文件描述符定义

文件描述符：是内核为了高效管理已被打开的文件所创建的索引，用于指向被打开的文件，所有执行I/O 操作的系统调用都通过文件描述符；文件描述符是一个简单的非负整数，用以标明每一个被进程所打开的文件，程序刚刚启动的时候，第一个打开的文件是 0 ，第二个是 1 ，依此类推。也可以理解为是一个文件的身份ID

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

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

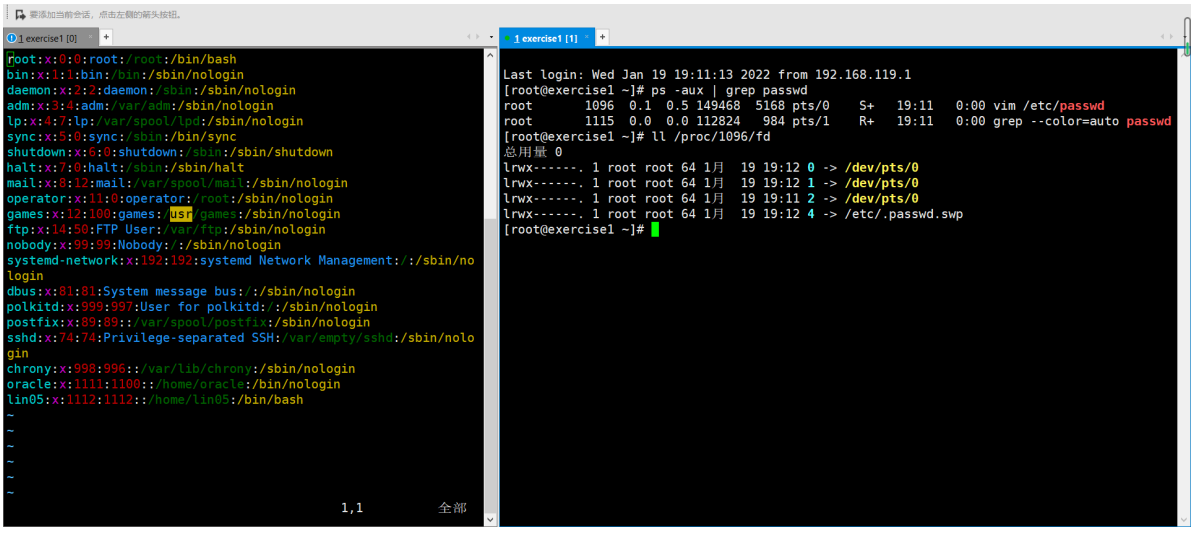
STDOUT 标准输出（默认正确）　　默认的设备是显示器　　文件编号为：1，也可以重定向到文件

STDERR 标准错误输出　　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

```
[root@exercise1 ~]# vim /etc/passwd
#然后在xshell复制ssh渠道，即复制这个终端，查看进程
[root@exercise1 ~]# ps -aux | grep passwd
root       1096  0.1  0.5 149468  5168 pts/0    S+   19:11   0:00 vim /etc/passwd
root       1115  0.0  0.0 112824   984 pts/1    R+   19:11   0:00 grep --color=auto passwd
[root@exercise1 ~]# ll /proc/1096/fd   #查看打开的文件，这个 fd 目录下，专门存文件描述符
总用量 0
lrwx------. 1 root root 64 1月  19 19:12 0 -> /dev/pts/0
lrwx------. 1 root root 64 1月  19 19:12 1 -> /dev/pts/0
lrwx------. 1 root root 64 1月  19 19:11 2 -> /dev/pts/0
lrwx------. 1 root root 64 1月  19 19:12 4 -> /etc/.passwd.swp
[root@exercise1 ~]#
```



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、标准输出和标准错误输出。

这3个文件分别对应文件描述符为 0、 1和2也就是宏替换STDIN_FILENO、STDOUT_FILENO 和 STDERR_FILENO。

/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。

4.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夹中，而不是直接显示在显示屏的屏幕上

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

>> 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
```

例子：

```
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@exercise1 ~]# cat /proc/cpuinfo > /opt/cpu.txt
[root@exercise1 ~]# ll /opt/
总用量 4
-rwxrwxrwx. 1 root root   0 1月  18 20:01 1.txt
-rw-r--r--. 1 root root 929 1月  19 19:32 cpu.txt
……
```

例 2： 将内核的版本信息追加到cpu.txt
[root@exercise1 ~]# uname -a >> /opt/cpu.txt

例 3： 清空一个文件
[root@exercise1 ~]# > /opt/cpu.txt

5.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@exercise1 ~]# grep root /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
[root@exercise1 ~]# grep root < /etc/passwd
root:x:0:0:root:/root:/bin/bash
operator:x:11:0:operator:/root:/sbin/nologin
```

例2：统计 users 文件的行数

```
[root@exercise1 ~]# vim /opt/test.txt
#插入以下内容
This is a good day
Today is Wednesday
[root@exercise1 ~]# wc -l /opt/test.txt   #会输出文件名
2 /opt/test.txt
[root@exercise1 ~]# wc -l < /opt/test.txt   #仅仅知道从标准输入读取内容，不会输出文件名
2
[root@exercise1 ~]#
```

例3：mysql中数据导入

[root@base ~]# mysql -uroot -p123456 < a.sql #将 a.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。

例4：同时替换输入和输出，执行command1，从文件infile读取内容，然后将输出写入到outfile中

command1 < infile > outfile

6.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。

例子

```
例 1：以<<EOF 开始，以 EOF 结尾
[root@exercise1 ~]# cat > /opt/test.txt << EOF
hhh
ooo
EOF
[root@exercise1 ~]# cat /opt/test.txt
hhh
ooo
```

例 2： 以 ccc 作为分界符
[root@exercise1 ~]# cat > /opt/test.txt << ccc
eof
EOF
ccc

```
[root@exercise1 ~]# cat /opt/test.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 #多数用于脚本
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@exercise1 ~]# cat << efo
> ========

> 1.mysql
> 2.httpd
> 3.oracle
> ========
> efo
========
1.mysql
2.httpd
3.oracle
========

7.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>　　；　标准输入：　1< 或简写　<　；　标准输出：　0> 或　>2 指的是标准错误输出的文件描述符 (在使用标准的输入和输出省略了 1、0 编号)

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

例子：

```
例 1: 将错误显示的内容和正确显示的内容分开
[root@exercise1 ~]# ls /etc/passwd xxx
ls: 无法访问xxx: 没有那个文件或目录
/etc/passwd
[root@exercise1 ~]# ls /etc/passwd xxx > /opt/error.txt
ls: 无法访问xxx: 没有那个文件或目录
[root@exercise1 ~]# cat /opt/error.txt
/etc/passwd
[root@exercise1 ~]# ls /etc/passwd xxx 2>
/opt/error.txt   #把错误信息覆盖到/opt/error.txt
/etc/passwd
[root@exercise1 ~]# cat /opt/error.txt
ls: 无法访问xxx: 没有那个文件或目录
[root@exercise1 ~]#
```

```
例 2：正确的内容写入一个文件 ，错误的写入一个文件
[root@exercise1 ~]# ls /tmp/ xxx > /opt/ok.txt 2>
/opt/error.txt
```

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

8.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@exercise1 ~]# echo aaaa >> /dev/null
[root@exercise1 ~]# 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。

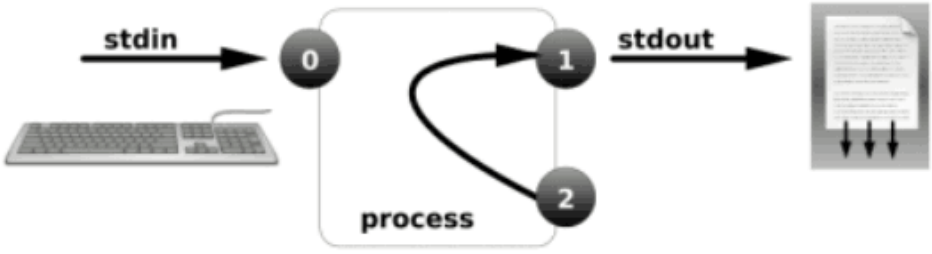
count 代表被复制的块数。

```
[root@exercise1 opt]# dd if=/dev/zero of=b.txt bs=1M count=50
记录了50+0 的读入
记录了50+0 的写出
52428800字节(52 MB)已复制，0.642576 秒，81.6 MB/秒
[root@exercise1 opt]# du -sh b.txt
50M b.txt
[root@exercise1 opt]# cat b.txt    #什么也不显示
```

9.&>和>&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:
```



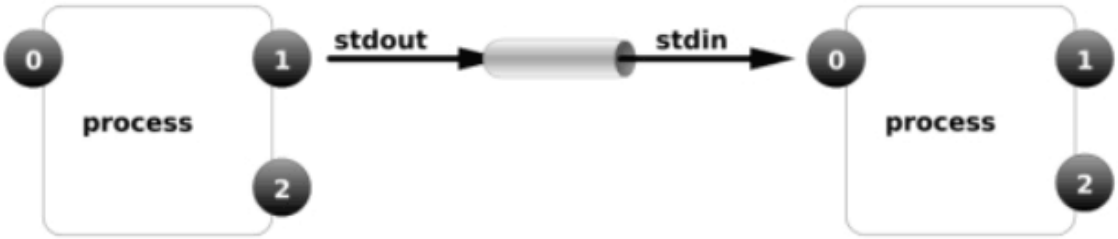
```
例 2 ：把正确和错误的消息输入到相同的位置

[root@exercise1 ~]# ls /tmp/ xxx > /opt/ok.txt 2>&1

或者

[root@exercise1 ~]# ls /tmp/ xxx 2> /opt/error.txt 1>&2
```


```
例 3 ： 工作中shell 脚本中的 >/dev/null 是什么意思
[root@exercise1 ~]# cat /etc/passwd > /dev/null
注：将标准输出全部重定向到/dev/null中,也就是将产生的所有信息丢弃
```

10.管道 | 的使用

作用：上一个命令的输出结果为下一个命令的输入

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



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，必须能够接收标准输入的数据流命令才行

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

4、有些命令如mv ,cp ,rm 不支持管道

```
[root@exercise1 ~]# ll /opt/ | grep txt | grep right
-rw-r--r--. 1 root root        0 1月  19 22:37 right.txt
[root@exercise1 ~]#
```

11.xargs命令

功能：捕获上一个命令的结果，作为下一个命令的参数

参数:

-i 告诉xargs命令用每项的名称替换{}

```
例1
[root@exercise1 opt]# vim a.txt
Aaa
Bbb
Ccc
Ddd
abcd
[root@exercise1 opt]# cat /opt/a.txt | grep a
Aaa
abcd
[root@exercise1 opt]# cat /opt/a.txt | xargs grep a
grep: Aaa: 没有那个文件或目录
grep: Bbb: 没有那个文件或目录
grep: Ccc: 没有那个文件或目录
grep: Ddd: 没有那个文件或目录
grep: abcd: 没有那个文件或目录
```

```
例2
[root@exercise1 opt]# touch a.txt
[root@exercise1 opt]# ls |xargs -i mv {} {}.bak    #等同于mv a.txt a.txt.bak
[root@exercise1 opt]# ll
总用量 0
-rw-r--r--. 1 root root 0 1月  19 23:01 a.txt.bak
```

12.tee 命令（了解）

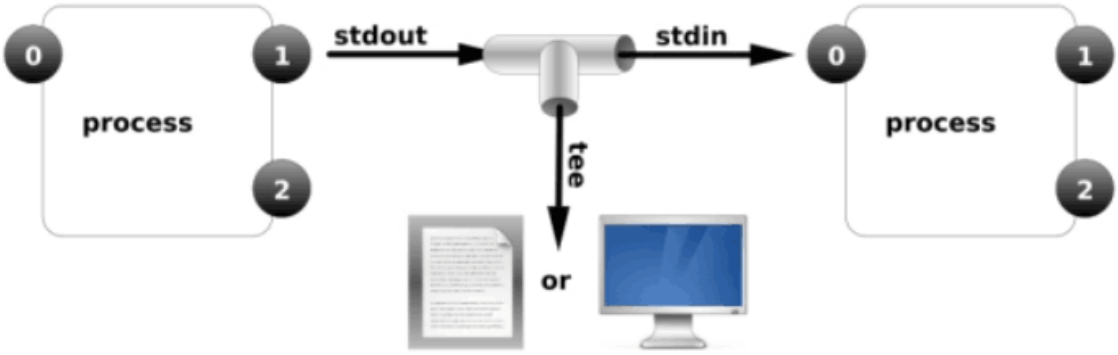
功能：读取标准输入的数据，并将其内容输出成文件。

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 指令会从标准输入读取数据，将其内容输出到标准输出，同时保存成文件



```
例 1 ：将磁盘使用的信息写入文件
[root@exercise1 opt]# df -h | tee /opt/disk.log
文件系统         容量   已用   可用 已用% 挂载点
/dev/sda3         18G  2.3G   16G   13% /
devtmpfs         479M     0  479M    0% /dev
tmpfs            489M     0  489M    0% /dev/shm
tmpfs            489M  6.7M  482M    2% /run
tmpfs            489M     0  489M    0% /sys/fs/cgroup
/dev/sr0         4.3G  4.3G     0  100% /mnt
/dev/sda1        197M   97M  100M   50% /boot
tmpfs             98M     0   98M    0% /run/user/0
```

例 2： 将文件系统使用的信息追加到文件
[root@exercise1 opt]# df -h | tee -a /opt/disk.log
注： 可以使用来记录日志

---

tee命令与重定向">"区别

```
[root@exercise1 opt]# df -h > /opt/disk.log   #直接覆盖进去
[root@exercise1 opt]# df -h | tee /opt/disk.log   #查询并显示结果写入
文件系统         容量   已用   可用 已用% 挂载点
/dev/sda3         18G  2.3G   16G   13% /
devtmpfs         479M     0  479M    0% /dev
tmpfs            489M     0  489M    0% /dev/shm
tmpfs            489M  6.7M  482M    2% /run
tmpfs            489M     0  489M    0% /sys/fs/cgroup
/dev/sr0         4.3G  4.3G     0  100% /mnt
/dev/sda1        197M   97M  100M   50% /boot
tmpfs             98M     0   98M    0% /run/user/0
```

13.命令判断

用到的三个特殊符号：　　；　　&&　　||

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

```
[root@exercise1 opt]# ls /opt/;touch b.txt
a.txt.bak  disk.log
[root@exercise1 opt]#
```

2、&& 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@exercise1 opt]# cd /opt/ && touch /opt/c.txt && ls /opt/
a.txt.bak  b.txt  c.txt  disk.log
[root@exercise1 opt]#
```

例 2： 源码编译经典使用方法

```
[root@exercise1 opt]# ./configure   &&   make -j 4 && make install     #现在没有源码包，所以此命令不能执行成功。
```

3、 || 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；或者如果前面的执行不成功，才 会去执行后面的命令

```
[root@exercise1 opt]# ls xxxx || cd /etc/
ls: 无法访问xxxx: 没有那个文件或目录
[root@exercise1 etc]#

[root@exercise1 etc]# ls /etc/passwd || cd /opt/
/etc/passwd
[root@exercise1 etc]#
```

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	

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@exercise1 etc]# cd /opt/back || mkdir /opt/back && touch /opt/back/back.tar && ls /opt/back
```