | cut -c 8- | cut -d ' ' -f 1 | sort | uniq -D
没有 sort 则会重复

## 比较 compare

cmp 发现第一处不同停止
comm 显示两个文件的相同与不同之处
diff 按行比较不同，显示所有不同的行的信息

## 文本文件处理命令

| 命令 | 功能 |
| --- | --- |

| | |
|---|---|
| tr | 字符替换 |
| sed | 流编辑器，常用于字符串替换 |
| paste | 纵向合并多个文本 |
| expand | 将文件中的制表符转换为空格 |
| unexpand | 将文件中的空格转换为制表符 |
| dos2unix | 将 DOS 格式的文本转换成 UNIX 格式 |
| unix2dos | 将 UNIX 格式的文本转换成 DOS 格式 |
| iconv | 将文本从一种编码转换成另一种编码 |

Sed 命令

sed 是一个流编辑器（stream editor）。sed 是一个非交互式的行编辑器，它在命令行中输入编辑命令、指定被处理的输入文件，然后在屏幕上查看输出。输入文件可以是指定的文件名，也可以来自一个管道的 输出。

与 vi 不同的是 sed 能够过滤来自管道的输入。在 sed 编辑器运行的时候不必人工干涉，所以 sed 常常被称作批编辑器 。

sed 默认不改变输入文件的内容，且总是将处理结果输出到标准输出，可以使用输出重定向将 sed 的输出保存到文件中。

格式

sed [选项] [-e] cmd1 [[-e cmd2] ... [-e cmdn]] [input-file]...

说明

在命令行上执行 sed 编辑命令。可以指定多个编辑命令，每个编辑命令前都要使用 -e 参数，sed 将对这些编辑命令依次进行处理。若只有一个编辑命令时，-e 可以省略。

每个 sed 的编辑命令 cmdX 均应使用单引号括起来。

input-file：sed 处理的文件列表，若省略，sed 将从标准输入中读取输入，也可以从输入重定向或管道获得输入。

选项

-r：使用扩展正则表达式进行模式匹配

-i：直接对输入文件进行 sed 的命令操作

iconv 命令

功能：将文件从一种编码转换成另一种编码

格式：iconv [选项] <输入文件>

选项
-f <encoding> : 指定原始文本编码。
-t <encoding> : 指定要转换的编码。
-o <output file> : 指定输出文件，而不是在标准输出上显示。
-l : 列出所有已知编码字符集。

# 进程管理

首先程序与进程是什么？程序与进程又有什么区别？
**程序**（program）：程序就是执行一系列有逻辑、有顺序结构的指令序列，可以实现某个功能。
**进程**（process）：进程是程序在一个数据集合上的一次执行过程，在早期的UNIX、Linux 2.4 及更早的版本中，它是系统进行资源分配和调度的独立基本单位。
就像做一道菜的菜谱（程序），当厨师按照这个菜谱（程序），把原材料（输入数据）进行加工和处理（洗、切、搅拌、煎、炒、烹、炸等），然后得到了一道菜（输出结果）。这就是程序的一次执行，就是进程。当执行完毕的时候，进程就结束了。可见，程序是静态的，而进程是动态的。
简单来说，程序是为了完成某种任务而设计的软件，比如 vim 是程序。什么是进程呢？进程就是把程序执行一次。
程序只是一系列指令的集合，是一个静止的实体，而进程不同，进程有以下的特性：

- **动态性：**进程的实质是一次程序执行的过程，有创建、撤销等状态的变化。而程序是一个静态的实体。
- **并发性：**进程可以做到在一个时间段内，有多个程序在运行中。程序只是静态的实体，所以不存在并发性。
- **独立性：**进程可以独立分配资源，独立接受调度，独立地运行。
- **异步性：**进程以不可预知的速度向前推进。
- **结构性：**进程拥有代码段、数据段、PCB（进程控制块，进程存在的唯一标志）。也正是因为有结构性，进程才可以做到独立地运行。

**并发：**在一个时间段内，宏观来看有多个程序都在活动，有条不紊的执行（每一瞬间只有一个在执行，只是在一段时间有多个程序都执行过）
**并行：**在每一个瞬间，都有多个程序都在同时执行，这个必须有多个 CPU 才行
引入进程是因为传统意义上的程序已经不足以描述 OS 中各种活动之间的动态性、并发性、独立性还有相互制约性。程序就像一个公司，只是一些证书，文件的堆积（静态实体）。而当公司运作起来就有各个部门的区分，财务部，技术部，销售部等等，就像各个进程，各个部门之间可以独立运做，也可以有交互（独立性、并发性）。
而随着程序的发展越做越大，又会继续细分，从而引入了线程的概念，当代多数操作系统 Linux 2.6 及更新的版本中，进程本身不是基本运行单位，而是线程的容器。就像上述所说的，每个部门又会细分为各个工作小组（线程），而工作小组需要的资源需要向上级（进程）申请。
**线程**（thread）是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发

多个线程，每条线程并行执行不同的任务。因为线程中几乎不包含系统资源，所以执行更快、更有效率。
简而言之,一个程序至少有一个进程,一个进程至少有一个线程。线程的划分尺度小于进程，使得多线程程序的并发性高。另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。就如下图所示：

大概明白进程是个什么样的存在后，我们需要进一步了解的就是进程分类。可以从两个角度来分：

- 以进程的功能与服务的对象来分；
- 以应用程序的服务类型来分；

第一个角度来看，我们可以分为用户进程与系统进程：

- 用户进程：通过执行用户程序、应用程序或称之为内核之外的系统程序而产生的进程，此类进程可以在用户的控制下运行或关闭。
- 系统进程：通过执行系统内核程序而产生的进程，比如可以执行内存资源分配和进程切换等相对底层的工作；而且该进程的运行不受用户的干预，即使是 root 用户也不能干预系统进程的运行。

第二角度来看，我们可以将进程分为交互进程、批处理进程、守护进程

- 交互进程：由一个 Shell 终端启动的进程，在执行过程中，需要与用户进行交互操作，可以运行于前台，也可以运行在后台。
- 批处理进程：该进程是一个进程集合，负责按顺序启动其他的进程。
- 守护进程：守护进程是一直运行的一种进程，在 Linux 系统启动时启动，在系统关闭时终止。它们独立于控制终端并且周期性的执行某种任务或等待处理某些发生的事件。例如 httpd 进程，一直处于运行状态，等待用户的访问。还有经常用的 计划任务进程 cron（crond）和 anacron（anacrond），这个进程是守护进程（daemon），可以按照计划周期性的执行用户设定的某些任务。

关于父进程与子进程便会提及这两个系统调用 fork() 与 exec()
**fork-exec** 是由 Dennis M. Ritchie 创造的
**fork()** 是一个系统调用（system call），它的主要作用就是为当前的进程创建一个新的进程，这个新的进程就是它的子进程，这个子进程除了父进程的返回值和 PID 以外其他的都一模一样，如进程的执行代码段，内存信息，文件描述，寄存器状态等等
**exec()** 也是系统调用，作用是切换子进程中的执行程序也就是替换其从父进程复制过来的代码段与数据段
既然子进程是通过父进程而衍生出来的，那么子进程的退出与资源的回收定然与父进程有很大的相关性。当一个子进程要正常的终止运行时，或者该进程结束时它的主函数 main() 会执行 exit(n); 或者 return n，这里的返回值 n 是一个信号，系统会把这个 SIGCHLD 信号传给其父进程，当然若是异常终止也往往是因为这个信号。
在将要结束的时候，子进程代码部分已经结束执行了，系统的资源也基本归还给系统了，但若是其进程的进程控制块（PCB）仍驻留在内存中，代表这个进程还存在（因为 PCB 就是进程存在的唯一标志，里面有 PID 等信息），并没有消亡，这样的进程称之为僵尸进程（Zombie）。
正常情况下，父进程会收到两个返回值： exit code（SIGCHLD 信号）与 reason for termination（结束原因）。之后，父进程会使用系统调用 wait(&status) 获取子进程的退出状态，然后内核从内存中释放已结束子进程的 PCB；而如若父进程没有这么做的话，子进程的 PCB 就会一直驻留在内存中，一直留在系统中成为僵尸进程（Zombie）。

虽然僵尸进程是已经放弃了几乎所有内存空间，没有任何可执行代码，也不能被调度，仅在进程列表中保留一个位置，记载该进程的退出状态等信息供其父进程收集，从而释放它
但是 Linux 系统中能使用的 PID 是有限的，如果系统中存在有大量的僵尸进程，系统将会因为没有可用的 PID 从而导致不能产生新的进程。
另外如果父进程结束（非正常的结束），未能及时收回子进程，子进程仍在运行，这样的子进程称之为孤儿进程。在 Linux 系统中，孤儿进程一般会被 init 进程所“收养”，成为 init 的子进程。由 init 来做善后处理，所以它并不至于像僵尸进程那样无人问津，不管不顾，僵尸进程大量存在会有危害。
进程 0 是系统引导时创建的一个特殊进程，也称之为内核初始化，其最后一个动作就是调用 fork() 创建出一个子进程运行 /sbin/init 可执行文件,而该进程就是 PID=1 的进程 1，而进程 0 就转为交换进程（也被称为空闲进程），进程 1 （init 进程）是第一个用户态的进程，再由它不断调用 fork() 来创建系统里其他的进程，所以它是所有进程的父进程或者祖先进程。同时它是一个守护程序，直到计算机关机才会停止。
每一个进程都会是一个进程组的成员，而且这个进程组是唯一存在的，他们是依靠 PGID（process group ID）来区别的，而每当一个进程被创建的时候，它便会成为其父进程所在组中的一员。
一般情况，进程组的 PGID 等同于进程组的第一个成员的 PID，并且这样的进程称为该进程组的领导者,也就是领导进程，进程一般通过使用 getpgrp() 系统调用来寻找其所在组的 PGID，领导进程可以先终结，此时进程组依然存在，并持有相同的 PGID，直到进程组中最后一个进程终结。
与进程组类似，每当一个进程被创建的时候，它便会成为其父进程所在 Session（会话）中的一员，每一个进程组都会在一个 Session 中，并且这个 Session 是唯一存在的，
Session 主要是针对一个 tty 建立，Session 中的每个进程都称为一个工作(job)。每个会话可以连接一个终端(control terminal)。当控制终端有输入输出时，都传递给该会话的前台进程组。Session 意义在于将多个 jobs 囊括在一个终端，并取其中的一个 job 作为前台，来直接接收该终端的输入输出以及终端信号。 其他 jobs 在后台运行。
**前台**（foreground）就是在终端中运行，能够与用户进行输入和输出的交互
**后台**（background）就是在终端中运行，但是不能与其进行任何的交互，也不会显示其执行的过程
我们都知道当一个进程在前台运作时我们可以用 ctrl + c 来终止它，但是若是在后台的话就不行了。
我们可以通过 & 这个符号，让我们的命令在后台中运行
ll &
图中所显示的 [1] 236 分别是该 job 的 job number 与该进程的 PID，而最后一行的 Done 表示该命令已经在后台执行完毕。
还可以通过 ctrl + z 使当前工作停止并调到后台中去。
被停止并放置在后台的工作可以使用 jobs 命令查看。（这里的描述有问题，应该是放置在后台的工作，因为在后台运行的工作也可以）
jobs
其中第一列显示的为被放置后台 job 的编号，而第二列的 ＋ 表示最近(刚刚、最后)被放置后台的 job，同时也表示预设的工作，也就是若是有什么针对后台 job 的操作，首先对预设的 job，- 表示倒数第二（也就是在预设之前的一个）被放置后台的工作，倒数第三个（再之前的）以后都不会有这样的符号修饰，第三列表示它们的状态，而最后一列表示该进程执行的命令。

可以通过 fg 命令将后台的工作调到前台来。
#后面不加参数提取预设工作，加参数提取指定工作的编号
#ubuntu 在 zsh 中需要 %，在 bash 中不需要 %
fg [%jobnumber]
之前通过 ctrl + z 使得工作停止放置在后台，若是想让其在后台运作，可以使用如下命令：
#与 fg 类似，加参则指定，不加参则取预设
bg [%jobnumber]
既然有方法将被放置在后台的工作提至前台或者让它从停止变成继续在后台运行，当然也有方法删除一个工作，或者重启等等。
#kill 的使用格式如下
kill -signal %jobnumber

#signal 从 1-64 个信号值可以选择，可以这样查看
kill -l
中常用的有这些信号值：

| 信号值 | 作用 |
| --- | --- |
| -1 | 重新读取参数运行，类似与 restart |
| -2 | 如同 ctrl+c 的操作退出 |
| -9 | 强制终止该任务 |
| -15 | 正常的方式终止该任务 |

**注意**
若是在使用 kill+信号值+pid，将会对 pid 对应的进程进行操作
若是在使用 kill+信号值+%jobnumber，这时所操作的对象是 job，这个数字就是就当前 bash 中后台的运行的 job 的 ID
kill -9 3　# 终止 3 号进程，pid=3
kill -9 %3　# 终止 3 号 job，job id=3
不管在测试的时候、在实际的生产环境中，还是自己的使用过程中，难免会遇到一些进程异常的情况，所以 Linux 为我们提供了一些工具来查看进程的状态信息。我们可以通过 top 实时的查看进程的状态，以及系统的一些信息（如 CPU、内存信息等），我们还可以通过 ps 来静态查看当前的进程信息，同时我们还可以使用 pstree 来查看当前活跃进程的树形结构。
top 工具是我们常用的一个查看工具，能实时的查看我们系统的一些关键信息的变化:
top
top 是一个在前台执行的程序，所以执行后便进入到这样的一个交互界面，正是因为交互界面我们才可以实时的获取到系统与进程的信息。在交互界面中我们可以通过一些指令来操作和筛选。在此之前我们先来了解显示了哪些信息。
load average 在 wikipedia 中的解释是 the system load is a measure of the amount of work that a computer system is doing 也就是对当前 CPU 工作量的度量，具体来说也就是指运行队列的平均长度，也就是等待 CPU 的平均进程数相关的一个计算值。
这是单个 CPU 单核的情况，而实际生活中我们需要将得到的这个值除以我们的核数来看。

我们可以通过以下的命令来查看 CPU 的个数与核心数
#查看物理 CPU 的个数
#cat /proc/cpuinfo |grep "physical id"|sort |uniq|wc -l

#每个 cpu 的核心数
cat /proc/cpuinfo |grep "physical id"|grep "0"|wc -l

通过上面的指数我们可以得知 load 的临界值为 1 ，但是在实际生活中，比较有经验的运维或者系统管理员会将临界值定为 0.7。这里的指数都是除以核心数以后的值，不要混淆了
通常我们都会先看 15 分钟的值来看这个大体的趋势，然后再看 5 分钟的值对比来看是否有下降的趋势。
查看 busybox 的代码可以知道，数据是每 5 秒钟就检查一次活跃的进程数，然后计算出该值，然后 load 从 /proc/loadavg 中读取的。而这个 load 的值是如何计算的呢，这是 load 的计算的源码

CPU 利用率是对一个时间段内 CPU 使用状况的统计，通过这个指标可以看出在某一个时间段内 CPU 被占用的情况，而 Load Average 是 CPU 的 Load，它所包含的信息不是 CPU 的使用率状况，而是在一段时间内 CPU 正在处理以及等待 CPU 处理的进程数情况统计信息，这两个指标并不一样。
**NICE 值**叫做静态优先级，是用户空间的一个优先级值，其取值范围是-20 至 19。这个值越小，表示进程”优先级”越高，而值越大“优先级”越低。nice 值中的 -20 到 19，中 -20 优先级最高， 0 是默认的值，而 19 优先级最低
**PR 值**表示 Priority 值叫动态优先级，是进程在内核中实际的优先级值，进程优先级的取值范围是通过一个宏定义的，这个宏的名称是 MAX_PRIO，它的值为 140。Linux 实际上实现了 140 个优先级范围，取值范围是从 0-139，这个值越小，优先级越高。而这其中的 0 - 99 是实时进程的值，而 100 - 139 是给用户的。
其中 PR 中的 100 to 139 值部分有这么一个对应 PR = 20 + (-20 to +19)，这里的 -20 to +19 便是 nice 值，所以说两个虽然都是优先级，而且有千丝万缕的关系，但是他们的值，他们的作用范围并不相同
** VIRT **任务所使用的虚拟内存的总数，其中包含所有的代码，数据，共享库和被换出 swap 空间的页面等所占据空间的总数
ps 也是我们最常用的查看进程的工具之一，我们通过这样的一个命令来了解一下，他能给我带来哪些信息
ps aux

通过 pstree 可以很直接的看到相同的进程数量，最主要的还是我们可以看到所

有进程之间的相关性。

在 Linux 进程概念实验中讲述了进程如何衍生，进程之间的相关性，这里先回顾一下，当一个进程结束的时候或者要异常结束的时候，会向其父进程返回一个或者接收一个

SIGHUP 信号而做出的结束进程或者其他的操作，这个 SIGHUP 信号不仅可以由系统发送，我们可以使用 kill 来发送这个信号来操作进程的结束或者重启等等。
已经尝试过使用 kill 命令可以管理一些 job，这里尝试用 kill 来直接对进程的 pid 操作，但是前提是要知道进程的 pid。
#首先在桌面找到并打开 gedit、gvim，用 ps 可以查看到
ps aux

#使用 9 这个信号强制结束 gedit 进程
kill -9 1608
我们在使用 ps 命令的时候可以看到大部分的进程都是处于休眠的状态 S，如果这些进程都被唤醒，那么该谁最先享受 CPU 的服务，后面的进程又该是一个什么样的顺序呢？进程调度的队列又该如何去排列呢？

需要靠该进程的优先级值来判定进程调度的优先级，而优先级的值就是上文所提到的 PR 与 nice 来控制与体现了。

nice 的值可以通过 nice 命令来修改的，而需要注意的是 nice 值可以调整的范围是 -20 ~ 19。 root 账户既可以调整自己的进程也可以调整其他用户的程序，并且是所有的值都可以用，而普通用户只可以调制属于自己的进程，并且其使用的范围只能是 0 ~ 19，因为系统为了避免一般用户抢占系统资源而设置的一个限制。

#这个实验在环境中无法做，因为权限不够，可以自己在本地尝试

#打开一个程序放在后台，或者用图形界面打开
nice -n -5 vim &

#用 ps 查看其优先级
ps -afxo user,ppid,pid,stat,pri,ni,time,command | grep vim
copy
我们还可以用 renice 来修改已经存在的进程的优先级=

renice -5 pid

# 存储管理

本地存储管理
存储管理与磁盘分区
技术指标：主轴转速，平均寻道时间，数据传输率，高速缓存，单碟容量
硬盘接口方式
FC-AL,SCSI,SAS,SATA
常用的分区工具
fdisk

sfdisk
GNU parted
　－高级分区操作（创建、复制、调整大小等等）

分区工作 fdisk sfdisk GNU parted
进入交互模式
Fdisk <硬盘设备名>
在命令行方式下显示指定硬盘分区表信息
Fdisk -l <硬盘设备名>

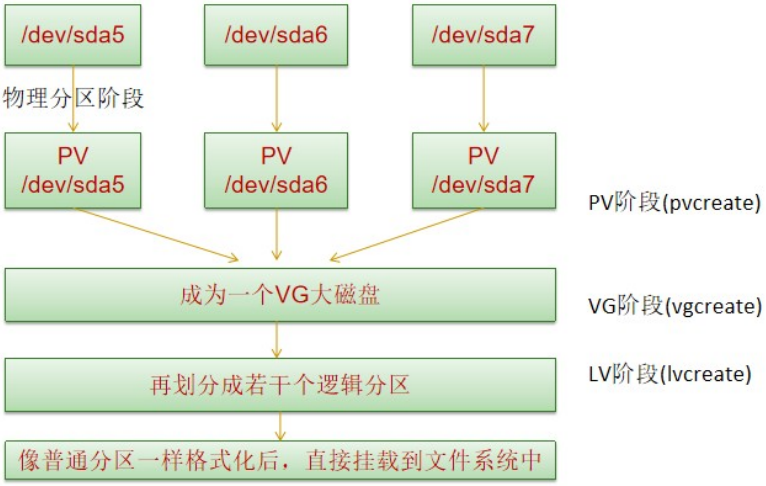
| 子命令 | 说明 | 子命令 | 说明 |
| --- | --- | --- | --- |
| a | 调整硬盘的启动分区 | p | 列出硬盘分区表 |
| d | 删除一个硬盘分区 | q | 退出 fdisk，不保存更改 |
| l | 列出所有支持的分区类型 | t | 更改分区类型 |
| m | 列出所有命令 | u | 切换所显示的分区大小的单 |
| n | 创建一个新的分区 | w | 把设置写入硬盘分区表之后 |

在安装 Linux 的过程中如何正确地评估各分区大小是一个难题，因为系统管理员不但要考虑到当前某个分区需要的容量，还要预见该分区以后可能需要的容量的最大值。

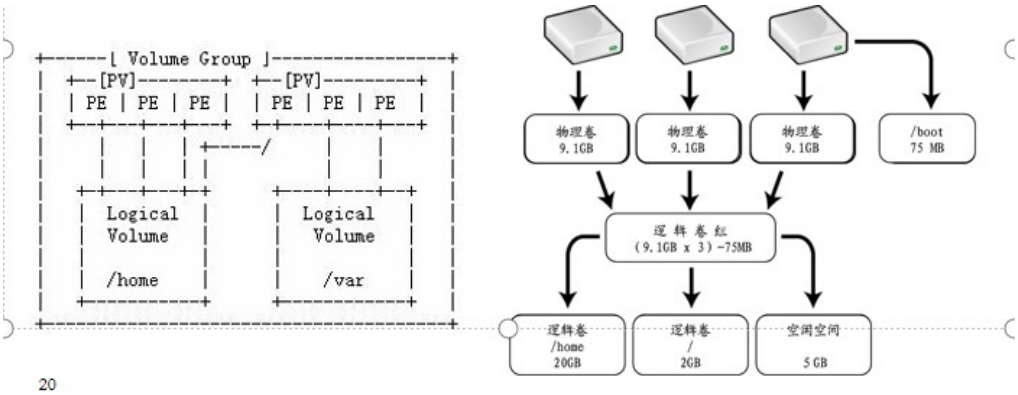
某个分区空间耗尽时，通常的解决方法是：
使用符号链接
　—— 破坏了 Linux 文件系统的标准结构
使用调整分区大小的工具(如:Patition Magic 等)
　—— 必须停机一段时间进行调整
备份整个系统、清除硬盘、重新对硬盘分区，然后恢复数据到新分区
　—— 必须停机一段时间进行恢复操作

磁盘分区工具
#parted [选项] <硬盘设备名>
命令行模式
# parted [选项] <硬盘设备名> <子命令> [<子命令参数>]
子命令
打印帮助信息：help [COMMAND]
显示分区表： print [free|NUMBER|all]
创建新分区：mkpart PART-TYPE [FS-TYPE] START END
删除指定分区：rm NUMBER
设置分区标记：set NUMBER FLAG STATE
LVM 是逻辑盘卷管理（Logical Volume Manager）的简称，它是 Linux 环境下对卷进行方便操作的抽象层。

LVM 是建立在硬盘和分区之上的一个逻辑层，来为文件系统屏蔽下层磁盘分区布局，从而提高磁盘分区管理的灵活性。
LVM 允许在多个物理设备间重新组织文件系统，包括重新设定文件系统的大小。
通过 LVM 可以轻松管理磁盘分区，如：将若干个磁盘分区连接为一个整块的卷组（volume group），形成一个存储池。
可以在卷组中随意创建逻辑卷（logical volumes），并进一步在逻辑卷上创建文件系统。
通过 LVM 可以方便的调整存储卷组的大小，并且可以对磁盘存储按照组的方式进行命名、管理和分配。




物理卷(physical volume, PV)在 LVM 系统中处于最底层
物理卷可以是整个硬盘、硬盘上的分区或从逻辑上与磁盘分区具有同样功能的设备（如：RAID）
物理卷是 LVM 的基本存储逻辑块，但和基本的物理存储介质（如分区、磁盘等）比较，却包含有与 LVM 相关的管理参数
每一个物理卷被划分为基本单元（称为 Physical Extent, PE），具有唯一编号的 PE 是可以被 LVM 寻址的最小存储单元
PE 的大小可根据实际情况在创建物理卷时指定，默认为 4MB
PE 的大小一旦确定将不能改变，同一个卷组中的所有物理卷的 PE 的大小需要一致
卷组(Volume Group, VG)建立在物理卷之上，它由一个或多个物理卷组成
卷组创建之后，可以动态添加物理卷到卷组中，在卷组上可以创建一个或多个“LVM 分区”（逻辑卷）
一个 LVM 系统中可以只有一个卷组，也可以包含多个卷组
LVM 的卷组类似于非 LVM 系统中的物理硬盘
逻辑卷(Logical Volume, LV)建立在卷组之上，它是从卷组中“切出”的一块空间
逻辑卷创建之后，其大小可以伸缩
LVM 的逻辑卷类似于非 LVM 系统中的硬盘分区，在逻辑卷之上可以建立文件系统 （比如 /home 或者 /usr 等）
/boot 分区不能位于卷组中，因为引导装载程序无法从逻辑卷中读取。
如果你想把 / 分区放在逻辑卷上，必须创建一个与卷组分离的 /boot 分区。




20

创建 LVM 类型的分区
在新硬盘上创建物理卷
将新创建的物理卷添加到卷组
在卷组中创建逻辑卷
在逻辑卷中创建文件系统
挂装创建的文件系统

PV 阶段
VG 阶段
LV 阶段
物理卷在 LV 没系统中处于最底层
可以是整个硬盘、硬盘上的分区
每一个物理卷被划分位基本单元 PE 具有唯一编号的 PE 是可以被 LVM 寻址的最小存储单元
PE 的大小可以根据实际情况
卷组 VG 建立在物理卷之上，由一个或多个物理卷组成
卷组创建
可以由一个卷组，也可以多个卷组
逻辑卷建立在卷组之上，它是从卷组中切除的一块空间
如 home 路径、
/boot 分区不能位于卷组中没因为引导装载程序无法从逻辑卷中读取
如果你想把/分区放在逻辑卷上，必须创建一个与卷组分离的/rooot 分区

创建物理卷
创建卷组
创建逻辑卷
创建物理卷
# pvcreate <磁盘或分区设备名>
创建卷组
# vgcreate <卷组名> <物理卷设备名> [...]
创建逻辑卷
# lvcreate <-L 逻辑卷大小> <-n 逻辑卷名> <卷组名>
# lvcreate <-l  PE 值> <-n 逻辑卷名> <卷组名>

查看物理卷。卷组。逻辑卷。’
查看物理卷
# pvdisplay [<物理卷设备名>]
查看卷组
# vgdisplay [<卷组名>]
查看逻辑卷
# lvdisplay [<逻辑卷卷设备名>]

存储管理工具
LBM Logical Volumn Manager
逻辑盘卷管理，建立在硬盘和分区之上
将若干个磁盘分区链接为一个整块的卷组（volumn group），形成一个存储池
若卷组中无剩余空间，首先扩展卷组
添加硬盘，在磁盘上创建 8e 类型的分区
在分区上创建物理卷
将物理卷添加到卷组中
# vgextend <卷组名> <物理卷设备名> [...]
若卷组中有剩余空间，扩展卷组中的逻辑卷
# lvextend <-L +逻辑卷增量> <逻辑卷设备名称>
# lvextend <-l +PE 值>  <逻辑卷设备名称>
对已扩展的逻辑卷中的文件系统进行容量扩展
# resize2fs <分区或逻辑卷设备名>
使用 umount 命令卸载文件系统
使用 e2fsck 命令检查文件系统
使用 resize2fs 命令缩减文件系统容量
缩减逻辑卷
# lvreduce <-L -逻辑卷增量> <逻辑卷设备名称>
# lvreduce <-l -PE 值>  <逻辑卷设备名称>

| 任务 | PV | VG | LV |
| --- | --- | --- | --- |
| 创建 | pvcreate | vgcreate | lvc |
| 删除 | pvremove | vgremove | lvr |
| 显示信息 | pvs | vgs | lvs |
| 扫描列表 | pvscan | vgscan | lvs |
| 显示属性 | pvdisplay | vgdisplay | lvc |
| 更改属性 | pvchange | vgchange | lvc |

| | | | |
|---|---|---|---|
| 扩展 | | vgextend | lve |
| 缩减 | | vgreduce | lvr |

Linux 文件系统
硬盘的分类
硬盘的接口
硬盘分区

逻辑卷管理
Linux 下的文件系统
文件系统是包括在一个磁盘（硬盘、光盘及其它存储设备）上的目录结构；一个磁盘设备可以包含一个或多个文件系统。
文件系统是在一个磁盘（硬盘、光盘及其它存储设备）上组织文件的方法。
文件系统是文件的数据结构或组织方法。
文件系统是基于被划分的存储设备上的一种文件的命名、存储、组织及读取的方法。

一个文件系统是有组织存储文件或数据的方法，目的是易于查询和存取。文件系统是基于一个存储设备，比如硬盘或光盘，并且包含文件文件物理位置的维护。
Linux 下的所有文件和目录以一个树状的结构组织构成了 Linux 中的文件系统
Linux 文件系统标准（Linux File System Standard，FSSTND）
文件系统层次结构标准（File System Hierarchy Standard，FHS）

Linux 的内核采用了称之为虚拟文件系统（Virtual File System，VFS）的技术，因此 Linux 可以支持多种不同的文件系统类型。
Linux 可支持的文件系统
Linux 目前几乎支持所有的 UNIX 类的文件系统，如 HFS、XFS、JFS、Minix FS 及 UFS 等
Linux 支持 NFS 文件系统
Linux 也支持 NTFS 和 vfat（FAT32）
Linux 支持
ext3/ext4
JFS（IBM）
XFS（ SGI ）
Reiserfs
日志文件系统的优点
提高了文件的存储安全性
降低了文件被破坏的机率
缩短了对磁盘的扫描时间
减少了磁盘整体扫描次数
ext2/ext3/ext4
Linux 使用的标准文件系统
swap
交换文件系统

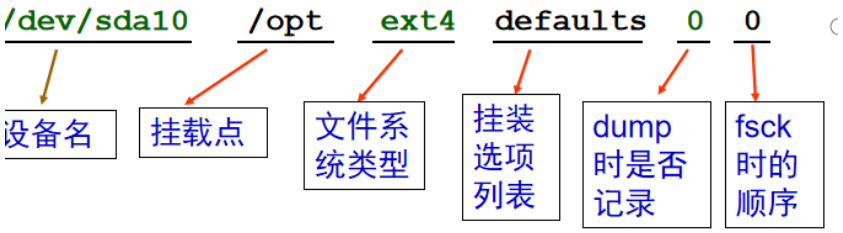
FAT32/vfat
Windows 文件系统
NFS
网络文件系统
iso9660
标准光盘文件系统
在硬盘上创建分区或逻辑卷
可以使用 fdisk 命令创建分区。
可以使用 LVM 的相关命令创建逻辑卷
在分区/LV 上建立文件系统
类似于在 Windows 下进行格式化操作。
挂装文件系统到系统中
手工挂装：使用 mount 命令
启动时自动挂装：编辑“/etc/fstab” 添加相应的配置行。
卸装文件系统
对于可移动介质上的文件系统，当使用完毕可以使用 umount 命令实施卸装操作。

挂载文件系统——mount 命令
功能：挂装文件系统
格式
mount [选项] [<分区设备名>] [<挂装点>]
常用选项
-t <文件系统类型>：指定文件系统类型
-r ：使用只读方式来挂载
-a：挂装/etc/fstab 文件中记录的设备
-o iocharset=cp936：使挂装的设备可以显示中文文件名
-o loop：使用回送设备挂装 ISO 文件和映像文件
挂装点目录必须存在
应该在挂装目录的上级目录下进行挂装操作
不该在同一个挂装点目录下挂装两个文件系统
当文件系统处于“busy”状态时不能进行卸装
文件系统何时处于“busy”状态
文件系统上面有打开的文件
某个进程的工作目录在此文件系统上
文件系统上面的缓存文件正在被使用
fuser 命令可以根据文件（目录、设备）查找使用它的进程，同时也提供了杀死这些进程的方法。
使用举例
查看挂接点有哪些进程需要杀掉
# fuser -cu /mount_point
杀死这些进程（向其发送[SIGKILL, 9]信号）
# fuser -ck /mount_point
查看是否还有进程在访问挂接点
# fuser -c /mount_point

卸载挂接点上的设备
# umount  /mount_point

什么是 LVM,LVM 如何管理
卸载文件系统
Umount 命令
挂载点目录必须存在
应该在怪载目录的上级目录的上级目录下进行挂载操作
不该在同一个挂载点
Fuser 可以根据文件（目录、设备）查找使用它的进程，同时提懂了杀死这些进程的方法
系统启动时，自动挂装文件系统

/etc/fstab
开机后系统会自动搜索该文件中的内容，对雷雨该文件重点文件系统进行自动挂载
fstab (file system table) 是一个纯文本文件，开机后，系统会自动搜索该文件中的内容，对列于该文件中的文件系统进行自动挂载。
系统重启时保留文件系统体系结构
配置文件系统体系结构
被 mount、fsck 和其它程序使用
使用 mount -a 命令挂载 /etc/fstab 中的所有文件系统
可以在设备栏使用文件系统卷标
/etc/fstab 包含的信息
每一行说明一个文件系统的挂载信息
每一行由 6 列信息组成，列与列之间用 TAB 键隔开，一般格式如下：




分区或 LV  挂装点 文件系统类型 挂装选项  备份频率  检查顺序
fs_spec  fs_file  fs_type  fs_options  fs_dump  fs_pass

fs_file：挂装点目录
fs_type：文件系统类型
fs_options：文件系统挂载选项
fs_dump：被”dump”命令使用来检查一个文件系统应该以多快频率进行转储，若不需要转储则该字段为“0”
fs_pass：被”fsck”命令用来决定在启动时需要被扫描的文件系统的顺序，若无需在启动时扫描则该字段为“0”

每一行说明一个文件系统的挂载信息
设备名 挂载点 文件系统类型 挂装选项列表 dump 时记录 fsck 时的顺序
EXT 2/3/4 文件系统
前端命令 mkfs 的格式
Mkfs-t <fstype> -c <分区设备名>
前端命令 mkfs 的格式
# mkfs -t <fstype> -c  <分区设备名>
-t fstype：指定文件系统类型
-c：建立文件系统前先检测有无坏块
举例
# mkfs -t ext3 -c /dev/hda2
# mkfs -t vfat /dev/hdb2
# mke2fs -c /dev/hda2
# mkfs.ext4 /dev/sda1

fsck 是操作系统扫描文件系统内容检查内部一致性的工具。
主要功能
检测并修正链接中断的目录
检测并修正错误时间标记
检测并修正指向错误磁盘区域的 i-node
命令格式
fsck [选项][-t 文件系统类型] <设备名> [特定文件系统的附加选项]

Fsck ——检查文件系统
Tune2fs
显示文件系统属性参数
tune2fs -l  <device>
dumpe2fs -h <device>
可调整的文件系统属性参数
保留块
默认挂载选项
fsck 频率
格式
tune2fs [<选项>] <设备名>
常用选项
-c：表示文件系统在 mount 次数达到设定后，需要运行 fsck 检查文件系统。
-i：文件系统的检查间隔时间。系统在达到时间间隔时，自动检查文件系统。
-j：为 ext2 文件系统添加文件系统日志，转换为 ext3 文件系统。
-m: 设置保留的空间百分比，预设为 5%。
-o: 设置默认加载参数。
-L: 为指定设备设置卷标，不大于 16 字符。

文件系统的 LABLE 和 UUID
标识块设备的传统方法

设备名
标识块设备的其他方法
文件系统 LABEL
文件系统的 UUID
在生成文件系统时自动为设备只当 UUID
磁盘限额是系统管理员用来监控和限制用户或组对磁盘的使用的工具
磁盘限额是系统管理员用来监控和限制用户或组对磁盘的使用的工具。
磁盘限额可以从两方面限制
限制用户或组可以拥有的 inode 数（即文件个数）
限制分配给用户或组的磁盘块的数目
磁盘配额是以每一使用者，每一文件系统为基础的。如果使用者可以在超过一个以上的文件系统上建立文件，那么必须在每一文件系统上分别设定。
硬限制：超过此设定值后不能继续存储新的文件。
软限制：超过此设定值后仍旧可以继续存储新的文件，同时系统发出警告信息, 建议用户清理自己的文件，释放出更多的空间。
时限：超过软限制多长时间之内（默认为 7 天）可以继续存储新的文件。
Edquota
 交互式编辑配额
edquota
命令式设置配额
setquota
将参考用户/组的配额复制给其他用户/组
edquota  -p <protoname> ……
setquota -p <protoname> ……
编辑指定用户的配额
edquota [-u]  [ -f filesystem ] <username>
编辑指定组的配额
edquota -g    [ -f filesystem ] <groupname>
编辑指定用户的配额时限
edquota -t [-u]  [ -f filesystem ]
编辑指定组的配额时限

# 用户管理

查看用户

请打开终端，输入命令：
$ who am i

# 或者

$ who mom likes

说明：在不同的 Linux 发行版中，在线安装方式会有一些差异，包括使用的命令及它们的包管理工具。本实验的环境是基于 Ubuntu 的，所以这里涉及的在线安装方式仅只适用于 Ubuntu 发行版，或其它基于 Ubuntu 的发行版如我国的 ubuntukylin(优麒麟)。Ubuntu 又是基于 Debian 的衍生发行版，使用的也是 Debian 的包管理工具 dpkg，所以一些操作也适用与 Debian。对于其他发行版，由于使用的包管理器不同，相应的命令和选项也会有一些差异，例如 RHEL、CentOS、openEuler 等使用 yum 或者 dnf。请查阅资料学习相应的命令。
通常 Linux 上的软件安装主要有四种方式：

- 在线安装
- 从磁盘安装 deb 软件包
- 从二进制软件包安装
- 从源代码编译安装

这几种安装方式各有优劣，而大多数软件包会采用多种方式发布软件，所以我们常常需要全部掌握这几种软件安装方式，以便适应各种环境。下面将介绍前三种安装方式，从源码编译安装你将在 Linux 程序设计中学习到。

在 Linux 系统里， root 账户拥有整个系统至高无上的权利，比如 新建/添加 用户。
root 权限，系统权限的一种，与 SYSTEM 权限可以理解成一个概念，但高于 Administrator 权限，root 是 Linux 和 UNIX 系统中的超级管理员用户帐户，该帐户拥有整个系统至高无上的权力，所有对象他都可以操作，所以很多黑客在入侵系统的时候，都要把权限提升到 root 权限，这个操作等同于在 Windows 下就是将新建的非法帐户添加到 Administrators 用户组。更比如安卓操作系统中（基于 Linux 内核）获得 root 权限之后就意味着已经获得了手机的最高权限，这时候你可以对手机中的任何文件（包括系统文件）执行所有增、删、改、查的操作。
大部分 Linux 系统在安装时都会建议用户新建一个用户而不是直接使用 root 用户进行登录，当然也有直接使用 root 登录的，例如 Kali（基于 Debian 的 Linux 发行版，集成大量工具软件，主要用于**数字取证**的操作系统）。一般我们登录系统时都是以普通账户的身份登录的，要创建用户需要 root 权限，这里就要用到 sudo 这个命令了。不过使用这个命令有两个前提：

- 要知道当前登录用户的密码
- 当前用户必须在 sudo 用户组。

说明：shiyanlou 用户也属于 sudo 用户组。

账户实质上就是一个用户在系统上的标识
系统依据账户来区分每个用户的文件、进程、任务，给每个用户提供特定的工作环境
Linux 系统下的用户账户有两种
普通用户账户
超级用户账户（管理员账户）
超级用户 yud=0，gid=0
普通用户 uid>=1000
系统用户：0<

用户名和 uid 被保存在/etc/passwd 文件
当用户登录时，它们被分配了一个主目录和一个运行的程序（shell）
组时用户的集合。
每个组都被分配了一个唯一的组 ID 号（GID）
标准组
可以容纳多个用户
私有组
只有用户自己
当创建一个新用户时，若没有指定他所属于的组，RHEL 就会建立一个和该用户同名的私有组，且用户被分配到这个私有组中
一个用户可以属于多个组，这些组可以是私有组，也可以是标准组。
默认启用 shadow oasswird 功能
一般不设置组口令
尽量使用私有组来提高系统安全性
不建议管理员直接编辑
账户验证信息文件
口令文件
文件权限
/etc/passwd
每一个用户一条记录，每条记录用分号杰哥的七个字段组成。
影子口令文件
组账号文件
组口令文件 每一组一 i 奥记录
用户默认配置文件‘新用户基本信息
添加用户账号 useradd

编辑账户验证信息文件
创建主目录

设置用户口令
Passw 的【用户名】
修改用户账号
Usermod 选项与 useradd 命令基本相同
Userdel 删除账号
Groupadd 添加组账号
Groupmod 修改组账号
Groupdel 删除的组账号
Gpasswd 修改组账号
批量用户管理工具
Newusers chpasswd
批量生成安全的口令
Pwgen
口令维护——禁用、删除和维护和口令时效
设置已存在用户的口令时效

用户切换命令
Su
直接切换为冲击用户
Sudo 执行系统管理命令，无需知道超级用户的口令，使用普通用户自己的口令即可。
Id groups whoami w/who
权限管理
Linux 允许多个用户同时在系统上个登录和工作
同归 uid、gid 来却分每个用户
每个进程都是用一个 uid 和一个或多个 gid 来运行
读权限 r 对文件的含义和对目录的含义
写权限 w
执行权限 x
目录上只有执行权限，不能列出目录列表也不能删除改目录
分配三种权限
文件和目录的使用者
属主、同组人、其他人
权限分配
属主的权限：用于限制文件或目录的创建者
属组的权限：用于限制
查看权限
D 文件类型 文件权限 硬链接数或目录包含的文件数 文件所有者 文件所有者所在的用户组 文件长度 文件上次修改时间和日期 文件名
3 套权限控制
文件类型所有者的权限 同组用户权限 其他用户的访问权限
可以用八进制数值表示
Chmod 改变文件或目录权限
Chown 改变文件或目录的属主
Chgrp 改变
Chmod 命令有两种设置方法，一个文件字模是一个八进制。-R 表示对目录中的所有文件或子目录进行递归操作
chown 改变文件的所有者
chmod 改变文件的权限
 chmod u+rw file1
 chmod u+x,g-x file2
 chmod g-x file1
 chmod 610 file1    #r-4/w-2/x-1
chgrp 改变文件的组

U user
G  同组用户
R4w2x1-0
使用三个数字模式来表示，分别代表用户 n1、同组用户 n2、其他用户 n3
每个数字模式由不同权限缩影的数字相加得到一个访问

改变文件/目录属主或组
Root 用户改变文件的所有者
Root 用户或所有者才能改变文件的所属组
默认的访问权限，由 unmask 去决定
默认生成掩码告诉系统当创建一个文件或目录时不应该赋予其哪些权限
系统不允许用户在创建
设置 umask 值的方法
使用 unmask 命令临时设置
在~/.bashrc 实现

三种特殊权限
Suid 使用命令所属用户的权限来运行，而不是命令执行者的权限
Sgid 使用命令的组权限来运行
Suid 和 sgid 都用 s 表示
Sticky-bit 用 t 表示
Suid 时占用属主的 x 位置来表示、sgid 占用组的 x 位置来表示、
使用一个单独 的数字模式（n0）由不同权限所对应的数字相加得到一个表示特殊权限

Ext2/3/4 的文件扩展说下
A atime 告诉系统不要修改
S sync 写操作写到错

A append oly
I immutable
Lsattr
Chattr
并不适用于所有点目录，注意如下目录的扩展属性

Posix 文件访问控制列表
ALC 标准
FACL 时 file access control lists 的缩写

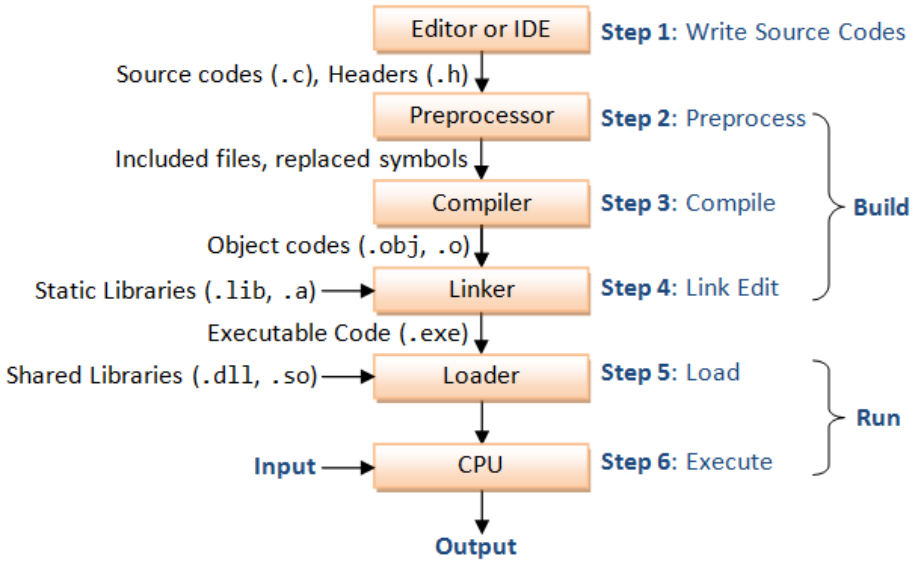
Acl
通过文件系统的挂装选项实现 ACL 支持
查看 ext4 文件的默认选项
存取 ACL
对指定文件或目录的存取控制列表
默认 ACL
ACL 工具
Gerfacl、serfacl
程序
进程
职业/任务
长须知识一个金泰的指令级和

进程时资源申请。调度、和独立运行的单位，因此，它使用系统中的运行资源；而程序不能申请系统资源
Linux 时多用户多任务系统
每个用户均可同时运行多个程序。为了区分每一个，进程号 pid 是唯一地
Linux 采用分时技术来处理
西东启动后第一个进程是 init systemd
Init 是唯一系统内核
它的 pid 是 1
出了 init 之外 每个进程都由父进程
Ruid。Rgideuid、egid
交互进程 由 shell 启动进程 批处理进程  守护进程 daemon
前台进程 后台进程
运行后台进程的方法是在命令行最后加上&
一个中断只能同时存在一个前台任务，但是可以由多个后台任务
查看系统中的进程使用 ps 命令查看进程状态信息
输出项包括：识别号 pid 中断

# 程序开发基础

C Programming Language
Edit/Compile/Link/Run
gcc for C program compiling in Linux
gdb for debugging
gprof for performance tuning
make and makefile

程序的编译过程




编译，链接和运行
编译器
Gcc  gnu compiler collection
Gnu 计划创建一个 unix-like 操作系统，免费软件。推动了在程序员之间的自由和合作
"GNU C Compiler“ □ "GNU Compiler Collection"
support many languages such as C (gcc), C++ (g++), Objective-C, Objective-C++, Java (gcj), Fortran (gfortran), Ada (gnat), Go (gccgo), OpenMP, Cilk Plus, and OpenAcc.

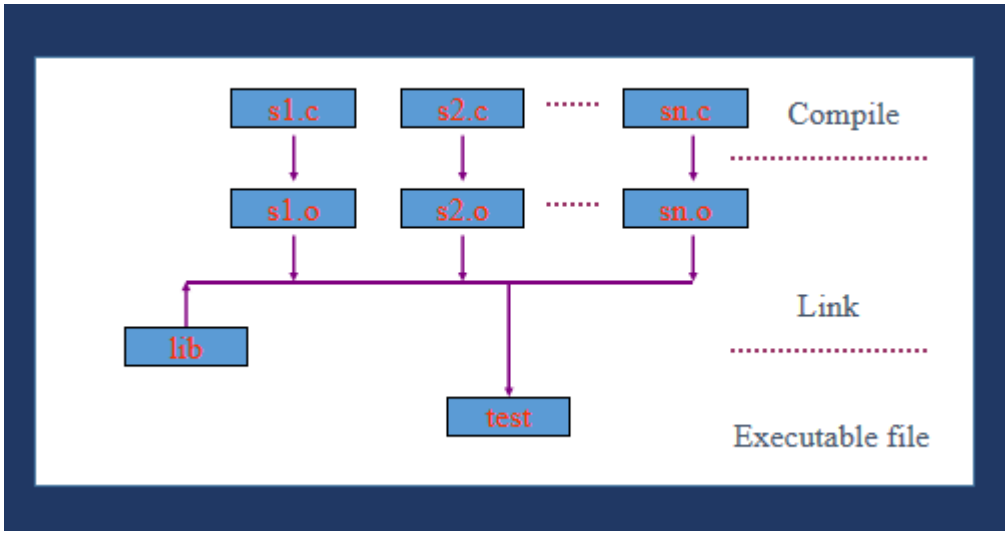
gcc 是一个开发应用和写操作系统的。也叫做 gnu toolchain

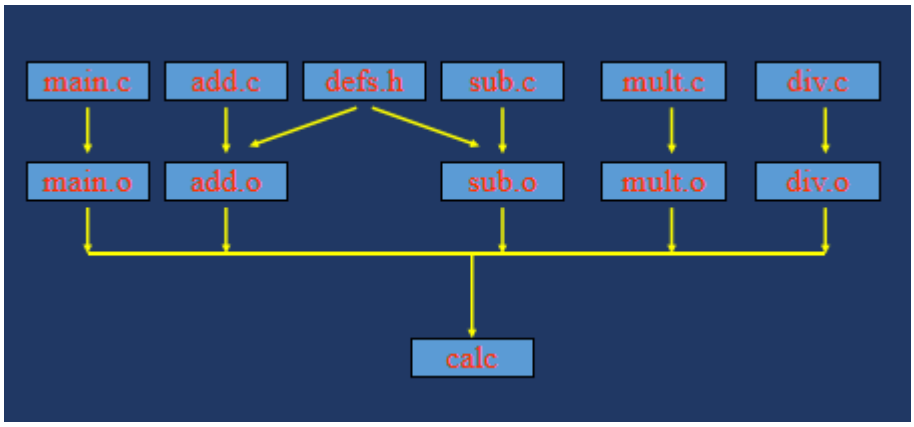
1、 编译器 gcc such as C/C++ and Objective-C/C++.
2、 Gnu make
3、 Gnu binutils
4、 Gnu debugger
5、 Gnu autotools
6、 Gnu bison

Gcc 是可移植的、交叉编译器，可以跨平台
也支持 windows（cygwin（unix emulator under windows）、mingw（minimalist gnu forwindows）。、mingu-w64）
Windows/intel 包括 x86 32 位指令集 i868 ‘x86——64 64 位指令集
32 位编译器能运行在 32 位、64 位向后兼容
但是 64 位编译器、程序只能运行在 64 位上
Cygwin 需要依赖
Mingw 更加简单
gcc
gcc [options] [filenames]
- options 为编译选项
filenames 为需要编译的文件名
gcc test.c  # create “a.out” file
gcc –o test test.c  # create “test” file
gcc test.c –o test

gdb
gcc –g –o test test.c
gdb test
Example:
gcc hello.c //生成可执行文件 a.out
gcc hello.c -o hello //生成可执行文件 hello
gcc -c hello.c -o hello.o //生成目标文件 hello.o
gcc hello1.c hello2.c -o hello //多文件编译
gcc -E hello.c -o hello.i //预处理后生成 hello.i
gcc -S hello.c -o hello.s //生成汇编代码 hello.s
gcc -c -I /usr/dev/mysql/include test.c -o test.o
gcc -L /usr/dev/mysql/lib -lmysqlclient test.o -o test

-o 生成可执行文件
-c 生成目标文件.o
-E 预处理后生成 hello.i
-S 生成汇编代码 hello.s
-I 头文件
-L 库的路径
Make
一个程序包含多个源文件
全部重新编译需要花费时间，你只需要编译一些文件



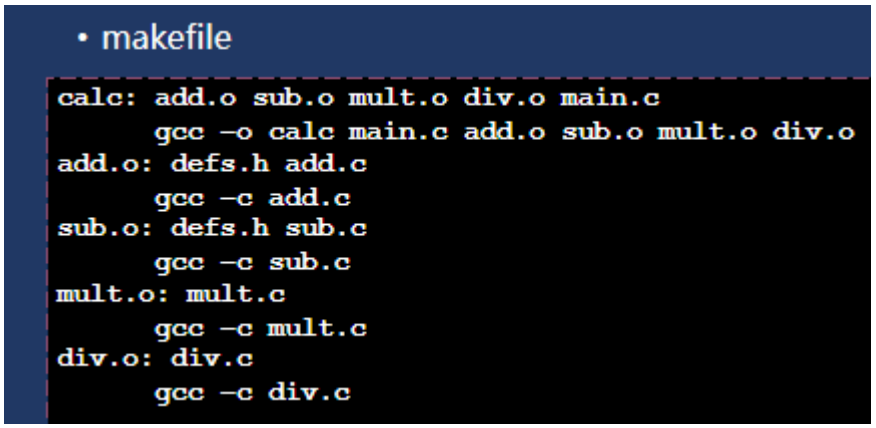



makefile 的结构

objfile: files…　　 文件依赖

(tab)[commands]　　 执行命令

(tab)[commands]


增加 clean

clean:

　　rm calc add.o sub.o mult.o div.o

How to use ‘make’?

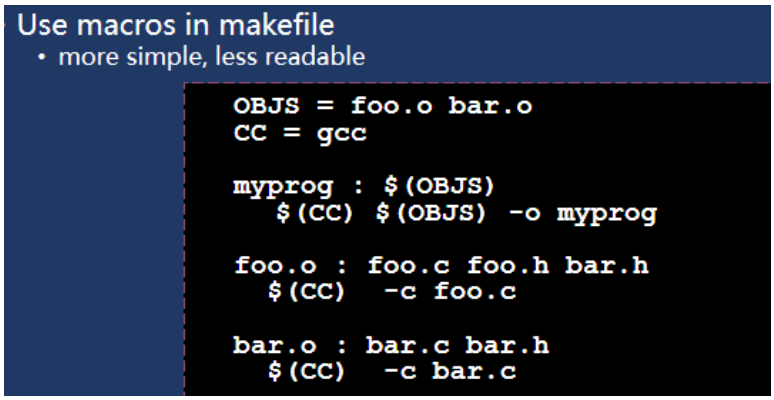
make

make –f myfile　 #"myfile" is makefile

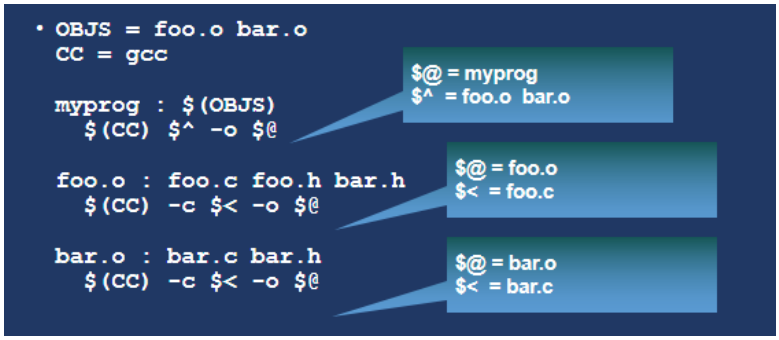
make clean　　#delete all object file
Default rules
Simplify the makefile
C program  object file

```
calc: main.c add.o sub.o mult.o div.o
      gcc -o calc main.c add.o sub.o mult.o div.o
add.o sub.o: defs.h
clean:
      rm add.o sub.o mult.o div.o main.o
```

#ifndef
*def
#dendif
objfile：files
Tab command
Myprog
Clean 并非真正的依赖目标而是伪目标
编写完成后
Make
Make -f myfile
Make clean 所有目标文件清除
默认的规则
.c 文件->.o 文件
为了简化内容，宏变量
可以 macros




Cc c 编译器
Cflags 特殊的选项
$@/$</$^ 当前目标的名字 依赖项的第一个文件 依赖项的所有文件




Example
CC=g++
#LD=/usr/bin/ld
INCLUDES=-I /usr/include/mysql
-I /usr/include

LIBS=-L/lib –L /usr/lib/mysql
-lmysqlclient -lpthread

CPPFLAGS=-DLINUX -D_DEBUGE -O0
-w -g –I ./src

libsrcs=$(wildcard src/*.cpp)

libobjs=$(libsrcs:.cpp=.o)

des_libsrcs=$(wildcard src/*.c)

des_libobjs=$(des_libsrcs:.c=.o)
静态方式定义宏

```
server.exe: $(libobjs) gnu-md5.o
            server.o $(des_libobjs)
 $(CC) -DDEBUGE -g -o $@ $^ -lpthread
                $(INCLUDE) $(LIBS)

%.o: %.cpp
   $(CC) $(CPPFLAGS) -g -c -o $@ $<
%.o: %.c
   gcc -g -c -o $@ $<

server.o: server.c server.h
      $(CC) -c -w -g -DLINUX -o $@ $<                          $(INCLUDES)

gnu-md5.o: gnu-md5.c gnu-md5.h
      $(CC) -c -g -o $@ $<
lean:
  rm -f server.exe server.o gnu-md5.o          $(libobjs) $(des_libobjs)

### 远程拷备到服务器
install:
      scp server.exe schkui@www.hostname.com:/var/bin_path/
```

Gdb 调试器
The gnu project debugger

1、 监视程序中变量的值
2、 在程序中设置断点
3、 程序性的单步执行

Gcc 编译的时候需要添加选项-g -ggdb
调试符号插入到生成的二进制代码
默认不加入调试符号信息
可执行文件大小会增加
调试信息分级-g1/-g2/-g3
-g2 默认扩展到符号表、行号、局部或外部变量的信息
-g3 包含级别 2 中的所有调试信息以及源代码中的宏
-g1 不包含局部变量和与行号有关的调试信息，因此只能用回溯跟踪和堆栈转储
回溯追踪：指的是监视程序在运行过程中函数调用历史。
堆栈转储：以原始的十六进制格式保存程序的执行环境。
任何调试选项都会急剧增大生成的二进制文件的大小，同时增加执行的开销，因此，通常仅用于开发和调试。
使用 gdb filename 启动 gdb，其中 filename 应为可执行文件
Gdb a .out
Gdb 常用命令

R 命令 使用命令 r 运行程序
如果在同一调试过程中需要多次运行程序(run)，后续再执行时便可直接使用 r 指令，系统会默认使用之前的参数。

进行调试过程中重新编译程序后，不必退出 gdb，使用 r 指令重新运行程序，gdb 会自动更新程序状态。

List（I）命令
用来列出源文件中的部分源代码。（需要编译时加入
-g 选项生成对应的编译符号）
l source_file_name.c:col （l 源文件名：行号）
l function_name，以函数为整体进行输出

断点 b
和继续执行 c 命令
Disable /enable 来停用/启用编号为 n 的断点
命令 b 可以在需要地方放置断点，程序在断点位置停止运行
格式：b 断点位置
其中，断点位置可以是行号，也可以是函数名(指定方式与 l 指令类似)，也可以是地址。
使用 c 命令从断点继续执行后续指令。使用命令 disable/enable 断点号 可以启用/停用某断点。使用指令 d 可删除所有的断点，d 1 删除 breakpoint 1.

```
b 10              //在源代码10行处放置断点
b main            //在main函数开始处放置断点
b *0x80480000     //在存放在0x80480000处的指令处放置断点，直接使用地址时需要使用 *地址 的格式
b 10 if a<10      //可以在断点中加入中断执行的条件，表示当a < 10 时才会中断程序执行
```

Watch 。

```
watch a          //当变量 a 的值发生变化时，中断程序执行
watch -l a       // watch指令指定了 -l 参数时，会将指令所接的表达式的计算结果作为地址，
                 // 观察该地址处的值的变化情况
rwatch a         // 当 a 的值被读取时，中断表达式的执行
```

可以为某一表达式设置观察点，当程序执行过程中表达式的值发生改变时，gdb 会中断程序执行，并显示表达式的变化情况
Disp 显示 打印命令 p
P 打印命令
disp(display)命令可以在每次程序暂停时显示指定变量的值
格式：disp 变量名
若输入的变量为数组名，则每次显示数组的所有元素，若为结构体，则输出结构体的所有成员的值。
p(print)命令也将变量的值打印出来，用法与 diap 类似，但结果只显示一次。

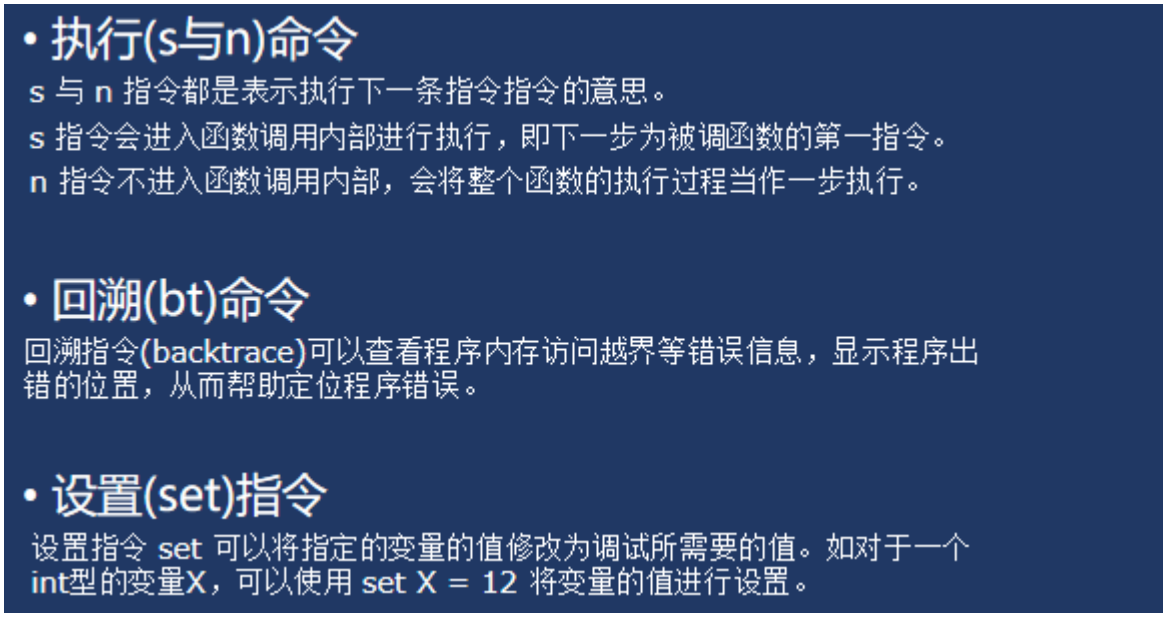
除变量外，p 命令还可以输出给定寄存器、给定地址处的值。
通过一些参数对打印格式进行规定，如 /x 表示以 16 进制格式打印值，/t 表示以二进制格式打印值。

其它显示类 info 命令

```
info reg          //输出所有寄存器的当前值
info frame        //输出栈帧的使用情况
info  b n         //其中 n 为指定的断点号，显示指定断点的状态信息，
                  //不加参数 n 时，会显示所有的断点的信息
```

内存检查 examine 命令
X fmt address
x 命令用于检查内存中某一区域的值，
格式：x fmt address
其中 address 为内存地址的表达式，fmt 由 /重复次数+格式化字符+尺寸字符 组成。
格式化字符有 o(octal，八进制)，x(hex，十六进制)， d(decimal，十进制)，u(unsigned decimal，无符号十进制)，t(binary，二进制)，f(float，浮点)，a(address，地址)，i(instruction，指令)，c(char，字符)，s(string，字符串). 尺寸字符有 b(byte)，h(halfword)， w(word)， g(giant, 8 bytes)

执行（s 与 n）命令
回溯（bt）命令
设置（set）指令




可以使用宏定义对一些常用指令进行定义。
格式：define 宏名，并根据提示输入宏定义，以 end 作为结尾标志。

Cmake
System call 系统调用

C 函数库为每个系统调用提供了一个同名函数。

调用函数的时候，如果有对应的系统调用，自动调用该系统调用。

从程序员使用角度看，系统调用与 C 库函数使用方法相似。
从执行角度看，系统调用运行在核心态，库函数运行在用户态。
printf 与 printk

系统调用引起状态切换，花销并不小。

调用返回时引起系统重新调度
整个系统调用的过程：
执行用户程序(如:fork)
根据 glibc 中的函数实现，取得系统调用号并执行 int $0x80 产生中断。
进行地址空间的转换和堆栈的切换，执行 SAVE_ALL。（进入内核模式）
中断处理，根据系统调用表调用内核函数。
执行内核函数。
执行 RESTORE_ALL 并返回用户模式。
引发系统进程调度。
四个重要的宏
SAVE_ALL
保存用户模式的寄存器和堆栈信息,然后切换到内核模式
RESTORE_ALL
与 SAVE_ALL 相反
SWITH_KERNELSPACE
实现地址空间的转换
SWITH_USERSPACE
arch/i386/kernel/entry.S
系统调用表
arch/i386/kernel/entry.S
配置 TCP/IP
配置 TCP/IP 涉及以下一些文件
/etc/resolv.conf－－DNS 服务器
/etc/host.conf－－域名解析次序
/etc/hosts－－本地域名到 IP 的映射文件
/etc/sysconfig/network－－网关、主机名
/etc/sysconfig/network-scripts/ifcfg.ethx
－－网卡参数的主要文件
/etc/network/interfaces－－Ubuntu 文件
ping
ping [ hostname | IP address ] [options]
ping 210.32.34.137 –c 10

netstat

显示与网络有关的各种数据结构，如显示网络连接、路由表和网络接口信息。
netstat -[ r | i ] [n]
netstat -i -n (显示网络接口)
netstat -r(显示路由表)
ifconfig
显示当前有效网络接口的状态
修改网络接口配置（暂时）
ifconfig [接口]
ifconfig eth0 #显示 eth0 网络接口的参数
ifconfig -a #显示所有网络接口参数

route
对内核的 IP 路由表进行操作。
route [add|del] [-net|-host] target [gw GW]
[netmask Nm] [metric N] [[dev] If]
使用 add 表示增加一条路由条目，del 则删除
/etc/inittab
启动配置文件,每行分为四个域
code:runlevels:action:command
code——用单个或两个字符序列作为本行的标识，在文件中是唯一的。而某些记录必须使用特定的 code 才能使系统工作正常。
runlevels——本行的运行级别。
action——指明 init 程序执行 command 的方式
command——给出相应记录行要执行的命令

Fork（）

exec() #Load and overwrite current image with new one
with fork() together, it can execute a shell command.
Example

execl ( “/bin/ls”, “ls”, “-l”, (char *)0 );
Terminate current process
exit(0)
exit(n)
exit(-1)
int open(const char *pathname, int flags,[mode_t mode])
pathname——字符指针，指向所要打开文件的路径名。可以是相对路径也可以是绝对路径。
flags——是打开文件的方式，在头文件 fcntl.h 中定义几个常量：
O_RDONLY——只读
O_WRONLY——只写

O_TRUNC——截断为 0 长度
O_CREAT——如果文件不存在，创建新文件
O_EXCL——互斥
O_RDWR——读写
O_APPEND——追加
mode——可选参数，只有当 flags 参数为 O_CREAT 时该参数才有效，表示文件的默认权限。一般情况下不使用这个参数。
返回值就是文件描述符，一个整数；打开出错时，返回-1。
例：
fd = open("file", O_RDWR | O_CREAT | O_TRUNC, 0644);

ssize_t read(int filedes, void *buffer, size_t n);
filedes——文件描述符
buffer——指向数组或结构的指针，读入的数据将填充到这里，一般就是数组本身的名字
n——期望读入数据的字节数
返回值是读入数据的字节数，非负的整数；读入出错时，返回-1。
ssize_t write(int filedes, const void *buffer, size_t n);
filedes——文件描述符
buffer——指向数据缓冲区的指针
n——要写入的数据的字节数

off_t lseek(int filedes, off_t offset, int start_flag);
filedes——一个已经打开的文件描述符
offset——表示新位置相对于起始位置的字节数
start_flag——整型，决定起始位置
SEEK_SET offset 是从文件的起始位置算起，通常为 0
SEEK_CUR offset 是相对文件读写的当前位置而言的，通常为 1
SEEK_END offset 是相对文件尾而言，通常为 2

# 基础架构

守护进程（Daemon）
始终在后台运行并响应合法请求的程序
SysVinit
Upstart
Systemd
使用 systemctl 管理服务
-缩写—全称
已运行的服务
为什么要安排调度进程任务
调度任务的守护进程

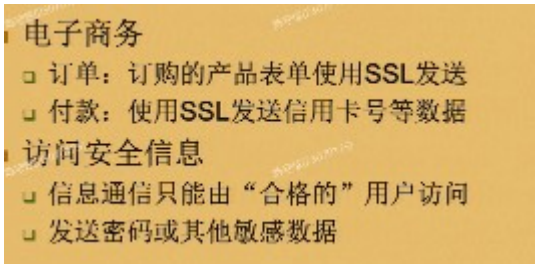
安排调度任务的
Crontab 文件

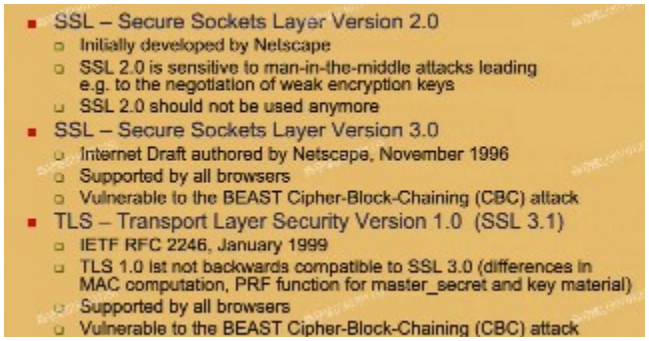
系统日常 的 cron 任务
系统日志服务
日志系统和系统体制
日志的用途
系统审计、检查测追踪和分析统计
日志的功能
用于记录系统、程序运行中发送的各种实践
通过阅读日志，有助于
日志系统
系统日志和内核消息捕捉的日志记录胸痛
主要功能
分类存放日志、方便日志管理
可将日志消息记录到远程主机
Syslog
Rsyslog 采用模块化设计，是 syslog 的替代品
实现了基本的 syslog 协议
输入模块
Imklg、imsock、imfile、imtcp
预处理模块
主队列
过滤模块
执行队列
输出模块
Rsyslog
全局指令
模板
输出通道
规则
Facility.priority action 动作
 日志级别
磁盘存储
统计/statistic
wc [-clw] file_list
Ex: ls -l | wc –l
磁盘存储/Disk store
df disk usage
du disk space
文件系统装卸/mount/unmount
Ex: mount –t iso9660 /dev/cdrom /mnt/cdrom
    umount /dev/cdrom
别名/alias

give alias name to complicated command
alias myls='ls –l | more'
SSH
TCP 协议的风险

1、 窃听——获取通信内容
2、 篡改——修改通信内容
3、 冒充——猫村他人身份参与同学

SSL=secure socket layer（安全套接字层）
1、 提供身份验证的客户端 ‘
2、 在一个公共通信通道发送之前对数据进行加密
3、 确保数据完整性
4、 有效率
5、 在双方协商使用的主要加密算法
对称加密
非对称加密
数字签名
数字签证（x509v.3）
明确和正式的规范
协商参数
在连接时的握手
重用先前谈判的参数



## TLS增强的基于TCP的应用协议

| 服务名 | 端口号 | 实现的安全服务 |
|---|---|---|
| https | 443/tcp | http protocol over TLS |
| smtps | 465/tcp | smtp protocol over TLS |
| smtp | 25/tcp | STARTTLS keyword (RFC 2487) |
| imaps | 993/tcp | imap4 protocol over TLS |
| imap4 | 143/tcp | STARTTLS keyword (RFC 2595) |
| pop3s | 995/tcp | pop3 protocol over TLS |
| pop3 | 110/tcp | STLS keyword (RFC 2595) |
| ldaps | 636/tcp | ldap protocol over TLS |
| nntps | 563/tcp | nntp protocol over TLS |
| FTPS-Data | 989/tcp | FTP Data over SSL/TLS |
| FTPS | 990/tcp | FTP Control over SSL/TLS |

## SSL协议组件

- 握手协议（Handshake protocol）
  - 允许当事人协商需要交易的安全性的不同算法
  - 允许当事人之间的任何身份验证
- 警报协议（Alert protocol）
  - 通知异常情况或报告问题
- 更改密码说明协议（Change Cipher Spec protocol）
  - 强制一个新的握手的执行重新协商安全参数，并重复认证
- 记录协议（Record protocol）
  - 涉及的压缩，加密和MAC

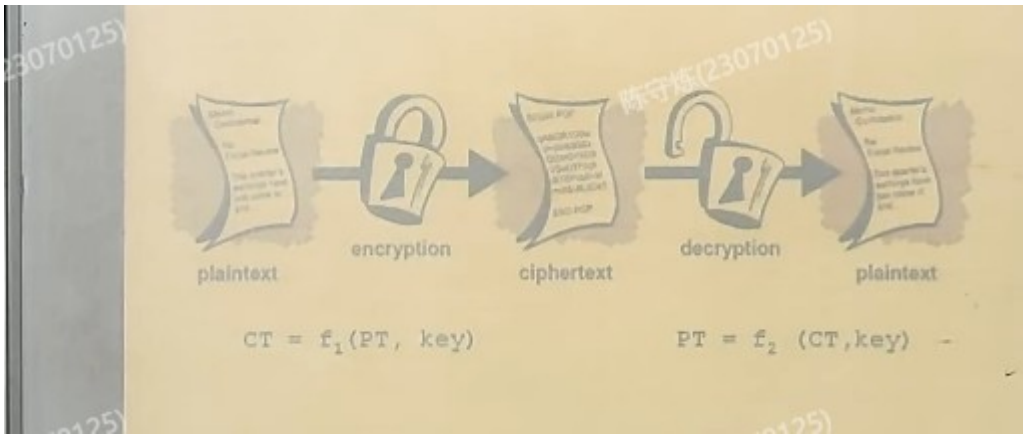
Open SSL

## OpenSSL特性

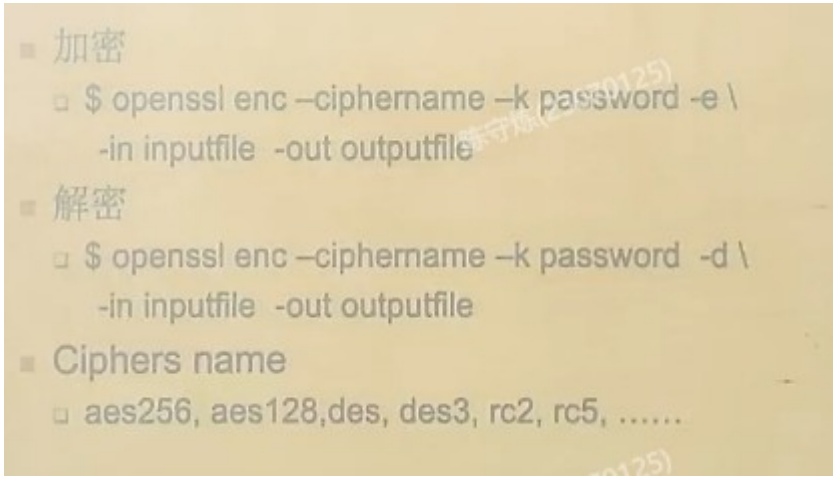
- 开源，基于一个Apache风格的许可证发布
- 提供了SSL v2/v3和TLS v1.0的全功能实现
- 用C语言开发，具有优秀的跨平台性能
- 基于PKI标准，支持 X509 证书标准
- 提供众多的加密和摘要算法库
- 提供了命令行界面（openssl命令）
- 提供了应用程序编程接口


OpenSSL 的命令和算法

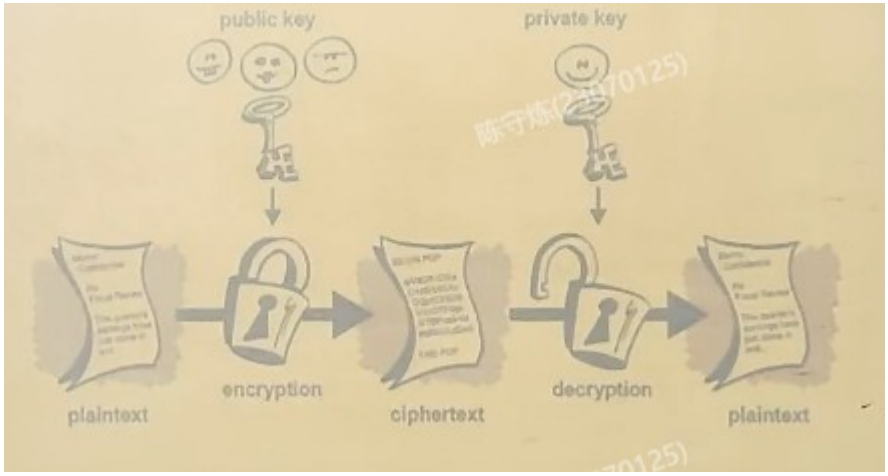





对称加密/解密



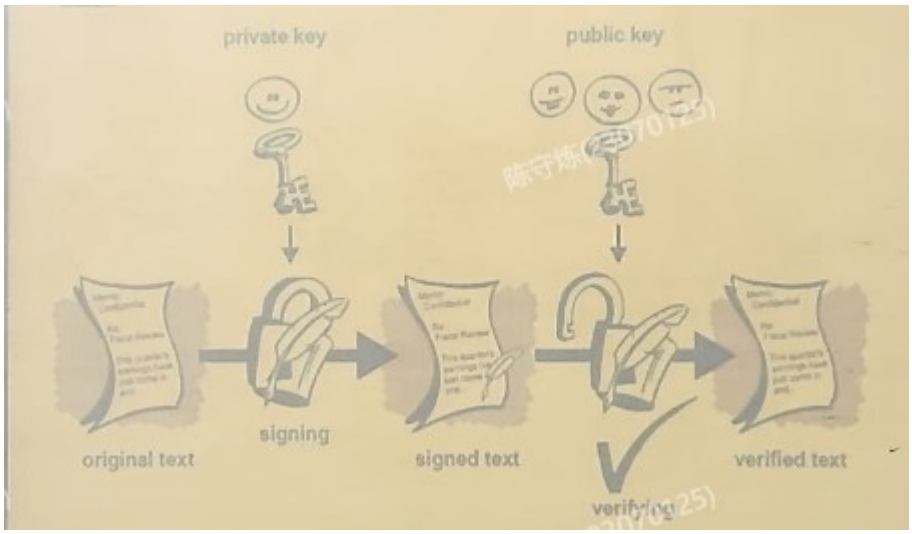

非对称加密和解密



## 加密/解密（非对称）

- 生成密钥对
  - $ openssl genrsa -out priv.keyfile 2048
  - $ openssl rsa -in priv.keyfile -pubout > pub.keyfile
- 用公钥加密
  - $ openssl rsautl -in inputfile -out outputfile \
    -pubin -inkey pub.keyfile -encrypt
- 用私钥解密
  - $ openssl rsautl -in inputfile -out outputfile \
    -inkey priv.keyfile -decrypt

数字签名



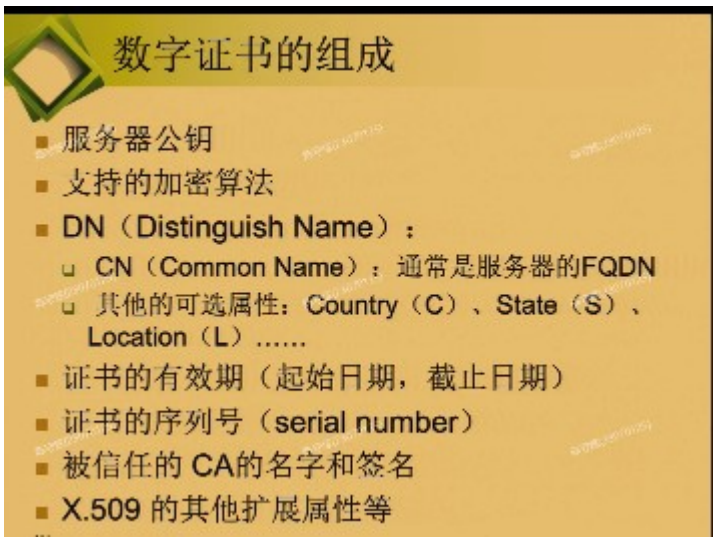

- 生成密钥对
  - $ openssl genrsa -out priv.keyfile 2048
  - $ openssl rsa -in priv.keyfile -pubout > pub.keyfile
- 用私钥签名
  - $ openssl rsautl -in inputfile -out outputfile \
    -inkey priv.keyfile -sign
- 用公钥验证
  - $ openssl rsautl -in inputfile -out outputfile \
    -pubin -inkey pub.keyfile -verify

## Hash/Digest 函数

- 生成 MD5 Hash
  - $ openssl dgst -md5 file
  - $ md5sum file
- 生成 SHA1 Hash
  - $ openssl dgst -sha1 file
  - $ sha1sum file
- 生成 SHA256 Hash
  - $ openssl dgst -sha256 file
  - $ sha256sum file

- PKI
  - 公钥基础设施（Public Key Infrastructure）
  - 是一个基于非对称加密技术实现并提供安全服务的具有通用性的安全基础设施。
  - 通过一组组件和规程，支持利用数字证书管理密钥并建立信任关系。
  - 同时融合了Hash算法以及对称加密技术。
  - 支持PKI的应用标准可以保护信息的完整性、保密性和不可否认性等安全特征。

- 证书（数字证书）
  - 将证书持有者的身份信息和其所拥有的公钥进行绑定的文件。
  - 证书文件还包含颁发证书的权威机构（CA）对该证书的签名。通过签名保障了证书的合法性和有效性。
  - 证书（和相关的私钥）可以提供诸如身份认证、完整性、机密性和不可否认性等安全保护。
  - 当前，通常使用的证书是 X.509 v3 标准的

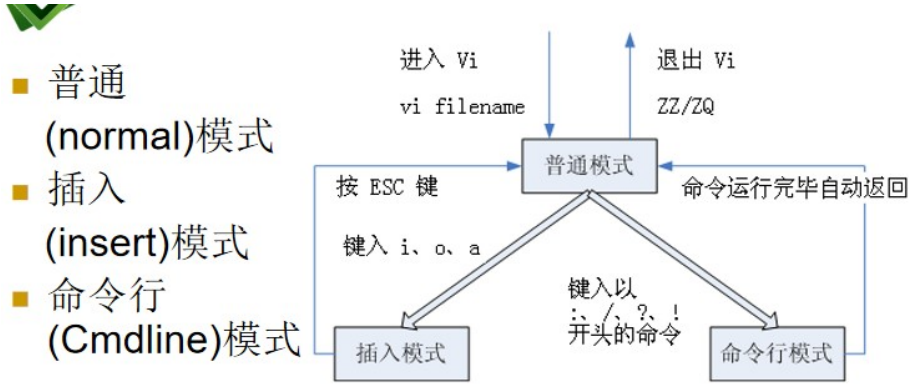



## Vim

vi 是 “Visual interface” 的简称，它可以执行输出、删除、查找、替换、块操作等众多文本操作，而且用户可以根据自己的需要对其进行定制，这是其他编辑程序所没有的。
vi 不是一个排版程序，它不像 M$ Word 或 WPS 那样可以对字体、格式、段落等其他属性进行编排，它只是一个文本编辑程序。
vi 是全屏幕文本编辑器，它没有菜单，只有命令。
vim 即 Vi IMproved，vi 克隆版本之一。

| 命令 | 说明 |
| --- | --- |
| vi | 直接进入 |
| vi filename | 打开或新建文件 filename，并将光标置于第一行首 |
| vi +n filename | 打开文件 filename，并将光标置于第 n 行首 |
| vi + filename | 打开文件 filename，并将光标置于最后一行首 |
| vi +/pattern filename | 打开文件 filename，并将光标置于第一个与 pattern 匹配的 |
| vi -r filename | 打开上次用 vi 编辑时发生系统崩溃，恢复 filename |

Vi 的 3 种运行模式




Normal 模式
在 shell 中输入 vim 启动编辑器时，即进入该模式。
无论什么时候，不管用户处于何种模式，只要按一下 Esc 键，即可使 vim 进入 Normal 模式。
在该模式下，用户可以输入各种合法的 vim 命令，用于管理自己的文档。此时从键盘上输入的任何字符都被当做编辑命令来解释。
若输入的字符是合法的 vim 命令，则 vim 在接受用户命令之后完成相应的动作。但需注意的是，所输入的命令并不在屏幕上显示出来。若输入的字符不是 vim 的合法命令，vim 会响铃报警。

G 用于直接跳转到文件尾
x 删除光标所在的字符
r 替换光标所在的字符
~ 切换光标所在字母的大小写
/和？用于查找字符串
dd、YY、p 分别用于剪切、复制和粘贴一行文本
u 取消上一次编辑操作（undo）
. 重复上一次编辑操作（redo）
ZZ 用于存盘退出 Vi
ZQ 用于不存盘退出 Vi
Insert 模式
在 Normal 模式下输入插入命令 i、附加命令 a 、打开命令 o、修改命令 c、取代命令 r 或替换命令 s 等都可以进入 Insert 模式。
在该模式下，用户输入的任何字符都被 vim 当做文件内容保存起来，并将其显示在屏幕上。在文本输入过程中，若想回到 Normal 模式下，按 Esc 键即可。
Command 模式
Normal 模式下，用户按冒号 “:” 即可进入 Command 模式，此时 vim 会在显示窗口的最后一行 (屏幕的最后一行) 显示一个 “:” 作为 Command 模式的提示符，等待输入命令。
多数文件管理都是在此模式下执行的 (如保存文件等)
Command 模式中所有的命令都必须按 <回车>后执行，命令执行完后，vim 自动回到 Normal 模式。
若在 Command 模式下输入命令过程中改变了主意，可按 Esc 键，或用退格键将输入的命令全部删除之后，再按一下退格键，即可使 vi 回到 Normal 模式下。
:n1,n2 co n3 用于块复制

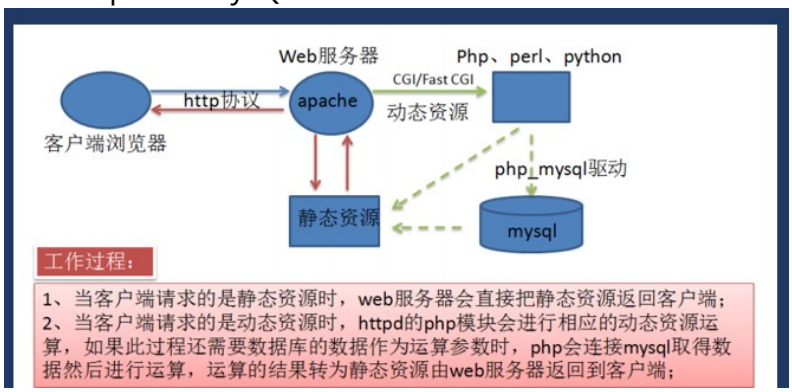
:n1,n2 m n3 用于块移动
:n1,n2 d 用于块删除
:w 保存当前编辑文件，但并不退出
:w newfile 存为另外一个名为 “newfile” 的文件
:wq 用于存盘退出 Vi
:q! 用于不存盘退出 Vi
:q 用于直接退出 Vi （未做修改）
设置 vi 环境
配置文件 ~/.vimrc
:set 显示设置的所有选项
:set all 显示所有可以设置的选项

:set autoindent 缩进,常用于程序的编写
:set noautoindent 取消缩进

显示隐藏行号 :set number/nonumber 简化:set nu/nonu
设置>>和<<缩进数量 :set shiftwidth=4
状态栏标尺 :set ruler/noruler
自动保存 :set autowrite/noautowrite
指示当前行 :set cursorline/nocursorline
语法高亮 :set syntax=on 或 :syntax on/off
Tab 宽度 :set tabstop=4
配色方案
:colorscheme #显示当前配色方案名称
:colorscheme 方案名称 #设置配置方案
例： :colorscheme murphy
配色方案位置：/usr/share/vim/vim74/colors
vim 插件
插件目录 ~/.vim/bundle
Vundle 插件管理器——vim bundle
在 github 上下载到 ~/.vim/bundle 目录中
git clone https://github.com/VundleVim/Vundle.vim.git
修改.vimrc 文件
:PluginInstall

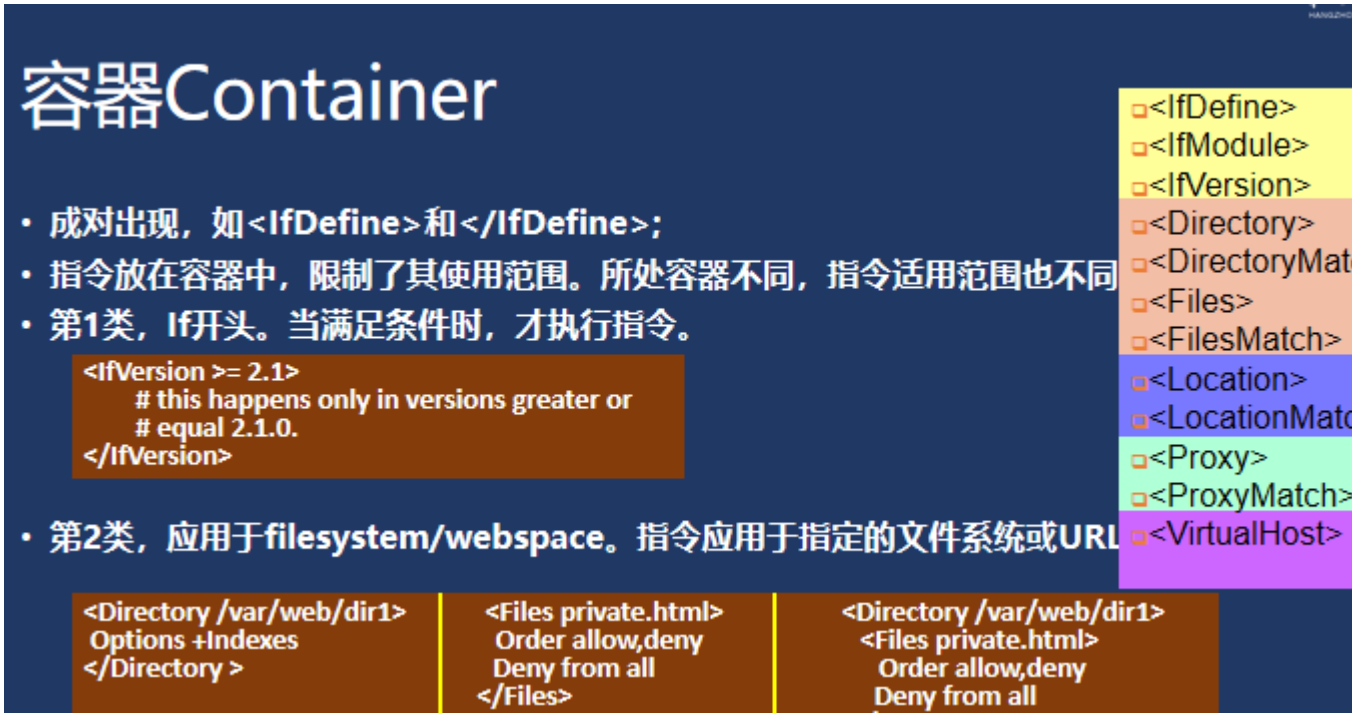
应用管理
浏览 WEB 网页的过程为：
服务器端开启 WEB 服务始终侦听 80 端口；
客户端主机根据本地设置的 DNS 服务器，首先查询网址的 IP 地址，查到网站服务器的 IP 地址后，向其发送浏览网页的申请；
当服务器收到浏览网页的申请时，WEB 服务分配一个进程负责对这个申请进行应答，同时继续侦听 80 端口，准备处理其他的申请；
根据申请的要求，WEB 服务到网页所保存的目录中去查找需要浏览的内容，由 WEB 服务

将内容发送给客户端。
APACHE 是被广泛应用的 Web 服务器。对于 Linux 用户来说，也是最容易使用的 Web 服务器，如果仅仅是想用 APACHE 提供基本的 Web 页面服务，可能根本不需要调整任何配置。
LANMP
Linux+Nginx+Apache+MySQL+PHP
LAMP
Linux+Nginx+MySQL+PHP
LNMP
Linux+Apache+MySQL+PHP

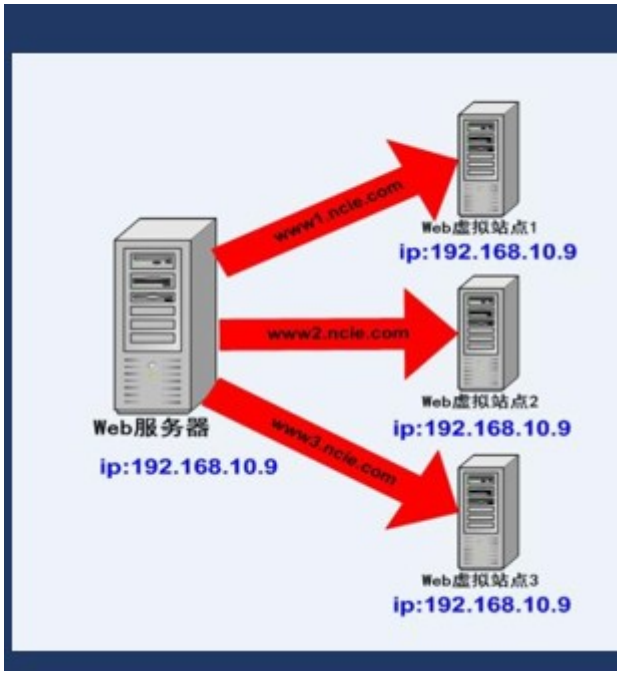



安装好 Apache 服务后，不用配置就可启动。
启动后在客户端浏览器地址栏输入 apache 服务器的 IP 地址，检查是否可以看到 Apache 的测试页，如果能看到说明安装是成功的。如 IP 地址为 127.0.0.1，测试页如图所示。
或者直接在 apache 服务器主机的浏览器地址栏输入“localhost”进行测试。
模块 Module
apache 支持模块支持
在服务器核心中只包含了最基本的功能
扩展的功能可以模块的形式加载到服务器中。
LoadModule
查看已编译进去了哪些基本模块，
#apachectl -

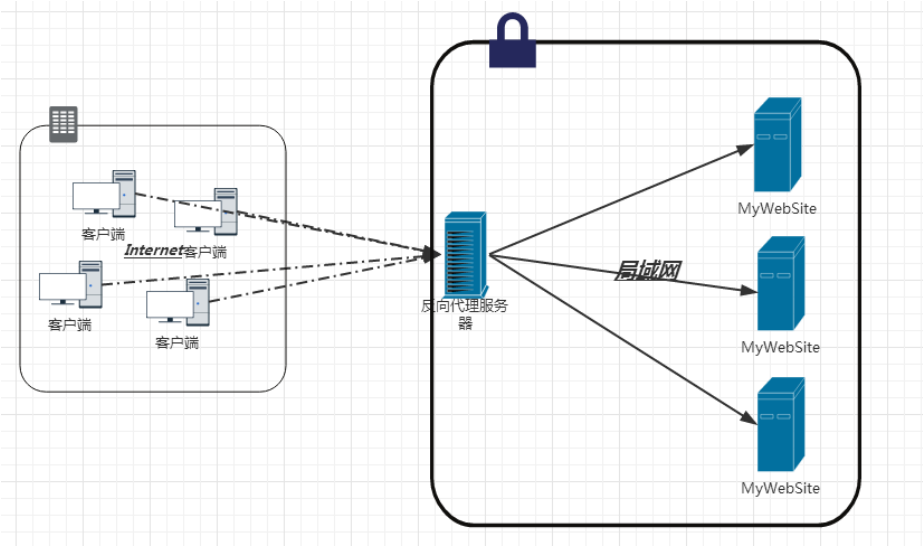
#/usr/sbin/httpd -l




虚拟主机概念：在同一台主机上，使用不同的配置文件，来配置不同的站点



代理 Proxy




为了限制指令的有效范围，可使用<Directory>, <DirectoryMatch>, <Files>, <FilesMatch>, <Location>, <LocationMatch> , <VirtualHost>等。
指令的有效范围分为几种类型：
server config：用于配置文件 httpd.conf ，不用于<VirtualHost> <Directory> 及文件.htaccess。
virtual host：<VirtualHost>
directory：<Directory>, <Location>, <Files>, and <Proxy>
.htaccess ：放置在文件.htaccess 中的指令应用范围为该目录（放置.htaccess 文件的目录）下所有的文件和子目录。.htaccess 文件被改变后会立即生效，因为每次请求都会读取该文件。可进一步控制哪些指令能放置在.htaccess 文件中（即哪些指令允许放置在.htaccess 文件中，并覆盖原来的值），用 AllowOverride 控制。