



# 第1章 Linux安装和编译



## 实验目的

- ❖ 了解Linux发展历史、功能和特点
- ❖ 学习和动手安装Linux操作系统
- ❖ 学习和动手编译Linux内核
- ❖ 掌握用C语言开发应用程序的全过程



# 主要内容

## ❖ 背景知识

- **Linux**简介
- **Linux**系统环境
- **Linux**用户管理命令
- **Linux**文件操作命令
- **Linux**网络管理命令
- **Linux**系统信息命令
- **Linux**编程环境

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核



# 初识Linux

## ❖ Linux是一个类UNIX系统

- 其他类UNIX系统包括，Solaris、Mac OS X等

## ❖ 一个自由的操作系统

- 起源于1991年，Linus Torvalds

## ❖ 一个系统多个桌面

- GNOME，KDE等

## ❖ 主要发行版本

- Red Hat Enterprise
- Fedora
- Ubuntu
- 中国的发行版本：Red Flag



# Linux精髓

## ❖ Linux代表一种开源文化

- 免费软件，开放源代码
- 自由软件：你可以在原有程序基础上开发自己的程序
- 促进了软件的发展

## ❖ 核心结构

- LINUX内核
- LINUX SHELL
- LINUX文件系统
- LINUX应用系统





# Linux特点

- ❖ 多用户
- ❖ 多任务
- ❖ 图形用户接口
- ❖ 硬件支持
- ❖ 网络连接
- ❖ 网络服务器
- ❖ 应用支持



# Linux与Windows的区别

## ❖ 文件系统

- Linux需要一个挂载在/的ext3分区和一个作为虚拟内存的swap分区
- Linux下没有盘符，可以通过设备名挂载，挂在信息在/dev/fstab
- `mount -t ntfs /dev/sda1 /mnt/win_c`

## ❖ Linux把所有的设备都映射到/dev目录下的一个文件

## ❖ 系统内核

- Windows是一个微内核系统，只提供基础功能，其他功能通过服务实现。
- Linux把所有操作系统功能作为一个模块做在其内核中。

## ❖ 用户管理

- 系统管理员是root，使用su命令切换



## Linux的常用软件

- ❖ 办公软件OpenOffice.org
- ❖ 浏览器Firefox
- ❖ 即时通讯Gaim（QQ使用LumaQQ或Eva）
- ❖ 播放音乐XMMS（若要支持MP3需加插件）
- ❖ 文本编辑gedit
- ❖ FTP下载gftp





# Linux下开发工具

## ❖ 命令行界面

## ❖ vi编辑器

- 命令模式和插入模式，通过a、i、Esc切换
- 插入模式
  - ✓ 可以输入文件内容
- 命令模式
  - ✓ 可以使用命令来操纵文件



## vi编辑器命令状态下的光标移动命令

- ❖ 方向键：移动光标
- ❖ w：移动光标到下一个单词
- ❖ b：移动光标到前一个单词
- ❖ 0或^：移动光标到行首
- ❖ \$：移动光标到行尾
- ❖ H：移动光标到屏幕左上角
- ❖ M：移动光标到屏幕中间行第一个字符
- ❖ L：移动光标到屏幕左下角
- ❖ G：移到最后一行
- ❖ gg或1G：移到第一行



## vi编辑器命令状态下的文本编辑命令

- ❖ **x**: 删除光标所在位置字符
- ❖ **X**: 删除光标前一位置字符
- ❖ **dw**: 删除到单词尾
- ❖ **d\$**: 删除到行尾
- ❖ **d0**: 从行首删除之当前位置
- ❖ **dd**: 删除当前行



## vi编辑器命令状态下的文件保存命令

- ❖ **Zz或:wq:** 保存当前文件，退出vi
- ❖ **:w:** 保存当前文件，继续编辑
- ❖ **:q:** 当文件未做改动时，退出
- ❖ **:q!:** 退出而不保存



# Linux的编程工具

## ❖ 编辑工具

- vi, emacs

## ❖ 编译、链接

- gcc, g++

- make命令

## ❖ 调试

- gdb

## ❖ 版本控制工具

- CVS等





# Linux的shell

- ❖ Shell和Windows下的cmd类似
- ❖ Shell提供了一个运行程序、管理文件系统、编译计算机代码、管理计算机的途径
- ❖ Shell比GUI功能强大得多
- ❖ 常用的Shell
  - Korn
  - Bourne
  - C
  - Bash（缺省值）
- ❖ 普通用户提示符\$, root提示符#

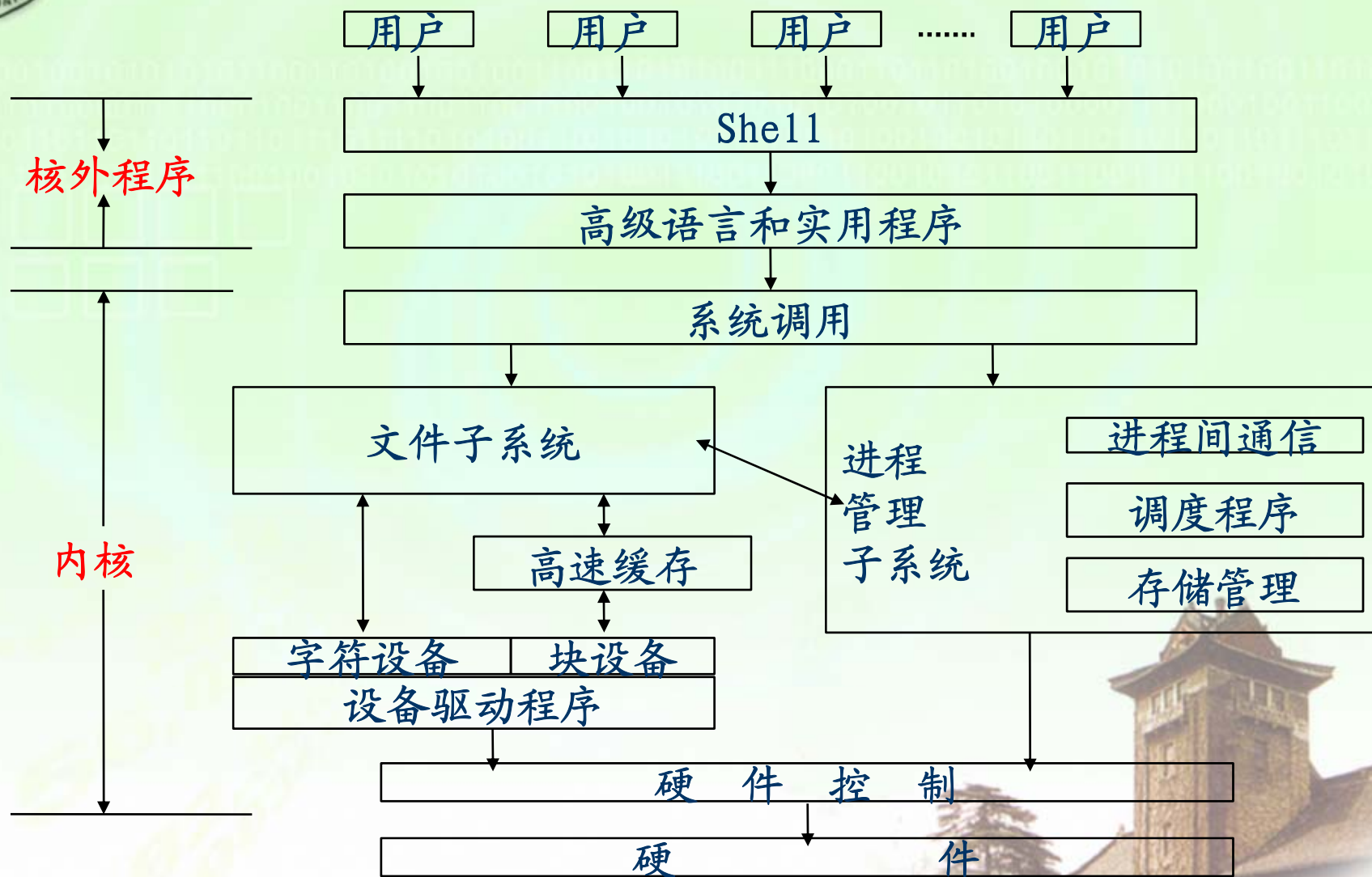


# Linux用户管理

- ❖ **Linux**是一个多用户的操作系统，注册用户要使用**Linux**系统资源，首先必须登录系统。使用完系统后，必须退出系统。
- ❖ **Linux**将用户分为**普通用户**和**超级用户**
  - 超级用户(**root**): 系统管理员一般使用超级用户帐号完成一些系统管理的工作
  - 普通用户: 一般的**Linux**使用者均为普通用户

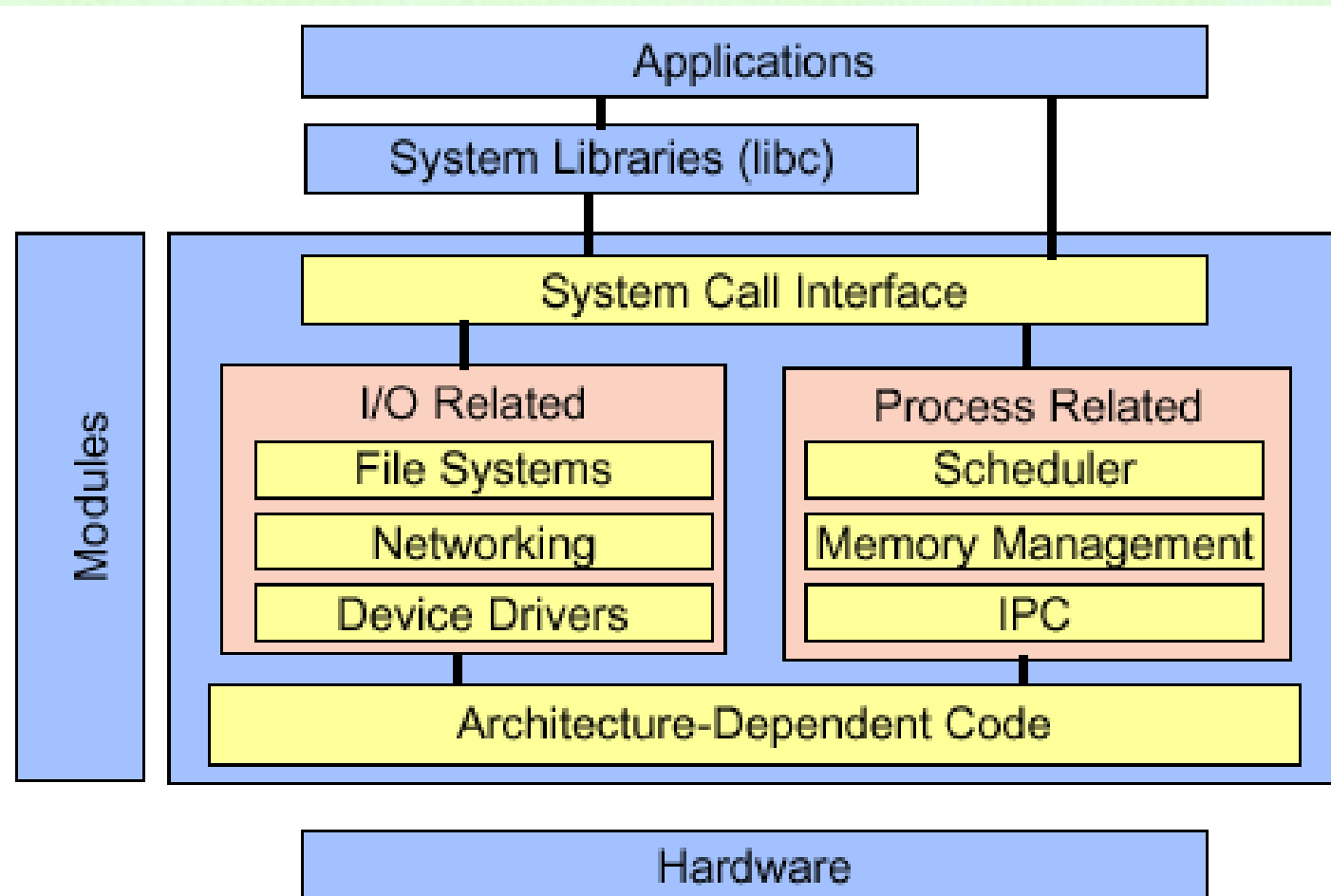


# Linux的系统结构





# 程序设计角度的Linux结构





# 主要内容

## ❖ 背景知识

- Linux简介
- **Linux系统环境**
- Linux用户管理命令
- Linux文件操作命令
- Linux网络管理命令
- Linux系统信息命令
- Linux编程环境

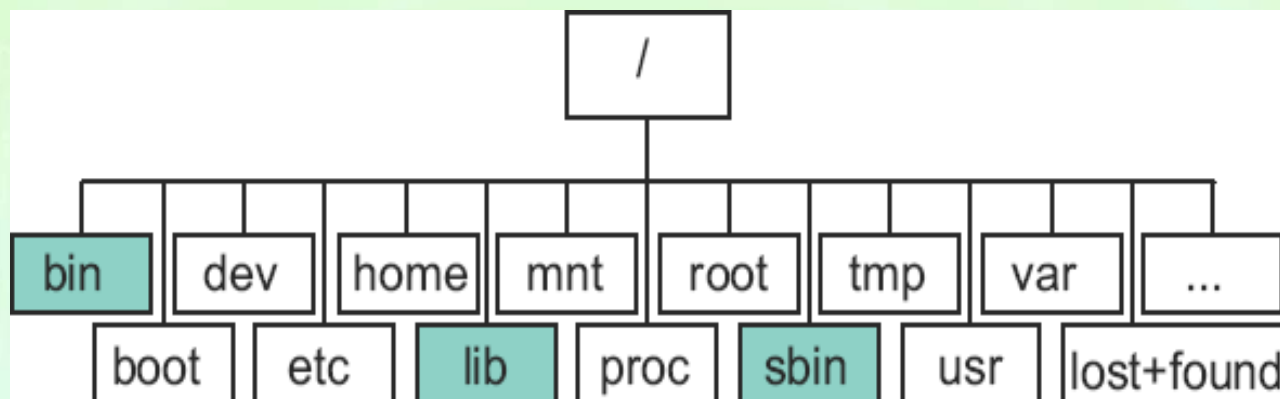
## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核





# 文件系统的层次结构



- ❖ /: 文件系统结构的起始点，称为根目录
- ❖ /home: 用户主目录
- ❖ /bin: 所有的标准指令和工具程序
- ❖ /usr: 系统所使用的文件和指令
- ❖ /usr/bin: 面向用户的命令和工具程序
- ❖ /usr/sbin: 系统管理员的命令
- ❖ /usr/lib: 编程语言库
- ❖ /usr/doc: Linux文档
- ❖ /usr/man: 在线的联机帮助手册
- ❖ /usr/spool: 假脱机文件
- ❖ /sbin: 系统管理员开启系统的命令
- ❖ /var: 时变的文件，例如邮箱文件
- ❖ /dev: 设备的文件接口
- ❖ /etc: 系统配置文件和所有其它系统文件



# 文件系统的概念

## ❖ 文件系统

- 磁盘上有特定格式的一片区域

## ❖ 文件

- 文件系统中存储数据的一个命名的对象

## ❖ 目录

- 其中包含许多文件项目的一类特殊文件

## ❖ 子目录

- 被包含在另一个目录中的目录，包含子目录的目录称为父目录

## ❖ 文件名

- 用来标识文件的字符串，保存在一个目录文件项中

## ❖ 路径名

- “/”字符结合在一起的一个或多个文件名的集合。它指定一个文件在文件系统中的位置



# 文件结构

- ❖ 无论文件是一个程序、一个文档、一个数据库、或是一个目录，操作系统都会赋予它下面的结构
  - 索引节点（**I节点**）：包含有关相应文件信息（文件权限、文件主、文件大小等）的一个记录
  - 数据：文件的实际内容



## 文件名命名规则

- ❖ 包含大写键、小写键、数字、#、@、\_
- ❖ 不包含空格
- ❖ 不包含以下字符：\* ? > < / ; \$ \ ‘ “
- ❖ 不能以 “+” 或者 “-” 开头
- ❖ 区分大小写
- ❖ 最长文件名：255





# 文件类型

## ❖ 普通文件：包含各种长度的字符串

- 文本文件：由ASCII字符构成
- 数据文件：由来自应用程序的数字型和文本型数据构成  
例如：电子表格、数据库等
- 可执行的二进制程序：由机器指令和数据构成

## ❖ 目录文件：由“**I节点号/文件名**”构成的列表

- I节点号是检索I节点表的下标，I节点存放所有文件的状态信息
- 文件名是给一个文件分配的文本形式的字符串，用来标识文件

## ❖ 设备文件： `/dev/tty1`

## ❖ 连接文件：存放文件系统中通向文件的路径**file 文件名**





## 用户登录系统

❖ 系统启动后，输入用户名并键入回车键，如

➤ **login: root**

❖ 输入用户口令，输入的口令不会在屏幕上显示出来。  
如果输入的口令有误，屏幕提示下列信息

**login incorrect**

**login:**

❖ 执行上述两步后，如果屏幕显示系统提示符,如

➤ **[root@localhost root] #**

➤ 说明已经成功登录到系统中，可以进行操作



# 远程登录系统

## ❖ 用户远程登录机群

- 普通用户通过telnet从机群外部登录到机群结点，首先要通过机群系统管理员建立帐户
- 管理员通过rlogin登录
- 通过ssh命令

## ❖ 在机群内部

- 由于每个普通用户帐户都是一个全局NFS帐户，可以通过rsh在机群内部进行访问

## ❖ 举例

- telnet VIP (VIP为机群系统对外的IP地址, 由用户设定该IP)
  - ✓ Login: team01
  - ✓ Password: \*\*\*\*\*
- rsh node161 (通过rsh访问机群内部的其他结点, node161为机群内结点的主机名)



## 退出系统

❖ 用户使用**Linux**系统之后，要退出系统的过程称为“注销”

❖ 方法有三种

- 键入: **exit**
- 按: **Ctrl+D** (相当于执行**exit**)
- **logout**



## 关闭系统

❖ 必须由**超级用户**在**shell**提示符下，键入关闭命令

- 系统将完整地执行关闭所有进程
- 释放占用资源，停止运行
- 切断电源

❖ 关闭系统的命令

- `[root@localhost /root] # shutdown -h now`
- `[root@localhost /root] # halt`
- `[root@localhost /root] # init 0`
- `[root@localhost /root] # poweroff`





# 关闭系统

## ❖ 重启系统

- **reboot** 命令
- **shutdown -r <time>** 命令
- 组合键 **Ctrl+Alt+Del**（只适用于控制台下）

## ❖ shutdown命令

- 语法: **shutdown [flag] <time> [warning message]**
- **flag:**
  - ✓ **-r** 重启
  - ✓ **-h** 关机
  - ✓ **-k** 不关机，只发消息
- **time:**
  - ✓ 绝对时间: **hh:mm**
  - ✓ 相对时间: **+mm(分钟)**
  - ✓ 立刻关机: **now(=+0)**





## 关闭系统注意事项

- ❖ **Linux**系统有一个磁盘缓存区，这个缓存区不是立即将所有数据写入磁盘的，而是隔段时间后，再将数据写入磁盘。因此，随手关掉电源可能会导致缓存没有回写，磁盘上的文件系统不完整
- ❖ 在**Linux**的多任务系统中，可能有许多程序正置于后台运行，只有通过正确的关机顺序，才可以保证所有的后台进程都能保存自己的数据



## shell命令

### ❖ 命令格式

➤ **command options arguments**

❖ 在shell提示符下，输入命令，然后按下**Enter**键

❖ shell识别大小写

❖ 如果找不到你输入的命令，会显示反馈信息：  
“**Command not Found**”

❖ 如果命令太长，要在第一行行尾键入 “\”字符和按下**Enter**键，在下一行的 “>”后接着输入



# 键盘快捷方式

## ❖ <ctrl-c>

- 停止命令

## ❖ <ctrl-d>

- 结束传输或者文件

## ❖ <ctrl-s>

- 临时停止输出

## ❖ <ctrl-q>

- 恢复输出

## ❖ <ctrl-u>

- 擦除整行

## ❖ <backspace>

- 纠正错误



## shell命令补齐功能

❖ 命令补齐是指当键入的字符足以确定目录中一个唯一的文件时，只须按 **Tab**键就可以自动补齐该文件名的剩下部分

❖ 举例

➢ `[root @redflag /root]#hist` **【Tab】**

➢ 系统将会自动帮助用户完成命令

✓ `[root @redflag /root]#history`





# 特殊bash变量

## ❖ HISTFILE

- 用于贮存历史命令的文件

## ❖ HISTSIZE

- 历史命令列表的大小

## ❖ HOME

- 当前用户的用户目录

## ❖ OLDPWD

- 前一个工作目录

## ❖ PATH

- bash寻找可执行文件的搜索路径

## ❖ PS1

- 命令行的一级提示符

## ❖ PS2

- 命令行的二级提示符

## ❖ PWD

- 当前工作目录

## ❖ SECONDS

- 当前shell开始后所流逝的秒数





## 历史记录

- ❖ **bash** 保留了一定数目的先前已经在**shell** 里输入过的命令
- ❖ 这个数目取决于一个叫做**HISTSIZE**的变量。
- ❖ 使用历史记录列表最简单的方法是用**上方向键**
- ❖ 另一个使用命令历史文件的方法是用 **bash** 的内部命令 **history** 和 **fc**(**fix** 命令)命令来显示和编辑历史命令



# history命令

## ❖ history有两种不同的调用方法

### ➤ history [n]

- ✓ 当 history 命令没有参数时，整个历史命令列表的内容将被显示出来
- ✓ 使用 n 参数的作用是仅有最后 n 个历史命令会被列出
- ✓ 例如，history 5 显示最后 5 个命令。

### ➤ history [-r|w|a|n] [filename]

- ✓ 用于修改命令历史列表文件的内容
- ✓ -r：读命令历史列表文件的内容并把其当作当前的命令历史列表
- ✓ -w：把当前的命令历史记录写入文件中并覆盖文件原来的内容
- ✓ -a：把当前的命令历史记录追加到文件中
- ✓ -n：将读取文件中的内容并加入到当前历史命令列表中
- ✓ 如果 filename 选项没有被指定，history 命令将用变量 HISTFILE 的值来代替



# alias命令

## ❖ 功能

- 命令别名通常是其他命令的缩写，用来减少键盘输入。

## ❖ 格式

- `alias [ alias-name='original-command' ]`

- ✓ alias-name 是用户给命令取的别名

- ✓ original-command 是原来的命令和参数

- 注意点

- ✓ 由于 Bash 是以空格或者回车来识别原来的命令的，所以如果不使用引号就可能导致 Bash 只截取第一个字，从而出现错误

- ✓ 在定义别名时，等号两边不能有空格，否则 shell 将不能决定要做什么。仅在命令中包含空格或特殊字符时才需要引号

- ✓ 如果键入不带任何参数的 alias 命令，将显示所有已定义的别名



## 两级提示符

### ❖ 第一级提示符

- 指在等待命令输入时的情况，默认值是\$符号
- 如果需要重新定义该提示符，只需修改PS1变量的值  
✓ PS1=”输入一个命令：“

### ❖ 第二级提示符

- 是当Bash为执行某条命令需要用户输入更多信息时显示的，默认为“>”
- 如果需要重新定义该提示符，只需修改PS2变量的值。  
✓ PS2=”更多信息：“

- ❖ 用户也可以使用一些事先已经定义好的特殊字符。这些特殊字符将使提示符中包含当前时间之类的信息





## bash提示符常用特殊字符

特殊字符	说明
!\	显示该命令的历史编号
\#	显示shell激活后，当前命令的历史编号
\\$	显示一个\$符号，如果当前用户是root则显示#符号
\\	显示一个反斜杠
\d	显示当前日期
\h	显示运行该shell的计算机主机名
\n	打印一个换行符，这将导致提示符跨行
\s	显示正在运行的Shell的名称
\t	显示当前时间
\u	显示当前用户的用户名
\W	显示当前工作目录基准名
\w	显示当前工作目录





## 举例

```
root@bliubing: /linuxbook/nic
文件(E)  编辑(E)  查看(V)  终端(T)  转到(G)  帮助(H)
[root@bliubing root]# alias nic="cd linuxbook/nic"
[root@bliubing root]# nic
[root@bliubing nic]# unalias
[root@bliubing nic]# nic
bash: cd: linuxbook/nic: 没有那个文件或目录
[root@bliubing nic]#
[root@bliubing nic]# PS1="输入一个命令:"
输入一个命令:PS1="\d"
— 2月 02date
— 2月 2 10:43:59 CST 2004
— 2月 02PS1="[\W@\h \w]"
[nic@bliubing ~/linuxbook/nic]
```



## 用户变量

- ❖ 变量名:可以由字母开头的任意字母、数字组成的序列
- ❖ 定义用户变量形式: `set var=sting`
- ❖ 取消变量定义: `unset var`
- ❖ 显示变量值: `echo`
- ❖ 示例
  - `set int=5`: 要生成一个值为整数的变量
  - 执行操作
    - ✓ `set var1=abcd`
    - ✓ `set var2=var1$efgh`
  - 执行上面两条语句, 变量`var2`的内容为: `abcdefgh`
  - `echo $var2`: 显示上面`var2`的值
    - ✓ 则输出: `abcdefgh`



## 操作环境设置技巧

### ❖ 引用常用的环境变量减少操作步骤

- 常用环境变量: **HOME**用户主目录; **PATH**检索路径;  
**SHELL**当前所用shell; **TERM**终端类型。如:

✓ **cd \$HOME**与 **cd**、**cd ~**一样, 将当前目录设置成自己的**用户主目录**

### ❖ 使用仿真终端提供的功能

- 如复制与粘贴可以减少键入的“笔误”
- 命令不能退出、结果难于预料时, 及时使用<sup>^</sup>C中断运行, 一般不能采用关闭终端的办法

### ❖ 使用**BASH**的“命令行编辑”功能, 方便命令的调试



# 通配符基础

## ❖ 基本通配符

- ? : 表示该位置可以是一个任意的单个字符
- \* : 表示该位置可以是若干个任意字符
- 方括号[charset]: 可替代charset集中的任何单个字符

## ❖ 示例

- [cChH]: 表示在文件的该位置中可出现任意单个的c或h字符的大小写形式
- [a-z]: 代替任意小写字母
- [a-zA-Z]: 可替代任意字母



## 通配符应用

### ❖ 在一条指令中用多个通配符

➤ `rm a*out*tmp?`

✓ 该命令可以删除一系列临时性的输出文件，如  
`ab.out.temp1`、`ab.out.temp1` 等

### ❖ UNIX或Linux系统可将一定相关的文件看作一个集合的一部分，用户可以用该集合去匹配

➤ 如果需要显示`nic-1.png`，`nic-2.png`，`nic-3.png`，`nic-4.png`，`nic-5.png`，只须要在终端的命令提示符后输入：

✓ `ls nic-[1-5].png`





## shell命令的输入和输出

### ❖ 执行一个shell命令时通常会自动打开三个标准文件

- 标准输入文件（stdin）：通常对应终端键盘
- 标准输出文件（stdout）：对应终端屏幕
- 标准错误输出文件（stderr）：对应终端屏幕

### ❖ 进程I/O处理过程

- 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
- 将错误信息送到标准错误文件中

### ❖ Linux系统为输入、输出的传送引入了另外两种机制

- 输入/输出重定向
- 管道



## 输入重定向

- ❖ 把命令（或可执行程序）的标准输入重定向到指定的文件中。输入重定向主要用于改变一个命令的输入源，特别是改变那些需要大量输入的输入源
- ❖ 另一种输入重定向称为**here文档**
  - 它告诉shell当前命令的标准输入来自命令行
  - **here文档**的重定向操作符使用 “<<”
  - 它将一对分隔符（**! ...!**）之间的正文重定向输入给命令
- ❖ 由于大多数命令都以参数的形式在命令行上指定输入文件的文件名，所以输入重定向并不经常使用
- ❖ 当要使用一个不接受文件名作为输入参数的命令，而需要的输入内容又存在一个文件里时，就能用输入重定向解决问题



## 输出重定向

- ❖ 把命令（或可执行程序）的标准输出或标准错误输出重新定向到指定文件中。这样，该命令的输出就不显示在屏幕上，而是写入到指定文件中
- ❖ 形式
  - 命令>文件名：覆盖原文件
  - 命令>>文件名：文件末追加信息



## 管道

- ❖ 将一个程序或命令的输出作为另一个程序或命令的输入
- ❖ 方法
  - 一种是通过一个临时文件将两个命令或程序结合在一起
  - 另一种是Linux所提供的管道功能
- ❖ 管道可以把一系列命令连接起来，这意味着第一个命令的输出会作为第二个命令的输入通过管道传给第二个命令，第二个命令的输出又会作为第三个命令的输入，以此类推
- ❖ 显示在屏幕上的的是管道行中最后一个命令的输出（如果命令行中未使用输出重定向）





## 命令替换

- ❖ 命令替换和重定向有些相似，但区别在于命令替换是将一个命令的输出作为另外一个命令的参数
- ❖ 命令格式
  - `command1 `command2``
    - ✓ `command2`的输出将作为`command1`的参数
    - ✓ 需要注意的是这里的`符号，被它括起来的内容将作为命令执行，执行后的结果作为`command1`的参数
- ❖ 示例
  - `$ cd `pwd``
    - ✓ 该命令将`pwd`命令列出的目录作为`cd`命令的参数，结果仍然是停留在当前目录下





# 进程基本管理

## ❖ 分类

- **前台进程**: 用户运行一个程序或执行一个命令就启动一个前台进程, 进程不结束, 终端就不出现系统提示符。
- **后台进程**: 用户在输入命令行后加上” **&**”字符就启动了一个后台进程, shell不等待命令退出, 立即重新显示提示符, 让该命令进程在后台运行
- **bg**命令可以将进程放到后台运行
- **fg**命令可以将后台进程放到前台运行

## ❖ 举例

- `[jjpr@zzh jjpr]$ sleep 10&`

`[1] 467`

`[jjpr@zzh jjpr]$`

1为作业号, 由shell分配。467为PID  
作业号不同于PID, 在系统中不一定唯一

- `find / -name myfile -print > /root/test &`



# 特殊进程

## ❖ 精灵（daemon）进程

- 又称守护进程，与终端和用户无关，负责侦听用户请求或者定期执行，平时则处于睡眠状态。

## ❖ 孤儿进程

- 父进程已经被删除或闲置的进程

## ❖ 僵尸进程

- 进程已经终止，但还没有撤消会影响系统效率



# ps命令

## ❖ 功能

- 查看系统中正在运行的进程

## ❖ 语法

- `ps [-ef][-n name][-t ttys][-p pids][-u users][-groups]`

## ❖ 参数选项

- `-f`: 产生某个进程的一个完整清单
- `-u`: 显示进程的用户名和启动时间等信息
- `-t n`: 显示第*n*个终端的进程
- `-e`: 显示所有的进程

## ❖ 说明

- `ps`可查看后台进程、前台进程，当`ps`命令行没有选项时，只显示与控制终端相关进程的基本信息
- 没有root权限，`ps`仅限以说明运行进程，报告用户的进程讯息



# ps命令

## ❖ 示例

### ➤ ps -ef 输出

用户 ID    进程号    父进程号    进程占用CPU的百分比    启动进程的终端号    进程开始的时间和日期

```
[root@localhost root]# ps -ef | more
UID          PID    PPID  C   STIME TTY          TIME CMD
root          1        0   2  20:51 ?           00:00:04 init
root          2        1   0  20:51 ?           00:00:00 [keventd]
root          3        1   0  20:51 ?           00:00:00 [kapmd]
root          4        1   0  20:51 ?           00:00:00 [ksoftirqd/0]
root          7        1   0  20:51 ?           00:00:00 [bdf flush]
root          5        1   0  20:51 ?           00:00:00 [kswapd]
root          6        1   0  20:51 ?           00:00:00 [kscand]
```

进程已占用的时间    运行的命令





# kill命令

## ❖ 功能

- 给进程发送信号

## ❖ 语法

- **kill** [参数] 进程1的PID 进程2的PID...]

## ❖ 参数

- **-s signal**: **signal**是信号类别，如**SIGKILL**

## ❖ **kill -l**：显示kill命令所能发送的信号种类，每个信号都有一个数值对应。 如：

编号	名字	含义
1	SIGHUP	挂起
2	SIGINT	中断 (对前台进程中断)
9	SIGKILL	中止 (不可捕捉和忽略，强行消亡)
15	SIGTERM	从kill来的软件中断信号 (默认) 该信号将通知进程退出。如果进程不接受该信号，可以通过参数 <b>-9</b> 强行结束进程。





## top命令

### ❖ 功能

- 实时监控进程状况
- top屏幕自动每5秒刷新一次，也可以用“top -d 20”，使得top屏幕每20秒刷新一次



# 主要内容

## ❖ 背景知识

- Linux简介
- Linux系统环境
- **Linux用户管理命令**
- Linux文件操作命令
- Linux网络管理命令
- Linux系统信息命令
- Linux编程环境

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核



## 用户账号文件(pwd)

- ❖ **pwd** 是一个文本文件，用于定义系统的用户账号，该文件位于 “**/etc**”目录下
- ❖ 包含了一个系统账户列表，给出每个账户一些有用的信息，例如，用户 **ID**、组 **ID**、主目录、**shell**等等
- ❖ 由于**所有用户**都对**pwd**有**读权限**，所以该文件中**只定义用户账号**，而**不保存口令**
- ❖ **pwd**文件中每行定义一个用户账号，一行中又划分为多个字段定义用户的账号的不同属性，各字段用 “**:**” 隔开



# 用户账号文件(pwd)

```
root@bliubing:~  
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)  
[root@bliubing root]# ll /etc/passwd  
-rw-r--r-- 1 root root 2405 2月 1 15:35 /etc/passwd  
[root@bliubing root]# head /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
bin:x:1:1:bin:/bin:/sbin/nologin  
daemon:x:2:2:daemon:/sbin:/sbin/nologin  
adm:x:3:4:adm:/var/adm:/sbin/nologin  
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin  
sync:x:5:0:sync:/sbin:/bin/sync  
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown  
halt:x:7:0:halt:/sbin:/sbin/halt  
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin  
news:x:9:13:news:/etc/news:  
[root@bliubing root]#
```





# passwd文件各字段说明

## ❖ Account

- 使用者在系统中的名字，它不能包含大写字母

## ❖ Password

- 用户口令，出于安全考虑，现在不使用该字段保存口令，而用字母“**x**”来填充该字段，真正的密码保存在shadow文件

## ❖ UID

- 用户 ID 号，惟一表示某用户的数字

## ❖ GID

- 用户所属的私有组号，该数字对应group文件中的GID

## ❖ GECOS

- 这字段是可选的，通常用于保存用户命名的信息

## ❖ Directory

- 用户的主目录，用户成功登录后的默认目录

## ❖ shell

- 用户所使用的shell，如该字段为空则使用“/bin/sh”





## 用户口令文件（shadow）

- ❖ 每行定义了一个用户信息，行中各字段各字段用“:”隔开
- ❖ 为进一步提高安全性，shadow文件中保存的是已加密的口令

- 登录名
- 加密口令
- 口令上次更改时距1970年1月1日的天数
- 口令更改后不可以更改的天数
- 口令更改后必须再更改的天数(有效期)
- 口令失效前警告用户的天数
- 口令失效后距账号被查封的天数
- 账号被封时距1970年1月1日的天数
- 保留未用

```
root@lbliubing:~  
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)  
[root@lbliubing root]# ll /etc/shadow  
-r----- 1 root root 1504 2月 1 15:35 /etc/shadow  
[root@lbliubing root]# head /etc/shadow  
root:$1$VdQwngYn$Rf7SwoY5CXzfZh8T9Qt6Yl:12412:0:99999:7:::  
bin:!:12412:0:99999:7:::  
daemon:!:12412:0:99999:7:::  
adm:!:12412:0:99999:7:::  
lp:!:12412:0:99999:7:::  
sync:!:12412:0:99999:7:::  
shutdown:!:12412:0:99999:7:::  
halt:!:12412:0:99999:7:::  
mail:!:12412:0:99999:7:::  
news:!:12412:0:99999:7:::  
[root@lbliubing root]#
```



## 用户组账号文件（group）

- ❖ 用户组是逻辑地组织用户账号集合的方便途径，它允许用户在组内共享文件
- ❖ 系统上的每一个文件都有一个用户和一个组的属主。使用“**ls -l**”命令可以看到每一个文件的属主和组
- ❖ 对于系统上的每个组，在**/etc/group**文件中有一行记录，记录的格式为

➢ **groupname : passwd : GID : userlist**

### ❖ group字段说明

- **groupname**: 是组的名字
- **passwd**: 组的加密口令
- **GID**: 系统区分不同组的ID，在**/etc/passwd**域中的**GID**域是用这个数来指定用户的缺省组
- **userlist**: 用“,”分开的用户名，列出这个组的成员



## group文件信息

```
root@bliubing:
文件(E) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
[root@bliubing root]# ls -l /etc/group
-rw-r--r--  1 root    root      798  2月  1 15:35 /etc/group
[root@bliubing root]# head /etc/group
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
tty:x:5:
disk:x:6:root
lp:x:7:daemon,lp
nem:x:8:
knem:x:9:
[root@bliubing root]#
```



## 用户组口令文件（gshadow）

- ❖ 用于定义用户组口令、组管理员等信息
- ❖ 该文件只有root用户可以读取
- ❖ gshadow文件中每行定义一个用户组信息，行中各字段间用“:”分隔，每行记录的格式为：
  - `groupname : encrypted password: group administrators: group members`
- ❖ 各字段含义
  - `groupname`: 用户组名称，该字段与group文件中的组名称对应
  - `encrypted password`: 用户组口令，该字段用于保存已加密的口令
  - `group administrators`: 组的管理员账号，管理员有权对该组添加删除账号
  - `group members`: 属于该组的用户成员列表，列表中多个用户间用“,”分隔





## gshadow文件信息

```
root@bliubing:~  
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)  
[root@bliubing root]# ls -l /etc/gshadow  
-r----- 1 root root 662 2月 1 15:35 /etc/gshadow  
[root@bliubing root]# head /etc/gshadow  
root:::root  
bin:::root,bin,daemon  
daemon:::root,bin,daemon  
sys:::root,bin,adm  
adm:::root,adm,daemon  
tty:::  
disk:::root  
lp:::daemon,lp  
nem:::  
knem:::  
[root@bliubing root]#
```





# 增加用户帐号(useradd)

## ❖ 命令

- `useradd -D [-g group][--b base][--s shell][--f inactive][--e expire]`

## ❖ 参数

- **-g**: 用于添加用户账号时指定该用户的私有组。如不指定“-g”参数，`useradd`命令将自动建立与用户账号同名的组作为该账号的私有组
- **-D**: 用于显示或设置`useradd`命令所使用的默认值

## ❖ 该命令工作机制

- 在`/etc/passwd`文件中增添了一行记录
- 在`/home`目录下创建新用户的主目录，并将`/etc/skel`目录中的文件拷贝到该目录中去
- 但是使用了该命令后，新建的用户暂时还无法登录，因为还没有为该用户设置口令
- 需要再用`passwd`命令为其设置口令后，才能登录
- 用户的UID和GID是`useradd`自动选取的，它是将`/etc/passwd`文件中的UID加1，将`etc/group`文件中的GID加1



## 增加用户帐号

- ❖ 增加新用户时，系统将为用户创建一个与用户名相同的组，称为私有组
- ❖ 举例：增加一个用户 “liuyidan”
  - `#useradd liuyidan` //建立用户账号
  - `#tail -l /etc/passwd` //查看password文件中添加的用户账号信息
  - `#tail -l /etc/shadow`
  - `#ls /home` //查看所建立账号的主目录



## 修改用户账号(usermod)

❖ 修改用户帐号的各种属性，包括用户主目录、私有组、登录、**shell**等内容

### ❖ 命令格式

➢ **usermod** [-LU][-c <备注>][-d <登入目录>][-e <有效期限>][-f <缓冲天数>][-g <群组>][-G <群组>][-l <帐号名称>][-s ][-u ][用户帐号]

### ❖ 参数说明

- **-c<备注>**: 修改用户帐号的备注文字
- **-d<登入目录>**: 修改用户登入时的目录
- **-e<有效期限>**: 修改帐号的有效期限
- **-f<缓冲天数>**: 修改在密码过期后多少天即关闭该帐号
- **-g<群组>**: 修改用户所属的群组



## 修改用户账号(usermod)

### ❖ 示例

- 修改用户名，把用户名“liuyidan”改名为“lyd”，使用的命令是
  - ✓ # usermod -l lyd liuyidan
- 锁定“lyd”用户，使其不能登录。命令如下
  - ✓ # usermod -L lyd
- 解锁“lyd”用户账号，使其可以登录。命令如下
  - ✓ # usermod -U lyd





# 删除用户 (userdel)

## ❖ 功能

- 删除指定的用户账号

## ❖ 语法格式

- `userdel [-r][用户账号]`

## ❖ 说明

- `userdel`命令可删除用户账号与相关的文件
- 参数“-r”是用来删除用户登入目录以及目录中所有文件。若不加参数，则仅删除用户账号，而不删除相关文件

## ❖ 示例

- `#grep lyd /etc/passwd` //查询用户账号lyd是否存在
- `#userdel lyd` //删除lyd账号
- `#grep lyd /etc/passwd` //再次查询用户账号lyd是否存在
- `#ls -d /home` //查询用户lyd的主目录是否还存在
- `#userdel -r lyd` //删除用户的同时删除其工作主目录



# 组增加命令(groupadd)

## ❖ 功能

- 可指定群组名称来建立新的群组账号，该组账号的ID值必须是惟一的，且数值不可为负
- 预设的最小值不得小于500，且每增加一个组账号ID值逐次增加。
- ID值0~499是保留给系统账号使用

## ❖ 语法格式

- `groupadd [-r] group`
  - ✓ 其中“-r”参数是用来建立系统账号。系统账号的ID值不能大于500

## ❖ 示例

- `# groupadd lbgroup //建立组账号lbgroup`
- `# grep lbgroup /etc/group //查询group文件中lbgroup组是否建立`
- `#groupadd -r syslbgroup //建立系统组账号`
- `# grep lbgroup /etc/group //查询group文件中syslbgroup组是否`



# 组账号修改 (groupmod)

## ❖ 功能

- 用来更改群组识别码或名称

## ❖ 语法格式

- `groupmod [-g <群组识别码> <-o>][-n <新群组名称>][群组名称]`

## ❖ 参数说明

- `-g <群组识别码>`: 设置欲使用的群组识别码
- `-o`: 重复使用群组识别码
- `-n <新群组名称>`: 设置欲使用的群组名称

## ❖ 举例

- `# grep lbgrou /etc/group` //查询group文件中lbgrou组属性
- `#groupmod -g 503 lbgrou` //改变lbgrou组的GID为503
- `# grep lbgrou /etc/group` //查询操作结果是否正确
- `#groupmod -n ydgroup lbgrou` //改变lbgrou组名为ydgroup
- `# grep 503 /etc/group` //查询操作结果是否正确



# 删除组账号（groupdel）

## ❖ 功能

- 删除指定的组账号
- 若该群组中仍包括某些用户，则必须先删除这些用户后，方能删除群组

## ❖ 语法格式

- `groupdel [群组名称]`





# 口令维护命令(pwd)

## ❖ 功能

- 使用useradd命令增加时，还需用pwd命令为新增用户设置口令
- 可以随时用pwd命令改变自己的口令

## ❖ 格式

- pwd [用户名]
  - ✓ 其中用户名为需要修改口令的用户名
  - ✓ 只有超级用户可以使用“pwd 用户名”修改其他用户的口令
  - ✓ 普通用户只能用不带参数的pwd命令修改自己的口令
- root用户pwd命令还可以使用一些参数选项这些参数选择包括
  - ✓ -S: 用于查询指定用户账号的状态
  - ✓ -l: 用于锁定账号的口令
  - ✓ -u: 解除锁定账号的口令
  - ✓ -d: 删除指定账号的口令



## 组中用户成员的维护 (gpasswd)

❖ 用于把一个账户添加到组、把一个账户从组中删除、把一个账户设为组管理员

➤ 添加用户到使用的命令格式为

✓ `gpasswd -a 用户账号名 组账号名`

➤ 从组中删除用户的命令格式为

✓ `gpasswd -d 用户账号名 组账号名`

➤ 设置用户为组管理员的命令格式为

✓ `gpasswd -A 组管理员用户列表 用户组`



# id命令

## ❖ 功能

- 用于显示用户当前的UID，gid以及所属群组的组列表

## ❖ 语法格式

- `id [选项] [用户名称]`

## ❖ 参数说明

- `-g`: 显示用户所属群组的ID
- `-G`: 显示用户所属附加群组的ID
- `-n`: 显示用户，所属群组或附加群组的名称
- `-r`: 显示实际ID
- `-u`: 显示用户ID



# su命令

## ❖ 功能

- 用来将当前用户转换为其他用户身份

## ❖ 语法格式

- `su [-flmp] [-][-c <指令>][[-s ][用户帐号]`
  - ✓ su命令可让用户暂时变更登入的身份
  - ✓ 变更时须输入所要变更的用户账号与密码

## ❖ 参数说明

- `-c<指令>`: 执行完指定的指令后，即恢复原来的身份
- `-f`: 适用于csh与tsch，使shell不用去读取启动文件
- `-`: 改变身份时，也同时变更工作目录，以及HOME，SHELL，USER，LOGNAME。此外，也会变更PATH变量
- `-m`，`-p`: 变更身份时，不要变更环境变量
- `-s`: 指定要执行的shell
- `[用户帐号]`: 指定要变更的用户。若不指定此参数，则预设变更为root





# 主要内容

## ❖ 背景知识

- Linux简介
- Linux系统环境
- Linux用户管理命令
- **Linux文件操作命令**
- Linux网络管理命令
- Linux系统信息命令
- Linux编程环境

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核



# 文件目录类命令汇总

## ❖ 浏览目录命令

➤ **cd**、**dir**、**ls**、**ll**

## ❖ 浏览文件命令

➤ **cat**、**more**、**less**

## ❖ 目录操作命令

➤ **mkdir**、**rmdir**

## ❖ 文件操作命令

➤ **touch**、**rm**、**cp**、**mv**、**ln**、**tar**、**gzip**、**gunzip**、**whereis**、**whatis**



# ls命令

## ❖ 语法

- `ls [参数] [路径或文件名]`

## ❖ 功能

- 列出文件或子目录的信息

## ❖ 参数

- `-a`：显示所有文件，包括以.`.`开头的隐藏文件
- `-l`：以长格式显示文件或子目录的信息
- `-i`：显示每个文件的索引(节点)号
- `-R`：显示目录及下级子目录结构
- `-S`：以文件大小排序



# ls命令

## ❖ ls -l输出格式

- 以长格式形式在每行显示一个目录或文件名，如：

```
drwxr-xr-x  2 jjpr1  group   512 Nov 18 10:24 .  
drwxrwxr-x 24 root   auth    512 Nov 16 10:33 ..  
-r-----  1 jjpr1  auth    0 Nov 18 10:24 .lastlogin
```

- 每一行的组成部分

权限 链接数 文件主 组 长度 建立/修改时间 目录或文件名

-rw- r-- r-- 1 jjpr jjpr 1299 Nov 200 9:33 hs

所有者 同组用户 其他组用户 八进制表示为644





# ls命令

## ❖ ls输出信息格式说明

### ➤ 用颜色代表不同文件

蓝色：目录

绿色：可执行文件

红色：压缩文件

浅蓝色：链接文件

灰色：其他文件

### ➤ 在ls -l显示结果中，**第一个字符**表示文件的类型

✓ -: 普通文件

✓ d: 目录

✓ c: 字符设备

✓ b: 块设备

✓ l: 链接文件



## cd命令

❖ 功能：切换目录

❖ 示例

➤ **cd [目录名]**

✓ [pp@linux home]\$ cd pp: 切换到当前目录下的pp子目录

✓ [pp@linux pp]\$ cd ..: 切换到上一层目录

✓ [pp@linux home]\$ cd /: 切换到系统根目录

✓ [pp@linux /]\$ cd: 切换到用户自家目录（或执行cd ~）

✓ [pp@linux pp]\$ cd /usr/bin: 切换到/usr/bin目录

➤ **cd ~user**

✓ 切换到user的注册目录



# pwd命令

## ❖ 功能

- 显示用户正在工作或当前所在的目录

## ❖ 格式

- `pwd`

## ❖ 举例

- `[pp@linux pp]$ pwd`

`/home/pp`

✓ 显示用户pp所在的当前目录是/home/pp



# mkdir命令

## ❖ 语法

- **mkdir [参数] 目录名**

## ❖ 参数选项

- **-p**: 建立目录时，如果父目录不存在，则此时可以与子目录一起建立，即一次可建立多级目录

## ❖ 举例

- **mkdir -p dir2/bak**
  - ✓ 在dir2目录下建立bak目录，如果dir2目录不存在，那么同时建立dir2目录
- **mkdir -p -m 700 newdir/subdir**
  - ✓ 同时创建父目录和子目录并指定权限（模式）





# rmmdir命令

## ❖ 语法

- `rmmdir [参数] 目录名`

## ❖ 选项

- `-p`: 一起删除父目录时，父目录下应无其他目录

## ❖ 举例

- `[root @redflag /root]#rmmdir test`

✓ 删除当前目录下的test目录。删除目录时，**被删除的目录下应无文件或子目录存在**

- `[root @redflag /root]#rmmdir -p longkey/test`

✓ 删除当前目录下的longkey/test目录。删除目录test时，**如果父目录longkey下无其他内容，则一起删除longkey目录**



# cp命令

## ❖ 功能

- 将文件复制为另一文件，或将数个文件复制到一个目录

## ❖ 语法

- `cp [options] sourcefile destfile`
- `cp [options] sourcefile ... directory`

## ❖ 参数

- `-p`: 保持原始文件属性
- `-f`: 如果目标文件已经存在，则覆盖它
- `-i`: 提示是否覆盖现有的普通目标文件
- `-r R`: 递归复制目录，包含目录下的各级子目录，`-R`允许拷贝设备节点和命名管道

✓ 注意：递归拷贝时，目的目录不能在原目录下

✓ 如：`cp -r //targetdir`，会使系统死循环或瘫痪



# cp命令

## ❖ 举例

➤ `$ cp aaa bbb`

✓ 将文件 aaa 复制(已存在), 并命名为 bbb

➤ `$ cp *.c Finished`

✓ 将所有C语言程序拷贝至 Finished 目录中



# rm命令

## ❖ 功能

- 删除文件或目录

## ❖ 格式

- **rm**[参数] <目标文件路径>

## ❖ 参数

- **-f**: 不加提示地删除已存在的文件
- **-i**: 交互删除
- **-r R**: 递归删除整个目录

## ❖ 举例

- **[pp@linux pp]\$ rm -f \*.txt**  
✓ 强迫删除所有以后缀名为txt文件





## rm命令

### ❖ rm命令参数-i使用

- -i参数: 删除文件时询问
- [pp@linux pp]\$ rm -i \*
- ✓ 删除当前目录下的所有文件
- ✓ rm:backup: is a directory ← 遇到目录会略过
- ✓ rm : remove 'myfiles.txt' ? Y
- 删除文件时会询问,可按Y或N键表示允许或拒绝删除文件
- 注意
  - ✓ 在系统的默认状态下, rm命令会对每个删除的文件一一询问
  - ✓ 如果用户确定要删除这些文件, 则可以使用参数-f来避免询问



## rm命令

### ❖ rm命令参数-r、-f的使用

- 递归删除（连子目录一同删除），这是一个相当常用的参数
- [pp@linux pp]\$ rm -r test
  - ✓ 删除test目录（含test目录下所有文件和子目录）
- [pp@linux pp]\$ rm -r \*
  - ✓ 删除所有文件（含当前目录所有文件、所有子目录和子目录下的文件）一般在删除目录时r和f一起用，避免麻烦
- [pp@linux pp]\$ rm -rf test
  - ✓ 强行删除、不加询问



# mv命令

## ❖ 功能

- 移动或更名现有文件或目录

## ❖ 语法

- `mv [ -fi ] source_file ... target_directory`
- `mv source_file target_file`（文件更名）

## ❖ 参数

- `-f`: 禁止提示
- `-i`: 目标文件或目录存在时，提示是否覆盖

## ❖ 举例

- `[pp@linux dir1]$ mv a.txt ../`: 将a.txt文件移动上层目录
- `[pp@linux dir1]$ mv a.txt b.txt`: 将a.txt改名为b.txt
- `[pp@linux dir1]$ mv dir2 ../`: 将dir2目录上移一层



# mvdir命令

## ❖ 功能

- 移动目录

## ❖ 格式

- **mvdir dirname newdirname**

## ❖ 注意

- 必须在一个文件系统





# ln命令

## ❖ 功能

- 该命令在文件之间创建链接
- 这种操作实际上是给系统中已有的某个文件指定另外一个可用于访问它的名称

## ❖ 语法

- ln [-s] 目标 链接名

## ❖ 参数

- -s: 建立符号链接(软链接symbolic link)
- 不加-s参数: 建立硬链接(hard link)
- 目标: 源文件或目录

## ❖ 说明

- 建立硬链接时, 链接文件和被链接文件必须位于同一个文件系统中, 并且不能建立指向目录的硬链接
- 而对符号链接, 则不存在这个问题。默认情况下, ln产生硬链接



# ln命令

## ❖ 硬链接

- 如果[链接名]是一个目录名，系统将在该目录之下建立一个或多个与“目标”同名的链接文件，链接文件和被链接文件内容完全相同
- 若[链接名]为一个文件，用户将被告知该文件已存在且不进行链接
- 如果指定了多个“目标”参数，那么最后一个参数必须为目录

## ❖ 符号链接

- 如果[链接名]已经存在但不是目录，将不做链接
- [链接名]可以是任何一个文件名（可包含路径），也可以是一个目录，并且允许它与“目标”不在同一个文件系统中
- 如果[链接名]是一个已经存在的目录，系统将在该目录下建立一个或多个与“目标”同名的文件，此新建的文件实际上是指向原“目标”的符号链接文件

## ❖ 提示

- 删除文件时，只有所有的链接全部删除，文件或目录才被删除



# cat命令

## ❖ 功能

- 显示文件的内容，也可以将数个文件合并成一个文件

## ❖ 格式

- `cat[参数]<文件名>`

## ❖ 常见的几种用法

- `cat` 输入模式
  - ✓ 显示用户输入的每一行数据，`Ctrl+d`结束
- `cat [-n] file`
  - ✓ 显示文件的内容，`-n`可加行号显示
- `cat >file`
  - ✓ 建立简短文本文件，`Ctrl+d`结束
- `cat file1 file2 ..... >filen`
  - ✓ 将多个文件集中到一个文件中
- `cat file1 >> file2`
  - ✓ 连接两个文件



# cat命令

## ❖ 举例

- [pp@linux pp]\$pp cat test.txt
  - ✓ 显示test.txt文件内容
- [pp@linux pp]\$pp cat test.txt | more
  - ✓ 逐页显示test.txt文件中的内容
- [pp@linux pp]\$pp cat test.txt >>test1.txt
  - ✓ 将test.txt的内容附加到test1.txt文件之后
- [pp@linux pp]\$pp cat test.txt test2.txt >readme.txt
  - ✓ 将test.txt和test2.txt文件合并成readme.txt文件





# more命令

## ❖ 功能

- 用于要显示的内容会超过一个画面长度的情况，让画面在显示满一页时暂停，此时可按空格键继续显示下一个画面，或按Q键停止显示

## ❖ 语法

- `more [-option] [ file ... ]`

## ❖ 常见几种用法

- `more file` 分屏显示file的内容
- `ls | more` 查找相关目录、文件
- `ls -al | more` 详细查找相关目录、文件信息
- `ps -ef | more` 查找相关进程信息



# less命令

## ❖ 功能

- less命令的用法与more命令类似，也可以用来浏览超过一页的文件
- 所不同的是less命令除了可以按空格键向下显示文件外，**还可以利用上下键来滚动文件**
- 当要结束浏览时，只要在less命令的提示符“:”下按**Q**键即可

## ❖ 举例

- [pp@linux etc]\$less named.conf
  - ✓ 显示/etc/named.conf的文本文件内容
- [pp@linux etc]\$ls -al | less
  - ✓ 以长格形式列出/etc目录中所有的内容，用户可按上下键浏览或按Q键跳离



# head命令

## ❖ 功能

- 显示文件的前几行

## ❖ 语法

- `head [参数] 文件名`

## ❖ 参数

- `-n num` : 显示文件的前num行
- `-c num` : 显示文件的前num个字符
- 缺省时, head显示文件的前10行

## ❖ 示例

- `head textfile1`
  - ✓ 显示textfile1文件的开始10行内容
- `head -20 textfile2`
  - ✓ 显示textfile2文件的开始20行内容



# tail命令

## ❖ 功能

- 显示文件的末尾几行

## ❖ 语法

- `tail[-f] [-c number]/[-n number] [file]`

## ❖ 参数

- `-f`: 使命令进入无限循环
- `-c`: 后的数字为字节位移, 缺省为10
- `-n`: 行位移, 缺省为10

## ❖ 示例

- `[root@linux root]# tail -10 /etc/passwd`
  - ✓ 显示/etc/passwd/文件的倒数10行内容
- `[root@linux root]# tail +10 /etc/passwd`
  - ✓ 显示/etc/passwd/文件的从第10行开始到末尾的内容





# sort命令

## ❖ 功能

- 将文件的内容排序输出

## ❖ 语法

- `sort [参数] 文件列表`

## ❖ 参数

- **-r**: 逆向排序，否则，从小到大排序
- **-n**: 按数值排序。否则，关键字以字符串比较大小
- **-t**: 指定字段（域）分割符。缺省是空格或Tab
- **-k start [,end]**: 限定关键字
  - ✓ **start**和**end**格式**m[n]**，m字段号，n字符号。
  - ✓ 默认从1开始，end缺省到行尾。
  - ✓ 如-k 2.5,2.9表示指定以第二字段的第5个字符至第9个字符为关键字。

## ❖ 示例

- 找出当前目录下字节数最大的5个文件：
  - ✓ `ls -l | sort -k5,5 -nr | head -n5`
- 将/etc/passwd中含有xxj050206??的账号找出来并以??为关键字逆向排序
  - ✓ `grep xxj050206 /etc/passwd | sort -t: -k1.10,1.11 -nr`



# uniq命令

## ❖ 功能

- 比较相邻的行，显示不重复的行

## ❖ 语法

- `uniq 文件名`

## ❖ 示例

- `uniq b.txt`

## ❖ 注意

- 该命令只是去掉相邻的重复行，不相邻的行并不被过
- `uniq`常和`sort`一起使用
  - ✓ 例如：`sort b.txt | uniq`



# file命令

## ❖ 功能

- 显示文件或目录的类型

## ❖ 语法

- **file** 文件名或目录

## ❖ 示例

- `[xxj05020612@s53 xxj05020612]$ file *`

**Desktop:**      **directory**

**mlzy1.12:** **ASCII English text, with overstriking**

**mlzy1.12.bak1:** **ASCII English text, with overstriking**

**mlzy2.12:**     **ISO-8859 text**

**xx05020612:**   **directory**



# find命令

## ❖ 功能

- 查找文件

## ❖ 语法

- **find** 搜索路径 匹配表达式

## ❖ 常用的匹配条件

- **-name filename**: 要查找的文件名。可使用通配符\*? , 但要加 “”
- **-user username**: 查找属于username的文件
- **-print**: 显示找到的文件路径名。通常要选。例如:
  - ✓ **find . -name “h\*” -print**: 查找当前目录下h开头的文件
  - ✓ **find / -name hosts -print**: 查找系统中名为hosts的文件





# find命令

## ❖ 常用的匹配条件

- **-exec cmd** 对找到的匹配的文件，执行由**cmd**表示的命令，参数**{}** 由find找到的当前的**文件路径名**取代，命令行末尾必须有**\;** 如：
  - ✓ **find /home -user xxj05020612 -exec cat {} \;**|more: 找出/home下属于用户xxj05020612的所有文件并显示其内容
- **-user**选项常被用在要删除用户账户之前的文件清除，如：
  - ✓ **find /home -user xxj05020652 -exec rm -r {} \;** : 找出/home下属于用户xxj05020652的所有文件并删除。
- **-atime n**查找前**n**天**访问过**的文件（仅**第n天这一天**）



# find命令

## ❖ 常用的匹配条件

- **-atime +n**查找前n天之前访问过的文件；**-n**前n天之后。

例如：

- ✓ **find \$HOME -user xxj05020612 -atime +3 -exec rm -r {}**

**|;**：找出用户xxj05020612主目录下属于自己的前3天之前访问过（3天以来没用过）的文件并删除

- ✓ **find \$HOME -user xxj05020612 -atime -3 -exec ls -l {}**

**|;**：找出用户xxj05020612主目录下属于自己的前3天以后访问过（跳过3天以来没用过）的文件并以长格式显示属性

- **-type filetype**指定查找的文件类型。filetype可以是：

- ✓ b块特殊文件；c字符特殊文件；d目录；f常规文件。如：

- **find . -type d -print**#查找当前目录下的所有目录
- **find . -type f -print** #查找当前目录下的所有普通文件



# find命令

## ❖ 常用的匹配条件

- **-size Number**和**-size Numberc**指定查找文件的大小。  
**Numberc**表示以**字节为单位**，否则以**块**(一般是512字节)为单位。**-Number** (或**-Numberc**) 表示查找比该值小的文件，否则查找比该值大的文件。如
  - ✓ **find . -size -10 -print**: 查找当前目录下所有长度小于10块的文件
  - ✓ **ls -l`find . -size -10c -print`**: 查找当前目录下所有长度小于10个字节的文件和实际长度
  - ✓ **find . -size +100c -size -200c -exec ls -s {} \;**: 列出当前目录100-200字节长的文件



# grep命令

## ❖ 功能

- 在文件中搜索匹配的字符并进行输出

## ❖ 格式

- `grep[参数] <要找的字符串> <要寻找字符串的源文件>`

## ❖ 参数

- `-v`: 列出不匹配串或正则表达式的行
- `-c`: 对匹配的行计数
- `-l`: 只显示包含匹配的文件的文件名
- `-h`: 抑制包含匹配文件的文件名的显示
- `-n`: 每个匹配行只按照相对的行号显示
- `-i`: 产生不区分大小写的匹配，缺省状态是区分大小写

## ❖ 举例

- `grep /usr /etc/passwd`

✓ 将在/usr下建立home目录的帐户显示出来





# touch命令

## ❖ 功能

- 生成一个空文件或修改文件的存取/修改的时间记录值

## ❖ 格式

- `touch [选项] MMDDhhmmYY 文件列表`

## ❖ 参数说明

- `-a`: 只更改访问时间
- `-c`: 若目标文件不存在，不建立空的目标文件
- 使用不带参数的touch命令将文件的时间修改为当前时间

## ❖ 举例

- `[pp@linux pp]$ touch *`
  - ✓ 将当前下的文件时间修改为系统的当前时间
- `[pp@linux pp]$ touch -d 20040210 test`
  - ✓ 将test文件的日期改为20040210
- `[pp@linux pp]$ touch abc`
  - ✓ 若abc文件存在，则修改为系统的当前时间；若不存在，则生成一个为当前时间的空文件



# wc命令

## ❖ 功能

- 统计指定文件中的字节数、字数、行数，并将统计结果显示输出

## ❖ 语法格式

- `wc [选项] 文件列表`

## ❖ 参数

- `c`: 统计字节数
- `-l`: 统计行数
- `-w`: 统计字数

## ❖ 说明

- 如果没有给出文件名，则从标准输入读取
- `wc`同时也给出所有指定文件的总统计数
- 字是由空格字符区分开的最大字符串

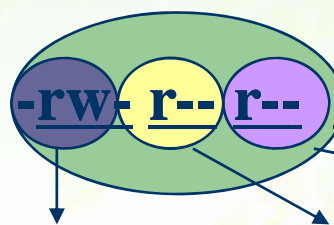


# 文件的保护方式

- ❖ 两种用户：超级用户、普通用户
- ❖ 三个等级：文件拥有者(**u**ser)、同组用户(**g**roup)、其他用户(**o**thers)，所有用户(**a**ll)
- ❖ 三种权限：读(**r**)、写(**w**)、执行(**x**)

ls -l 长格式显示目录内容:

权限 链接数 文件主 组 长度 建立/修改时间 目录/  
文件名

 1 jjpr jjpr

所有者 同组用户 其他组用户

1299 Nov 200 9:33 hs



# umask命令

## ❖ 功能

- 用户创建文件或目录时屏蔽某些权限
- 注意，x权不会自动添加

## ❖ 实质

- 二进制数，对应的屏蔽权限bit=1，常写成八进制(对目录来说)。例如，屏蔽GW和OW时为022，有如下对应关系

权限	UR	UW	UX	GR	GW	GX	OR	OW	OX
bit	0	0	0	0	1	0	0	1	0
八进制	0			2			2		





## umask命令

### ❖ 命令格式: **umask [-S] [代码]**

- S表示用符号方式显示
- 代码为屏蔽码数字或 **许可权限**的符号方式
- 如, 以下两个命令等效
  - ✓ **umask u=rwx,g=rx,o=**  $\Leftrightarrow$  **umask 027**



# chown命令

## ❖ 功能

- 改变文件或目录的拥有者或所属组

## ❖ 语法

- **chown** [-R] 用户名[:组名] 文件或目录名

## ❖ 参数

- **-R** : 对当前目录下的所有文件与子目录进行相同的拥有者变更
- 只有root才有此权限



# chgrp命令

## ❖ 功能

- 改变文件或目录的所属组

## ❖ 语法

- `chgrp [-R] 新文件属组 文件或目录名`

## ❖ 参数

- **-R** : 对当前目录下的所有文件与子目录进行相同的权限变更(即以递归的方式逐个变更)
- 只有属主和root才有此权限
- 在多数系统中, 要求属主必须也属于新组的成员



# chmod命令

- ❖ 功能: 改变文件或目录的存取权限
- ❖ 语法: **chmod [-R] 模式** 文件或目录名
- ❖ 参数:
  - **-R**: 对目前目录下的所有文件与子目录进行相同的权限变更(即以递归的方式逐个变更)

## ❖ 模式

字符表示方式:

谁 (用户)

操作符

许可权

u 文件主

+ 增加权限

r 读

g 同组人

- 取消权限

w 写

o 其他人

= 重新设置

x 执行

a 所有人 (缺省值) (同时删除旧的权限)

数字表示方式:

r w x

r w x

r - x

r - -

4 2 1

4 2 1

4 0 1

4 0 0

7

5

4





## chmod命令

### ❖ 示例1

- [jjpr@zzh jjpr]\$ chmod 664 hs  
✓ #将文件hs的权限设为rw-rw-r—
- [jjpr@zzh jjpr]\$ ls -l hs  
✓ -rw-rw-r-- 1 jjpr jjpr 1299 Nov 20 09:33 hs
- [jjpr@zzh jjpr]\$

### ❖ 示例2

- [jjpr@zzh jjpr]\$ chmod g-w hs  
✓ #取消同组用户对hs的写权
- [jjpr@zzh jjpr]\$ ls -l hs  
✓ -rw-r--r-- 1 jjpr jjpr 1299 Nov 20 09:33 hs
- [jjpr@zzh jjpr]\$



# tar命令

## ❖ 功能

- 文件归档

## ❖ 语法

- **tar** [参数] 文件或目录名

## ❖ 参数

- **-c** : 建立一个.tar文件
- **-v** : 列出处理过程中的详细信息
- **-f** : 指定新文件名
- **-x** : 解压某个文件
- **-u** : 仅仅添加比文档文件更新的文件, 如原文档中不存在旧的文件, 则追加它到文档中, 如存在则更新它
- **-z** : 用zip命令压缩或用unzip解压



## tar命令

- ❖ 归档时，根据一系列文件名称编制档案，依次读取文件、写入文件标题，然后写入文件内容
- ❖ 如果保存时使用目录的绝对路径，数据只能恢复到原来目录上。例如
  - 若unix和APP均为目录，采用如下命令生成档案文件
    - ✓ `tar cvf $HOME/xxj.tar /home/unix ./APP`
    - ✓ 该命令在home目录下产生档案文件(包) xxj.tar，子目录APP中的文件可以恢复到任意目录的APP子目录下
    - ✓ 而/home/unix中的文件只能恢复到/home/unix目录下
    - ✓ 如果不存在档案目录，tar命令将自动创建



# tar命令

## ❖ 示例

- `$ tar -cvf back.tar backup`
  - ✓ 备份backup目录下的所有文件和目录
- `$ tar -xvf backup.tar`
  - ✓ 将备份backup.tar文件还原
- `$ tar -czvf backup.tar.gz backup`
  - ✓ 备份backup目录下的所有文件和目录，并以zip压缩，命名文件为backup.tar.gz
- `$ tar -xzvf backup.tar.gz`
  - ✓ 将备份backup.tar.z文件还原
- `$ tar -tf backup.tar`
  - ✓ 列出备份文件backup.tar的内容
- `$ tar -tzf backup.tar.gz`
  - ✓ 列出备份文件backup.tar.gz的内容
- `$ tar -rf backup.tar file1`
  - ✓ 在备份backup.tar的尾部添加文件 file1





## gzip和gunzip

- ❖除了.zip文件的压缩格式外，在Linux系统下更常见的是.gz文件的压缩格式，这种文件一般是由gzip命令所产生
- ❖zip命令具有将许多文件压缩成一个文件的功能，但gzip却不能，所以gzip一般会与tar一起使用
- ❖目前，大部分或见到的压缩文件都是用tar将所有文件打包成一个文件，再用gzip进行压缩，所以我们所看到的扩展名为.tar.gz或.tgz的文件，大多数就是这种类型的文件



# gzip和gunzip

## ❖ 使用方法

- `[root@linux test]# gzip test.txt`
  - ✓ 压缩文件时，不需要任何参数
- `[root@linux test]# gzip -l test.txt.gz`
  - ✓ 显示压缩率
- `[root@linux test]# gunzip test.txt.gz`
  - ✓ 解压缩



# 常见的压缩工具与解压缩工具

压缩工具	解压缩工具	后缀	例
<b>compress</b>	<b>uncompress</b>	<b>.Z</b>	<b>rfell8.text.Z</b>
<b>gzip</b>	<b>gunzip</b>	<b>.gz</b>	<b>textfile.gz</b>
<b>zip</b>	<b>unzip</b>	<b>.zip</b>	<b>package.zip</b>



# 主要内容

## ❖ 背景知识

- Linux简介
- Linux系统环境
- Linux用户管理命令
- Linux文件操作命令
- **Linux网络管理命令**
- Linux系统信息命令
- Linux编程环境

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核





# 网络配置方法

## ❖ X windows环境

- 控制面板 → 网络

## ❖ 字符界面

- 主机名的显示或设置

- ✓ 命令: `hostname`

- ✓ 配置文件: `/etc/hosts/etc/sysconfig/network`

- ✓ 说明: `hostname`命令未修改配置文件

- IP地址、掩码的显示或设置

- ✓ `ifconfig`命令

- ✓ `route add default gw` 网关IP

- ✓ `netconfig(或setup)`命令

- ✓ 执行`service network restart`命令使网络设置生效



# ifconfig命令

## ❖ 功能

- 配置或显示网卡信息

## ❖ 说明

- 只有root可以配置网卡参数，普通用户只能显示网卡信息
- 从 ifconfig 中得到IP地址
  - ✓ 使用 **ifconfig -a** 可以得到所有网络界面的IP地址。
- 下面的脚本可以直接输出 IP 地址：  
`$ /sbin/ifconfig -a|awk '/inet/{print $2}'`



# ping命令

## ❖ 功能

- 检查网络连通性

## ❖ 语法

- `ping <ip地址>`

## ❖ 举例

- **PING 202.207.125.54**

✓ (202.207.125.54) 56(84) bytes of data.

✓ 64 bytes from 202.207.125.54: icmp\_seq=0 ttl=64  
time=0.331 ms

✓ 64 bytes from 202.207.125.54: icmp\_seq=1 ttl=64  
time=0.124 ms



# netstat命令

## ❖ 功能

- 检查网络状态

## ❖ 语法

- **netstat** [参数选项]

## ❖ 参数

- **-I** 自动配置的接口状态。

## ❖ 示例

- **netstat -I**: 显示网络接口的数据包传输统计，如下

Name	Mtu	Network	Address	Ipkts	Ierrs	Opkts	Oerrs	Coll
net1	1500	202.207.124	jrpr	158548	0	100619	0	485
lo0	8232	loopback	loopback	1066	0	1066	0	0





# write、wall、mesg命令

**write username**向指定用户发送消息，每行(回车)发送一次，直到<sup>^</sup>D结束。

例

```
write xxj05020612
Hell?           #发送
How are you?    #发送
^D             #发送
EOF
```

**wall**向所有用户发送消息，直到<sup>^</sup>D结束并发送。

例

```
wall
Hell?
How are you?
^D           #发送
```

**mesg** 显示当前终端是否允许别人发来消息  
**mesg n (y)** 禁止（允许）别人发来消息



# FTP主要功能

## ❖ get命令

- 格式: ftp>get 源文件 目标文件
  - ✓ 源文件: 远程计算机 **当前目录**下的文件名
  - ✓ 目标文件: 本地计算机 **当前目录**下要创建的文件名。缺省为源文件名
- 例: ftp>get aa.asm bb.asm

## ❖ put命令

- 格式: ftp>put 源文件 目标文件
- 功能: 将本地计算机 **当前目录**下的指定文件传送到远程计算机的 **当前目录**下。目标文件缺省为源文件名

## ❖ dir命令

- 格式: ftp>dir [<目录名> <本地文件名>]
  - ✓ 目录名: 远程计算机上, 缺省为当前目录文件名, 本地计算机当前目录下, 把列出的内容输出到该文件。省略时显示在终端上



# FTP主要功能

## ❖ ascii和binary命令

### ➤ Ascii

- ✓ 设定FTP的传送状态，直到遇到**binary**命令
- ✓ 传送ASCII码文本文件，并自动调整文件的格式
- ✓ 不能传送程序、压缩文件等含有非ASCII码的文件

### ➤ binary

- ✓ 设定FTP的传送状态，直到遇到ascii命令
- ✓ 可传送任意文件
- ✓ 不对文件格式进行调整

## ❖ help、? 命令

- **help <命令>**: 显示关于此命令的一段帮助文字;
- **Help**: 显示全部命令
- **help**与**?** 等效



## 匿名FTP

- ❖ 以**anonymous**或**ftp**作为用户登录，接受任何字符串为口令，一般要求用电子邮件地址作为口令
- ❖ 对大文件的处理一般**FTP**服务器的大文件都先压缩后供用户索取，索取后的压缩文件必须经过**对应的解压缩**才能使用对于不同的系统，一般压缩与解压缩工具不同
- ❖ 批文件处理主要完成一批相关文件、目录、目录树的处理





## 远程登录（telnet）

### ❖ 基本连接

- 方法1: #telnet <主机名>
- 例: 在UNIX系统下
- #telnet unix.sjzpc.edu.cn （连接到unix.sjzpc.edu.cn）

login:student

Password:\_\_\_\_\_

%

- 方法2: #telnet <IP地址>
- 例: #telnet 202.207.120.254

- 方法3:

telnet>open <主机名> 或telnet>open <IP地址>

### ❖ 退出

- ^D或telnet>quit



# 主要内容

## ❖ 背景知识

- Linux简介
- Linux系统环境
- Linux用户管理命令
- Linux文件操作命令
- Linux网络管理命令
- **Linux系统信息命令**
- Linux编程环境

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核



## date、cal、clock命令

### ❖ 功能

- 显示/修改当前的日期时间

### ❖ 示例

- [root@linux root]# date 121010232004

✓ 将时间更改为12月10日10点23分2004年  
时间显示格式: [MMDDhhmmYY]

- [root@linux root]# cal

✓ 显示日历

- [root@linux root]# clock

✓ 显示日期时间

```
[root@linux root]# date
Wed Mar  9 17:35:55 CST 2005
[root@linux root]# cal
      March 2005
Su Mo Tu We Th Fr Sa
        1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

[root@linux root]# clock
Thu 10 Mar 2005 04:38:59 PM CST  0.832619 seconds
[root@linux root]# _
```



# Cal命令

## ❖ 功能

- 显示一个日历

## ❖ 格式

- `cal [参数] 月 年`

## ❖ 示例

- `[root@linux root]# cal` —→ 显示当月的日历

- `[root@linux root]# cal 4 2004`

显示2004年4月的日历

- `[root@linux root]# cal - y 2003`

显示2003年的日历





# dmesg命令

## ❖ 功能

- 显示系统诊断信息、操作系统版本号、物理内存的大小以及其它信息

```
[root@host ~]# dmesg | more
Linux version 2.4.18-14 (bhcompile@stripples.devel.redhat.com) (gcc version 3.2
20020903 (Red Hat Linux 8.0 3.2-7)) #1 Wed Sep 4 13:35:50 EDT 2002
BIOS-provided physical RAM map:
 BIOS-e820: 0000000000000000 - 0000000000009f800 (usable)
 BIOS-e820: 0000000000009f800 - 000000000000a0000 (reserved)
 BIOS-e820: 000000000000ca000 - 000000000000cc000 (reserved)
 BIOS-e820: 000000000000dc000 - 000000000000e0000 (reserved)
 BIOS-e820: 000000000000e4000 - 00000000000100000 (reserved)
 BIOS-e820: 00000000000100000 - 000000000007ef0000 (usable)
 BIOS-e820: 000000000007ef0000 - 000000000007eff000 (ACPI data)
 BIOS-e820: 000000000007eff000 - 000000000007f00000 (ACPI NUS)
 BIOS-e820: 000000000007f00000 - 000000000008000000 (usable)
 BIOS-e820: 00000000000fec0000 - 00000000000fec10000 (reserved)
 BIOS-e820: 00000000000fee0000 - 00000000000fee01000 (reserved)
 BIOS-e820: 00000000000ffe0000 - 00000000000100000000 (reserved)
0MB HIGHMEM available.
128MB LOWMEM available.
On node 0 totalpages: 32768
zone(0): 4096 pages.
zone(1): 28672 pages.
zone(2): 0 pages.
Kernel command line: auto BOOT_IMAGE=linux ro BOOT_FILE=/boot/vmlinuz-2.4.18-14
root=LABEL=
```



# df命令

## ❖ 功能

- 用于查看文件系统的各个分区的占用情况

```
[root@linux root]# df
Filesystem      1k-blocks      Used Available Use% Mounted on
/dev/sda1        806368      156924     608480   21% /
/dev/sda3        600864       16468     553876    3% /home
none             30736         0        30736    0% /dev/shm
/dev/sda2       2419288     1713288     583104   75% /usr
[root@linux root]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       787M  154M  594M   21% /
/dev/sda3       587M   17M  540M    3% /home
none            30M    0    30M    0% /dev/shm
/dev/sda2       2.3G  1.7G  569M   75% /usr
[root@linux root]# _
```



# du命令

## ❖ 功能

- 查看某个目录中各级子目录所使用的硬盘空间数

## ❖ 格式

- **du [参数] <目录名>**

```
[root@linux root]# du
4      ../kde/share/config
4      ../kde/share/servicetypes
4      ../kde/share/mimelnk
4      ../kde/share/applnk
4      ../kde/share/services
24     ../kde/share
152    ../kde/tmp-localhost.localdomain
180    ../kde
252    .
[root@linux root]# du -h
4.0k   ../kde/share/config
4.0k   ../kde/share/servicetypes
4.0k   ../kde/share/mimelnk
4.0k   ../kde/share/applnk
4.0k   ../kde/share/services
24k    ../kde/share
152k   ../kde/tmp-localhost.localdomain
180k   ../kde
252k   .
[root@linux root]# _
```



# free命令

## ❖ 功能

- 用于查看系统内存，虚拟内存（交换空间）的大小占用情况

```
[root@linux root]# free
              total        used         free       shared    buffers     cached
Mem:          61476       56264         5212           0        2528      29476
-/+ buffers/cache:      24260       37216
Swap:        128480         288      128192
[root@linux root]# free -m
              total        used         free       shared    buffers     cached
Mem:           60          54           5           0           2          28
-/+ buffers/cache:         23          36
Swap:          125           0         125
[root@linux root]# _
```





# 主要内容

## ❖ 背景知识

- Linux简介
- Linux系统环境
- Linux用户管理命令
- Linux文件操作命令
- Linux网络管理命令
- Linux系统信息命令
- **Linux编程环境**

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核



# 编程语言

## ❖ 高级变成语言

- C/C++, Java, Fortran...
- ELF binary format
  - ✓ Executable and Linkable Format
  - ✓ 工具接口标准委员会(TIS)选择了正在发展中的ELF体系上不同操作系统之间可移植的二进制文件格式

## ❖ 脚本

- Shell: sh/bash, csh, ksh
- Perl, Python, tcl/tk, sed, awk...



# 开发工具

## ❖ GCC

- GNU C Compiler -> GNU Compiler Collection
- The gcc command: Front end

## ❖ GDB

- GNU Debugger
- The gdb command
- xxdgb, ddd...

## ❖ Binary utilities

- as, ld, ar, ldd...

## ❖ Make



# 集成环境

## ❖ IDE

- Emacs/xemacs
- Kdevelop
- Eclipse
- Kylix3

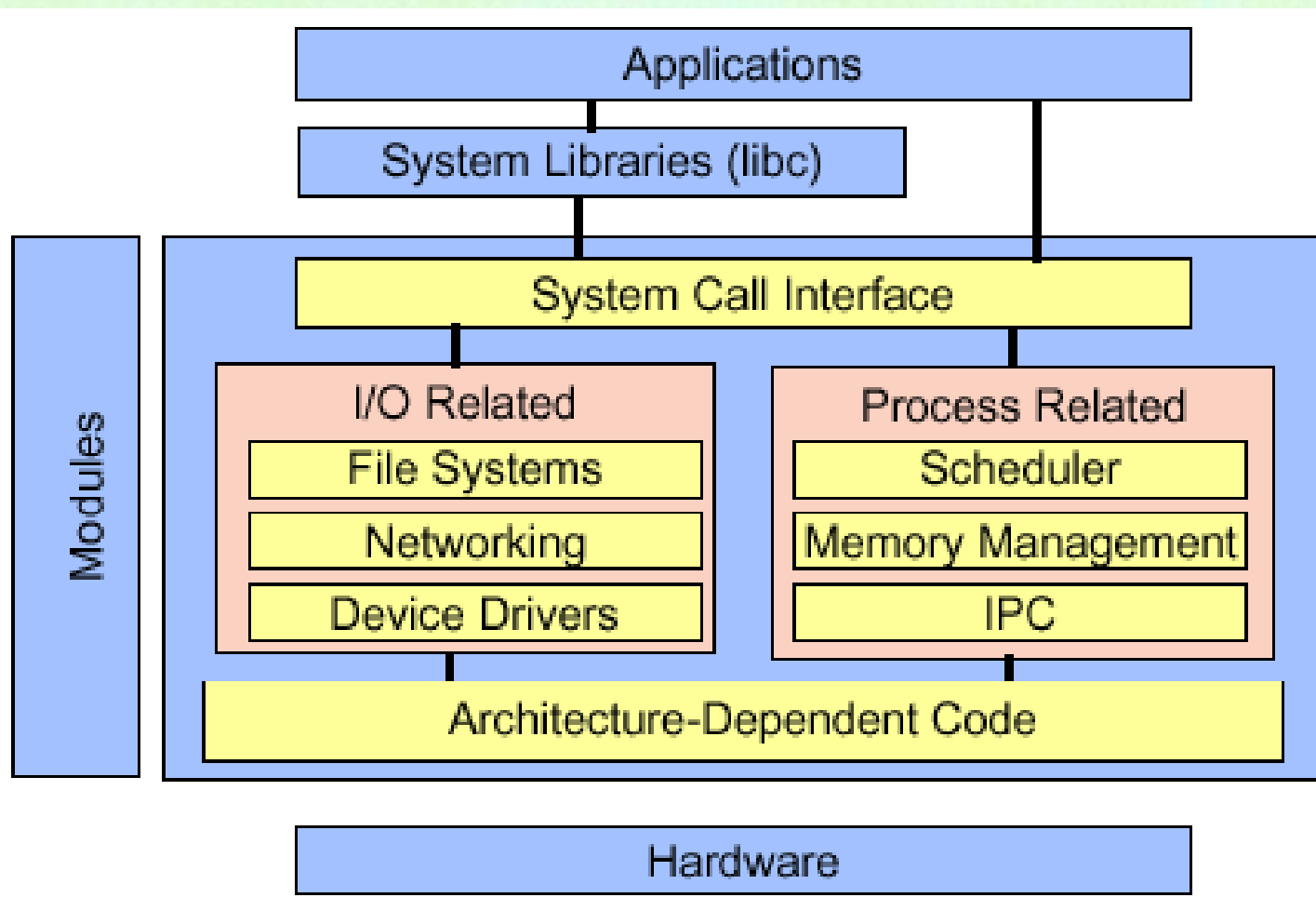
## ❖ Command line

- Editor: vi/vim/gvim, emacs/xemacs, pico
- Source Reader: source navigator; vi/emacs+ ctags/etags
- Configure Tools: automake, autoconf, m4





# C程序员的系统视图





# 系统调用与函数库

## ❖ 系统调用

- Linux内核的对外接口；用户程序和内核之间唯一的接口

## ❖ 函数库

- 依赖于系统调用
- 一般来说，标准函数库建立在系统调用的上层，提供的功能比系统调用强，使用也比较方便
- 例：标准I/O库
- Static Libraries (.a files)
  - ✓ Lab (gcc + ar)
- Dynamic Libraries/Shared Objects (.so files)
  - ✓ Lab (gcc)



## 文件名后缀

.c	C source code which must be preprocessed
.i	C source code which should not be preprocessed
.cc .cp .cpp .CPP. c++ .C .cxx	C++ source code which must be preprocessed
.ii	C++ source code which should not be preprocessed
.h	C or C++ header file to be turned into a precompiled header
.H .hh	C++ header file to be turned into a precompiled header
.s	Assembler code
.S	Assembler code which must be preprocessed
.o	Object file
.a	Static library file (archive file)
.so	Dynamic library file (shared object)



## GNU Tools

- ❖ **GNU tools**和其他一些优秀的开源软件可以完全覆盖上述类型的软件开发工具。
- ❖ 为了更好的开发嵌入式系统，需要熟悉如下一些软件
  - **GCC**
  - **Binutils**—辅助GCC的主要软件
  - **gdb**
  - **make**
  - **cvcs**





# GCC

- ❖ 很多人认为GCC只是一个C编译器，其实GCC = GNU Compiler Collection
- ❖ 目前，GCC可以支持多种高级语言，如
  - C、C++
  - ADA
  - Object C
  - JAVA
  - Fortran
  - PASCAL



# GCC下的工具

## ❖ **cpp** — 预处理器

- GNU C编译器在编译前自动使用**cpp**对用户程序进行预处理

## ❖ **gcc**

- 符合ISO等标准的C编译器

## ❖ **g++**

- 基本符合ISO标准的C++编译器

## ❖ **gcj**

- GCC的java前端

## ❖ **gnat**

- GCC的GNU ADA 95前端



## GNU Tools—gcc

- ❖ gcc是一个强大的工具集合，它包含了预处理器、编译器、汇编器、链接器等组件。它会在需要的时候调用其他组件
- ❖ 输入文件的类型和传递给gcc的参数决定了gcc调用具体的哪些组件
- ❖ 对于开发者，它提供的足够多的参数，可以让开发者全面控制代码的生成，这对嵌入式系统级的软件开发非常重要



# gcc使用举例

## ❖ 源程序

```
//gcctest.c  
  
#include <stdio.h>  
  
int main()  
{  
    int i,j;  
    i=0;  
    j=0;  
    i=j+1;  
    printf("Hello World!\n");  
    printf("i=j+1=%d\n",i);  
}
```





## gcc使用举例

### ❖ 编译和运行

```
[donger@donger gcctest]$ ls
gcctest.c
[donger@donger gcctest]$ gcc -o gcctest gcctest.c
[donger@donger gcctest]$ ls
gcctest  gcctest.c
[donger@donger gcctest]$ ./gcctest
Hello World!
i=j+1=1
[donger@donger gcctest]$
```

编译

运行



# gcc的工作过程

## ❖ 使用-v选项，可以看到许多被隐藏的信息

```
[donger@donger gcctest]$ gcc -o gcctest gcctest.c -v
使用内建 specs。
目标: i386-redhat-linux
配置为: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-libgcj-multifile --enable-languages=c,c++,objc,java,f95,ada --enable-java-awt=gtk --with-java-home=/usr/lib/jvm/java-1.4.2-gcj-1.4.2.0/jre --host=i386-redhat-linux
线程模型: posix
gcc 版本 4.0.0 20050519 (Red Hat 4.0.0-8)
/usr/libexec/gcc/i386-redhat-linux/4.0.0/cc1 -quiet -v gcctest.c -quiet -dumpbase gcctest.c -auxbase gcctest -version -o /tmp/ccRVXfB5.s
ignoring nonexistent directory "/usr/lib/gcc/i386-redhat-linux/4.0.0/../../../../i386-redhat-linux/include"
#include "...": 搜索从这里开始:
#include <...>: 搜索从这里开始:
/usr/local/include
/usr/lib/gcc/i386-redhat-linux/4.0.0/include
/usr/include
搜索列表结束。
GNU C version 4.0.0 20050519 (Red Hat 4.0.0-8) (i386-redhat-linux)
    compiled by GNU C version 4.0.0 20050519 (Red Hat 4.0.0-8).
GGC 准则: --param ggc-min-expand=42 --param ggc-min-heapsize=23850
as -V -Qy -o /tmp/ccizRCv4.o /tmp/ccRVXfB5.s
GNU assembler version 2.15.94.0.2.2 (i386-redhat-linux) using BFD version 2.15.94.0.2.2 20041220
/usr/libexec/gcc/i386-redhat-linux/4.0.0/collect2 --eh-frame-hdr -m elf_i386 -dynamic-linker /lib/ld-linux.so.2 -o gcctest /usr/lib/gcc/i386-redhat-linux/4.0.0/../../../../crt1.o /usr/lib/gcc/i386-redhat-linux/4.0.0/../../../../crti.o /usr/lib/gcc/i386-redhat-linux/4.0.0/crtbegin.o -L/usr/lib/gcc/i386-redhat-linux/4.0.0 -L/usr/lib/gcc/i386-redhat-linux/4.0.0 -L/usr/lib/gcc/i386-redhat-linux/4.0.0/../../../../tmp/ccizRCv4.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed -lgcc_s --no-as-needed /usr/lib/gcc/i386-redhat-linux/4.0.0/crtend.o /usr/lib/gcc/i386-redhat-linux/4.0.0/../../../../crtfn.o
[donger@donger gcctest]$
```



# gcc的编译过程

## ❖ 一般情况下，C程序的编译过程为

- 预处理
- 编译成汇编代码
- 汇编成目标代码
- 链接



## 预处理

❖ 使用**-E**参数输出文件的后缀为“.cpp”

```
gcc -E -o gcctest.cpp gcctest.c
```

❖ 使用**wc**命令比较预处理后的文件与源文件，可以看到两个文件的差异



```
[donger@donger gcctest]$ gcc -E -o gcctest.cpp gcctest.c
```

```
[donger@donger gcctest]$ ls
```

预编译

```
gcctest.c  gcctest.cpp
```

```
[donger@donger gcctest]$ wc gcctest.c gcctest.cpp
```

```
15      15      132 gcctest.c
```

```
929    2102   18137 gcctest.cpp
```

行数 单词数 字节数

```
944    2117   18269 总用量
```

```
[donger@donger gcctest]$ wc --help
```

用法: wc [选项]... [文件]...

Print newline, word, and byte counts for each FILE, and a total line if

more than one FILE is specified. With no FILE, or when FILE is -,

read standard input.

-c, --bytes print the byte counts

-m, --chars print the character counts

-l, --lines print the newline counts

-L, --max-line-length print the length of the longest line

-w, --words print the word counts

--help 显示此帮助信息并离开

--version 显示版本信息并离开

Report bugs to <bug-coreutils@gnu.org>.

```
[donger@donger gcctest]$ █
```



## 编译成汇编代码

### ❖ 预处理文件→汇编代码

- 使用 **-x** 参数说明根据指定的步骤进行工作
- **cpp-output** 指明从预处理得到的文件开始编译
- 使用 **-S** 说明生成汇编代码后停止工作

```
gcc -x cpp-output -S -o gcctest.s gcctest.cpp
```

### ❖ 也可以直接编译到汇编代码

```
gcc -S gcctest.c
```



## 编译成汇编代码

```
[donger@donger gcctest]$ ls
gcctest.c  gcctest.cpp
[donger@donger gcctest]$ gcc -x cpp-output -S -o gcctest.o gcctest.cpp
[donger@donger gcctest]$ ls
gcctest.c  gcctest.cpp  gcctest.o
[donger@donger gcctest]$
```

预处理文件→汇编代码

```
[donger@donger gcctest]$ ls
gcctest.c
[donger@donger gcctest]$ gcc -S gcctest.c
[donger@donger gcctest]$ ls
gcctest.c  gcctest.s
[donger@donger gcctest]$
```

直接编译到汇编代码



## 编译成目标代码

### ❖ 汇编代码→目标代码

➤ `gcc -x assembler -c gcctest.s`

```
[donger@donger gcctest]$ ls
gcctest.c  gcctest.s
[donger@donger gcctest]$ gcc -x assembler -c gcctest.s
[donger@donger gcctest]$ ls
gcctest.c  gcctest.o  gcctest.s
[donger@donger gcctest]$ █
```

汇编代码→目标代码





# 编译成目标代码

## ❖ 直接编译成目标代码

➤ **gcc -c gcctest.c**

```
[donger@donger gcctest]$ ls
gcctest.c
[donger@donger gcctest]$ gcc -c gcctest.c
[donger@donger gcctest]$ ls
gcctest.c  gcctest.o
[donger@donger gcctest]$ █
```

直接编译成目标代码

## ❖ 使用汇编器生成目标代码

➤ **as -o gcctest.o gcctest.s**

```
[donger@donger gcctest]$ ls
gcctest.c  gcctest.s
[donger@donger gcctest]$ as -o gcctest.o gcctest.s
[donger@donger gcctest]$ ls
gcctest.c  gcctest.o  gcctest.s
[donger@donger gcctest]$ █
```

使用汇编器



## 编译成执行代码

### ❖ 目标代码→执行代码

➤ **gcc -o gcctest gcctest.o**

```
[donger@donger gcctest]$ ls
gcctest.c  gcctest.o
[donger@donger gcctest]$ gcc -o gcctest gcctest.o
[donger@donger gcctest]$ ls
gcctest gcctest.c  gcctest.o
[donger@donger gcctest]$ █
```

### ❖ 直接生成执行代码

➤ **gcc -o gcctest gcctest.c**

```
[donger@donger gcctest]$ ls
gcctest.c
[donger@donger gcctest]$ gcc -o gcctest gcctest.c
[donger@donger gcctest]$ ls
gcctest gcctest.c
[donger@donger gcctest]$ █
```



## gcc的高级选项

### ❖ -Wall: 打开所有的警告信息

```
[donger@donger gcctest]$ ls
gcctest.c
[donger@donger gcctest]$ gcc -Wall -o gcctest gcctest.c
gcctest.c: 在函数 'main' 中:
gcctest.c:13: 警告: 在有返回值的函数中, 控制流程到达函数尾
[donger@donger gcctest]$ █
```



## 根据警告信息检查源程序

```
//gcctest.c  
  
#include <stdio.h>  
  
int main() main函数的返回值为int  
{  
    int i,j;  
    i=0;  
    j=0;  
    i=j+1;  
    printf("Hello World!\n");  
    printf("i=j+1=%d\n",i);  
} 在函数的末尾应当返回一个值
```





## 修改源程序

```
//gcctest.c
```

```
#include <stdio.h>
```

```
int main()
```

```
{
```

```
    int i,j;
```

```
    i=0;
```

```
    j=0;
```

```
    i=j+1;
```

```
    printf("Hello World!\n");
```

```
    printf("i=j+1=%d\n",i);
```

```
    return 0;
```

```
}
```

```
[donger@donger gcctest]$ ls
```

```
gcctest.c
```

```
[donger@donger gcctest]$ gcc -Wall  
-o gcctest gcctest.c
```

```
[donger@donger gcctest]$ ls
```

```
gcctest  gcctest.c
```

```
[donger@donger gcctest]$ █
```



## 库文件使用举例

### ❖ 代码中使用Add和Minus函数

```
// test.c
#include <stdio.h>

int main()
{
    int a = 8;
    int b = 3;
    printf("a=%d\tb=%d\n",a,b);
    int sum = Add(a,b);
    printf("a+b=%d\n",sum);
    int cha = Minus(a,b);
    printf("a-b=%d\n",cha);
    return 0;
}
```



## 在编译时指定库文件

```
[root@donger gcctest]# ls
add.c  gcctest.c  minus.c  mytest.c  test.c
[root@donger gcctest]# gcc -o test test.c -ltest
[root@donger gcctest]# ./test
a=8      b=3
a+b=11
a-b=5
[root@donger gcctest]#
```

指明将libtest.a链接进来

运行结果



# GNU Toolchain—gdb

❖ **gdb = GNU debugger**

❖ **GNU tools中的调试器，功能强大**

- 设置断点
- 监视、修改变量
- 单步执行
- 显示/修改寄存器的值
- 堆栈查看
- 远程调试





# GDB常用命令

file	打开要调试的文件
break/tbreak	设置断点，可以是行号、函数名及地址(以*开头) <b>tbreak</b> : 设置临时断点
run	执行当前调试的程序
list	列出源代码的一部分
next	执行一条语句但不进入函数内部
step	执行一条语句，是函数则进入函数内部
display	显示表达式的值
print	临时显示表达式的值
kill	中止正在调试的程序
quit	推出gdb
shell	不退出gdb就执行shell命令
make	不退出gdb就执行make



## gdb使用举例

❖ 源代码如下

```
// bug.c

#include <stdio.h>
#include <stdlib.h>

static char buff[256];
static char* string;

int main()
{
    printf("input a string:");
    gets(string);

    printf("\n Your string is:%s\n",string);
    return 0;
}
```

编译:

gcc -o bug bug.c



## 编译并运行

```
[root@donger gcctest]# ls
add.c  bug.c      minus.c  mytest.c  test.c  ts
add.o  gcctest.c  minus.o  test      test.o
[root@donger gcctest]# gcc -o bug bug.c 编译
/tmp/ccgab33e.o(.text+0x36): In function `main':
bug.c: warning: the `gets' function is dangerous
and should not be used.
[root@donger gcctest]# ls
add.c  bug.c      minus.o  test.c
add.o  gcctest.c  mytest.c  test.o
bug    minus.c    test     ts
[root@donger gcctest]# ./bug
input a string:hello
段错误
[root@donger gcctest]#
```



## 使用gdb调试bug

```
[root@donger gcctest]# gdb bug
GNU gdb Red Hat Linux (6.3.0.0-1.21rh)
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(no debugging symbols found)
Using host libthread_db library "/lib/libthread_db.so.1".

(gdb) run 运行bug
Starting program: /home/donger/gcctest/bug
Reading symbols from shared object read from target memory...(no debugging sy
mbols found)...done.
Loaded system supplied DSO at 0xeac000
(no debugging symbols found)
(no debugging symbols found)
 输入字符串hello
Program received signal SIGSEGV, Segmentation fault.
0x002647e0 in main () from /lib/libc.so.6
(gdb) where 显示出错位置
#0 0x002647e0 in gets () from /lib/libc.so.6
#1 0x080483ea in main ()
(gdb) 能不能看到源代码呢?
```





## 使用gcc的-g参数

❖ gcc -g -o bug bug.c

❖ 重新调试

```
Program received signal SIGSEGV, Segmentation fault.  
0x002647e0 in gets () from /lib/libc.so.6  
(gdb) where  
#0 0x002647e0 in gets () from /lib/libc.so.6  
#1 0x080483ea in main () at bug.c:12  
(gdb) list 列出源代码  
12         gets(string);  
13  
14         printf("\n Your string is:%s\n",string);  
15         return 0;  
16     }  
(gdb) █
```



## 使用gcc的-g参数

```
(gdb) break 12
```

设置断点

```
Breakpoint 2 at 0x80483dc: file bug.c, line 12.
```

```
(gdb) next
```

```
Single stepping until exit from function gets,  
which has no line number information.
```

```
Program terminated with signal SIGSEGV, Segmentation fault.  
The program no longer exists.
```

```
(gdb) print string
```

```
$1 = 0x0
```

```
(gdb) █
```

? 怎么修改前面的源代码呢?



## 使用GNU make管理项目

- ❖ **GNU make**是一种代码维护工具，在使用**GNU编译器开发大型应用时**，往往要使用**make管理项目**
  - 如果不使用**make**管理项目，就必须重复使用多个复杂的命令行维护项目和生成目标代码
- ❖ **Make**通过将命令行保存到**makefile**中简化编译工作
- ❖ **Make**的**主要任务**是根据**makefile**中定义的规则和步骤，根据各个模块的更新情况，自动完成整个软件项目的维护和代码生成工作



## 使用GNU make管理项目

- ❖ **Make**可以识别出**makefile**中哪些文件已经被修改，并且在再次编译的时候只编译这些文件，从而提高编译的效率
  - **Make**会检查文件的修改和生成时间戳，如果目标文件的修改或者生成时间戳比它的任意一个依赖文件旧，则**make**就执行**makefile**文件中描述的相应命令，以便更新目的文件
  - 只更新那些需要更新的文件，而不重新处理那些并不过时的文件





# 使用GNU make管理项目

## ❖ 特点

- 适合于支持多文件构成的大中型软件项目的编译，链接，清除中间文件等管理工作
- 提供和识别多种默认规则，方便对大型软件项目的管理
- 支持对多目录的软件项目进行递归管理
- 对软件项目具有很好的可维护性和扩展性



# Makefile

❖ Makefile告诉make该做什么、怎么做

❖ Makefile主要定义

➤ 依赖关系

✓ 即有关哪些文件的最新版本是依赖于哪些别的文件产生或者组成的

➤ 需要用什麼命令来产生目标文件的最新版本

➤ 以及一些其他的功能



# Makefile的规则

## ❖ 规则内容

- 要创建的**目标**（文件）
- 创建目标（文件）所**依赖的文件列表**
- 通过依赖文件创建目标文件的**命令组**

## ❖ 规则一般形式

*target ... : prerequisites ...*

*<tab>command*

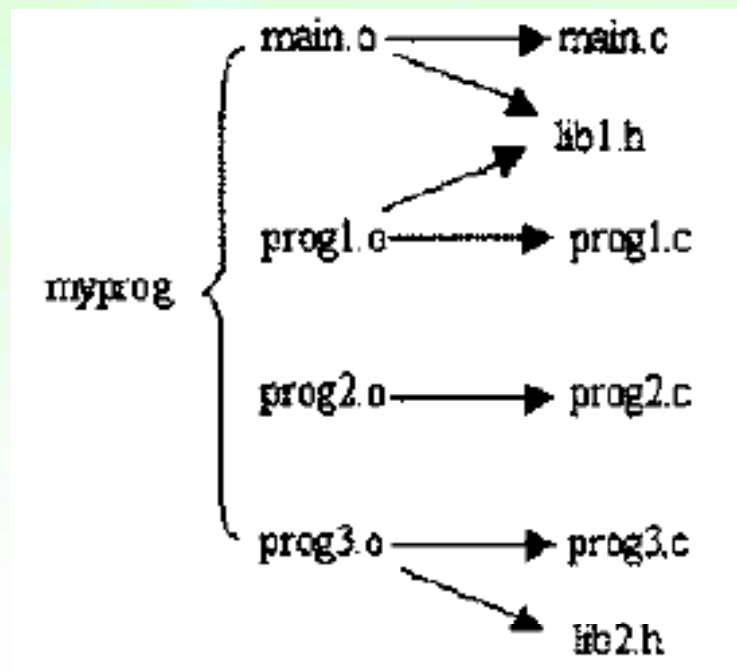
*<tab>...*

*<tab>...*

## ❖ 例如

`test:test.c`

`gcc -O -o test test.c`





# 一个简单的Makefile

edit : main.o kbd.o command.o display.o insert.o search.o files.o utils.o

cc -o edit main.o kbd.o command.o display.o insert.o \  
search.o files.o utils.o

main.o : main.c defs.h

cc -c main.c

kbd.o : kbd.c defs.h command.h

cc -c kbd.c

command.o : command.c defs.h command.h

cc -c command.c

display.o : display.c defs.h buffer.h

cc -c display.c

insert.o : insert.c defs.h buffer.h

cc -c insert.c

search.o : search.c defs.h buffer.h

cc -c search.c

files.o : files.c defs.h buffer.h command.h

cc -c files.c

utils.o : utils.c defs.h

cc -c utils.c

clean :

rm edit main.o kbd.o command.o display.o insert.o \

search.o files.o utils.o





# Make的工作过程

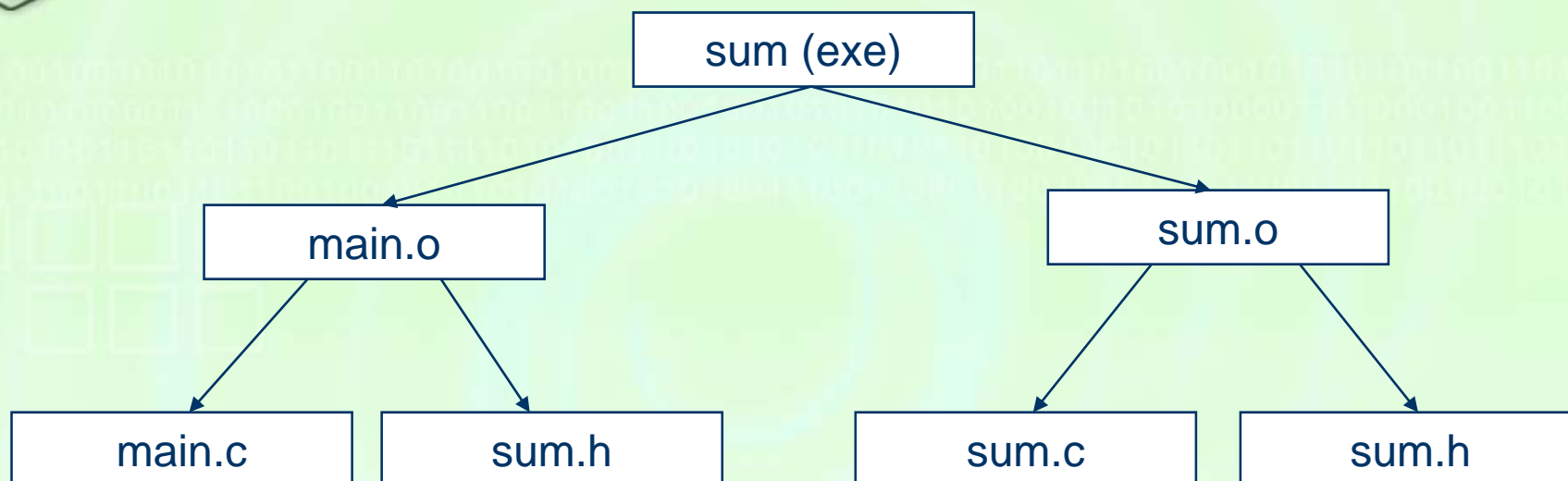
## ❖ default goal

- 在缺省的情况下，make从makefile中的第一个目标开始执行

## ❖ Make的工作过程类似一次深度优先遍历过程



# Makefile举例



**sum: main.o sum.o**

**gcc -o sum main.o sum.o**

**main.o: main.c sum.h**

**gcc -c main.c**

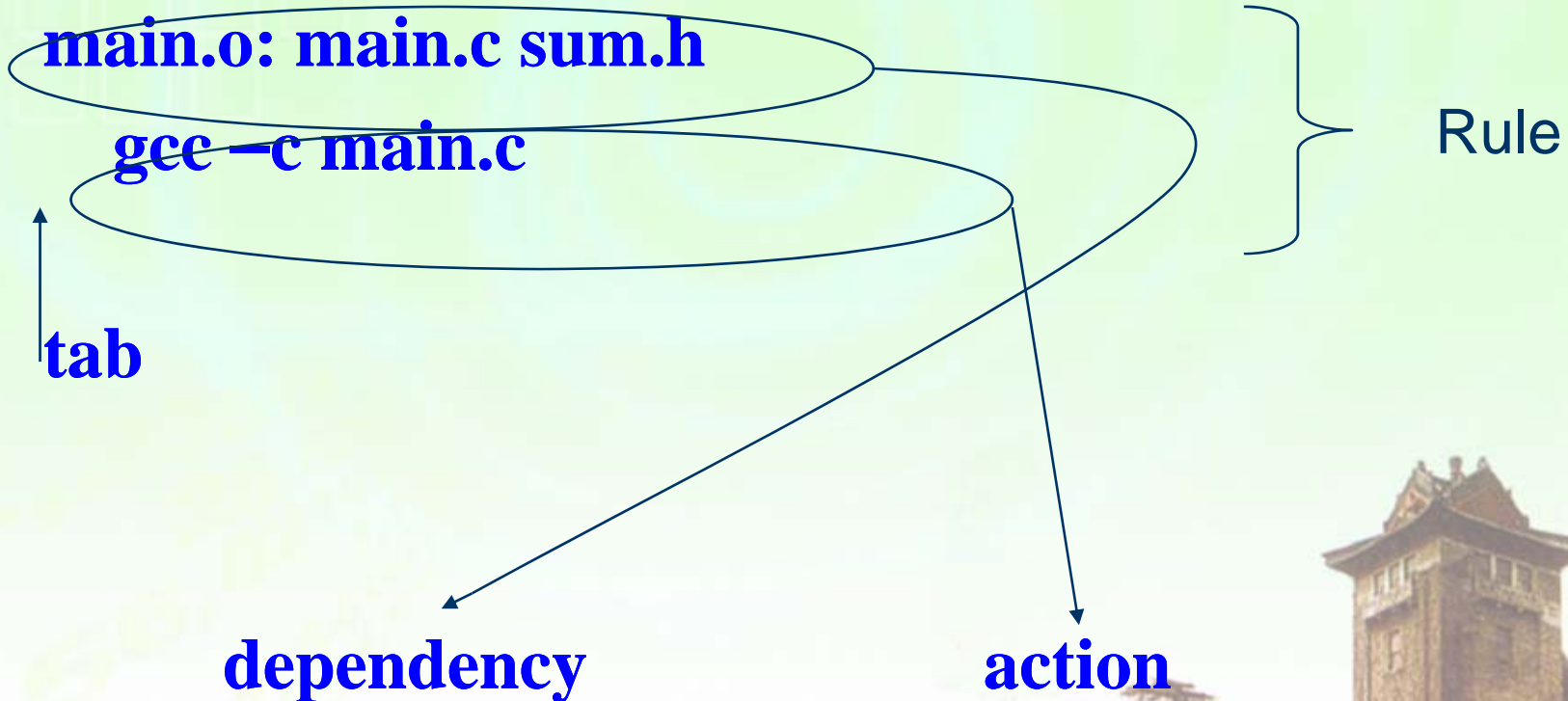
**sum.o: sum.c sum.h**

**gcc -c sum.c**



# Makefile举例

## ❖ 规则说明





## 默认依赖规则说明

### ❖ .o依赖于相应的.c文件

**sum: main.o sum.o**

**gcc -o sum main.o sum.o**

**main.o: main.c sum.h**

**gcc -c main.c**

**sum.o: sum.c sum.h**

**gcc -c sum.c**

**sum: main.o sum.o**

**gcc -o sum main.o sum.o**

**main.o: sum.h**

**gcc -c main.c**

**sum.o: sum.h**

**gcc -c sum.c**



# Makefile中的变量

## ❖ 使用变量可以

- 降低错误风险
- 简化Makefile

## ❖ 例：objects变量（\$(objects)）

```
objects = main.o kbd.o command.o \  
        display.o insert.o search.o files.o utils.o  
edit: $(objects)  
    cc -o edit $(objects)
```

## ❖ 有点像环境变量

- 环境变量在make过程中被解释成make的变量

## ❖ 可以被用来

- 贮存一个文件名列表
- 贮存可执行文件名。如用变量代替编译器名
- 贮存编译器FLAG





## 预定义变量

❖ **Make**使用了许多**预定义的变量**，如

- **AR**: 归档维护程序的名称，默认值为 **ar**
- **ARFLAGS**: 归档维护程序的选项
- **AS**: 汇编程序的名称，默认值为 **as**
- **ASFLAGS**: 汇编程序的选项
- **CC**: C 编译器的名称，默认值为 **cc**
- **CFLAGS**: C 编译器的选项
- **CXX**: C++ 编译器的名称，默认值为 **g++**
- **CPPFLAGS**: C 预编译的选项
- .....



# 简化后的Makefile文件

```
objects = main.o kbd.o command.o display.o \  
insert.o search.o files.o utils.o
```

```
edit : $(objects)  
    cc -o edit $(objects)
```

```
main.o : defs.h  
kbd.o : defs.h command.h  
command.o : defs.h command.h  
display.o : defs.h buffer.h  
insert.o : defs.h buffer.h  
search.o : defs.h buffer.h  
files.o : defs.h buffer.h command.h  
utils.o : defs.h
```

```
.PHONY : clean  
clean :
```

```
    rm edit $(objects)
```



## 内部变量

- ❖ **\$@**: 目标文件的完整名称
- ❖ **\$\***: 不包含扩展名的目标文件名称
- ❖ **\$<**: 依赖列表中的**第一个依赖文件**
- ❖ **\$+**: 所有的依赖文件，以空格分开，并以出现的先后为序，可能包含重复的依赖文件
- ❖ **\$^**: **整个依赖列表**（除去所有重复的文件名）
- ❖ **\$?**: 所有的依赖文件，以空格分开，这些依赖文件的修改日期比目标的创建日期晚
- ❖ **\$%**: 如果目标是归档成员，则该变量表示目标的归档成员名称
  - 例如，如果目标名称为 **mytarget.so(image.o)**，则 **\$@** 为 **mytarget.so**，而 **\$%** 为 **image.o**。不需要括号括住
- ❖ 例如:

```
CC = gcc
CFLAGS = -Wall -O -g
foo.o : foo.c foo.h bar.h
$(CC) $(CFLAGS) -c $< -o $@
```



## 隐含规则 (Implicit Rules)

- ❖ 内置的规则
- ❖ 告诉**make**当没有给出某些命令的时候如何处理
- ❖ 用户可以使用预定义的变量改变隐含规则的工作方式，如
  - 一个C编译的具体命令将会是
    - ✓ `$(CC) $(CFLAGS) $(CPPFLAGS) $(TARGET_ARCH) -c $< -o $@`



## 设定目标（Phony Targets）

### ❖ 设定目标

- 目标不是一个文件
- 其目的是为了能让一些命令得以执行

### ❖ 使用**PHONY**显式声明设定目标

- **.PHONY: clean**

### ❖ 使用设定目标实现多个目的

- **all: prog1 prog2**





## 典型的设定目标

❖ 设定目标也可以用来描述一些其他的动作。例如，想把中间文件和可执行文件删除，可以在 **makefile** 里设立这样一个规则：

**clean:**

**\$rm \*.o exec\_file**

- 前提是没有其它的规则依靠这个 ‘clean’ 目的，它将永远不会被执行。
- 但是，如果明确的使用命令 ‘make clean’，make 会把这个目的做为它的主要目标，执行那些 rm 命令



## Makefile中的函数 (Functions)

❖ 用来计算出要操作的文件、目标或者要执行的命令

❖ 使用方法

➢ **\$(function arguments)**

❖ 典型的函数

➢ **\$(subst from, to, text)** : 替换字符串

✓ **\$(subst ee, EE, feet on the street)**

✓ 相当于 **`fEEt on the strEEt'**

➢ **\$(patsubst pattern, replacement, text)**

✓ **\$(patsubst %.c, %.o, x.c.c bar.c)**

✓ 相当于 **`x.c.o bar.o'**

➢ **\$(wildcard pattern)**

✓ **\$(wildcard \*.c)**

✓ **objects := \$(wildcard \*.o)**



# Makefile中的条件语句

**conditional-directive**

**text-if-true**

**endif**

**or**

**conditional-directive**

**text-if-true**

**else**

**text-if-false**

**endif**



## 四种条件语句

- ❖ `ifeq...else...endif`
- ❖ `ifneq...else...endif`
- ❖ `ifdef...else...endif`
- ❖ `ifndef...else...endif`



## 带条件的makefile举例

```
sum: main.o sum.o
    gcc -o sum main.o sum.o
main.o: main.c sum.h
    gcc -c main.c
#deciding which file to compile to create sum.o
ifeq ($(USE_SUM), 1)
sum.o: sum1.c sum.h
    gcc -c sum1.c -o $@
else
sum.o: sum2.c sum.h
    gcc -c sum2.c -o $@
endif
```





# 作业：分析makefile

**# Makefile to compare sorting routines**

**BASE = /home/blufox/base**

**CC = gcc**

**CFLAGS = -O -Wall**

**EFILE = \$(BASE)/bin/compare\_sorts**

**INCLS = -I\$(LOC)/include**

**LIBS = \$(LOC)/lib/g\_lib.a \  
\$(LOC)/lib/h\_lib.a**

**LOC = /usr/local**

**OBJS = main.o another\_qsort.o chk\_order.o \  
compare.o quicksort.o**

**\$(EFILE): \$(OBJS)**

**@echo "linking ..."**

**@\$(CC) \$(CFLAGS) -o \$@ \$(OBJS) \$(LIBS)**

**\$(OBJS): compare\_sorts.h**

**\$(CC) \$(CFLAGS) \$(INCLS) -c \$\*.c**

**# Clean intermediate files**

**clean:**

**rm \*~ \$(OBJS)**



# GNU Package的典型安装

## ❖ FAQ 0039 @ smth

- 下载源代码包foo-1.0.tar.gz
- tar xvzf foo-1.0.tar.gz
- cd foo-1.0
- ./configure
- make
- (su) make install

问题1: 配置脚本configure  
是怎么生成的?

问题2: configure脚本怎么知  
道该如何生成Makefile

幕后英雄——  
GNU Auto Tools:  
autoconf, automake,  
libtool, autoscan,  
autoheader.....



## 编译与安装集成工具？

- ❖ **GNU Auto Tools**是上个世纪**90**年代开始发展起来的一系列辅助开发、打安装包的自动化工具
- ❖ 各种工具分别开发，但是协同工作的很好。比如 **autoconf, automake, libtool**等等



# Hello World创建步骤

- ❖ 1. hello.c

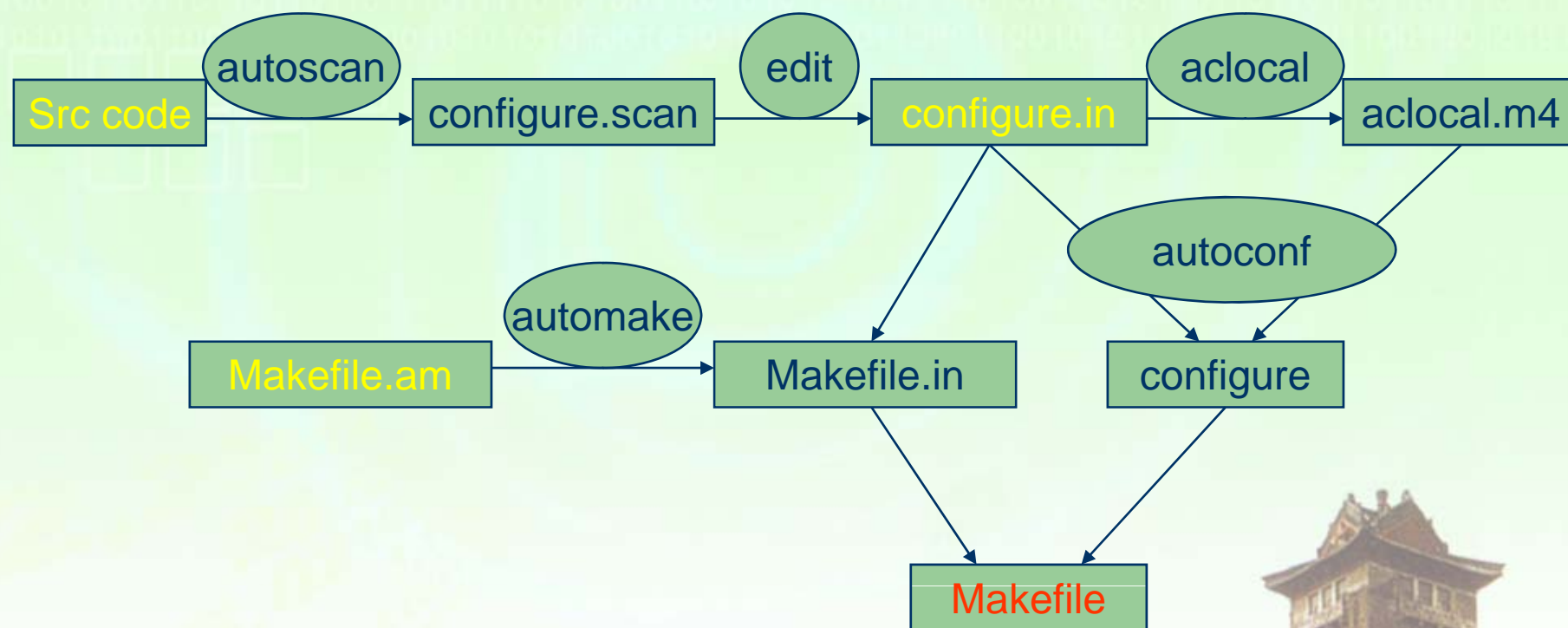
```
#include <stdio.h>
int main() { printf("Hello World!\n"); return 0; }
```
- ❖ 2. Makefile.am

```
bin_PROGRAMS = hello
hello_SOURCES = hello.c
```
- ❖ 3. configure.in

```
AC_INIT(hello.c)
AM_INIT_AUTOMAKE(hello, 0.1)
AC_PROG_CC
AC_OUTPUT(Makefile)
```
- ❖ 4. aclocal: 生成aclocal.m4文件
- ❖ 5. autoconf: 生成configure文件
- ❖ 6. automake --add-missing --foreign
- ❖ 7. ./configure
- ❖ 8. make (make install 安装, make dist 产生安装包)



# 处理过程







## configure.in

### ❖ 是configure脚本的输入文件

➤ 解决在不同unix变种之间移植程序的问题

✓ 库名可能不同

✓ 应用程序名可能不同

✓ 结构和常量的定义可能不同.....

❖ configure脚本完成autoconf与automake的初始化工作，为不同的平台定义相应的宏，检测并指定适当的程序名、库名、结构和常量名等等，指定要为哪些目录输出Makefile文件



# configure.in

## ❖ configure.in 内容

- 基本初始化部分：包括AC\_INIT (必须第一个出现), AM\_INIT\_AUTOMAKE(程序包名, 版本号), AC\_CONFIG\_HEADER
- 可选宏：如AC\_ARG\_ENABLE
- 检测某些程序的存在性
- 检查程序用到的库
- 检查某些头文件是否存在
- 检查Typedefs and structures
- 检查Functions
- 指定在哪些目录输出Makefile



# Makefile.am

## ❖ 一种比Makefile更高层次的规则

- 指定要生成什么目标，它由什么源文件生成，要安装到什么目录

## ❖ 可执行文件

`bin_PROGRAMS = foo`

`foo_SOURCES = foo1.c foo1.h foo2.c`

`foo_LDADD = foo3.o -lm foo4.a`

`foo_LDFLAGS = -L<lib_path>`

`foo_DEPENDENCIES =`



# Makefile.am

## ❖ 对静态库

**lib\_LIBRARIES = libfoo.a**

**foo\_a\_SOURCES =**

**foo\_a\_LDADD =**

**foo\_a\_LIBADD =**

**foo\_a\_LDFLAGS =**

只在make时做静态连接用，不安装的库：

**noinst\_LIBRARIES = libfoo.a**

## ❖ 对头文件

➤ **include\_HEADERS = foo.h**

## ❖ 对数据文件

➤ **data\_DATA = data1 data2**





# Makefile.am

## ❖ 全局变量(对所有目标都适用)

**INCLUDES** = -I/dir1 -I/dir2

**LDFLAGS** = -L/dir1 -L/dir2

**LDADD** = foo.o foo.a -lfoo

**EXTRA\_DIST** = file1 file2 源程序和一些默认的文件自动打入.tar.gz包，其它文件若要进入.tar.gz包可以用这种方法，比如配置文件，数据文件等等

**SUBDIRS** = dir1 dir2 在处理本目录之前要递归处理哪些子目录





# Makefile.am

## ❖ 标准安装路径

- `$(prefix) = /usr/local` 是所有安装目录的默认前缀
- 可以通过 `./configure --prefix=<new_prefix>` 的方法覆盖
- 其它的预定义目录如: `bindir = $(prefix)/bin`, `libdir = $(prefix)/lib`, `datadir = $(prefix)/share`, `sysconfdir = $(prefix)/etc`, ...

## ❖ 定义一个新的安装路径? 比如 `config`

- 可定义 `confdir = $(prefix)/config`,
- 然后 `conf_DATA = file1 file2`, 则 `file1`, `file2` 会作为数据文件安装到 `$(prefix)/config` 目录下



## Makefile.am

❖ 尽量用相对路径引用源程序的位置，以下两个变量是预定义好的

- `$(top_srcdir)`无论在哪个目录层次，该变量定义了包含src目录的目录位置，用于引用源程序
- `$(top_builddir)`定义了生成目标文件上最上层目录，用于引用.o等编译出来的目标文件
- .....



## 由Makefile.am生成Makefile

❖ Automake 调用Makefile.am（相对简单）生成Makefile（相对复杂）

❖ 说明

- Makefile.am is 85 lines (大多数样板文件)
- Makefile (generated) is 672 lines



## configure作用

❖ **configure**脚本生成的**Makefile**中已经带了很多常用的目标如: **check, all, install, uninstall, clean, dist, distcheck, distclean, tags, maintainerclean**

- If configure or make did it, make **distclean** undoes it
- If make did it, make **clean** undoes it
- If make **install** did it, make **uninstall** undoes it
- If you did it, make **maintainer-clean** undoes it





# Autoconf、Automake功能

## ❖ Autoconf

- 根据用户提供的configure.in文件，生成一个名为configure的脚本
- 该脚本可搜集有关移植性的平台相关信息，这些信息被用来生成Makefiles，配置头文件和其它平台相关的文件

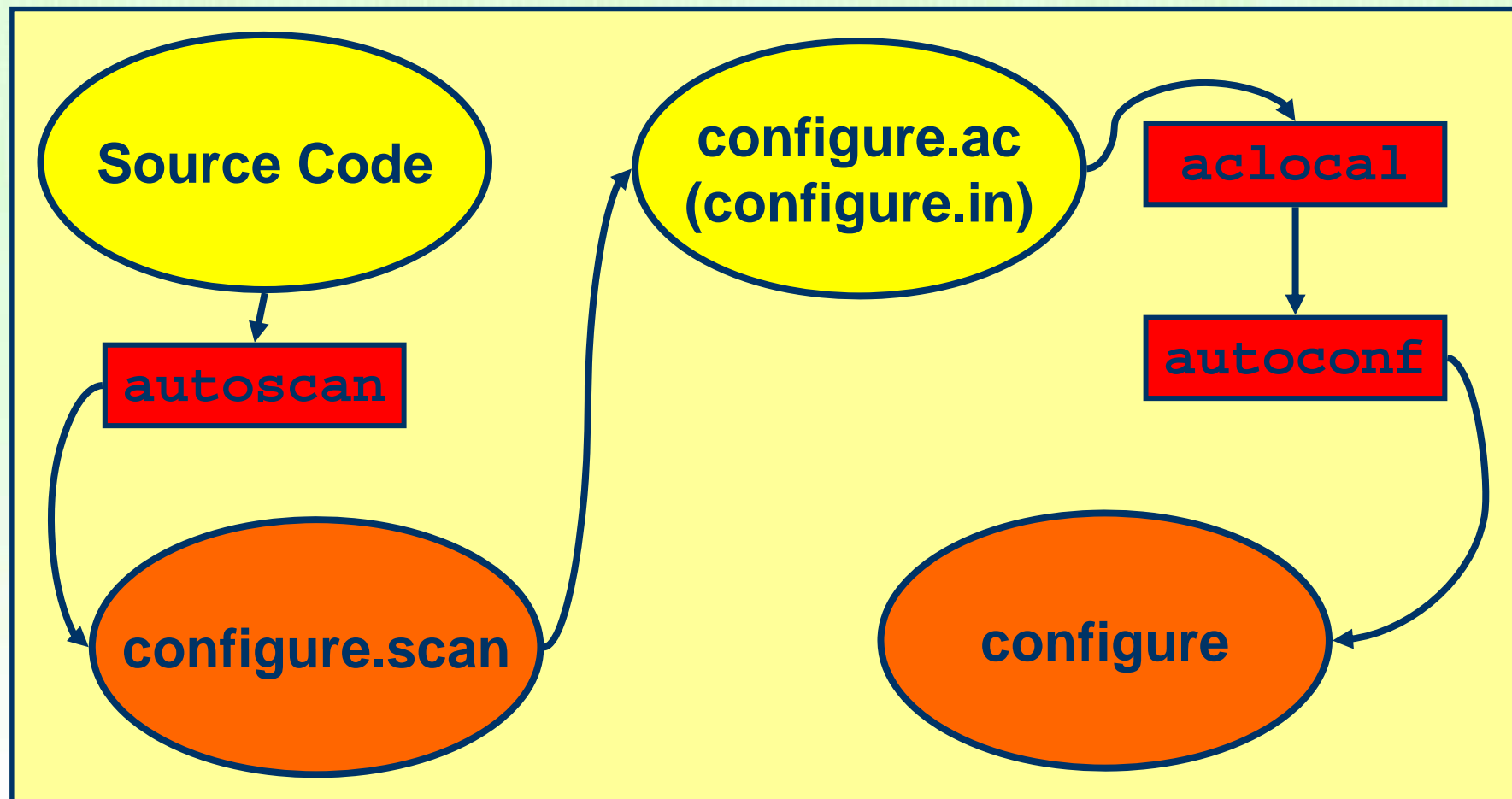
## ❖ Automake

- 根据用户提供的一个高层次的生成规则Makefile.am，生成Makefile文件的模板Makefile.in
- Automake生成的Makefiles符合GNU的Makefile标准，用户无需再手工编写Makefile文件



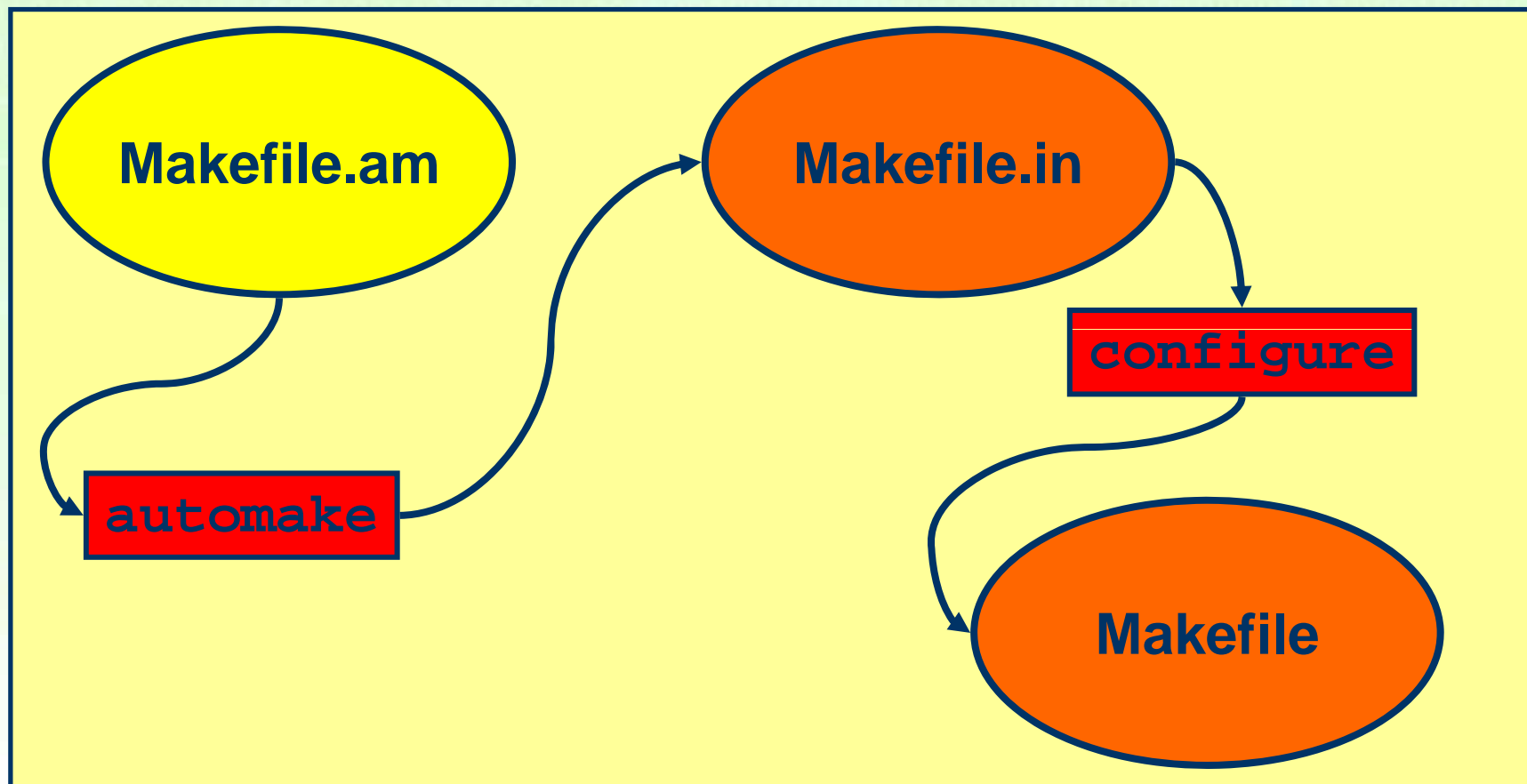


# Autoconf





# automake





# 主要内容

## ❖ 背景知识

- Linux简介
- Linux系统环境
- Linux用户管理命令
- Linux文件操作命令
- Linux网络管理命令
- Linux系统信息命令
- Linux编程环境

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核



# 安装Ubuntu 8.0.4

## ❖ 实验说明

- 学习和动手安装发行版本Ubuntu 8.0.4
- 掌握操作系统的系统配置
- 了解建立操作系统应用环境的过程

## ❖ 解决方案

- Ubuntu硬件需求
- 软件获取途径
  - ✓ <http://www.ubuntu.org.cn>
- 安装模式



# 安装Ubuntu 8.0.4

## ❖ 安装步骤

- 选择系统语言
- 选择时区、设定时间
- 选择所使用的键盘
- 进行分区
  - ✓ /swap 交换分区
  - ✓ /根分区
- 设定帐号
- 确认配置





# 主要内容

## ❖ 背景知识

- Linux简介
- Linux系统环境
- Linux用户管理命令
- Linux文件操作命令
- Linux网络管理命令
- Linux系统信息命令
- Linux编程环境

## ❖ 实验内容

- 安装Ubuntu 8.0.4
- 编译Linux内核



# 编译Linux内核

## ❖ 实验说明

- 学习编译Linux内核
- 在Ubuntu 8.0.4下动手编译并安装一个新的Linux2.6内核
- 掌握操作系统的系统配置和常用的编译选项

## ❖ 解决方案

- 获取内核所需软件包
- 下载源代码
- 解压缩
- 给内核打补丁
- 配置内核
- 编译内核
- 安装内核



# 第1章 Linux安装与编译