

**Linux**

**实验报告**

|  |  |
| --- | --- |
| **学生姓名** | **吴家豪** |
| **学 号** | **8208220422** |
| **专业班级** | **大数据2204** |
| **指导教师** | **胡小龙** |
| **学 院** | **计算机学院** |
| **完成时间** | **2024-6-27** |

**实验一 Linux的安装（课后）**

1、实验目的

* 1. 了解硬盘分区的概念和方法；
  2. 掌握硬盘的分区规划；
  3. 掌握Linux操作系统的安装和配置过程。

2、实验设备

一台pc机、RedHat Linux 7.2以上版本、VMware Workstation v5.5

3、实验原理

Linux可以以多种方式安装在PC机上： (1)独立分区安装、 (2)DOS分区安装和 (3)虚拟机VMWare下安装。鉴于VMware下安装对原来系统影响较小且不影响本实验目的，因此采用VMWare下安装方式。

4、实验步逐

(1) 在Windows XP下安装VMware 5.5

1. 配置虚拟机
2. 启动虚拟机
3. 启动Linux安装过程
4. 安装过程配置
5. 安装后配置
6. 第1次启动 VMWare下Linux操作系统

5、实验记录

1. **记录详细安装过程**

创建虚拟机



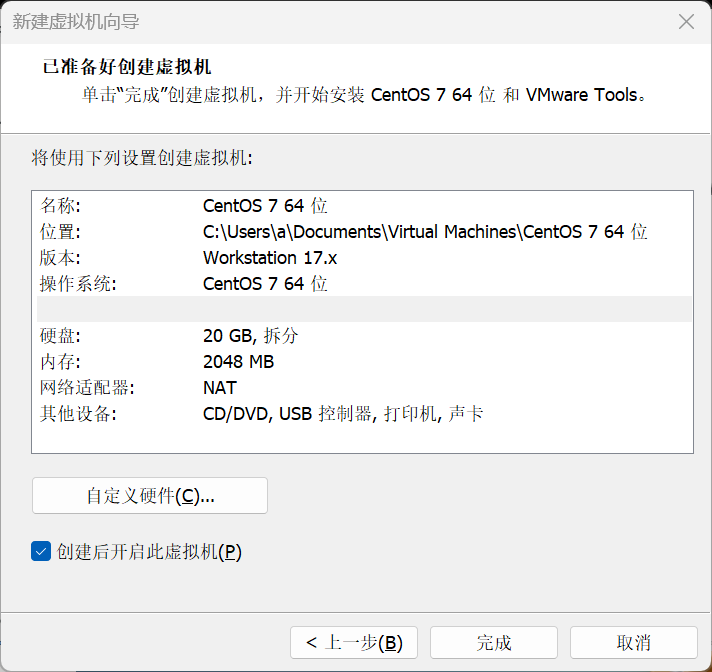
导入相应光盘映像文件



创建虚拟机用户名及密码



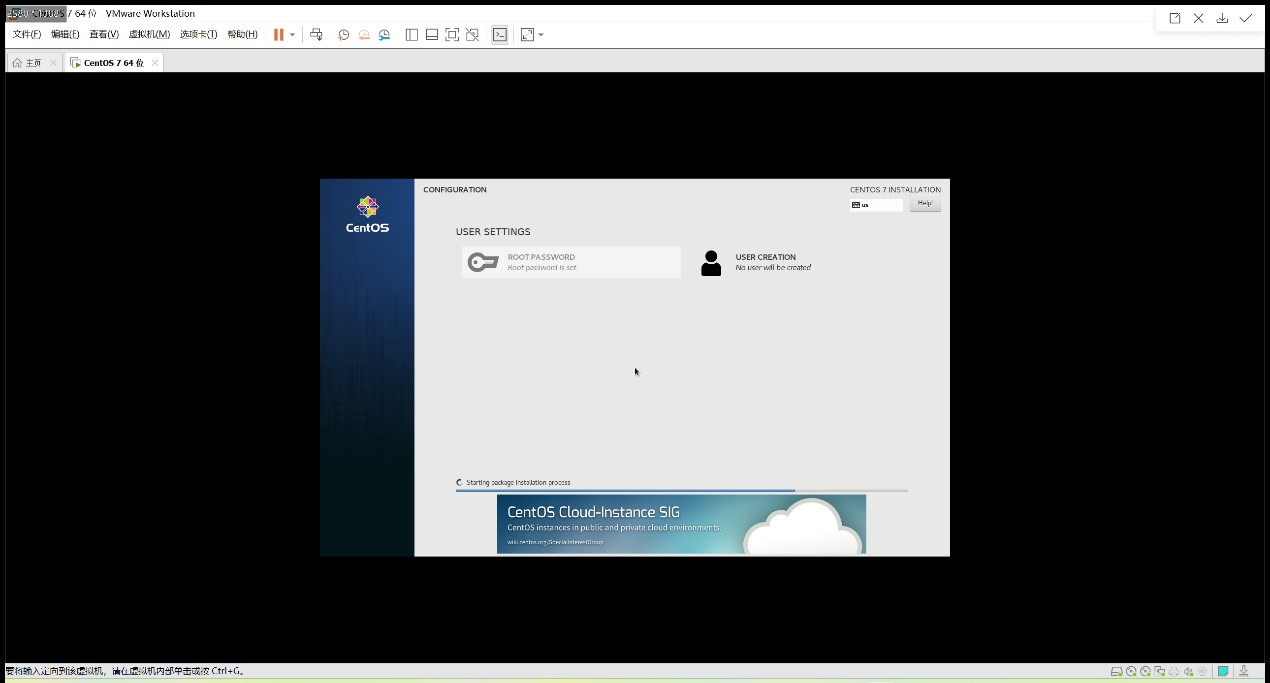
设置虚拟机的硬件



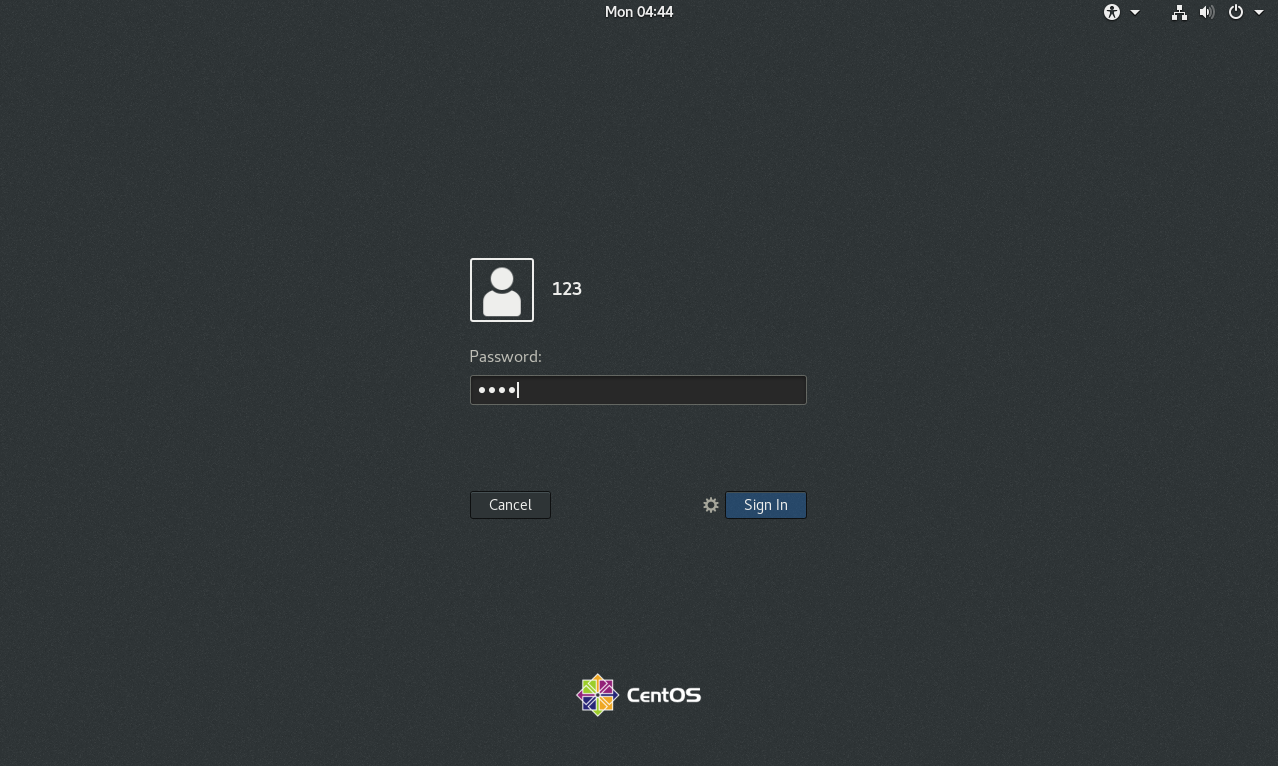
Ps：可打开任务管理器查看自身电脑状态，设置合适的内核及内存，我设置的可用内存为2GB



打开虚拟机，Linux系统初始化



登录界面

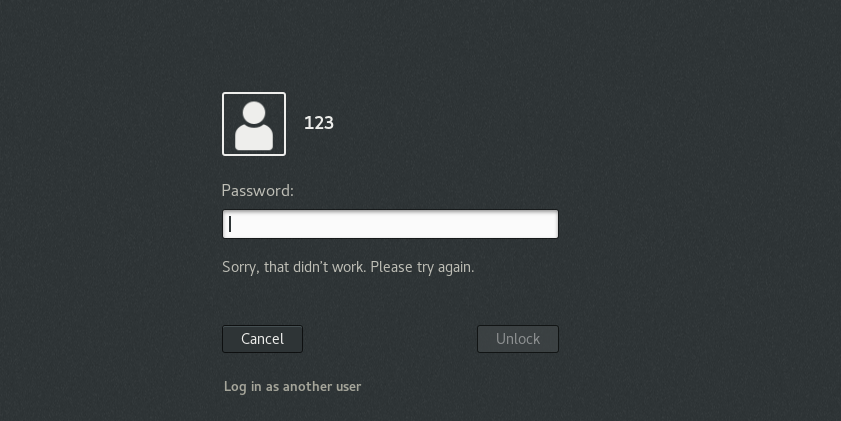


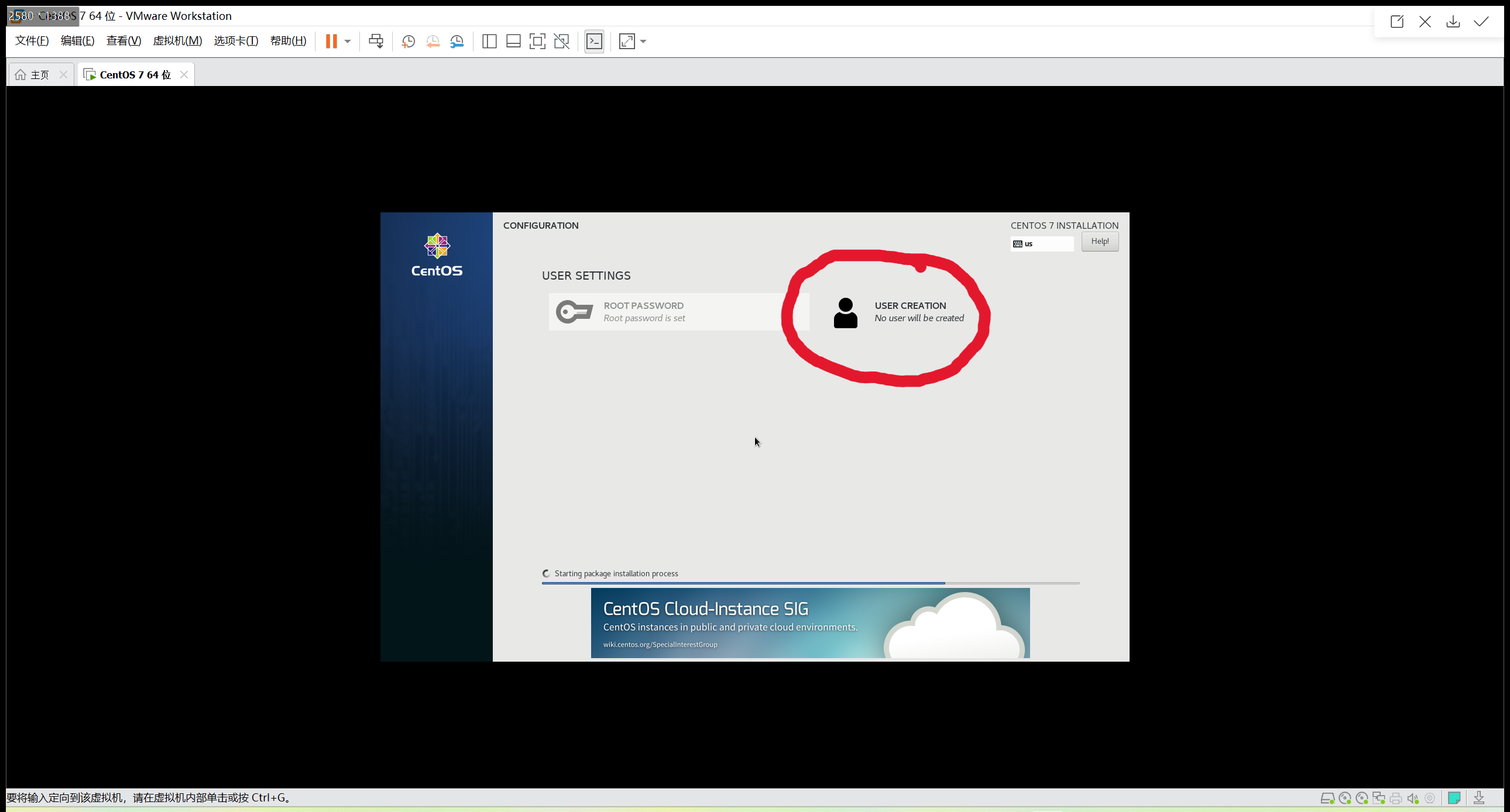
打开后的界面



1. 安装过程中出现的问题及其解决措施

设置完用户名，总是显示密码错误，登陆失败进不去，多次尝试重建虚拟机，发现需要在这个界面点击该按钮，进入系统用户创建界面创建用户。





**实验2 Linux基本操作**

1、实验目的

(1)复习Linux基本命令。

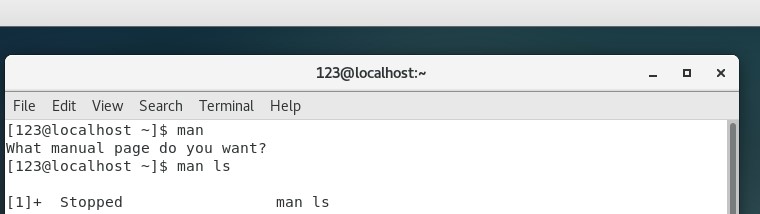
(2)掌握常用Linux命令

**2、实验内容**

**建议**在自己安装的Linux系统、华为云平台、OpenEuler等系统下操作使用。

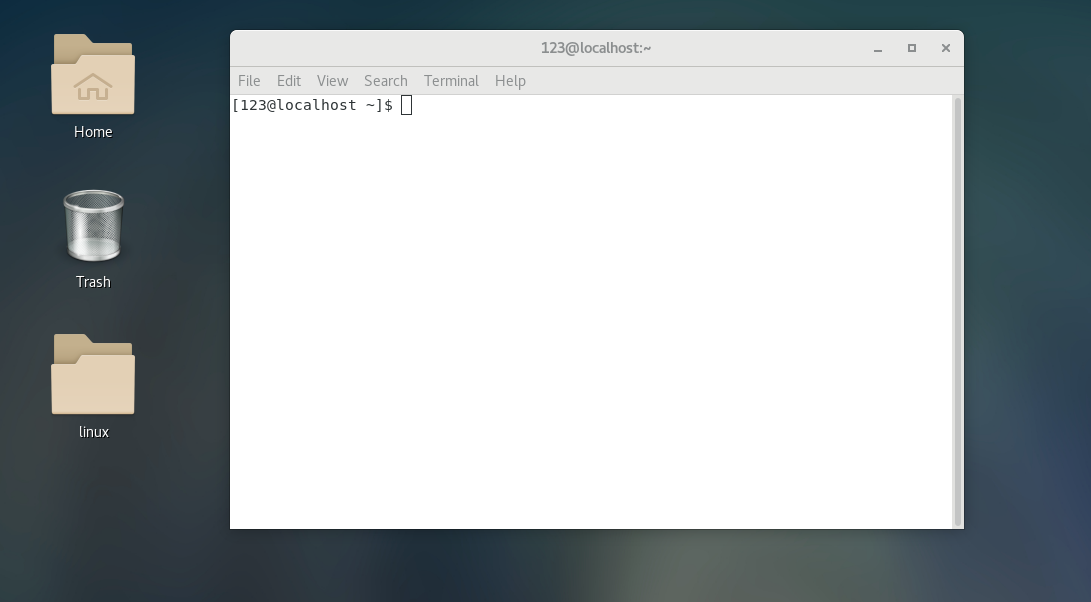
(1)练习命令行模式下的常用命令：

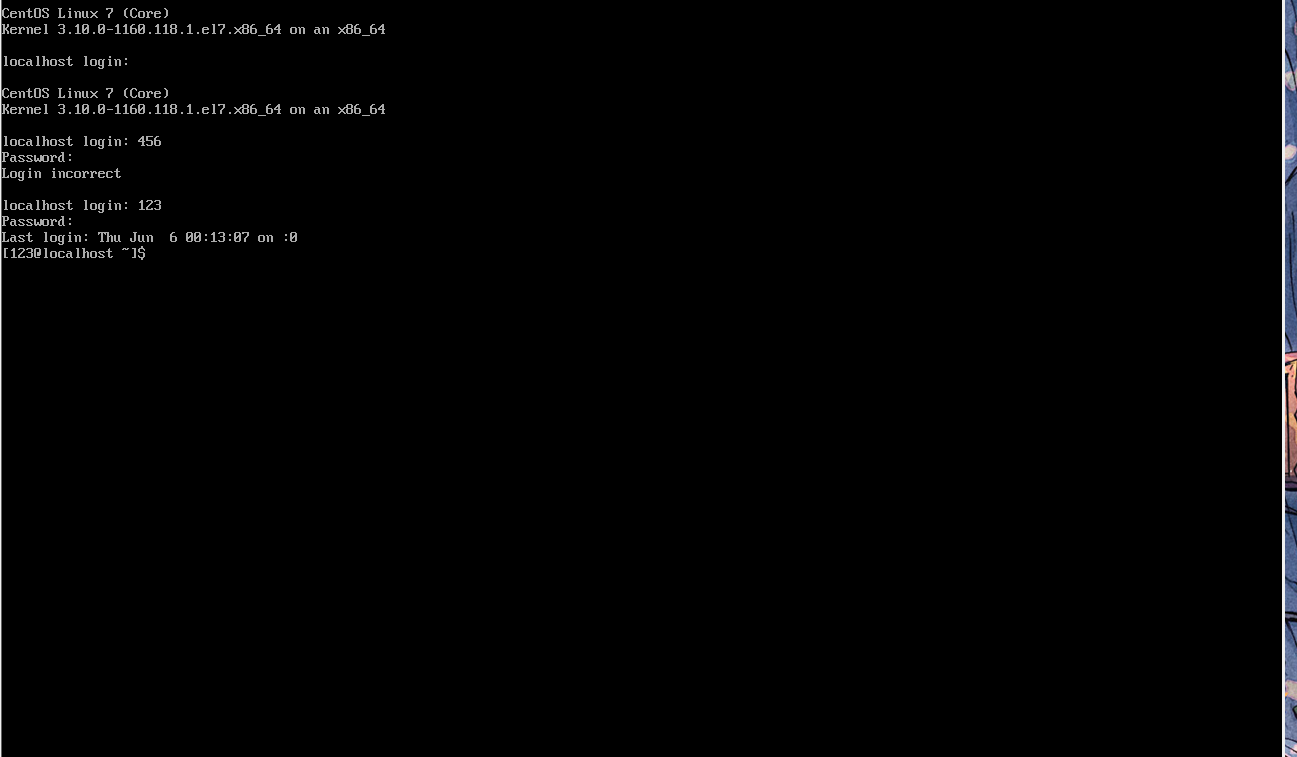
**man**:命令帮助

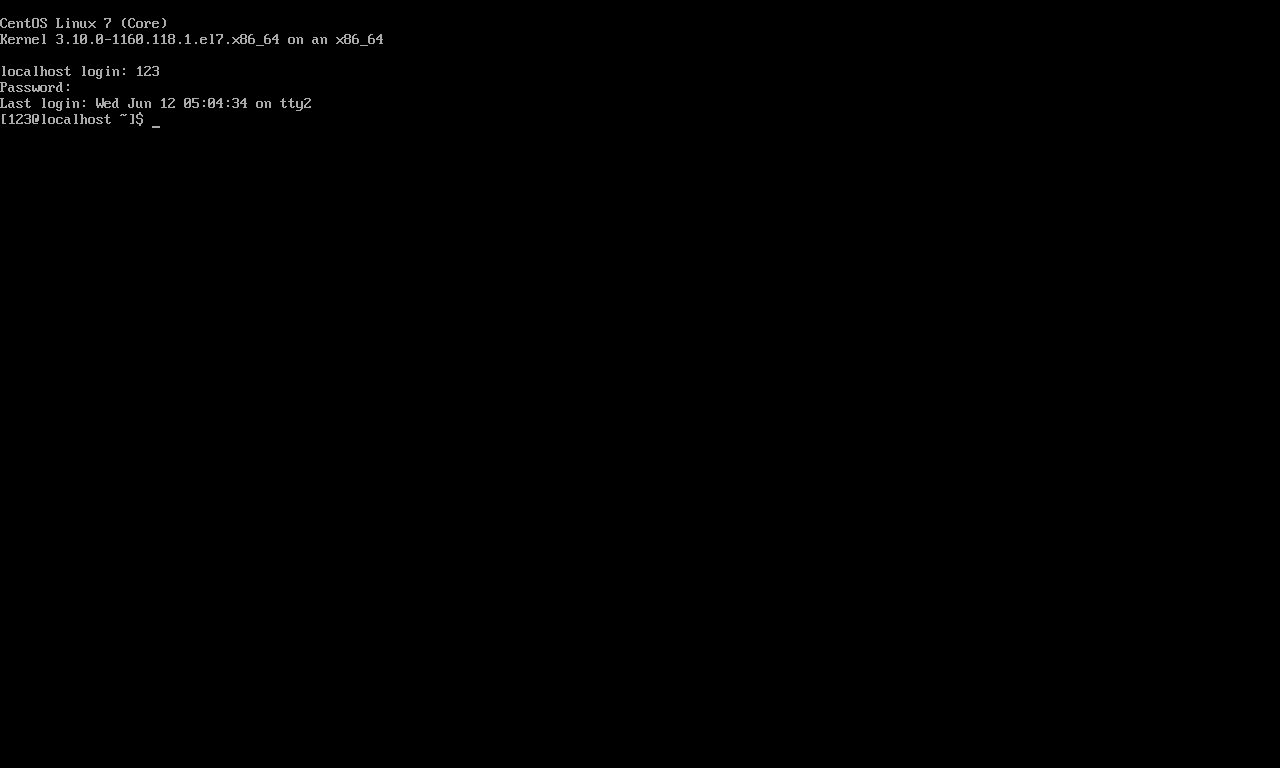


**ALT-Fx**:虚终端切换

在桌面环境，需要使用CTRL + ALT + F1  到 F6 来切换，以下是在桌面环境下CTRL + ALT + F1到F3的三个界面：





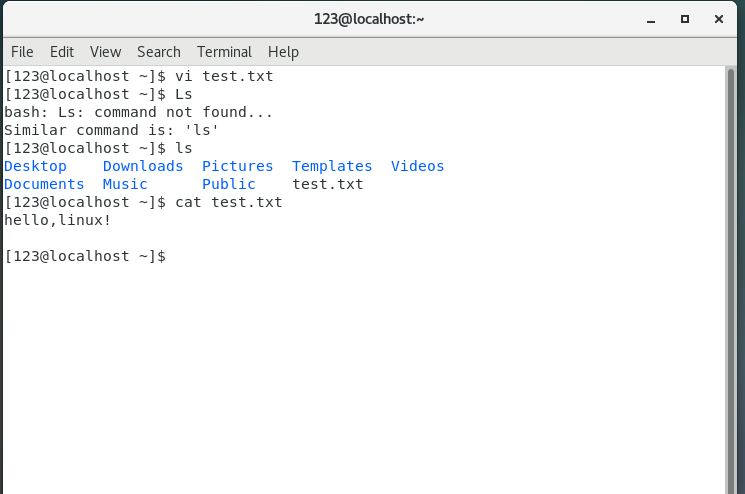


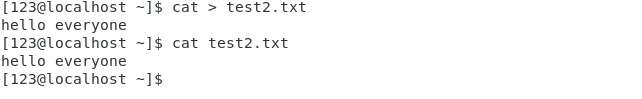
常用命令:

**cat**：

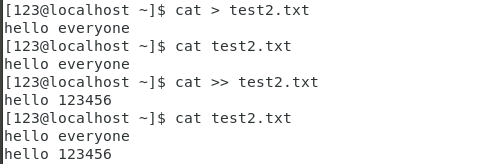
cat filename.txt查看文件内容

以下是创建test.txt ，并查看文件内容

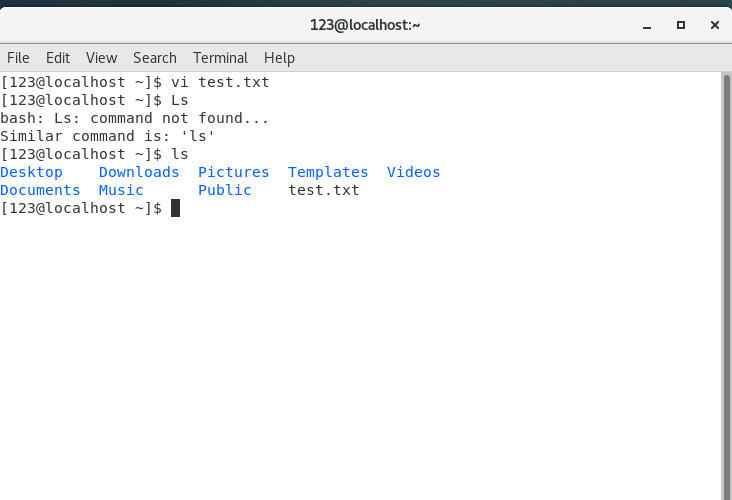
cat > newfile.txt 创建新文件，并输入文件内容 按ctrl+d可退出编辑状态

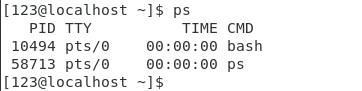


cat >> existingfile.txt 追加内容到文件



ls ：显示文件目录

**ps**：显示当前进程



**chmod**：改变文件的权限

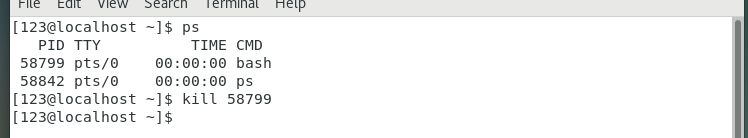
chmod permissions filename，其中permissions是新权限的表示，filename是文件或目录的名称。

+x 添加执行权限



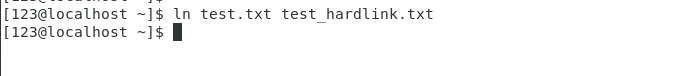
**kill**：发送信号到进程

kill PID 发送终止进程的信号到指定进程，其中 PID 是你想要终止的进程的进程ID。



**ln**：创建链接，主要有硬链接和符号链接两种类型

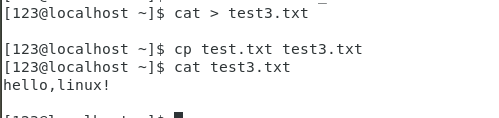
以下是创建连接的示例：





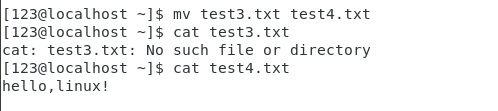
**cp**：复制文件或目录的命令行工具

cp 目标文件 要复制的文件



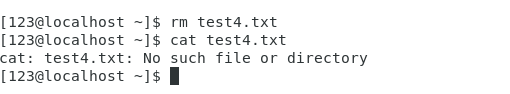
**mv**： 可以移动文件，重命名文件，移动目录

以下是一个重命名的示例：将test3.txt重命名为test4.txt



**rm**：删除文件或目录

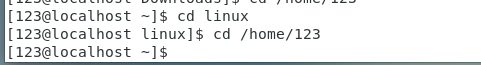
以下是删除test4.txt的示例



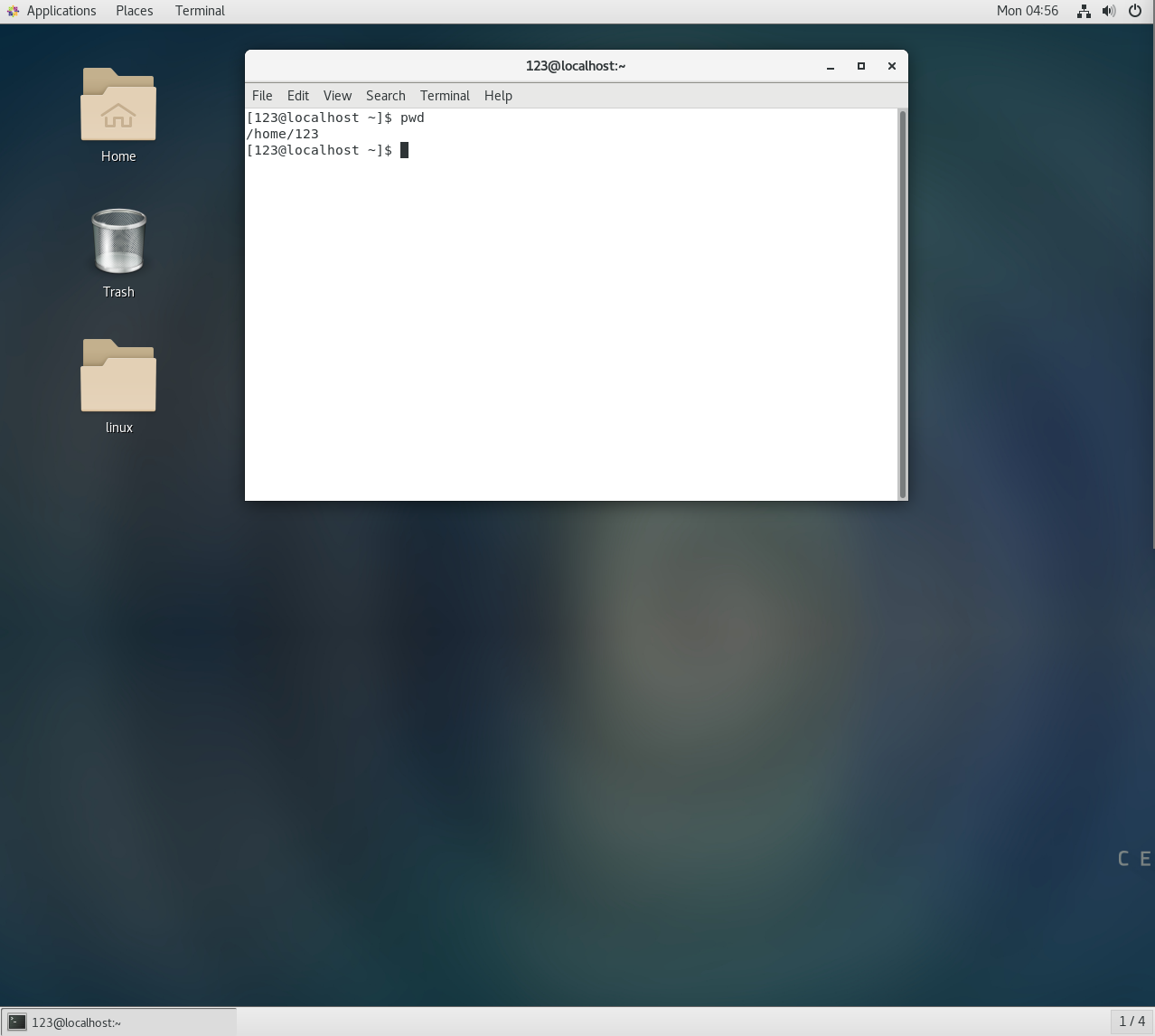
**cd**：用于改变当前工作目录

**切换到指定目录**cd /path/to/directory, 这将切换当前工作目录到指定的绝对路径 /path/to/directory。

**以下是linux和home目录之间的切换**



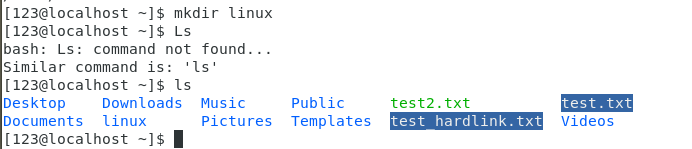
**pwd**: 显示当前工作目录的完整路径



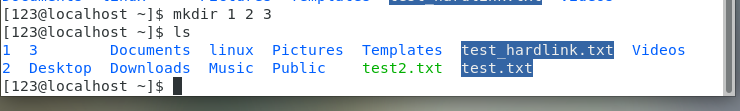
**mkdir**: 创建新目录

mkdir directory\_name这将在当前工作目录下创建一个名为 directory\_name 的新目录，可创建多个目录

以下创建了一个linux的目录



以下创建了 1 2 3 三个目录



**chown**：改变文件或目录的所有者（owner），所属组（group）

**改变文件或目录的所有者chown new\_owner filename**

**查询所有用户：（后面的命令创建的用户apple和banana）**

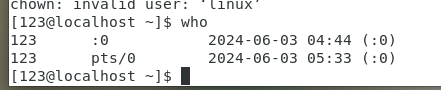


将test2.txt 转移到apple用户下，hown 命令不会移动文件或改变它的位置，它只更改文件的元数据，使用 chown 命令改变一个文件的所有权时，这个文件仍然保留在它原来的目录中。

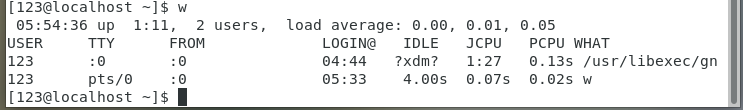
查看文件test2.txt所属目录 ls -l test2.txt，可以看到用户apple 是文件的所有者，而123是文件所属的用户名组



**who**：显示当前登录的用户：

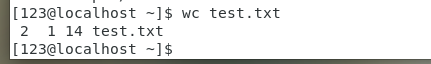


**w**：显示当前已登录用户的信息, 包括他们正在执行的命令和他们的登录时间

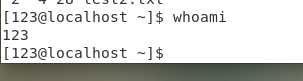


**wc**: 计算文件中的行数、单词数、字节数和字符数

可输入 -l 只计算行数，-w只计算单词数，-c只计算字节数，-m只计算字符数，直接输入则显示行数，单词数，字节数



**whoami**：显示当前执行命令用户的用户名



**date**：**显示当前日期和时间**

**若要求格式化输出时间可输入：** date +"%Y-%m-%d %H:%M:%S"

**以下为直接输出的示例**



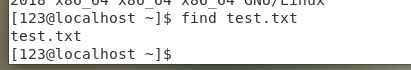
**uname**：显示系统信息，输入-a显示所有系统信息



**find**：用于在文件系统中搜索符合条件的文件和目录

在指定目录下搜索文件和目录：

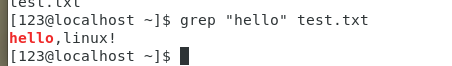
find /path/to/search 这将在 /path/to/search 目录及其所有子目录下搜索所有文件和目录。



**grep**：用于搜索文件中的文本模式

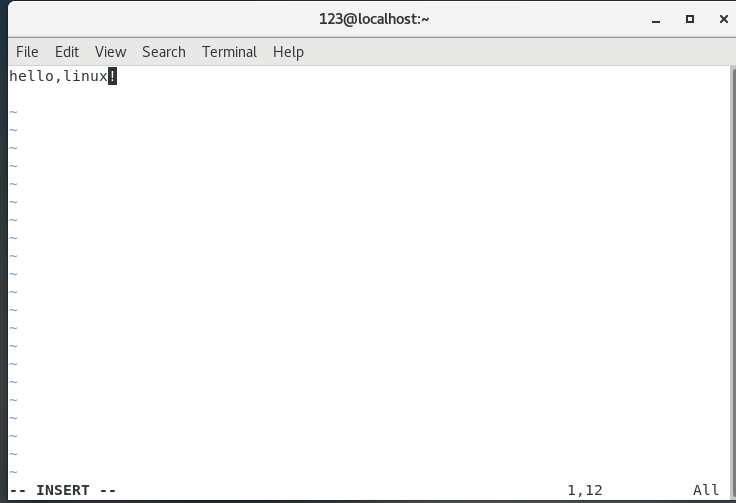
**在文件中搜索文本grep "hello" filename** ，将在 filename 文件中搜索所有包含 "hello" 文本的行。

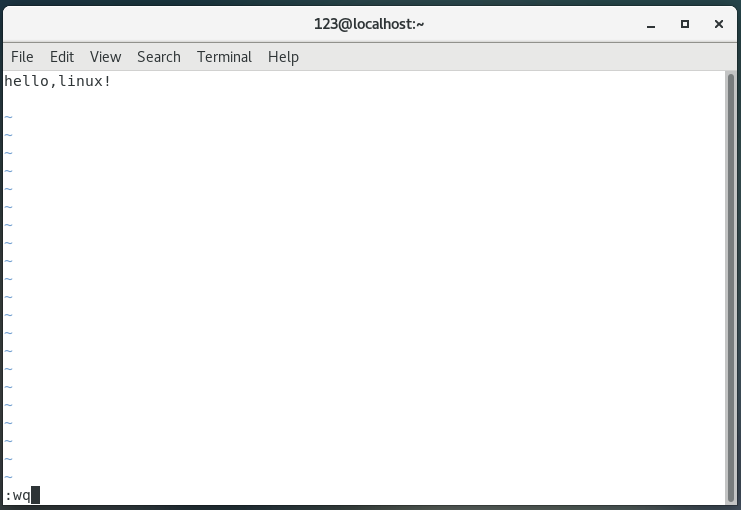
以下为搜索hello的示例



(2) 编辑器vi的使用：使用vi建立并修改一个文本文件

在终端输入vi test.txt创建文本文件，按i可进入编辑模式输入文本内容（界面左下角会显示—INSERT--）此时可输入内容，按Esc可退出编辑模式，可输入:wq保存并退出文本文件（依旧是看左下角，输入的内容会在左下角显示，当正确输入指令即可退出，输入:q不保存退出，输入:w可保存，输入:wq，可保存退出，这个只需要按一次Enter）





(3) 档案管理

**rpm**：软件包管理器

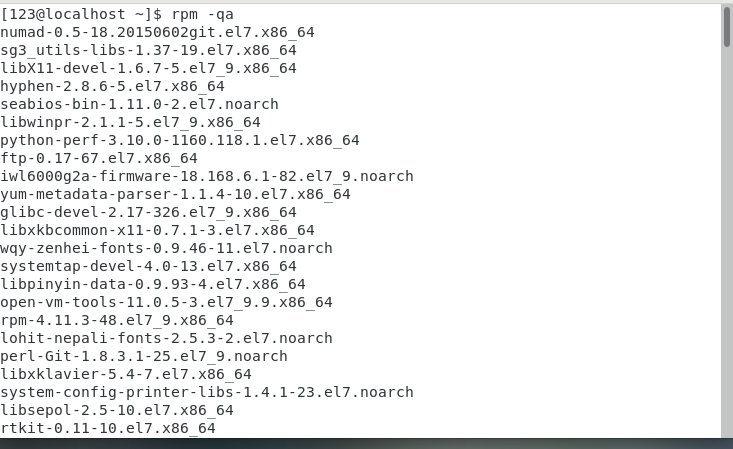
若要安装软件包可输入： sudo rpm -ivh package.rpm

-i：安装软件包。

-v：显示详细的安装信息。

-h：以散列符号显示安装进度。

rpm -qa 查询已安装软件包

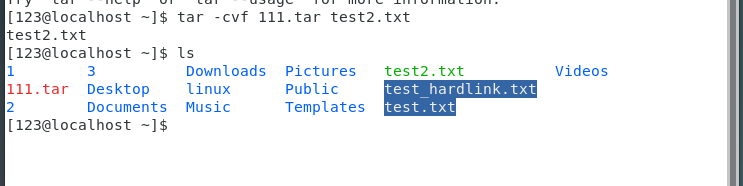


**gzip**：压缩或解压文件

**tar**：归档

tar -cvf archive\_name.tar directory\_or\_file：创建一个名为 archive\_name.tar 的归档文件，其中 directory\_or\_file 是要归档的目录或文件。

以下是一个创建归档文件111.tar的示例

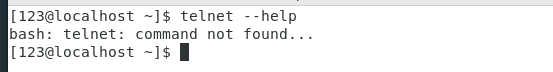


(4) 网络命令

**telnet**：

输入telnet相关命令显示没有这种命令，centos7默认情况下不安装telnet命令，可选择手动安装



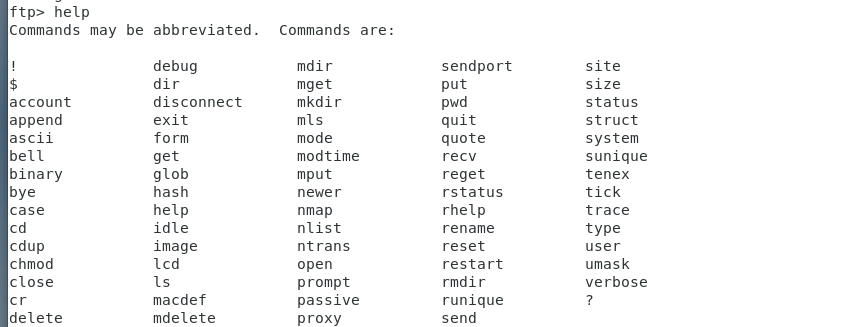
**ftp**：是一种网络协议，用于在网络上的计算机之间传输文件。FTP是互联网上使用最广泛的文件传输方法之一，它允许用户上传和下载文件，以及执行其他文件管理操作，如重命名、移动和删除文件



命令行输入ftp即可启动ftp客服端



ftp的命令：



输入bye 或 quit 可退出



**talk**

Centos7 中默认没有talk命令



网络命令：

1. telnet：用于远程登录其他主机。可以使用telnet 主机名来尝试远程登录。

②ftp：用于文件传输。可以使用ftp 主机名来连接到FTP服务器，然后进行文件上传和下载。

③talk：用于与其他用户进行实时聊天。可以使用talk 用户名来开始与另一个用户进行聊天。

3、实验记录

* 1. 在实验过程中，注意记录实验现象
  2. 使用过程体会、心得、技巧

1. 学会了使用which查看命令是否在目录中存在，例如在终端的命令行中输入which telnet 会显示命令找不到，可以先输入这个命令检查命令是否存在在进行操作。
2. 在不同的虚拟终端中执行不同的任务可以提高多任务处理效率。
3. ls命令可以快速找到特定的文件或目录。

ps命令可以快速找到特定的进程。

cat命令可以快速查看文件内容。

1. 学习vi的命令模式快捷键可以大大提高编辑效率，:wq保存并退出，:q!不保存强制退出
2. tar命令的-czvf选项可以创建一个压缩的归档文件，-xzvf可以解压。

gzip通常与tar结合使用，创建.tar.gz文件。

rpm命令的-qa可以列出所有已安装的包，-qf可以找出一个文件属于哪个包

实验3 Linux系统管理

**建议**在自己安装的Linux系统、华为云平台、OpenEuler等系统下操作使用。

**1、实验目的**

(1)掌握Linux基本系统管理命令

(2)掌握基本管理方法

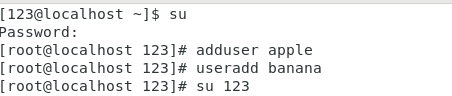
(3)初步了解Linux中与系统管理相关的有关配置文件

2、实验内容

(1) 用户管理

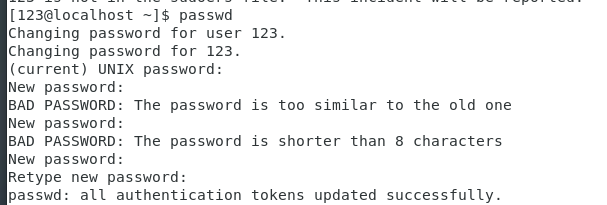
**adduser、useradd**：创建新用户

以下是创建apple和bannana的示例：



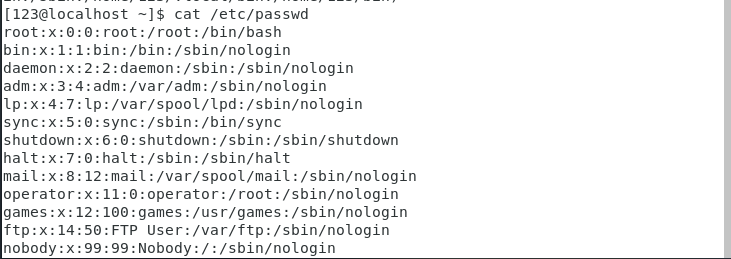


**passwd**：修改密码



**/etc/passwd** ：

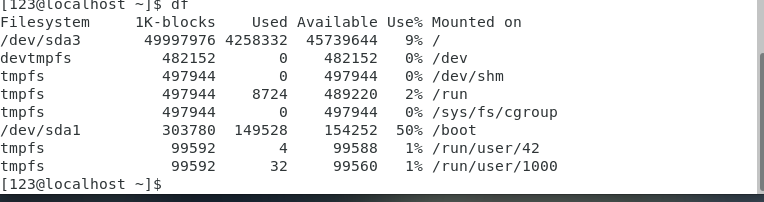
输入命令 cat /etc/passwd 查看所有用户



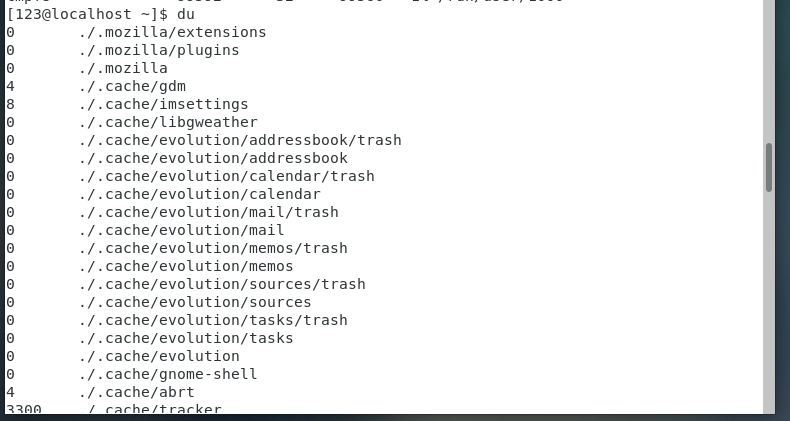


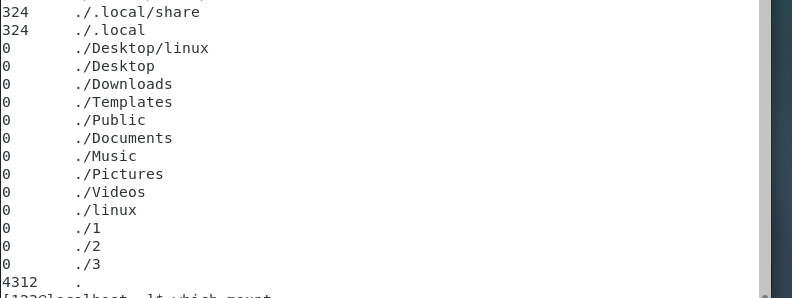
(2) 文件系统管理

**df**：检查文件系统的磁盘空间占用情况



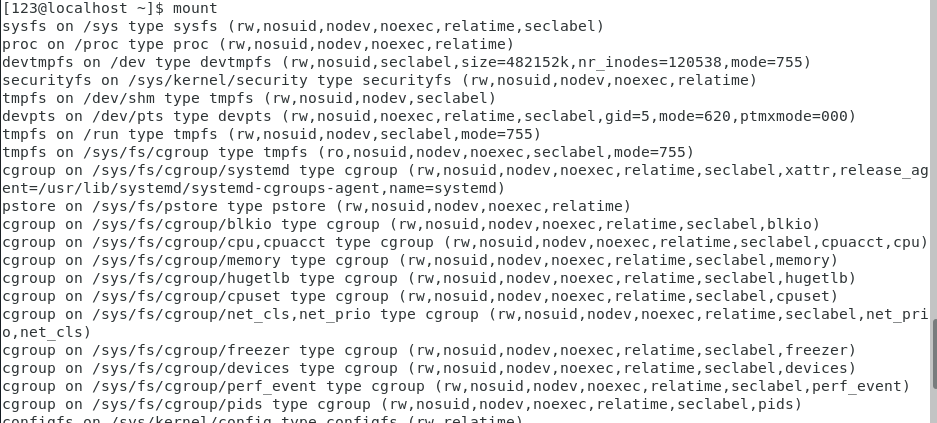
**du**：用于估计文件和目录在磁盘上所占空间



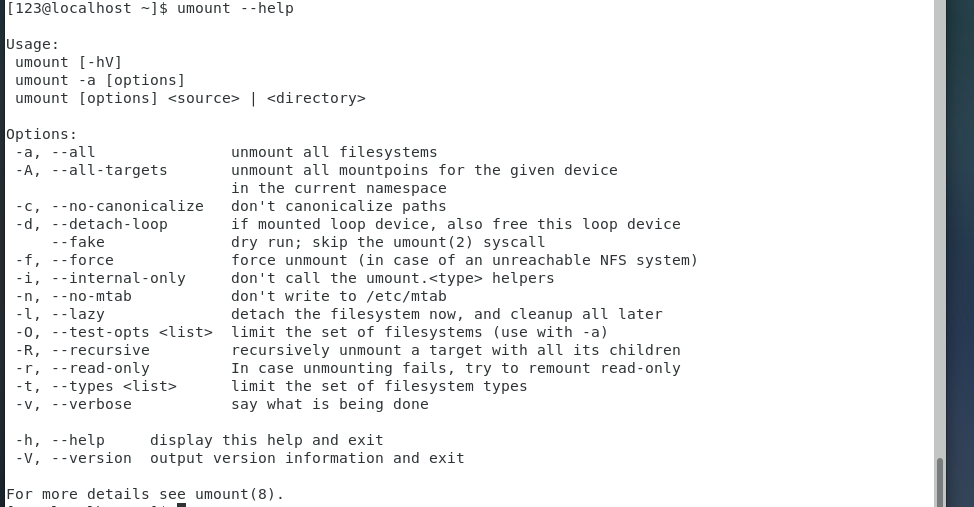


**mount**：中用于挂载文件系统。挂载是将存储设备（如硬盘、分区、USB 驱动器等）上的文件系统链接到系统目录树中的过程，使得你可以访问该存储设备上的文件。

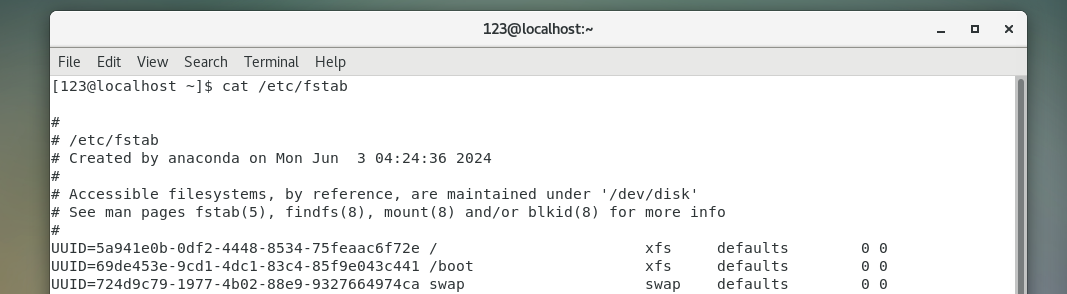
以下是查看所有挂载的文件系统的示例：

**umount**：用于卸载已经挂载的文件系统

以下是显示卸载帮助信息的示例：

命令使用：安装usb盘或者光盘，查看分析/etc/fstab文件

输入 cat /etc/fstab 查看文件内容



文件内容显示了三个主要的挂载点，这些挂载点在系统启动时会被自动挂载。下面是对这些挂载点的简要解释：

**根文件系统：**

设备标识：UUID=5a941e0b-0df2-4448-8534-75feaac6f72e

挂载点：/（根目录）

文件系统类型：xfs

挂载选项：defaults（使用默认挂载选项）

转储：0（不包括在备份中）

fsck顺序：0（在启动时不检查此文件系统）

**/boot 目录：**

设备标识：UUID=69de453e-9cd1-4dc1-83c4-85f9e043c441

挂载点：/boot（启动加载器和内核等的目录）

文件系统类型：xfs

挂载选项：defaults

转储：0

fsck顺序：0

**交换空间：**

设备标识：UUID=724d9c79-1977-4b02-88e9-9327664974ca

挂载点：swap（交换空间，用于虚拟内存）

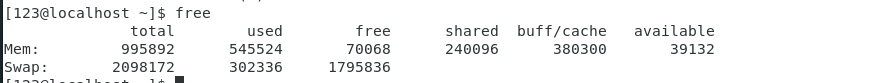
类型：swap（Linux 交换空间）

挂载选项：defaults

转储：0

fsck顺序：0

(3) 存储器

**free**：了解存储器使用情况

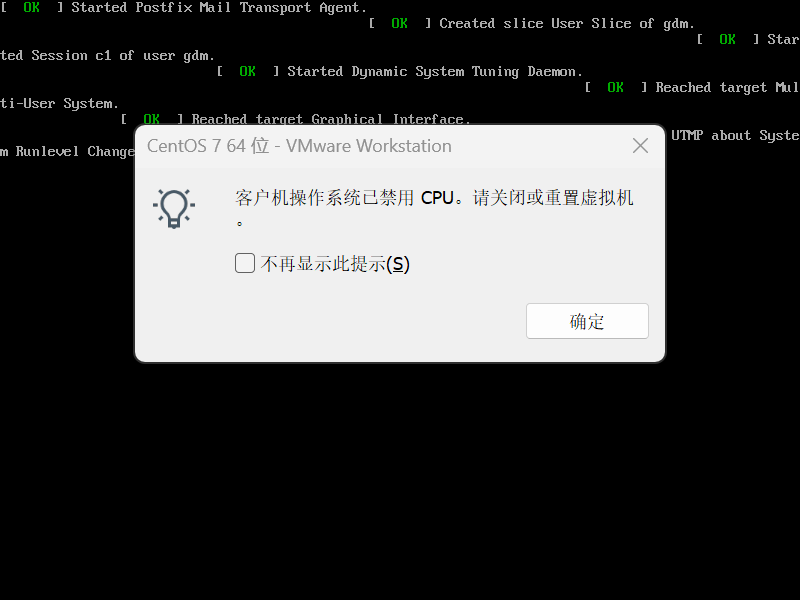
(4) 关机

halt、shutdown –h、init 0

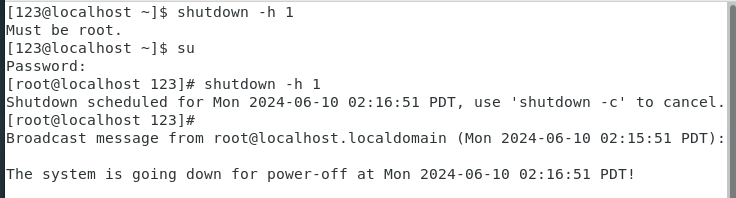
reboot、shutdown -r now

并比较它们之间的差别

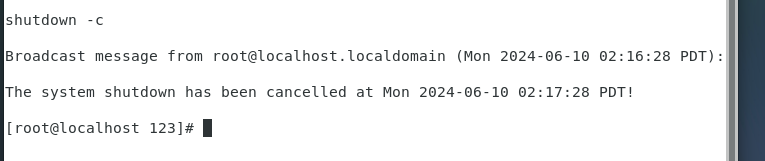
**Halt**：立即尝试停止所有进程并关闭系统



**Shutdown -h** ：可以设置时间关闭系统



输入shutdown -c 可以取消命令

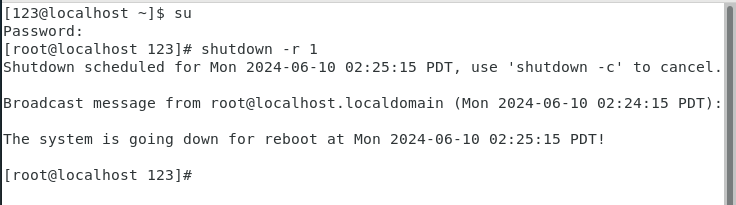


**init 0** ：将系统的运行级别改变到0，通常意味着关闭系统

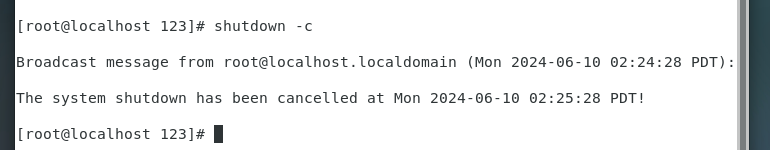


**reboot**  ：立即重启系统

**shutdown -r** ：指定时间重启



可输入shutdown -c取消



比较：

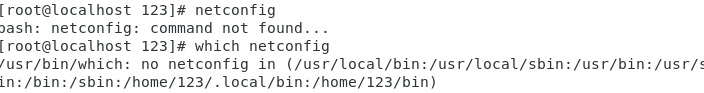
1. halt和shutdown -h now通常做相同的事情，即关闭系统，init 0与halt和shutdown -h now类似，用于关机，但它通过改变系统的运行级别来实现。
2. reboot和shutdown -r now用于重启系统，作用相同。

③都需要足够的权限，可以输入su进入 root权限，然后再使用这些命令

(5) 网络配置

netconfig的使用

输入netconfig命令，显示没有，使用which查询，显示目录下没有该命令



Centos7 虚拟机可使用nmcli 命令行工具来管理网络配置

输入命令查看所有的网络接口



创建一个新的网络连接

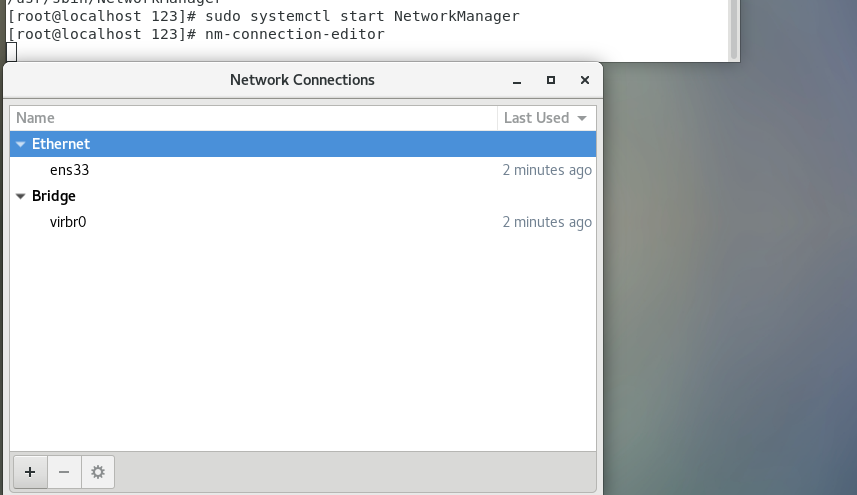
nmcli con modify ens33 ipv4.address '192.168.1.100/24'



也可以使用NetworkManager进行网络配置

输入命令启动服务：sudo systemctl start NetworkManager

输入命令启动图形工具： nm-connection-editor

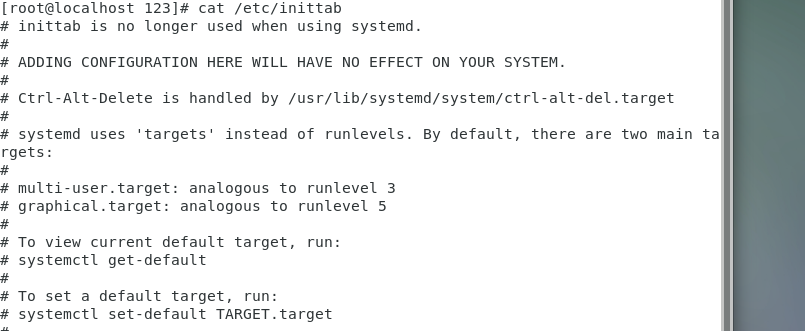


(6)启动配置

**/etc/inittab**：查看、修改并观察其响应

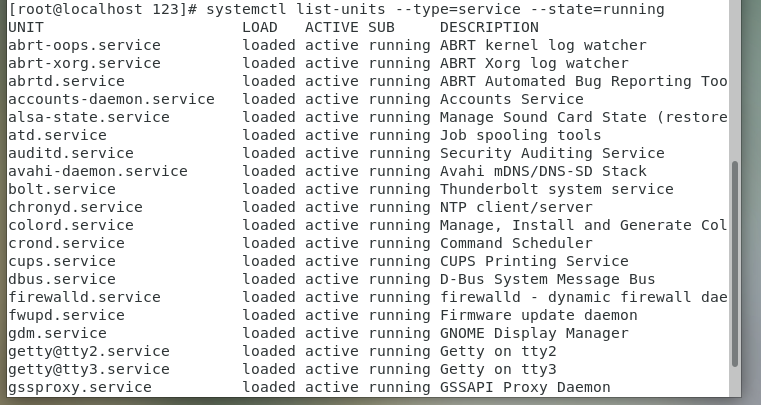


输入cat /etc/inittab 查看，显示/etc/inittab 文件已经不再被使用



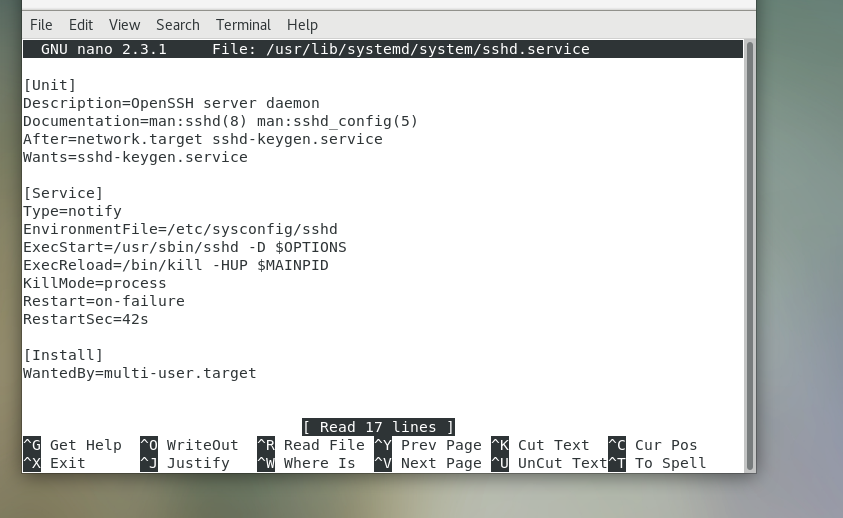
输入命令systemctl get-default显示系统当前使用的目标，可知启动配置不再通过/etc/inittab文件来管理。相反，systemd使用自己的配置文件来管理

查看服务状态: systemctl list-units --type=service --state=running



编辑辑服务配置文件，可以使用文本编辑器。例如，要编辑sshd.service服务，可以使用：

sudo nano /usr/lib/systemd/system/sshd.service



应用更改：

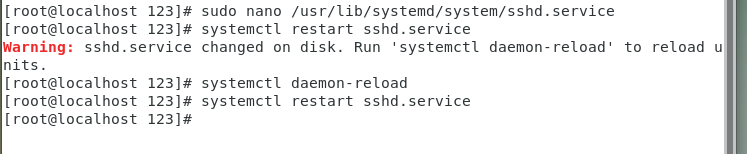
编辑完文件后，需要重新加载或重新启动服务或目标以应用更改。

例如，要重新加载sshd.service服务，可以使用：

systemctl daemon-reload

systemctl restart sshd.service

两个命令要一起使用



systemd使用自己的配置文件来管理服务、目标（targets）、挂载点（mounts）和单元（units）。

以下是使用systemd的一些基本步骤和概念：

①服务（services）: 服务是systemd中最重要的单元类型之一，它定义了系统启动时应该运行的进程和服务的配置。服务文件通常位于/usr/lib/systemd/system或/etc/systemd/system目录下。

②目标（targets）: 目标定义了一组服务的集合，这些服务在系统启动时应该一起启动或停止。例如，multi-user.target和graphical.target是两个主要的系统启动目标。

③挂载点（mounts）: 挂载点定义了在系统启动时应该挂载的文件系统。这些文件系统可以是本地硬盘上的分区，也可以是网络文件系统（NFS）等。

④单元（units）: 除了服务和目标之外，systemd还支持其他类型的单元，如设备单元（devices）、路径单元（paths）和socket单元（sockets）。

3、实验记录

(1)在实验过程中，注意记录实验现象   
(2)使用过程体会、心得、技巧

①halt和shutdown -h now通常做相同的事情，即关闭系统，init 0与halt和shutdown -h now类似，用于关机，但它通过改变系统的运行级别来实现。reboot和shutdown -r now用于重启系统，作用相同。这些命令都需要足够的权限，可以输入su进入 root权限，然后再使用这些命令

1. netconfig 命令在现代Linux系统中不再常用，它被NetworkManager 取代。可以使用 NetworkManager 图形界面或 nmcli 命令行工具来配置和管理网络连接。
2. /etc/inittab文件在现代Linux系统中已不再使用，它被 systemd取代。systemd 使用自己的配置文件来管理服务、目标、挂载点等。
3. 一些旧的命令或文件已经被新的所取代，需要了解新的命令及文件的使用方法来正确地对系统进行操作。

实验4 Linux Shell程序设计

**建议**在自己安装的Linux系统、华为云平台、OpenEuler等系统下操作使用。

**1、实验目的**

(1)掌握Linux shell程序运行方法

(2)掌握Linux Shell程序基本语法

(3)了解Linux环境变量

(3)编写简单Linux shell程序

2、实验内容

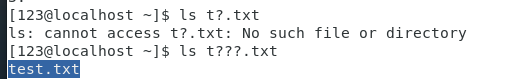
1. 查看/etc/.profile文件：相当于DOS下autoexe.bat
2. 通配符

\*: 匹配任意数量的字符



**?**：匹配任意单个字符

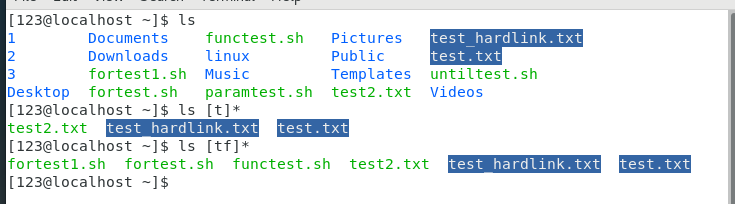
输入一个？，匹配不到test.txt，当输入？？？时，可匹配到test.txt，说明？只能匹配单个字符



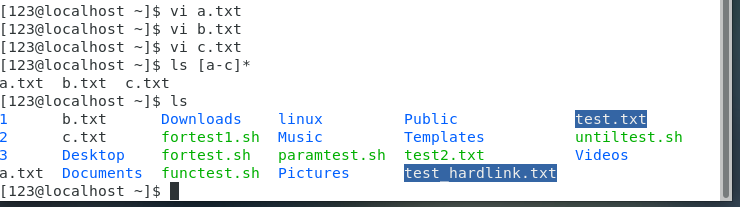
**[ ]**：匹配方括号内任意一个字符

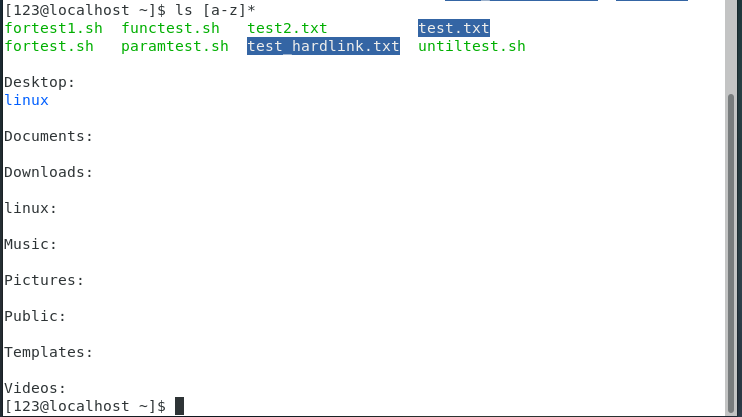
以[tf]\*为例，列出所有t或f开头的文件





$ls [a-c]\* ：列出当前目录下所有以 a、b 或 c 开头的文件和目录



$ls [a,m,t]\*：列出 a，m，t开头的文件



* 1. **重定向和管道的使用**

**输出重定向**：

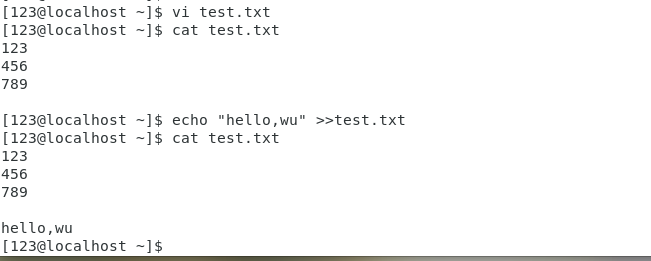
>：将输出重定向到文件，如果文件已存在，会被覆盖。

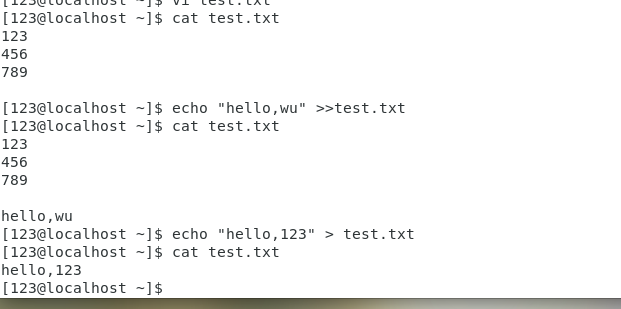
>>：将输出追加到文件末尾，而不是覆盖它。

2>：将错误输出重定向到文件。

&>：将标准输出和错误输出都重定向到同一个文件。

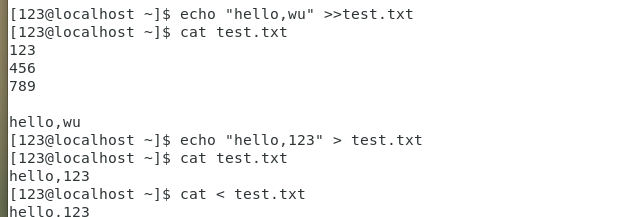
以下是追加和覆盖的示例：





**输入重定向：**

<：从文件读取输入。



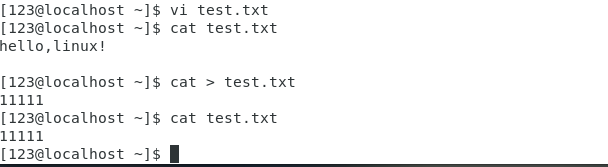
**管道**

管道（|）允许你将一个命令的输出作为另一个命令的输入。这是通过将前一个命令的标准输出连接到下一个命令的标准输入来实现的。

**ls | more**：分页显示当前目录下的文件和目录列表



cat > test.txt:创建一个test.txt文件，如果文件存在则会被清空并覆盖



(4) 变量

$lookup=/usr/mydir

$echo $lookup

设置变量，echo打印出变量，显示正确，说明lookup被成功设置



$export lookup：让进程使用

(5) 编辑并运行以下程序（shell程序控制结构）

1 fortest

#!/bin/bash

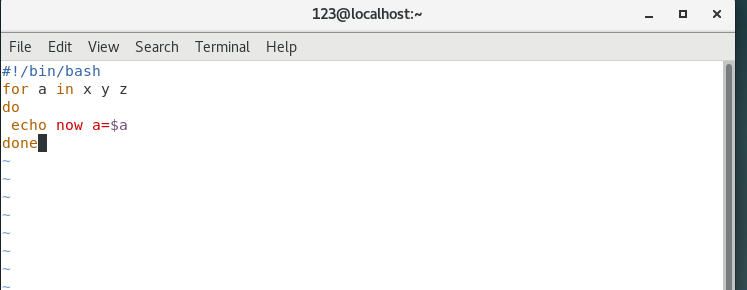
for a in x y z

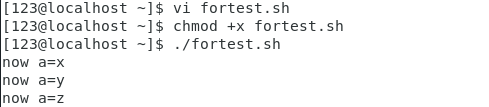
do

echo now a=$a

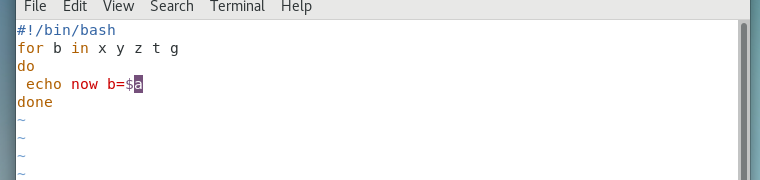
done

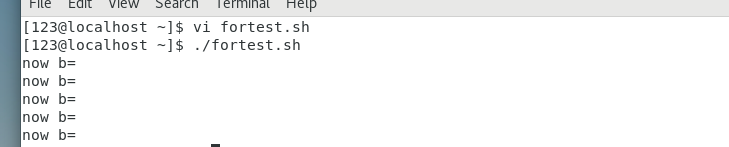
用vi创建fortest.sh,赋予执行权限，运行代码



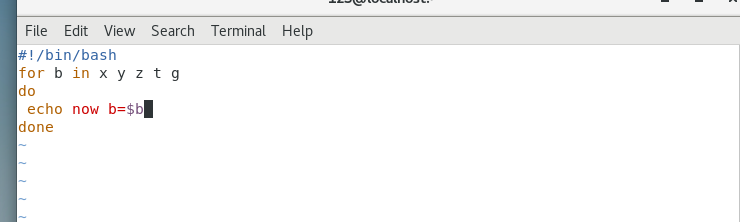


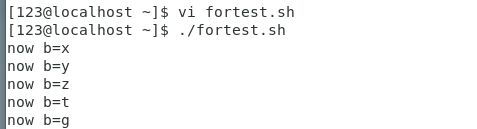
修改参数： b=$a





修改参数：b=$b





2 fortest1

#!/bin/bash

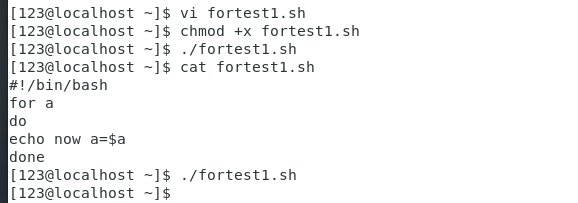
for a

do

echo now a=$a

done

用vi创建fortest1.sh，赋予执行权限，运行代码，但是脚本缺少迭代的元素，for循环不能正确执行，因此没有输出



3 functest

#!/bin/bash

setup()

{

echo setup…

}

do\_date()

{

date

}

chgdir()

{

cd $1

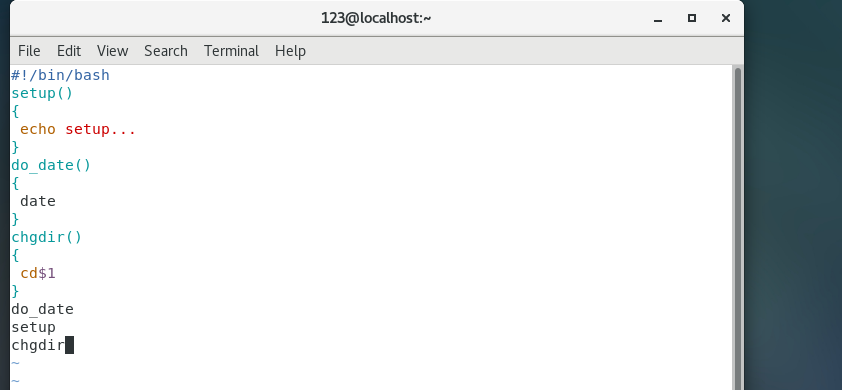
}

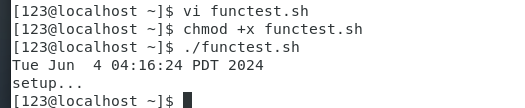
do\_date

setup

chgdir

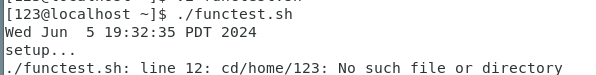
这个脚本定义了setuo，do\_date，chgdir函数，调用了这些函数，setup被调用时打印出setup… ，do\_date被调用时会打印当前日期和时间，chgdir接受一个参数（目标目录的路径）时，会尝试切换到该目录

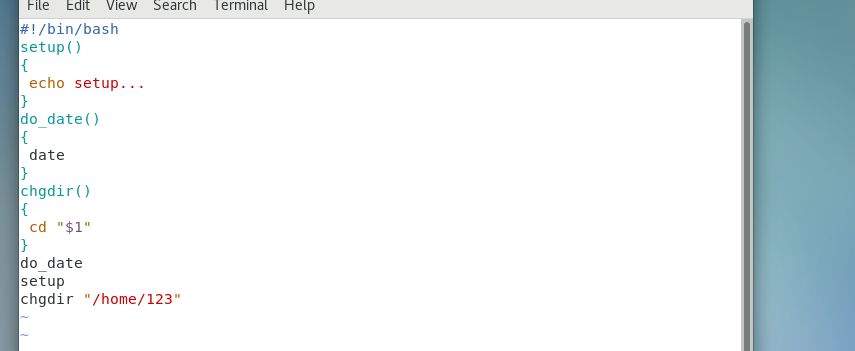


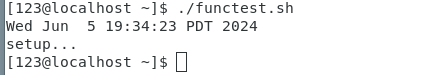


修改参数: chgdir “/home/123”

发现显示错误，研究发现是chgdir(){cd $1} 这个函数输入的参数与定义的输入参数格式不同，导致失败



修改cd $1为cd “$1”，发现可以，但是有无法判断是否转换成功，缺少直观显示，因此再次修改代码

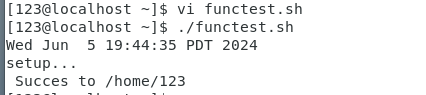


修改代码：

结果显示，切换目录成功

参数输入的缩进与设置的缩进非常影响结果





4 paramtest

#!/bin/bash

echo filename:$0

echo arguments:$\*

echo number arg:$#

echo arg2:$2

shift

echo number arg:$#

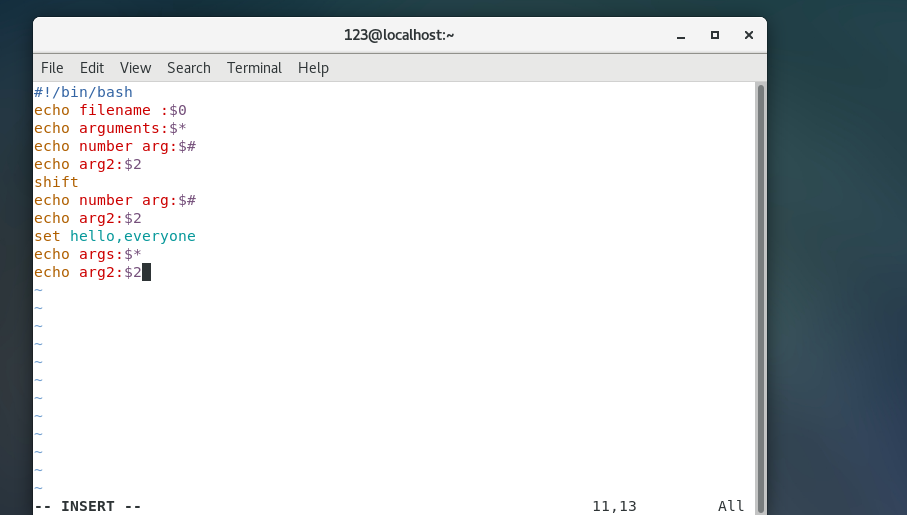
echo arg2:$2

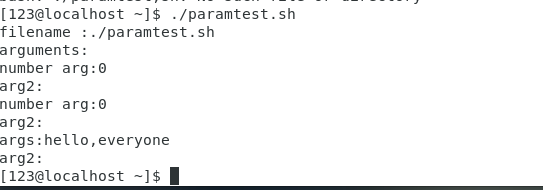
set hello,everone

echo args:$\*

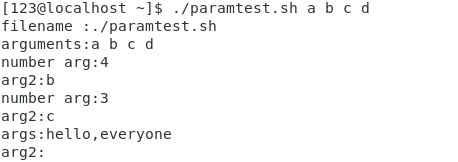
echo arg2:$2

运行原代码，部分显示为空，原因为没有设置参数

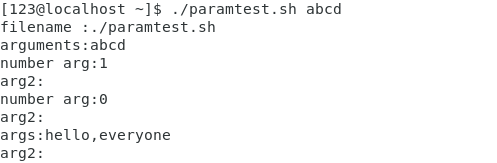




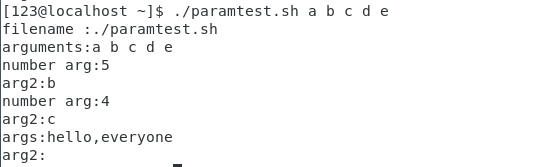
添加参数的输出：中间设置空格将参数分开



设置参数时未分开为一个参数



设置五个参数：

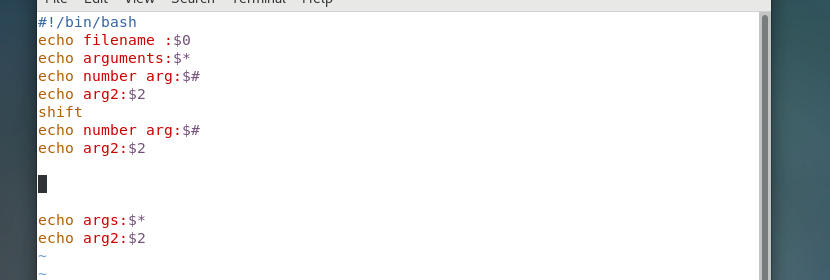


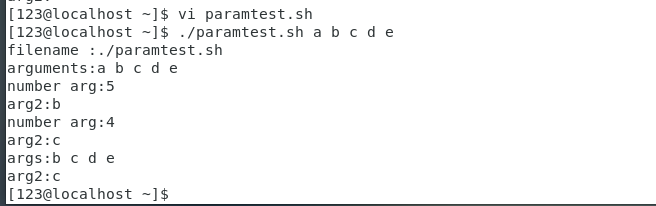
set行代码将 hello，everyone作为一个单独的参数传递到下面，使得args中只有一个参数，arg2的输出为空。（set 语句好像有错误）

修改代码：

删除set

后面两个echo可以正常输出，该代码用了左移一位，使得输入的五个参数显示减少一个





5 untiltest

#!/bin/bash

number=0

until (test $number –gt 5)

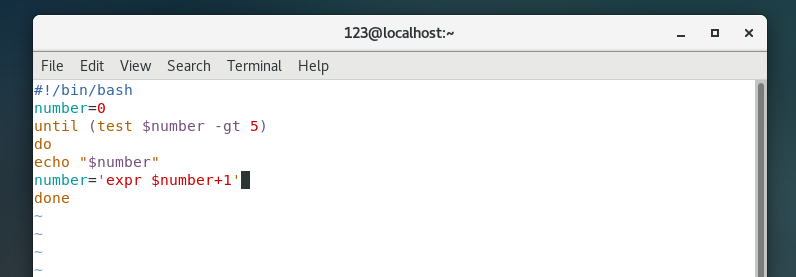
do

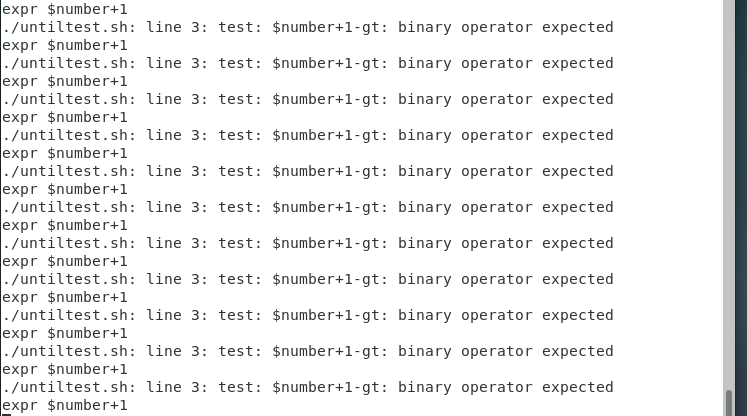
echo “ $number”

number=`expr $number + 1`

done

创建untiltest.sh文件，chmod +x 赋予文件可执行权限，运行脚本，





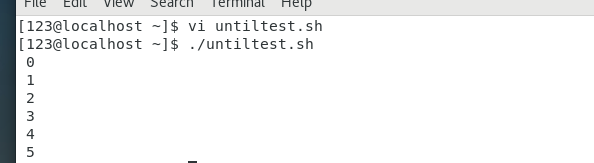
修改代码：

原代码想要实现number从0开始，每次循环number增加1，直到number大于5时停止，每次循环打印当前number的值

以下是对代码的修改：

number='expr $number + 1' 这行代码是错误的，因为它试图将字符串赋值给 number 变量，而不是执行算术运算。正确的方式是使用 ((number=number+1))





3、实验要求

1. 每个shell程序事例运行多遍，并给不同参数
2. 详细记录程序调试及运行结果

实验5 Linux 高级程序设计

**建议**在自己安装的Linux系统、华为云平台、OpenEuler等系统下操作使用。

**1、实验目的**

(1)了解Linux操作系统下应用程序开发流程

(2)掌握gun工具链的使用

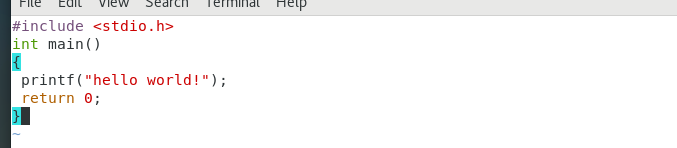
(3)了解Linux高级编程技巧（例如IPC机制、系统调用等）

2、实验内容

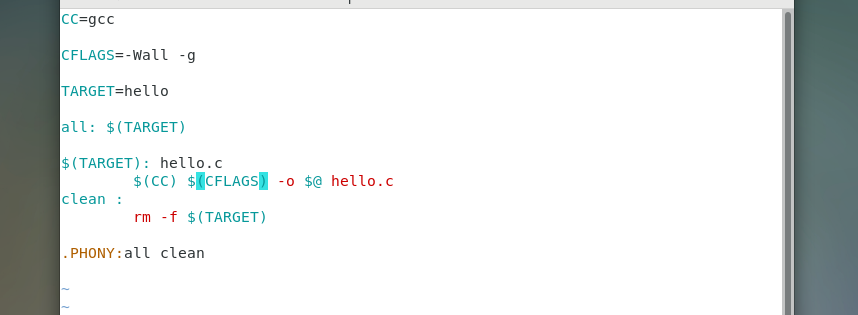
1. 编写一个简单的C语言程序，编写Makefile文件。了解编译过程，并用gdb进行调试。

**制表符Ctrl+v+tab**

创建hello.c



创建Makefile



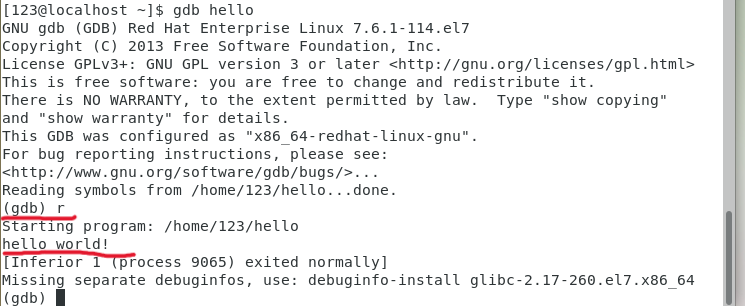
创建成功



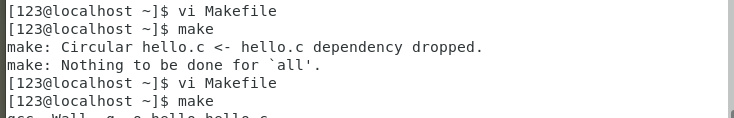
gdb r 运行程序输出hello world，运行成功

在gdb提示符下，可以使用以下命令：

* list 或 l：显示源代码。
* break 或 b：设置断点。
* run 或 r：运行程序。
* next 或 n：执行到下一行代码。
* step 或 s：步入函数。
* continue 或 c：继续执行。
* print 或 p：打印变量的值。
* quit 或 q：退出gdb。



创建失败：



1. 以下任选其一：
   1. 编写一个多进程通信程序，采用Message Queue或shared Memory或者Maped File机制进行通信
   2. 编写一个多线程程序(pthread)，实现2程同步互斥

C代码：

1. #include <stdio.h>
2. #include <pthread.h>
3. #include <unistd.h>
4. *// 互斥锁变量*
5. pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;
6. *// 共享资源*
7. int shared\_resource = 0;
8. *// 线程函数*
9. void\* thread\_function(void\* arg) {
10. int thread\_id = \*((int\*)arg);
11. for (int i = 0; i < 5; i++) {
12. *// 锁定互斥锁*
13. pthread\_mutex\_lock(&mutex);
14. *// 临界区开始*
15. shared\_resource++;
16. printf("Thread %d: shared\_resource = %d\n", thread\_id, shared\_resource);
17. *// 临界区结束*
18. *// 释放互斥锁*
19. pthread\_mutex\_unlock(&mutex);
20. *// 模拟一些工作*
21. sleep(1);
22. }
23. return NULL;
24. }
25. int main() {
26. pthread\_t thread1, thread2;
27. int arg1 = 1, arg2 = 2;
28. *// 创建线程*
29. pthread\_create(&thread1, NULL, thread\_function, &arg1);
30. pthread\_create(&thread2, NULL, thread\_function, &arg2);
31. *// 等待线程结束*
32. pthread\_join(thread1, NULL);
33. pthread\_join(thread2, NULL);
34. *// 销毁互斥锁*
35. pthread\_mutex\_destroy(&mutex);
36. return 0;
37. }

创建 thread\_sync.c文件，写入代码

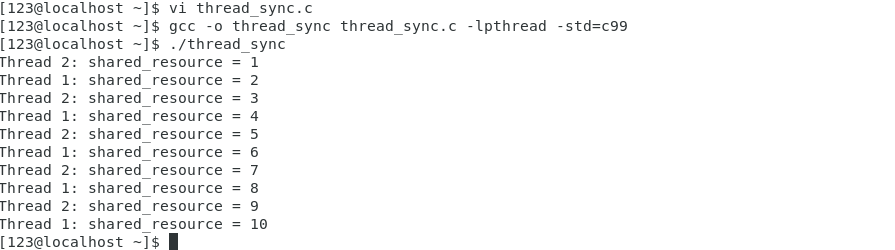


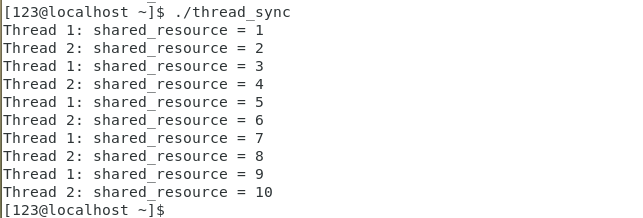
在终端输入命令gcc -o thread\_sync thread\_sync.c -lpthread -std=c99，以创建可执行程序

输入./thread\_sync以执行程序

以下是两次测试的结果：

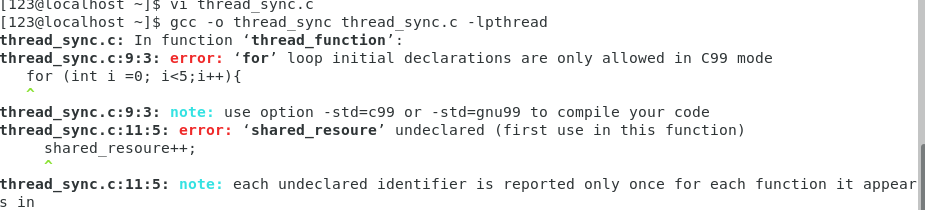
可见线程1 ，线程2同步互斥





遇到的问题:

①第一次输入命令gcc -o thread\_sync thread\_sync.c -lpthread 命令时，创建可执行文件失败，查询资料才知道在C99模式下才允许在for循环的初始化部分声明变量。要解决这个问题，可以在编译时添加-std=c99或-std=gnu99选项，以启用C99标准。②变量名拼写错误，shared\_resoure应该是shared\_resource，输入的时候没注意



3、实验要求

* 1. 写出源程序，并编译运行
  2. 详细记录程序调试及运行结果

实验6 Linux内核

1、实验目的

(1)了解Linux操作系内核

(2)掌握内核编译和内核升级方法

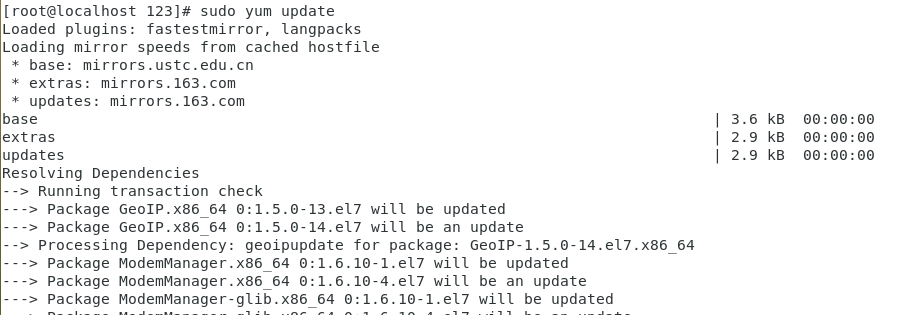
(3)了解有关内核编程

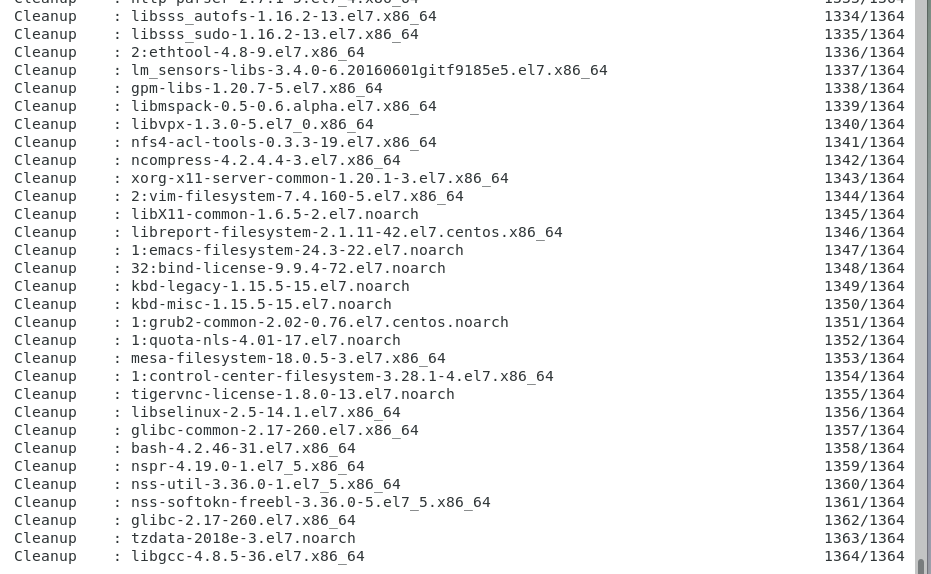
2、实验内容

1. 内核配置和编译（课后完成）
2. 更新系统

输入su 进入管理者模式

输入 sudo yum update



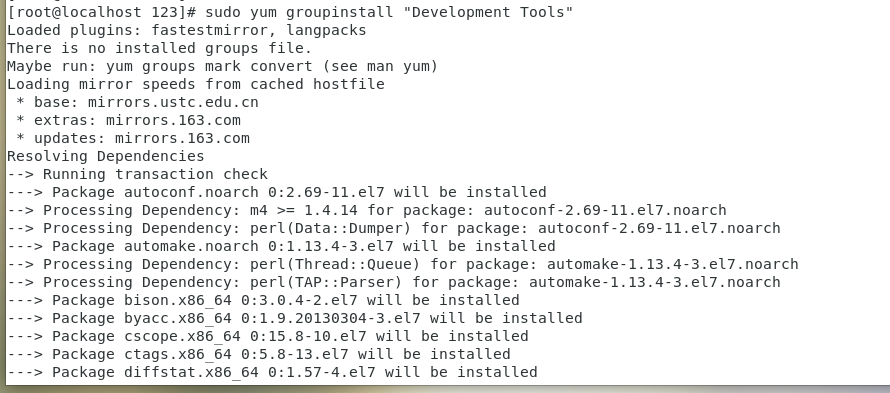


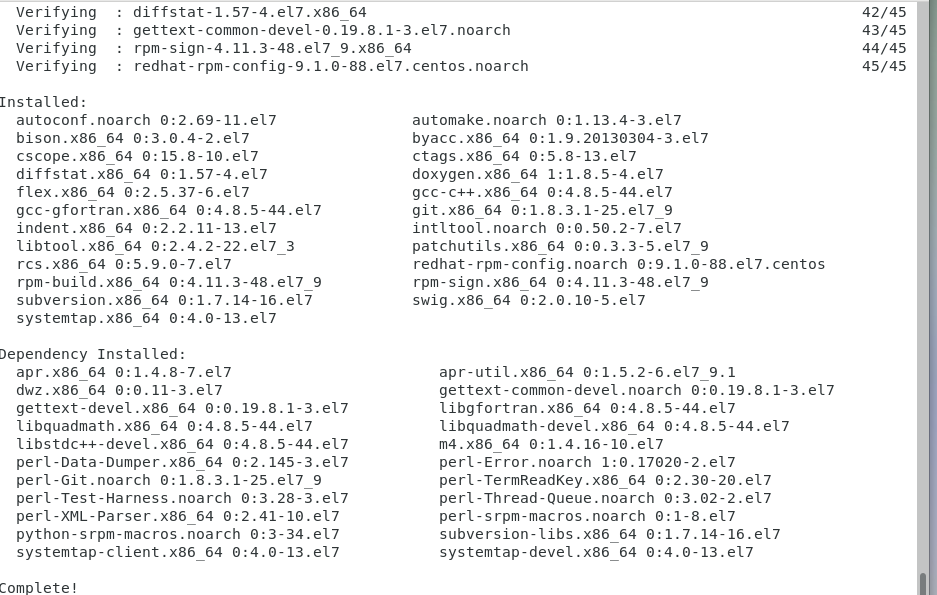
更新完成：



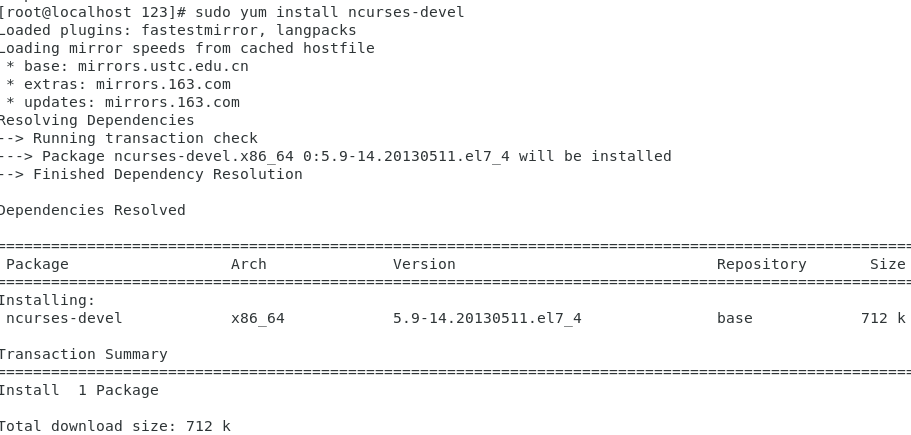
1. 安装编译内核所需的依赖包

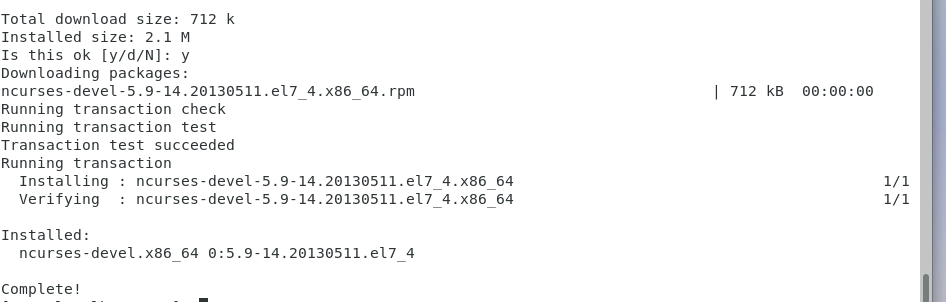
sudo yum groupinstall "Development Tools"



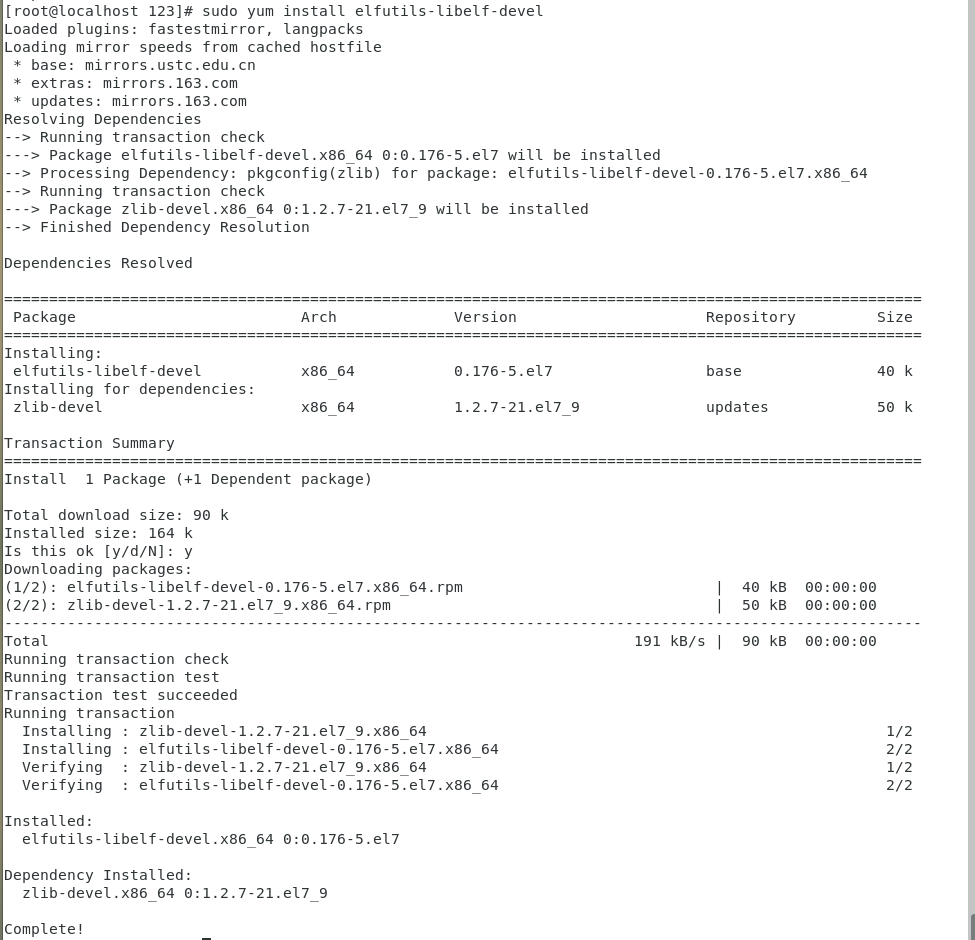


sudo yum install ncurses-devel





sudo yum install elfutils-libelf-devel



1. 获取内核源码

#输入命令获取内核源码包

sudo yum install kernel-devel

获取最新内核

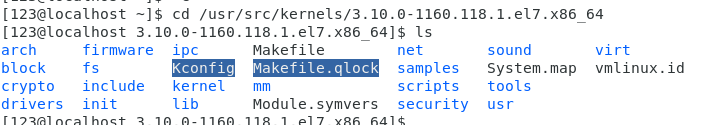


查看内核源码;

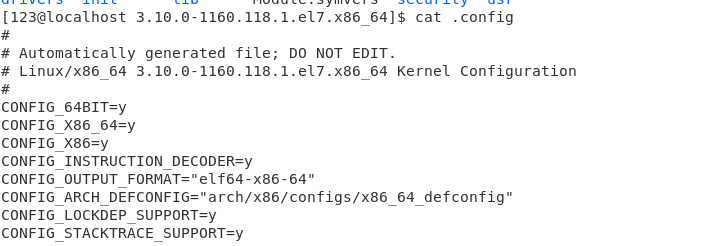
ls /usr/src/kernels/



输入内核版本号进入其中一个内核目录

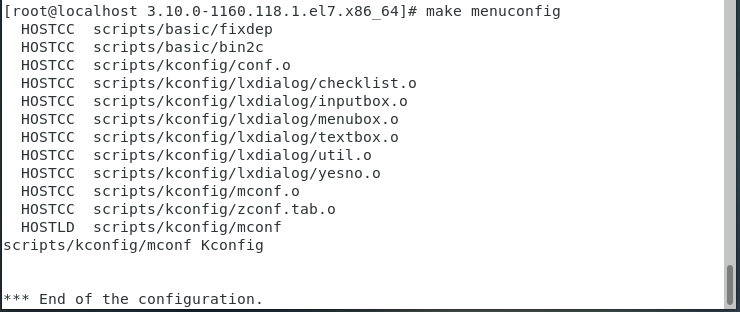


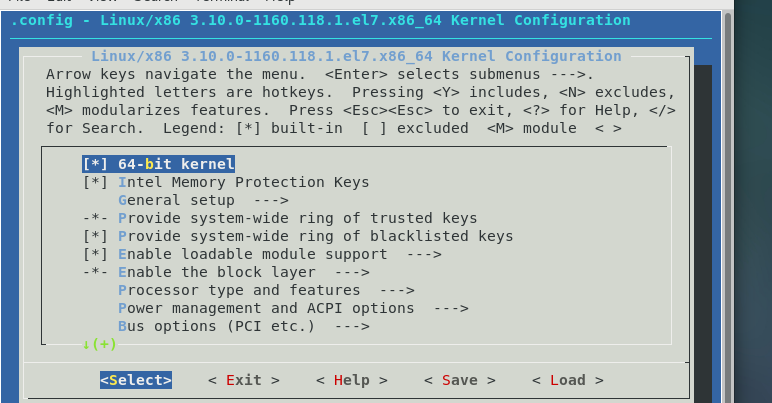
查看内核配置：



图形化地查看内核配置：

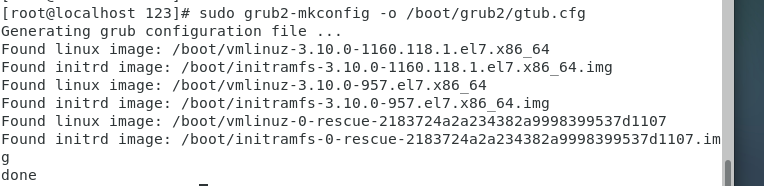
make menuconfig



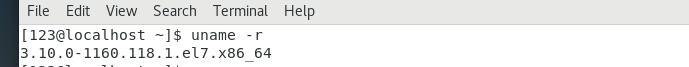


**更新引导加载器**：

sudo grub2-mkconfig -o /boot/grub2/grub.cfg



重启系统，选择新内核，查看内核号，可以看出已应用新内核

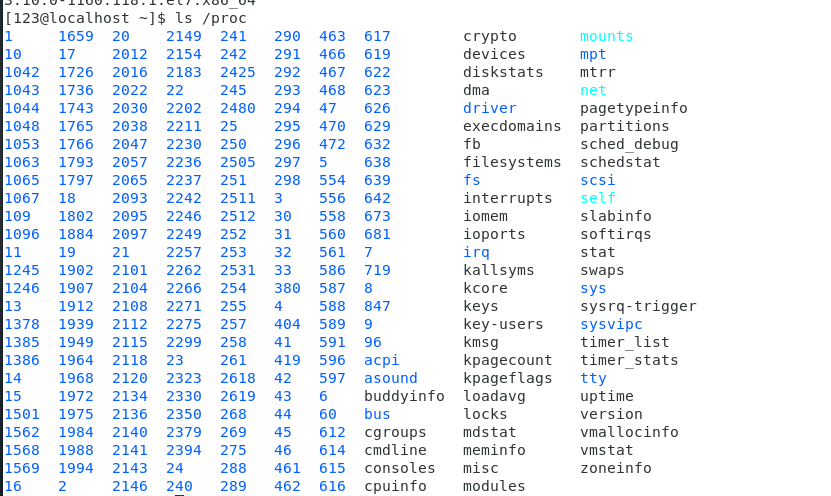


1. 察看/proc内存文件系统下的文件，写出每个目录内容

查看/proc下的所有目录和文件：

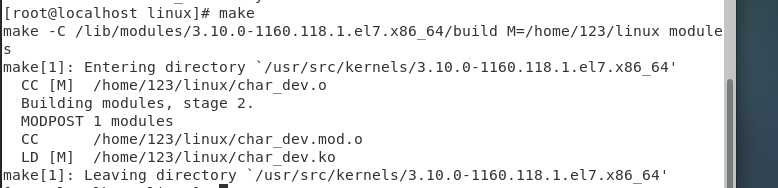
ls /proc

1. **数字**: 这些是进程ID，每个数字代表一个正在运行的进程。例如， /proc/1 是PID为1的进程的信息。
2. **cpuinfo**: 包含有关CPU的信息，如型号、数量、速度等。
3. **meminfo**: 提供系统的内存使用情况，包括物理内存、交换空间等。
4. **version**: 显示Linux内核的版本信息。
5. **cmdline**: 显示启动当前内核的命令行参数。
6. **uptime**: 显示系统已经运行的时间和系统的负载。
7. **loadavg**: 显示系统的平均负载。
8. **stat**: 包含系统的统计信息，如进程创建、上下文切换等。
9. **kmsg**: 包含内核的消息。
10. **cgroups**: 显示控制组（cgroups）的相关信息。
11. **filesystems**: 显示系统支持的文件系统类型。
12. **partitions**: 显示当前系统中的块设备分区。
13. **interrupts**: 显示中断统计信息。
14. **softirqs**: 显示软中断统计信息。
15. **diskstats**: 显示磁盘统计信息。
16. **buddyinfo**: 显示内存分配器的伙伴系统状态。
17. **kpagecount**: 显示每个页面大小类别的页面数量。
18. **kpageflags**: 显示页面的分配状态。
19. **schedstat**: 显示调度器的统计信息。
20. **sched\_debug**: 包含调度器调试信息。
21. **locks**: 显示文件锁的信息。
22. **mdstat**: 显示多磁盘（RAID）设备的统计信息。
23. **modules**: 列出已加载的内核模块。
24. **bus**: 显示系统中的总线信息。
25. **devices**: 显示系统中的设备信息。
26. **drivers**: 显示加载的驱动程序列表。
27. **iomem**: 显示I/O内存资源。
28. **ioports**: 显示I/O端口资源。
29. **tty**: 显示终端设备的信息。
30. **fb**: 显示帧缓冲设备的信息。
31. **asound**: 显示声音子系统的信息。
32. **crypto**: 显示加密设备的状态。
33. **net**: 显示网络接口和协议统计信息。
34. **mpt**: 显示MPT（LSI FC909/FC929/FC919/FC921/FC922）设备的统 计信息。
35. **sys**: 包含系统调用的信息。
36. **sysrq-trigger**: 显示系统请求（sysrq）触发器的状态。
37. **timer\_list**: 显示当前活动的定时器列表。
38. **timer\_stats**: 显示定时器的统计信息。
39. **acpi**: 显示ACPI（高级配置和电源接口）的信息。
40. **vmallocinfo**: 显示虚拟内存分配的信息。
41. **zoneinfo**: 显示内存区域的信息。



* 1. 编写一个简单的字符虚拟设备程序（以模块方式编译、加载、使用）

make生成char\_dev.ko 文件

加载模块：

insmod char\_dev.ko



检查模块是否加载：

lsmod | grep char\_dev

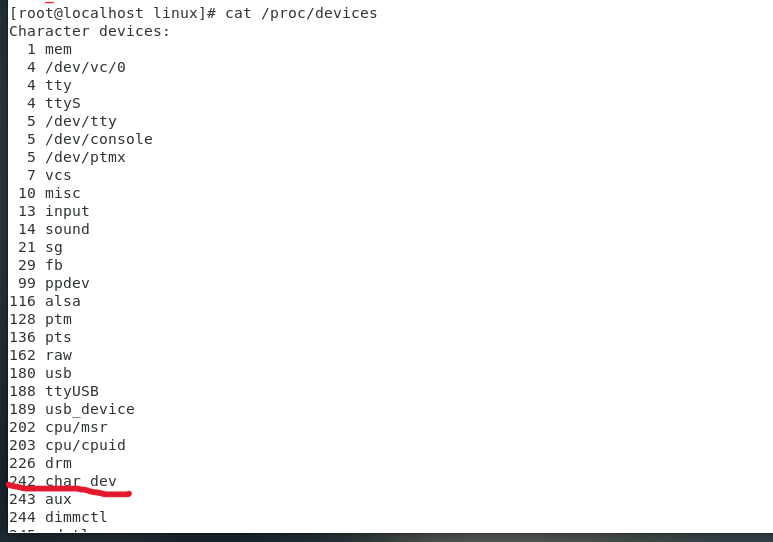
可以看到模块正确加载



在 /dev 目录下创建设备文件：

sudo mknod /dev/char\_dev c 0 0

输入cat /proc/devices 查询主设备号，可以看到是242



创建设备文件:

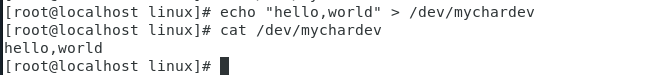
sudo mknod /dev/mychardev c 242 0

创建一个名为 mychardev 的字符设备文件，并且你的设备驱动程序使用的主设备号是 242，次设备号是 0。



测试输入:

echo "hello, world" > /dev/mychardev



再次测试：

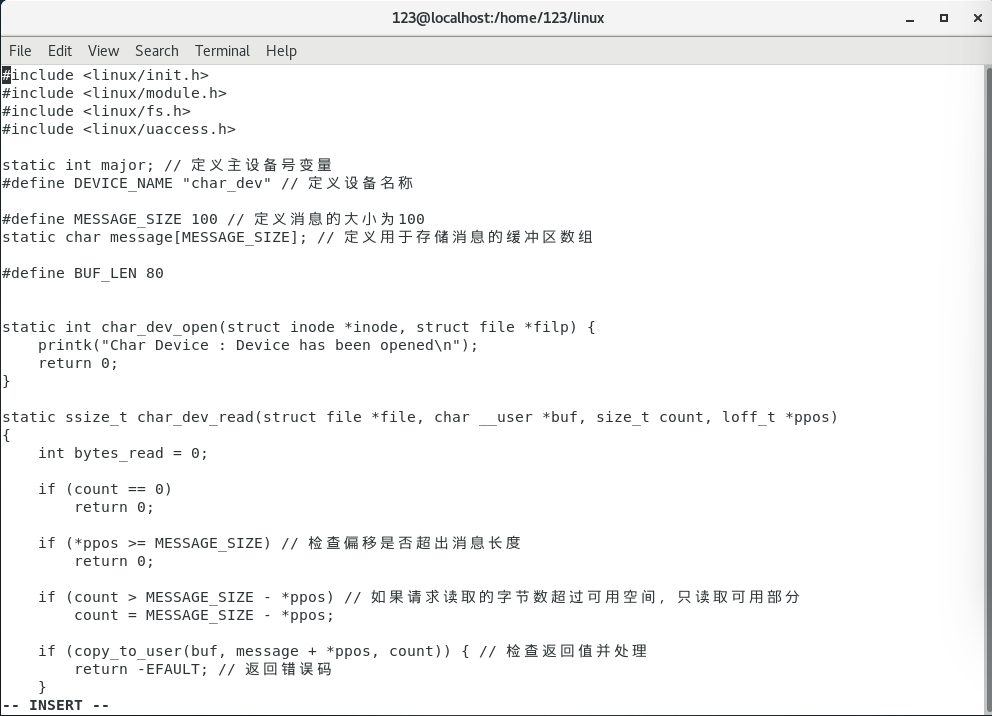
echo 'hello , linux ! ' > /dev/mychardev

可以正确显示输入的字符



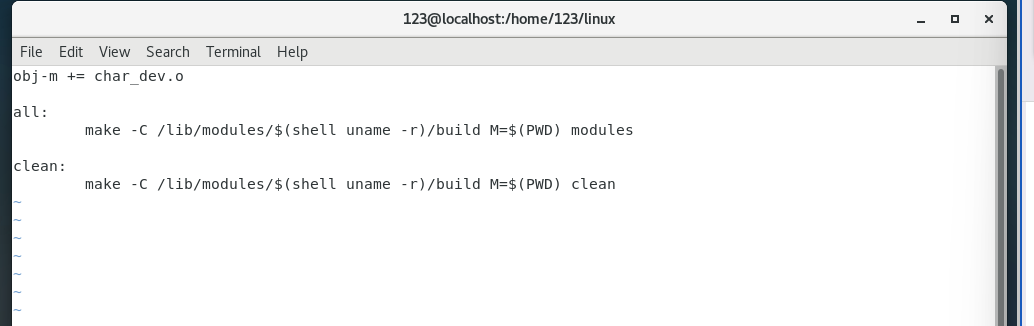
**源代码：**

char\_dev.c



1. #include <linux/init.h>
2. #include <linux/module.h>
3. #include <linux/fs.h>
4. #include <linux/uaccess.h>
5. static int major; *// 定义主设备号变量*
6. #define DEVICE\_NAME "char\_dev" *// 定义设备名称*
7. #define MESSAGE\_SIZE 100 *// 定义消息的大小为100*
8. static char message[MESSAGE\_SIZE]; *// 定义用于存储消息的缓冲区数组*
9. #define BUF\_LEN 80
10. static int char\_dev\_open(struct inode \*inode, struct file \*filp) {
11. printk("Char Device : Device has been opened\n");
12. return 0;
13. }
14. static ssize\_t char\_dev\_read(struct file \*file, char \_\_user \*buf, size\_t count, loff\_t \*ppos)
15. {
16. int bytes\_read = 0;
17. if (count == 0)
18. return 0;
19. if (\*ppos >= MESSAGE\_SIZE) *// 检查偏移是否超出消息长度*
20. return 0;
21. if (count > MESSAGE\_SIZE - \*ppos) *// 如果请求读取的字节数超过可用空间，只读取可用部分*
22. count = MESSAGE\_SIZE - \*ppos;
23. if (copy\_to\_user(buf, message + \*ppos, count)) { *// 检查返回值并处理*
24. return -EFAULT; *// 返回错误码*
25. }
26. \*ppos += count;
27. bytes\_read = count;
28. return bytes\_read;
29. }
30. static ssize\_t char\_dev\_write(struct file \*file, const char \_\_user \*buf, size\_t count, loff\_t \*ppos)
31. {
32. int bytes\_written = 0;
33. if (count == 0)
34. return 0;
35. if (\*ppos >= MESSAGE\_SIZE) *// 检查偏移是否超出消息长度*
36. return -ENOSPC; *// 返回空间不足错误码*
37. if (count > MESSAGE\_SIZE - \*ppos) *// 如果写入的字节数超过可用空间，只写入可用部分*
38. count = MESSAGE\_SIZE - \*ppos;
39. if (copy\_from\_user(message + \*ppos, buf, count)) { *// 检查返回值并处理*
40. return -EFAULT; *// 返回错误码*
41. }
42. \*ppos += count;
43. bytes\_written = count;
44. return bytes\_written;
45. }
46. static int char\_dev\_release(struct inode \*inode, struct file \*filp) {
47. printk("Char Device: Device has been closed\n");
48. return 0;
49. }
50. static struct file\_operations char\_dev\_fops = {
51. .owner = THIS\_MODULE,
52. .read = char\_dev\_read,
53. .write = char\_dev\_write,
54. .open = char\_dev\_open,
55. .release = char\_dev\_release
56. };
57. static int \_\_init char\_dev\_init(void) {
58. major = register\_chrdev(0, DEVICE\_NAME, &char\_dev\_fops);
59. printk("Char Device : Device has been registered with major number %d\n", major);
60. return 0;
61. }
62. static void \_\_exit char\_dev\_exit(void) {
63. unregister\_chrdev(major, DEVICE\_NAME);
64. printk("Char Device : Device has been unregistered\n");
65. }
66. module\_init(char\_dev\_init);
67. module\_exit(char\_dev\_exit);
68. MODULE\_LICENSE("GPL");
69. MODULE\_AUTHOR("Your Name");
70. MODULE\_DESCRIPTION("A simple character device driver");

Makefile



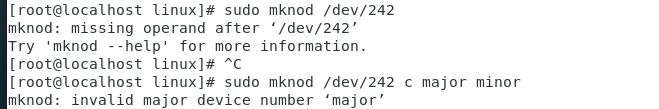
1. obj-m += char\_dev.o
2. all:
3. make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
4. clean:
5. make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

遇到的问题:

输入make命令创建编译模块显示错误，查询发现是Makefile文件的缩进有问题，修改Makefile文件，可以正确运行



创建设备文件命令不理解，通过查询资料知道正确的输入方式



* 1. 内核模块

已经在(3)中进行实现

内核模块（Kernel Module）是用于Linux内核的动态可加载代码块，它们可以在运行时被加载或卸载，而不需要重新启动系统.

内核模块的加载：sudo insmod 模块文件路径.ko

内核模块的卸载：sudo rmmod 模块名称

内核模块的查看：lsmod

编写内核模块的基本步骤：

①编写模块代码: 使用C语言编写内核模块代码，通常遵循内核编程的规范。

②编译模块: 使用 make 命令和适当的内核配置文件（通常是 .config 文件）来编译模块。

③安装模块: 将编译好的模块文件（通常是 .ko 文件）复制到 /lib/modules/内核版本/kernel/ 目录下。

④加载模块: 使用上述提到的 insmod命令来加载模块。

3、实验要求

* + 1. 给出源程序
    2. 记录调试过程和运行结果