Linux 云计算集群架构师

学神 IT 教育：从零基础到实战，从入门到精通！

联系方式：

学神 IT 教育官方网站: http://www.xuegod.cn

Linux 云计算架构师进阶学习群 QQ 群: 1072932914

学习顾问：小语老师　　学习顾问：边边老师　　学神微信公众号

微信扫码添加学习顾问微信，同时扫码关注学神公众号了解最新行业动态，获取更多学习资料及答疑就业服务！

第十一章　重定向和文件的查找

本节所讲内容：

11.1　文件描述符 0、1、2
11.2　重定向的含义-管道的使用-tee 命令
11.3　which-whereis-locate-grep-find 查找命令
11.4　命令判断

LINUX 下一切皆文件
文件又可分为：普通文件、目录文件、链接文件、设备文件
LINUX 系统使用文件来描述各种硬件设备资源，如：/dev/sda　/dev/sdb　/dev/sr0

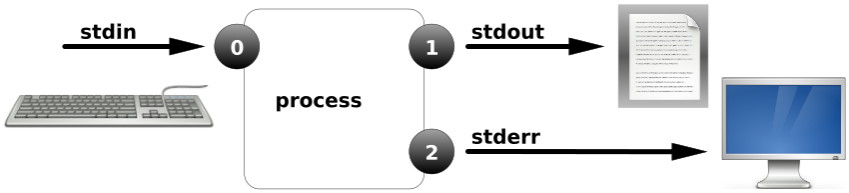
11.1　文件描述符

用户通过操作系统处理信息的过程中,使用的交互设备文件（键盘，鼠标，显示器）

Number	Channel name	Description	Default connection	Usage
0	stdin	Standard input	Keyboard	read only
1	stdout	Standard output	Terminal	write only
2	stderr	Standard error	Terminal	write only
3+	*filename*	Other files	*none*	read and/or write

11.1.1　输入输出标准说明

STDIN 标准输入　默认的设备是键盘　文件编号为：0
STDOUT 标准输出　默认的设备是显示器　文件编号为：1　，也可以重定向到文件
STDERR 标准错误　默认的设备是显示器　文件编号为：2　，也可以重定向到文件



查看一个进程打开了哪些文件?
语法：　ll /proc/进程 ID/fd
例 1：
[root@xuegod63 ~]# vim /etc/passwd
新开客户端查看
[root@xuegod63 ~]# ps -axu | grep passwd
root　　4602　2.1　0.2 151600　5300 pts/2　　S+　　15:30　　0:00 vim /etc/passwd
[root@xuegod63 ~]# ll /proc/4602/fd　　#查看打开的文件
总用量 0
lrwx------ 1 root root 64 5 月　14 15:30 0 -> /dev/pts/2
lrwx------ 1 root root 64 5 月　14 15:30 1 -> /dev/pts/2
lrwx------ 1 root root 64 5 月　14 15:30 2 -> /dev/pts/2
lrwx------. 1 root root 64 4 月　28 14:00 4 -> /etc/.passwd.swp

注： 这些 0,1,2,4 就是文件的描述符。一个进程启动时,都会打开 3 个文件:标准输入、标准输出和标准出错处理。这 3 个文件分别对应文件描述符为 0、1 和 2 也就是替换 STDIN_FILENO（标准输入）、STDOUT_FILENO（标准输出）和 STDERR_FILENO（标准错误）。

STDIN_FILENO 属于系统 API 接口库，其声明为 int 型，是一个打开文件句柄，对应的函数主要包括 open/read/write/close 等系统级调用。

操作系统一级提供的文件 API 都是以文件描述符来表示文件。STDIN_FILENO 就是标准输入设备（一般是键盘）的文件描述符。

/proc/进程 ID/fd #这个 fd 目录下，专门存文件描述符

注：对文件描述符的操作就是对文件本身的操作。 我可以直接通过操作文件描述来修改文件。

例 2：查看和临时设置一个进程最多可以打开几个文件，即：一个进程可以打开的文件描述符限制

```
[root@xuegod63 ~]# ulimit -n          #查看一个进程最多可以同时打开的文件数
1024
[root@xuegod63 ~]# ulimit -n 2048   #修改一个进程最多可以同时打开的文件数为 2048
[root@xuegod63 ~]# ulimit -n
2048
```

永久修改，会在第三阶段讲系统调优时讲。

11.2 重定向的含义-管道的使用-tee 命令

11.2.1 输出重定向

定义：将命令的正常输出结果保存到指定的文件中,而不是直接显示在显示器的屏幕上

重定向输出使用”>” “>>” 操作符号

语法： > 文件名 #表示将标准输出的内容，写到后面的文件中，如果此文件名已经存在，将会覆盖原文件中的内容，如果不存在，则创建文件并写入内容

> > 文件名 #表示将标准输出的内容，追加到后面的文件中。若重定向的输出的文件不存在，则会新建该文件

例 1：查看当前主机的 CPU 的类型保存到 cpu.txt 文件中(而不是直接显示到屏幕上)

```
[root@xuegod63 ~]# cat /proc/cpuinfo > cpu.txt
```

例 2：将内核的版本信息追加到 cpu.txt

```
[root@xuegod63 ~]# uname -a >> cpu.txt
```

例 3：清空一个文件

```
[root@xuegod63 ~]# > cpu.txt
```

11.2.2 输入重定向

例 1：将命令中接收输入的途径由默认的键盘改为其他文件.而不是等待从键盘输入

```
[root@xuegod63 mnt]# grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@xuegod63 mnt]# grep root < /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

例 2：mysql 中数据导入

例： [root@xuegod63 ~]# mysql -uroot -p123456 < xuegod.sql #将 xuegod.sql 导入

mysql 数据库中。 这个命令现在不能执行，大家先知道有这种写法就可以了。后期在第二阶段讲 mysql 时，会讲。

11.2.3 EOF

EOF 本意是 End Of File，表明到了文件末尾。”EOF“通常与”<<“结合使用，“<<EOF“表示后续的输入作为子命令或子 shell 的输入，直到遇到”EOF“，再次返回到主调 shell，可将其理解为分界符（delimiter）。既然是分界符，那么形式自然不是固定的，这里可以将”EOF“可以进行自定义，但是前后的”EOF“必须成对出现且不能和 shell 命令冲突。

例 1：以<<EOF 开始，以 EOF 结尾 。

```
[root@bogon ~]# cat > a.txt <<EOF
> dfsd
> sdfs
> sdf
> dfs
> EOF
```

注：只输入红色文字部分，>为 EOF 的交互输入提示符。

```
[root@bogon ~]# cat a.txt
dfsd
sdfs
sdf
dfs
```

例 2：以 ccc 作为分界符

```
[root@localhost ~]# cat > a.txt <<ccc
> eof
> EOF
> ccc
[root@localhost ~]# cat a.txt
eof
EOF
```

例 3：在脚本中我们可以通过重定向输入来打印消息菜单

在使用的时候需要在”<< “右边跟一对终止符。终止符是可以自定义

```
[root@xuegod63 mnt]# vim    eof.sh    #写入以下内容
#!/bin/bash

cat <<efo
========================
1.mysql
2.httpd
3.oracle
========================
efo
[root@xuegod63 ~]# chmod +x eof.sh
[root@xuegod63 ~]# ./eof.sh   #查看效果
```

11.2.4 错误重定向

将命令执行过程中出现的错误信息（选项或参数错误）保存到指定的文件,而不是直接显示到显示器

作用：错误信息保存到文件

操作符：错误重定向符号：2>

2 指的是标准错误输出的文件描述符

在实际应用中，错误重定向可以用来收集执行的错误信息.为排错提供依据；对于 shell 脚本还可以将无关紧要的错误信息重定向到空文件/dev/null 中，以保持脚本输出的简洁

例 1：将错误显示的内容和正确显示的内容分开

```
[root@xuegod63 mnt]# ls /etc/passwd xxx
ls: 无法访问 xxx: 没有那个文件或目录
/etc/passwd
[root@xuegod63 mnt]# ls /etc/passwd xxx > a.txt
ls: 无法访问 xxx: 没有那个文件或目录
[root@xuegod63 mnt]# cat a.txt
/etc/passwd
[root@xuegod63 mnt]# ls /etc/passwd xxx 2> a.txt
/etc/passwd
[root@xuegod63 mnt]# cat a.txt
ls: 无法访问 xxx: 没有那个文件或目录
```

注：使用 2> 操作符时,会像使用 > 一样覆盖目标文件的内容，若追加而不覆盖文件的内容即可使用 2>> 操作符

11.2.5 null 黑洞和 zero 空文件

1、把/dev/null 看作"黑洞"，所有写入它的内容都会永远丢失. 而尝试从它那儿读取内容则什么也读不到. 然而 /dev/null 对命令行和脚本都非常的有用.

```
[root@xuegod63 ~]# echo aaaa > /dev/null
[root@xuegod63 ~]# cat /dev/null      #什么信息也看不到
```

2、/dev/zero 在类 UNIX 操作系统中, /dev/zero 是一个特殊的文件，当你读它的时候，它会提供无限的空字符(NULL, ASCII NUL, 0x00)。典型用法是用它来产生一个特定大小的空白文件。

例：使用 dd 命令产生一个 50M 的文件

参数：

if 代表输入文件。如果不指定 if，默认就会从 stdin 中读取输入。

of 代表输出文件。如果不指定 of，默认就会将 stdout 作为默认输出。

bs 代表字节为单位的块大小。

count 代表被复制的块数。

```
[root@xuegod63 mnt]# dd if=/dev/zero of=b.txt bs=1M count=50
50+0 records in
50+0 records out
52428800 bytes (52 MB) copied, 0.228653 s, 229 MB/s
[root@xuegod63 mnt]# du -sh b.txt
50M  b.txt
[root@xuegod63 mnt]# cat b.txt   #什么也不显示
```

例 2：正确的内容写入一个文件，错误的写入一个文件

[root@xuegod63 mnt]# ls /tmp xxxx >ok.txt 2> err.txt

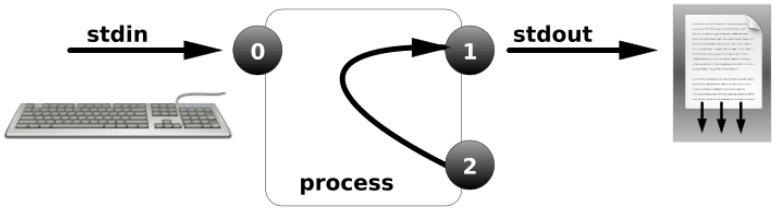
11.2.6 >&符号

&表示等同于的意思

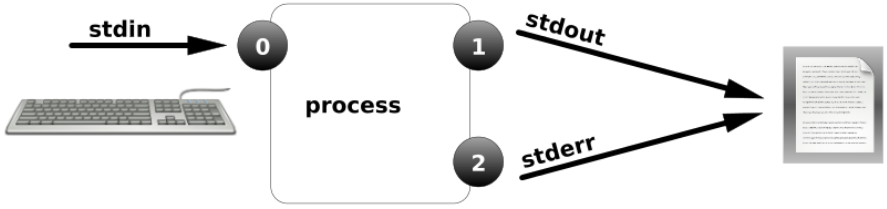
例 1：把正确和错误的消息输入到相同的位置

1>&2 把标准输出重定向到标准错误

2>&1 把标准错误重定向到标准输出，如图:



例 2：把正确和错误的消息输入到相同的位置



[root@xuegod63 mnt]# ls /tmp xxxx >1.txt 2>&1

或:

[root@xuegod63 mnt]# ls /tmp xxxx 2>2.txt 1>&2

问题来了，为什么一定要把重定向写在后面，而不是写在前面呢？比如下面这样

[root@xuegod63 mnt]# ls /tmp xxx 2>&1 >1.txt

思考：ls /tmp xxxx >1.txt 2>&1

本来 1 和 2 指向屏幕

执行>1.txt 后，1 指向 1.txt 文件

执行 2>&1 后，2 指向 1，而 1 指向 1.txt，因此 2 也指向 1.txt

思考：ls /tmp xxxx 2>&1 >1.txt

本来 1 和 2 指向屏幕

执行 2>&1 后，2 指向了 1，因此 2 仍指向屏幕

执行>1.txt 后，1 指向了 1.txt 文件，2 仍然指向屏幕

例 3：互动： 工作中 shell 脚本中的 >/dev/null 2>&1 是什么意思?

[root@xuegod63 ~]# cat /etc/passwd xxx >/dev/null 2>&1

每次都写> /dev/null 2>&1 太麻烦，能简写吗?

有两种写法

&> /dev/null

>& /dev/null

以上两种写法都和 > /dev/null 2>&1 一个语义

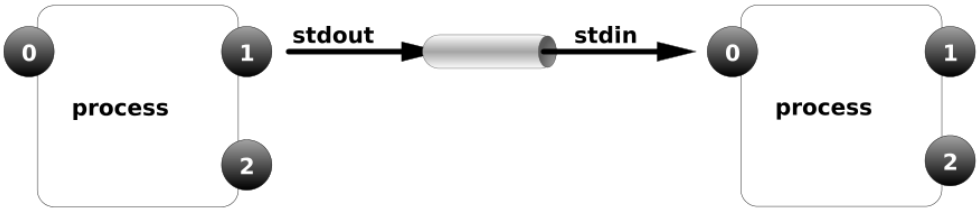
那么 &> 和 >&有区别吗？ 语义上是没有任何区别的，但是第一种方式是最佳选择，一般使用第一种。

[root@xuegod63 ~]# cat /etc/passwd xxx &> /dev/null

注：将标准输出和错误输出全部重定向到/dev/null 中,也就是将产生的所有信息丢弃.

11.2.7 管道 | 的使用

语法：command-a | command-b | command-c | .....



上图可以简单理解为：管道符左边命令的结果作为管道符右边命令的标准输入

注意：

1、管道命令只处理前一个命令正确输出，不处理错误输出

2、管道右边的命令，必须是能够接收标准输入数据的命令才行

3、管道符可以把两条命令连起来，它可以链接多个命令使用

[root@xuegod63 ~]# ps -axu | grep sshd

root 1089 0.0 0.2 105996 4088 ? Ss 20:19 0:00 /usr/sbin/sshd -D

root 43262 0.0 0.0 112680 984 pts/1 S+ 21:36 0:00 grep --color=auto sshd

11.2.8 tee 命令

功能：读取标准输入的数据，并将其内容输出成文件。

语法：tee [-a][--help][--version][文件...]

参数：

-a, --append 内容追加到给定的文件而非覆盖

--help 在线帮助

tee 指令会从标准输入设备读取数据，将其内容输出到标准输出设备，同时保存成文件

例 1：将磁盘使用的信息写入文件

[root@xuegod63 ~]# df -h | tee disk.log

例 2：将文件系统使用的信息追加到文件

[root@xuegod63 ~]# df -h | tee -a disk.log

注： 可以使用来记录日志

[root@xuegod63 ~]# df -hT | cat >> disk.log

11.3 文件查找常用命令

11.3.2 which-whereis-locate-grep find 命令使用

查找文件一般有以下几个命令：

which 查看可执行文件的位置

whereis 查看可执行文件的位置及相关文件

locate 配合数据库缓存，快速查看文件位置

grep 过滤匹配，它是一个文件搜索工具

find 查找相关文件

举例：

```
[root@xuegod63 ~]# which cd
/usr/bin/cd
[root@xuegod63 ~]# whereis cd
cd: /usr/bin/cd /usr/share/man/man1/cd.1.gz /usr/share/man/man1p/cd.1p.gz
[root@xuegod63 ~]# whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.g
```

locate

locate 命令和 find -name 功能差不多，是它的另外一种写法，但是这个要比 find 搜索快的多，因为 find 命令查找的是具体目录文件，而 locate 它搜索的是一个数据库 /var/lib/mlocate/mlocate.db，这个数据库中存有本地所有的文件信息；这个数据库是 Linux 自动创建并每天自动更新维护。相关的配置信息在/etc/updatedb.conf，查看定时任务信息在 /etc/cron.daily/mlocate

```
[root@xuegod63 ~]# yum -y install mlocate
[root@xuegod63 mnt]# touch /opt/xuegod.txt
[root@xuegod63 mnt]# locate xuegod.txt    #发现找不到
[root@xuegod63 mnt]# updatedb     #如果对当天文件查找，需要手动更新数据库 updatedb
[root@xuegod63 mnt]# locate xuegod
```

grep 查找使用

作用：过滤,它能够使用正则表达式来搜索文本，并把结果打印出来

参数：

-v 取反

-i 忽略大小写

^# 以#开头

#$ 以#结尾

^$ 空行

-n 对过滤的内容加上行号

| 或者的意思

```
[root@xuegod63 ~]#   ps -aux | grep sshd | grep -v grep
root        1089  0.0  0.2 105996  4088 ?        Ss   20:19   0:00 /usr/sbin/sshd –D

[root@xuegod63 ~]# cat /etc/passwd | grep ^a #以 a 开头

[root@xuegod63 ~]# grep bash$ /etc/passwd   #以 bash 结尾

[root@xuegod63 ~]# grep "games\|root" /etc/passwd | wc -l
```

36

注： \ 表示转义符

[root@xuegod63 ~]# grep -E "nologin|root" /etc/passwd | wc -l

[root@xuegod63 ~]# egrep "nologin|root" /etc/passwd | wc -l

#查看包括 nologin 或 root 的行

36

注：egrep 是 grep 加强版本

[root@xuegod63 ~]# grep ^$ -v /etc/rsyslog.conf -n

[root@xuegod63 ~]# grep ^$ -v /etc/rsyslog.conf | grep -v ^#

[root@xuegod63 ~]# egrep "^$|^#" -v /etc/rsyslog.conf

11.3.3 find 命令使用

格式：find pathname -options [-print]

命令字 路径名称 选项 输出

参数：

pathname: find 命令所查找的目录路径，不输入代表当前目录例如用 . 来表示当前目录，用 / 来表示系统根目录。

find 命令选项:

-name 按照文件名查找文件。 “名称”

-perm 按照文件权限来查找文件。666 777 等

-user 按照文件属主来查找文件

-group 按照文件所属的组来查找文件

-mtime -n / +n 按照文件的更改时间来查找文件，

- n 表示文件更改时间距现在 n 天以内

+ n 表示文件更改时间距现在 n 天以前

-type 查找某一类型的文件

b - 块设备文件

d - 目录

c - 字符设备文件

p - 管道文件

l- 符号链接文件

f - 普通文件

-size n 查找符合指定的文件大小的文件

-exec 对匹配的文件执行该参数所给出的其他 linux 命令， 相应命令的形式为'命令 {} \;，注意{ }和 \; 之间的空格，{}代表查到的内容

例 1：查看当前目录下所有的 TXT 格式的文件

[root@xuegod63 mnt]# find . -name "*.txt"

./a.txt

./xuegod.txt

2、按照更改时间或访问时间等查找文件

如果希望按照更改时间来查找文件，可以使用 mtime,atime 或 ctime 选项

mtime: 文件最后一次修改的时间

atime: 最后一次访问时间

ctime: 文件的最后一次变化时间，也就是修改时间

例 1：希望在 root 目录下查找更改时间在 1 天以内，被黑客修改的文件

[root@xuegod63 ~]# find /root/ -mtime -1

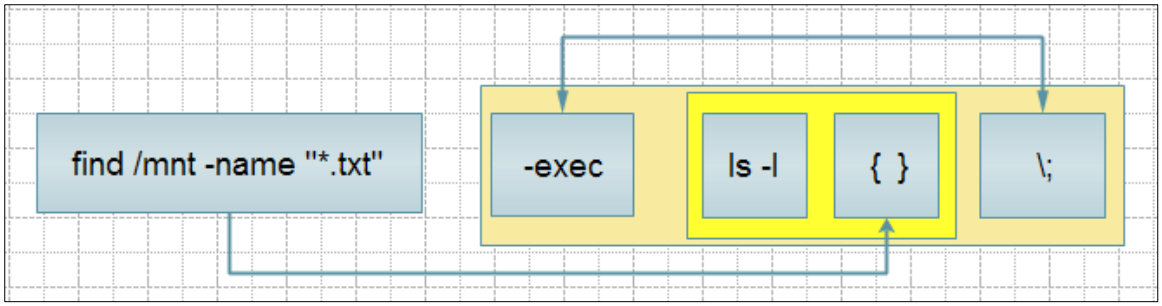
-mtime -1，指的是当前时间为 2021-04-10 22:30，2021-04-09 22:30~ 2021-04-10 22:30 之间修改的文件

-mtime 1，指的是当前时间为 2021-04-10 22:30，2021-04-08 22:30~ 2021-04-09 22:30 之间修改的文件

-mtime +1，指的是当前时间为 2021-04-10 22:30，2021-04-08 22:30 之前修改的文件

对查找内容执行相应命令

-exec 这个选项参数后面可以跟自定义的 SHELL 命令，格式如下：



例 2：

[root@xuegod63 ~]# touch {1,2,3}.back

[root@xuegod63 mnt]# find . -name "*.back" -exec ls -l {} \;

例 3：

[root@xuegod63 ~]# find . -name "*.back" -exec mv {} /opt \;

[root@xuegod63 ~]# ls /opt/

1.back 2.back 3.back rh xuegod.txt

例 4：把查找到的文件复制到一个指定的目录

[root@xuegod63 mnt]# find /root -name "*.txt" -exec cp {} /opt \;

例 5：xargs 和 find 命令结合 复制文件 -i 表示 find 传递给 xargs 的结果 由{}来代替 （了解）xargs 又称管道命令。简单的说 就是把 其他命令给它的数据 传递给它后面的命令作为参数

[root@xuegod63 ~]# rm -rf /opt/*

[root@xuegod63 ~]# find . -name "*.txt" | xargs -i cp {} /opt

[root@xuegod63 ~]# ls /opt/

例 6：查找多个类型文件

比较符的使用：

-a and 并且

-o or 或者

+ 超过
- 低于

```
[root@xuegod63 ~]# touch a.pdf back.sh
[root@xuegod63 ~]# find . -name "*.sh" -o -name "*.pdf"
[root@xuegod63 ~]# find /etc -size +20k -a -size -50k | wc -l
22
[root@xuegod63 ~]# find /etc -size +20k  | wc -l
49
```

例 7： 按权限查找：-perm

```
[root@xuegod63 ~]# find /bin/ -perm 755    # 等于 0755 权限的文件或目录
[root@xuegod63 ~]# find /bin/ -perm  -644 #  -表示至少，至少有 644 权限的文件或目录
```

例：查看系统中权限至少为 777 的文件或目录

创建一些测试文件：

```
[root@xuegod63 ~]# mkdir ccc
[root@xuegod63 ~]# chmod 777 ccc
[root@xuegod63 ~]# mkdir test
[root@xuegod63 ~]# chmod 1777 test
 [root@xuegod63 ~]# touch b.sh
[root@xuegod63 ~]# chmod 4777 b.sh
```

查找：

```
[root@xuegod63 ~]# find /root/ -perm 777
[root@xuegod63 ~]# find /root/ -perm 1777
[root@xuegod63 ~]# find /root/ -perm  4777
```

例：把系统中权限不低于 777 的危险目录查找出来

```
[root@xuegod63 ~]# find /root/ -perm  -777   #至少有 777 权限
```

例：把系统中权限不低于 777 的危险文件查找出来

```
[root@xuegod63 ~]# find /root -type f -perm  -777
```

例 8：查找的目录深度：

-maxdepth 1 #只查找目录第一层的文件和目录

如：查找/bin 目录下权限等于 644 的文件

```
[root@xuegod63 ~]# find /etc/ -maxdepth 1 -perm 644 | wc -l
[root@xuegod63 ~]# find /etc/ -maxdepth 2 -perm 644 | wc -l
[root@xuegod63 ~]# find /etc/ -maxdepth 4 -perm 644 | wc -l
[root@xuegod63 ~]# find /etc/ -maxdepth 1 -perm 644 -exec ls -lh {} \;
[root@xuegod63 ~]# find /etc/ -maxdepth 4 -perm 644 -exec ls -lh {} \;
```

例 9：查找系统中所有属于用户 user1 的文件，并把这个文件，放到/root/findresults 目录下

注意：/root/findresults 这个需要提前创建好。

```
[root@xuegod63 ~]# mkdir /root/findresults
```

[root@xuegod63 ~]# useradd user1

[root@xuegod63 ~]# find / -user user1 -exec cp -a {} /root/findresults/ \;
#参数： -a #复制时，保留原来文件的所有属性
报错：
find: ‘/proc/43475/task/43475/fd/6’：没有那个文件或目录
find: ‘/proc/43475/task/43475/fdinfo/6’：没有那个文件或目录
find: ‘/proc/43475/fd/6’：没有那个文件或目录
find: ‘/proc/43475/fdinfo/6’：没有那个文件或目录
cp: 无法以目录"/home/user1" 来覆盖非目录"/root/findresults/user1"
互动： 同一个目录下，可以创建文件 user1 和文件夹 user1 吗？同一个目录下创建的文件名和目录名一样吗?
答：不可以
[root@xuegod63 ~]# touch abc
[root@xuegod63 ~]# mkdir abc
mkdir: 无法创建目录"abc": 文件已存在
解决：
[root@xuegod63 ~]# find / -user user1 #发现
[root@xuegod63 ~]# ll /var/spool/mail/user1 #查看这个文件
[root@xuegod63 ~]# ll /home/
发现/var/spool/mail/user1 和/home/user1 的名字是一样的。 而两者都要复制到/root/findresults/下，先复制了/var/spool/mail/user1，所以/home/user1 就不能复制了。
[root@xuegod63 ~]# mv /var/spool/mail/user1 /var/spool/mail/user1.mail
[root@xuegod63 ~]# rm -rf /root/findresults/* /root/findresults/.*
[root@xuegod63 ~]# find / -user user1 -exec cp -a {} /root/findresults/ \;
[root@xuegod63 ~]# mv /var/spool/mail/user1.mail /var/spool/mail/user1
#再修改过来

11.4 命令判断

11.4.1 常用的三个特殊符号

1、 ；分号 不考虑指令的相关性，连续执行，分号； 不保证命令全部执行成功的
例：[root@xuegod63 mnt]# 123 ； echo aaa
&& 逻辑与====》它是只有在前面的命令执行成功后，后面的命令才会去执行
例 1：如果/opt 目录存在，则在/opt 下面新建一个文件 a.txt
[root@xuegod63 ~]# ls /etc/passwd && touch ccc.txt && touch ls
[root@xuegod63 ~]# ls xxx && touch abc.txt
例 2：源码编译经典使用方法
[root@xuegod63 ~]# ./configure && make -j 4 && make install #我现在没有源码包，所以此命令不能执行成功。大家了解一下这个经典用法。

2、 || 逻辑或===》如果前面的命令执行成功，后面的命令就不去执行了；如果前面的执行不成功，才会去执行后面的命令 ()
例 1：

[root@xuegod63 etc]# ls xxx || cd /mnt
ls: 无法访问 xxx: 没有那个文件或目录
[root@xuegod63 mnt]# pwd
/mnt
[root@xuegod63 mnt]# ls /etc/passwd || cd /etc
/etc/passwd
总结：
命令情况　说 明
命令 1 && 命令 2　如果命令 1 执行，且执行正确($? = 0)，然后执行命令 2
如果命令 1 执行完成，但是执行错误（$? ≠0），那么后面的命令是不会执行的
命令 1 || 命令 2 如果命令 1 执行，且执行正确($? = 0)，那么命令 2 不执行
如果命令 1 执行，但执行错误($? ≠ 0)，那么命令 2 执行

运算顺序：LINUX 执行命令，是从左到右一个一个执行,从上到下执行
例：[root@xuegod63 ~]# cd /opt/back || mkdir /opt/back && touch /opt/back/back.tar && ls /opt/back

&& 逻辑与====》它是只有在前面的命令执行成功后，后面的命令才会去执行
[root@xuegod63 ~]# ls /etc/passwd || touch ccc.txt && touch nbc.txt
ls /etc/passwd 执行成功了，touch ccc.txt 就不会执行，此时$? 值为 0，从左到右一个一个执行，接下来执行&&，左面$?值为 0，&&继续执行右面， 创建 nbc.txt

总结：

11.1　文件描述符 0、1、2
11.2　重定向的含义-管道的使用-tee 命令
11.3　which-whereis-locate-grep-find 查找命令
11.4　命令判断