```
adfasfd
# read VAR < a.txt
# echo $VAR
adfasfd
while 循环读取每一行作为 read 的标准输入：
# cat a.txt |while read LINE; do echo $LINE; done
123
abc
分别变量赋值：
# read a b c
1 2 3
# echo $a
1
# echo $b
2
# echo $c
3
# echo 1 2 3 | while read a b c;do echo "$a $b $c"; done
1 2 3
```

第九章　Shell 信号发送与捕捉

9.1 Linux 信号类型

信号（Signal）：信号是在软件层次上对中断机制的一种模拟，通过给一个进程发送信号，执行相应的处理函数。
进程可以通过三种方式来响应一个信号：
1）忽略信号，即对信号不做任何处理，其中有两个信号不能忽略：SIGKILL 及 SIGSTOP。
2）捕捉信号。
3）执行缺省操作，Linux 对每种信号都规定了默认操作。
Linux 究竟采用上述三种方式的哪一个来响应信号呢？取决于传递给响应的 API 函数。
Linux 支持的信号有：

编号	信号名称	缺省动作	描述
1	SIGHUP	终止	终止进程，挂起
2	SIGINT	终止	键盘输入中断命令，一般是 CTRL+C
3	SIGQUIT	CoreDump	键盘输入退出命令，一般是 CTRL+\
4	SIGILL	CoreDump	非法指令
5	SIGTRAP	CoreDump	trap 指令发出，一般调试用

6	SIGABRT	CoreDump	abort(3)发出的终止信号
7	SIGBUS	CoreDump	非法地址
8	SIGFPE	CoreDump	浮点数异常
9	SIGKILL	终止	立即停止进程，不能捕获，不能忽略
10	SIGUSR1	终止	用户自定义信号 1，像 Nginx 就支持 USR1 信号，用于重载配置，重新打开日志
11	SIGSEGV	CoreDump	无效内存引用
12	SIGUSR2	终止	用户自定义信号 2
13	SIGPIPE	终止	管道不能访问
14	SIGALRM	终止	时钟信号，alrm(2)发出的终止信号
15	SIGTERM	终止	终止信号，进程会先关闭正在运行的任务或打开的文件再终止，有时候有的进程在运行任务会忽略此信号。不能捕捉
16	SIGSTKFLT	终止	处理器栈错误
17	SIGCHLD	可忽略	子进程结束时，父进程收到的信号
18	SIGCONT	可忽略	让终止的进程继续执行
19	SIGSTOP	停止	停止进程，不能忽略，不能捕获
20	SIGSTP	停止	停止进程，一般是 CTRL+Z
21	SIGTTIN	停止	后台进程从终端读数据
22	SIGTTOU	停止	后台进程从终端写数据
23	SIGURG	可忽略	紧急数组是否到达 socket
24	SIGXCPU	CoreDump	超出 CPU 占用资源限制
25	SIGXFSZ	CoreDump	超出文件大小资源限制
26	SIGVTALRM	终止	虚拟时钟信号，类似于 SIGALRM，但计算的是进程占用的时间
27	SIGPROF	终止	类似与 SIGALRM，但计算的是进程占用 CPU 的时间
28	SIGWINCH	可忽略	窗口大小改变发出的信号

29	SIGIO	终止	文件描述符准备就绪，可以输入/输出操作了
30	SIGPWR	终止	电源失败
31	SIGSYS	CoreDump	非法系统调用

CoreDump（核心转储）：当程序运行过程中异常退出时，内核把当前程序在内存状况存储在一个 core 文件中，以便调试。执行命令 ulimit -c 如果是 0 则没有开启，也不会生成 core dump 文件，可通过 ulimit -c unlimited 命令临时开启 core dump 功能，只对当前终端环境有效，如果想永久生效，可修改/etc/security/limites.conf 文件，添加一行 "* soft core unlimited"
默认生成的 core 文件保存在可执行文件所在的目录下，文件名为 core。如果想修改 core 文件保存路径，可通过修改内核参数：echo "/tmp/corefile-%e-%p-%t" > /proc/sys/kernel/core_pattern
则文件名格式为 core-命名名-pid-时间戳

Linux 支持两种信号：
一种是标准信号，编号 1-31，称为非可靠信号（非实时），不支持队列，信号可能会丢失，比如发送多次相同的信号，进程只能收到一次，如果第一个信号没有处理完，第二个信号将会丢弃。
另一种是扩展信号，编号 32-64，称为可靠信号（实时），支持队列，发多少次进程就可以收到多少次。
信号类型比较多，我们只要了解下，记住几个常用信号就行了，红色标记的我觉得需要记下。

发送信号一般有两种情况：
一种是内核检测到系统事件，比如键盘输入 CTRL+C 会发送 SIGINT 信号。
另一种是通过系统调用 kill 命令来向一个进程发送信号。

9.2 kill 命令

kill 命令发送信号给进程。
命令格式：kill [-s sigspec | -n signum | -sigspec] pid | jobspec ...
kill -l [sigspec]
-s # 信号名称
-n # 信号编号
-l # 打印编号 1-31 信号名称
示例：

```
给一个进程发送终止信号：
kill -s SIGTERM pid
或
kill -n 15 pid
或
kill -15 pid
或
kill -TREM pid
```

9.3 trap 命令

trap 命令定义 shell 脚本在运行时根据接收的信号做相应的处理。

命令格式：trap [-lp] [[arg] signal_spec ...]
-l   # 打印编号 1-64 编号信号名称
arg   # 捕获信号后执行的命令或者函数
signal_spec # 信号名或编号
一般捕捉信号后，做以下几个动作：
1）清除临时文件
2）忽略该信号
3）询问用户是否终止脚本执行
示例 1：按 CTRL+C 不退出循环

```
#!/bin/bash
trap "" 2      # 不指定 arg 就不做任何操作，后面也可以写多个信号，以空格分隔
for i in {1..10}; do
     echo $i
     sleep 1
done
# bash a.sh
1
2
3
^C4
5
6
^C7
8
9
10
```

示例 2：循环打印数字，按 CTRL+C 退出，并打印退出提示

```
#!/bin/bash
trap "echo 'exit...';exit" 2
for i in {1..10}; do
     echo $i
     sleep 1
done
# bash test.sh
1
2
3
^Cexit...
```

示例 3：让用户选择是否终止循环

```
#!/bin/bash
trap "func" 2
func() {
     read -p "Terminate the process? (Y/N): " input
     if [ $input == "Y" ]; then
            exit
     fi
```

```
}
for i in {1..10}; do
     echo $i
     sleep 1
done

# bash a.sh
1
2
3
^CTerminate the process? (Y/N): Y
# bash a.sh
1
2
3
^CTerminate the process? (Y/N): N
4
5
6
...
```

第十章　Shell 编程时常用的系统文件

10.1 Linux 系统目录结构

/	根目录，所有文件的第一级目录
/home	普通用户家目录
/root	超级用户家目录
/usr	用户命令、应用程序等目录
/var	应用数据、日志等目录
/lib	库文件和内核模块目录
/etc	系统和软件配置文件
/bin	可执行程序目录
/boot	内核加载所需的文件，grub 引导
/dev	设备文件目录，比如磁盘驱动