实验一 Linux常用命令（一）

一、实验目的

1、熟悉Shell环境和Shell命令格式

2、熟悉使用Linux文件系统操作命令

3、熟悉使用文件权限管理命令

二、实验工具

1、Linux操作系统

三、实验内容和步骤

1、Shell环境和Shell命令格式

如果实验用计算机上Linux以图形方式启动，登录系统后打开终端窗口。方法：

桌面：右键菜单－Terminal（终端）；或：主菜单－系统工具－Terminal（终端）。

命令格式：command [-option(s)] [option argument(s)] [command argument(s)]

步骤1：使用who命令

步骤2：使用date命令

步骤3：使用上下键提取历史命令who，执行。

2、文件系统操作命令

（1）改变目录命令：cd 目标目录路径

步骤1：使用绝对路径依次进入Linux根目录下的所有子目录。

例如：cd /usr/bin

步骤2：使用绝对路径依次进入用户目录。

步骤3：使用相对路径进入子目录

1）使用绝对路径进入/lib目录

2）使用相对路径进入/lib/i868目录

3）使用相对路径退回到/lib目录

（2）列出并观察目录中的文件

步骤1：进入任意目录，使用无参数ls命令列出并观察其中所有文件；

步骤2：进入任意目录，使用通配符列出其中部分文件；

例如：cd /bin

ls c\*

步骤3：使用－l选项，列出目录中文件的各种属性，并观察识别这些属性。

步骤4：直接列出子目录中的文件。例如：ls i868

步骤5：仅列出子目录本身（－d）

步骤6：使用－F选项，显示文件类型标志。

步骤7：使用－a或－A选项，观察结果有什么变化。

步骤8：综合使用以上选项。

（3）目录的创建与删除

步骤1：在用户目录中，使用相对路径创建一个任意名称的目录，如“tmp”。

mkdir tmp

步骤2：删除该目录

rmdir tmp

步骤3：在根目录中，使用绝对路径在用户命令中创建一个子目录。

mkdir ~/tmp

步骤4：删除该目录

rmdir ～/tmp

步骤5：使用－p参数一次创建多级目录。如：

mkdir ～/tmp/dir1

步骤6：使用－p参数删除多级目录。如：

cd ~

rmdir tmp/dir1

（4）文件和目录拷贝命令：cp

步骤1：从/usr/include目录中，任选一个文件拷贝到用户目录的tmp子目录中

mkdir ~/tmp

cp /usr/include/signal.h ~/tmp

步骤2：从/usr/include目录中，任选一个文件拷贝到用户目录的tmp子目录中，并改变其文件名。

cp /usr/include/signal.h ~/tmp/signal.bak

步骤3：思考并试验，如果步骤1中不使用第一条命令，拷贝的结果是什么？

（5）文件移动或文件目录更名命令：mv

步骤1：将上面signal.bak更名为signal.h

cd ~/tmp

mv signal.bak signal.h

步骤2：将signal.h移动到上一层目录

mv signal.h ..

步骤3：将tmp目录更名为temp

mv ../tmp ../temp

步骤4：拷贝/usr/include中所有s开头的.h文件到/usr/temp目录

cp /usr/include/s\*.h /usr/temp

步骤5：将temp目录移动到tmp目录中

cd /usr

mkdir /tmp

mv temp tmp

步骤6：拷贝/usr/include/signal.h到/usr/tmp/temp目录中，观察提示。

步骤7：使用－f选项重复上述命令，观察是否有提示。

（6）建立符号链接命令：ln

步骤1：在用户目录中创建对/usr/include目录的符号链接

cd ~

ln –s /usr/include

ls include

步骤2：删除该链接

rm include

步骤3：观察/usr/include是否被删除。

步骤4：与步骤1相同，但命名链接目录为abc

ln -s /usr/include abc

ls abc

rm -f abc

3、文件权限管理命令

（1）改变文件权限（chmod）

步骤1：使用chmod给include目录中的任意文件的属主或属组或其它用户添加可执行的权限。

例：chmod u+x signal.h

步骤2：删除可执行权限

步骤3：设置属组权限为可读＋可执行。

步骤4：一次性设置文件的所有权限为可读＋可执行。

（2）改变权限屏蔽

步骤1：改变为屏蔽所有用户的可执行权限

步骤2：创建目录test，观察其权限。

步骤3：尝试进入test目录。是否能否进入，思考为什么？

实例：

$ 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 444 file          　　同上例

$ 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）字母法：chmod (u g o a) (+ - =) (r w x) (文件名)

以上是chmod的用法，每个括号是一个参数， 前三个括号主要放在一起使用即 chmod + 设置模式 + 文件名；下面先介绍这些参数的意义：

[u g o a] 含义

u user 表示该文件的所有者

g group 表示与该文件的所有者属于同一组( group )者，即用户组

o other 表示其它用户组

a all 表示这三者皆是

[+ - =] 含义

+ 增加权限

- 撤销权限

= 设定权限

[r w x] 含义

r read 表示可读取，对于一个目录，如果没有r权限，那么就意味着不能通过ls查看这个目录的内容。

w write 表示可写入，对于一个目录，如果没有w权限，那么就意味着不能在目录下创建新的文件。

x excute 表示可执行，对于一个目录，如果没有x权限，那么就意味着不能通过cd进入这个目录。

（4）数字法

数字法是基于字母法的表示，如果字母法懂了，数字法就好懂，并且易于使用。

用法：chmod + 数字组合 + 文件名

数字组合一般包含三个数字：

第一个数字对应字母法的用户u（user）

第二个数字对应字母法的用户g（group）

第三个数字对应字母法的用户o（other）

另外：

r (read) ----------------> 4

w (write) ----------------> 2

x (excute) ----------------> 1

示例详解：

数字法：chmod 777 文件名

对应的字母法： chmod u+rwx, g+rwx, o+rwx 文件名

第一个数字7：代表用户 u 的权限 rwx， 4 (r) + 2 (w) + 1 (x) = 7

第二个数字7：代表用户 g 的权限 rwx, 4 (r) + 2 (w) + 1 (x) = 7

第三个数字7：代表用户 o 的权限 rwx, 4 (r) + 2 (w) + 1 (x) = 7

实验二 Linux常用命令（二）

一、实验目的

1、熟悉使用Linux文件内容操作命令

2、熟悉使用进程管理命令

3、熟悉使用联机帮助命令

二、实验工具

1、Linux操作系统

三、实验内容和步骤

1、Linux文件内容显示命令

（1）cat命令，显示文件内容

步骤1：将/usr/include/stdio.h拷贝到~/tmp目录中，并进入tmp目录。

步骤2：使用简单cat命令显示其内容

步骤3：使用－n选项，输出行号；

步骤4：使用－s选项，合并空白行；

步骤5：使用－b选项，对非空白行编号；

（2）more命令，分页显示文件内容

步骤1：使用简单more命令显示stdio.h的内容

使用空格键、回车键和B键上下翻看内容。

步骤2：使用－num选项，控制每页的行数；

步骤3：使用－s选项，合并空白行；

步骤4：使用/pattern选项，先搜索字符串file，再显示；

（3）less命令，分页显示文件内容

步骤1：使用简单less命令显示stdio.h的内容

使用PAGEUP、PAGEDOWN、回车键和空格键上下翻看内容。

（4）head命令，显示文件开头内容

步骤1：使用head命令显示stdio.h的前10行内容

head –n 10 stdio.h

head --lines=10 stdio.h

步骤2：使用head命令显示stdio.h的前10个字符内容；

head -c 10 stdio.h

head --bytes=10 stdio.h

步骤3：使用－v选项，显示文件名；

（5）tail命令，显示文件最后的内容

步骤1：使用tail命令显示stdio.h的后10行内容

tail–n 10 stdio.h

tail--lines=10 stdio.h

步骤2：使用tail命令显示stdio.h的最后10个字符内容；

tail-c 10 stdio.h

tail--bytes=10 stdio.h

步骤3：使用－v选项，显示文件名；

（5）touch命令

步骤1：使用touch命令，修改stdio.h的文件时间，并使用ls查看结果。

touch -t 202109200810 stdio.h

步骤2：使用touch命令，创建一个名为test的空文件。

Touch test

2、Linux文件内容搜索、排序和比较命令

（1）grep命令，搜索文件内容

步骤1：在stdio.h中搜索并显示包含extern字符串的所有行

grep extern stdio.h

步骤2：在stdio.h中搜索所有宏定义的语句,

grep “#define\*” stdio.h

步骤3：在stdio.h中搜索所有以#开头的行

grep ^# stdio.h

步骤4：在stdio.h中搜索所有被包含的头文件

grep ‘\.h’ stdio.h

说明：头文件特点为字符串“.h”。但“.”是一个元字符，需要使用转义方式表示为“\.”。

步骤5：自己设想查找目标，并定义合理的字符串模式。

（2）sort命令，排序文件内容

步骤1：对stdio.h内容进行排序

sort stdio.h

步骤2：对stdio.h内容进行排序，并输出到文件test中, 查看排序结果

sort –o test stdio.h

more test

步骤3：同上，但要求合并重复行（使用-u选项）

步骤4：检查test和stdio.h是否已排序，观察输出结果

sort –c test

sort –c stdio.h

（3）uniq命令，合并重复行

步骤1：重复sort命令的步骤2，生成一个test文件

步骤2：使用uniq命令合并其中的重复行，并显示每行重复的次数

uniq －c test

（4）comm命令，比较两个文件

步骤1：准备两个文件:file1和file2

echo file1: > file1

echo line1 >> file1

echo file2: > file2

echo line1 >> file2

echo line2 >> file2

步骤2：比较两个文件，观察输出的三列的含义

comm. file1 file2

步骤3：比较两个文件，只输出其中相同的行。

comm. -12 file1 file2

步骤4：比较两个文件，只输出file2中独有的行。

comm. -13 file1 file2

（5）diff命令，比较两个文件的差异

步骤1：简单比较

diff file1 file2

步骤2：比较并输出ed脚本

diff –e file1 file2

（6）wc命令，统计文件内容

步骤1：全面统计，理解输出的结果

wc file1

wc file2

步骤2：仅统计词数

wc –w file2

步骤3：仅统计行数

wc –l file2

步骤4：仅统计字节术

wc –c file2

3、进程管理命令

（1）PS命令，查看进程状态

步骤1：简单ps

步骤2：列出系统中的所有进程

ps –e

步骤3：仅列出与当前TTY相关的进程

打开一个新的TTY

在原有tty和新tty中分别输入ps –a，观察比较结果。

步骤4：分别使用-x, -f, -l, -r, -u显示进程信息，比较结果。

（2）kill命令，终止进程

步骤1：查看tty1的PID。

步骤2：在tty0中输入杀进程的命令

kill –s kill PID

或：kill –s 9 PID

步骤3：列出所有进程信号，观察结果

kill -l

4、熟悉使用联机帮助命令

（1）man命令

步骤1：使用man命令查看ls命令的帮助信息

步骤2：使用man命令查看其它命令的帮助

（2）help命令

步骤1：使用help命令查看cd命令的帮助信息

四、实验总结

请写出自己遇到的问题。

实验三Linux用户管理实验

一、实验目的

1、了解Linux系统中的用户管理

2、掌握Linux系统中用户管理相关的常用命令

二、实验内容

1、Linux是一个支持多用户的操作系统，其中root用户必不可少，root用户具有超级管理权限。其他用户具有一般权限，Linux还支持用户组管理。

2、最常使用的一些命令：

（1）who命令：显示系统中当前有哪些用户在线，显示的了用户ID，终端，登录IP，上线时间，停留时间，CPU使用量，动作等等。

（2）passwd命令：用来更改用户的密码

（3）su命令：从一般用户切换到root用户，需要输入密码

（4）useradd命令：添加新用户

（5）userdel命令：删除用户

（6）usermod命令：修改用户账号

（7）groupadd命令：增加新的用户组

（8）groupdel命令：删除用户组

（9）groupmod命令：修改用户组

（10）gpasswd命令：管理组成员

（11）last命令：显示系统开机以来或是从每月初登录用户的信息

三、练习

1、在Linux下将root用户的密码修改为Linux#1

2、创建用户user01,user02,user03，创建用户组group1，group2

3、将用户user01、user02划归group1组，将用户user03划归group

四、实验总结

实验四 Linux网络配置

一、实验目的

1、熟悉Linux下的网络配置方法。

2、掌握Linux下的网络命令。

3、掌握TCP/IP网络配置文件及常见配置。

二、实验环境

具有Linux的计算机、局域网网络环境。

三、实验内容

1、使用setup菜单完成TCP/IP网络配置；

（1）在命令行运行setup，选择“Network Configuration”；

（2）选择“eth0”；

（3）取消“Use DHCP”；

（4）配置IP地址和子网掩码为192.168.1.15/255.255.255.0，网关为192.168.1.1；

（5）逐层退出；

（6）运行ifdown eth0禁用以太网卡；

（7）运行ifup eth0启用以太网卡并重新读取配置；

（8）使用ifconfig检查eth0网卡的IP网络配置。

请写出你自己的inet addr Bcast Mask。

2、网络配置常用命令

（1）hostname

1）查看本机的hostname

#hostname

1. 修改本机的hostname

#hostname newname

运行后立即生效(重新启动shell)，但是在系统重启后会丢失所做的修改，如果要永久更改系统的hostname，就要修改相关的配置置文件。

（2）ifconfig

#  ifconfig eth0 down

#  ifconfig eth0 up

（3）ifdown、ifup

#  ifdown eth0

#  ifconfig

# ifup eth0

#  ifconfig

（4）route

# route

# route add default gw  网关IP地址

# route

# route route del  default gw  网关IP地址

# route

（5）ping

#ping       www.baidu.com

（6）traceroute

#traceroute 10.20.4.123

（7）netstat //Netstat用于显示与IP、[TCP](https://baike.so.com/doc/3381795.html" \t "https://baike.so.com/doc/_blank)、UDP和ICMP协议相关的统计数据

#netstat –n  |  more

（8）nslookup //nslookup命令用于查询DNS的记录，查看域名解析是否正常，在网络故障的时候用来诊断网络问题。

#nslookup www.baidu.com

要求：掌握以上各个命令的含义、作用以及用法。

3、网络配置文件

#more    /etc/sysconfig/network

#more    /etc/hosts

#more    /etc/host.conf

#more    /etc/resolv.conf

#more    /etc/init.d/network

#more    /etc/network/inteerface

四、实验总结

请写出自己遇到的问题。

实验五文本编辑器vi的使用

1. 实验目的

1、掌握文本编辑器vi的使用方法

二、实验内容

1、vi的三种工作模式：

（1）命令模式：执行相关文本编辑命令

（2）输入模式：输入文本

（3）末行模式：实现查找、替换、保存、多文件操作等等功能

2、进入vi，直接在Shell提示符下键入vi [文件名称]，如果该文件在当前目录不存在，则vi创建之。

3、退出vi

（1）在命令模式下输入“：wq”，保存文件并退出vi

（2）若不需要保存文件，输入“：q”

（3）若文件已修改，但不保存，输入“：q！”强制退出vi

（4）其它一些不常用的方法在此省略。

4、命令模式下的常用编辑命令

（1）启动vi后，进入的是vi的命令模式

（2）按i键，进入输入模式，可以进行文本的编辑，在输入模式下，按esc键，可切换回命令模式

i：光标位置不变，可在光标左侧插入正文

a：光标位置向后退一格，可在光标左侧插入正文

o：在光标所在行的下一行增添新行

O：在光标所在行的上一行增添新行

I：光标跳到当前行的开头

A：光标跳到当前行的末尾

（3）光标的移动

k、j、h、l分别等同于上、下、左、右箭头键

Ctrl+b，向上翻一页

Ctrl+f，向下翻一页

nH，将光标移到屏幕的第n行

nL，将光标移到屏幕的倒数第n行

（4）删除文本

nX，删除光标所指向的后n个字符

D，删除光标右侧的所有字符（包括光标所指向的字符）

db，删除光标左侧的全部字符

ndd，删除当前行和当前行以后的n行内容

（5）粘贴和复制

p，将缓冲区的内容粘贴到当前字符的右侧

P，将缓冲区的内容粘贴到当前字符的左侧

yy，复制当前行到内存缓冲区

nyy，复制n行内容到内存缓冲区

（6）搜索字符串

/str1，正向搜索字符串str1

n，继续搜索

？str2，反向搜索字符串str2

（7）撤销和重复

u，撤销前一条命令的执行结果

.，重复最后一条命令

5、末行模式下的命令

：n，将光标移动到第n行

：nw file，将第n行写入file文件

：n，mw file，将第n行至第m行写入file文件

：w，将编辑的内容写入原始文件

：wq，将编辑的内容写入原始文件并退出编辑程序

：w file，将编辑的内容写入file文件，保持原有文件的内容不变

：f file，将当前文件重命名为file

：e file，编辑新文件file代替原有内容

：f，打印当前文件的状态，如文件的行数，光标所在的行号等

：!<命令>，执行相应shell命令

6、三种工作模式的切换

（1）在Linux shell下，键入vi或vi <文件名>进入命令模式

（2）在命令模式下，键入：进入末行模式

（3）在命令模式下，键入文本编辑命令如i，a，o等进入文本输入模式

（4）在文本输入模式下，按esc键进入命令模式

（5）在末行模式下，按backspace键或del键进入命令模式

（6）在末行模式下，键入q或wq，退出vi，返后到Linux shell下

三、实验总结

请写出自己遇到的问题。

实验六使用GCC编译C语言程序及使用GDB调试程序

一、实验目的

1、了解GNU gcc编译器

2、掌握使用GCC编译C语言程序的方法

3、掌握使用GDB调试程序的方法

二、实验内容

1、使用vi或其它文本编辑器，输入C语言程序，并保存为test.c。

2、在Linux shell下，输入命令gcc –g -o test test.c。

3、编译正确后，输入命令./test运行程序，观察程序运行结果。

4、若编译错误，根据提示信息，进入程序查错，再回到第二步，直至程序语法无误。

5、使用GDB调试程序。

附1：GCC使用方法和常用选项

使用GCC编译C程序生成可执行文件需要经历4个步骤：

预处理，这一步需要分析各种命令，如#define、#include、#ifdef等。Gcc调用cpp程序来进行预处理编译，这一步将根据输入文件产生汇编语言，gcc调用ccl进行编译工作汇编，这一步中将汇编语言作为输入，产生具有.o扩展名的目标文件，gcc调用as进行汇编工作连接，这一步中各目标文件.o被放在可执行文件的适当位置上，该程序引用的函数也放在可执行文件中，gcc调用ld来完成。

Gcc命令的基本用法为：gcc [option] [filename]，命令行选项指定的操作将在命令行上每个给出的文件上执行。例如：

Gcc –o prog main.c test1.c test2.c

其中，“-o prog”指定输出的可执行文件名为prog，如果没有指定-o参数，gcc将使用默认的可执行文件名a.out

Gcc的命令选项有许多项，但经常使用的几个选项是：

1、-c：只预处理，编译和汇编源程序，不进行连接

2、-o exefile，确定输出文件为exefile，如果没有该选项，默认输出为可执行文件a.out

3、-Dmacro或-Dmacro=defn，其作用类似于源程序代码中的#define

4、-O，对程序编译进行优化，编译后可执行文件的长度和执行时间缩短，但编译过程耗时变长，对主机性能要求较高。

5、-O2，比-O更好地优化

6、-g，告诉gcc产生能被GNU调试器使用的调试信息以便调试程序

7、-Idir，将dir目录加到搜寻头文件的目录列表中。并优先于在gcc中默认的搜寻目录.

附2：GDB的使用方法和常用命令

GDB能够观察一个程序在执行时的内部活动，或程序出错时发生了什

么，GDB的主要功能有以下几点：

1、设置运行环境和参数，运行指定程序

2、让程序在指定条件下停止和运行

3、在程序运行停止后，检查变量、内存或寄存器的值，查看程序运行情况

4、修改正在调试的程序的源代码，这样可以在线修正某个bug引起的问题，然后继续查找下一个bug

GDB的使用可以直接在shell命令行下输入gdb并回车，再在gdb命令行下指定要调试的程序。也可以用gdb filename在启动时指定要调试的程序名。

为了使GDB能够正常工作，必须使程序在编译时包含调试信息，即在

使用gcc时加-g选项即可。

在GDB启动后，就进入GDB命令方式，下面是一些常用的GDB命令：

1、file，载入程序命令，如要加载hello程序用file hello

2、quit，退出gdb命令，也可以用ctrl+d来退出

3、run，运行程序命令，如果程序需要参数，可在run指令后接着输入参数。

4、info，查看程序信息

5、list，列出源程序命令，如list FUNCTION，list LINENUM，list FILENAME：FUNCTION

6、break，设置断点，在程序被载入并且当前没有正在运行时，就能设置、修改、删除断点。如break FUNCTION，break LINENUM，break \*ADDRESS，break……if COND

7、watch，设置监视点命令，监视某个表达式或变量，当它被读或被写时让程序停下来，如watch EXPRESSION

8、print，显示表达式的值，如print exp

9、step或next，单步执行指令，step可以跟踪进入一个函数，而next指令不会

10、continue，继续执行命令

11、make，产生可执行文件，通过make不用退出gdb就可以重新产生可执行文件

12、shell命令，不离开gdb就可以执行shell命令

实验七Makefile实验

一、实验目的

1、了解Makefile的基本概念和基本结构

2、初步掌握编写简单Makefile的方法

3、了解递归Make的编译过程

4、初步掌握利用GNU Make编译应用程序的方法

二、实验原理

在Linux或Unix环境下，对于只含有几个源代码文件的小程序（如hello.c）的编译，可以手工键入gcc命令对源代码文件逐个进行编译；然而在大型的项目开发中，可能涉及几十到几百个源文件，采用手工键入的方式进行编译，则非常不方便，而且一旦修改了源代码，尤其头文件发生了的修改，采用手工方式进行编译和维护的工作量相当大，而且容易出错。所以在Linux或Unix环境下，人们通常利用GNU make工具来自动完成应用程序的维护和编译工作。实际上，GNU make工具通过一个称为Makefile的文件来完成对应用程序的自动维护和编译工作。Makefile是按照某种脚本语法编写的文本文件，而GNU make能够对Makefile中指令进行解释并执行编译操作。Makefile文件定义了一系列的规则来指定哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作。GNU make工作时的执行步骤如下：

1、读入所有的Makefile。  
2、读入被include的其它Makefile。  
3、初始化文件中的变量。  
4、推导隐晦规则，并分析所有规则。  
5、为所有的目标文件创建依赖关系链。  
6、根据依赖关系，决定哪些目标要重新生成。  
7、执行生成命令。

1-5步为第一个阶段，6-7为第二个阶段。第一个阶段中，如果定义的变量被使用了，那么，make会把其展开在使用的位置。但make并不会完全马上展开，make使用的是拖延战术，如果变量出现在依赖关系的规则中，那么仅当这条依赖被决定要使用了，变量才会在其内部展开。下面对makefile的相关问题进行简单介绍：

**1、Makefile的基本结构**

Makefile的一般结构：

**target……：dependency……**

**command……**

结构中各部分的含义：

（1）、target（目标）：一个目标文件，可以是Object文件，也可以是执行文件。还可以是一个标签（Label）。

（2）、dependency（依赖）：要生成目标文件（target）所依赖哪些文件

（3）、command（命令）：创建项目时需要运行的shell命令（**注：命令（command）部分的每行的缩进必须要使用Tab而不能使用多个空格**）。

Makefile实际上是一个文件的依赖关系，也就是说， target这一个或多个的目标文件依赖于dependency中的文件，其生成规则定义在命令command中。如果依赖文件（dependency）中有一个以上的文件比目标（target）文件要新的话，shell命令（command）所定义的命令就会被执行。这就是Makefile的规则。也就是Makefile中最核心的内容。

例如，假设有一个C源文件test.c，该源文件包含有自定义的头文件test.h，则目标文件test.o明确依赖于两个源文件：test.c和test.h。如果只希望利用gcc命令来生成test.o目标文件，这时，就可以利用如下的makefile来定义test.o的创建规则：

#This makefile just is a example.

test.o: test.c test.h

gcc –c test.c

从上面的例子注意到，第一个字符为#的行表示注释行。第一个非注释行指定test.o为目标，并且依赖于test.c和test.h文件。随后的行指定了如何从目标所依赖的文件建立目标。

当test.c或test.h文件在编译之后又被修改，则make工具可自动重新编译test.o，如果在前后两次编译之间，test.c和test.h均没有被修改，而且test.o还存在的话，就没有必要重新编译。这种依赖关系在多源文件的程序编译中尤其重要。通过这种依赖关系的定义，make工具可避免许多不必要的编译工作。

一个makefile文件中可定义多个目标，利用make target命令可指定要编译的目标，如果不指定目标，则使用第一个目标。通常，makefile中定义有clean目标，可用来清除编译过程中的中间文件

# This makefile just is a example.

test.o: test.c test.h

gcc -c  test.c

clean:

rm -f \*.o

运行make clean时，执行rm –f \*.o命令，删除编译过程中生成的所有中间文件。

**2、Makefile的基本内容**

Makefile一般包括包含：显式规则、隐晦规则、变量定义、文件指示和注释等五个内容。

（1）、显式规则：显式规则说明如何生成一个或多个的目标文件。这是由Makefile的书写者明显指出，要生成的文件，文件的依赖文件，生成的命令。

（2）、变量定义。在Makefile中可以定义一系列的变量，变量一般都是字符串，当Makefile被执行时，变量的值会被扩展到相应的引用位置上。

（3）、**隐含规则**：由于GNU make具有自动推导功能，所以隐晦规则可以比较粗糙地简略地书写Makefile，然后由GNU make的自动推导功能完成隐晦规则的内容。

（4）、文件指示。其包括了三个部分，一个是在一个Makefile中引用另一个Makefile，就像C语言中的include一样；另一个是指根据某些情况指定Makefile中的有效部分，就像C语言中的预编译#if一样；还有就是定义一个多行的命令。

（5）、注释。Makefile中只有行注释，和UNIX的Shell脚本一样，其注释是用“#”字符，如果你要在你的Makefile中使用“#”字符，可以用反斜框进行转义，如：“\#”。

**2.1 Makefile中的变量**

（1）、Makefile中定义的变量，与C/C++语言中的宏一样，代表一个文本字串，在Makefile被执行时候变量会自动地展开在所使用的地方。Makefile中的变量可以使用在“目标”，“依赖目标”，“命令”或Makefile的其它部分中。

（2）、Makefile中变量的命名字可以包含字符、数字，下划线（可以是数字开头），但不应该含有“:”、“#”、“=”或是空字符（空格、回车等）。

（3）、Makefile中变量是大小写敏感的，“foo”、“Foo”和“FOO”是三个不同的变量名。传统的Makefile的变量名是全大写的命名方式

（4）、变量在声明时需要给予初值，而在使用时，需要在变量名前加上“$”符号

# makefile test for hello program

#written by Emdoor

CC=gcc

CFLAGS=

***OBJS=hello.o***

all: hello

hello: ***$(OBJS)***

$(CC) $(CFLAGS) ***$(OBJS)*** -o hello

hello.o: hello.c

$(CC) $(CFLAGS) -c hello.c -o ***$(OBJS)***

clean:

rm –rf hello \*.o

上面自定义变量OBJS表示hello.o，当makefile被执行时，变量会在使用它的地方精确地展开，就像C/C++中的宏一样。上述makfile变量展开后的形式为：

# makefile test for hello program

#written by Emdoor

CC=gcc

CFLAGS=

***OBJS=hello.o***

all: hello

hello: ***hello.o***

**gcchello.o** -o hello

hello.o: hello.c

**gcc** -c hello.c -o **hello.o**

clean:

rm -rf hello \*.o

**GNU make的主要预定义变量**

GNU make 有许多预定义的变量，这些变量具有特殊的含义，可在规则中使用。以下给出了一些主要的预定义变量，除这些变量外，GNU make 还将所有的环境变量作为自己的预定义变量。

**$@** ——表示规则中的目标文件集。在模式规则中，如果有多个目标，那么，"$@"就是匹配于目标中模式定义的集合。

**$%** ——仅当目标是函数库文件中，表示规则中的目标成员名。例如，如果一个目标是"foo.a(bar.o)"，那么，"$%"就是"bar.o"，"$@"就是"foo.a"。如果目标不是函数库文件（Unix下是[.a]，Windows下是[.lib]），那么，其值为空。

**$<** ——依赖目标中的第一个目标名字。如果依赖目标是以模式（即"%"）定义的，那么"$<"将是符合模式的一系列的文件集。注意，其是一个一个取出来的。

**$?** ——所有比目标新的依赖目标的集合。以空格分隔。

**$^** ——所有的依赖目标的集合。以空格分隔。如果在依赖目标中有多个重复的，那个这个变量会去除重复的依赖目标，只保留一份。

$+ ——这个变量很像"$^"，也是所有依赖目标的集合。只是它不去除重复的依赖目标。

**命令的变量。**

AR 函数库打包程序。默认命令是“ar”。

AS    汇编语言编译程序。默认命令是“as”。

CC C语言编译程序。默认命令是“cc”。

CXX  C++语言编译程序。默认命令是“g++”。

CO    从 RCS文件中扩展文件程序。默认命令是“co”。

CPP C程序的预处理器（输出是标准输出设备）。默认命令是“$(CC) –E”。

FC Fortran 和 Ratfor 的编译器和预处理程序。默认命令是“f77”。

GET 从SCCS文件中扩展文件的程序。默认命令是“get”。

LEX  Lex方法分析器程序（针对于C或Ratfor）。默认命令是“lex”。

PC    Pascal语言编译程序。默认命令是“pc”。

YACC    Yacc文法分析器（针对于C程序）。默认命令是“yacc”。

YACCR    Yacc文法分析器（针对于Ratfor程序）。默认命令是“yacc –r”。

MAKEINFO    转换Texinfo源文件（.texi）到Info文件程序。默认命令是“makeinfo”。

TEX    从TeX源文件创建TeX DVI文件的程序。默认命令是“tex”。

TEXI2DVI    从Texinfo源文件创建军TeX DVI 文件的程序。默认命令是“texi2dvi”。

WEAVE    转换Web到TeX的程序。默认命令是“weave”。

CWEAVE     转换C Web 到 TeX的程序。默认命令是“cweave”。

TANGLE    转换Web到Pascal语言的程序。默认命令是“tangle”。

CTANGLE    转换C Web 到 C。默认命令是“ctangle”。

RM 删除文件命令。默认命令是“rm –f”。

**命令参数变量：**

下面的这些变量都是相关上面的命令的参数。如果没有指明其默认值，那么其默认值都是空。  
ARFLAGS     函数库打包程序AR命令的参数。默认值是“rv”。  
ASFLAGS     汇编语言编译器参数。（当明显地调用“.s”或“.S”文件时）。  
CFLAGS     C语言编译器参数。  
CXXFLAGS     C++语言编译器参数。  
COFLAGS     RCS命令参数。  
CPPFLAGS     C预处理器参数。（ C 和 Fortran 编译器也会用到）。  
FFLAGS    Fortran语言编译器参数。  
GFLAGS     SCCS “get”程序参数。  
LDFLAGS    链接器参数。（如：“ld”）  
LFLAGS     Lex文法分析器参数。  
PFLAGS Pascal语言编译器参数。

RFLAGS     Ratfor 程序的Fortran 编译器参数。  
YFLAGS     Yacc文法分析器参数。

**2.2隐含规则**

GNU make 包含有一些内置的或隐含的规则，这些规则定义了如何从不同的依赖文件建立特定类型的目标。

GNU make 支持两种类型的隐含规则：

（1）、后缀规则（Suffix Rule）。后缀规则是定义隐含规则的老风格方法。后缀规则定义了将一个具有某个后缀的文件（例如，.c 文件）转换为具有另外一种后缀的文件（例如，.o 文件）的方法。每个后缀规则以两个成对出现的后缀名定义，例如，将 .c 文件转换为 .o 文件的后缀规则可定义为：

.c.o:

$(CC) $(CCFLAGS) $(CPPFLAGS) -c -o $@ $<

（2）、 模式规则（pattern rules）。这种规则更加通用，因为可以利用模式规则定义更加复杂的依赖性规则。模式规则看起来非常类似于正则规则，但在目标名称的前面多了一个 % 号，同时可用来定义目标和依赖文件之间的关系，例如下面的模式规则定义了如何将任意一个 X.c 文件转换为 X.o 文件：

%.c:%.o

$(CC) $(CCFLAGS) $(CPPFLAGS) -c -o $@ $<

**2.3 文件引用**

在Makefile使用include关键字可以把别的Makefile包含进来，这很像C语言的#include，被包含的文件会原模原样的放在当前文件的包含位置。

例如：有这样几个Makefile：a.mk、b.mk、c.mk，还有一个文件叫foo.make，以及一个变量$(bar)，其包含了e.mk和f.mk，那么，下面的语句：

    include foo.make \*.mk $(bar)

    等价于：

    include foo.make a.mk b.mk c.mk e.mk f.mk

make命令开始时，会把找寻include所指出的其它Makefile，并把其内容安置在当前的位置。如果文件都没有指定绝对路径或是相对路径的话，make首先会在当前目录下寻找，如果当前目录下没有找到，那么，make还会在下面的几个目录下找：

（1）如果make执行时，有“-I”或“--include-dir”参数，那么make就会在这个参数所指定的目录下去寻找。  
（2）、如果目录<prefix>/include（一般是：/usr/local/bin或/usr/include）存在的话，make也会去找。

如果有文件没有找到的话，make会生成一条警告信息，但不会马上出现致命错误。它会继续载入其它的文件，一旦完成makefile的读取，make会再重试这些没有找到，或是不能读取的文件，如果还是不行，make才会出现一条致命信息。

**2.4 Makefile中的函数**

在Makefile中可以使用函数来处理变量，从而让命令或规则更为的灵活和具有智能，函数调用，很像变量的使用，也是以“$”来标识的，函数调用后，函数的返回值可以当做变量来使用。

例如：'wildcard' 的函数，可以展开成一列所有符合由其参数描述的文件名。文件间以空格间隔。语法如下：

$(wildcard PATTERN...)

用'wildcard' 函数找出目录中所有的".c"文件：SOURCES = $(wildcard \*.c)。实际上，GNU make还是许多如**字符串处理函数、文件名操作函数等**其他函数

**3、运行 make**

**3.1 Make的执行**

一般来说，最简单的就是直接在命令行下输入make命令，GNU make找寻默认的Makefile的规则是在当前目录下依次找三个文件——“GNUmakefile”、“makefile”和“Makefile”。其按顺序找这三个文件，一旦找到，就开始读取这个文件并执行，也可以给make命令指定一个特殊名字的Makefile。要达到这个功能，要求使用make的“-f”或是“--file”参数，例如：make –f Hello.makefile

3.2 GNU make命令选项

 GNU make 命令还有一些其他选项，下面给出了这些选项。

命令行选项              含义

-C DIR           在读取 makefile 之前改变到指定的目录 DIR。

-f FILE           以指定的 FILE 文件作为 makefile。

-h                   显示所有的 make 选项。

-i                  忽略所有的命令执行错误。

-I DIR             当包含其他 makefile 文件时，可利用该选项指定搜索目录。

-n                   只打印要执行的命令，但不执行这些命令。

-p                 显示 make 变量数据库和隐含规则。

-s                 在执行命令时不显示命令。

-w                在处理 makefile 之前和之后，显示工作目录。

-W FILE       假定文件 FILE 已经被修改。

【实验仪器】

1、装有Linux操作系统的PC机一台；

2、实验开发平台一套

【实验内容】

一、使用命令行的方式手动编译程序方法

1、利用文本编辑器创建hello.c文件

//hello.c

//written by Emdoor

#include <stdio.h>

int main()

{

printf("Welcome Emdoor!\n");

return 1;

}

2、手动编译hello应用程序

在hello.c的目录的终端下输入：

[root@local]$ gcc –c hello.c

[root@local]$gcc hello.o –o hello

通过ls命令查看当前目录下是否生成源代码hello.c的object文件hello.o和可执行文件hello，运行可执行文件hello。查看一下运行结果。

[root@local]$./hello

3、修改hello.c文件，重新手动编译应用程序。

4、删除hello.o和hello文件

[root@local]$rm –f hello.o

[root@local]$rm –f hello

二、利用 GNU make 自动编译应用程序方法

1. 利用文本编辑器创建一个makefile文件，并将其保存到与hello.c相同的目录下。

# makefile test for hello program

#written by Emdoor

CC=gcc

CFLAGS=

all: hello

hello: hello.o

$(CC) $(CFLAGS) hello.o -o hello

hello.o: hello.c

$(CC) $(CFLAGS) -c hello.c -o hello.o

clean:

rm –rf hello \*.o

2、先后执行如下命令

[root@local]$make

[root@local]$ls

[root@local]$./hello

查看并记录所生成的文件和运行的结果。

3、执行make clean命令：

[root@local]$make clean

4、修改hello.c文件，重复第2、3步操作，查看并记录所生成的文件和运行结果，并与手动编译进行比较，写出你的结论。

5、重新编辑makefile文件（斜黑体表示修改部分）

# makefile test for hello program

#written by Emdoor

CC=gcc

CFLAGS=

***OBJS=hello.o***

all: hello

hello: ***$(OBJS)***

$(CC) $(CFLAGS) ***$^ -o $@***

hello.o: hello.c

$(CC) $(CFLAGS) ***–c $< -o $@***

clean:

rm –rf hello \*.o

6、重复第2，3步操作，查看并记录所生成的文件和运行的结果。比较这两种操作，写出你的结论。同时指出$^ 、$@、$<在上述Makefile中的含义。

三、多个.c文件的编译

1、创建文件hello1.c、hello2.c、hello.h和makefile

//hello1.c

//written by Emdoor

#include <stdio.h>

int main()

{

printf("Welcome Emdoor!\n");

test2();

return 1;

}

//hello2.c

//written by Emdoor

＃include "hello2.h"

#include <stdio.h>

void test2(void)

{

printf("Welcome Emdoor! –hello2\n");

}

//hello2.h

//written by Emdoor

void test2(void);

# makefile test for multi files program

#written by Emdoor

CC=gcc

CFLAGS=

OBJS=hello1.o hello2.o

all: hello

hello: $(OBJS)

$(CC) $(CFLAGS) $^ -o $@

hello1.o: hello1.c

$(CC) $(CFLAGS) -c $< -o $@

hello2.o: hello2.c

$(CC) $(CFLAGS) -c $< -o $@

clean:

rm –rf hello \*.o

2、先后执行如下命令

[root@local]$make

[root@local]$ls

[root@local]$./hello

查看并记录所生成的文件和运行的结果, 写出你的结论。

3、修改makefile文件（斜黑体表示修改部分）

# makefile test for multi files program

#written by Emdoor

CC=gcc

CFLAGS=

***CFILES=$(wildcard \*.c)***

OBJS=***$(CFILES:%.c=%.o)***

all: hello

hello: $(OBJS)

$(CC) $(CFLAGS) ***–o hello $(OBJS)***

***.c.o:***

***$(CC) –c $<***

clean:

rm –rf hello \*.o

4、重复第2步操作，查看并记录所生成的文件和运行的结果, 写出你的结论。并指出wildcard、.c.o的含义和变量CFILES代表的内容。

【思考题】

1. 根据提供的Linux操作系统源码中得Makefile结构，分析在工程中多级目录中存在着多个makefile时，编译的顺序如何？
2. 根据Makefile中变量定义规则，如果实验中的hello.c文件编译到目标平台中运行，应该怎样修改Makefile中变量参数？

实验八 shell编程

一、实验目的

1、掌握shell程序的建立和执行方式；

2、掌握shell中各种变量和函数的使用；

3、熟练掌握shell程序设计中各种控制结构语句是使用；

4、了解shell编程中特殊字符的含义。

二、实验工具

1、Linux操作系统

三、实验内容

（一）shell程序的建立和执行

1．依次执行下列命令，理解反馈信息的意义。

$ date

$ pwd

$ cd ..

$ cd

2．建立shell程序脚本

$ vi ex1

date

pwd

cd ..

cd

存盘退出。

3．检查文件是否存在。

$ ls

4．执行shell程序ex1，显示反馈信息与单步命令操作相同。

方式一：输入定向到shell脚本

$ bash < ex1

方式二：以脚本名作为参数

$ bash ex1

$ bash ex1 /usr

方式三：将shell脚本的权限设置为可执行，然后在提示符下直接执行它

步骤1：$ chmod a+x ex1

步骤2：$ mv ex1 /usr/bin

步骤３：$ ex1

$ ex1 /usr

5. 在文本编辑器中录入下面shell程序，保存为ex2，然后执行之。

#!/bin/bash

# If no arguments, then listing the current directory.

# Otherwise, listing each subdirectory.

if test $# = 0

then ls .

else

for i

do

ls -l $i | grep '^d'

done

fi

（二）shell变量

1．用户定义的变量。

单步执行下述命令，练习变量赋值，理解反馈信息。

$ dir=/home/mengqc/ex1

$ echo $dir

$ echo dir

$ today=Sunday

$ echo $today $Today

$ str="Happy New Year."

$ echo "Wish You $str"

2．read命令。

（1）单步执行下述命令：

$ read name -----输入read命令

zhangsan -----输入name的值

$ echo "Your Name is $name."

Your Name is zhangsan -----显示输出的结果

$ read a b c -----read命令有三个参数

cuit cn edu -----输入三个字符串，中间以空格隔开

$ echo "Email : $a. $c. $b"

Email : cuit.edu.cn -----显示输出结果

（2）将上述四个单步命令编辑为shell程序，取名ex3。

$ vi ex3 (输入四行命令，编后存盘)。

（3）运行shell程序ex3。

$ bash ex3

3．特殊变量

（1）建立一个内容如下的shell程序ex4：

echo “Program name is $0”

echo “There are totally $# parameters passed to this program”

echo “The last is $?”

echo “The parameters are $\*”

（2）按如下执行程序ex4，观察反馈信息：

$ bash ex4 this is a test program

（三）控制结构

1．if语句

（1）

理解并建立shell程序ex5：

Echo “The current directory is `pwd`”

if test - f "$1"　　　　　　 # 如果位置参数$1对应的文件

then echo "$1 is an ordinary file." # 是普通文件“- F”则显示本行

else echo "$1 is not an　ordinary file." # 否则显示本行

fi

执行ex5，并理解反馈信息：

$ bash ex5 ex4

（2）

理解并建立shell程序ex6：

if test - f "$1"

then cat $1

else if test - d "$1"

then (cd $1 ;cat \* )

else echo "$1 is neither a file nor a directory."

fi

fi

执行ex6，并理解反馈信息。

2．while语句

（1）

理解并建立shell程序ex7：

while [ $1 ]

do

if [ -f $1 ]

then echo "display : $1"

cat $1

else echo " $1 is not a file name."

fi

shift #后续位置参数左移

done

执行ex7，并理解反馈信息。

（2）编写求前五个偶数之和的shell程序ex8：

#!/bin/bash

loopcount=0

result=0

while [ $loopcount -lt 5 ] #判断符号的前后应该有空格

do

loopcount=`expr $loopcount + 1` #运算符左右要留空格

increment=`expr $loopcount \\* 2`

result=`expr $result + $increment` #等号左右端不能留空格

done

echo “result is $result”

3．for语句

（1）

理解并建立shell程序ex9：

for day in Monday Wednesday Friday Sunday

do

echo $day

done

执行ex9，并理解反馈信息。

（2）

理解并建立shell程序ex10：

mkdir backup

for filename in ‘ls’

do

cp $filename backup/$filename

if ($? -ne 0) then

echo “copy for $filename failed”

fi

done

执行ex10，并理解反馈信息。

4．Select语句

理解并建立shell程序ex11：

select item in Continue Finsh

do

if [ $item = “Finsh” ]; then

break

fi

done

执行ex11，并理解反馈信息。

（四）函数

1．理解并建立shell程序ex12：

displaymonth() {

case $1 in

01 | 1) echo “Month is January”;;

02 | 2) echo “Month is February” ;;

03 | 3) echo “Month is March” ;;

04 | 4) echo “Month is April” ;;

05 | 5) echo “Month is May” ;;

06 | 6) echo “Month is June” ;;

07 | 7) echo “Month is July” ;;

08 | 8) echo “Month is August” ;;

09 | 9) echo “Month is September” ;;

10) echo “Month is October” ;;

11) echo “Month is November” ;;

12) echo “Month is December” ;;

\*) echo “Invalid parameter” ;;

esac

}

displaymonth $1

执行ex12，并理解反馈信息。

$ bash ex12 3

$ bash ex12 10