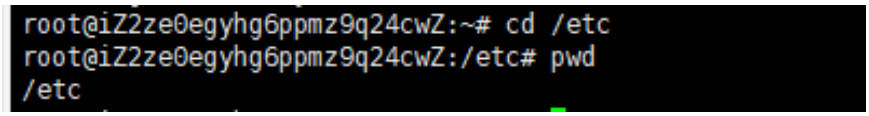
Linux 实验

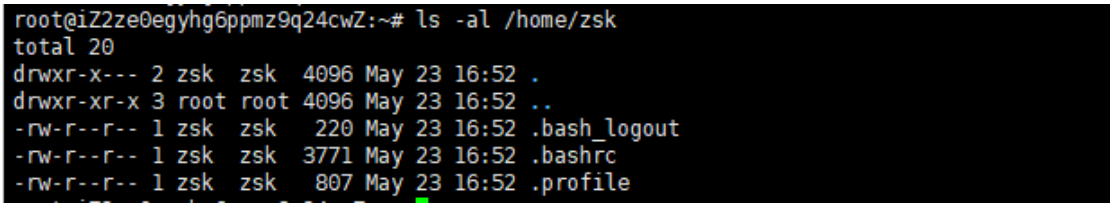
2021213142 张世康 计 5

实验一 Linux 常用命令（一）

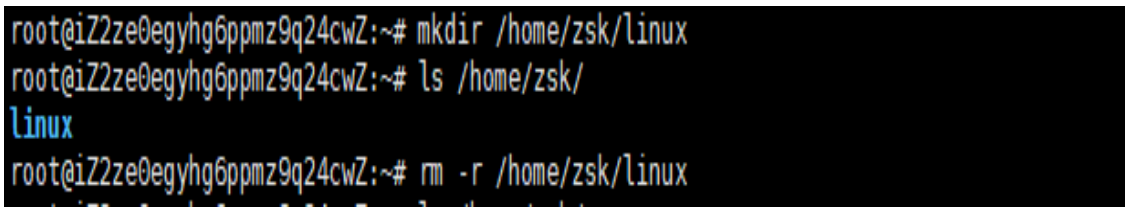
1. 使用命令切换到 /etc 目录，并显示当前工作目录路径；



2. 使用命令显示 /home/{用户名} 目录下所有文件目录的详细信息，包括隐藏文件；



3. 使用命令创建目录 /home/{用户名}/linux ，然后删除该目录；



4. 使用命令 cat用输出重定向在 /home/{用户名} 目录下创建文件 foo ，文件内容为 "Hello, Linux!" ，并

查

看该文件的内容;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# echo "Hello, Linux!" > /home/zsk/foo
root@iZ2ze0egyhg6ppmz9q24cwZ:~# cat /home/zsk/foo
Hello, Linux!
```

5. 使用命令创建目录 /home/{用户名}/foo.bak ， 然后将 /home/{用户名}/foo 文件复制到该目录下， 最后将该目录及其目录下的文件一起删除;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# mkdir /home/zsk/foo.bak
root@iZ2ze0egyhg6ppmz9q24cwZ:~# cp /home/zsk/foo /home/zsk/foo.bak/
root@iZ2ze0egyhg6ppmz9q24cwZ:~# rm -r /home/zsk/foo.bak
```

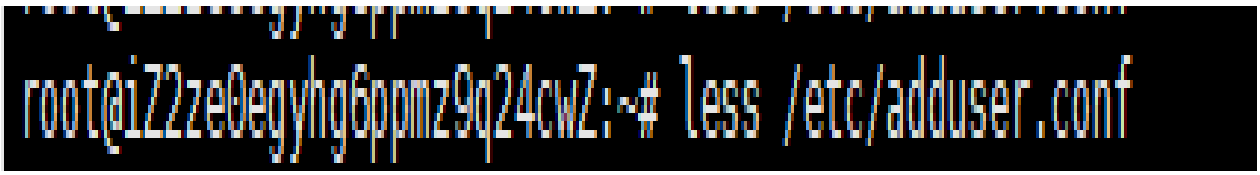
6. 查看文件 /etc/adduser.conf 的前 3 行内容, 查看文件 /etc/adduser.conf 的最后 5 行内容;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# head -n 3 /etc/adduser.conf
# /etc/adduser.conf: `adduser' configuration.
# See adduser(8) and adduser.conf(5) for full documentation.

root@iZ2ze0egyhg6ppmz9q24cwZ:~# tail -n 5 /etc/adduser.conf
# check user and group names also against this regular expression.
#NAME_REGEX="^[a-z][-a-z0-9_]*\$"

# use extrausers by default
#USE_EXTRAUSERS=1
```

7. 分屏查看文件 /etc/adduser.conf 的内容;



```
# /etc/adduser.conf: `adduser' configuration.
# See adduser(8) and adduser.conf(5) for full documentation.

# The DSHELL variable specifies the default login shell on your
# system.
DSHELL=/bin/bash

# The DHOME variable specifies the directory containing users' home
# directories.
DHOME=/home

# If GROUPHOMES is "yes", then the home directories will be created as
# /home/groupname/user.
GROUPHOMES=no

# If LETTERHOMES is "yes", then the created home directories will have
# an extra directory - the first letter of the user name. For example:
# /home/u/user.
LETTERHOMES=no

# The SKEL variable specifies the directory containing "skeletal" user
# files; in other words, files such as a sample .profile that will be
# copied to the new user's home directory when it is created.
SKEL=/etc/skel

# FIRST_SYSTEM_[GU]ID to LAST_SYSTEM_[GU]ID inclusive is the range for UIDs
# for dynamically allocated administrative and system accounts/groups.
# Please note that system software, such as the users allocated by the base-passwd
# package, may assume that UIDs less than 100 are unallocated.
FIRST_SYSTEM_UID=100
LAST_SYSTEM_UID=999

FIRST_SYSTEM_GID=100
LAST_SYSTEM_GID=999

# FIRST_[GU]ID to LAST_[GU]ID inclusive is the range of UIDs of dynamically
# allocated user accounts/groups.
FIRST_UID=1000
LAST_UID=59999

FIRST_GID=1000
LAST_GID=59999

# The USERGROUPS variable can be either "yes" or "no".  If "yes" each
# created user will be given their own group to use as a default.  If
# "no", each created user will be placed in the group whose gid is
# USERS_GID (see below).
USERGROUPS=yes

# If USERGROUPS is "no", then USERS_GID should be the GID of the group
# `users' (or the equivalent group) on your system.
USERS_GID=100

# If DIR_MODE is set, directories will be created with the specified
# mode. Otherwise the default mode 0755 will be used.
```

8. 使用命令 cat 用输出重定向在 /home/{用户名}目录下创建文件 bar.txt ， 文件内容为：

google 110 5000

baidu 100 5000

guge 50 3000

sohu 100 4500

其中：第 1 列为公司名称，第 2 列为公司人数，第 3 列为员工平均工资。

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# echo 'google 110 5000
baidu 100 5000
guge 50 3000
sohu 100 4500' > /home/zsk/bar.txt
root@iZ2ze0egyhg6ppmz9q24cwZ:~# cat /home/zsk/bar.txt
google 110 5000
baidu 100 5000
guge 50 3000
sohu 100 4500
```

使用 sort 命令完成下列排序:

（1） 按公司字母顺序排序;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sort /home/zsk/bar.txt
baidu 100 5000
google 110 5000
guge 50 3000
sohu 100 4500
```

（2） 按公司人数排序;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sort -n -k 2 /home/zsk/bar.txt
guge 50 3000
baidu 100 5000
sohu 100 4500
google 110 5000
```

（3） 按公司人数排序，人数相同的按照员工平均工资升序排序;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sort -n -k 2 -k 3 /home/zsk/bar.txt
guge 50 3000
sohu 100 4500
baidu 100 5000
google 110 5000
```

（4）按员工工资降序排序，如工资相同，则按公司人数升序排序；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sort -n -r -k 3 -k 2 /home/zsk/bar.txt
google 110 5000
baidu 100 5000
sohu 100 4500
guge 50 3000
```

（5）从公司英文名称的第 2 个字母开始进行排序

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sort -k 1.2 /home/zsk/bar.txt
baidu 100 5000
sohu 100 4500
google 110 5000
guge 50 3000
```

实验二 Linux 常用命令（二）

1. 查找指定文件

（1）在用户目录下新建目录 baz，在 baz 下新建文件 qux，并写如任意几行内容;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# mkdir ~/baz
echo "This is line 1" > ~/baz/qux
echo "This is line 2" >> ~/baz/qux
echo "This is line 3" >> ~/baz/qux
```

（2）在用户目录下查找文件 qux，并显

示该文件位置信息;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# find ~/ -name qux
/root/baz/qux
```

（3）统计文件 qux 中所包含内容的行数、字数和字节数;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# wc ~/baz/qux
 3 12 45 /root/baz/qux
```

(4) 在用户目录下查找文件 qux ，并删除该文件;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# find ~/ -name qux -delete
```

(5) 查看文件夹 baz 内容, 看一下是否删除了文件 qux 。

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# ls ~/baz
```

2. 文件打包

（1）在用户目录下新建文件夹 path1 ，在 path1 下新建文件 file1 和 file2 ;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# mkdir ~/path1
echo "This is file1" > ~/path1/file1
echo "This is file2" > ~/path1/file2
```

（2）在用户目录下新建文件夹 path2，在 path2 下新建文件 file3；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# mkdir ~/path2
echo "This is file3" > ~/path2/file3
```

（3）在用户目录下新建文件 file4；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# echo "This is file4" > ~/file4
```

（4）在用户目录下对文件夹 path1 和 file4 进行打包，生成文件 package.tar；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# tar cvf package.tar ~/path1 ~/file4
tar: Removing leading `/' from member names
/root/path1/
/root/path1/file1
tar: Removing leading `/' from hard link targets
/root/path1/file2
/root/file4
```

（5）查看包 package.tar 的内容；

```
tar: Removing leading `/' from hard link targets
/root/path1/file2
/root/file4
root@iZ2ze0egyhg6ppmz9q24cwZ:~# tar tvf package.tar
drwxr-xr-x root/root         0 2023-05-23 17:35 root/path1/
-rw-r--r-- root/root        14 2023-05-23 17:35 root/path1/file1
-rw-r--r-- root/root        14 2023-05-23 17:35 root/path1/file2
-rw-r--r-- root/root        14 2023-05-23 17:36 root/file4
```

（6）向包 package.tar 里添加文件夹 path2 的内容；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# tar rvf package.tar ~/path2
tar: Removing leading `/' from member names
/root/path2/
/root/path2/file3
tar: Removing leading `/' from hard link targets
```

（7）将包 package.tar 复制到用户目录下的新建文件夹 path3 中；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# mkdir ~/path3
cp package.tar ~/path3
```

（8）进入 path3 文件夹，并还原包 package.tar 的内容。

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# tar xvf package.tar~cd ~/path3
tar xvf package.tar
tar: package.tar~cd: Cannot open: No such file or directory
tar: Error is not recoverable: exiting now
root/path1/
root/path1/file1
root/path1/file2
root/file4
root/path2/
root/path2/file3
```

3. 符号链接内容

（1）新建文件 foo.txt，内容为 123；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# echo "123" > ~/foo.txt
```

（2）建立 foo.txt 的硬链接文件 bar.txt，并比较 bar.txt 的内容和 foo.txt 是否相同，要求用 comm

或 diff 命令；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# ln ~/foo.txt ~/bar.txt
diff ~/foo.txt ~/bar.txt
```

（3）查看 foo.txt 和 bar.txt 的 i 节点号（inode）是否相同；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# ls -i ~/foo.txt ~/bar.txt
1046562 /root/bar.txt  1046562 /root/foo.txt
```

（4）修改 bar.txt 的内容为 abc，然后通过命令判断 foo.txt 与 bar.txt 是否相同；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# echo "abc" > ~/bar.txt
diff ~/foo.txt ~/bar.txt
```

（5）删除 foo.txt 文件，然后查看 bar.txt 文件的 inode 及内容；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# rm ~/foo.txt
ls -i ~/bar.txt
cat ~/bar.txt
1046562 /root/bar.txt
abc
```

（6）创建文件 bar.txt 的符号链接文件 baz.txt，然后查看 bar.txt 和 baz.txt 的 inode 号，并观察两者是否相同，比较 bar.txt 和 baz.txt 的文件

内 容 是 否 相 同 ；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# ln -s ~/bar.txt ~/baz.txt
ls -i ~/bar.txt ~/baz.txt
diff ~/bar.txt ~/baz.txt
1046562 /root/bar.txt  1046563 /root/baz.txt
```

（7）删除 bar.txt ，查看文件 baz.txt ，观察系统给出什么提示信息。

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# rm ~/bar.txt
cat ~/baz.txt
cat: /root/baz.txt: No such file or directory
```

4. 权限管理

（ 1 ） 新 建 文 件 qux.txt ；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# touch ~/qux.txt
```

（2）为文件 qux.txt 增加执行权限（所有 用 户 都 可 以 执 行 ）

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# chmod a+x ~/qux.txt
```

实验三 vim 编辑器及 gcc 编译器的使用

1. vim 编辑器和 gcc 编译器的简单使用:

（1）在用户目录下新建一个目录，命名为 workspace1 ；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# mkdir ~/workspace1
```

（2）进入目录 workspace1 ；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# cd ~/workspace1
```

（4）在 workspace1 下用 vim 编辑器新建一个 c 语言程序文件，文件名为 test.c ，内容为:

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~/workspace1# #include <stdio.h>
int main() {
    printf("Hello, World!");
    return 0;
}
-bash: syntax error near unexpected token `('
-bash: syntax error near unexpected token `"Hello, World!"'
-bash: return: can only `return' from a function or sourced script
-bash: syntax error near unexpected token `}'
```

（4）保存 test.c 的内容，并退出；

按下 Shift + : 进入命令行模式，输入 wq 并按下 Enter 保存并退出

（5）编译 test.c 文件，生成可执行文件 test ，并执行，查看执行结果。

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~/workspace1# ./test~gcc -o test test.c
./test
```

2. vim 编辑器的详细使用:

（1）在用户目录下创建一个名为 workspace2 的目录；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~/workspace1# mkdir ~/workspace2
```

（2）进入 workspace2 目录;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~/workspace1# cd ~/workspace2
```

（3）使用以下命令:

将文件 /etc/gai.conf 的内容复制到当前目录下的新建文件 gai.conf 中;

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~/workspace2# cat /etc/gai.conf > ./gai.conf
```

（4）使用 vim 编辑当前目录下的 gai.conf ;

```
# Configuration for getaddrinfo(3).
#
# So far only configuration for the destination address sorting is needed.
# RFC 3484 governs the sorting.  But the RFC also says that system
# administrators should be able to overwrite the defaults.  This can be
# achieved here.
#
# All lines have an initial identifier specifying the option followed by
# up to two values.  Information specified in this file replaces the
# default information.  Complete absence of data of one kind causes the
# appropriate default information to be used.  The supported commands includ
#
# reload  <yes|no>
#    If set to yes, each getaddrinfo(3) call will check whether this file
#    changed and if necessary reload.  This option should not really be
#    used.  There are possible runtime problems.  The default is no.
#
# label   <mask>   <value>
#    Add another rule to the RFC 3484 label table.  See section 2.1 in
#    RFC 3484.  The default is:
#
#label ::1/128       0
#label ::/0          1
#label 2002::/16     2
#label ::/96         3
#label ::ffff:0:0/96 4
#label fec0::/10     5
#label fc00::/7      6
#label 2001:0::/32   7
#
#    This default differs from the tables given in RFC 3484 by handling
#    (now obsolete) site-local IPv6 addresses and Unique Local Addresses.
#    The reason for this difference is that these addresses are never
#    NATed while IPv4 site-local addresses most probably are.  Given
#    the precedence of IPv6 over IPv4 (see below) on machines having only
#    site-local IPv4 and IPv6 addresses a lookup for a global address would
#    see the IPv6 be preferred.  The result is a long delay because the
#    site-local IPv6 addresses cannot be used while the IPv4 address is
#    (at least for the foreseeable future) NATed.  We also treat Teredo
#    tunnels special.
#
# precedence  <mask>   <value>
#    Add another rule to the RFC 3484 precedence table.  See section 2.1
#    and 10.3 in RFC 3484.  The default is:
#
#precedence  ::1/128       50
#precedence  ::/0          40
#precedence  2002::/16     30
#precedence ::/96          20
#precedence ::ffff:0:0/96  10
#
#    For sites which prefer IPv4 connections change the last line to
#
#precedence ::ffff:0:0/96  100

"gai.conf" 65L, 2584B
```

（5）将光标移到第 18 行；

18G

（6）复制该行内容；**yy**

（7）将光标移到最后一行行首；

G0

（8）粘贴复制行的内容；**p**

```
#include <stdio.h>
int main( )
{
  printf(“hello world!\n”);
  return 0;
}
cat /etc/gai.conf > ./gai.conf
```

（9）撤销第 8 步的动作；**u**

（10）存盘但不退出；**:w**

（11）将光标移到首行；**gg**

（12）插入模式下输入 "Hello, this is vim world!" ； **i**

（13）删除字符串 ″this″ ； **dw**

（14）强制退出 vim ，不存盘 **:q!**

实验四 用户和用户组管理

1. 创建一个名为 foo ， 描述信息为 bar ，登录 shell 为 /bin/sh ，家目录为 /home/foo 的用户，并设置登陆口令为 123456 ；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sudo useradd -c "bar" -s /bin/sh -d /home/foo -p $(openssl passwd -1 123456) foo
```

2. 使用命令从 root 用户切换到用户 foo ， 修改 foo 的 UID 为 2000 ， 其 shell 类型为 /bin/csh ；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sudo su - foo
sudo usermod -u 2000 -s /bin/csh foo
```

3. 从用户 foo 切换到 root ； **exit**

4. 删除 foo 用户，并在删除该用户的同时一并删除其家目录；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sudo userdel -r foo
```

5. 使用命令 newusers 批量创建用户，并使用命令 chpasswd 为这些批量创建的用户设置密码（密码也需要批量设置），查看 /etc/passwd 文件检查用户是否创建成功；

该文件是以冒号分隔的三个字段组成，其分别对应用户名、用户密码和用户 **ID**。其中，用户密码字段需要填上 **x** 占位。

然后，执行以下命令批量创建用户：

```
sudo newusers users.txt
```

使用以下命令为这些用户设置密码：

```
echo "user1:password1" | sudo chpasswd
echo "user2:password2" | sudo chpasswd
```

最后，查看 /etc/passwd 文件，检查用户是否创建成功：

```
cat /etc/passwd
```

6. 创建用户组 group1 ，并在创建时设置其 GID 为 3000 ；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sudo groupadd -g 3000 group1
```

7. 在用户组 group1 中添加两个之前批量创建的用户；

```
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sudo groupadd -g 3000 group1
root@iZ2ze0egyhg6ppmz9q24cwZ:~# sudo usermod -a -G group1 user1
sudo usermod -a -G group1 user2
```

8. 切换到 group1 组中的任一用户，在该用户下使用 sudo 命令查看 /etc/shadow 文件，检查上述操作是否可

以执行；若不能执行，修改 sudoers 文件使得该用户可以查看文件

/etc/shadow 的内容

首先，执行以下命令切换到 group1 组中的任一用户：

```
sudo su - user1
```

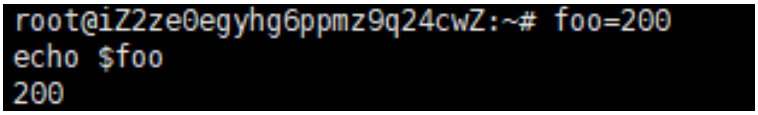
然后，使用以下命令查看 /etc/shadow 文件：

```
sudo cat /etc/s
```

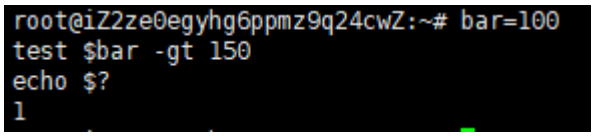
实验五 Shell 程序的创建及条件判断语句

请给出以下的 Linux 指令

1. 定义变量 foo 的值为 200，并将其显示在屏幕上（终端上执行）；



2. 定义变量 bar 的值为 100，并使用 test 命令比较其值是否大于 150，并显示 test 命令的退出码（终端上执行）；



2. 创建一个 Shell 程序，其功能为显示计算机主机名（hostname）和系统时间（date）；

在终端中创建名为 myscript.sh 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
echo "Hostname: $(hostname)"
echo "System time: $(date)"
```

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x myscript.sh
```

然后执行以下命令运行脚本：

```
./myscript.sh
```

输出类似下面的结果：

```
Hostname: mycomputer
System time: Sun May 23 22:48:29 CST 2023
```

3. 创建一个 Shell 程序，要求可以处理一个输入参数，判断该输入参数是否为水仙花数；

```
#!/bin/bash
n=$1
sum=0
temp=$n
```

```
while [ $temp -gt 0 ]
do
    digit=$((temp % 10))
    sum=$((sum + digit * digit * digit))
    temp=$((temp / 10))
done
if [ $n -eq $sum ]
then
    echo "$n is a narcissistic number."
else
    echo "$n is not a narcissistic number."
fi
```

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x narcissistic.sh
```

然后执行以下命令测试脚本：

```
./narcissistic.sh 153
```

输出的结果为 153 is a narcissistic number

5. 创建一个 Shell 程序，输入 3 个参数，计算 3 个输入变量的和并输出；

在终端中创建名为 **sum.sh** 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
sum=$(( $1 + $2 + $3 ))
echo "Sum: $sum"
```

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x sum.sh
```

然后执行以下命令测试脚本：

6. 创建一个 Shell 程序，输入学生成绩，给出该成绩对应的等级： 90 分以上为 A ， 80-90 为 B ， 70-80 为 C ， 60-70 为 D ，小于 60 分为 E 。要求使用实现。

   在终端中创建名为 **grade.sh** 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
score=$1
if [ $score -ge 90 ]; then
    echo "Grade: A"
elif [ $score -ge 80 ]; then
    echo "Grade: B"
elif [ $score -ge 70 ]; then
    echo "Grade: C"
elif [ $score -ge 60 ]; then
    echo "Grade: D"
else
    echo "Grade: E"
fi
```

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x grade.sh
```

然后执行以下命令测试脚本：

```
./grade.sh 85
```

输出的结果为 **Grade: B**，表示成绩为 **85** 分，对应等级为 **B**。

实验六 Shell 循环控制语句

1. 编写一个 Shell 脚本，利用 for 循环把当前目录下的所有 *.c 文件复制到指定的目录中（如 ~/workspace ）；

可以事先在当前目录下建立若干 *.c 文件用于测试。

在终端中创建名为 copy.sh 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
for file in *.c; do
    cp "$file" ~/workspace/
done
echo "Copy complete"
```

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x copy.sh
```

然后执行以下命令测试脚本：

```
./copy.sh
```

2.编写 Shell 脚本，利用 while 循环求前 10 个偶数之和，并输出结果；

创建 sum_even.sh 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
n=1
sum=0
while [ $n -le 10 ]; do
    if [ $(($n % 2)) -eq 0 ]; then
        sum=$(( $sum + $n ))
    fi
    n=$(( $n + 1 ))
done
echo "Sum of first 10 even numbers: $sum"
```

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x sum_even.sh
```

然后执行以下命令测试脚本：

```
./sum_even.sh
```

该脚本将会输出前 10 个偶数的和，即“Sum of first 10 even numbers: 30”。

3.编写 Shell 脚本，利用 until 循环求 1 到 10 的平方和，并输出结果；

创建名为 square_sum.sh 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
n=1
sum=0
until [ $n -gt 10 ]; do
    sum=$(( $sum + $n * $n ))
    n=$(( $n + 1 ))
done
echo "Square sum from 1 to 10: $sum"
```

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x square_sum.sh
```

然后执行以下命令测试脚本：

```
./square_sum.sh
```

该脚本将会输出从 1 到 10 的平方和，即“Square sum from 1 to 10: 385”。

4. 运行下列程序，并观察程序的运行结果。将程序中的 --- 分别替换为 break 、 break 2 、 continue 、 continue 2 ，并观察四种情况下的实验结果。

```
#!/bin/bash
for i in a b c d; do
  echo -n $i
  for j in 1 2 3 4 5 6 7 8 9 10;
```

do

if [[$j -eq 5]]; then

---

fi

echo –n $j

done

echo ''

done

```
#!/bin/bash
for i in a b c d; do
 echo -n $i
 for j in 1 2 3 4 5 6 7 8 9 10; do
 if [[ $j -eq 5 ]]; then
 ---
 fi
 echo -n $j
 done
```

```
    echo ''
  done
```

运行该脚本将输出以下内容：

```
  a12345678910
  b12345678910
  c12345678910
  d12345678910
```

4.1 把 — 替换为 break，结果为：

```
  a1234
```

程序在循环到 $j 的值为 5 时，使用了 break 语句，中断了内部循环，然后继续进行外部循环。

4.2 把 — 替换为 break 2，结果为：

```
  a12
```

程序在循环到 $j 的值为 5 时，使用了 break 2 语句，中断了内部循环和外部循环，直接跳出了整个循环。

4.3 把 — 替换为 continue，结果为：

```
  a1234678910
  b1234678910
  c1234678910
  d1234678910
```

程序在循环到 $j 的值为 5 时，使用了 continue 语句，跳过了本次内部循环中后面的代码，继续进行内部循环的下一次迭代。

4.4 把 — 替换为 continue 2，结果为：

```
a12
b12
c12
d12
```

实验七 Shell 函数

1. 编写 Shell 脚本，实现一个函数，对两个数的和进行求解，并输出结果;

创建名为 sum_func.sh 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
```

```
sum() {
    local result=$(($1 + $2))
    echo $result
}

a=2
b=3
s=$(sum $a $b)
echo "The sum of $a and $b is: $s"
```

函数 sum 接收两个参数，将它们相加，并返回结果。然后可以直接调用该函数求解两个数的和，并将结果输出。

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x sum_func.sh
```

然后执行以下命令测试脚本：

```
./sum_func.sh
```

该脚本将会输出两个数的和，即“The sum of 2 and 3 is: 5”。

2. 编写 Shell 脚本，在脚本中定义一个递归函数，实现 n 的阶乘的求解；

创建名为 factorial_func.sh 的文件，使用编辑器打开该文件并输入以下内容：

```
#!/bin/bash
factorial() {
    if [ $1 -eq 1 ]; then
        echo 1
    else
        local x=$(($1 - 1))
        local y=$(factorial $x)
        echo $(($1 * $y))
    fi
}

n=5
f=$(factorial $n)
echo "Factorial of $n is: $f"
```

函数 factorial 接收一个参数 n，如果 n = 1，则返回结果 1；否则计算 n-1 的阶乘并乘以 n，再返回结果。这样就可以通过递归调用 factorial 函数来实现 n 的阶乘的求解，并将结果输出。

保存并退出编辑器，然后执行以下命令使脚本文件具有可执行权限：

```
chmod +x factorial_func.sh
```

然后执行以下命令测试脚本：

```
./factorial_func.sh
```

该脚本将会输出 n 的阶乘，即“Factorial of 5 is: 120”。

3. 一个 Shell 脚本的内容如下所示： 试运行该程序，并观察程序运行结果，理解函数嵌套的含义

   该 Shell 脚本包含了三个嵌套的函数 first、second 和 third。 first 函数定义了一个 second 函数， second 函数定义了一个 third 函数。其中，third 函数仅执行了一条打印语句。

   运行该程序，执行以下命令：

   ```
   chmod +x nested_function.sh
   ./nested_function.sh
   ```

   程序运行结果如下：

   ```
   starting...
   -1- here is in the first func.
   -2- here is in the second func.
   -3- here is in the third func.
   ```

   首先，打印出了 "starting..."，说明程序已经开始运行。 然后，first 函数被调用，打印出了 "-1-

here is in the first func."。接下来，second 函数被调用，打印出了 "-2- here is in the second func."。然后，third 函数被调用，打印出了 "-3- here is in the third func."。 最后，整个程序执行完毕。

可以看出，嵌套函数层层调用，最终完成了程序的任务。这种嵌套函数的设计方式可以使程序更加清晰、易于维护，也使函数的功能划分更为精细。

实验八　sed 和 awk

1.（1）删除 $ 符号：

```
sed -i 's/\$//g' quote.txt
```

其中，-i 表示直接修改源文件，s/\$//g 表示将每行中的所有 $ 字符删除。

（2）显示包含 music 字字的行内容及行号：

```
sed -n '/music/p' quote.txt
```

其中，-n 表示只显示包含匹配文本的行，/music/p 表示匹配包含 music 字符的行，并将其打印出来。

（3）在第 4 行后面追加内容：“hello world!”：

```
sed -i '4a\hello world!' quote.txt
```

其中，-i 表示直接修改源文件，4a\hello world! 表示在第 4 行后面插入字符串 hello world!。

（4）将文本“The”替换为“Quod”：

```
sed -i 's/The/Quod/g' quote.txt
```

其中，-i 表示直接修改源文件，s/The/Quod/g 表示将每行中的所有 The 字符替换为 Quod。

（5）将第 3 行内容修改为：“This is the third line.”：

```
sed -i '3s/.*/This is the third line./'
quote.txt
```

其中，-i 表示直接修改源文件，3s/.*/This is the third line./ 表示将第 3 行的文本全部替换为 This is the third line.。

（6）删除第 2 行内容：

```
sed -i '2d' quote.txt
```

其中，-i 表示直接修改源文件，2d 表示删除第 2 行。

（7）设置 Shell 变量 var 的值为 evening ，用 sed 命令查找匹配 var 变量值的行：

```
var="evening"
sed -n "/$var/p" quote.txt
```

2. 文件 numbers.txt 的内容如下所示：

one : two : three

four : five : six

注：每个冒号前后都有空格。

试使用 awk 命令实现如下功能：分别以 空格 和 冒号 做分隔符，显示第 2 列的内容，观察两者的区别;

使用 **awk** 命令，以空格为分隔符，显示第 **2** 列的内容：

```
awk '{print $2}' numbers.txt
```

输出为：

```
:
five
```

可以看到，由于冒号也被视为一个分隔符，因此第二列显示了冒号。

现在，使用 **awk** 命令，以冒号为分隔符，显示第 **2** 列的内容：

```
awk -F ':' '{print $2}' numbers.txt
```

输出为：

```
 two
 five
```

可以看到，此次输出只包括第二列，并且没有冒号。这是由于“**-F**”选项指定了分隔符为冒号。

3. 已知文件 foo.txt 中存储的都是数字，且每行都包含 3 个数字，数字之前以空格作为分隔符。试找出 foo.txt 中的所有偶数进行打印，并输出偶数的个数。

可以使用 **awk** 命令读取 **foo.txt** 文件，并匹配其中的偶数。

具体操作如下：

```
awk '{for(i=1;i<=NF;i++) if($i%2==0) {count++;
printf("%d ", $i)} } END {print "Total even
numbers: ", count}' foo.txt
```

解释说明：

- **awk** 遍历每一行，内部 **for** 循环遍历行中的所有数字。
- 判断每个数字是否为偶数，如果是则输出，并累加计数器 **count**。
- 循环结束后输出偶数的总个数。

注：**NF** 是 **awk** 内置的变量，表示当前行中的字段数。

如果 **foo.txt** 文件中包含以下内容：

```
1 2 3
```

4 5 6

7 8 9

那么运行上述命令后会输出：

2 4 6 8 Total even numbers: 4

输出的信息包括了每个偶数和总的偶数个数（这里的第二个信息为 4，表示该文件中一共有 4 个偶数）。

4. 脚本的功能是在文件 info.txt 中搜索指定的模式，并计算出匹配结果的数量。脚本的详细解释如下：

   脚本的第一行（#!/bin/bash）指定了运行脚本时要使用的 shell 程序。

   read -p "enter search pattern: " pattern 这行代码会提示用户输入要搜索的模式，并将其保存在前面定义的 pattern 变量中。

   awk "/$pattern/"'{ nmatches++; print } END { print nmatches, "found." }' info.txt 这一行将使用 awk 命令在 info.txt 文件中搜索指定的模式，并计算匹配结果的数量。具体操作如下：

- **$pattern** 是一个正则表达式，用来匹配文件中的文本。
- **/{pattern}/** 表示匹配文件中包含指定模式的所有行。
- **nmatches++** 会在每一行匹配成功时自增 **nmatches** 变量的值。
- **print** 会输出匹配成功的行。
- **END { print nmatches, “found.” }** 会在文件遍历结束时输出匹配结果的数量。

最终脚本会将匹配成功的行和匹配结果的数量输出到控制台。