**Linux程序设计**

**一、入门介绍**

1. 有关bin目录的说明

@/bin :用于存放启动系统时用的程序

@/usr/bin :用于存放用户使用的标准程序

@/usr/local/bin :用于存放软件安装的程序

@/usr/sbin :系统管理员可使用的程序

1. Linux与MS-DOS和Windows的区别

（1）Linux和UNIX使用冒号（:）分隔PATH变量

MS-DOS和Windows使用分号（:）分隔PATH变量

1. Linux用正斜线（/）分隔文件名里的目录名

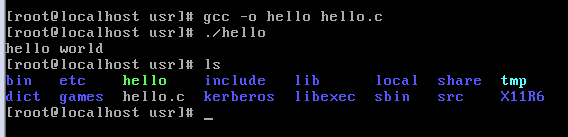
Windows用反斜线（\）分隔文件里的目录名

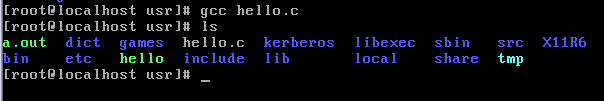
1. 第一个Linux C程序

使用vi编辑器编写一个hello.c程序



编译并运行C程序



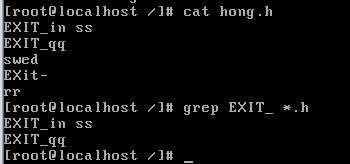


（说明1： “./hello” 中的” . ” 代表当前目录）

（说明2： “gcc -o hello hello.c” 中的” -o hello” 代表指定汇编输出名，不指定则，默认为a.out）

1. 命令grep ，通过关键字来搜索文件

grep EXIT\_ \*.h //当前目录下的所有以.h结尾且含有”EXIT\_”字符的文件



1. 库文件

@标准系统库文件一般存在于/lib 和 /usr/lib 下

@C语言编译器默认只搜索标准C语言库

库文件命名：一般以lib开头，加什么库，后 . 类型结尾。

@libc c语言库

@libm 数学库

@.a 代表传统的静态函数库

@.so 代表共享函数库

gcc -o fred fred.c /usr/lib/libm.a

编译fred.c函数除了用标准C语言函数外，还需要数学库来编译

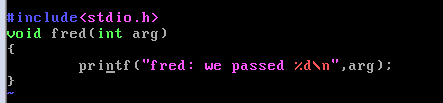
gcc -o fred fred.c -lm （缩写格式：功能和上面一样）

标准目录（/usr/lib）下名为libm.a 的库函数

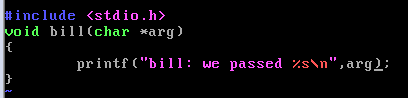
gcc -o fred -L/usr/openwin/lib fred.c -lx11

大写的L ， 指明用路径/usr/openwin/lib下的libx11库来编译fred.c文件

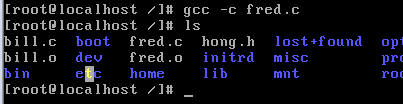
1. 测试
2. 创建文件fred.c文件



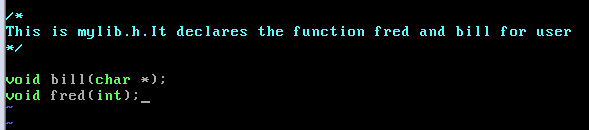
1. 创建文件bill.c文件



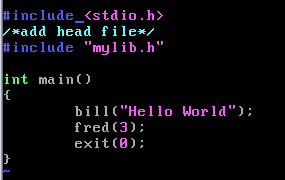
1. 编译这两个文件（注意：用gcc -c bill.c fred.c ,编译产生的文件名以 .o 结尾）



1. 创建一个头文件mylib.h （将要用的函数声明进去）



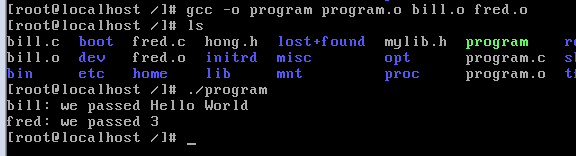
1. 创建调用程序program.c（注意：用双引号，不要用尖括号）



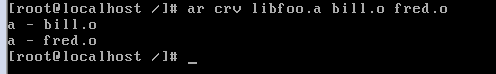
1. 编译program.c文件（注意：用gcc -c program.c ）



1. 一起编译运行（注意：用gcc -o .....来编译）

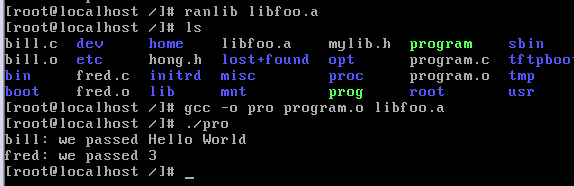


1. 使用ar程序来创建libfoo.a库文件

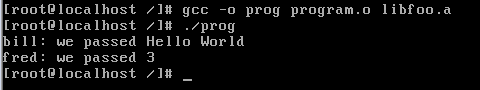


某些系统中需要执行该命令

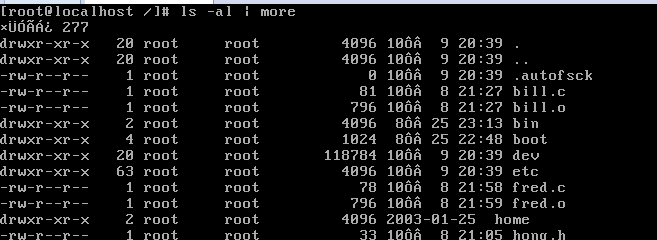
ranlib libfoo.a



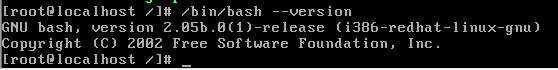
1. 编译使用



1. shell编程
2. ls -al | more //文件列表的分屏显示



1. /bin/bash --version //查看bash的版本号



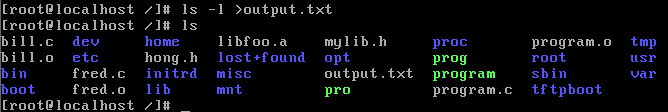
1. 符号介绍

“<” 和 “>” 是对输入和输出进行重定向

“|” 是在同时执行的程序之间实现数据的管道传递

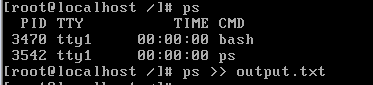
“$(...)” 是获取子进程的输出

1. Ls -l > output.txt //把ls命令的输出保存到文件output.txt中



（注意：该命令会在当前目录中新建或覆盖原有output.txt文件内容）

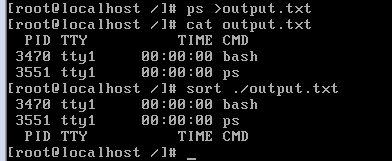
1. ps >> output.txt //ps命令的输出附加到指定文件尾部



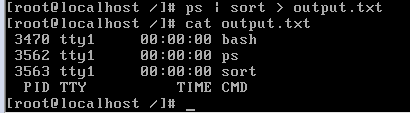
1. Kill -HUP 1234 > killout.txt 2>killerr.txt //杀死一个进程的信息输 出到killout.txt文件中，若出现错误，则错误流放到killerr.txt文件中
2. Kill -HUP 1234 > killouterr.txt 2>&1 //杀死一个进程的信息输出到killouterr.txt中，然后将标准错误输出到killouterr.txt文件中，（不覆盖）
3. Kill -1 1234 > /dev/null 2>&1 //标准输出流和标准错误流放到回收站中
4. 管道

不用管道之前：（1）输出：ps > output.txt

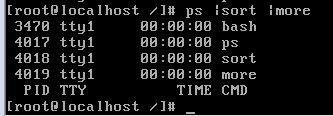
（2）排序：sort output.txt



用管道之后：（1）ps | sort >output.txt

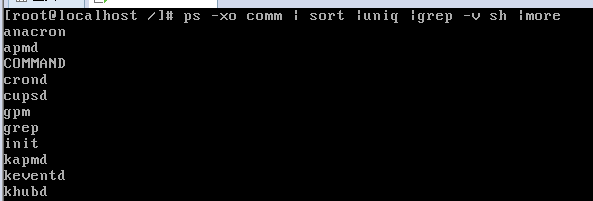


用列：ps | sort | more //ps的输出排序，然后分页显示



用列：ps -xo comm | sort |uniq |grep -v sh |more

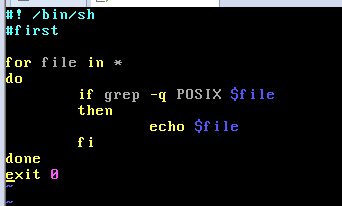
//ps命令输出，排序，再用uniq命令去除名字相同的进程，然后grep -v sh命令删除名为sh的进程，最后分屏显示



1. 检测文件类型：file filename



脚本程序定义：



脚本程序first说明：

#：注释符

#！：特殊注释符，告诉系统同一行上跟在它后面的那个参数是用来执行脚本的程序

Exit ：确保脚本程序能返回一个有意义的退出码

脚本程序的运行：  
 运行方法一： /bin/bash ./first



运行方法二： chmod +x first //将文件first变为可执行

./first //运行当前目录下的first文件

1. 管理员修改文件的访问权限

Chown root /usr/local/bin //修改文件的拥有者为root用户

Chgrp root /use/local/bin //修改文件的所在组为root组

Chmod 755 /usr/local/bin //修改文件的读、写、可执行权限

@读4，写2 ，可执行1

@7=4+2+1即rwx

@- - - - - - - - -：有9个，前三个代表文件拥有者权限，中间三个代表文件所在组权限，后三个代表文件所在组以外权限

@在Linux中，若拥有目录的写权限，就可以对目录进行添加和删除文件

1. Shell语法
2. 脚本变量（一般小写，由用户自己定义）
3. 环境变量（大写，有系统定义）

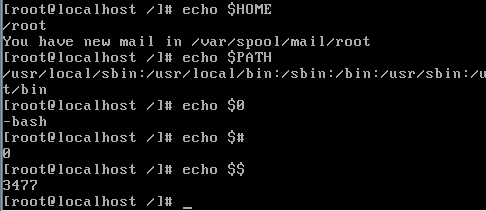
$HOME ：当前用户的家目录

$PATH ：以冒号分隔，用来搜索命令目录列表

$0 ：shell 脚本的名字

$# ：传递给脚本参数的个数

$$ ：shell脚本的进程号



1. 参数变量

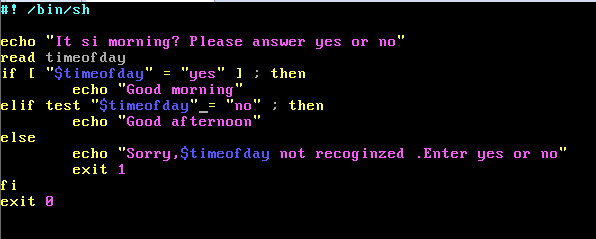
S1 ，s2，. . . . //脚本程序的参数

S\* //该变量中列出脚本的所有参数

S@ //该变量中列出脚本的所有参数，参数之间有空格分开

1. “test” 或 “[ ]” 命令可以使用的三类比较

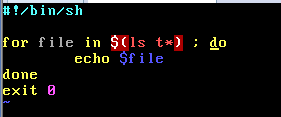
|  |  |
| --- | --- |
| 字符串比较 | 结果 |
| String1 = string2 | 如果两个字符串相同则结果为真 |
| String1 != string2 | 如果两个字符串不同则结果为真 |
| -n string | 若果字符串不为空则结果为真 |
| -z string | 如果字符串为null（一个空串）则结果为真 |
| 算术比较 |  |
| Expression1 -eq expression2 | 如果两个表达式相等则结果为真 |
| Expression1 -ne expression2 | 如果两个表达式不相等则结果为真 |
| Expression1 -gt expression2 | 如果1>2则结果为真 |
| Expression1 -ge expression2 | 如果1>=2则结果为真 |
| Expression1 -lt expression2 | 如果1<2则结果为真 |
| Expression1 -le expression2 | 如果1<=2则结果为真 |
| ! expression | 如果表达式为假则结果为真 |
| 文件条件测试 |  |
| -d file | 如果文件是目录则结果为真 |
| -e file （不可用） |  |
| -f file | 如果文件是普通文件则结果为真 |
| -g file | 如果文件的set-group-id位被设置则结果为真 |
| -r file | 如果文件可读则结果为真 |
| -s file | 如果文件的大小不为0则结果为真 |
| -u file | 如果文件的set-user-id位被设置则结果为真 |
| -w file | 如果文件可写则结果为真 |
| -x file | 如果文件可执行则结果为真 |



用例说明：（1）”$timeofday” = “yes” :比较时，=号两边必须要有空格

：赋值时，=号两边不能有空格

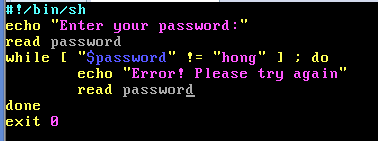
1. if或 elif 和then 在同一行时，要用 ”;” 号隔开
2. Exit 0 :每个脚本程序都有一个返回码，是个好习惯
3. If test “$timeofday” = “no” :为变量加双引号，可防止变量值为null时发生语法错误
4. echo ：输出，自动换行
5. Read ：通过键盘的输入为变量赋值
6. For 语句



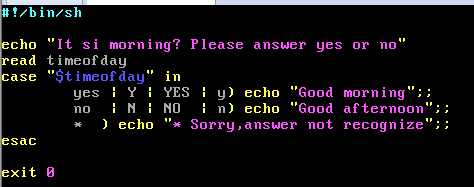
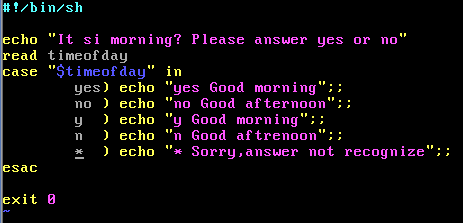
用列说明：（1）for 和do 在同一行 要用 ”;” 分号分隔

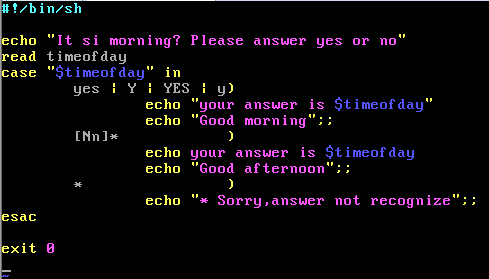
（2）for file in $(ls t\*) :for 命令的参数表来自$(ls t\*) 命令的输出

1. While语句，until语句



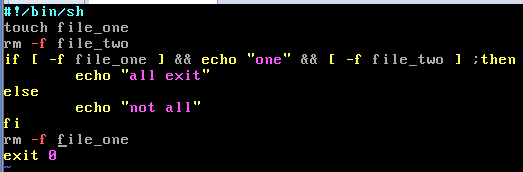
1. Case语句

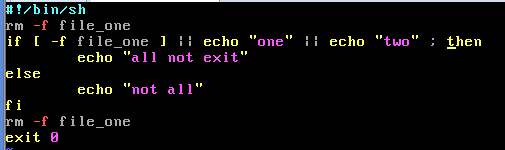




用例说明：（1）yes ) echo “yes good morning”;; :双分号 ”;;” 必须写，作为结束标志

1. yes | Y | YES | y) echo “good morning”;; : ”|” 符号代表或
2. \* ) echo : ”\*” 符号为通配符
3. [Nn] ：限制允许出现的字符
4. 命令列表（&& 和 ||）

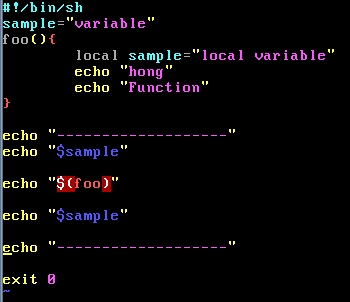
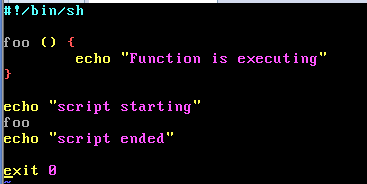




说明：（1）”&&” 符号为与运算，只有前一个正确才会执行后一个

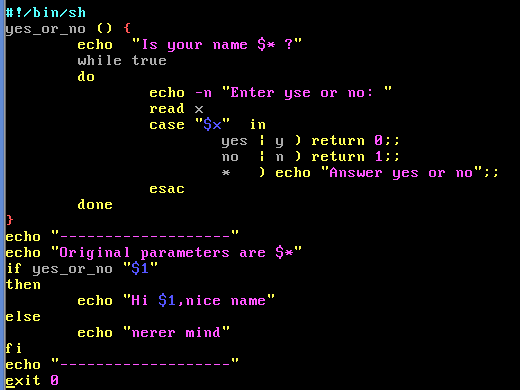
（2）”||” 符号为或运算，只有前一个错误才会执行后一个

1. touch file\_one ：若文件名存在则不会新个名为file\_two的文建，否则新建一个名为file\_one的空文件。
2. Rm -f file\_two :删除一件
3. 函数



说明：（1）foo （）{ . . . . . . } ：函数的定义，foo 为函数名

1. foo ：调用函数foo
2. 函数先定义后使用
3. Local sample=”local variable” :定义局部变量sample
4. echo “ $(foo)” :输出 “ $(foo)” 命令的值





1. while true ：表示永远为真，注意写法和显示
2. echo “Is your name $\* ?” : “$\*” 表示所有参数
3. return 0 ：return 关键字， 0 为真 ；1 为假
4. If yes\_or\_no “$1” : “ $1” 表示第一个参数值
5. ./test\_function hong long : hong long 作为脚本运行的输入参数
6. 命令

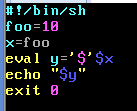
@break ：跳出for，while ，until 循环

@ : ：空命令，代表true，”:” 作为一行开头，代表该行的注释，（一般用#注释）

@continue ：使for，while ，until 循环跳转到下一次循环

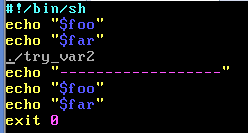
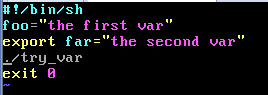
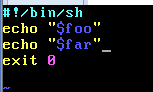
@. ：点（.）命令用于在当前shell中执行命令

@eval ：像一个额外的$符，给出一个变量的值的值



@exec ：将当前shell替换为不同的程序

@export ：导出当前变量作为该shell下衍生的子进程的环境变量



Try\_var try\_var2 try\_var3

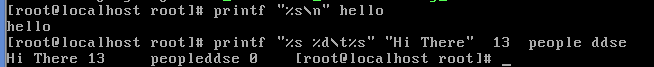
说明（1）./try\_var ：脚本文件的调用（指明调用那个文件）

（2）export far=”the second var” ：导出变量far，可供被调用的脚本使用

@expr 将他的参数当做一个表达式求值

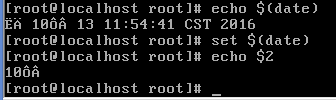
|  |  |
| --- | --- |
| 表达式求值 | 说明 |
| expr1 | expr2 | 若果expr1非0，则等于expr1 ，否则等于expr2 |
| expr1 & expr2 | 只要有一个表达式为0，则结果为0，否则等于expr1 |
| expr1 = expr2 | 等于 |
| expr1 > expr2 | 大于 |
| expr1 >= expr2 | 大于等于 |
| expr1 < expr2 | 小于 |
| expr1 <= expr2 | 小于等于 |
| expr1 != expr2 | 不等于 |
| expr1 + expr2 | 加 |
| expr1 - expr2 | 减 |
| expr1 \* expr2 | 乘 |
| expr1 / expr2 | 除 |
| expr1 % expr2 | 取余 |

@printf ：标准化输出



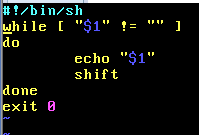
@return

@set ：为shell设置参数变量，



日期的使用

@shift ：把所有参数位置左移动一个位置，原来$1的参数丢失，$0保持不变



说明：循环移动输出参数的值

@trap ：信号处理

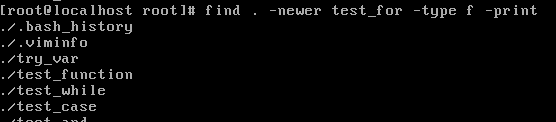
@unset ：从环境中删除变量或函数



@find ：用于搜索文件 find [path] [options] [tests] [actions]

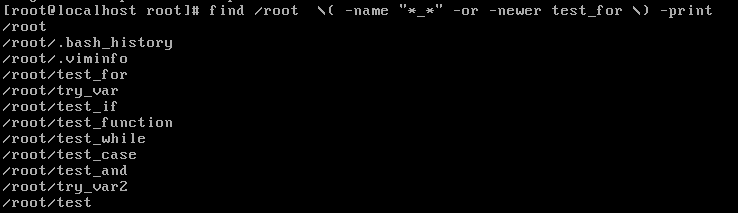
|  |  |  |  |
| --- | --- | --- | --- |
| 选项 | | 含义 | |
| -depth | | 在查看目录本身之前先搜索目录的内容 | |
| -follow | | 跟随符号链接 | |
| -maxdepths N | | 最多搜索N层目录 | |
| -mount (或 -xdev) | | 不搜索其他文件系统中的目录 | |
| 测试 | | 含义 | |
| -atime N | | 文件在N天之前被最后访问过 | |
| -mtime N | | 文件在N天之前被最后修改过 | |
| -name pattern | |  | |
| -newer otherfile | | 文件比otherfile文件要新 | |
| -type c | | 文件的类型是c型 | |
| -user username | | 文件的拥有者是指定的用户username | |
| 操作符（短格式） | 操作符（长格式） | | 含义 |
| ！ | -not | | 测试取反 |
| -a | -and | | 两个测试都必须为真 |
| -o | -or | | 两个测试只是有一个为真 |
| 动作 | | 含义 | |
| -exec command | | 这个动作必须使用\;字符来结束 | |
| -ok command | | 这个动作必须使用\;字符来结束 | |
| -print | | 打印文件名 | |
| -ls | | 对当前文件使用命令ls -dils | |
|  | |  | |

使用实例：



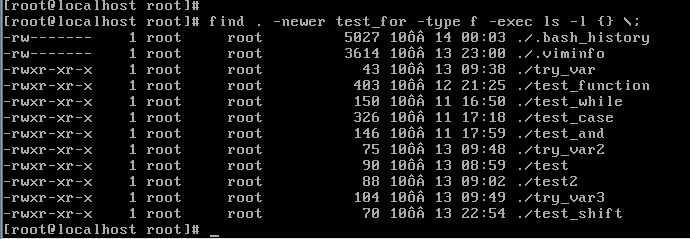
说明：（1）find ：查找命令

1. . :当前目录下
2. -newer test\_for ：比test\_for要新的文件或目录
3. -type f ：指定为文件
4. -print : 输出



说明：（1）/root : 在/root 目录下查找

1. -or :或操作，不要忘了前面短杠“-”
2. \( -name “\*\_\*” -or -never test\_for \) :使用反斜杠 ”\” 来转义圆括号，注意括号内必须留有空格



说明：（1）-exec ls -l ：以 详细列表的形式输出

1. { } ：输出一遍，注意不可省略
2. \; :以字符 “ \;” 作为命令的结束

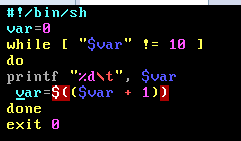
@grep ：在文件中搜索字符串

Grep [options] pattern [files]

|  |  |
| --- | --- |
| 选项 | 含义 |
| -c | 输出匹配行的数目，而不是输出匹配的行 |
| -E | 启用扩展表达式 |
| -h | 取消每个输出行的普通前缀，即匹配查询模式的文件名 |
| -i | 忽略大小写 |
| -l | 只列出包含匹配行的文件名，而不输出真正的匹配行 |
| -v | 对匹配模式取反，即搜索不匹配行 |

1. 命令的执行

@算术扩展

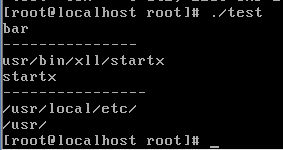
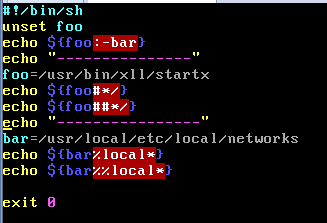


说明：（1）命令printf 的使用（注意与print 不同）

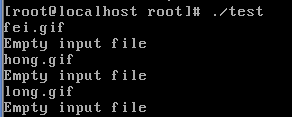
（2）$(($var + 1)) :两对圆括号，代表算术替换，而一对圆括号用于命令执行和输出，算术运算变量”$var” 不用双引号

@参数扩展

|  |  |
| --- | --- |
| 参数扩展 | 说明 |
| ${param:-default} | 如果param为空，就把它设置为default的值 |
| ${#param} | 给出param的长度 |
| ${param%word} | 从尾部开始删除与word匹配最小部分，返回剩余部分 |
| ${param%%word} | 从尾部开始删除与word匹配最长部分，返回剩余部分 |
| ${param#word} | 从头部开始删除与word匹配最小部分，返回剩余部分 |
| ${param##word} | 从头部开始删除与word匹配最长部分，返回剩余部分 |



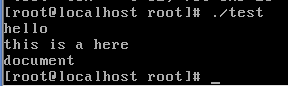
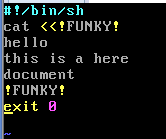
说明：（1）



说明：（1）cjpeg $image > ${image%%gif}jpg ：将格式为gif的文件重新创建一个为jpg格式

@here文档

Here文档以”<<”符号开始，紧跟一个特殊字符序列，最终以该字符序列结束



说明：（1）cat <<!FUNKY! ：cat命令， “<<” ：标明here文档的开始 ，!FUNKY! ：特殊字符序列

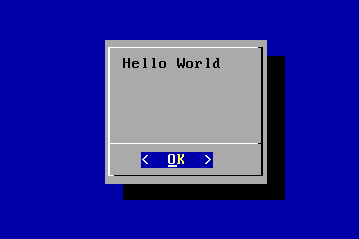
（2）!FUNKY! :here 文档结束

@调试脚本程序

|  |  |  |
| --- | --- | --- |
| 命令行选项 | Set选项 | 说明 |
| sh -n <script> | set -o noexec （或set -n） | 只检查语法错误，不执行命令 |
| sh -v <script> | set -o verbose （或set -v） | 在执行命令之前回显它们 |
| sh -x <script> | set -o xtrace （或set -x） | 在处理完命令之后回显它们 |
| sh -u <script> | set -o nounset （或set -u） | 如果使用了没定义的变量，就给出出错消息 |
|  | set +o noexec （或set +n） | 关闭该功能（注：- ：表示开启 ；+ 表示关闭） |

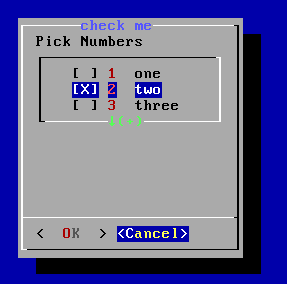
1. Dialog 工具

|  |  |  |
| --- | --- | --- |
| 类型 | 用于创建类型的选项 | 含义 |
| 复选框 | --checklist | 允许用户显示一个选项列表，每个选项都可被单独选择 |
| 信息框 | --infobox | 在显示消息后，对话框将立即返回，但并不消失 |
| 输入框 | --inputbox | 允许用户输入文本 |
| 菜单框 | --menu | 允许用户选择列表中的一项 |
| 消息框 | --msgbox | 向用户显示一条消息，同时显示一个ok按钮 |
| 单选框 | --radiolist | 允许用户选择列表中的一项 |
| 文本框 | --textbox | 允许用户在带有滚动条的文本框中显示一个文件的内容 |
| 是/否框 | --yesno | 允许用户图文==提问，用户可以选择yes/no |



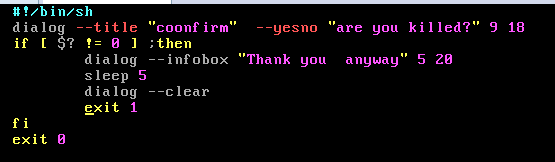
说明：该命令





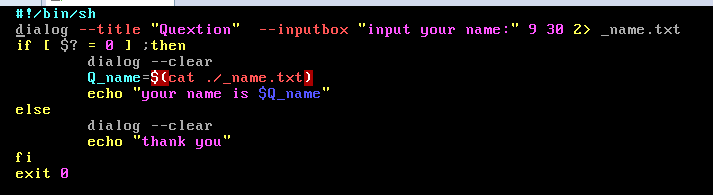
说明：（1）dialog ：对话框

1. --title “check me” ：对话框标题
2. --checklist ：对话框类型
3. “Pick Numbers” ：对话框提示信息
4. 15 25 3 ：对话框高度15行，宽度25个字符，3行高度用于数据项
5. 1 “one” off :选项的编号，文本，状态



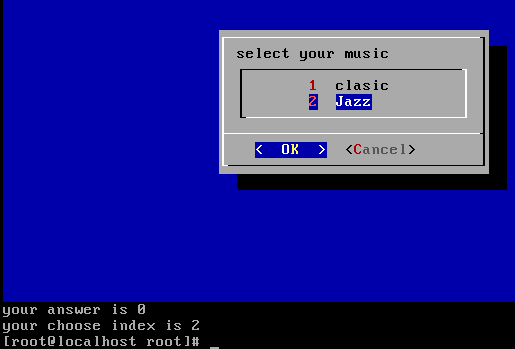
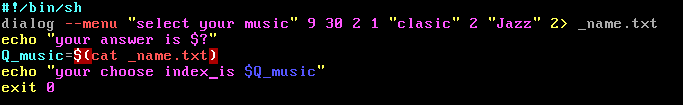
说明：（1）$? :保存前一个命令或脚本的退出码

1. dialog --clear : 清除对话框
2. Sleep 5 ：线程等待5秒
3. Dialog --infobox ：显示消息，需要手动关闭



说明：（1）2> \_name.txt ：重定向标准错误流2到文件\_name.txt中

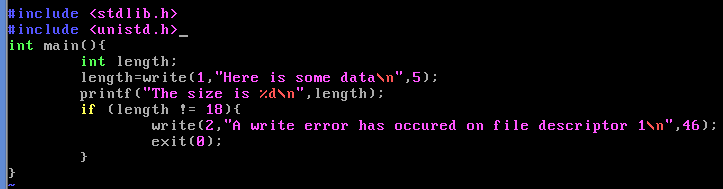
1. Q\_name=$(cat ./\_name.txt) ：給变量Q\_name赋值



1. 文件操作
2. 重要的3个设备文件：
3. /dev/console ：控制台，错误信息和诊断信息通常会被发到这个设备
4. /dev/tty ：控制终端
5. /dev/null ：空设备，所有写向这个设备的输出都将被丢弃
6. 系统调用
7. write 系统调用

#include <unisted.h>

size\_t write(int fildes,const void \*buf ,size\_t nbytes);

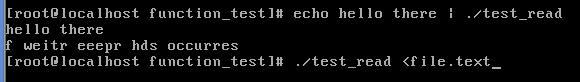
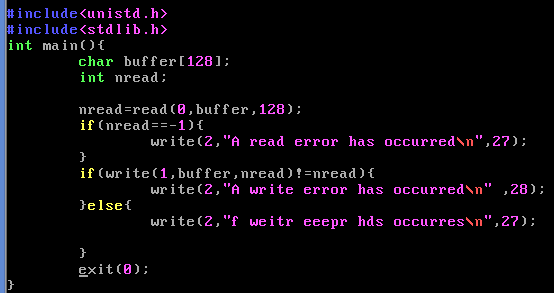


说明：1）write函数返回int型（size\_t）

1. 参数：int fildes ：表示开始写的位置
2. 参数 size\_t nbytes :表示需要写的个数
3. 参数 const void \*buf ：表示要写的流
4. 效果：在后台打印出写的流
5. read系统调用

#include <unistd.h>

size\_t read(int fildes,void \*buf,size\_t nbytes);



说明：1）write函数返回int型（size\_t）

1. 参数：int fildes ：表示开始读的位置
2. 参数 size\_t nbytes :表示需要读的个数
3. 参数 void \*buf ：表示存储读取的流
4. 输入方式：echo 管道命令输入， 文件重定向输入
5. open系统调用

#include <fcntl.h>

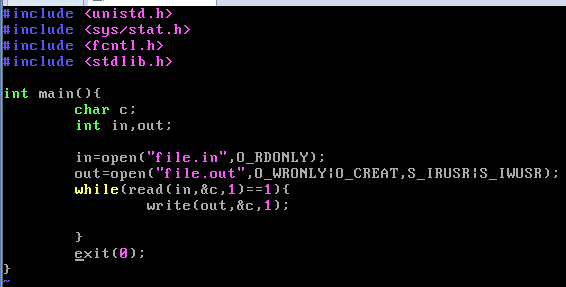
#include <sys/types.h>

#include <sys/stat.h>

int open(const char \*path, int oflags);

int open(const char \* path, int oflags, mode\_t mode);

举例：复制文本文件



1. umask ,系统变量，当文件被创建时，为文件的访问权限设定一个掩码
2. close系统调用

#include <unistd.h>

int close(int fildes);

1. ioctl系统调用：

#include <unistd.h>

int ioctl(int fildes,int cmd,.....);

1. lseek系统调用：对文件的读写指针进行设置

#include <unistd.h>

#include <sys/types.h>

off\_t lseek(int fildes,off\_t offset,int whence);

说明:offset :用来指定位置

Whence ：偏移值的用法，可取值为SEEK\_SET ,SEEK\_CUR, SEEK\_END

|  |  |
| --- | --- |
| SEEK\_SET | offset是一个绝对位置 |
| SEEK\_CUR | offset是相对于当前位置的一个相对位置 |
| SEEK\_END | offset是相对于文件尾的一个相对位置 |

1. fatat、stat、lstat 系统调用

#include <unistd.h>

#include <sys/stat.h>

#include <sys/types.h>

int fstat(int fildes, struct stat \*buf);

int stst(const char \*path, struct stat \*buf);

int lstat(const char \*path, struct stat \*buf); 将路径文件的信息存储在buf中

说明：stat结构的成员

|  |  |
| --- | --- |
| stat成员 | 说明 |
| st\_mode | 文件权限和文件类型信息 |
| st\_ino | 与该文件关联的inode |
| st\_dev | 保存文件的设备 |
| st\_uid | 文件属主的UID号 |
| st\_gid | 文件属主的GID号 |
| st\_atime | 文件上一次被访问的时间 |
| st\_ctime | 文件的权限、属主、组或内容上一次被改变的时间 |
| st\_mtime | 文件的内容上一次被修改的时间 |
| st\_nlink | 该文件上硬链接的个数 |

|  |  |
| --- | --- |
| St\_mod标志类型 |  |
| S\_IFBLK | 文件是一个特殊的块设备 |
| S\_IFDIR | 文件是一个目录 |
| S\_IFCRH | 文件是一个特殊的字符设备 |
| S\_IFIFO | 文件是一个FIFO（命名管道） |
| S\_IFREG | 文件是一个普通文件 |
| S\_FLNK | 文件是一个符号链接 |
| 其他标志 |  |
| S\_ISUID | 文件设置了SUID位 |
| S\_ISGID | 文件设置了SGID位 |
|  |  |

1. dup和dup2系统调用：复制文件描述符的方法

#include<unistd.h>

int dup(int fildes);

int dup2(int fildes, int fildes2);

1. 标准I/O库
2. fopen函数

#include <stdio.h>

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

|  |  |
| --- | --- |
| model | 说明 |
| “r” 或 “rb” | 以只读方式打开 |
| “w” 或 “wb” | 以写方式打开，并把文件长度截断为零 |
| “a” 或 “ab” | 以写方式打开，新内容加在文件尾 |
| “r+” 或 “rb+”或 “r+b” | 以更新的方式打开（读或写） |
| “w+” 或 “wb+”或 “w+b” | 以更新的方式打开，并把文件长度截短为领 |
| “a+” 或 “ab+”或 “a+b” | 以更新的方式打开，新的内容加在文件尾 |

1. fread函数：从文件流中读取数据到缓冲区

#include <stdio.h>

size\_t fread(void \*ptr , size\_t size , size\_t nitems , File \*stream);

说明：1）stream ：文件流

2）\*ptr ：缓冲区

1. fwrite函数：将缓冲区的数据写到文件流

#include <stdio.h>

size\_t fwrite(const void \*ptr , size\_t size , size\_t nitems , File \*stream);

说明：1）stream ：文件流

2）\*ptr ：缓冲区

1. fclose函数：关闭文件流，使所有尚未写出的数据都写出

#include <stdio.h>

int fclose(FILE \*stream);

1. fflush函数：使所有尚未写出的数据都写出

#include <stdio.h>

int fflush(FILE \*stream);

1. fseek 函数：与lseek系统调用对应的文件流函数

#include <stdio.h>

int fseek(FILE \*stream , long int offset , int whence);

1. fgetc、getc、getchar函数：从文件流中取出下一个字节

#include <stdio.h>

int fgetc(FILE \*stream);

int getc(FILE \*stream);

int getchar();

1. fputc、putc、putchar函数：把一个字符写到输出文件流中

#include <stdio.h>

int fputc(int c , FILE \*stream);

int putc(int c , FILE \*stream);

int putchar(int c);

1. fgets、gets函数：从输入文件流中读取一个字符串

#include <stdio.h>

char fputc(char \*s , int n , FILE \*stream);

char \*gets(char \*s);

1. printf、fprintf、sprintf函数：格式化输出

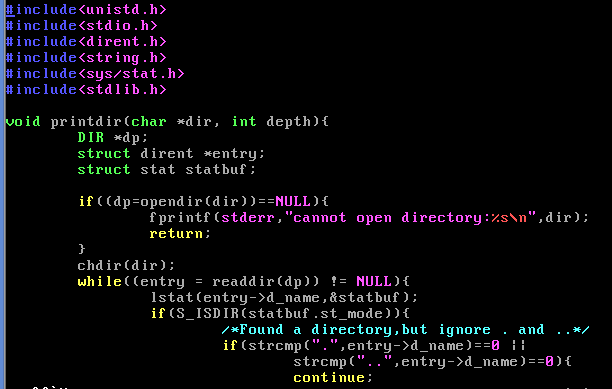
#include <stdio.h>

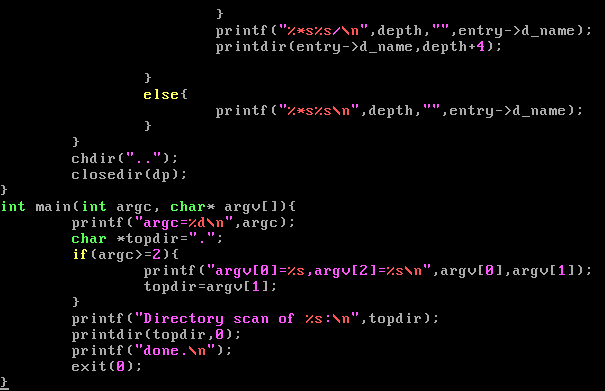
int printf(const char \*format,...);

int fprintf(FILE \*stream , conost char \*format,......);

int sprintf(char \*s , const char \*format,...);

|  |  |
| --- | --- |
| 转义输出格式 | 说明 |
| %d 或%i | 以十进制格式输出一个整数 |
| %o 或 %x | 以八进制或十六进制输出一个整数 |
| %c | 输出一个字符 |
| %s | 输出一个字符串 |
| %f | 输出一个（单精度）浮点数 |
| %e | 以科学计数法格式输出一个双精度浮点数 |
| %g | 以普通格式输出一个双精度浮点数 |



说明：该程序是输出目录的内容，以树结构展现

1）DIR :数据结构类型，目录

2）dirent， stat

3）opendir(dir) ：用于判断该目录存不存在

4）chdir(dir) ：切换到该目录

5）readdir(dp) ：读取该目录的下一个文件（或子目录）

6）entry->d\_name ：文件或目录的名字

7) lstat(entry->d\_name,&statbuf) ：将文件信息存储在statbuf结构中

8）S\_ISDIR(statbuf.st\_mode) ：判断该文件是文件还是目录

9) strcmp(“.” , entry\_d\_name) ：判断字符串相等

10）printf( “ %\*s%s\n”, depth, “”, entry->d\_name) :输出文件名，“%\*s”：格式输出

11）closedir(dp) ：关闭打开的文件

12）int main( int argc ,char \* argv[] ) ：argc 运行的参数个数， argv[]存储参数

1. 报告出现的错误

#include<stdio.h>

char \*strerror(int errnum); ：把错误代码映射为一个字符串

void perror(const char \*s); ：把errno变量中报告的当前错误映射到一个字符串，并把它输出到标准错误输出流

|  |  |
| --- | --- |
| errno错误代码的取值 | 说明 |
| EPERM | 操作不允许 |
| ENOENT | 文件或目录不存在 |
| EINTR | 系统调用被中断 |
| EIO | I/O错误 |
| EBUSY | 设备或资源忙 |
| EEXIST | 文件存在 |
| EINVAL | 无效参数 |
| EMFILE | 打开的文件过多 |
| ENODEV | 设备不存在 |
| EISDIR | 是一个目录 |
| ENOTDIR | 不是一个目录 |

1. /proc文件系统