Linux基础

## 系统启动流程

POST（自检）

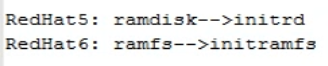
--> BIOS(boot sequence 启动次序设定，外部IO设备中的哪一个放了操作系统，找到设备的MBR)

--> MBR(bootloader ,446字节，bootloader 加载内核，现在多使用GPT格式支持大容量磁盘，获得控制权后，加载选定的内核，将控制权给内核)

--> grub(bootloader的一种，/boot/grub/grub.conf 加载内核)

--> kernel(内核展开，探测硬件并装载驱动, 到根分区上加载根文件系统，/bin /sbin必须在根分区)

--> initrd （initramdis,内存文件系统模拟根文件系统(虚根)，然后进行根的切换)-->rootfs



--> /sbin/init（根据/etc/inittab配置文件,上帝开始造人，用户空间的程序）

kernel 和 initrd在同一分区 被bootloader加载进内存，kernel展开后,需要使用到initrd

内核功能：

进程管理

资源监控

内存管理

文件系统

网络功能

硬件驱动

安全机制

内核设计风格

核心：动态加载各个外围模块

内核： **/lib/modules/内核版本号命名的目录**  **各种外围模块 /lib/modules/2.6.32**

**内核名称： vmlinuz-2.6.32**

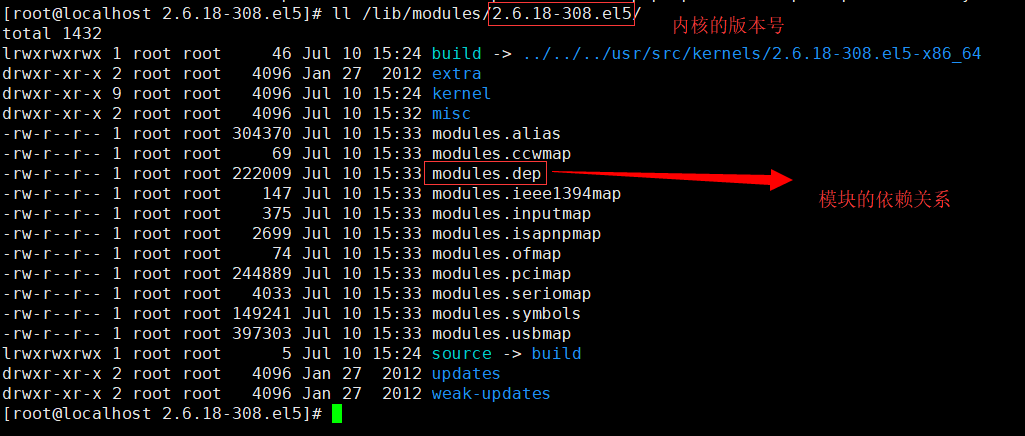
单核：内核功能放在一起，一个功能故障，整个系统就故障了。

ko : (kernel object) 内核的对象模块

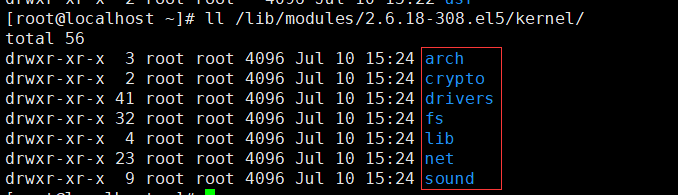
so : (system object) 系统库文件

微内核：内核各个功能独立做成一个子系统，由内核统一调度。显然这更科学合理高效，但是各个子系统的功能协调复杂了。如windows，Solaris

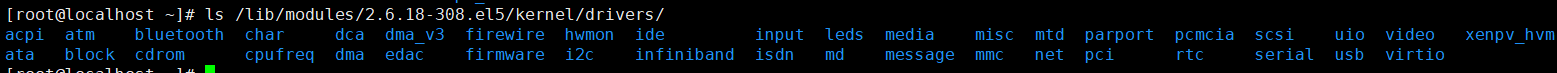
linux 中和了上面两种内核设计风格



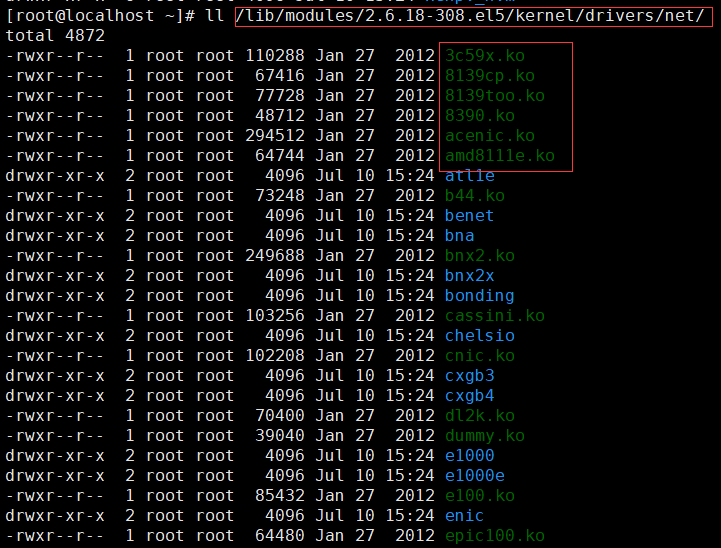
内核



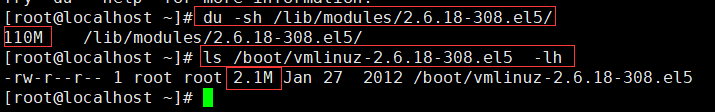
驱动



内核需要net功能的时候，内核就装载进来就可以使用

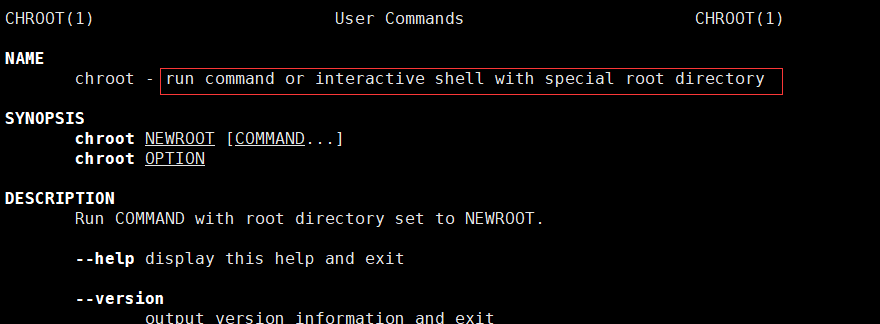


内核大小



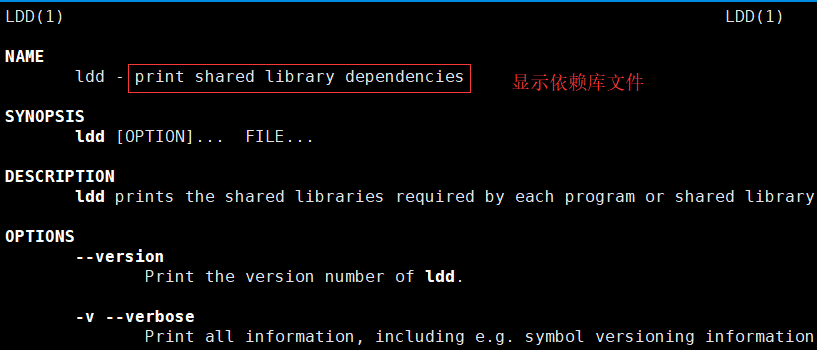
## chroot

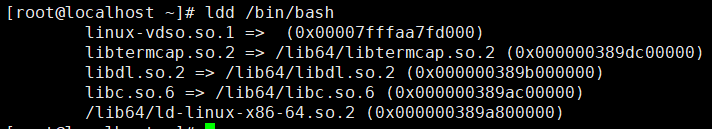
切换根



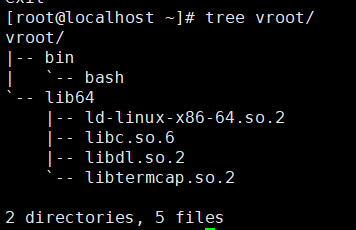
## ldd

显示二进制文件所依赖的共享库文件





自己创建一个根,进行根切换





问题提出： 内核加载驱动，需要根文件系统的支持（驱动存放在文件系统上），但是这个时候，根文件系统还没有工作就是说驱动都没有加载进来，磁盘是不可能工作寻找到驱动文件数据。

解决办法： 由内核负责创建一个临时的根文件系统，我们可以称之为虚根，这个过程叫ramdisk(把内存当作磁盘) ramfs（内存文件系统） 。在这个虚根里面，对硬件（如磁盘）做映射， /proc /dev /sys 。然后等待硬件（如磁盘）映射完毕后，真正的根文件系统启动了，磁盘也就可以真正的工作了。这时，将 /proc /sys /dev

搬到真正的根文件系统里，这个过程就是根切换。 有了根切换，就可以解决上面的问题。

rh5 ramdisk --> initrd

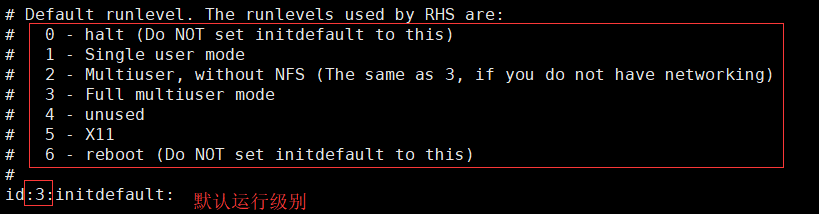
rh6 ramfs--> initramfs

--> /sbin/init

## 运行级别

就是启动的服务不同，就像windows中的安全模式（之加载基本的驱动程序），VGA模式

6个级别



## bootloader

MBR包含bootloader ，但是MBR属于磁盘，bootloader是由操作系统提供的。

微软是这样干的，window的bootloader只允许引导自己，不引导别人，而且win8以后，非但不允许引导别人，反而还将MBR给锁定了。不允许并存。美其名曰防止病毒感染MBR

然而Linux就很开放，可以引导别人，

Linux bootloader有2种

**LILO**： Linux loader 不能引导1024柱面以后的分区。不支持大硬盘，早期使用,

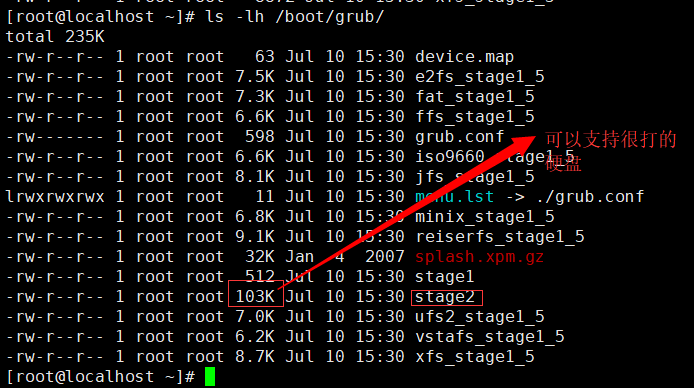
多用于嵌入式

**GRUB**: Grand unified Bootloadre

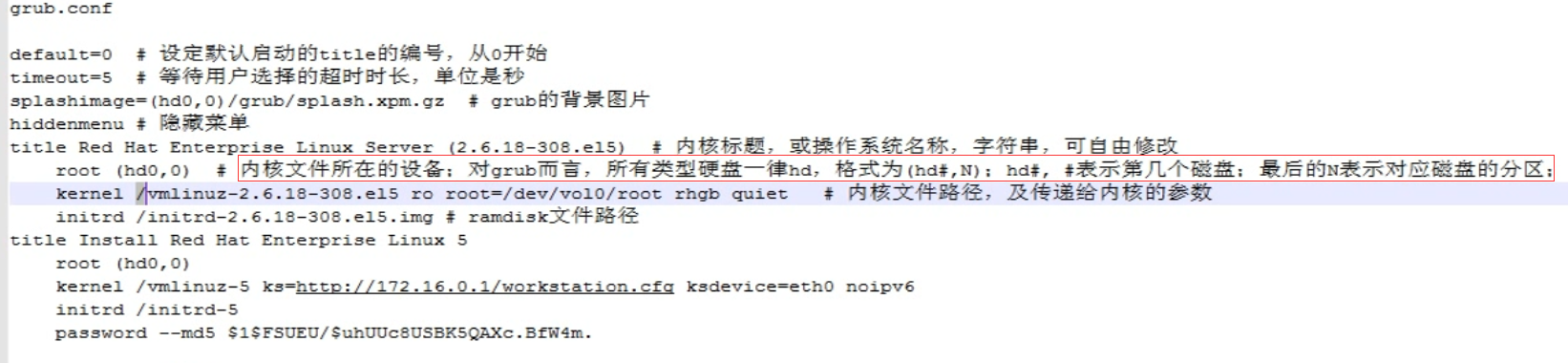
stage1 ： MBR 引导 stage2

stage1.5 : 识别不同类别的文件系统

stage2 ： /boot/grub 引导操作系统



stage2 103k





**root**: 内核文件所在的设备，对grub而言，所有类型的硬盘一律hd，格式为（hd#,N）#标识第几个硬盘，最后的N标识对应磁盘分区。

**kernel** : ro root=LABEL=/ rhgb quiet 内核文件路径，以及传递给内核的参数，在proc目录下，可以查看到



**initrd** : ramdisk ,虚根

## 终端启动时，修改grub运行级别

/boot/grub/grub.conf 配置

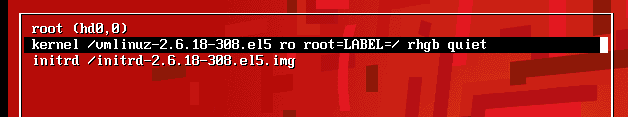


e 键

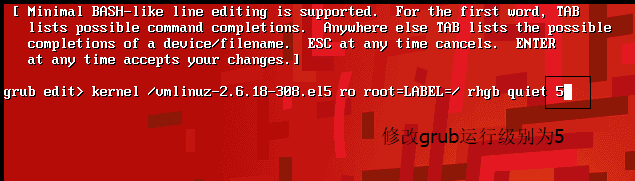


↑ ↓ 选择，e 编辑 d删除 o 新行 b 启动

敲↓键



敲e键



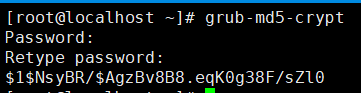
这里修改运行级别1，然后敲enter键 ，修改完以后敲b键



单用户模式不需要密码就可以进入所以单用户模式很危险，就算我不知道密码，我可以通过启动的时候进入单用户模式，修改文件。为了解决这个问题，在grub.conf文件中加入密码

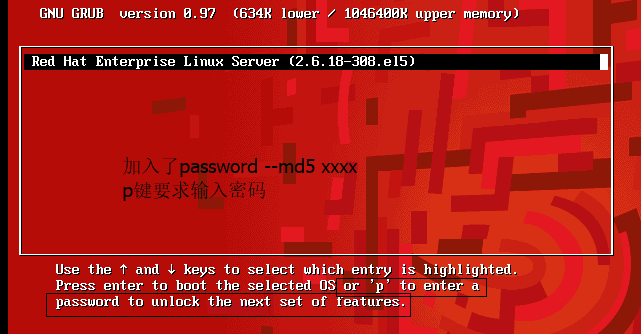
这里使用命令 **grub-md5-crypt**

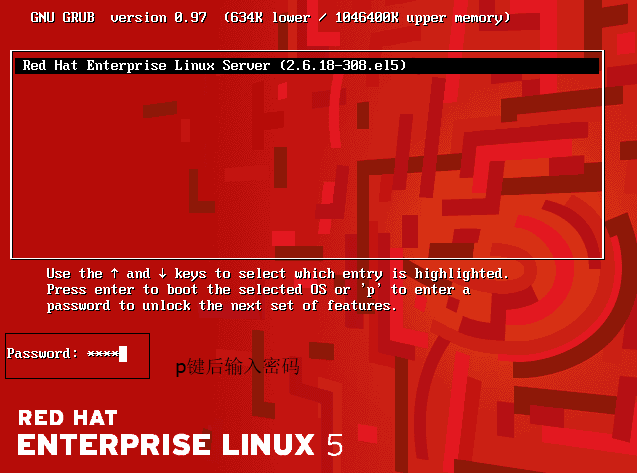
对’root’加密



配置grub.conf







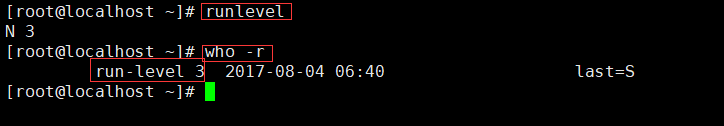
输入密码后就可以继续后面的操作，这是防止单用户模式后，所有人都可以去系统上搞一下。

使用init 3 切回多用户运行级别模式

## runlevel

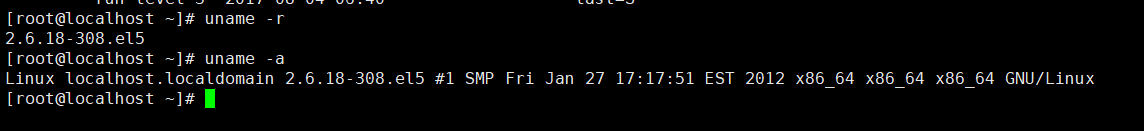
查看当前系统运行级别





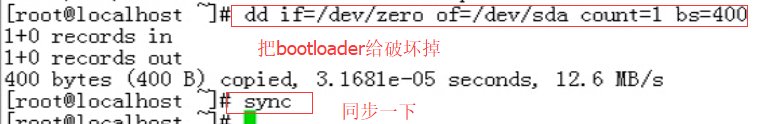
## uname -r

显示内核release

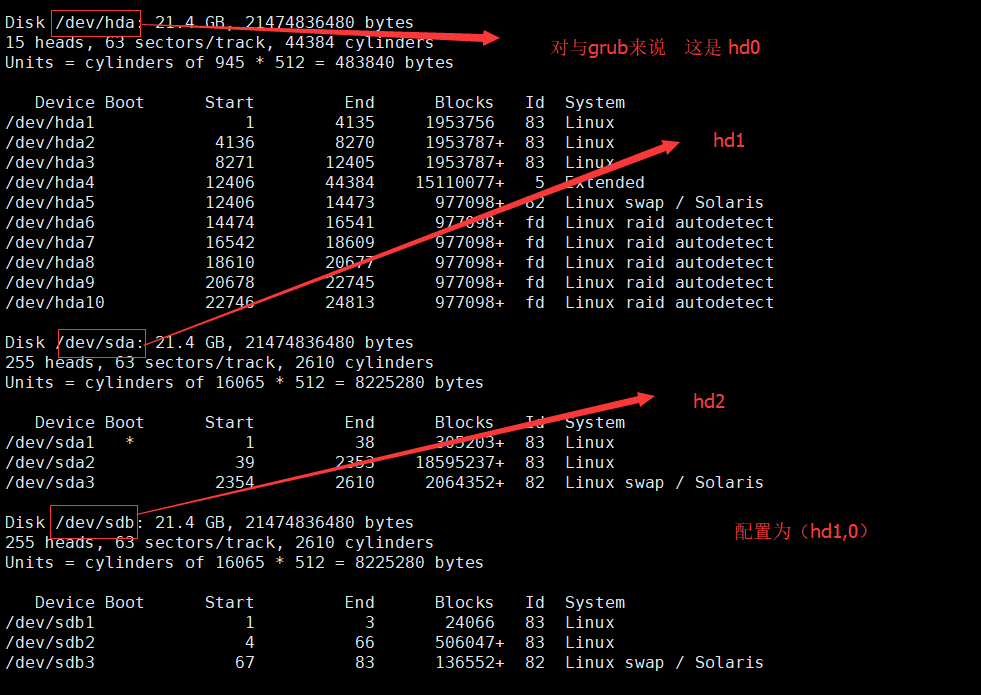


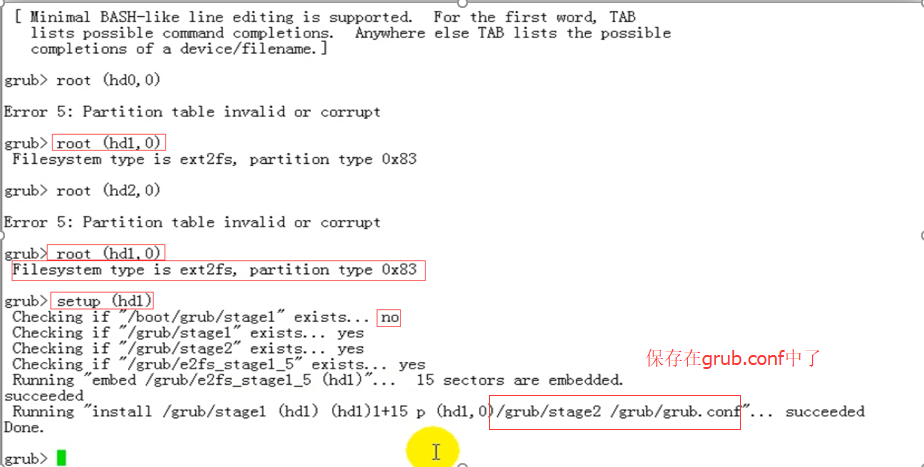
## 安装修复grub

第一种方式:



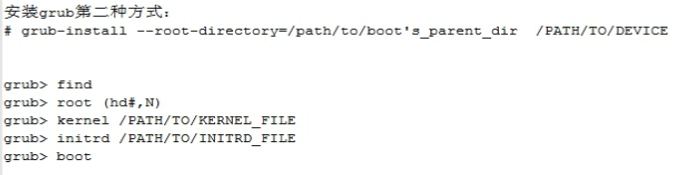
grub命令





第二种方式：

grub-install --root-directory=/xxx/xxx(boot的父目录) devce



## kernel 初始化过程

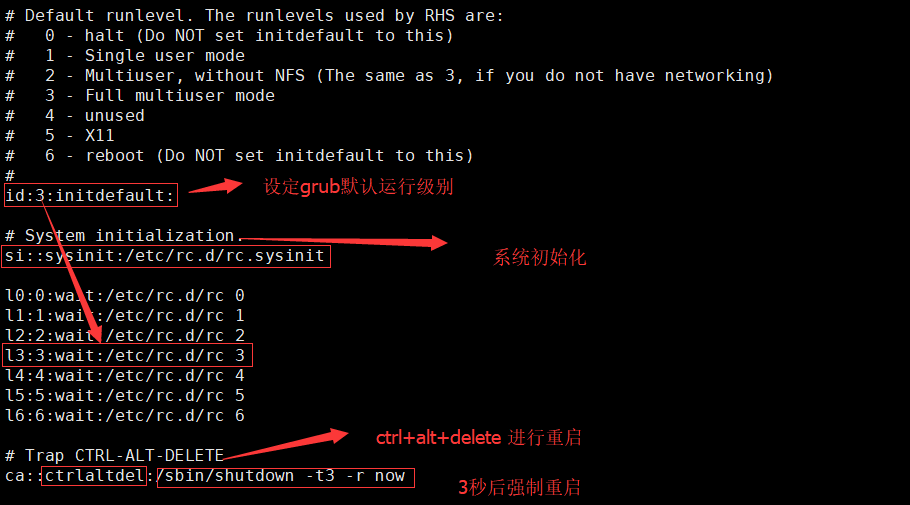
1. 设备探测
2. 驱动初始化（可能会从initrd（initramfs）文件中装载驱动模块）
3. 以只读方式挂载根文件系统
4. 装载第一个进程init（PID = 1）

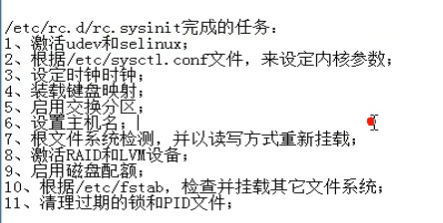
## init程序

红帽5 /sbin/init (/etc/inittab) 古老

红帽6.3 ubantu的 upstart使用event driven驱动d-bus启动并行启动进程，比传统的要快，稳定

systemd : 并行启动更快





**si : : sysinit:/etc/rc.d/rc.sysinit**

id : runlevels : action : process

id : 标识符

runlevels: 运行级别

action : 在什么情况下执行

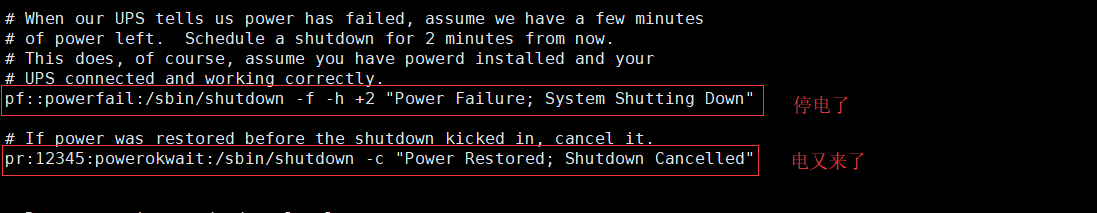
process: 要运行的程序

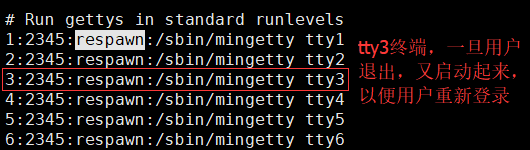
initdefault: 设定默认的运行级别

sysinit: 系统初始化

wait: 等待级别切换此级别时

respawn ： 一旦程序终止，会重新启动



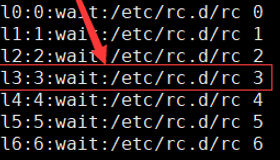


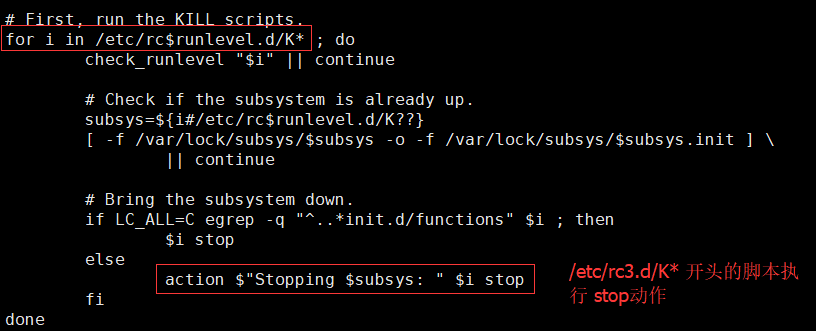
如果是5级别，就要启动图形界面



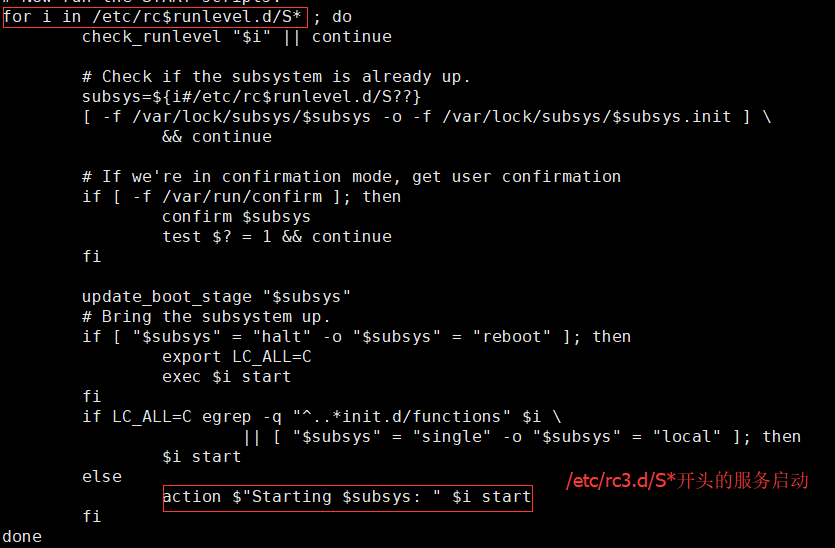
/etc/rc.d/rc

这里传入运行级别

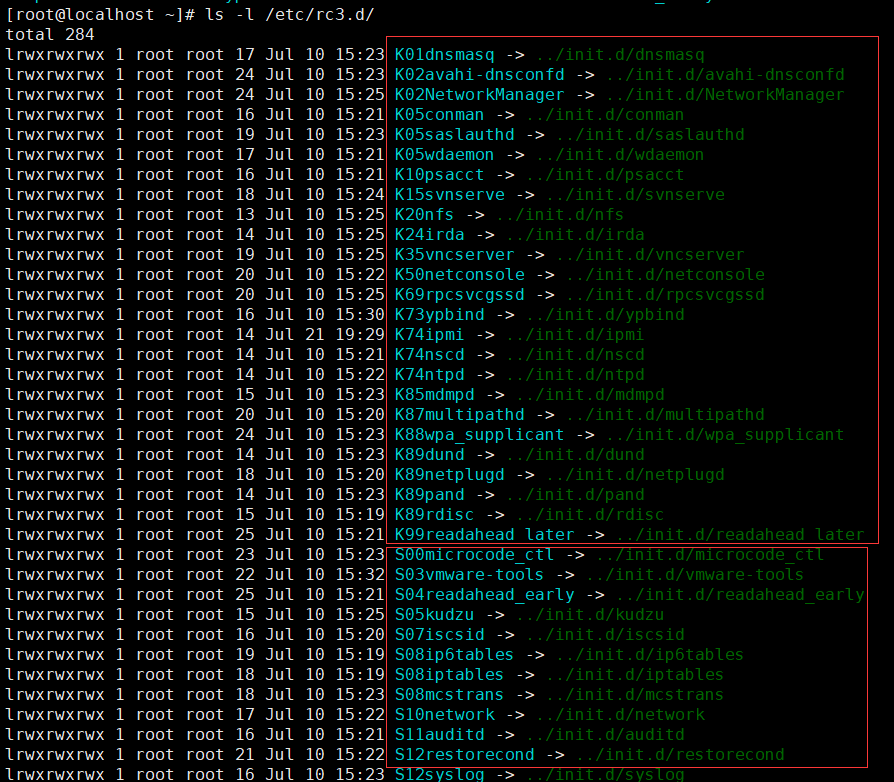




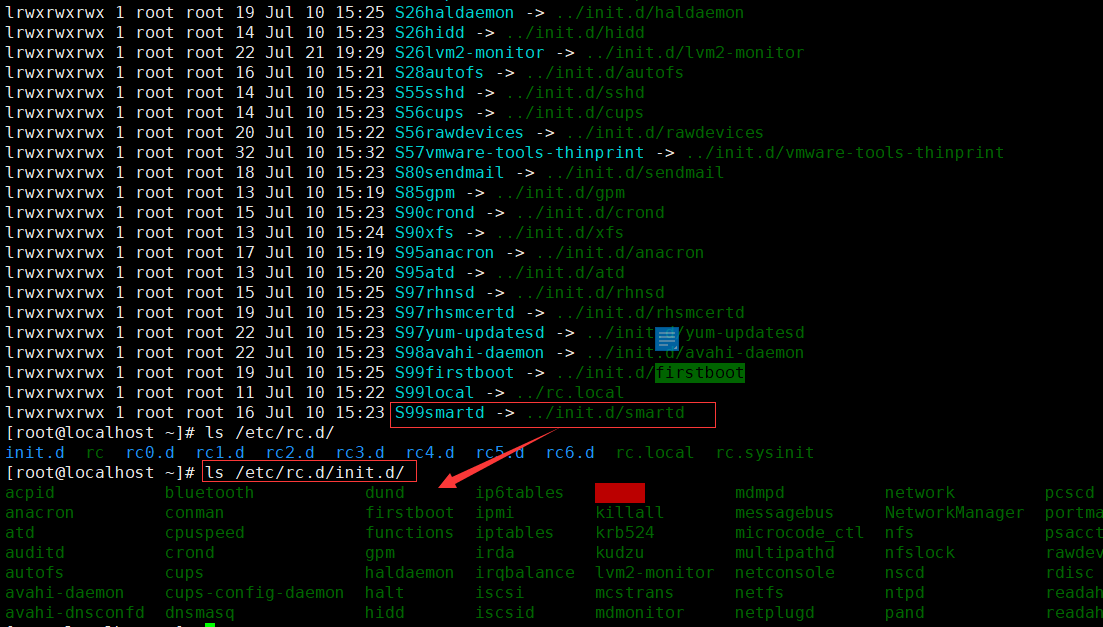
/etc/rc.d/rc 脚本文件



/etc/rc(0-6).d 蓝色标识链接文件，rc(0-6)这些文件中，因为运行级别不同，所以每个文件所启动的服务，停止的服务是相同的。



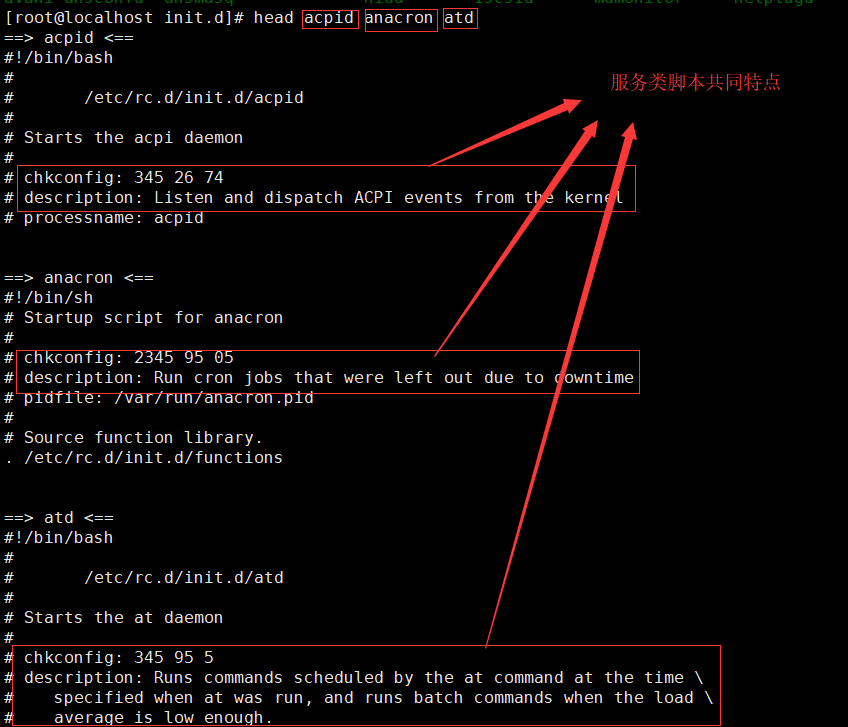
../init.d/\* 放在/etc/rc.d/init.d目录下 这些文件就是系统初始化的需要的文件



/etc/rc.d/init.d /etc/init 目录下的脚本是服务类的脚本

他们遵循 sysV风格 最起码要接收4个参数

start | stop | restart | status | reload | configtest



**chkconfig**: 345 26 74 -->分别表示 runlevels SS KK

runlevels： - 表示没有级别，默认为S\*开头的链接

runlevels 表示默认创建为S\*开头的链接，S后面的启动的优先级为SS所表示的数字

除此之外的级别默认为创建K\*开头的链接,K后面关闭优先次序为KK所表示的数字。

**description**: 说明此脚本的简单功能 \, 续行

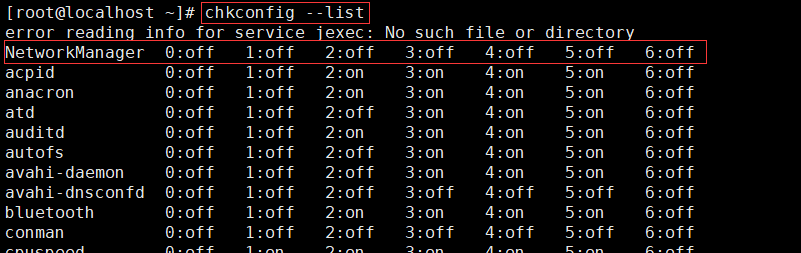
虽然都是#注释，但是chkconfig会去解析

## chkconfig

chkconfig命令主要用来更新（启动或停止）和查询系统服务的运行级信息。谨记chkconfig不是立即自动禁止或激活一个服务，它只是简单的改变了符号连接。

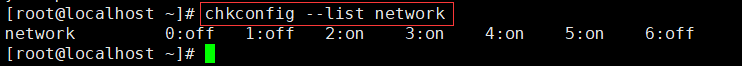
### --list

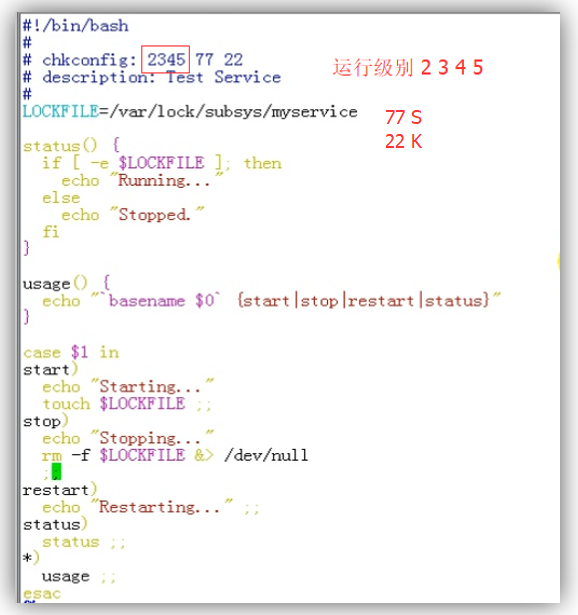
查看所有独立守护服务



网络功能所在运行级别上的状态是否开启和关闭

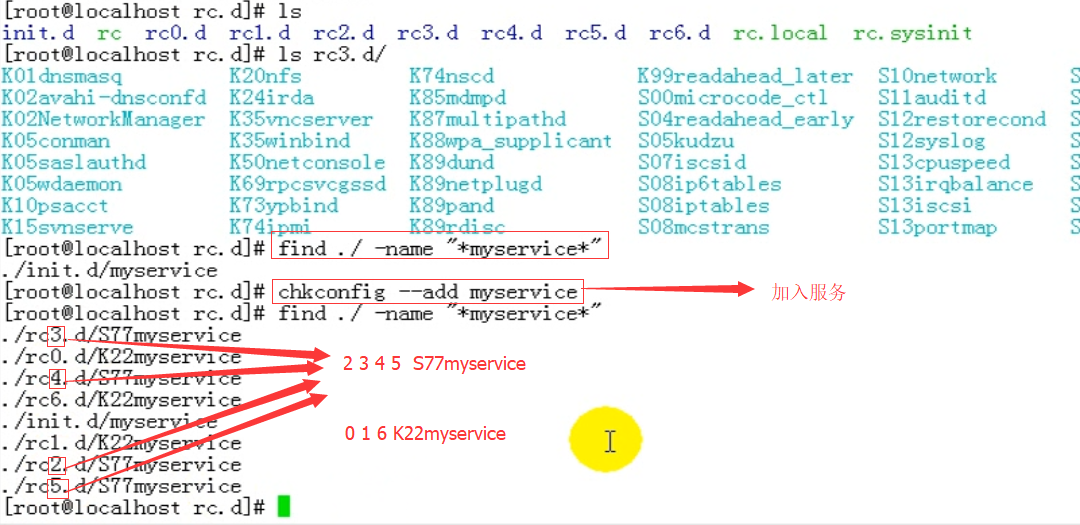
**chkconfig --list network**





### --add

添加服务



加入后，就受chkconfig命令控制，我们称为**独立守护进程或者独立守护服务。在运行级别自启动和自关闭。**

### --del

删除服务

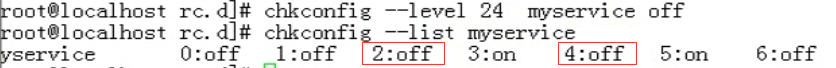


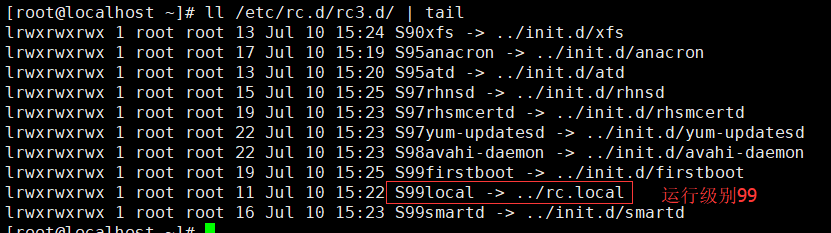
### --level

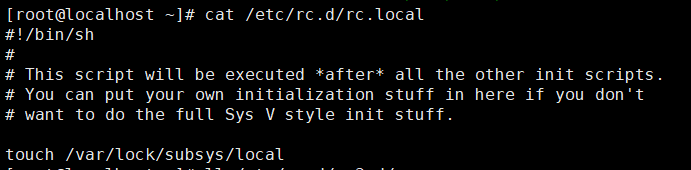
不用修改myservice ,就可以修改级别的开启和关闭

chkconfig [--level 24] myservice off

如果省略级别指定，默认为2345







rc.lcoal 文件中定义系统启动**最后一个**（S99）要执行的命令脚本

## /etc/inittab的任务

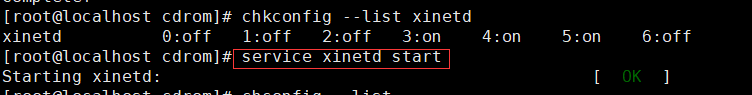
1. 设定默认运行级别 （0-6）
2. 运行系统初始化脚本
3. 运行指定运行级别对应目录下的脚本，K\* stop S\* start
4. 设定ctrl+alt+del 的动作
5. 定义UPS电源在电源故障/恢复时执行的操作
6. 定义启动的虚拟终端（2345级别）
7. 定义启动图形终端（只在5级别）

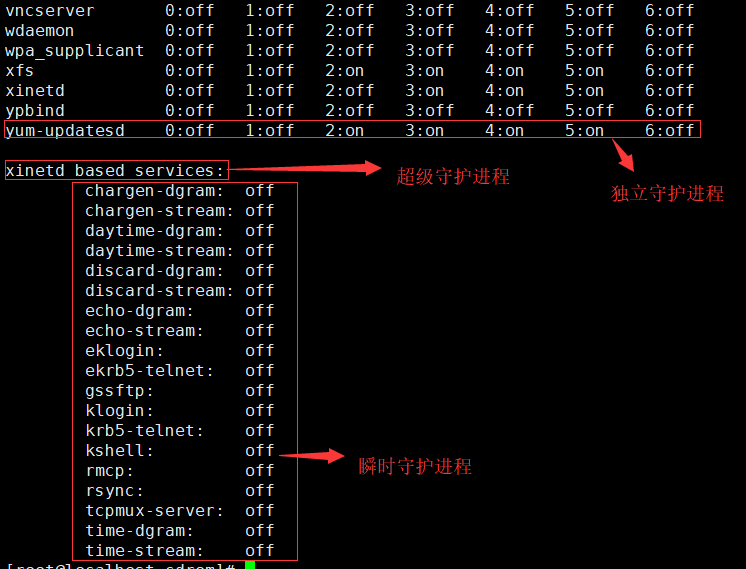
守护进程类型：

独立守护进程，

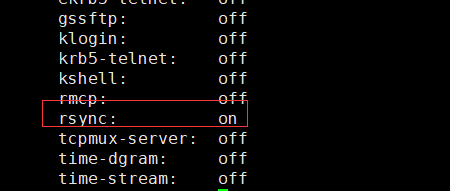
xinitd 超级守护进程，代理人

瞬时守护进程，不需要关联至运行级别









## 内核

内核： /boot/vmlinuz-version

内核模块（ko）：/lib/modules/version

装载模块：

insmod

modprobe

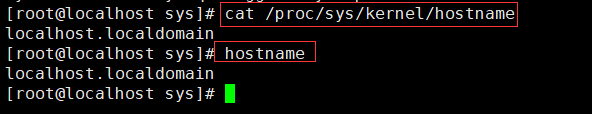
用户空间访问，监控内核的方式

/proc /sys 伪文件系统，多是可读写的。

设定内核参数值

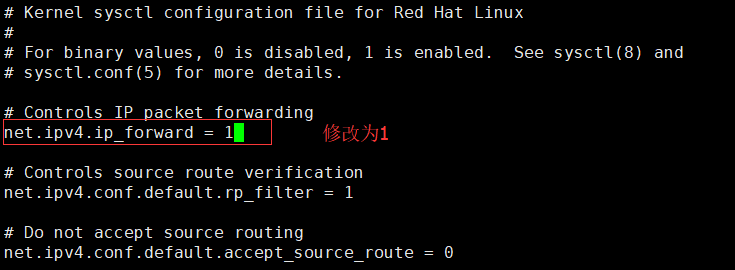
1. echo value > /proc/sys/文件
2. sysctl -w kernel.hostname=localhost.hc

能立即生效，但无法永久有效



如果要永久有效，永久有效，但不能立即生效

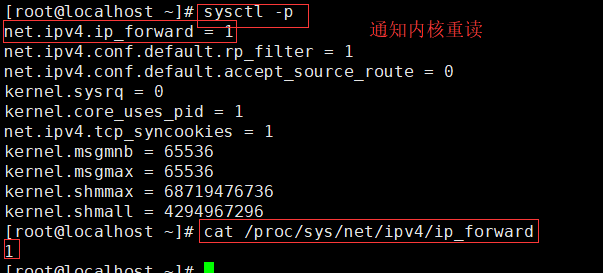
/etc/sysctl.con文件中



没有立即生效



sysctl -p 通知内核重读，立即生效。



显示内核所有参数和值

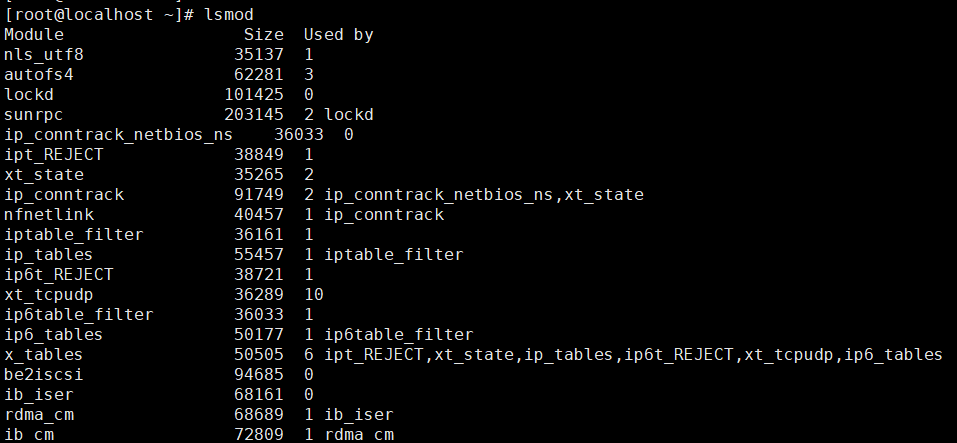




内核模块管理

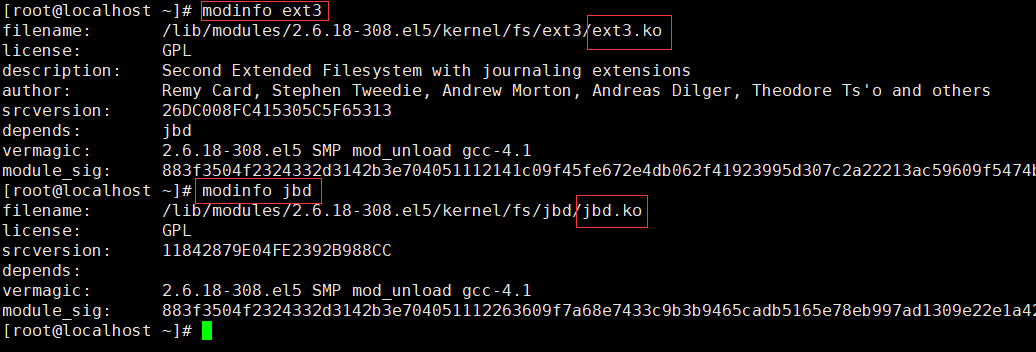
## lsmod

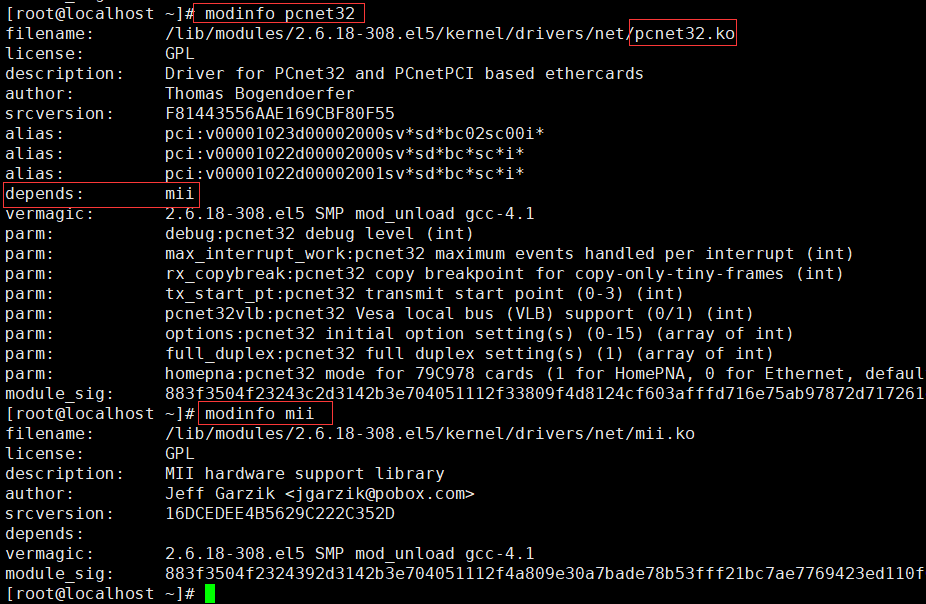
lsmod 查看内核模块列表



## modinfo

modinfo xxx 查看某模块的详细信息





## modprobe

modeprobe xxx **:** 装载某模块

modeprobe -r xxx : 卸载某模块

## insmod

insmod /xxx/xxx.ko

装载模块 必须要只当路径



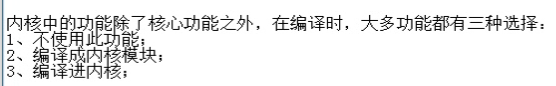
## rmmod

卸载模块

rmmod xxx

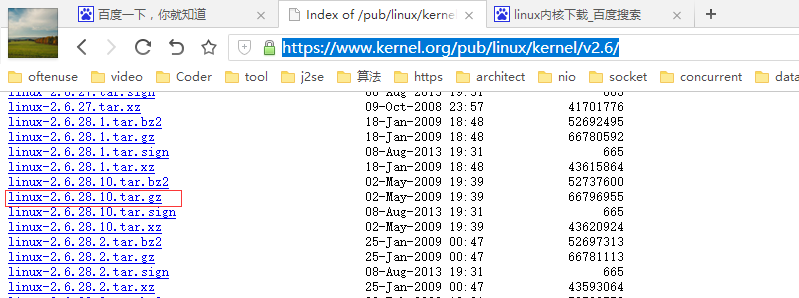
## depmod

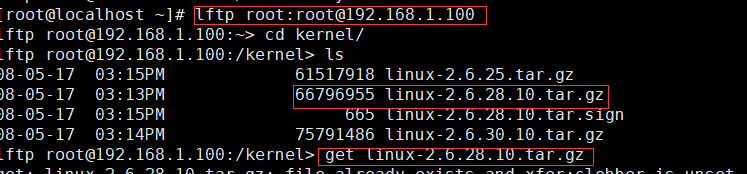
depmod /xxx/xxx



## 如何手动编译和安装自己平台内核

1. 到官网下载内核

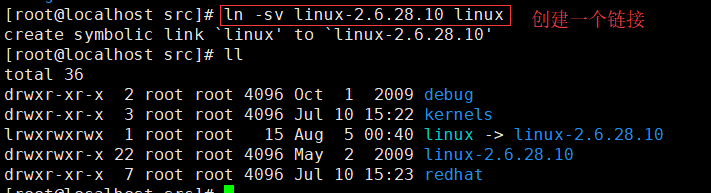




1. 解压到 /usr/src



1. 创建链接

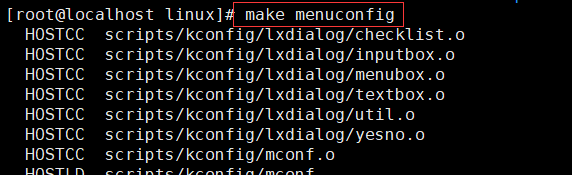


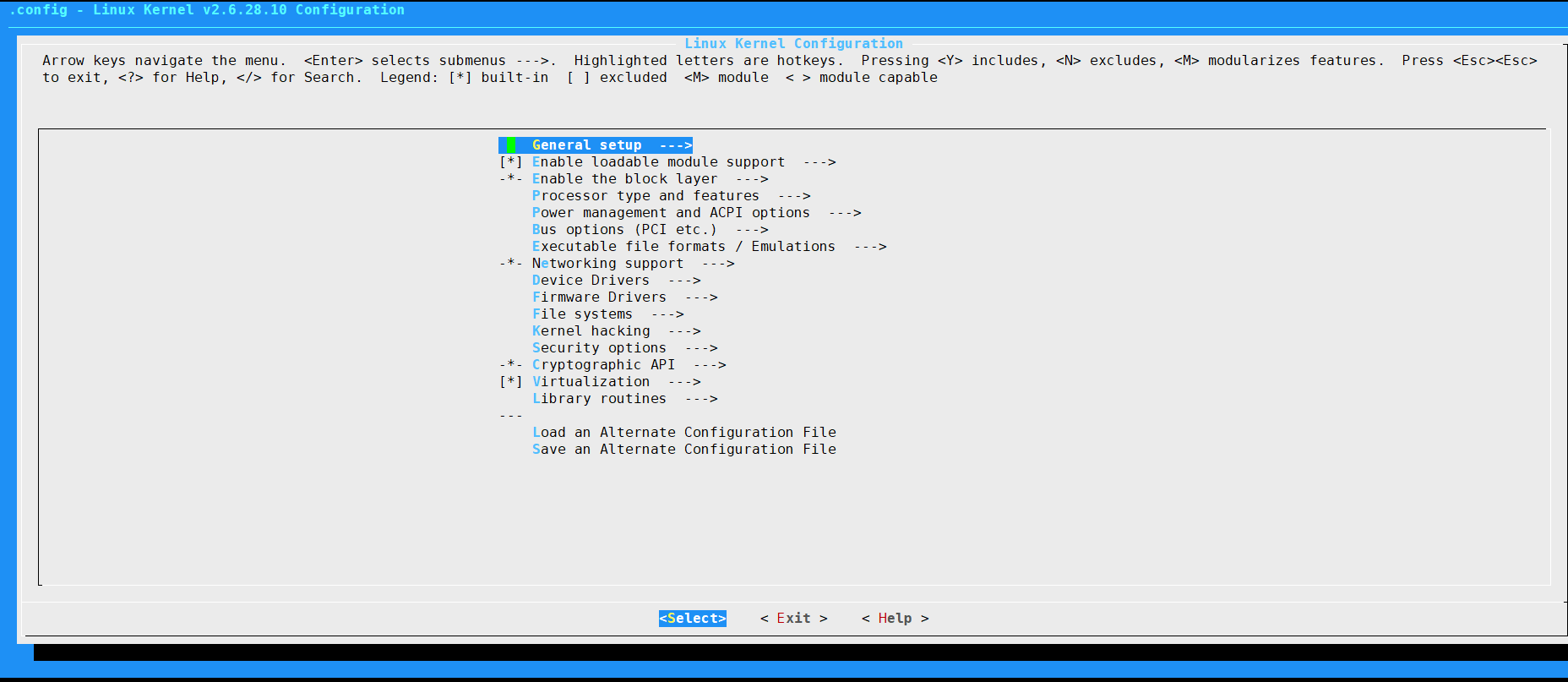
1. make

make gconfig: Gnome桌面环境使用，需要安装图形开发库 GNOME Software Development

make kconfig: KDE桌面环境使用，需要安装图形开发库

make menuconfig 简单的

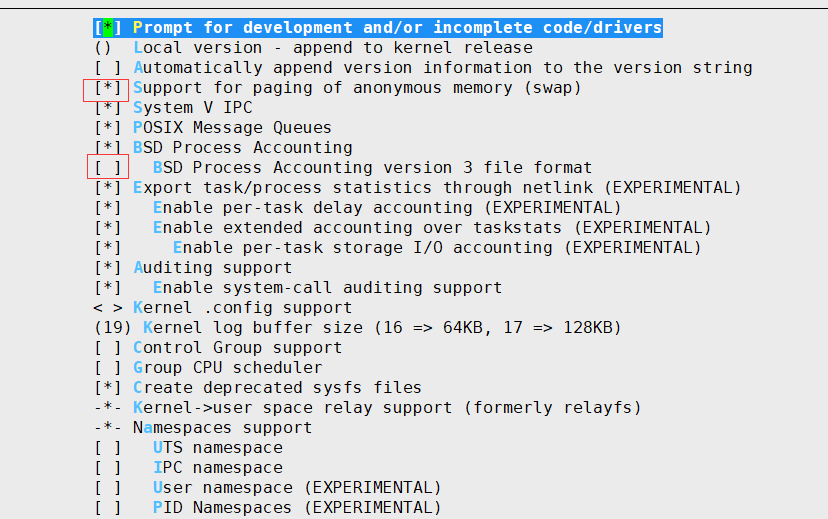




\* 表示做进内核

M 表示做成模块

[ ] 表示不启用此功能



如果自己配置的话，对内核必须非常熟悉

1. 我们以当前系统编译好的内核配置文件为模板来自定义自己的内核



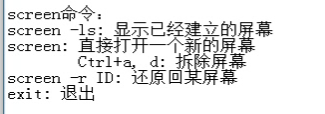
1. make
2. make moduls\_instal
3. make install

## screen

解决远程链接

yum install screen





二次编译是清理,清理前，如果有需要，先备份.config文件

make clean 清理之前编译好的二进制模块

make mrproper 清理残留包括.config文件

## 组装系统

grub--> kernel --> initrd-->rootfs(/sbin/init,/bin/bash)

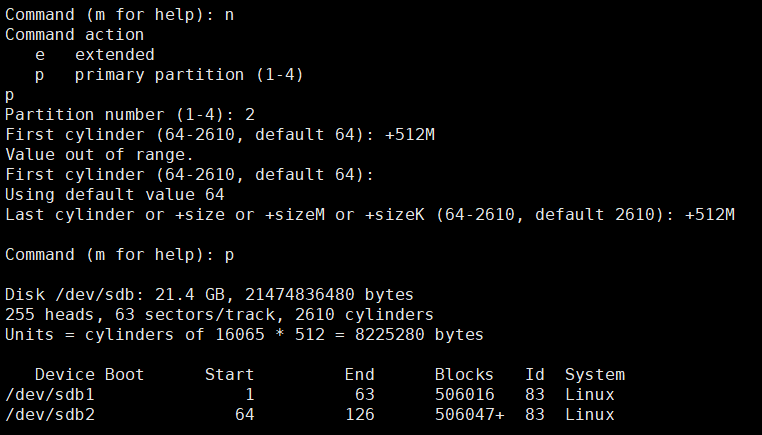
/mnt/boot

/mnt/sysroot

bin,sbin,etc,var,tmp,proc,sys,dev,lib,usr,home

1. 准备一块硬盘,并高级格式化进行挂载

dd if=/dev/zero of=/dev/hda bs=512 count=1

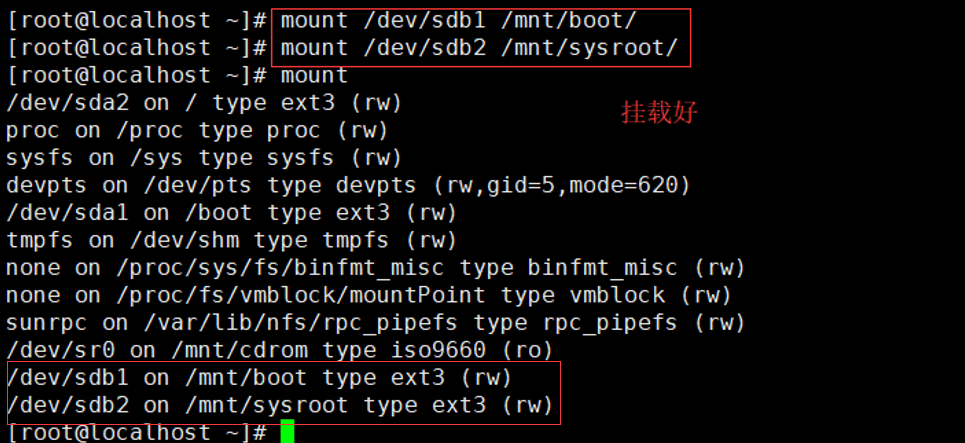


格式化



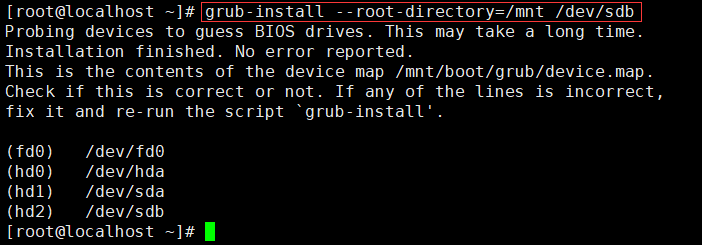


挂载



1. 安装grub

grub-install --root-directory=/mnt /dev/hda



grub 安装成功



1. 复制一个内核



1. initrd文件

**mkinitrd** initrd 文件路径 内核版本

mkinitrd /boot/initrd-`uname -r`.img `uname -r`

这里不能这样用，因为这是在当前系统上,所以这里只有手动展开**initrd-2.6.18-308.el5.img**

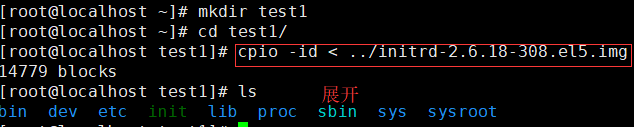




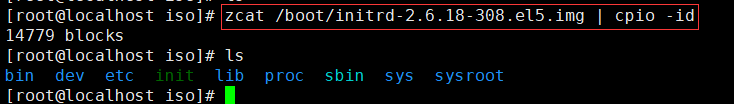








也可以这样干，一个命令搞定



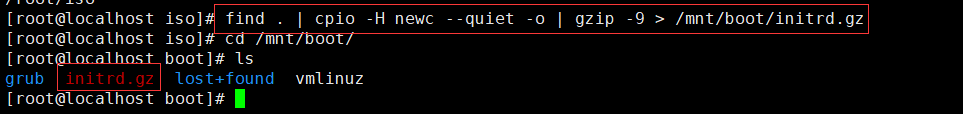
vi init



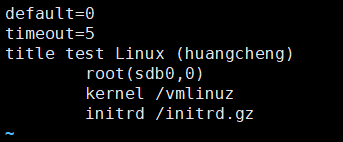
对以下文件进行打包归档



find . | cpio -H newc --quiet -o | gzip -9 > /mnt/boot/initrd.gz

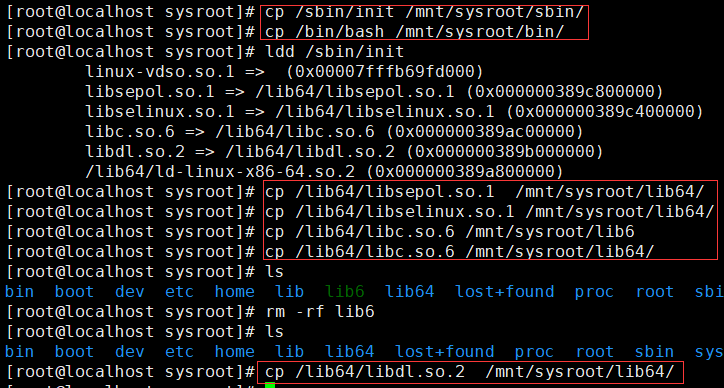


1. 编辑 /mnt/boot/grub/grub.conf文件

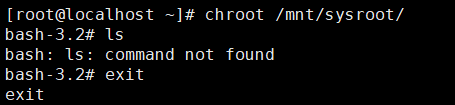




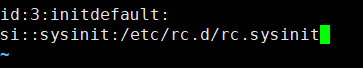
1. 拷贝文件



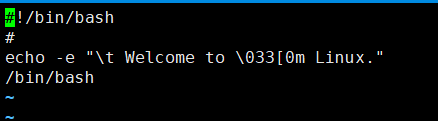
可以切换 chroot



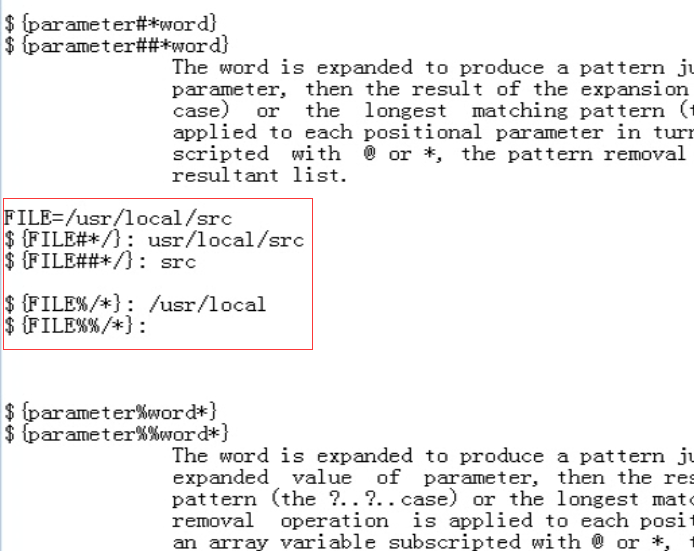
1. vi /mnt/sysroot/etc/inittab



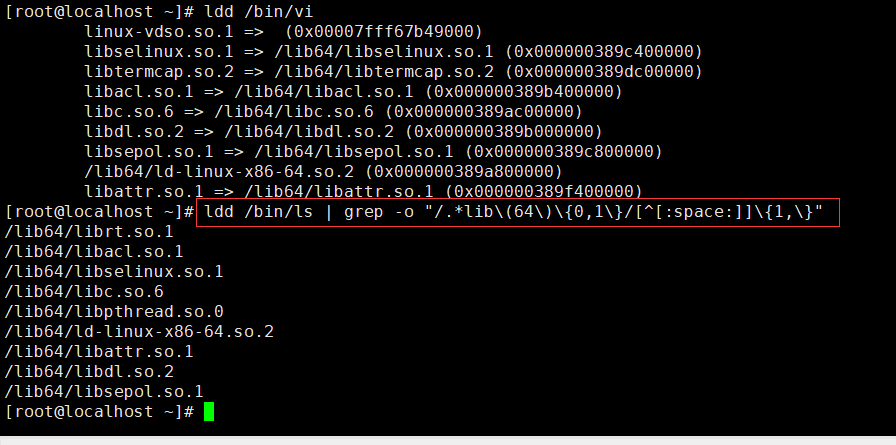
1. vi etc/rc.d/rc.sysinit



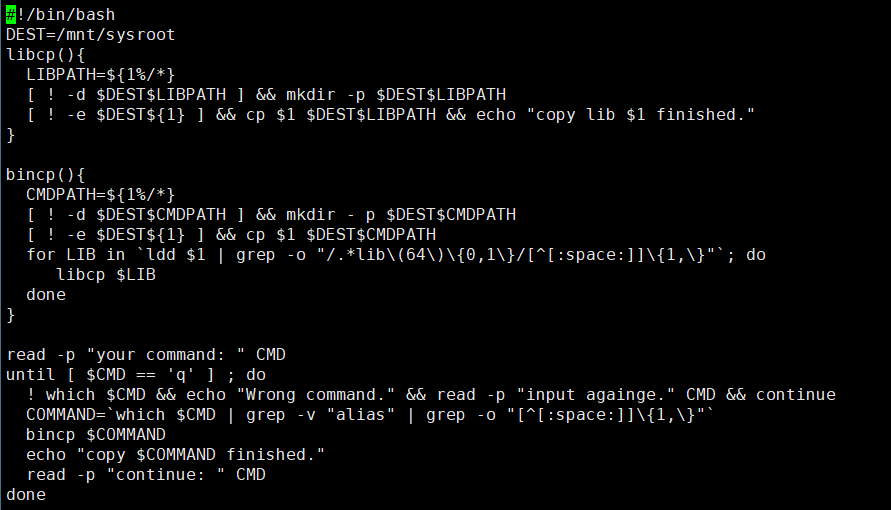
1. chmod +x etc/rc.d/rc.sysinit
2. 重建虚拟机 使用第一步定义的硬盘就ok
3. cp lib 脚本



ldd /bin/ls | grep -o "/.\*lib\(64\)\{0,1\}/[^[:space:]]\{1,\}"



拷贝命令的依赖库文件



#!/bin/bash

DEST=/mnt/sysroot

libcp(){

LIBPATH=${1%/\*}

[ ! -d $DEST$LIBPATH ] && mkdir -p $DEST$LIBPATH

[ ! -e $DEST${1} ] && cp $1 $DEST$LIBPATH && echo "copy lib $1 finished."

}

bincp(){

CMDPATH=${1%/\*}

[ ! -d $DEST$CMDPATH ] && mkdir - p $DEST$CMDPATH

[ ! -e $DEST${1} ] && cp $1 $DEST$CMDPATH

for LIB in `ldd $1 | grep -o "/.\*lib\(64\)\{0,1\}/[^[:space:]]\{1,\}"`; do

libcp $LIB

done

}

read -p "your command: " CMD

until [ $CMD == 'q' ] ; do

! which $CMD && echo "Wrong command." && read -p "input againge." CMD && continue

COMMAND=`which $CMD | grep -v "alias" | grep -o "[^[:space:]]\{1,\}"`

bincp $COMMAND

echo "copy $COMMAND finished."

read -p "continue: " CMD

done