一、基于虚拟技术的 Linux 安装

1. 新建虚拟机

a) 安装位置剩余空间最少 20G
b) 虚拟内存选择小于等于 2G
c) 网络连接-----桥接
d) 虚拟磁盘类型 IDE
e) 虚拟磁盘拆分成多文件、适中

2. 修改虚拟机设置

a) 移除 autoinst.iso 光驱、floppy（软盘）、打印机等
b) 添加串口 Serial Port
c) 创建共享文件夹

3. 安装设置

【注】:（命令）linux rescue #用于系统修复

a) 检测安装介质、安装号码 -------跳过
b) 创建分区

类型	大小	作用	挂载点	
swap	内存 2 倍	虚拟内存		交换分区
ext3	100M	引导分区	/boot	
ext3	剩余全部	根	/	

c) 软件选择“软件开发”“网络服务”“现在定制”
d) 防火墙-----禁用
e) SELinux-----允许

键盘组合键：<Ctrl +Alt+Backspace> ------- 作用：重启 X-windows

二、系统的优化与基本设置

1. 安装虚拟机工具（VMware Tools）

a．挂载安装光盘
b．进入光盘目录　[root@localhost ~]# cd /media/
c．将 VMware Tools 解包到 /usr/local/src
tar –zxvf VMwaretools[tab][tab] –C /usr/local/src
d.执行安装脚本 *.pl

2. 修改终端下显示模式

a．打开/etc/grub.conf
b.　在 kernel 行尾加入 vga=791

3. 修改系统默认运行级别

a．打开/etc/inittab
b.　将 initdefault 行中的 5 改为 3

【注】修改完 2、3 文件后重启，系统将默认进入文本状态

Linux 的运行级别

0 *关机*
1 *单用户无需登录直接获得 root 权限（系统维护）*
2 *多用户、无网络（排除网络故障）*
3 *多用户、文本模式（系统、网络管理）*
4 *自定义*
5 *图形（桌面用户）*
6 *重启*

4. 关闭冗余服务

在终端下执行 ntsysv

保留服务	haldaemon	图形界面支持
	kudzu	即插既用设备
	messagebus	图形界面支持
	network	
	nfs	nfs 服务（网络文件系统可以将服务器上的目录共享给远程的计算机，
	protmap	

		并在远程计算机上挂载）
	vmware-tools	
	vmware-tal。。。。	
	xfs	超级服务器---可以管理很多不能独立运行的网络服务
	xineted	

三、文件系统

	目录	作用	备注
/	boot	Linux 内核、引导程序（grub、lilo）	内核的作用： 文件管理、内存管理、进程管理、设备管理
	bin	普通用户可以执行的命令	
	sbin	只有超级用户可以执行的命令	
	etc	配置文件目录	
	proc	内核启动映像	观察内核的运行状态、微调内核的一些参数
	sys	系统总线映射	
	usr	第三方软件目录	/usr/include C 头文件
	root	超级用户专属目录	
	home	普通用户家目录	
	media	光盘挂载目录	
	mnt	外部存储设备挂载目录	
	tmp	临时文件目录	
	var	日志（网站）等需要频繁访问的文件	
	lib	C 库目录	
	dev	设备文件目录	每个文件对应一个设备，而设备可能存在，也可能不存在
		/dev/ttyn	物理终端
		/dev/ptsn	虚拟终端
		/dev/zero	只读 ——0
		/dev/null	只写 ——丢弃
		/dev/random	只读 ——随机数
		/dev/consde	操作系统所必须，关系到系统登录

四、常用操作

1. <Alt+Fn> n=1~7　——物理终端间切换
2. 文本下启动图形界面　startx
3. 退出图形界面 <Ctrl+Alt+Backspace>
4. 图形界面下运行文本<Ctrl+Alt+Fn> n=1~6
5. 文本进入图形　<Alt+F7>
6. 终止程序运行 <Ctrl+C>
7. 执行程序运行命令时，在末尾家&——表示让程序在后台运行
8. 作业调度
   jobs　#查看作业号
   fg 作业号　#将后台作业调入前台
   <Ctrl+Z>　表示挂起前台作业
   bg 作业号　#将挂起作业调入后台
9. <Ctrl+L>　清屏
10. history　#显示历史命令　　默认最多 1500 条
11. ！历史命令编号　#重新执行该命令
12. 历史相似命令
    a．<Ctrl+r>　　注：在宿主机即虚拟机外将重启虚拟机
    b．输入相似部分
    c．回车执行
13. Tab 用法
    a．补全命令
    b．显示相似命令
    c．补全目录或文件

附：C 语言输入输出

1. printf

只能写标准输出中输出的字符串

2. fprintf

可以写任意文件中输出的字符串

【注】在 Linux 中一切皆文件

a．stdin ——标准输入

b．stdout ——标准输出

将数据存入缓冲区，待缓冲区满或遇到换行字符或程序结束时，才向输出设备上输出内容

c．stderr ——标准错误输出（不经过缓冲区直接输出）

例：fprintf(stdout, “Hello”);

3. 返回值 return

a．return EXIT_SUCCESS;

EXIT_SUCCESS 宏定义在 stdlib.h 文件中，宏值为 0，表示程序运行成功并退出

b．return EXIT_FAILURE;

EXIT_FAILURE 宏定义在 stdlib.h 文件中，宏值为 1，表示程序运行失败，并退出

注：返回值使用宏定义目的在于提高程序的可读性，在 main 中使用也可以在调用 exit 函数中使用。

void exit（int status）结束当前进程，status 形参的值用于程序退出的值，相当于 main 函数中的 return 值

4. 异常处理

assert(0) //如果形参的值为真时，不做任何处理；否则退出程序，并报错

例：

```
#include <stdio.h>
#include <stdlib.h>
void fun(void)
{
        int n = 0;
        assert(n!=0);
        fprintf(stdout, "%d\n", 5/n);
}
int main(int argc, char* argv[])
{
        fun();
        return EXIT_SUCCESS;
}
```

例 2：

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3
 4 int main(int argc, char* argv[])
 5 {
 6         int n = 0;
 7
 8         while(1)
 9         {
10                 sleep(1);
11                 fprintf(stderr, "\r%02d", n++);
12         }
13
14         return EXIT_SUCCESS;
15 }
```

五、常用命令

a. 开关机命令

1. reboot
2. shutdown -r now } 重启
3. init 6
4. shutdown -h 0
5. poweroff } 关机
6. init 3
7. logout
8. exit } 退出登录（终端下）

b. 文件操作命令

1. pwd

功能：显示当前工作目录的绝对路径

2. ls

功能：显示文件和目录列表
语法：ls [参数列表] [文件或目录名称列表]
参数： -a 显示所有文件包括隐藏文件（文件名以.开始的文件）
-l 以长格式显示文件完整信息
-h 以适当单位显示文件大小（必须与-l 配合）
-R 递归显示目录中的所有文件
-i 显示文件的 i 节点的值
节点：每个文件系统会对磁盘上的文件进行编号，这个号码在当前文件系统中是唯一的
【注】中括号——可选项
尖括号——必选
列表用空格分隔，不特殊说明，次序不限
【附】提示符：$PS1 用于定义 shell 提示符，显示内容与颜色
例：echo $PS1
\u : 当前用户
\h : 当前主机
\w: 当前工作目录
\$: 提示符类型 { # 超级用户 / $ 普通用户

\A：24 小时格式显示 HH:MM

我的设置：

PS1="\e[36m\A \e[32m[\u\e[31m@\e[33m\h \e[32m\w]\e[31m\$\e[30m"

显示效果：

ls 长格式显示说明：

第一列：文件权限及类型说明

1	2 3 4	5 6 7	8 9 10

文件类型　　所有者权限　所属者权限　其他人权限

[　- 普通文件

d 目录

l 符号链接（软链接）

p 管道文件

c 字符设备文件——顺序读写

b 块设备文件——随机读写（意义：统一设备访问接口，除特殊设备（网卡））

s 套接字文件

文件颜色

黑色——普通文件　蓝——目录　黄——设备　浅蓝——符号链接

红——压缩文件　绿——可执行文件　红闪——丢失目标的符号链接

紫——套接字　棕——管道文件

第二列：硬连接数（一个文件有几个文件名，而它的 i 节点号相同）

第三、四列：文件大小（空目录大小 4096）

注：设备文件表示主次设备号，设备号相同，表示使用相同的驱动程序，次设备号，表示不同的子设备

第五列：文件时间　　ctime　创建时间

atime　最后访问

mtime　最后修改

注：alias　ll='ls -l'　　(临时修改命令别名)

3. cd

功能：改变工作目录

语法：cd　[目标目录]

示例：cd　-　（回到上一次所在工作目录）

4. mkdir

功能：创建目录

语法：mkdir　[参数]　[目录列表]

参数：-p　递归创建

5. touch

功能：修改文件时间或创建一个空文件
语法：touch　[文件列表]
注：如果文件存在用来修改或目录的时间（当前系统时间），否则创建一空的普通文件

6. rm

功能：删除文件或目录
语法：rm　[参数]　[文件或目录列表]
参数：　-r　删除目录时加此参数
　　-i　每删除一个文件或目录时提醒
　　-f　删除前不提醒
注：rm 命令默认情况下，不提示，直接删除，所以在很多操作系统上位 rm 命令起别名：alias rm='rm -i'

7. mv

功能：文件目录移动或更目
语法：mv　[参数]　源文件或目录　目标文件或目录
参数：-f　覆盖前不提醒（强制删除 force）
注：目标文件或目录，存在——移动；不存在——更名

8. cp

功能：复制文件或目录
语法：cp　[参数]　源文件或目录列表　目标文件或目录
参数：　-r　复制目录，默认只复制文件不复制目录
　　-f　覆盖前不提醒
　　-a　不改变文件的权限和属性
注：如果文件已存在，覆盖时保持原有权限属性不变

9. ln

功能：创建链接文件
语法：ln　[参数]　源文件或目录　链接文件名
参数：-s　创建软连接（符号链接）文件
注：默认情况下。ln 创建硬连接文件

10.cat

功能：查看文件
语法：cat　[参数]　文件名
参数：　-n　显示行号
　　-b　空行不显示行号
　　-s　连续多个空行，只显示一个空行

11.more

功能：分页显示文件
　用空格翻页，q 退出

12.less

功能：同 more
用<pageup/down>翻页，q 退出

13.tail

功能：显示文件尾部内容
语法：tail　[-n　行号]　文件名

14.head

功能：显示文件首部内容，使用同 tail

c. 查找相关

1. find

功能：文件查找
语法：find　起始路径　-name　文件名 [-ls]
说明：基于文件名查找，[-ls]用于长格式显示查找结果
注：文件名中可以使用通配符
　　*可以通配任意个任意字符
　　？可以通配一个任意字符

2. grep

功能：基于文件内容查找
语法：grep [参数] 查找内容 起始路径或文件名
参数： -n 显示行号
-r 递归查找
-i 忽略大小写

3. updatedb 与 locate

功能：基于数据库进行查找
语法： 创建数据库（整个磁盘文件名数据库） updated
查找 locate 部分或全部文件名
注：在更新数据库后的文件改变无法查找

4. which

功能：命令或命令别名查找
语法：which 命令或命令别名

d. 用户相关

1. group

功能：添加组
语法：group 组名 [-g 组 ID]
注：系统会为用户和组进行编号，小于 500 的为系统用户或系统组，默认情况下新添加的组和用户的 ID 由 500 开始

2. useradd 或 adduser

功能：添加用户
语法：useradd [-g 组名] [-d 家目录] 用户名
参数： -g——省略
i. 先创建与用户名相同的组
ii. 创建用户并添加到此组
-d——省略 创建家目录 /home/用户名

3. userdel

功能：删除用户
语法：userdel [-d] 用户名
参数：-d 删除用户的同时删除其家目录

4. group

功能：删除组
语法：group 组名
注：必须是空组才能删除

5. passwd

功能：修改用户密码
语法：passwd [用户名]
注：不加用户名，修改当前用户密码；否则，修改指定用户密码，但仅限超级用户

6. su 与 exit

功能：用于切换用户
语法：切换 su [-] [目标用户]
参数： - ——不写，只切换用户不改变环境设置
省略目标用户——相当于 root
示例：
root { su linfeng ——$PATH (root)
root { su - linfeng ——$PATH(linfeng)
返回原用户 exit

e. 文件权限相关

1. chgrp

功能：改变文件所属组
语法：chgrp 组名 文件或目录名列表 [参数]
参数：-R 递归修改

2. chown

功能：改变文件或目录的所有者和所属组
语法：chown [参数] 所有者.所属组 文件或目录列表
参数：-R
说明： 所有者 —— 只修改所有者
所有者.所属组 —— 两者都修改
.所属组 —— 只修改所属组
示例：chown -R linfeng.embedded a.c

3. chmod

功能：改变文件权限
语法：chmod [参数] 权限表达式 文件或目录列表
参数：-R
权限表达式：

i. 八进制
ii. 组合表达

所有者	u	+	r
所属组	g		w
其他人	r	-	x
以上三组	a	=	

示例： chmod u+x test #给 test 文件所有者执行权限
chmod ug+rw test

附：
etc/passwd #该文件存储用户的配置信息
etc/group #该文件存储用户组配置信息

f. 磁盘管理相关

1. fdisk

功能： （1）分区管理
fdisk 磁盘设备文件名
示例：fdisk /dev/sda #对/dev/sda 磁盘进行分区
（2）磁盘设备查看
fdisk -l

2. mount

功能：磁盘设备挂载
语法：mount　[-t　分类类型]　[-o　挂载参数列表]　设备文件名 挂载点
分区类型：

类型	用途	备注
vfat	Win 下 fat 和 fat32	通常可省略参数 -t
ext3	Linux	
iso9660	光盘	
smbfs	Win 下网络邻居共享目录，Linux 对应 samba 文件系统	
nfs	Network file system	
yaffs	嵌入式设备，可读写文件系统	
ntfs	Win 下 ntfs	

挂载参数：ro　只读
rw　读写
gid　指定挂载点组 ID
uid　指定挂载点用户 ID
mode=xxx　指定挂载点权限
iocharset=cp936　（字符设备乱码问题）

注：在 UNIX 族操作系统上，将存储设备与目录进行关联的操作称为**挂载**，被关联的目录称为**挂载点**。

3. umount

功能：卸载
语法：umount　挂载点或设备文件名
注：device a busy 的原因　——　此设备正在使用

4. df

功能：文件系统查看
语法：df　[参数]　[分区设备文件名或挂载点]
参数：-h　以适当单位显示大小
示例：df　-h　/dev/sda1

5. du

功能：查看目录使用情况
语法：du　[参数]　[目录名]

参数： -s 只显示总计使用情况
-h 同上
示例：du -sh /mnt/usb

g. 系统管理

1. env

功能：查看系统环境变量
变量格式： 变量=值
常见变量：

HOSTNAME	当前主机名
SHELL	当前 shell
HISTSIZE	历史命令记录数量
USER	当前用户名
LS_COLORS	指定 ls 显示不同文件的颜色
PATH	命令路径
PWD	当前路径
LANG	当前语言类型
HOME	家目录
LOGNAME	登录用户名
OLDPWD	上一次所在工作目录

2. date

功能：查看和修改系统时间
语法： （1）查看 date
（2）修改 date MMDDHHmm[[cc]YY][.SS]
月 日 时 分 年 秒

3. hwclock

功能：查看和修改 rtc 时钟
语法： （1）查看 hwclock
（2）修改 hwclock -s （以 rtc 时钟同步系统时钟）
hwclock -h （以系统时钟修改 rtc 时钟）

4. ifconfig

功能：IP 查看与设置
语法：（1）查看 ifconfig [网卡设备名]
（2）配置 ifconfig 网卡设备名 IP 地址 [netmask(子网掩码)]
注：网卡名 eth m[n]
m ——物理网卡 ID
n ——虚拟网卡编号（一个网卡最多虚拟四个网卡）

h. 包管理命令

1.tar

功能：打包或解包
语法：1）打包 tar zcvfj 包文件名 目录
参数：z gzip 压缩
c 创建包（必选）
v 显示打包过程（可选）
f 打包成文件（必选）
j bizip 压缩
压缩类型比较：

类型	压缩比	速度	扩展名
bizip	大	慢	.tar .bz2
gzip	小	快	.tar .gt 或.tgz .

2）解包 tar zxvf 包文件名 [-C 目标目录]
参数：x 解包
-C 指定解包路径
3）查看包 tar ztvf 包文件名

2. rpm

功能：红帽包管理
用法：1）安装包
rpm -ivhU --nodeps rpm 包文件名
参数：-i 安装（install）
-v 显示安装过程
-h 显示安装进度
-U 若包已安装则更新
--nodeps 不检查依赖关系直接安装
2）查看系统已安装了哪些 rpm 包
rpm -qa

3）卸载 rpm 包

rpm　　-e　包名

4）查看已安装文件

rpm　　-ql　包名

㊙：“|”管道符

cmd1 | cmd2　——　cmd1 的标准输出将为 cmd2 的标准输入

附：

创建以下目录结构满足学习需要

```
/work/
|-- C
|-- C++
|-- database
|-- embedded
|     |-- bootloader
|     |-- filesystem
|     |-- kernel
|     |-- qt
|     `-- toolchains
|-- process
|-- project
|-- qt
|-- shell
|-- signall
|-- socket
|     |-- tcp
|     |     |-- client
|     |     `-- server
|     `-- udp
|-- software
`-- systemcall
      |-- base
      |-- jpegdisplay
      `-- waveplay
```

遇到问题根据提示解决进行分析

1）在哪里产生
2）为什么产生
3）解决方案
4）解决问题（能备份一定要备份，数据无价）

I. 进程相关

1. ps

功能：查看进程
语法：ps [参数]
参数：a 显示所有终端进程
u 显示进程详细信息
x 显示系统进程
示例：ps aux
每列意义：USER 进程拥有者
PID 进程号
注：进程号范围 1~65535，每个进程加 1，循环递增使用
进程 ID 为 1 的进程为 init 进程是系统第一个进程，又称为初始进程，是所有进程的“父”进程，在系统运行过程中其一直驻留内存
%CPU
%MEM 内存占用率
VSZ 虚拟内存大小
RSS 物理内存大小
TTY 进程所依赖的终端
- ? 不依赖任何终端
- ttyn 占用终端
- ptsn 虚拟终端

注：在 Linux 中所有设备个数标号几乎都从 0 开始，因此 tty2 将表示第三个终端

STAT 进程运行状态
- S 休眠
- R 运行中
- Z 僵尸进程

START 启动时间
TIME 运行时间（实际占用处理器的时间）
COMMAND 启动命令

2. top

功能：动态进程查看
语法：top
命令：m 显示内存统计信息
q 退出

3. free

功能：显示内存信息

语法：free [参数]
参数： -g
-m
-k
-b
注：基本单位，默认以 k 为单位

4. kill

功能：给指定进程发送指定信号
语法 1：查看信号列表
kill -l

SIGUP	挂起
SIGINT	终端中断信号
SIGAPRT	程序异常终止
SIGFPE	浮点数例外
SIGKILL	进程终止
SIGSEGV	段错误（内存非法访问，win 下蓝屏）

语法 2：为进程发送信号
kill -信号值 进程 ID
注：信号——在 Linux 操作系统上用于实现进程间的简单通信

5. killall

功能：结束进程（基于进程启动命令）
语法：killall 进程启动命令
示例：killall httpd #daemon 进程守护
注：内部实际发送的信号为 SIGKILL
注：httpd 网站服务器程序，IIS，apache（阿帕奇）

六、VI 编辑器

a. 进入与退出

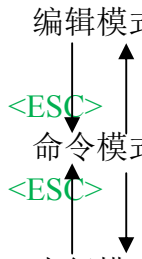
a) 1. 进入

Vi [文件名称] [+行号]

b) 2. 退出

:wq	保存并退出
:q	退出
:q!	不保存退出（强制退出）
:w	保存
:w 文件名	另存为
:w!	强制保存

b. 工作模式

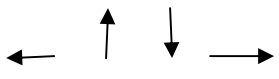


插入	I	。。。。。行首
	i	插入到当前光标前
追加	A	。。。。。行尾
	a	追加到当前光标后
新行	O	。。。。。。。上。。。。。
	o	在当前行下新建一行
替换	S	。。。。。。。行
	s	替换当前字符

c. 编辑命令

复制	[n]yy	复制 n 行，复制一行时可省略 n
剪切	[n]dd	剪切 n 行，剪切一行时可省略 n
粘贴	p	将剪切板内容粘贴到当前行下（可视情况除外）
撤销	u	撤销编辑操作
恢复	:redo	恢复已经撤销的操作

d. 光标移动



<pageUp> <pageDown>
<home> <end>

W	下一字符首（白空格——空格、制表符、换行）
w	下一单词首（白空格，字符类型不同划分单词）

E　　下一字符串尾
e　　下一单词尾
gg　　文件首
G　　文件尾
:行号　调到第几行

e. 查找与替换

1. 完全匹配查找

1) 将光标移动到要查找的单词上
2) 按“#”
3) 大写“N”下一处
   小写“n”上一处

2. 部分匹配查找

1) /字符串
2) 大写“N”下一处
   小写“n”上一处

3. 替换

:s/原串/新串[/g]　#g 表示全局替换

f. 分屏操作

水平分屏　:split　文件名
垂直分屏　:vs　文件名
<Ctrl+w> . W　下一屏
<Ctrl+w> . W　上一屏

g. 可视

主要用于非整行的复制、粘贴、剪切操作

1. 将光标移动到可视的开始或结束处，按“v”
2. 通过光标移动命令将光标移动到可视的结束或开始处
3. 通过“d”剪切、“y”复制
4. 通过“p”可将剪切或复制的可视内容粘贴到光标后

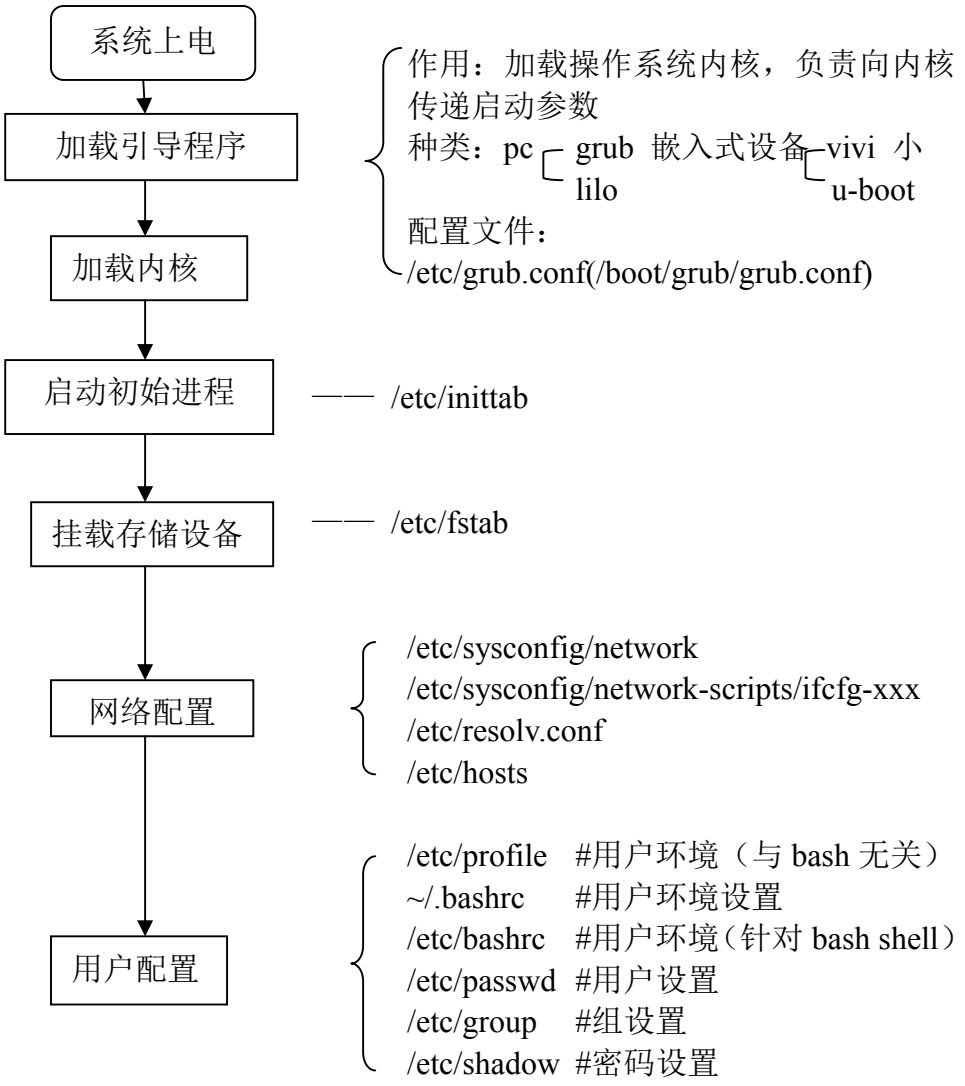
f. Vi 配置

行号　:set nu
　　:set nonu
语法加亮　:syntax on
　　:syntax off
去除查找内容加亮　:nohlsearch
设置水平制表符缩进　:set ts=n　#n 为缩进量

永久配置

i. 修改~/.bashrc　#加入 vi='vim'的别名命令
ii. 编辑 vi 配置脚本
　　~/.vimrc

七、Linux 主要配置文件



1. /etc/grub.conf

a. 全局变量（顶格）

default=值　#用于指定默认引导的操作系统（由 title 次序决定其值）

timeoutout=值　#默认引导超时时间

splashimage=(hd0,0)/grub/splash.xpm.gz　#背景图

#个性制作——320*480 索引图——保存 xpm 格式——gzip 压缩

#(hd0,0)表示第一块硬盘的第一分区，相当于/boot/

Hiddenmenu　　#隐藏操作系统菜单选项

title #菜单项文件

password=xxxxx　　#MD5 密码（128 位）

b. 局部变量（一个制表符）

root 用于指定引导分区（内核所在分区，即/boot 文件系统）

kernel　/vmlinuz-2.6.18-308.el5 ro root=LABEL=/ rhgb quiet vga=791

#用于指定内核引导参数

1.内核所在文件 2.以只读方式挂载根分区

3.用于指定根文件系统的位置[标签]（等同于 root=/dev/hda3）

4.启动时不打印任何信息

5.用于指定终端显示模式

vga——frame buffer 模式（以内存作显存，可不用显卡）

791——十六进制表示 1024*768*16bpps（16 表示十六位色，屏幕上一点，占两个字节用来放三原色，rgb565）

5r	6g	5b

initrd　#系统启动镜像文件

chainloader　+1　#工具链

注：

注：生成 MD5 密码命令

[root@linfeng ~]# grub-md5-crypt

```
sudo apt-get install hwinfo
安装完成后
sudo hwinfo --framebuffer
结果就可以看到：
Mode 0x0300: 640x400 (+640), 8 bits
Mode 0x0301: 640x480 (+640), 8 bits
Mode 0x0303: 800x600 (+800), 8 bits
Mode 0x0305: 1024x768 (+1024), 8 bits
Mode 0x0307: 1280x1024 (+1280), 8 bits
Mode 0x0311: 640x480 (+1280), 16 bits
Mode 0x0312: 640x480 (+2560), 24 bits
http://forum.ubuntu.org.cn/viewtopic.php?t=236915
```

2. /etc/inittab

格式：　　标号:运行级别:动作:命令

1. 自定义（2~3 个字符）
2. 0~6 之间，0 项或多项，表示此配置项在哪个初始化级别下生效，不写表示所有级别均生效
3. 默认
   - initdefault　#指定系统默认初始级别
   - sysinit　#指定系统初始化脚本(系统参数、环境变量）
   - wait　#等待该项执行完毕后，再执行其他配置项

3. /etc/fstab

格式：

设备文件名 挂载点 文件系统　挂载参数 开机是否磁盘检测 挂载次序

前四项参考 mount　　1——检测

0——不检查

由 1~n 逐次挂载。0 最后挂载，

注：如果在文件中出现的存储设备，挂载时仅需指定设备文件或挂载点即可。

例如：fatab 中有一行

/dev/cdrom　/media　iso9660　defualts　0　0

挂载此设备时，仅需执行　mount　/dev/cdrom

mount　/media

4. /etc/sysconfig/network

NETWORKING=yes/no　#系统启动时是否开启 IPV4 网络支持

HOSTNAME=xx.xx.com　#当前主机名称（全名）

GATEWAY=xx.xx.xx.xx　#网关 IP

注：网络是否启动与网关地址配置，重启网络后生效

主机名配置重启系统后生效

重启网络方法：

Linux 下服务管理命令　service　network　restart / start / stop / status　#状态

（见 ntsysv）

5. etc/sysconfig/network-scripts/ifcfg-xxx(ethm[:n]/lo(回环))

DEVICE=xxx　#设备名等同于（完全相同）

BOOTPROTO=static/dhcp　#静态设置或动态获取

BROADCAST=xxxx　#广播地址

HWADDR=xxxx　　#MAC 地址一般不能改动(可以没有,但不能错)
IPADDR=xxxx　　#IP 地址
NETMASK=xxxx　　#子网掩码
NETWORK=xxxx　　#网络
ONBOOT=yes/no　　#网卡是否随网络启动（no——禁用网卡）

注：网卡重启生效

重启网卡　　ifdown　　网卡设备
　　Ifup　　网卡设备

6. /etc/resolv.conf

DNS 客户端设置——（作用是主机名转为 IP 地址）

search　xxxxx　　#没用可以删除
nameserver　　DNS 服务器 IP　　#（最多三个）

7. /etc/hosts

主机表文件——主要负责本机 IP 与主机名解析与反解析
（不添加主机表，可能导致图形界面起不来或启动缓慢）

8. /etc/profile

用户环境——所有用户均执行此文件，重新登录生效（Bash 无关）

9. ~/.bashrc

用户环境设置——只针对当前用户有效，重新登录后生效

10. /etc/bashrc

用户环境设置——针对所有用户,重新登录后生效(针对 Bash shell 设置)

11. /etc/passwd

用户设置——格式

用户名：是否需要登录密码：用户 ID：组 ID：用户说明：家目录：用户 shell

X-需要　空-不需要

注：修改后立即生效

12. /etc/group

组设置——格式：

组名：组密码：组 ID：组扩展用户列表用“，”分隔

有无无所谓

13. /etc/shadow

密码设置——格式：

用户名：MD5 密码：。。。。。。。。。。

附：

开机后进入单用户系统

1. 按“e”进入编辑模式
2. 修改内核参数，在最后加 1（即可进入单用户）

<Ctrl+d>　　结束单用户

取消 Tab 键报警声

进入/etc/inputrc

删除 set bell-style none 行前面的注释符#

八、shell 编程

Shell：用于用户与内核进行交互的一段程序
分类： 1）GUI 用户图形接口
2）CUI 终端用户接口
[ash —— 嵌入式设备（小）
[bash —— pc 机（全）
[csh —— 网络设备（网）

1. shell 脚本的构成

1） 命令
2） 变量
3） 注释（只有单行注释，以#号开始，至行尾结束）
4） 解释器说明
#!解释器路径
#!/bin/sh——符号链接，用于指定当前系统使用的默认 shell
注：嵌入式系统中的应用
a) 系统的环境设置
b) 启动或配置应用
c) 设备的加载与参数设置
Shell 脚本默认扩展名为.sh
echo #显示字符串、变量
参数：-n 不换行显示

2. Shell 脚本的执行方法

1) 使用当前 shell 解释执行
语法： source 脚本文件名（可包含相对或绝对路径）
. 脚本文件名（可包含相对或绝对路径）
2) 使用指定 shell 解释执行
语法： shell 命令 脚本文件名
示例： sh test.sh
3) 直接运行（脚本中指定 shell 执行）
语法： 路径/脚本文件名
示例： ./test.sh

3. 变量的分类和使用

三种形式变量：

a) 系统变量：由操作系统设置（env）

b) 预定义变量：

$? —— 上一条命令的执行结果（任何命令执行成功返回 0，执行失败返回非 0，而非 0 值一般为错误号）

$0~$9 —— shell 脚本执行时的命令行参数

例：./test.sh [参数列表(空格分隔)] #实际使用$1~$9,$0 代表自身

c) 自定义变量：

i. 变量无类型

ii. 无需声明

语法：

i. 赋值 规定：变量名大写，单词下划线连接；等号两端不能有空格；值中包含空格要使用引号括起来

ii. 引用：$A

iii. 释放：unset A B C （变量列表空格分隔）

注：‘单引号’不支持变量，”双引号”支持变量

`反引号`用于括起来一条 shell 命令，当执行到包含反引号的语句时，当先执行反引号中的命令

4. 测试语句

1) 文件测试

语法：test 测试符 文件名/目录

[测试符 文件名/目录]

测试符：

-r 读

-w 写

-x 执行

-d 目录

-f 文件

-L 符号链接

-e 存在

2) 数值测试

语法：test 数值 1 测试符 数值 2

[数值 1 测试符 数值 2]

测试符：

e 等于

n 不等于

l 小于

g 大于

t、q 无意义

示例： -lt -le -nl -eq -ge

3) 字符串测试

语法 1：test 串 1 测试符 串 2

[串 1 测试符 串 2]

测试符： == !=

注：测试中一旦出现变量要用双引号将其括起来，否则可能出错

语法 2：test 测试符 串

[测试符 串]

测试符： -n 是否非空

-z

4) 逻辑测试

-a and 与

-o or 或

! 非

5) 算数运算

语法：expr 操作数 1 运算符 操作数 2

运算符：+ - * /

(运算符两边加空格，乘法需要使用转义*)

6) 特殊符号

? * 通配符

` ` 反引号

\ 转义符

	模式	符号	说明
输入重定向	新建	<	将符号后内容定向到符号前命令的 stdin
	追加	<<	
输出重定向	新建	>	将符号前命令的 stdout 定向到符号后的文件
	追加	>>	（不清空原有内容）

语法： 输入重定向 cmd < 串

输出重定向 cmd > 文件

注：标准错误输出，是将 stderr 定向到 stdout 然后输出(2>&1)

stdin 0

stdout 1

stderr 2

例: ls adsadas >>test 2>&1

ls adsadsa >>/dev/null 2>&1

从命令行读取数据

语法：read 变量名

5. 流程控制

1) 分支（if）

```
if测试 1
then
  命令 1
elif   测试 2   ┐
then            ├ 可重复，0~任意次
  命令 2        ┘
else            ┐
  命令 3        ├ 可选
                ┘
fi
```

2) 分支（case）

```
case  $变量  in                                  ┌ 数值)          5)
模式 1)     #模式用于设置匹配方式                ├ 字符串)        stop)
    命令                                         └ 正则表达式)    9[0-9])
    ;;
模式 2)
    命令
    ;;
模式 n)
    命令
    ;;
esac
```

3) 循环（while）

```
while   测试      #真值时循环
do
    命令
done

until   测试      #假值时循环
do
    命令
done
```

4) 循环（for）

```
for 变量名  in  值列表（空格分隔）
do
    命令
done
```

6. 函数

用法：

定义： function 函数名

{

#函数体

}

调用： 函数名 参数列表

注：参数列表与命令行参数用法一样，$0~$9

函数在 shell 当中相当于一条命令

附：

1. 命名法：

   匈牙利法：字母全小写，单词以下划线分隔

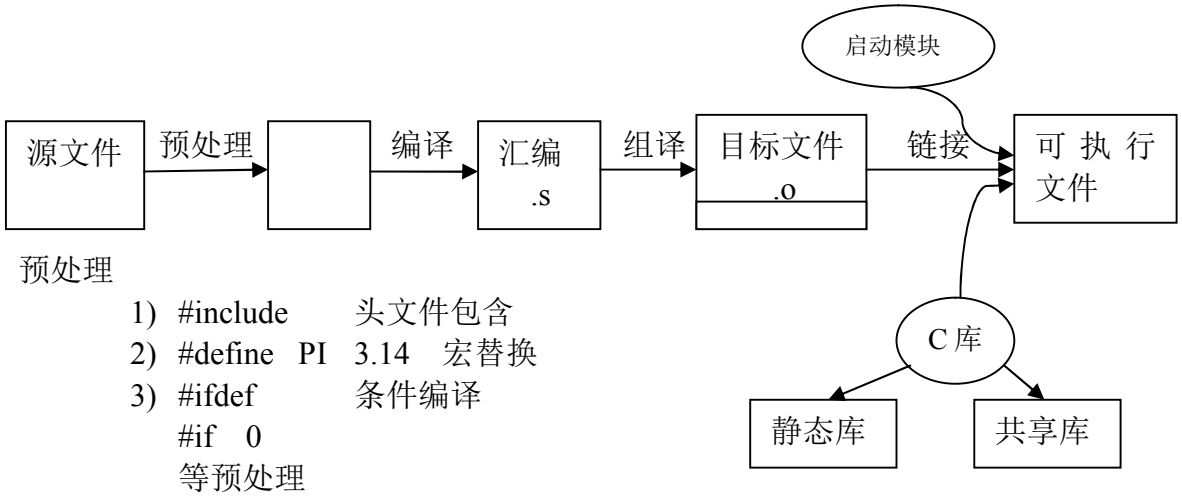
   骆驼法：函数——每个单词首字母大写

   变量——第二个单词开始首字母大写
2. exit 1 #退出当前 shell
3. 目录的可执行权限

   表示是否可以进入这个目录，软连接文件的权限都是 777

九、C 开发工具

1. Gcc —— GNU（GUN is Not Unix）



预处理

1) #include 头文件包含
2) #define PI 3.14 宏替换
3) #ifdef 条件编译
   #if 0
   等预处理

静态库：编译时加载，运行时无关
共享库：编译时检测，运行时加载

2.语法：

gcc [参数] [源文件列表]

3.参数：

c) 过程控制

Gcc 默认编译到链接阶段，生成可执行文件
-E 仅编译到预处理阶段，将预处理后的代码作为 stdout
-S 进行到编译阶段，生成同名的.s 汇编文件
-c 进行到组译阶段，生成同名的.o 目标文件
-o 文件名 用于指定编译器输出文件时的文件名
注：编译器自带宏 __FILE__, __FUNCTION__, __LINE__
表示 当前文件 所在函数 当前行号

d) 预处理阶段

-I 路径　　用于为当前预编译添加默认头文件（需要用到的头文件不在默认目录时）

例：gcc main.c -o main -I. #包含头文件在当前目录

-include 头文件　用于指定当前预处理包含的头文件（需要调用的头文件不在该文件中时）

例：gcc main.c -include def.h #文件中需要引用 def.h 头文件

-D 宏名　　用于向当前预处理添加宏定义

例：gcc main.c -DSUMMER #给文件一个宏参数

注：

i. 宏定义时，
宏体中出现运算符，必须将宏体用括号括起来；如果是带参宏，宏体中的宏参，必须用括号括起来。
如：#define DIV(a, b) ((a)+(b))

ii. 头文件
< > 在默认头文件目录查找
“ ” 现在当前目录下查找，再到默认头文件目录下查找

iii. 条件编译

```
#ifndef DEF_H    //如果没有定义这个宏（该头文件之前的代码中）
#define DEF_H    //定义一个 DEF_H 宏
#define VALUE 8
#endif //DEF_H    判断对应哪个 if，防止缺省
```

防止头文件重复，宏名起发为当前文件名称大写

e) 链接阶段

-l 库名　　指定链接库

例：./sin -lm #math.h 头文件需要调用链接库 (见 man sin)

注：C 库名称规范

libxxx.so.版本号 —— 共享库

libxxx.a.版本号 ┐ —— 静态库

lib.xxx.la.版本号 ┘

前缀 库名 类型

-L 路径　　添加链接库默认路径

-static　　使用静态链接库

-s　　去除冗余 —— 如标识符（见 nm）

f) 编译阶段

-On　　n 为 0~4 的整数，用于指定编译器对代码的优化级别，数值越大优化级别越高（-O0 为默认）

注：优化是编译器修改变量的存储位置与流程控制

建议：优化级别为 2 或 3

-Wall　　显示所有警告信息

建议编译参数：-Wall　-O3　-o 文件名

4.制作共享库

语法：　gcc　-fpic　-shared　源文件列表　-o　库文件名

地址可重定位　　共享库

示例：　gcc　-Wall　-O3　-fpic　-shared　*.c　-o　libxxx.so

使用：　gcc　-Wall　-O3　-s　main.c　-o　main　-L. –I.

共享库运行时不能加载解决方法：

方法 1：添加到共享库默认路径为/lib/（针对于自己制作的共享库不提倡）

方法 2：通过环境变量指定共享库位置

LD_LIBARAY_PATH 变量末尾加上自己的“：库路径”

方法 3：

a. 修改 Linux 配置文件/etc/ld.so.conf，在此文件中新起一行写上库路径

b. 执行命令 ldconfig，更新系统共享库的 hash 表

5.制作静态库

1. 将要制作的静态库的源文件编译成目标文件
2. 归档（archive）

   ar　-r　静态库名　目标文件列表
3. 编译　略（参照共享库）
4. 运行　略

6.相关文件命令

1. ldd　——查看可执行文件运行时所需要的共享库

   用法：ldd　可执行文件名
2. file　——查看文件类型、架构等信息

   用法：file　文件名
3. stat　——查看文件属性、权限、时间等信息

   用法：stat　文件名
4. nm　——查看可执行文件中的标识符（包括库文件）

   用法：nm　可执行文件名　　#strip 的文件无效（gcc 参数-s）
5. strip ——去除可执行文件中的冗余信息（包括库文件）

   用法：strip 可执行文件
6. cproto　——由.c 源文件生成.h 头文件

   用法：cproto　源文件名　#重定向到所需的.h 文件中

7.编译错误

1. 错误提示中由行号、文件、函数名是编译时出错，语法错误
2. 不显示行号等信息，链接错误

8.ftp 使用

1. ftp　服务器 IP 或主机名　　#进入 ftp 登录会话
2. 输入用户名、密码
   匿名用户，用户名：ftp　密码：无（直接回车）
   成功后进入 ftp shell
   操作命令：
   ls　　查看服务器文件列表
   cd　　切换在服务器中的目录
   get 服务器上的文件名　　下载一个文件
   put 本地文件名　　上传一个文件
   mget 服务器上的文件名（可带通配符）下载多个文件
   mput 本地文件名（可带通配符）　上传多个文件
   by　退出 ftp shell
3. 保存位置为当前打开 ftp 的目录

9.源码文件的安装过程

1. 解包到指定目录（/usr/local/src/）
2. 进入源码目录
3. 对源码软件进行配置
   配置方法：（执行配置脚本）
   ./configure　[--help]　#不加参数默认安装
   注：为什么要配置？
   a. 检测系统架构
   b. 检查库是否满足条件
   c. 检测编译器版本
   d. 生成编译规则文件
4. 编译　　make
5. 安装　　make install
   注：默认情况下，手动源码安装的库文件，在/usr/local/lib/目录下

附：规范

1. 多个源文件和头文件构成的项目，要为其创建一个目录，项目目录。该项目名应对应项目的可执行文件名
2. #include　<　>（中间加空格）
3. 运算符两侧必须加空格，一元运算符除外。
4. 逗号后加空格。
5. 声明语句和可执行语句之间加空格
6. return 之前加空格
7. 一条语句过长时，要分多行书写
8. 用水平制表符进行缩进，缩进要按层次，制表符宽度为 4（set　ts=4）
9. 文件末尾加空行

十、C 语言

1.数据类型

g) 基本数据类型

主要针对内存，为了节省内存

整型	char	1	Unsigned 值域 0~255
			Signed 值域-128~127
	short	2	Unsigned 值域 0~65535
			Signed 值域-32768~32767
	int	2	字节数-16 位系统以下
		4	字节数-32 位系统以上
	long	4	以 1 开头的 10 位整数
	long long	8	（4+4 = 8）
	short long	3	（（2+4）\ 2 =3）
实型	float	4	（精度）小数点后 6 位
	double	8	（精度）小数点后 16 位

值域范围： n = 字节数 * 8

⎧ 有符号 0 ~ 2^n – 1

⎩ 无符号 -2^(n-1) ~ 2^(n-1) – 1

常量表示法：

5	5L	5.0	5.0f	.5	075	0x123f	0UL
Int	Long	Double	float	double	八进制 int	十六进制 int	0 unsigned long
2E3	-2E-3	0.8E-5	.8E-5	8E0.5			
2*10^3	-2*10^-3	都 double					
‘a’	‘\n’	‘\m’	‘\0’	‘\123’		‘\x1234’	
字符 a	换行	m	0	字符的八进制表示法（转义字符开始不多于 3 位的八进制数）		字符的十六进制表示法（转义字符+x 开始后不多于 4 位的十六进制数）	

h) 衍生数据类型

i. 数组

声明：类型　数组名　[行长度][列长度]={{…}, {…}, ….{…}}

在内存中按行存储

ii. 指针

声明：类型　*变量名

存储内存地址的变量，为了存储所有地址，所占字节数与该系统的地址总线有关。一般与 int 类型的字节数相同。

指针的运算：

指针与整型　　p + n　p – n

如：p + 1 —— 指向下一个元素（元素表示给指针的类型单元）

Int　*P = (int *)0x1000 0000;

P + 2;

所以此时 P 为 0x1000 0008

指针与指针　　p – q

指针与指针进行减法（两指针类型必须相同），只有两指针指向同一数组时，才有意义，其运算结果为两指针间有多少个元素。

iii. 枚举

enum　枚举类型　{枚举列表}

例：enum WEEK{Monday = 5，　Tuesday，　。。。。}

i) 构造数据类型

1. 结构体
   i. 先定义类型
   ii. 通过定义的类型声明变量
   iii. 结构体取成员的运算符有两个，分别是 . 和 ->
   . 是通过结构体变量取成员 如：s.a;
   ->是通过结构体地址取成员　如：ps->a;

注：内存对齐

I. 4 字节对齐

II. 如果结构体中的所有成员的数据宽度均未达到 4 字节，按最大成员数据宽度进行对齐。

示例：

```
struct std1{
    int a;        //4
    char b;       //1
                  //占空 3
    float c;      //4
    char d;       //1
                  //占空 3
    double e; //8
}
sizeof(std1) = 24
struct std2{
    char c;    //1
               //占空 1
    short s;   //2
};
sizeof(std2) = 4
```

2. 共同体

所有成员使用同一块内存

```
union st{
        char a[2];
        short b;
};
int main(int argc, char *argv[])
{
        union st s;
        s.b = 0x1122;

        fprintf(stdout, “%x, %x\n”, s.a[0], s.a[1]);

        return EXIT_SUCCESS;
}
```

结果：22,11

判断当前系统是大端系统还是小端系统：

在上例基础上修改

```
s.b = 0x0001;
fprintf(stdout, "这是%s 端系统\n", s.a[1] ? “大” ： ”小”);
```

3. 位段结构

```
        struct st{
                unsigned int a : 5;     //5 表示位数
                unsigned int b : 8;
                unsigned int c : 9;
                unsigned int d : 1;
        };
        sizeof(struct st) = 4
```

对齐方式和结构体相同，位数不能超过前边类型的宽度

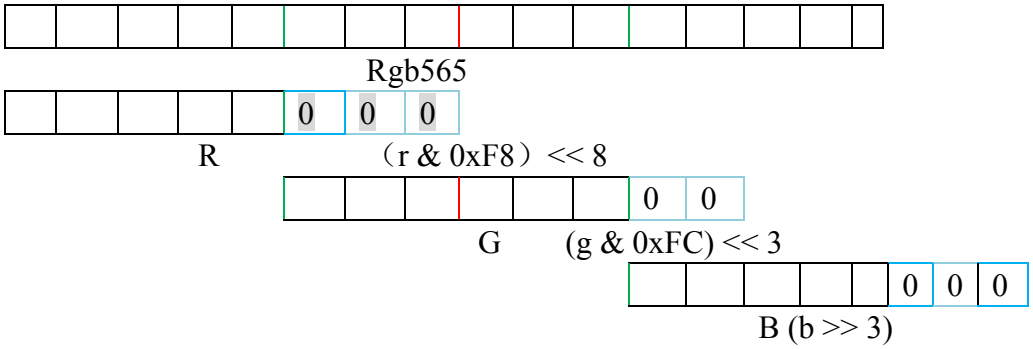
2.运算符

C 语言中有 42 个运算符

赋值	=	+=	-=	/=	%=	*=		6
算术	+	-	*	/	%	++	--	7
关系	>	<	<=	=>	==	!=		6
逻辑	&&	||	!					3
	*(取值)	&(取址)						2
位运算	&	|	^	~	<<	>>		5
	&=	|=	^=		<<=	=>>		5
三元	? :							
逗号	,		sizeof					

注：a << n　相当于　a*2^n
　　~1 为 -2　~2 为 -3　~3　为 -4
1 原码 0000 0001　　取反后　1111 1110
-2 原码 1000 0010　反码 1111 1101　补码 1111 1110
在计算机中负数都以补码形式存在，所以~1 为-2

附：rgb565



Rgb565　=　((r & 0xF8) << 8) | ((g & 0xFC) << 3) | (b >> 3);

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
int main(int argc, char *argv[])
{
    uint8_t r = 0xFF;
    uint8_t g = 0xCC;
    uint8_t b = 0x99;
    uint16_t rgb = 0;

    rgb = ((r & 0xF8) << 8) |
          ((g & 0xFC) << 3) |
          (b >> 3);

    fprintf(stdout, "%4X\n", rgb);

    return EXIT_SUCCESS;
}
```

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

typedef union{
    uint16_t rgb;
    struct {
        uint16_t b : 5;
        uint16_t g : 6;
        uint16_t r : 5;
    }colors;
}Color;
int main(int argc, char *argv[])
{
    uint8_t r = 0xFF;
    uint8_t g = 0xCC;
    uint8_t b = 0x99;
    Color color;

    color.colors.r = r >> 3;
    color.colors.g = g >> 2;
    color.colors.b = b >> 3;

    fprintf(stdout, "%04X\n", color.rgb);

    return EXIT_SUCCESS;
}
```

十一、MAKE 工具

1.编译规则文件

规则文件 { GUNMakefile / makefile / Makefile（一般） } 高 → 低 优先级

一般编写使用 Makefile 命名，其他两个多用于在一个项目中存在检测

2.规则文件由规则构成

语法：

目标 ：[依赖列表]
tab 命令
。
。
。
（多行或 0 行）

规则：

i. 先递归、 后迭代
（检测依赖关系）（根据目标与依赖的时间关系，决定是否重新执行规则里的命令）

ii. 在规则文件中，第一个目标为默认目标，如果在下达 make 命令时，不指定目标，那就使用默认目标，为递归结点。
也可以在 make 时，指定一个目标

iii. 没有依赖的目标为伪目标，最好在此目标之上先做说明。
如：PHONY：clean
clean：
rm –f circle.o area.o main.o

3.变量

I. 自定义变量
赋值：变量名=值列表（空格分隔） #多行加转义符
引用：$(变量名)
注：
= 引用时才展开变量值
:= 赋值时直接展开变量（替换后不变）
+= 连接形成列表
?= 如果原来变量没有赋值时，才有效

II. 预定义变量

$@	本条规则的目标	
$^	本条规则所有依赖	
$<	本条规则第一个依赖	
CC	默认的 C 编译器	
CFLANGS	指定编译阶段的参数	

III. Make 的推断

推断简单的编译命令

多目标生成时，可将多个目标作为依赖关系，重新设定一个目标

4.自动生成 Makefile

a. qmake -project #使用 QT 自带工具生成一个 round.pro 项目文件

b. qmake #通过 xxx.pro 项目文件生成 Makefile 编译规则文件

c. vi Makefile 文件，对其根据需要修改
- 编译参数
- 指定编译器
- 头文件指定
- 库名指定

d. make #编译