****

Linux应用实践

**实验指导书**

**刘斯韵**

**15677061086**

**87184205**

**桂林电子科技大学**

**计算机与信息安全学院**

**2021.3**

目录

[实验一、 Linux常用命令（4学时） 1](#_Toc39162186)

[1、 实验目的： 1](#_Toc39162187)

[2、 实验内容： 1](#_Toc39162188)

[(1) 练习ls命令 1](#_Toc39162189)

[(2) 练习cd命令 2](#_Toc39162190)

[(3) 练习cp 命令 2](#_Toc39162191)

[(4) 练习rm命令 2](#_Toc39162192)

[(5) 练习mv命令 2](#_Toc39162193)

[(6) 熟悉find命令 2](#_Toc39162194)

[(7) 熟悉mkdir命令 3](#_Toc39162195)

[(8) 熟悉rmdir 命令 3](#_Toc39162196)

[3、 实验结果和问题分析： 4](#_Toc39162197)

[实验二、 vi编辑器的使用（4学时） 5](#_Toc39162198)

[1、 实验目的： 5](#_Toc39162199)

[2、 实验内容： 5](#_Toc39162200)

[(1) vi编辑器的启动和退出 5](#_Toc39162201)

[(2) 熟悉vi三种模式 5](#_Toc39162202)

[(3) 实验文件保存 5](#_Toc39162203)

[(4) 熟悉命令行模式 6](#_Toc39162204)

[3、 实验结果和问题分析： 7](#_Toc39162205)

[实验三、 服务器的配置与管理（4学时） 8](#_Toc39162206)

[1、 实验目的： 8](#_Toc39162207)

[2、 实验内容： 8](#_Toc39162208)

[(1) SSH网络服务器的配置 8](#_Toc39162209)

[(2) Apache Web服务器的配置 8](#_Toc39162210)

[(3) NFS服务器 8](#_Toc39162211)

[(4) DHCP服务器的配置 9](#_Toc39162212)

[3、 实验结果和问题分析： 9](#_Toc39162213)

[实验四、 shell编程（4学时） 10](#_Toc39162214)

[1、 实验目的： 10](#_Toc39162215)

[2、 shell相关知识： 10](#_Toc39162216)

[(1) 基础知识 10](#_Toc39162217)

[(2) 变量 10](#_Toc39162218)

[(3) shell 字符串 12](#_Toc39162219)

[(4) Shell 数组 13](#_Toc39162220)

[(5) Shell 注释 14](#_Toc39162221)

[(6) Shell 传递参数 15](#_Toc39162222)

[3、 实验内容： 17](#_Toc39162223)

[4、 实验结果和问题分析： 17](#_Toc39162224)

[实验五、 C语言开发工具（4学时） 18](#_Toc39162225)

[1、 实验目的： 18](#_Toc39162226)

[2、 实验内容： 18](#_Toc39162227)

[(1) gcc编译 18](#_Toc39162228)

[(2) gdb调试 18](#_Toc39162229)

[(3) makefile编写 19](#_Toc39162230)

[(4) 编写程序 20](#_Toc39162231)

[3、 实验结果和问题分析： 20](#_Toc39162232)

[实验六、 文件系统编程（4学时） 21](#_Toc39162233)

[1、 实验目的： 21](#_Toc39162234)

[2、 基础知识： 21](#_Toc39162235)

[(1) 文件系统 21](#_Toc39162236)

[(2) 系统调用 21](#_Toc39162237)

[(3) 标准I/O库函数 21](#_Toc39162238)

[(4) API 21](#_Toc39162239)

[(5) 常见的文件类型 21](#_Toc39162240)

[(6) 文件操作的过程 22](#_Toc39162241)

[(7) 系统调用API接口 22](#_Toc39162242)

[(8) 标准I/O库函数API接口 24](#_Toc39162243)

[3、 实验内容： 27](#_Toc39162244)

[4、 实验结果和问题分析： 27](#_Toc39162245)

[实验七、 综合实例（4学时） 28](#_Toc39162246)

[1、 实验目的： 28](#_Toc39162247)

[2、 实验内容： 28](#_Toc39162248)

[3、 实验结果和问题分析： 28](#_Toc39162249)

[实验八、 综合设计实践（4学时） 29](#_Toc39162250)

[1、 实验目的： 29](#_Toc39162251)

[2、 实验内容： 29](#_Toc39162252)

[(1) 初级题目 29](#_Toc39162253)

[(2) 中级题目 29](#_Toc39162254)

[(3) 高级题目 29](#_Toc39162255)

[3、 实验结果和问题分析： 30](#_Toc39162256)

# Linux常用命令（4学时）

## 实验目的：

1. 熟悉Linux系统环境；
2. 常用Linux shell命令练习。

## 实验内容：

本次实验为第一次实验，要求完成本次实验所有内容。本课程有推荐使用华为的EulerOS或CentOS，也可使用RedHat的系统或Ubuntu。课程中会同时使用yum和apt进行讲解。希望同学们可以同时搭建起yum或apt。本次实验具体实验内容如下：

### 练习ls命令

作用：展示目录中内容。参数如下：

|  |  |
| --- | --- |
| **参数** | **作用** |
| -A | 全部的档案，连同隐藏档，但不包括 . 与 .. 这两个目录，一起列出来～ |
| -d | 仅列出目录本身，而不是列出目录内的档案数据 |
| -f | 直接列出结果，而不进行排序 (ls 预设会以档名排序！) |
| -F | 根据档案、目录等信息，给予附加数据结构，例如： |
| \* | 代表可执行档 /：代表目录 =：代表 socket 档案 |：代表 FIFO 档案 |
| -h | 将档案容量以人类较易读的方式(例如 GB, KB 等等)列出来 |
| -i | 列出 inode 位置，而非列出档案属性 |
| -l | 长数据串行出，包含档案的属性等等数据 |
| -n | 列出 UID 与 GID 而非使用者与群组的名称 (UID与GID会在账号管理提到！) |
| -r | 将排序结果反向输出，例如：原本档名由小到大，反向则为由大到小 |
| -R | 连同子目录内容一起列出来 |
| -S | 以档案容量大小排序！ |
| -t | 依时间排序 |
| --color=never | 不要依据档案特性给予颜色显示 |
| --color=always | 显示颜色 |
| --color=auto | 让系统自行依据设定来判断是否给予颜色 |
| --full-time | 以完整时间模式 (包含年、月、日、时、分) 输出 |
| --time={atime,ctime} | 输出 access 时间或 改变权限属性时间 (ctime) 而非内容变更时间 (modification time) |

例如：

ls [-aAdfFhilRS] 目录名称

ls [--color={none,auto,always}] 目录名称

ls [--full-time] 目录名称

### 练习cd命令

作用：进入目录。参数如下：

|  |  |
| --- | --- |
| **命令** | **作用** |
| cd /root/Docements | 切换到目录/root/Docements |
| cd ./path | 切换到当前目录下的path目录中，“.”表示当前目录 |
| cd ../path | 切换到上层目录中的path目录中，“..”表示上一层目录 |

### 练习cp 命令

作用：复制。参数如下：

|  |  |
| --- | --- |
| **参数** | **作用** |
| -a | 将文件的特性一起复制 |
| -p | 连同文件的属性一起复制，而非使用默认方式，与-a相似，常用于备份 |
| -i | 若目标文件已经存在时，在覆盖时会先询问操作的进行 |
| -r | 递归持续复制，用于目录的复制行为 |
| -u | 目标文件与源文件有差异时才会复制 |

### 练习rm命令

作用：删除。参数如下：

|  |  |
| --- | --- |
| **参数** | **作用** |
| -f | 就是force的意思，忽略不存在的文件，不会出现警告消息 |
| -i | 互动模式，在删除前会询问用户是否操作 |
| -r | 递归删除，最常用于目录删除 |

### 练习mv命令

作用：移动文件。参数如下：

|  |  |
| --- | --- |
| **参数** | **作用** |
| -f | force强制的意思，如果目标文件已经存在，不会询问而直接覆盖 |
| -i | 若目标文件已经存在，就会询问是否覆盖 |
| -u | 若目标文件已经存在，且比目标文件新，才会更新 |

### 熟悉find命令

作用：搜索文件。使用方式如下：

find [PATH] [option] [action]

#### 与时间有关参数

|  |  |
| --- | --- |
| **参数** | **作用** |
| -mtime n | n为数字，意思为在n天之前的“一天内”被更改过的文件 |
| -mtime +n | 列出在n天之前（不含n天本身）被更改过的文件名 |
| -mtime -n | 列出在n天之内（含n天本身）被更改过的文件名 |
| -newer file | 列出比file还要新的文件名 |

例如：

find /root -mtime 0 在当前目录下查找今天之内有改动的文件。

#### 与用户或用户组有关参数

|  |  |
| --- | --- |
| **参数** | **作用** |
| -user name | 列出文件所有者为name的文件 |
| -group name | 列出文件所属用户组为name的文件 |
| -uid n | 列出文件所有者为用户ID为n的文件 |
| -gid n | 列出文件所属用户组为用户组ID为n的文件 |

例如：

find /home/hadoop -user Hadoop 在目录/home/hadoop中找出所有者为hadoop的文件。

#### 与文件权限及名称有关参数

|  |  |
| --- | --- |
| **参数** | **作用** |
| -name filename | 找出文件名为filename的文件 |
| -size [+-]SIZE | 找出比SIZE还要大（+）或小（-）的文件 |
| -tpye TYPE | 查找文件的类型为TYPE的文件，TYPE的值主要有：一般文件（f)、设备文件（b、c）、目录（d）、连接文件（l）、socket（s）、FIFO管道文件（p） |
| -perm mode | 查找文件权限刚好等于mode的文件，mode用数字表示，如0755 |
| -perm -mode | 查找文件权限必须要全部包括mode权限的文件，mode用数字表示 |
| -perm +mode | 查找文件权限包含任一mode的权限的文件，mode用数字表示 |

例如：

find / -name passwd 查找文件名为passwd的文件。

find . -perm 0755 查找当前目录中文件权限的0755的文件。

find . -size +12k 查找当前目录中大于12KB的文件，注意c表示byte。

### 熟悉mkdir命令

作用：创建目录。格式如下：

mkdir [选项] 目录。参数如下：

|  |  |
| --- | --- |
| **参数** | **作用** |
| -m --mode=模式 | 设定权限<模式> (类似 chmod)，而不是 rwxrwxrwx 减 umask |
| -p --parents | 可以是一个路径名称。此时若路径中的某些目录尚不存在,加上此选项后,系统将自动建立好那些尚不存在的目录,即一次可以建立多个目录; |
| -v, --verbose | 每次创建新目录都显示信息 |

### 熟悉rmdir 命令

作用：删除目录。格式如下：

rmdir [选项] 目录。参数如下：

|  |  |
| --- | --- |
| **参数** | **作用** |
| -p | 递归删除目录dirname，当子目录删除后其父目录为空时，也一同被删除。如果整个路径被删除或者由于某种原因保留部分路径，则系统在标准输出上显示相应的信息。 |
| -v –verbose | 显示指令执行过程 |

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。

# vi编辑器的使用（4学时）

## 实验目的：

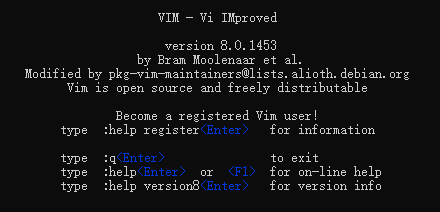
1. 掌握vi编辑器的启动和退出方法；
2. 掌握vi编辑器录入文件内容的方法；
3. 掌握vi编辑器各种命令使用；
4. 对比vi和Windows下文本编辑器的优缺点。

## 实验内容：

本实验要求完成本次实验所有内容，具体实验内容如下。

### vi编辑器的启动和退出

启动Shell终端，输入vi或者vim。



在末行输入 :wq 或者 :q! 即可退出？

### 熟悉vi三种模式

vi分为三种状态，分别是命令模式（command mode）、插入模式（Insert mode）和底行模式（last line mode），各模式的功能区分如下：

#### 命令行模式

控制屏幕光标的移动，字符、字或行的删除，移动复制某区段及进入Insert mode下，或者到 last line mode。

#### 插入模式

只有在Insert mode下，才可以做文字输入，按「ESC」键可回到命令行模式。

#### 底行模式

将文件保存或退出vi，也可以设置编辑环境，如寻找字符串、列出行号等。

### 实验文件保存

:w filename ：以filename为文件名保存文件；

:wq ：保存文件并退出；

:q! ：不保存并退出。

### 熟悉命令行模式

#### 移动光标

vi可以直接用键盘上的光标来上下左右移动，但正规的vi是用小写英文字母「h」、「j」、「k」、「l」，分别控制光标左、下、上、右移一格

|  |  |
| --- | --- |
| **按键** | **作用** |
| h | 左移光标一个字符 |
| l | 右移光标一个字符 |
| k | 光标上移一行 |
| j | 光标下移一行 |
| ^ | 光标移动至行首 |
| 0 | 数字"0"，光标移至文章的开头 |
| G | 光标移至文章的最后 |
| $ | 光标移动至行尾 |
| Ctrl + f | 向前翻屏 |
| Ctrl + b | 向后翻屏 |
| Ctrl + d | 向前翻半屏 |
| Ctrl + u | 向后翻半屏 |
| w | 移到下个字开头 |
| e | 移到下个字结尾 |
| b | 回到上个字开头 |

#### 删除

|  |  |
| --- | --- |
| **按键** | **作用** |
| x | 每按一次，删除光标所在位置的"后面"一个字符 |
| #x | 例如，「6x」表示删除光标所在位置的"后面"6个字符 |
| X | 大写的X，每按一次，删除光标所在位置的"前面"一个字符 |
| #X | 例如，「20X」表示删除光标所在位置的"前面"20个字符 |
| dd | 删除光标所在行 |
| #dd | 从光标所在行开始删除#行 |

#### 复制

|  |  |
| --- | --- |
| **按键** | **作用** |
| yw | 将光标所在之处到字尾的字符复制到缓冲区中 |
| #yw | 复制#个字到缓冲区 |
| yy | 复制光标所在行到缓冲区 |
| #yy | 例如，「6yy」表示拷贝从光标所在的该行"往下数"6行文字 |
| p | 将缓冲区内的字符贴到光标所在位置。注意所有与"y"有关的复制命令都必须与"p"配合才能完成复制与粘贴功能 |

#### 常用快捷键

|  |  |
| --- | --- |
| **按键** | **作用** |
| i | 在光标位置前插入字符 |
| a | 在光标所在位置的后一个字符开始增加 |
| o | 插入新的一行，从行首开始输入 |
| ESC | 从输入状态退至命令状态 |
| u | 取消操作 |
| cw | 更改光标所在位置的一个字 |
| #cw | 更改光标所在位置的#个字 |

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。

# 服务器的配置与管理（4学时）

## 实验目的：

1、掌握常用的网络服务的安装、配置与管理；

2、根据情况选择以下某种服务器：SSH网络服务、Apache Web服务、NFS服务、DHCP服务等。

## 实验内容：

实验要求完成本次实验所有内容。具体实验内容如下：

### SSH网络服务器的配置

#### ssh定义

ssh是一种网络协议，用于计算机之间的加密登录。如果一个用户从本地计算机，使用ssh协议登录另一台远程计算机，我们就可以认为，这种登录是安全的，即使被中途截获，密码也不会泄露。需要指出的是，ssh只有一种协议，存在多种实现，既有商业实现，也有开源实现。

#### 基本用法

（1）使用某个用户（例如user）登录远程主机host，命令：ssh user@host

（2）如果本地用户名和远程用户名一致，则登录时可以省略用户名，命令：ssh host

（3）ssh的默认端口是22，也就是说，你的登录请求会送进远程主机的22端口。使用-p参数，可以修改这个端口。命令：ssh –p 端口号 user@host

#### 安装ssh

apt-get install openssh-server

#### 查看ssh服务的开启状况

ps -e|grep ssh

#### 开启服务

sudo /etc/init.d/ssh start

### Apache Web服务器的配置

#### 安装apache

yum install httpd

#### 安装apache 服务管理

systemctl start httpd

#### 安装epel-release

yum -y install epel-release

### NFS服务器

#### 启动NFS守护进程

service nfs start

#### 查看NFS守护进程的当前状态

service nfs status

#### 停止NFS守护进程

service nfs stop

#### 如果修改了NFS的配置，重新启动NFS的守护进程

service nfs restart

### DHCP服务器的配置

#### 安装DHCP

yum -y install dhcp

#### 启动DHCP服务

service dhcpd start

#### 配置DHCP配置文件

vim /etc/dhcpd.cof

#### DHCP配置文件说明

|  |  |
| --- | --- |
| **配置** | **说明** |
| ddns-update-style interim | 动态dns的更新方式 |
| ignore client-updates | 忽略客户端更新 |
| subnet 192.168.1.0 netmask 255.255.255.0 | 子网信息，定义IP地址池，可以分配多个 |
| option routers | 默认网关地址 |
| option subnet-mask | 子网掩码 |
| option domain-name | 域名，搜索域 |
| option domain-name-servers | dns服务器地址，多个使用“,”隔开，对于Linux而言最多三个 |
| range | 指定地址池可分配地址范围 |
| default-lease-time | 默认租约长度 |
| max-lease-time | 最大租约长度 |
| host | 定义保留地址 |

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。

# shell编程（4学时）

## 实验目的：

1、掌握Bash shell的建立和执行方式；

2、掌握Bash shell的基本语法，并能够编写shell脚本。

## shell相关知识：

### 基础知识

#### 什么是shell编程

高大上的解释，往往让人摸不住头脑。一句话概括就是：shell编程就是对一堆Linux命令的逻辑化处理。

#### 为什么要会shell编程

举个简单的例子，我们做javaweb开发的，在以前，如果要在本地将程序打包，然后部署到远程服务器（抛开现在的ci， 原始的方法), 我们以前的做法通常会经历如下几个步骤：1、拉取最新代码（git pull）；2、编译打包；3、上传并部署到远程服务器。每次打包都要经历这一个阶段，效率低又烦躁。而此时，我们可以编写一个shell脚本，然后每次只需要运行一下这个shell脚本，即可实现打包部署这一系列动作，彻底解放双手，多好。

### 变量

shell编程中分为两种变量，第一种是我们自己定义的变量（自定义变量），第二种是Linux已定义的环境变量（环境变量，例如：$PATH, $HOME等，这类变量我们可以直接使用）。

#!/bin/bash

#使用环境变量

echo $PATH

#自定义变量hello

hello="hello world"

echo $hello

以上演示了自定义变量和系统环境变量的用法，使用很简单，就是使用$符号加上变量名就行了。记住：定义变量不用$符号，使用变量要加$就行了。在第5行中，我们在自定义变量时，使用了双引号，在shell编程中，如果变量出现空格或者引号，那么也必须加引号， 否则就可以省略。还有一点需要注意，定义变量的时候，“=”左右不要有空格。

将Linux命令执行结果赋值给变量

#!/bin/bash

path=$(pwd)

files=`ls -al`

echo current path: $path

echo files: $files

以上2行和第3行分别演示了两种方式来将Linux命令执行结果保存到变量。第2行将pwd执行结果（当前所在目录）赋值给path变量。第3行将ls -al命令执行结果（列出当前目录下所有的文件及文件夹）赋值给变量。注意：第三行的符号不是单引号，是键盘上“~”这个按键。

注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。同时，变量名的命名须遵循如下规则：

1、只能使用英文字母，数字和下划线，首个字符不能以数字开头；

2、不能有空格，可以使用下划线（\_）；

3、使用标点符号；

4、使用bash里的关键字（可用help命令查看保留关键字）。

**有效的 Shell 变量名示例如下：**

LXL

LD\_LIBRARY\_PATH

\_var

var2

**无效的变量命名：**

?var=123

user\*name=lxl

除了显式地直接赋值，还可以用语句给变量赋值，如：

for file in `ls /etc`

或

for file in $(ls /etc)

以上语句将 /etc 下目录的文件名循环出来。

使用一个定义过的变量，只要在变量名前面加美元符号即可，如：

your\_name="qinjx"

echo $your\_name

echo ${your\_name}

变量名外面的花括号是可选的，加不加都行，加花括号是为了帮助解释器识别变量的边界，比如下面这种情况：

for skill in Ada Coffe Action Java; do

echo "I am good at ${skill}Script"

done

如果不给skill变量加花括号，写成echo "I am good at $skillScript"，解释器就会把$skillScript当成一个变量（其值为空），代码执行结果就不是我们期望的样子了。

推荐给所有变量加上花括号，这是个好的编程习惯。

已定义的变量，可以被重新定义，如：

your\_name="tom"

echo $your\_name

your\_name="alibaba"

echo $your\_name

这样写是合法的，但注意，第二次赋值的时候不能写$your\_name="alibaba"，使用变量的时候才加美元符（$）。

#### 只读变量

使用 readonly 命令可以将变量定义为只读变量，只读变量的值不能被改变。下面的例子尝试更改只读变量，结果报错：

#!/bin/bash

myUrl="http://www.google.com"

readonly myUrl

myUrl="http://www.163.com"

运行脚本，结果如下：

/bin/sh: NAME: This variable is read only.

#### 删除变量

使用 unset 命令可以删除变量。语法：

unset variable\_name

变量被删除后不能再次使用。unset 命令不能删除只读变量。下面代码执行将没有任何输出。

#!/bin/sh

myUrl="http://www.163.com"

unset myUrl

echo $myUrl

#### 变量类型

运行shell时，会同时存在三种变量：局部变量、环境变量和shell变量。

##### 局部变量

局部变量在脚本或命令中定义，仅在当前shell实例中有效，其他shell启动的程序不能访问局部变量。

##### 环境变量

所有的程序，包括shell启动的程序，都能访问环境变量，有些程序需要环境变量来保证其正常运行。必要的时候shell脚本也可以定义环境变量。

##### shell变量

shell变量是由shell程序设置的特殊变量。shell变量中有一部分是环境变量，有一部分是局部变量，这些变量保证了shell的正常运行。

### shell 字符串

字符串是shell编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。单双引号的区别跟PHP类似。

#### 单引号

str='this is a string'

单引号字符串的限制：

1、单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；

2、单引号字串中不能出现单独一个的单引号（对单引号使用转义符后也不行），但可成对出现，作为字符串拼接使用。

#### 双引号

your\_name='lxl'

str="Hello, I know you are \"$your\_name\"! \n"

echo -e $str

输出结果为：

Hello, I know you are "lxl"!

双引号的优点：

1、双引号里可以有变量；

2、双引号里可以出现转义字符。

#### 拼接字符串

your\_name="lxl"

# 使用双引号拼接

greeting="hello, "$your\_name" !"

greeting\_1="hello, ${your\_name} !"

echo $greeting $greeting\_1

# 使用单引号拼接

greeting\_2='hello, '$your\_name' !'

greeting\_3='hello, ${your\_name} !'

echo $greeting\_2 $greeting\_3

输出结果为：

hello, lxl ! hello, lxl !

hello, lxl ! hello, ${your\_name} !

#### 获取字符串长度

string="abcd"

echo ${#string} #输出 4

#### 提取子字符串

以下实例从字符串第 2 个字符开始截取 4 个字符：

string="lxl is a great site"

echo ${string:1:4} # 输出 xl i

#### 查找子字符串

查找字符 i 或 o 的位置(哪个字母先出现就计算哪个)：

string="lxl is a great site"

echo `expr index "$string" io` # 输出 5

注意： 以上脚本中 ` 是反引号，而不是单引号 '。

### Shell 数组

bash支持一维数组（不支持多维数组），并且没有限定数组的大小。类似于 C 语言，数组元素的下标由 0 开始编号。获取数组中的元素要利用下标，下标可以是整数或算术表达式，其值应大于或等于 0。

#### 定义数组

在 Shell 中，用括号来表示数组，数组元素用"空格"符号分割开。定义数组的一般形式为：

数组名=(值1 值2 ... 值n)

例如：

array\_name=(value0 value1 value2 value3)

或者

array\_name=(

value0

value1

value2

value3

)

还可以单独定义数组的各个分量：

array\_name[0]=value0

array\_name[1]=value1

array\_name[n]=valuen

可以不使用连续的下标，而且下标的范围没有限制。

#### 读取数组

读取数组元素值的一般格式是：

${数组名[下标]}

例如：

valuen=${array\_name[n]}

使用 @ 符号可以获取数组中的所有元素，例如：

echo ${array\_name[@]}

#### 获取数组的长度

获取数组长度的方法与获取字符串长度的方法相同，例如：

# 取得数组元素的个数

length=${#array\_name[@]}

# 或者

length=${#array\_name[\*]}

# 取得数组单个元素的长度

lengthn=${#array\_name[n]}

### Shell 注释

#### 单行注释

以 # 开头的行就是注释，会被解释器忽略。通过每一行加一个 # 号设置多行注释，像这样：

#--------------------------------------------

# 这是一个注释

#--------------------------------------------

##### 用户配置区 开始 #####

#

#

# 这里可以添加脚本描述信息

#

#

##### 用户配置区 结束 #####

如果在开发过程中，遇到大段的代码需要临时注释起来，过一段时间又取消注释，每一行加个#符号较麻烦，可以把这一段要注释的代码用一对花括号括起来，定义成一个函数，没有地方调用这个函数，这块代码就不会执行，达到了和注释一样的效果。

#### 多行注释

多行注释还可以使用以下格式：

:<<EOF

注释内容...

注释内容...

注释内容...

EOF

EOF 也可以使用其他符号:

:<<'

注释内容...

注释内容...

注释内容...

'

:<<!

注释内容...

注释内容...

注释内容...

!

### Shell 传递参数

我们可以在执行 Shell 脚本时，向脚本传递参数，脚本内获取参数的格式为：$n。n 代表一个数字，1 为执行脚本的第一个参数，2 为执行脚本的第二个参数，以此类推。以下向脚本传递三个参数，并分别输出，其中 $0 为执行的文件名：

#!/bin/bash

echo "Shell 传递参数实例！";

echo "执行的文件名：$0";

echo "第一个参数为：$1";

echo "第二个参数为：$2";

echo "第三个参数为：$3";

为脚本设置可执行权限，并执行脚本，输出结果如下所示：

$ chmod +x test.sh

$ ./test.sh 1 2 3

执行的文件名：./test.sh，第一个参数为1，第二个参数为2，第三个参数为3。另外，还有几个特殊字符用来处理参数：

|  |  |
| --- | --- |
| **参数处理** | **说明** |
| $# | 传递到脚本的参数个数 |
| $\* | 以一个单字符串显示所有向脚本传递的参数。如"$\*"用「"」包括起来的情况、以"$1 $2 … $n"的形式输出所有参数。 |
| $$ | 脚本运行的当前进程ID号 |
| $! | 后台运行的最后一个进程的ID号 |
| $@ | 与$\*相同，但是使用时加引号，并在引号中返回每个参数。如"$@"用「"」包括起来的情况、以"$1" "$2" … "$n" 的形式输出所有参数。 |
| $- | 显示Shell使用的当前选项，与set命令功能相同。 |
| $? | 显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误。 |

#!/bin/bash

echo "Shell 传递参数实例！";

echo "第一个参数为：$1";

echo "参数个数为：$#";

echo "传递的参数作为一个字符串显示：$\*";

执行脚本，输出结果如下所示：

$ chmod +x test.sh

$ ./test.sh 1 2 3

第一个参数为1，参数个数为3，传递的参数作为一个字符串显示：1 2 3

#### $\* 与 $@ 异同：

**相同点：**都是引用所有参数。

**不同点：**只有在双引号中体现出来。假设在脚本运行时写了三个参数 1、2、3，，则 " \* " 等价于 "1 2 3"（传递了一个参数），而 "@" 等价于 "1" "2" "3"（传递了三个参数）。

举例：

#!/bin/bash

echo "-- \$\* 演示 ---"

for i in "$\*"; do

echo $i

done

echo "-- \$@ 演示 ---"

for i in "$@"; do

echo $i

done

执行脚本，输出结果如下所示：

$ chmod +x test.sh

$ ./test.sh 1 2 3

-- $\* 演示 ---

1 2 3

-- $@ 演示 ---

1

2

3

## 实验内容：

1. 使用shell编程实现一个20以内的乘法口诀表；
2. 使用shell编程实现解答数独题。

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。

# C语言开发工具（4学时）

## 实验目的：

* 1. 掌握gcc的编译流程、编译方法和使用；
  2. 掌握Linux调试器工具GDB的使用。

## 实验内容：

实验要求完成本次实验所有内容。编写下列代码并保存为hello.c：

#include <stdio.h>

int main()

{

printf("hello!\n");

return 1;

}

执行 gcc hello.c -o hello

执行 ./hello

观察输出，并思考总结上述几个命令的功能和意义。

### gcc编译

以上是简单的变异过程，具体的编译过程如下：

#### 预处理：

gcc -E 文件.c -o 文件.i

处理以#开头的代码如：头文件、宏定义、条件编译。

#### 编译：

gcc -S 文件.i -o 文件.s

把c代码翻译成汇编语言。

#### 汇编：

gcc -c 文件.s -o 文件.o

把汇编代码翻译成二进制。

#### 链接：

gcc 所有文件.o -o 文件名 链接程序需要用到的其他文件；

gcc -I 指定头文件路径；

gcc -static 静态链接。

### gdb调试

先创建一个test.c的文件写入代码，然后执行命令：

gcc test.c -o test -g

gdb test

可以进入调试模式，下面是几个调试相关命令：

|  |  |
| --- | --- |
| **命令** | **作用** |
| list | 显示代码 |
| run | 运行代码 |
| break | 下断 断点+行数或者函数名。可以直接输入 b |
| step | 一步一步执行，可以直接输入 s |

如下列代码：

#include <stdio.h>

int iterate(int value)

{

if(1 == value)

return 1;

return iterate(value - 1) + value;

}

int main()

{

printf("%d\n", iterate(10));

return 1;

}

输入 gcc test.c -g -o test。如果没有编译和链接方面的错误，可执行文件test。调试的步骤基本如下所示，

1. 输入gdb test
2. 进入到gdb的调试界面之后，输入list，查看test.c源文件
3. 输入 b main，在main函数上下断点
4. 输入run，启动test程序
5. 程序在main开始的地方设置了断点，所以程序在printf处断住
6. 单步跟踪。按s可以进入到函数，按n则越过函数
7. 如果希望从断点处继续运行程序，输入c
8. 希望程序运行到函数结束，输入finish
9. 查看断点信息，输入 info break
10. 查看堆栈信息，输入bt
11. 查看内存，输入 x/64xh 内存地址
12. 删除断点，输入delete break 断点序号
13. 查看函数局部变量的数值，输入print 变量名
14. 修改内存值，输入 print \*地址=数值
15. 实时打印变量的值，输入display 变量名
16. 查看函数的汇编代码，输入 disassemble 函数名
17. 输入quit 退出调试

### ****makefile编写****

#### 编写add.c文件，

#include "test.h"

#include <stdio.h>

int add(int a, int b)

{

return a + b;

}

int main()

{

printf(" 2 + 3 = %d\n", add(2, 3));

printf(" 2 - 3 = %d\n", sub(2, 3));

return 1;

}

#### 编写sub.c文件，

#include "test.h"

int sub(int a, int b)

{

return a - b;

}

#### 编写test.h

#ifndef \_TEST\_H

#define \_TEST\_H

int add(int a, int b);

int sub(int a, int b);

#endif

#### 编写makefile

test: add.o sub.o

gcc -o test add.o sub.o

add.o: add.c test.h

gcc -c add.c

sub.o: sub.c test.h

gcc -c sub.c

clean:

rm -rf test

rm -rf \*.o

#### 执行make

观察执行情况，并调试成功。

### 编写程序

要求使用C语言实现一个数独生成器，可以随机生成数独题，并调试输出。

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。

# 文件系统编程（4学时）

## 实验目的：

* 1. 理解Linux对文件系统的操作方法；
  2. 理解库函数和系统调用的区别；
  3. 掌握常用的与文件操作有关的库函数和系统调用。

## 基础知识：

### 文件系统

文件系统是Linux组织系统资源的一种方式；文件是对系统资源的一个抽象，是对系统资源进行访问的一个通用接口。内存、硬盘、一般设备以及进程间通信的通道等资源都被表示为文件。对这些资源的操作，就是对一个文件进行相关操作。我们要清楚系统调用和库函数以及API的相关概念。

### 系统调用

系统编程是指直接利用系统的底层接口进行编程。操作系统通过给用户提供一组特殊"接口"，以便用户程序可以通过这组特殊接口来获取操作系统的内核提供的服务，包括进程控制(进程间通信)、文件系统控制、系统控制、存储管理、网络管理、socket管理、用户管理等，这种调用接口的方法也称之为"系统调用"。"系统调用接口"，即一般通过CPU的软中断指令实现从用户态到内核态的调用,这是一种由应用程序主动发起的模式切换，让软中断处理完毕返回后重新切换到用户态，实现系统调用的返回。

### 标准I/O库函数

C标准库提供了文件的标准I/O库，与系统调用的区别在于实现了一个垮平台的用户态缓冲的解决方案。标准I/O库使用简单，与系统调用I/O相似，也包括打开、读写、关闭这些操作。

### API

对于一些预先定义的函数，就是C标准库提供的编程接口（API），是为开发人员提供编写应用程序的API，从而使我们开发的应用程序无需访问源码。注意：API和系统调用并不是一一对应的关系，因为C标准库提供的有些接口根本不需要请求内核服务就可以完成，比如内存的复制、数学上的计算；或者有些API可能需要经过多次的系统调用才能完成其功能。

### 常见的文件类型

#### 普通文件

数据存储在磁盘中，可以随机访问文件的内容。Linux系统中的文件是面向字节的，文件的内容以字节为单位进行存储和访问。

#### 目录

目录是一种特殊的文件。

#### 管道文件

管道是Linux中的一种进程间通信的机制。

#### 设备文件

设备文件没有具体的内容，对设备文件的读写操作实际上与某个设备的输入输出操作关联在一起。

#### 符号链接

符号链接的内容是指向另一个文件的路径。当对符号链接进行操作时，系统会根据情况将这个操作转移到它所指向的文件上去，而不是对它本身进行操作。

#### Socket

Socket也是一种进程间通信的方式，与管道不同的是，它们可以在不同的主机上进行通信，也就是网络通信。

### 文件操作的过程

文件操作过程一般又三个操作组成：1、打开文件；2、读写等操作；3关闭文件。这三个操作需要文件描述符，在系统调用中，我们主要是通过文件描述符实现文件的相应操作。

#### 文件描述符的特点

1. 非负整数；
2. 表示各种类型的打开的文件；
3. 只要使用文件描述符可指定所操作的文件。

#### 打开文件

打开文件成功后，应用程序将获得文件描述符。

#### 读写操作

程序使用文件描述符对文件进行读写等操作。

#### 关闭文件

全部操作完毕后，应用程序需要将文件关闭以释放用于管理打开文件的内存。

### 系统调用API接口

#### 创建文件

调用方式：

#include<sys/types.h>

#include<sys/stat.h>

int creat(const char \*filename, mode\_t mode);

说明：

|  |  |
| --- | --- |
| **项目** | **说明** |
| filename | 创建文件名（包含路径、缺省为当前目录） |
| mode | 文件属性：  0：无任何权限  S\_IRUSR或1：可读  S\_IWUSR或2：可写  S\_IXUSR或3：可执行  S\_IRWXU或7：均可 |
| mode\_t | 表示unsigned int，无符号整数。 |
| 返回值 | 创建成功返回文件描述符fd；  错误返回-1。 |

#### 打开文件

调用方式：

#include<sys/types.h>

#include<sys/stat.h>

int open(const char \*pathname, int flags);//文件已创建

int open(const char \*pathname, int flags, mode\_t mode);//文件没有创建

说明：

非阻塞方式，即当文件打开失败时，进程不阻塞，跳过该部分执行进程剩余部分或者结束进程。

|  |  |
| --- | --- |
| **项目** | **说明** |
| flags | 打开标识：  O\_RDONLY：只读方式；  O\_WRONLY：写方式；  O\_RDWR：可读可写；  O\_APPEND：追加方式打开；  O\_CREAT：创建一个文件；  O\_NOBLOCK：非阻塞方式打开； |
| 返回值 | 打开成功返回文件描述符fd；  错误返回-1，并把错误代码设给err。 |

#### 读文件

调用方式：

#include<unistd.h>

int read(int fd, const \*buf, size\_t length)

说明：

从文件描述符fd指向的文件中，读取length个字节到buf所指向的缓存区中。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 实际读取的字节数length。 |

#### 写文件

调用方式：

#include<unistd.h>

int write(int fd,const \*buf, size\_t length)

说明：

向文件描述符fd指向的文件，写入length个字节到buf所指向的缓存区中。size\_t为unsigned int类型，在32位系统中size\_t是4字节的，在64位系统中size\_t是8字节的。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 实际写入的字节数length |

#### 关闭文件

调用方式：

#include<unistd.h>

int close(int fd);

说明：

关闭文件，调用close()会让数据写回磁盘，并释放该文件所占的资源。

|  |  |
| --- | --- |
| **项目** | **说明** |
| fd | 文件描述符 |
| 返回值 | 返回正常为0；  失败为-1 |

#### 访问判断

调用方式：

int access(const char \*pathname, int mode)

说明：

判断文件是否可以进行读写等操作。具体mode有：读权限R\_OK、写权限W\_OK、执行权限X\_OK、文件存在F\_OK。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 测试成功，函数返回为0；  否者返回为-1 |

### 标准I/O库函数API接口

相对于系统I/O函数库，标准I/O函数库实现了跨平台用户缓冲机制，这有利于提高系统进行I/O操作的效率。

#### 打开文件

调用方式：

FILE \*fopen(const char \*filename, const char \*mode);

说明：

打开一个文件流并为文件建立相应的信息区以存放文件信息，建立文件缓冲区来暂时存放输入输出数据，完成后返回指针。

|  |  |
| --- | --- |
| **项目** | **说明** |
| filename | 被打开文件的名称，包含路径 |
| mode | 字符串，用于表示打开的模式 |
| 返回值 | 成功返回一个FILE类型的文件指针（即fp指向文件信息区的起始地址），失败返回NULL |

举例：

FILE \*fp;

if(fp=fopen("文件名","r")==NULL)

{

printf("cannot open this file\n");

exit(0);

}

#### 关闭文件

调用方式：

int fclose(FILE \*fp);

说明：

关闭打开的文件，关闭前系统会自动调用fflush函数更新数据即将文件流中的数据写入文件。

|  |  |
| --- | --- |
| **项目** | **说明** |
| fd | 文件描述符 |
| 返回值 | 关闭成功返回0，否者返回EOF并设置变量errno的值以指示错误 |

#### 文件读数据块

调用方式：

size\_t fread(void \*buffer, size\_t size, size\_t count, FILE \*fp);

说明：

从fp指向的文件流中读取count个数据块且为size字节到缓存区。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 返回数据块个数，size\_t类型 |

#### 文件写数据块

调用方式：

fwrite(void \*buffer, size\_t size, size\_t count, FILE \*fp);

说明：

从缓冲区写出count个数据块到fp指向的文件。

#### 文件读字符

调用方式：

int fget(FILE \*fp);

说明：

从文件读出一个字符。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 成功返回该字符；  失败返回文件结束标志EOF(-1) |

#### 文件写字符

调用方式：

int fput(int ch, FILE \*fp);

说明：

将一个字符ch写入一个文件中。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 成功返回该字符；  失败返回结束标识EOF(-1) |

#### 标准输出写字符

调用方式：

int putchar(int c);

说明：

等价于int fput(c, stdout);

#### 是否文件结束符

调用方式：

int feof(FILE \*fp)

说明：

检查文件读写位置是否移动文件末尾。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 文件结束返回非0；  文件未结束返回0 |

#### 文件读字符串

调用方式：

char \*fgets(char \*str, int n, FILE \*fp)

说明：

从指定的流fp读取一行，并把它存储在str所指向的字符串内。当读取n-1个字符时，或者读取到换行符时，或者到达文件末尾时返回。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 成功返回数组str地址；  失败返回NULL。 |

#### 文件写字符串

调用方式：

char fputs(const char \*str, FILE \*fp)

说明：

把str所指向的字符串写入到文件指针变量fp所指向的文件中。

|  |  |
| --- | --- |
| **项目** | **说明** |
| 返回值 | 非负数表示写入成功，  有错误发生则返回EOF |

举例：

while(fgets(str[i],10,fp)!=NULL)

{

printf("%s",str[i]);

i++;

}

#### 文件定位

调用方式：

int fseek(FILE \*fp, long offset, int whence)

说明：

用于移动文件流的读写位置。

|  |  |
| --- | --- |
| **项目** | **说明** |
| whence | SEEK\_SET 0：对文件开始位置偏移量；  SEEK\_CUR 1：对文件当前位置的偏移量；  SEEK\_END 2：对文件末尾的偏移量 |
| 返回值 | 0表示操作成功；  -1表示操作失败并且errno变量的值被设置为错误码 |

#### 其他文件定位函数

调用方式：

void rewind(FILE \*fp);

说明：

将读写位置移动到文件流的开头，

调用方式：

long ftell(FILE \*fp)；

说明：

得到文件流的读写位置，返回值为文件流的当前相对于头文件的读写位置。

调用方式：

int fflush(FILE \*fp);

说明：

将文件流中的数据更新到文件中，成功返回0，否者将返回EOF并且设置变量errno的值以指示错误。在fclose关闭文件之前，系统会自动调用fflush更新数据。

#### 标准I/O错误处理

当标准I/O操作发生错误处理时，一般返回NULL指针或者EOF，我们可以通过errno变量得到错误码。

调用方式：

int ferror(FILE \*fp);

int feof(FILE \*fp);

说明：

ferror()函数用于判断文件流是否发生错误，若返回非0值则表示发生了错误.

feof()函数用于判断对文件流的读写是否已经达尾部，若返回非0值则表示已经达尾部。

## 实验内容：

要求在Linux中遍历一个目录，并将该目录所有文件拷贝到另外一个目录中，要求“复制”时使用系统函数，“粘贴”时使用标准库函数。并对比这两种方式的区别，说说你更愿意使用哪种方式？觉得哪种好？

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。

# 综合实例（4学时）

## 实验目的：

* 1. 掌握Linux系统下进行综合应用开发的基本方法；
  2. 掌握Linux系统下进行综合应用开发的合适工具选型和应用；
  3. 掌握在Linux应用开发中获取资料和在线帮助的方法。

## 实验内容：

实验要求以自由组合方式，每两个同学为一小组，每小组单独完成本次实验所有内容。具体实验内容如下：

1. 实现一个网络文件传输工具，可以使用文件和目录的传输；
2. 增加共享目录，可以增加和删除共享的目录和文件，供网络查看；
3. 可以查看对方共享目录和文件，并选择下载；
4. 可以向对方传送文件和目录，并且该文件和目录会自动被对方共享。

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。

# 综合设计实践（4学时）

## 实验目的：

* 1. 掌握中大型项目的需求分析、系统设计、编码、测试，以及项目的部署运行，具备复杂工程问题的解决能力；
  2. 掌握综合应用Linux知识点进行管理和软件开发的能力；
  3. 掌握文献检索、系统建模、文档写作等方面的能力。

## 实验内容：

从下列不同难易程度的选题中选择题目进行开发实际，要求至少完成题目，学生可自由选择。

### 初级题目

初级1：分别使用shell编程和C语言编程实现对目录的复制操作，并对C语言版本编写makefile文件进行编译安装到指定目录下。

初级2：用 C 语言编程实现两种Linux命令，如list命令、tree命令、ping命令等，并编写makefile文件进行编译安装到指定目录下。

初级3：用shell脚本监控系统的各项参数，包括操作系统版本、内核版本、主机名、登录用户、内存和磁盘使用情况等。并可以将脚本加入系统环境中，可以直接在终端里执行；

初级4：DHCP 服务器的安装、配置与运行，并能够初步阅读开机自动启动的执行脚本，写出学习报告；

### 中级题目

中级1：编写shell程序实现“学生信息管理”，包括查看、增加、删除、修改、统计等基本功能；

中级2：基于TCP的聊天程序，包括客户端和服务器端，允许多个客户端向服务器发送请求；

中级3：基于libcap的网络数据包嗅探与解析，要求能够捕获数据包，并对端口和IP地址等方面进行解析；

中级4：基于内核Inotify机制的实时文件保护，要求对指定的文件和目录进行监控，包括对增、删、改等操作，可以使用原生 Inotify API，也可以使用python包的二次封装。

### 高级题目

高级1：Tensorflow开发框架搭建与应用开发；

高级2：Spark/hadoop 框架的搭建与应用开发；

高级3：区块链 Hyperledger Fabric 超级账本架构应用开发；

高级4：LAMP 部署配置与应用开发：基于 B/S 模式的实用试题库系统。

## 实验结果和问题分析：

将源程序关键代码、运行结果（截图）和结果分析以及实验中遇到的问题和解决问题的方法，写在实验报告上。