一、Linux 系统用户接口

1、实验说明

学习在 Linux 字符界面下，使用常用的 Linux 命令来进行系统操作，如文件操作、目录操作、文件系统操作、设备操作、进程状态查询、进程控制、查看系统信息、字符串过滤及搜索等。

2、Linux 常用命令及常用选项

1）系统信息及在线帮助

1、**uname**：显示系统信息，如主机名、内核版本、硬件平台、内核名称等。

- 命令格式：uname［选项］...
- 常用选项：

  -a：显示系统所有信息

  -n：显示系统主机名

  -s：显示内核名称

  -r：显示内核版本信息

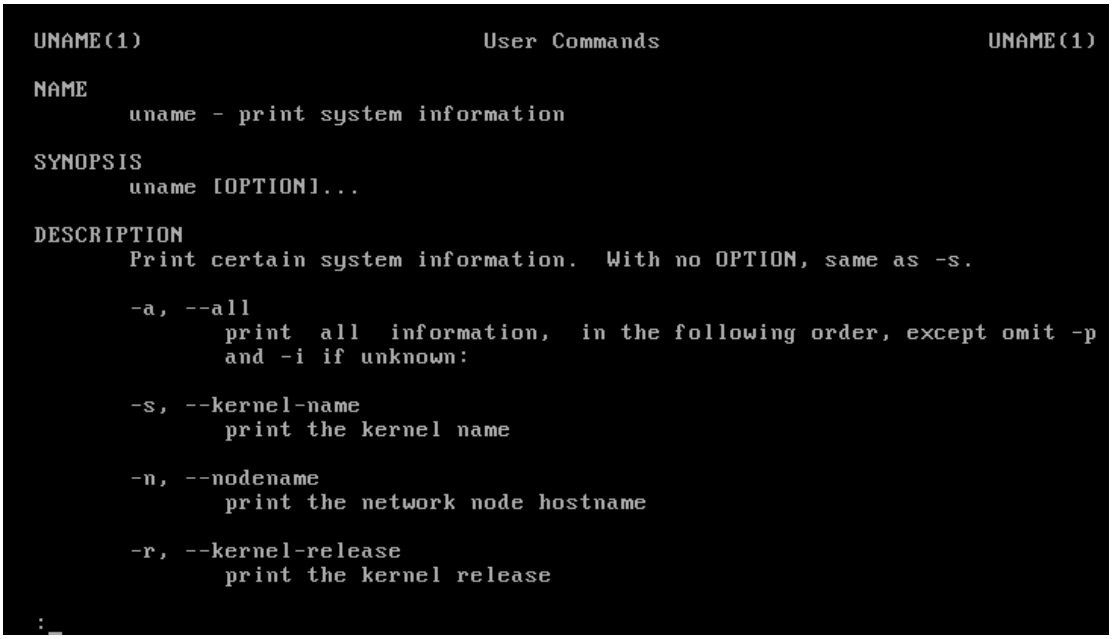
  -i：显示硬件平台信息
- 示例：

```
[root@localhost ~]# uname -a
Linux localhost.localdomain 2.6.18-164.el5xen #1 SMP Thu Sep 3 04:47:32 EDT 2009
 i686 i686 i386 GNU/Linux
[root@localhost ~]# uname -n
localhost.localdomain
[root@localhost ~]# uname -r
2.6.18-164.el5xen
[root@localhost ~]# uname -s
Linux
[root@localhost ~]# uname -i
i386
```

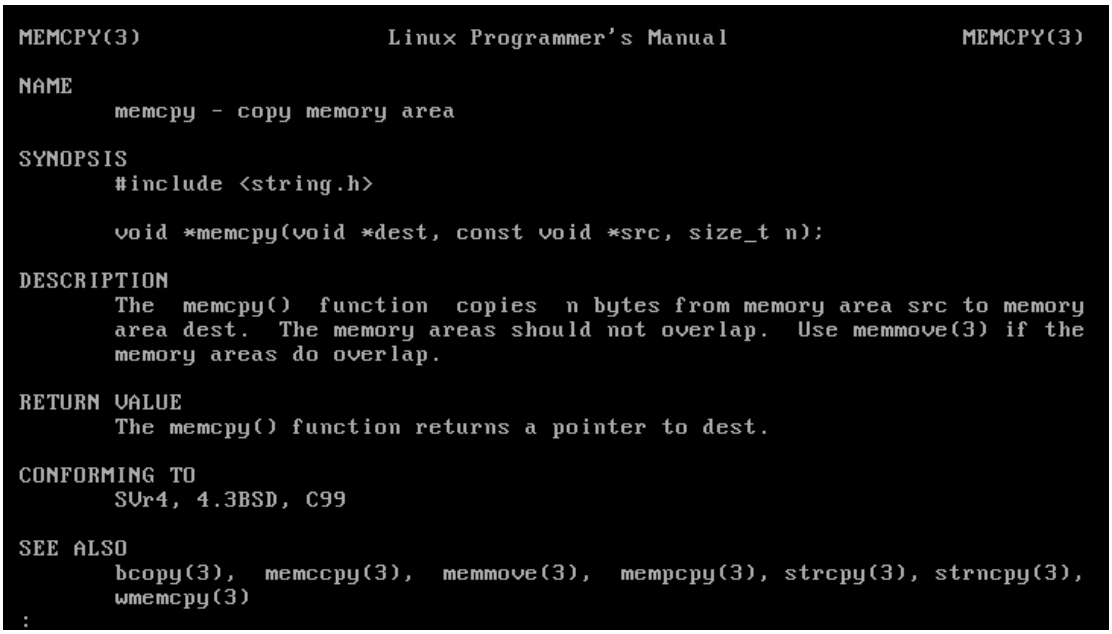
2、**man**：Linux 在线帮助，在弹出的界面中，按上箭头、下箭头上下移动，或按 PgUp、PgDn 键上下翻页，按 q 退出。

- 命令格式：man ...
- 示例

  #man uname　　查看 uname 命令信息



#man memcpy 查看 memcpy 函数信息



2）目录及文件操作

1、 ls：列出文件及目录信息。

- 命令格式：uname［选项］...
- 常用选项：

  -a 显示指定目录下所有子目录与文件，包括隐藏文件。

  -A 显示指定目录下所有子目录与文件，包括隐藏文件。但不列出“.”和 “..”。

-c 按文件的修改时间排序。

-l 以长格式来显示文件的详细信息。这个选项最常用。

-r 按字母逆序或最早优先的顺序显示输出结果。

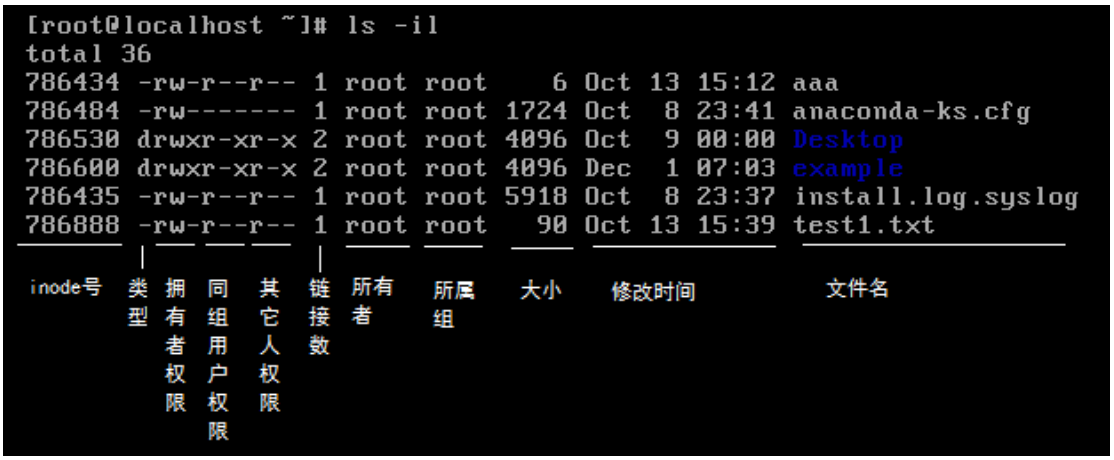
-t 显示时按修改时间（最近优先）而不是按名字排序。若文件修改时间相同，则按字典顺序。

-u 显示时按文件上次存取的时间（最近优先）而不是按名字排序。

-i 显示文件或目录的 inode 号

- 示例：

#ls -il（下面是对显示信息的简要说明）



1）inode 号：该文件或目录的索引节点号

2）类型：使用不同的字符代表不同的文件类型

-：普通文件

d：目录

b：块设备文件

c：字符设备文件

l：软链接文件

s：套接字（socket）文件

p：管道（pipe）文件

3）权限：每个文件可针对拥有者（创建者）、同组用户以及其他用户设置读、写、执行权限，以以下字符表示不同的权限：

r：读权限

w：写权限

x：执行权限

-：没有权限

2、 **chmod**：文件拥有者（属主）或特权用户修改文件访问权限。

- 命令格式：chmod ［选项］权限 文件名
- 常用选项：

  -c：输出被改变文件信息

  -R：递归遍历子目录，把修改应到目录下所有文件和子目录

  --reference=filename：参照 filename 的权限来设置

  -v：无论修改是否成功，输出每个文件的信息
- 示例：

  #chmod u+x file　　给 file 的属主增加执行权限

  #chmod 751 file　　给 file 的属主分配读、写、执行(7)的权限，给 file 的所在组分配读、执行(5)的权限，给其他用户分配执行(1)的权限

  #chmod u=rwx,g=rx,o=x file　　上例的另一种形式

  #chmod =r file　　为所有用户分配读权限

  #chmod a-wx,a+r file　　同上例

  #chmod -R u+r directory 递归地给 directory 目录下所有文件和子目录的属主分配读的权限

3、 **cd**：改变当前目录。

- 命令格式：cd ...
- 常用示例：

  #cd ccec　　相对路径，当前路径下的 ccec 目录

  #cd /home/ccec　　绝对路径

  #cd ~　　进入当前用户的 home 目录

  #cd -　　返回上一次访问的目录

  #cd ..　　进入当前目录的父目录

4、 **pwd**：显示当前路径的绝对路径。

- 命令格式：pwd

5、 cp：拷贝文件或目录。

- 命令格式：cp［选项］源文件/目录名 目的文件/目录名
- 常用选项：

  -a：常在拷贝目录时使用。保留链接、文件属性，并递归地拷贝目录，其作用等于 dpR 选项的组合。

  -r：若给出的源文件是一目录文件，此时 cp 将递归复制该目录下所有的子目录和文件，此时目标文件必须为一个目录名。

  -d：拷贝时保留链接。

  -f：删除已经存在的目标文件而不提示。

  -i：和 f 选项相反，在覆盖目标文件之前将给出提示要求用户确认。是交互式拷贝。

  - p：此时 cp 除复制源文件的内容外，还将把其修改时间和访问权限也复制到新文件中。

- 示例：

  #cp file1 file2　　将文件 file1 拷贝到文件 file2

  #cp direct1 direct2 -r　　将目录 direct1 拷贝到 direct2

6、 mv：移动文件到另一个目录，也可使用该命令重命名文件。

- 命令格式：mv［选项］源文件/目录名 目标文件/目录名
- 常用选项：

  -f：覆盖已经存在的目标文件而不提示。

  -i：覆盖已存在文件之前将给出提示要求用户确认。

- 示例

  #mv file1 file2　　将文件 file1 重命名为 file2

  #mv file1 ../file1　　将文件 file1 移动到当前目录父目录

7、 mkdir：在当前目录下创建子目录。

- 命令格式：mkdir［选项］目录名
- 常用选项：

－m：设定目录权限，类似于 chmod

－v：每次创建新目录都显示信息。

- 示例

#mkdir test1　　创建 test1 子目录

#mkdir －m 777 test　　创建 test 目录，并赋予所有人读、写、执行权限

8、 **rm**：删除文件或目录。

- 命令格式：rm［选项］文件/目录名
- 常用选项：

-r：若给出的源文件是一目录文件，此时 rm 将递归删除该目录下所有的子目录和文件。

-f：删除已经存在的目标文件而不提示。

-i：在删除文件之前将给出提示要求用户确认。

- 示例：

#rm testdirect －rf　　删除 testdirect 目录，不做提示

#rm file1　　删除 file1 文件

9、 **rmdir**：删除空目录。

- 命令格式：uname［选项］目录名
- 常用选项：

-p：递归删除目录，当子目录删除后，其父目录为空时，也一并被删除。

- 示例

#rmdir dirname

#rmdir pdir/cdir　　删除子目录 cdir，如 cdir 被删除后，pdir 为空，pdir 一并被删除

10、 **find**：搜索文件。

- 命令格式：find 查找路径［选项］…
- 常用选项：

-name　　按名字查找

-perm　　按执行权限来查找

-user　　按文件拥有者来查找

-mtime　　按文件修改时间来查找

-atime　　按文件访问时间来查找

-ctime　　按文件创建时间来查找

-type　　按文件类型来查找，参数可以是 b（块设备）、c（字符设备）、d（目录）、p（管道）、l（符号链接）、f（普通文件）

- 示例：

#find . -name “*.txt”　　在当前目录(含子目录)查找.txt 文件

#find / -name test　　在根目录（含子目录）查找 test 文件

#find ~ -type l　　在 home 目录查找符号链接文件

11、 **cat**：显示一个或多个文件的信息。

- 命令格式：cat [选项] ...
- 常用选项：

-n ：由 1 开始对所有输出的行数编号

-b ：和-n 相似，但对于空白行不编号

-s ：当遇到有连续两行以上的空白行，替换为一行的空白行

- 示例：

#cat -n test1.txt　把 test1.txt 内容加上行号显示出来（包括空行）

#cat -b test1.txt test2.txt　把 test1.txt 和 test2.txt 的内容显示出来,test2.txt 的内容显示在 test1.txt 后面(除空行外加上行号)

12、 **more**：显示文件的内容，空格向下翻页，常通过管道与其它命令配合使用。

- 命令格式：more [选项]...
- 常用选项：

+n：从第 n 行开始显示

-n：定义屏幕大小为 n 行

-c：从顶部清屏，然后显示

-s：把连续的多个空行显示为一行

- 示例

#more test.txt　　显示 test.txt 内容

#more +10 test.txt　　从第 10 行开始显示 test.txt 内容

#ls -il | more　　列出当前目录文件信息，空格翻页

13、 **less**：显示文件内容，空格、PgDn 向下翻页，PgUp 向上翻页，上、下箭头上下翻行，按 q 退出。

- 命令格式：less [选项] ...
- 常用选项：

-e：文件内容显示完毕后，自动退出

-f：强制显示文件

-N：每一行行首显示行号

-s：将连续多个空行压缩成一行显示

-S：在单行显示较长的内容，而不换行显示

-x<数字>：将 TAB 字符显示为指定个数的空格字符。

- 示例

#less -N test.cpp　　显示 test.cpp 内容，前面显示行号

#less -x 2 test.cpp　显示 test.cpp 内容，TAB 定义为 2 个空格

#ls -il | less　　列出当前目录信息，使用 less 分页显示

3）信息搜索及过滤

1、 **grep**：一个强大的文本搜索工具，可以使用正则表达式进行搜索，并把匹配的行打印出来。

- 命令格式：grep [选项] ...
- 常用选项：

-c：只输出匹配行的计数。

-I：不区分大 小写(只适用于单字符)。

-h：查询多文件时不显示文件名。

-l：查询多文件时只输出包含匹配字符的文件名。

-n：显示匹配行及 行号。

-s：不显示不存在或无匹配文本的错误信息。

-v：显示不包含匹配文本的所有行。

- 基本正则表达式：

  \ 忽略正则表达式中特殊字符的原有含义

  ^ 匹配正则表达式的开始行

  $ 匹配正则表达式的结束行

  [ ] 单个字符；如[A] 即 A 符合要求

  [-] 范围 ；如[A-Z]即 A，B，C 一直到 Z 都符合要求

  . 所有的单个字符

  * 所有字符，长度可以为 0

- 示例：

  #ls -l | grep test　　列出当前目录中仅包含 test 的文件信息

  #cat test.txt | grep ‘^a’ 仅显示出 test.txt 中以字符 a 开头的行

  #ps -aux | grep test　　仅显示出当前进程信息中包含 test 的进程信息

4）进程状态查询及控制

1、 **ps**：显示当前时刻，进程的状态。

- 命令格式：ps [选项] ...
- 常用选项：

  a：列出所有进程

  -a：列出统一终端下所有的进程

  -w 显示加宽可以显示较多的信息

  -au 显示较详细的资讯

  -aux 显示所有包含其他使用者的进程的详细信息

- 示例

  #ps - aux　　列出当前所有进程的详细信息

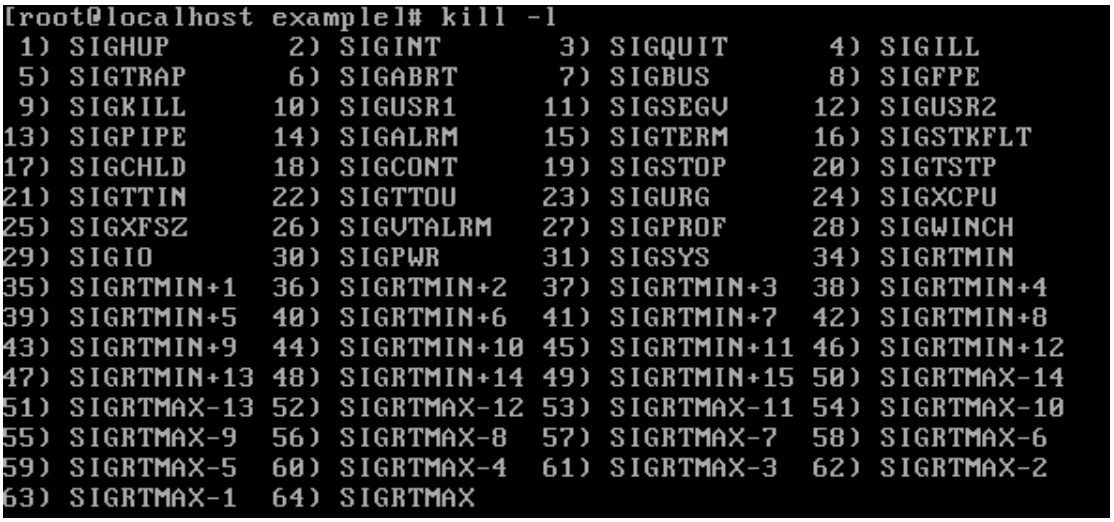
2、 **kill**：终止进程，或者向进程传送信号。

- 命令格式：kill［选项］［参数］［进程号］...
- 常用选项：

  -l：列出所有可用信号

  -u：指定用户
- 示例：

  kill -l



  可以使用 kill 命令来对进程进程终止、暂停、恢复等操作。假设某进程的 PID 为 1234：

  #kill 1234　　默认向进程发送 SIGTERM 信息（终止进程）

  #kill -SIGKILL 1234　若上例不能终止进程，可发该信号强行终止

  #kill -9 1234　　同上例，所有信号可以以其对应的数字替代

  #kill -SIGSTOP 1234　暂停进程（同 Ctrl + Z）

  #kill -SIGCONT　1234　进程继续运行

  #kill -SIGINT　1234　进程中断（同 Ctrl + C）

  #kill －SIGQUIT 1234 进程退出（同 Ctrl + \）

5）分区及文件系统管理

1、 **df**：检查文件系统挂载点、使用情况，可以使用该命令来检查硬盘空间使用了多少，还剩下多少。

- 命令格式：df［选项］...
- 常用选项：

-a：显示所有文件系统的磁盘使用情况，包括 0 块（block）的文件系统，如/proc 文件系统。

-k：以 k 字节为单位显示。

-i：显示 i 节点信息，而不是磁盘块。

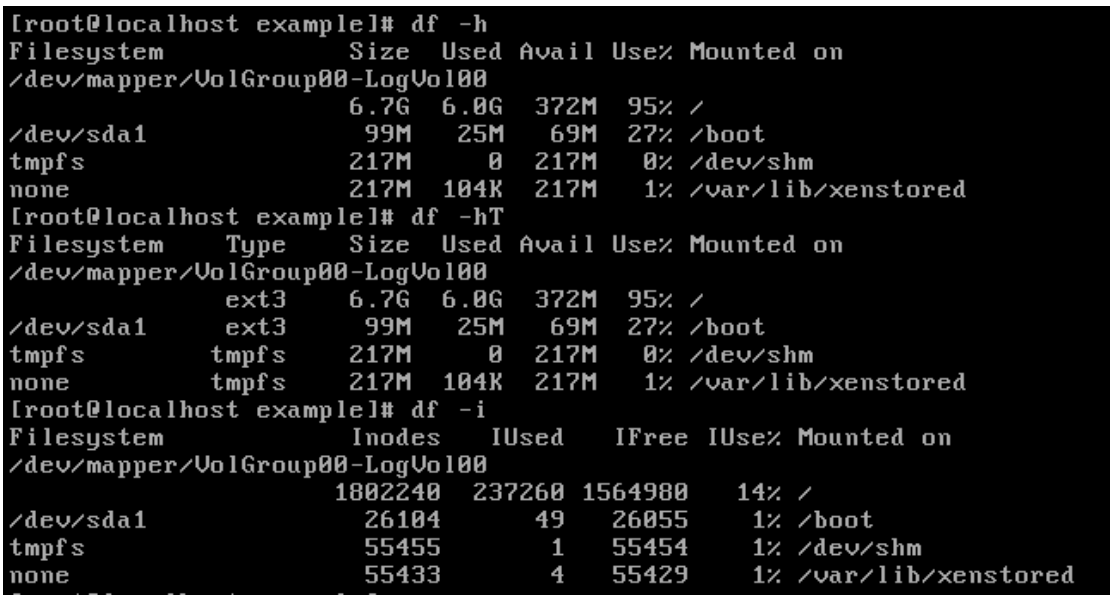
-t：显示各指定类型的文件系统的磁盘空间使用情况。

-x：列出不是某一指定类型文件系统的磁盘空间使用情况（与 t 相反）。

-T：显示文件系统类型。

-h：以友好的方式显示

- 示例：

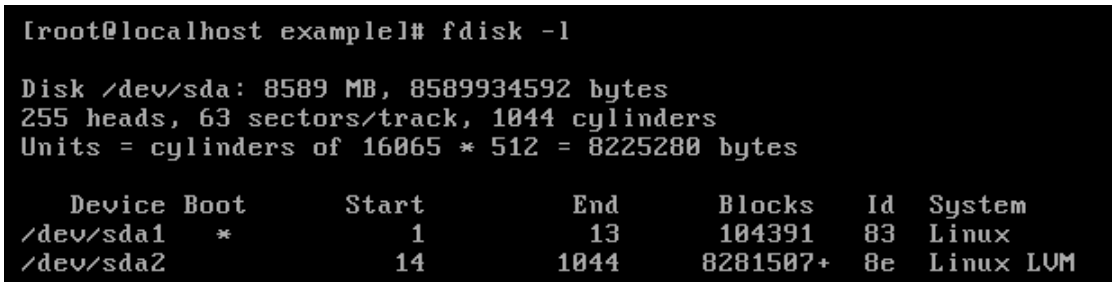


2、 **fdisk**：常用于查看系统中硬盘及分区信息，也可用于对硬盘分区进程操作，如删除分区、添加分区等。注意：该命令需要 root 权限，慎用对分区操作。

- 命令格式：fdisk［选项］［设备名］
- 常用选项：

  -l：列出所有或指定设备的信息
- 示例：

  #fdisk -l　　列出所有硬盘的信息



#fdisk /dev/sda　　在之后的界面中，可对/dev/sda 分区进行操作

3、 mount：挂载文件系统

- 命令格式：mount［选项］设备名 挂载点
- 常用选项：

-t<文件系统类型> 指定设备的文件系统类型，常见的有：

1）minix：linux 早期文件系统　2）ext2/ext3： linux 常用文件系统

3）msdos： MS-DOS 的 fat16　4）vfat： fat32

5）nfs：网络文件系统　　6）iso9660：CD-ROM 光盘标准文件系统

7）ntfs：windows NTFS　　8）hpfs OS/2 文件系统

-o<挂接方式> 设备的挂接方式，常见的有：

1）loop：用来把一个文件当成硬盘分区挂接上系统

2）ro：采用只读方式挂接设备

3）rw：采用读写方式挂接设备

4）iocharset：指定访问文件系统所用字符集

- 示例：

#mount　　显示所有文件系统挂载信息

#mount -a　　挂载/etc/mtab 中记载的所有文件系统

#mount -t ext2 /dev/fd0 /mnt/floppy　挂载 ext2 格式的软盘（逻辑设备名为/dev/fd0）到/mnt/floppy 目录

#mount -t vfat /dev/hda1 /mnt/disk　挂载 fat32 格式的硬盘分区（逻辑设备名/dev/hda1）到/mnt/disk 目录

#mount -t iso9660 /dev/hdc /mnt/cdrom　装载一个光盘（逻辑设备名为/sev/hdc）到/mnt/cdrom 目录

#mount -o loop -t iso9660 /home/mydisk.iso /mnt/cdrom　　将

一个 iso 镜像文件挂载到/mnt/cdrom 目录

4、 **umount**：卸载文件系统。注意如该文件系统正在使用，将会卸载失败

- 命令格式：uname [选项] [设备名] [挂载点]
- 常用选项：

  -a：卸载/etc/mtab 中记录的所有文件系统

  -t：仅卸载选项中所指定的文件系统
- 示例：

  #umount -a　　卸载/etc/mtab 中记录的所有文件系统

  #umount /dev/sda1　　卸载逻辑设备/dev/sda1

  #umount /mnt/cdrom　　卸载挂载在/mnt/cdrom 目录下的文件系统

二、Linux 系统编程界面

1、实验说明

学习使用 vi/vim 来编写 C 及 C++代码；学习使用 gcc 及 g++来编译链接 C 及 C++代码，学习使用 gdb 进行代码调试。

2、vi/vim 简介

vi/vim 是 Linux、Unix 字符界面下常用的编辑工具，也是系统管理员常用的一种编辑工具。很多 Linux 发行版都默认安装了 vi/vim。vim 是 vi 的升级版，和 vi 的基本操作相同，其相对于 vi 的优点主要在于可以根据文件类型高亮显示某些关键字，如 C 语言关键字，便于编程。

vi/vim 有两种状态：命令状态和编辑状态。

1）命令状态：可以输入相关命令，如文件保存、退出、字符搜索、剪切等操作；vi/vim 启动时，默认进入命令状态。在编辑状态下，按 ESC 键，即可进入命令状态；

2）编辑状态：在该状态下进行字符编辑。在命令状态下，按 i/a/I/A/O/o 等键即可进入编辑状态。

1）vi/vim 常用命令

表 1 vi/vim 常用命令（命令状态下使用）

命令	功能说明
插入字符、行，执行下面操作后，进入编辑状态	
a	进入插入模式，在光标所在处后面添加文本
i	进入插入模式，在光标所在处前面添加文本
A	进入插入模式，在光标所在行末尾添加文本
I	进入插入模式，在光标所在行行首添加文本（非空字符前）
o	进入插入模式，在光标所在行下新建一行
O	进入插入模式，在光标所在行上新建一行
R	进入替换模式，覆盖光标所在处文本

剪切、粘贴、恢复操作	
dd	剪切光标所在行
Ndd	N 代表一个数字，剪切从光标所在行开始的连续 N 行
yy	拷贝光标所在行
Nyy	N 代表一个数字，复制从光标所在行开始的连续 N 行
yw	复制从光标开始到行末的字符
Nyw	N 代表一个数字，复制从光标开始到行末的 N 个单词
y^	复制从光标开始到行首的字符
y$	复制从光标开始到行末的字符
p	粘贴剪切板的内容在光标后（或所在行的下一行，针对整行复制）
P	粘贴剪切板的内容在光标前（或所在行的上一行，针对整行复制）
u	撤销上一步所做的操作
保存、退出、打开多个文件	
:q!	强制退出，不保存
:w	保存文件，使用:w file，将当前文件保存为 file
:wq	保存退出
:new	在当前窗口新建一个文本，使用:new file，打开 file 文件，使用 Ctrl+ww 在多个窗口间切换
设置行号，跳转	
:set nu	显示行号，使用:set nu!或:set nonu 可以取消显示行号
n+	向下跳 n 行
n-	向上跳 n 行
nG	跳到行号为 n 的行
G	跳到最后一行
H	跳到第一行
查找、替换	
/***	查找并高亮显示***的字符串，如/abc

:s	:s/old/new//,用 new 替换行中首次出现的 old :s/old/new/g,用 new 替换行中所有的 old :n,m s/old/new/g,用 new 替换从 n 到 m 行中所有 new :%s/old/new/g,用 new 替换当前文件中所有 old

2）vi/vim 使用示例

如要编辑当前目录下名为 helloworld.c 的文件：

step1：输入 vim helloworld.c，即可进入 vim 窗口，如 helloworld.c 不存在，则新建该文件，否则是打开该文件。vim 默认处于命令状态。

step2：按 i，进入编辑状态。

step3：编辑代码。

step4：按 Esc，回到命令状态。

step5：输入:wq，保存并退出。

3）vi/vim 常见问题及解决方法

问题 1：按 Ctrl+s 键（Windows 下的保存快捷键）后，发现 vim 对后续按键不再反应。原因是 Ctrl+s 命令在 linux 下是取消回显命令，所键入字符不显示在屏幕上，按 Ctrl+q 即可恢复回显。

问题 2：在启动 vim 时，没有键入文件名。vim 默认会创建一个新的文件，编辑完成后，进入命令状态，键入:w filename，将其保存为 filename。

问题 3：vim 非正常退出后，再次编辑该文件时，会出现“swap file .helloworld.c.swp already exists!”的提示（假设 helloworld.c 是 vim 非正常退出时编辑的文件名），使用 rm .helloworld.c.swp 删除该文件，重新编辑即可。

3、Linux 下 C 程序开发

C 语言的编译器被简称为 cc，不同厂商的类 UNIX 系统所带的 C 语言编译器均包含不同的功能和选项。Linux 系统中，通常使用 GNU C 编译器，简称为 gcc，下面以 HelloWorld 为例，简单介绍 Linux 下 C 语言开发过程。

1）简要 C 语言开发过程

step1：使用 vim 编辑 hello.c

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   printf("Hello World \n");
   exit(0);
}
```

step2：编译 hello.c

```
#gcc -o hello hello.c
```

gcc 命令将 hello.c 编译成可执行文件 hello，如不加-o 选项，编译器会把编译后的可执行文件命名为 a.out。

step3：执行 hello

```
#./hello
Hello World
```

在 hello 前面添加./，是让 shell 在当前目录下寻找可执行文件，如不添加./，shell 会在 PATH 环境变量设置的目录中去寻找该可执行文件，但这些目录中通常不会包含当前目录。

注：对于复杂的大型程序，一般编写 makefile 文件来进行编译链接，makefile 文件的编写请参考相关资料。

2）gcc 常用选项

gcc 选项很多，下表列出 gcc 常用的一些选项。

表 2 gcc 常用选项

选项	说明
-c	只做预处理、编译和汇编，不作链接，常用于不含 main 的子程序
-S	只进行预处理和编译，生成.s 汇编文件
-o	指定输出的目标文件名
-Idir	头文件搜索路径中添加目录 dir

-Ldir	库文件搜索路径中添加目录 dir
-lname	链接 libname.so 库来编译程序
-g	编译器编译时加入 debug 信息，供 gdb 使用
-O[0~3]	编译器优化，数字越大，优化级别越高，0 表示不优化

3）Linux 下 C++程序开发

Linux 下 C++程序开发过程和 C 程序开发过程类似，但编译时使用 g++命令。下面仍以 HelloWorld 为例，简要说明其开发过程。g++常用编译选项和 gcc 类似，这里不再赘述。

step1：使用 vim 编辑 hello.cpp

```
#include <iostream>
int main()
{
   cout << "Hello World" << endl;
   exit(0);
}
```

step2：编译 hello.cpp

```
#g++ -o hello hello.cpp
```

step3：执行 hello

```
#./hello
Hello World
```

4、使用 gdb 调试 C/C++程序

gdb 是 Unix/Linux 下的一个功能强大的程序调试工具。当程序出现段错误（segment fault）或者逻辑错误时，可以使用 gdb 进行调试。gdb 主要有四大功能，即启动程序，可以按照自定义要求随心所欲的运行程序；可让被调试程序在所指定的调置断点处停住（断点可以是条件表达式）；当程序被停住时，可以检查程序中所发生的事；动态改变程序的执行环境。

1）gdb 常用调试命令

可以使用#gdb program 启动目标代码进行调试，但目标代码编译时，必须使用-g 选项编译。进入调试界面后，可以输入相关 gdb 命令控制目标代码的运行。下表为 gdb 常用的调试命令。

表 3 gdb 常用调试命令

命令	说明
list（或 l）	列出源代码，接着上次位置往下列，每次列 10 行
list 行号	从给定行号开始列出源代码
list 函数名	列出某个函数的源代码
break（或 b）行号	在给定行号出设置断点，gdb 会给出一个断点号
break 函数名	在给定函数开头设置断点
delete breakpoint 断点号	删除给定的断点
start	开始执行程序，停在 main 函数第一句前面等待命令
run（或 r）	开始执行程序，直到遇到断点
next（或 n）	执行下一条语句
step（或 s）	执行下一条语句，如是函数调用，则进入函数中
continue（或 c）	继续执行程序，直到遇到断点
finish	连续运行到当前函数返回，然后停下来等待命令
print（或 p）	打印表达式的值
display 变量名	跟踪查看某个变量的值，每次停下来都显示该变量的值
undisplay 跟踪显示号	取消对变量的跟踪查看
set var	修改变量的值
quit	退出 gdb

2）gdb 调试示例

下面以一个例子简要说明使用 gdb 的调试过程。

step1：编辑源代码 tst_gdb.c

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

```
int add()
{
    int sum =0, i;
    for(i = 0; i < 10; i ++)
    {
        sum += i;
    }
    return sum;
}
int main()
{
    int result = 0;
    result = add();
    printf(“the result is %d \n”, result);
    return 0;
}
```

step2：编译链接源代码，形成目标代码，注意使用-g 编译选项。

```
gcc -g  -o tst_gdb tst_gdb.c
```

step3：使用 gdb 进行跟踪调试，如下图所示。

```
[root@localhost example]# gdb tst_gdb  <--------------  启动gdb
GNU gdb Fedora (6.8-37.el5)
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i386-redhat-linux-gnu"...
(gdb) l                                 <--------------  列出源码，l相当于list
warning: Source file is more recent than executable.
6               for(i = 0; i < 10; i++)
7               {
8                       sum += i;
9               }
10              return sum;
11      }
12
13      int main()
14      {
15              int result = 0;
(gdb) l                                 <----------------  继续列出源码
16              result = add();
17              printf("the result is %d\n", result);
18              return 0;
19      }
(gdb) b 16                              <----------------  在第16行处设置断点
Breakpoint 1 at 0x80483c7: file tst_gdb.c, line 16.
(gdb) r                                 <----------------  开始运行
Starting program: /root/example/tst_gdb

Breakpoint 1, main () at tst_gdb.c:16   <----------------  停在第一个断点处
16              result = add();
(gdb) s                                 <----------------  进入add函数
add () at tst_gdb.c:5
5               int sum = 0, i;
(gdb) n                                 <----------------  下一语句
6               for(i = 0; i < 10; i++)
(gdb) display sum                       <----------------  跟踪sum变量
1: sum = 0
(gdb) n                                 <----------------  下一语句
8                       sum += i;
1: sum = 0
(gdb) n
6               for(i = 0; i < 10; i++)
1: sum = 0
(gdb) n
8                       sum += i;
1: sum = 0
(gdb) n
6               for(i = 0; i < 10; i++)
1: sum = 1
(gdb) n
8                       sum += i;
1: sum = 1
(gdb) n
6               for(i = 0; i < 10; i++)
1: sum = 3
(gdb) c                                 <----------------  继续运行
Continuing.
the result is 45                        <----------------  打印结果

Program exited normally.
(gdb) q                                 <----------------  退出gdb
```