

*Linux*环境及开发工具应用实践

# -系统操作及管理

[zhaofang@email.buptsse.cn](mailto:zhaofang@email.buptsse.cn)



软件学院 赵方

# 目录

## 1. **shell** 编程概述



## 2. **Shell**的变量



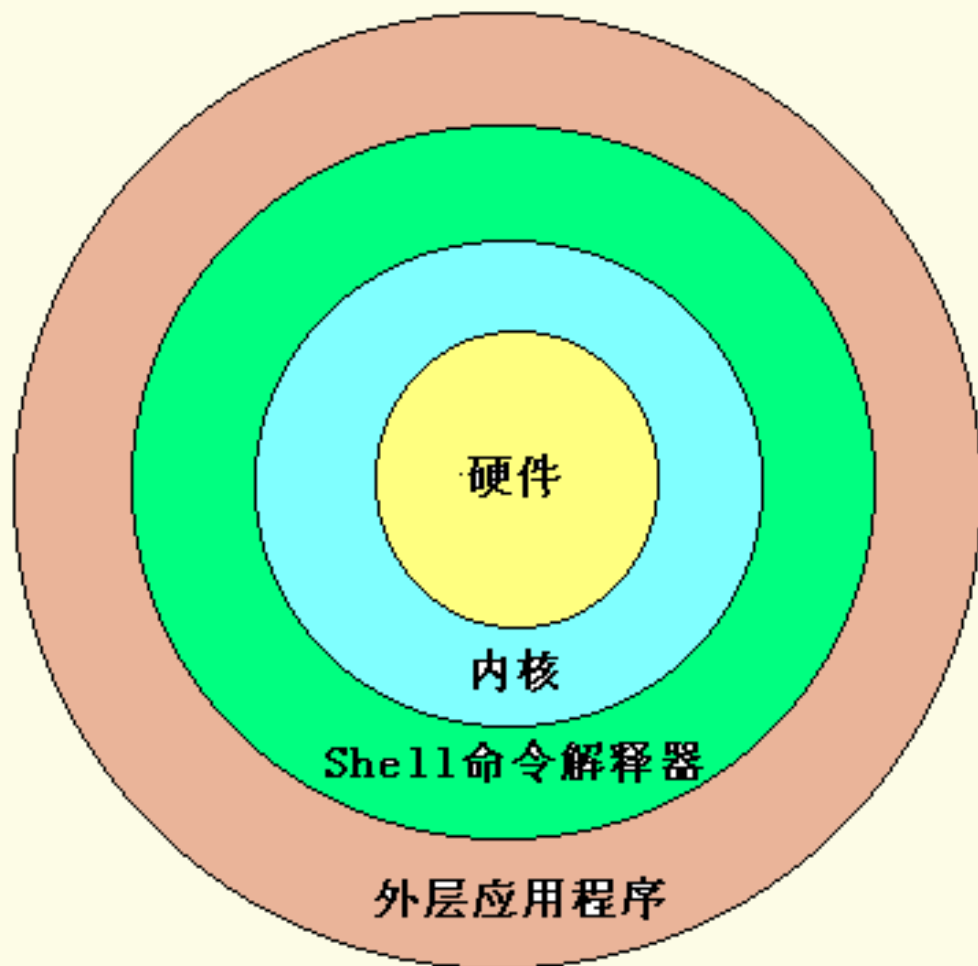
## 3. **Shell**流程控制



## 4. **Shell**的函数



# 什么是 Shell



# Shell程序语言的特点

Shell允许通过编程来完成复杂的功能处理，但其作为语言与高级语言比较具有不同的特点：

- ☆ Shell是解释性的，多数高级语言是编译性的；
- 🕒 Shell语言与高级语言处理的对象不同；
- 🕒 Shell与系统有密切的关系；
- 🕒 Shell易编写、调试、灵活性较强；
- 🕒 Shell作为命令级语言，命令组合功能很强。

# Shell的版本

## 常用Shell在交互式使用方面的比较

Shell	别名	兼容 Bourne	兼容 C	兼容 Korn	兼容 Bash
Bourne	(sh)	是	否	否	否
Korn	(ksh)	是	否	是	是
Bourne Again	(bash)	是	否	是	是
POSIX	(psh)	是	否	是	是
C	(csh)	是	是	是	否
TC	(tcsh)	是	是	是	否
Z	(zsh)	是	是	是	否

**Shell**有两种主要语法类型： Bourne和C，彼此不兼容，

Bourne家族： sh ksh bash psh zsh

C 家族： csh tcsh

其中： bash和 zsh在不同程度上支持 csh 的语法。

# Shell版本的选择

- ❖ 各种shell的应用：
- ❖ Solaris和FreeBSD中是Bourne Shell。
- ❖ 在HP-UX中是POSIX Shell。
- ❖ 在Linux中是Bourn Again Shell。

# Shell启动的过程

- ❖ Shell启动时读入下面两个文件：
- ❖ /etc/profile
- ❖ .profile
- ❖ 过程：
  - ❖ 1、Shell检查文件/etc/profile是否存在。
  - ❖ 2、如果存在，则shell读取其中信息。否则，略过该文件，不显示错误信息。
  - ❖ 3、shell检查文件.profile是否在用户起始目录中。
  - ❖ 4、如果存在，则shell读取该文件。否则，shell略过该文件，不显示错误信息。
  - ❖ 5、这些文件被读取后，shell显示一个提示符。

# Shell的特点

- ❖ 很多系统文件都是用shell语言编写，如.profile，.login文件。
- ❖ 把已有的命令进行适当组合，可以构成新的命令，并且组合方式很简单；
- ❖ 可以进行交互式处理，用户与系统之间通过shell进行会话；
- ❖ 结构化的程序模块，提供了多种控制流程语句；
- ❖ 灵活地利用位置参数传递参数值；
- ❖ 提供通配符、输入/输出重定向、管道线等机制，便于模式匹配、I/O处理和数据传输；
- ❖ 便于用户开发新的命令，利用shell过程可把用户编写的可执行程序与UNIX命令结合在一起，作为新的命令使用；
- ❖ 可以对环境进行控制，以满足用户的应用需求；
- ❖ 可以对数据和文件进行基本测试。



# Shell的功能

- ❖ Shell不仅是一种命令解释器，还是一种强大的语言，有类似ALGOI语言的编程语法。它包含如下特征：
- ❖ 1、进程控制
- ❖ 2、变量
- ❖ 3、正则表达式
- ❖ 4、流控制
- ❖ 5、强大的输入输出控制
- ❖ 6、函数

# Shell编程

- ❖ 1. 如果Script的第一个非空白字符不是"#", 则它会使用Bourne Shell。
- ❖ 2. 如果Script的第一个非空白字符是"#"时, 但不以"#!"开头时, 则它会使用C Shell。
- ❖ 3. 如果Script以“#!”开头, 则“#!”后面所写的就是所使用的Shell, 而且要将整个路径名称指出来。
- ❖ 这里建议使用第三种方式指定Shell, 以确保所执行的就是所要的。Bourne Shell的路径名称为/bin/sh, 而C Shell 则为/bin/csh。

# Shell编程

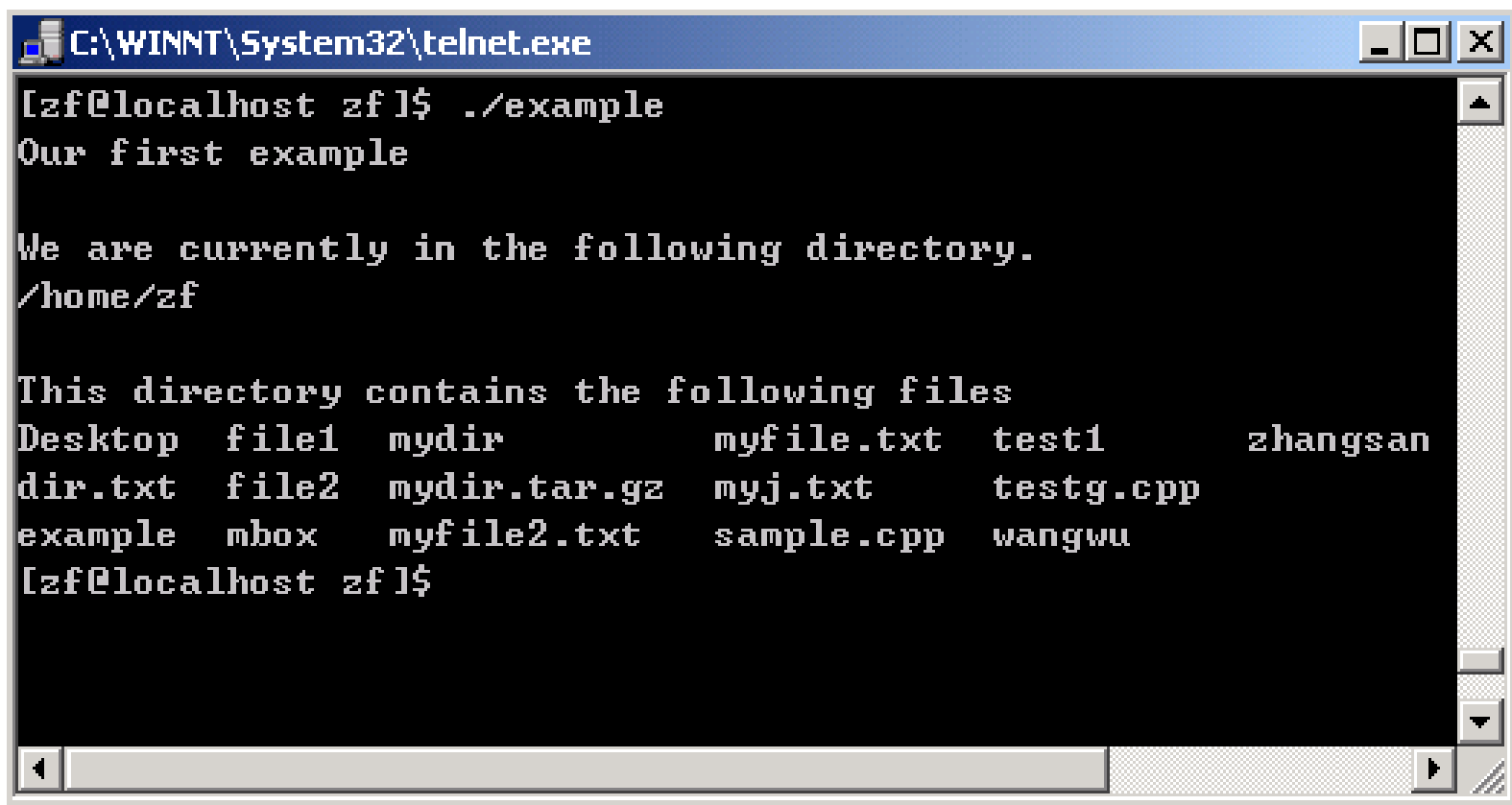
- ❖ 执行shell脚本的方式基本上有三种：
- ❖ 1、以脚本文件名作为shell的参数：
  - ❖ `$sh 脚本文件名 [参数]`
  - 2、输入定向到shell脚本
- ❖ `$sh <脚本文件名`
- ❖ 3、将shell脚本的文件权限改为可执行的
  - ❖ `$chmod u+x 脚本文件名`

# 一个简单的shell程序

- ❖ `$ cat example01`
- ❖ `#!/bin/bash`
- ❖ `#This is to show what a example looks like.`
- ❖ `echo "Our first example"`
- ❖ `echo #` This inserts an empty line in output.
- ❖ `echo "We are currently in the following directory."`
- ❖ `pwd`
- ❖ `echo`
- ❖ `echo "This directory contains the following files"`
- ❖ `ls`

# 一个简单的shell程序

❖ 运行结果:



The screenshot shows a Windows-style window titled "C:\WINNT\System32\telnet.exe". The terminal content is as follows:

```
[zf@localhost zf]$ ./example
Our first example

We are currently in the following directory.
/home/zf

This directory contains the following files
Desktop  file1  mydir          myfile.txt  test1       zhangsan
dir.txt  file2  mydir.tar.gz   myj.txt     testg.cpp
example  mbox  myfile2.txt    sample.cpp  wangwu
[zf@localhost zf]$
```

# Shell变量

- ☞ Shell实际上是基于字符串的程序设计语言，也具有变量。变量的名字必须以字母或下划线开头，可以包括字母、数字和下划线。
- ☞ Shell变量能够而且只能存储正文字符串，即它只有一种类型的变量——串变量。
- ☞ 从赋值的形式上看，则可以分成四种类型的变量或变量形式。

# 使用变量

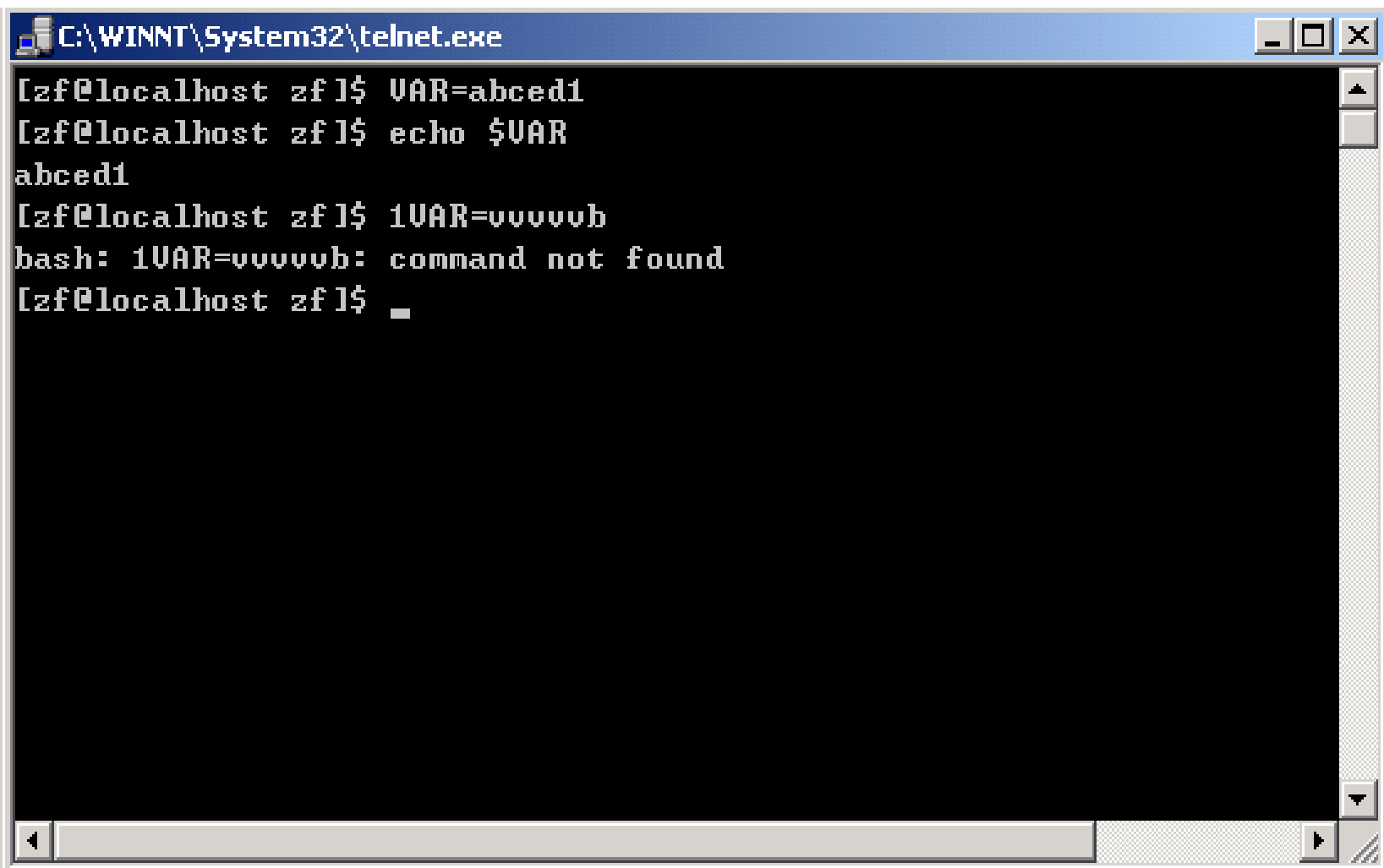
- ❖ 变量是**shell**传递数据的一种方法。变量是用来代表每个值的符号名。
- ❖ **Shell**有两类变量：临时变量和永久变量。
- ❖ 临时变量是**shell**程序内部定义的，其使用范围仅限于定义它的程序，对其它程序不可见。包括：用户自定义变量、位置变量和预定义变量。
- ❖ 永久变量是环境变量，其值不随**shell**脚本的执行结束而消失。

# 用户自定义变量

- ❖ 用户定义的变量由字母或下划线打头，由字母、数字或下划线序列组成，并且大小写字母意义不同。变量名长度没有限制。
- ❖ 在使用变量值时，要在变量名前加上前缀“\$”。



# 设置和使用变量



```
C:\WINNT\System32\telnet.exe

[zf@localhost zf]$ VAR=abcd1
[zf@localhost zf]$ echo $VAR
abcd1
[zf@localhost zf]$ 1VAR=vvvvvb
bash: 1VAR=vvvvvb: command not found
[zf@localhost zf]$ _
```

# 用户自定义变量

## ☞ 语法规式: **name=string**

赋值号 “=” 两边不允许有空白符;

```
nodehost="beijing.UUCP "  
path=/bin:/usr/bin:/etc/bin  
count=10
```

## ☞ 允许多个赋值操作, 按从右到左的顺序进行;

```
# A=$B B=abc C="OK"
```

```
# echo $A $B $C
```

```
abc abc OK
```

## ☞ 当引用一个未设置的变量时, 其隐含值为空;

```
# echo "$mail is path of mailbox"
```

```
is path of mailbox
```

# 用户自定义变量（续1）

☞ 如果用双引号 “ ” 将值括起来，则括起来的字符串允许出现空格、制表符和换行符的特殊字符，而且允许有变量替换。

例1: **#** MAIL=/var/mail/fk  
      **#** var="\$MAIL is a path of mailbox"  
      **#** echo \$var  
\_\_\_\_ /var/mail/fk is a path of mailbox

例2: **#** str="This is \n a book"  
      **#** echo \$str  
      This is  
      a book

# 用户自定义变量（续2）

☞ 如果用单引号 ‘’ 将值括起来，则括起来的字符串允许出现空格、制表符和换行符的特殊字符，但不允许有变量替换。

例1 **#** BOOK="English book"

**#** MSG='\$BOOK'

**#** echo \$MSG

\$BOOK

例2 **#** msg=' Today is \t Sunday'

**#** echo \$msg

Today is            Sunday

# 用户自定义变量（续3）

☞ 引用变量的值时，可以用花括号{}将变量名称括起来，使变量名称与它的后续字符分隔开，如果紧跟在变量名称后面的字符是字母、数字或下划线时，必须要使用花括号。

例： **#** str='This is a string'  
     **#** echo "\${str}ent test of variables"  
     This is a stringent test of variables  
     **#** echo "\$strent test of variables"  
     test of variables

# 设置和使用变量

C:\WINNT\System32\telnet.exe

```
[Bossdev]$ FIRST=`pwd`  
[Bossdev]$ echo $FIRST  
/dev_u/zhaofang/test  
[Bossdev]$ SECOND=$(ls -l)  
[Bossdev]$ echo $SECOND  
总数 16 -rwxr--r-- 1 zhaofang devgrp 170 1月 7日 15:12 assign_logfile -rwxr--r--  
1 zhaofang devgrp 80 1月 7日 15:02 display_parameters  
[Bossdev]$ THIRD=$FIRST  
[Bossdev]$ echo $THIRD  
/dev_u/zhaofang/test  
[Bossdev]$ _
```

# 简单变量赋值

C:\WINNT\System32\telnet.exe

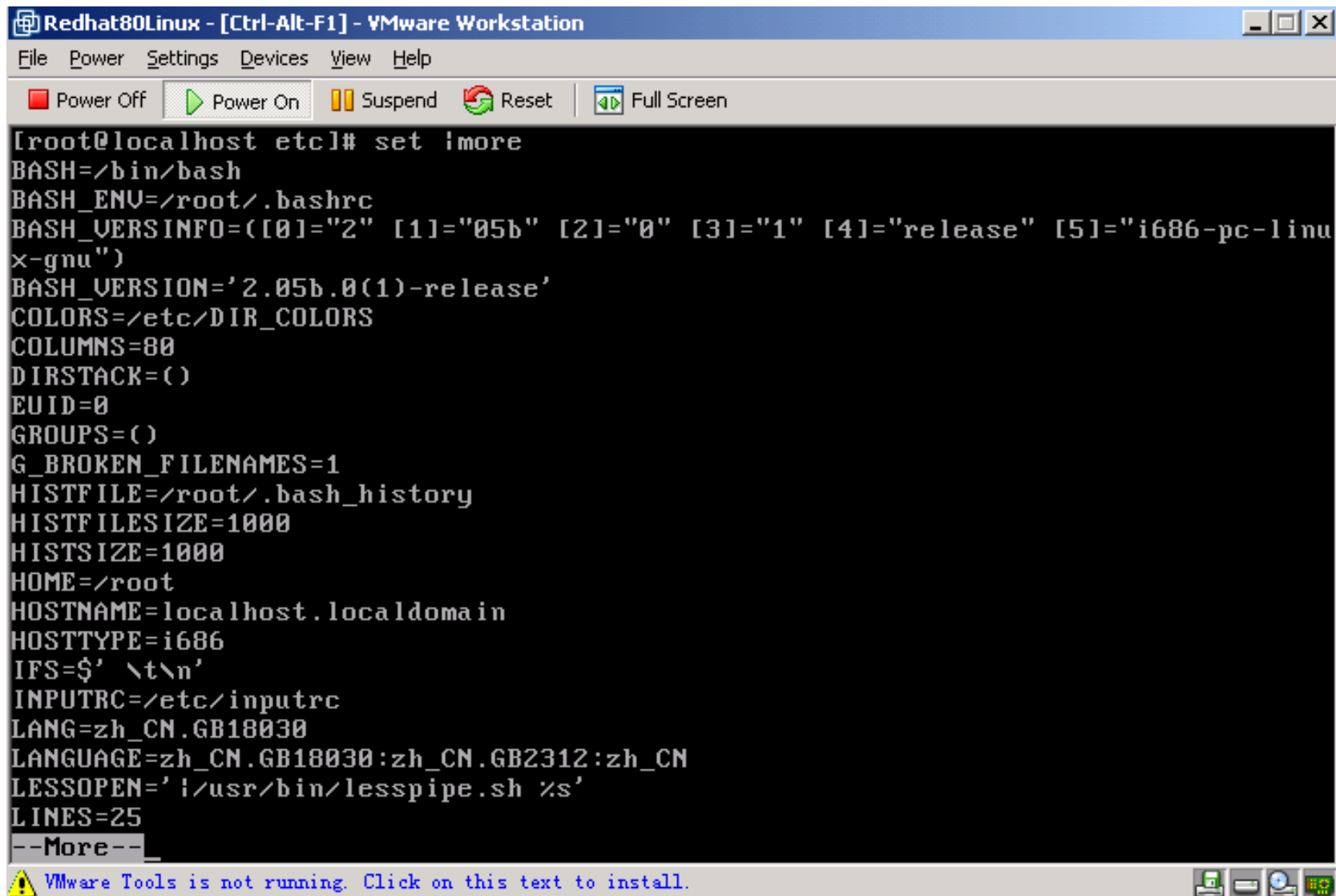
```
[Bossdev]$ cat assign_logfile
#The following line is the assignment command.
LOGFILE="monday.dat"
echo "The value of logfile is :"
#The dollar sign is used to access a variable's value.
echo $LOGFILE
[Bossdev]$ ./assign_logfile
The value of logfile is :
monday.dat
[Bossdev]$
```

# 设置和使用变量

- ❖ 列出所有的变量:
- ❖ **set** 命令
- ❖ 包含多个字的变量:
- ❖ `$NAME=Mike Ron`      运行时出错, 应改为:
- ❖ `$NAME="Mike Ron"` 或 `$NAME='Mike Ron'`
- ❖ 修改变量:
- ❖ `$NAME="$NAME Junior"`



# 设置和使用变量



```
Redhat80Linux - [Ctrl-Alt-F1] - VMware Workstation
File Power Settings Devices View Help
Power Off Power On Suspend Reset Full Screen

[root@localhost etc]# set imore
BASH=/bin/bash
BASH_ENV=/root/.bashrc
BASH_VERSIONINFO=( [0]="2" [1]="05b" [2]="0" [3]="1" [4]="release" [5]="i686-pc-linu
x-gnu" )
BASH_VERSION='2.05b.0(1)-release'
COLORS=/etc/DIR_COLORS
COLUMNS=80
DIRSTACK=()
EUID=0
GROUPS=()
G_BROKEN_FILENAMES=1
HISTFILE=/root/.bash_history
HISTFILESIZE=1000
HISTSIZE=1000
HOME=/root
HOSTNAME=localhost.localdomain
HOSTTYPE=i686
IFS=$' \t\n'
INPUTRC=/etc/inputrc
LANG=zh_CN.GB18030
LANGUAGE=zh_CN.GB18030:zh_CN.GB2312:zh_CN
LESSOPEN='! /usr/bin/lesspipe.sh %s'
LINES=25
--More--
```

VMware Tools is not running. Click on this text to install.

Company name

# 设置和使用变量

```
Redhat80Linux - [Ctrl-Alt-F1] - VMware Workstation
File Power Settings Devices View Help

Power Off Power On Suspend Reset Full Screen

LINES=25
LOGNAME=root
LS_COLORS='no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:cd=40;33;
01:or=01;05;37;41:mi=01;05;37;41:ex=01;32:*.cmd=01;32:*.exe=01;32:*.com=01;32:*.
btm=01;32:*.bat=01;32:*.sh=01;32:*.csh=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31
:*.taz=01;31:*.lzh=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:
*.bz=01;31:*.tz=01;31:*.rpm=01;31:*.cpio=01;31:*.jpg=01;35:*.gif=01;35:*.bmp=01;
35:*.xbm=01;35:*.xpm=01;35:*.png=01;35:*.tif=01;35:'
MACHTYPE=i686-pc-linux-gnu
MAIL=/var/spool/mail/root
MAILCHECK=60
OLDPWD=/
OPTERR=1
OPTIND=1
OSTYPE=linux-gnu
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin
:/root/bin
PIPESTATUS=([""])
PPID=731
PS1='[\u@\h \W]\$ '
PS2='> '
PS4='+ '
PWD=/etc
SHELL=/bin/bash
--More--_

VMware Tools is not running. Click on this text to install.
```

# 位置变量和特殊变量

- ❖ Shell解释执行用户的命令时，将命令行的第一个字作为命令名，而其它字作为参数。由出现在命令行上的位置确定的参数称为位置参数。
- ❖ 例如：
- ❖ `$example file1 file2 file3`
- ❖ `$0` 这个程序的文件名 `example`  
`$n` 这个程序的第n个参数值，`n=1..9`

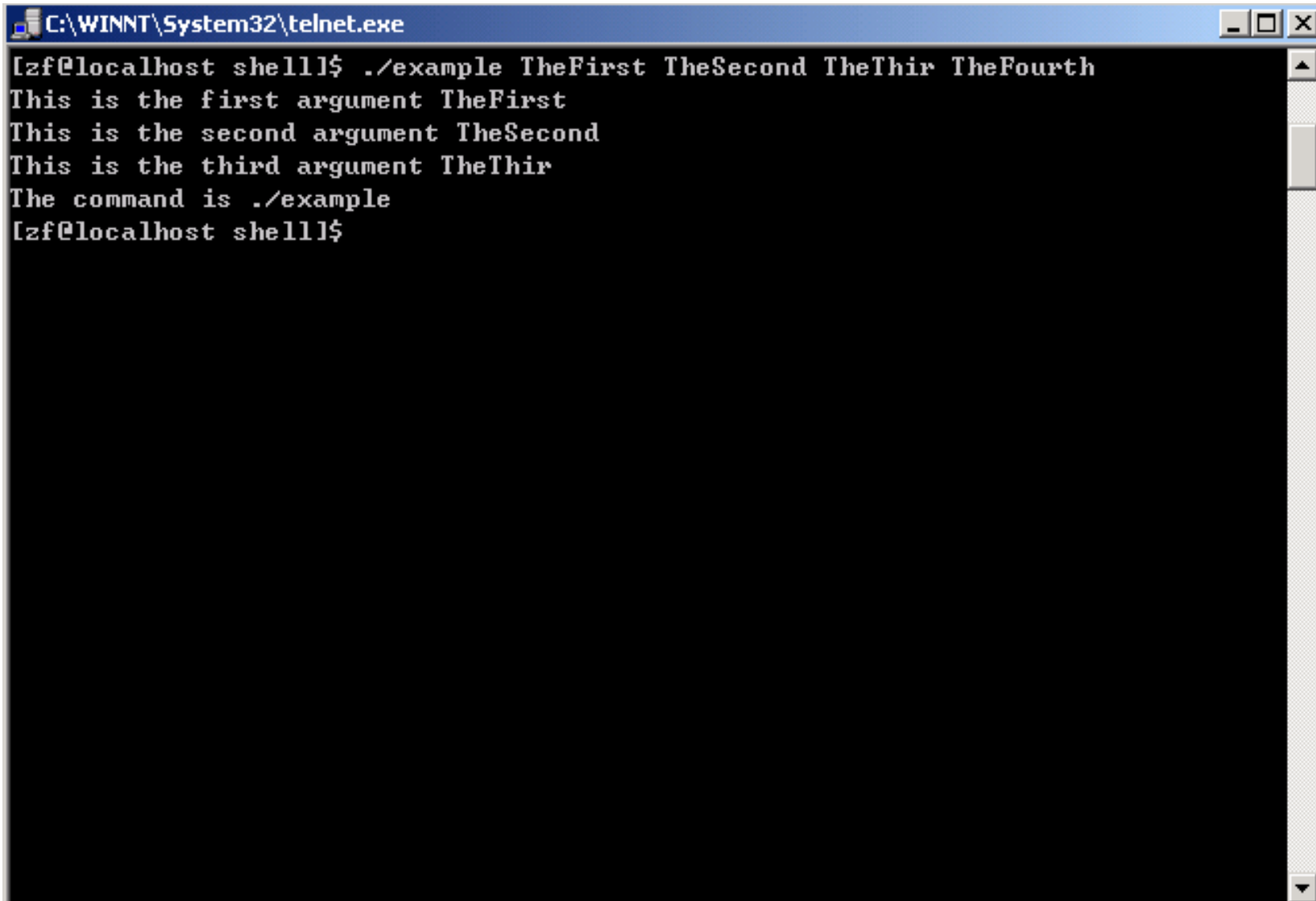
# 位置变量和特殊变量

[illegible]

# 位置变量和特殊变量

```
C:\WINNT\System32\telnet.exe  
[zf@localhost zf]$ ./example  
This is the first argument  
This is the second argument  
This is the third argument  
The command is ./example  
[zf@localhost zf]$
```

# 位置变量和特殊变量

A screenshot of a Windows telnet window titled "C:\WINNT\System32\telnet.exe". The window shows a shell session where a user runs the command `./example TheFirst TheSecond TheThir TheFourth`. The script `./example` prints four lines of output: "This is the first argument TheFirst", "This is the second argument TheSecond", "This is the third argument TheThir", and "The command is ./example". The session ends with the prompt `[zf@localhost shell]$`.

```
C:\WINNT\System32\telnet.exe
[zf@localhost shell]$ ./example TheFirst TheSecond TheThir TheFourth
This is the first argument TheFirst
This is the second argument TheSecond
This is the third argument TheThir
The command is ./example
[zf@localhost shell]$
```

# 特殊变量

- ❖ 有些变量是一开始执行**Script**时就会设定，且不能被修改，但我们不叫它只读的系统变量，而叫它特殊变量。这些变量当一执行程序时就有了，用户无法将一般的系统变量设定成只读的。以下是一些特殊变量：

**\$\*** 这个程序的所有参数

**\$#** 这个程序的参数个数

**\$\$** 这个程序的**PID**

**\$!** 执行上一个后台指令的**PID**

**\$?** 执行上一个指令的返回值

# 预定义的特殊变量

在Shell中有一组特殊的变量，其变量名和变量值只有Shell本身才可以设置。

👉 **`$#`** — 记录传递给Shell的自变量个数；

例1: `# myprog a b c`                      则 `$#` 的值为3

```
例2:  if test $# -lt 2
        then
            echo "two or more args required"
            exit
        fi
```



# 预定义的特殊变量（续1）

☞ **\$?** — 取最近一次命令执行后的退出状态(返回码) :  
执行成功返回码为0, 执行失败返回码为1;

例: **# test -r my\_file** (假设my\_file文件不可读)  
**# echo \$?**  
1

☞ **\$\$** — 当前Shell的进程号 ;

☞ **\$!** — 取最后一个在后台运行的(使用 “&”)  
进程的进程号 ;

# 预定义的特殊变量（续2）

👉 **\$\*** — 匹配所有位置变量；

☆ **\$\*** 匹配 **\$1 \$2 \$3 ...**

🕒 **"\$\*"** 匹配 **"\$1 \$2 \$3 ..."**

👉 **\$@** — 匹配所有位置变量；

☆ **\$@** 匹配 **\$1 \$2 \$3 ...**

🕒 **"\$@"** 匹配 **"\$1" "\$2" "\$3 " ...**

👉 **\$-** — Shell的标志位，既在Shell启动时使用的选项，或用**set**命令方式所提供的选项。

# 特殊变量

选定 C:\WINNT\System32\telnet.exe

```
echo There are $# arguments to the command $0:$*
```

```
ehco first argument is :$1
```

```
echo second argument is :$2
```

```
echo here the arguments are :$@
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

```
~
```

# 特殊变量

 C:\WINNT\System32\telnet.exe

```
[zf@localhost zf]$ ./argument This is a test!
```

```
There are 4 arguments to the command ./argument: This is a test!
```

```
first argument is : This
```

```
second argument is : is
```

```
here the arguments are : This is a test!
```

```
[zf@localhost zf]$
```

# 系统环境变量

❖ **/etc/profile**: 定义系统全局的工作环境，用户主目录下的**.profile**: 定义该用户的工作环境。

❖ 主要环境变量有:

**HOME** 用户主目录

**PATH** 搜索路径

**PS1** shell提示符  
前所处的目录

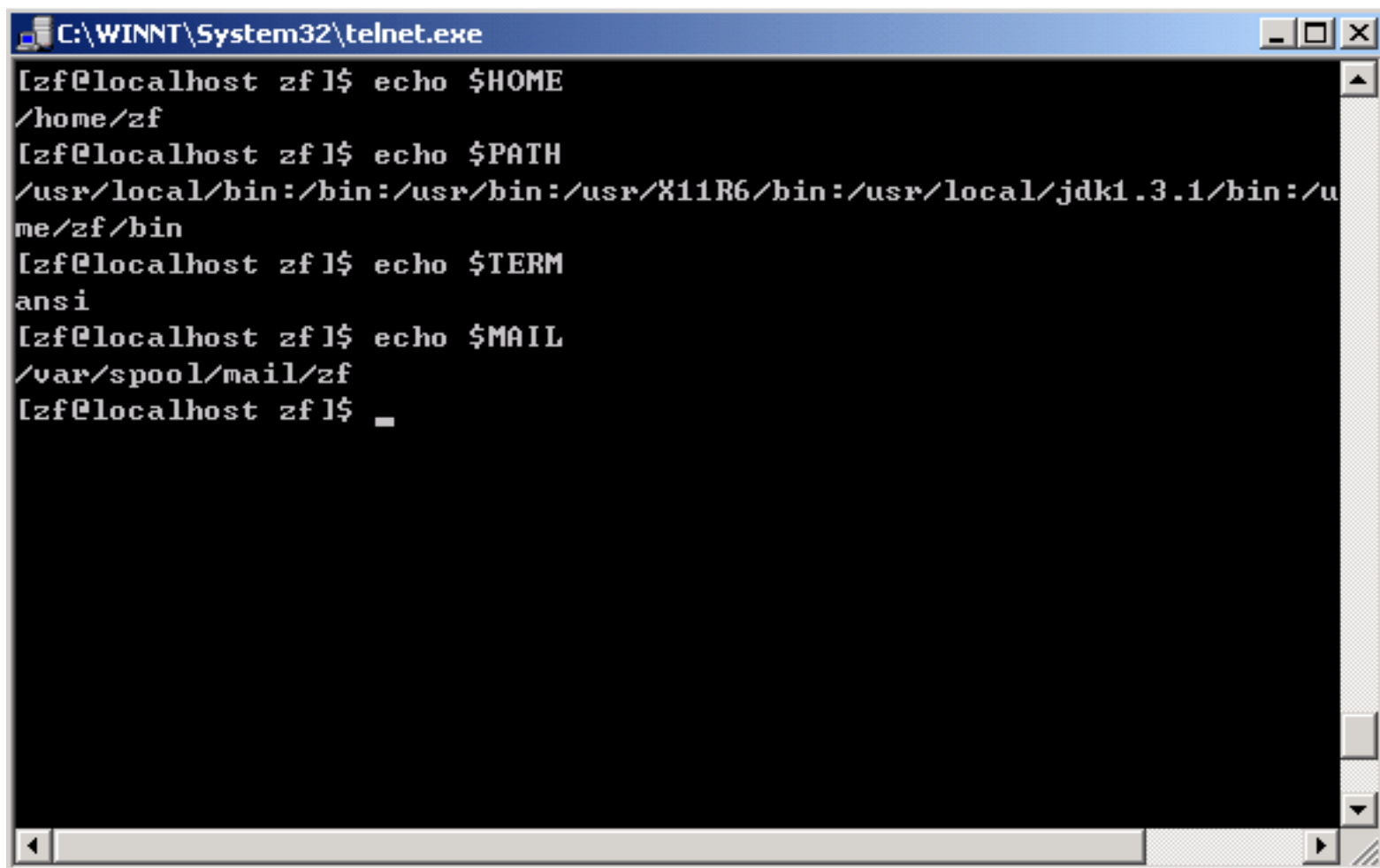
**PWD** 用户当

**MAIL** 邮箱的路径  
类型

**TERM** 使用的终端

# 系统环境变量

## ❖ 显示环境变量:

A screenshot of a Windows telnet window titled "C:\WINNT\System32\telnet.exe". The window shows a terminal session with the following commands and outputs:



```
[zf@localhost zf]$ echo $HOME
/home/zf
[zf@localhost zf]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/jdk1.3.1/bin:/u
me/zf/bin
[zf@localhost zf]$ echo $TERM
ansi
[zf@localhost zf]$ echo $MAIL
/var/spool/mail/zf
[zf@localhost zf]$ _
```

The terminal text is displayed on a black background with white characters. The window has standard Windows controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

# 环境变量设置举例

- ❖ `$ PS1 = "My Computer =>"`
- ❖ `My Computer =>`
- ❖ `PATH= /usr/bin:/usr/ccs/bin:/usr/contrib/bin:/bin`
- ❖ `export`
- ❖ 或:
- ❖ `PATH= /usr/bin:$HOME/bin:/usr/contrib/bin:/bin`
- ❖ 使用**export**命令：通过它可以导出**shell**变量，这样可以对子进程有效。

# .profile文件示例

  C:\WINNT\System32\telnet.exe  
[zf@localhost zf]\$ cat .bash\_profile  
# .bash\_profile  
  
# Get the aliases and functions  
if [ -f ~/.bashrc ]; then  
 . ~/.bashrc  
fi  
  
# User specific environment and startup programs  
  
PATH=\$PATH:\$HOME/bin  
BASH\_ENV=\$HOME/.bashrc  
  
export BASH\_ENV PATH  
unset USERNAME  
  
[zf@localhost zf]\$



# 引号

- ❖ 1、双引号 “”
- ❖ 由双引号括起来的字符（除\$、倒引号、反斜线）通常都作为普通字符对待。
- ❖ 2、单引号 ‘’
- ❖ 由单引号括起来的字符均作为普通字符对待。
- ❖ 3、倒引号 `
- ❖ 由倒引号括起来的字符串被**shell**解释成命令行处理。

# 引号

 C:\WINNT\System32\telnet.exe

```
[zf@localhost zf]$ DIR='pwd'
```

```
[zf@localhost zf]$ echo "This is argument $DIR"
```

```
This is argument pwd
```

```
[zf@localhost zf]$ DIR2='pwd'
```

```
[zf@localhost zf]$ echo "This is argument $DIR2"
```

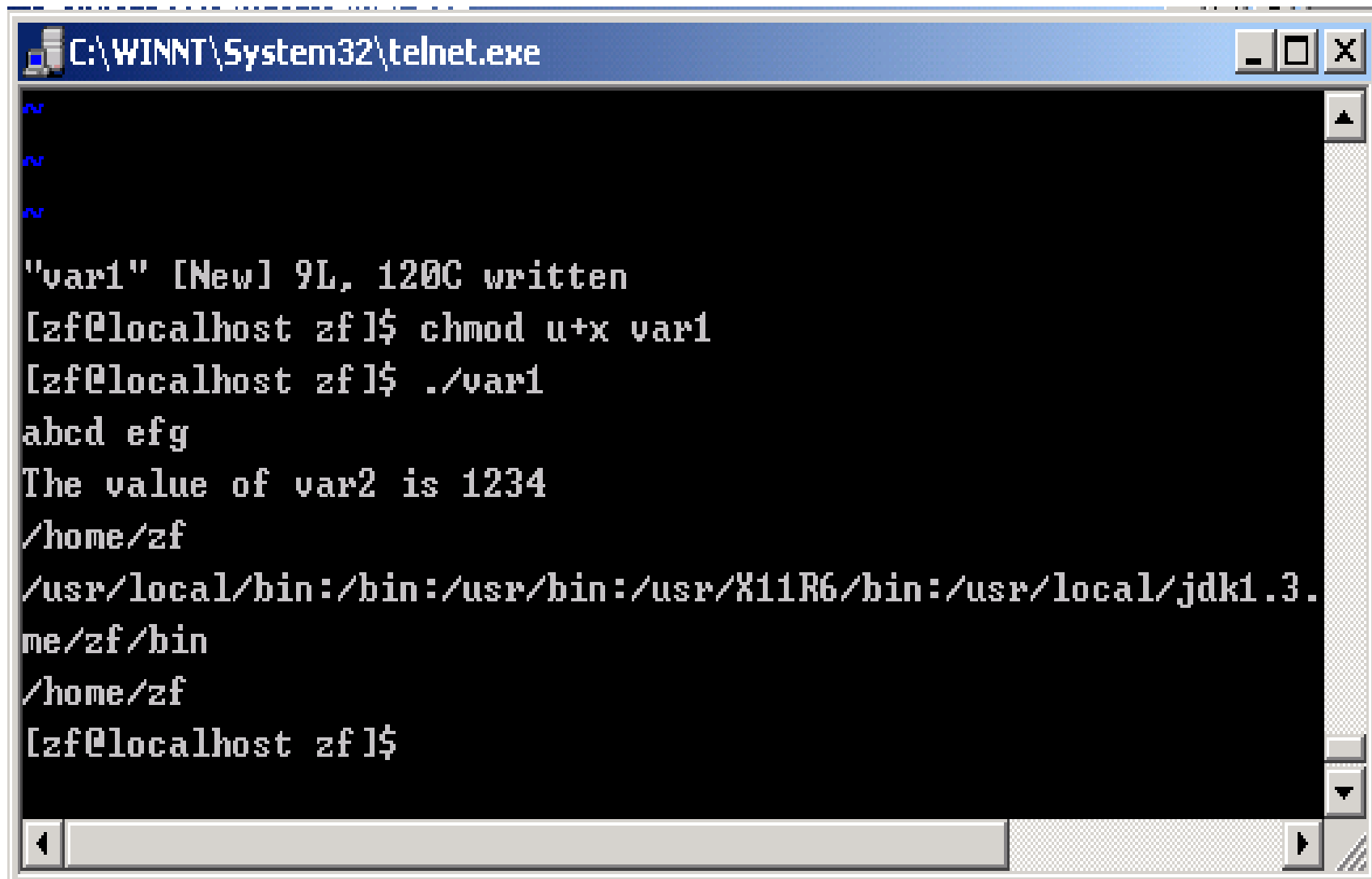
```
This is argument /home/zf
```

```
[zf@localhost zf]$ _
```

# 变量使用的命令

- ❖ `#!/bin/bash`
- ❖ `var1="abcd efg"`
- ❖ `echo $var1`
- ❖ `var2=1234`
- ❖ `echo "The value of var2 is $var2"`
  
- ❖ `echo $HOME`
- ❖ `echo $PATH`
- ❖ `echo $PWD`

# 变量使用的命令



```
C:\WINNT\System32\telnet.exe

"var1" [New] 9L, 120C written
[zf@localhost zf]$ chmod u+x var1
[zf@localhost zf]$ ./var1
abcd efg
The value of var2 is 1234
/home/zf
/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/jdk1.3.
me/zf/bin
/home/zf
[zf@localhost zf]$
```

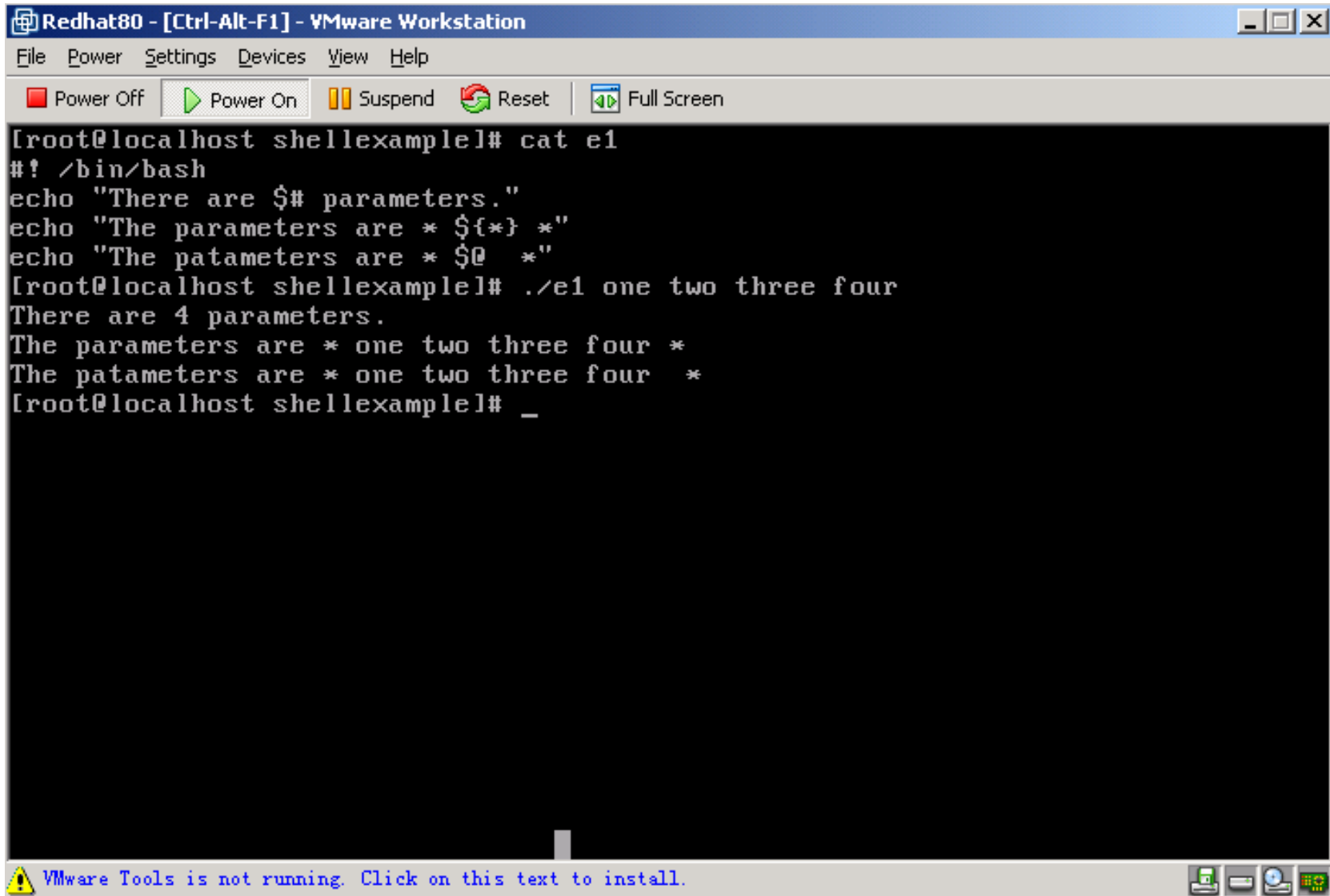
# 特殊字符的引用（续）

## 👉 特殊字符串引用的例外

引用双引号、单引号和转意符都不能消除对 **echo** 命令有特殊功能的控制字符串（逃逸字符）的特殊含义。这些控制字符串是：

- \b** Backspace
- \c** 显示后不换行
- \f** 在终端上屏幕的开始处显示
- \n** 换行
- \r** 回车
- \t** 制表符
- \v** 垂直制表符
- \** 反斜框

# \$\*和\$@的用法

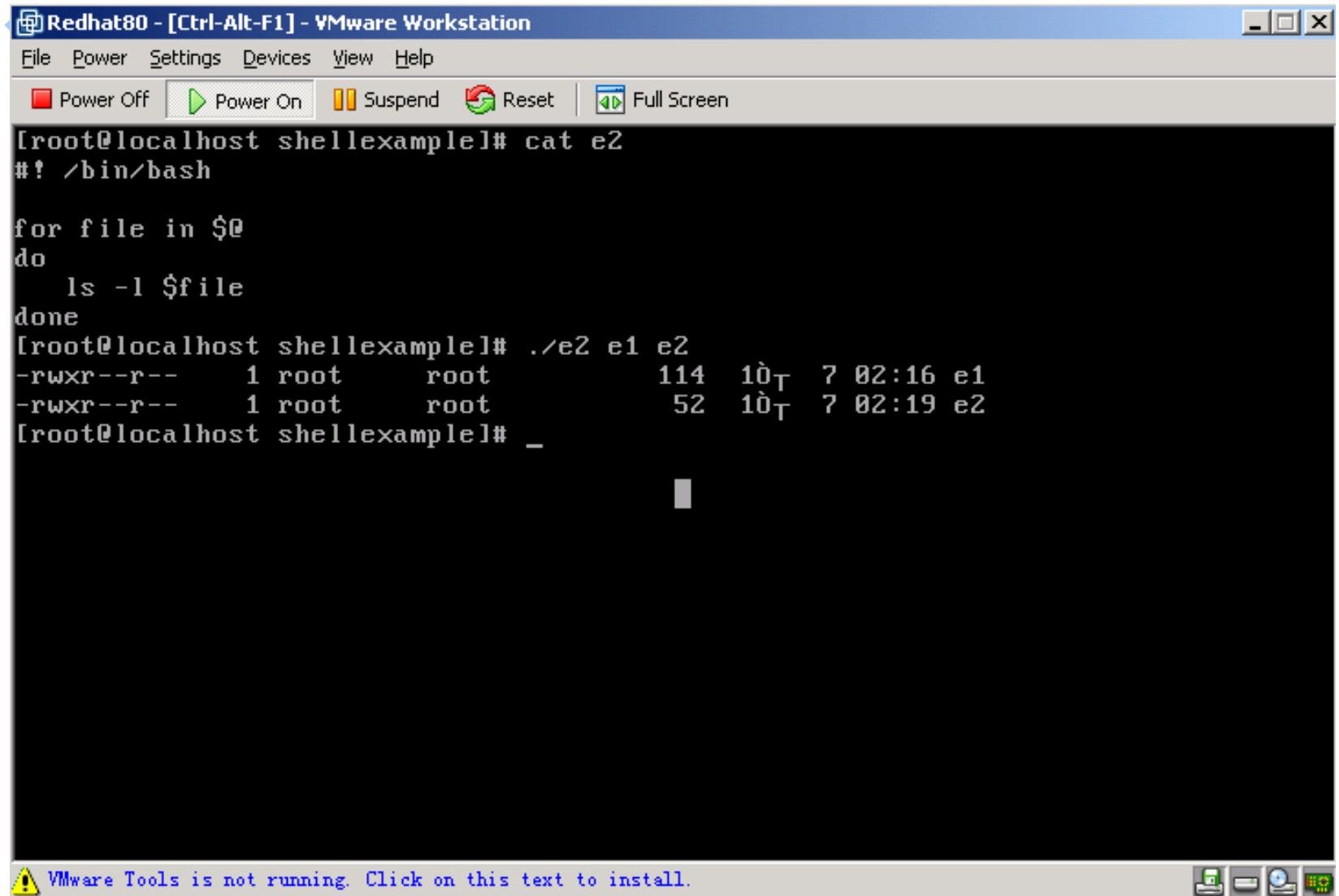


The screenshot shows a terminal window titled "Redhat80 - [Ctrl-Alt-F1] - VMware Workstation". The terminal is running a bash shell. The user enters the command `cat e1`, which displays the contents of a file named `e1`. The file contains several lines of text, including `#!/bin/bash`, `echo "There are $# parameters."`, `echo "The parameters are * ${*} *"`, and `echo "The parameters are * $@ *"`. The user then runs `./e1 one two three four`, which outputs the results of these echo statements, demonstrating how `$#`, `${*}`, and `$@` expand to the arguments passed to the script.

```
[root@localhost shellexample]# cat e1
#!/bin/bash
echo "There are $# parameters."
echo "The parameters are * ${*} *"
echo "The parameters are * $@ *"
[root@localhost shellexample]# ./e1 one two three four
There are 4 parameters.
The parameters are * one two three four *
The parameters are * one two three four *
[root@localhost shellexample]# _
```

At the bottom of the window, a message states: "VMware Tools is not running. Click on this text to install." with a yellow warning icon.

# \$\*和\$@的用法



The screenshot shows a terminal window titled "Redhat80 - [Ctrl-Alt-F1] - VMware Workstation". The terminal displays the following commands and output:

```
[root@localhost shellexample1]# cat e2
#!/bin/bash

for file in $@
do
    ls -l $file
done
[root@localhost shellexample1]# ./e2 e1 e2
-rwxr--r--  1 root    root      114  10_T  7 02:16 e1
-rwxr--r--  1 root    root       52  10_T  7 02:19 e2
[root@localhost shellexample1]# _
```

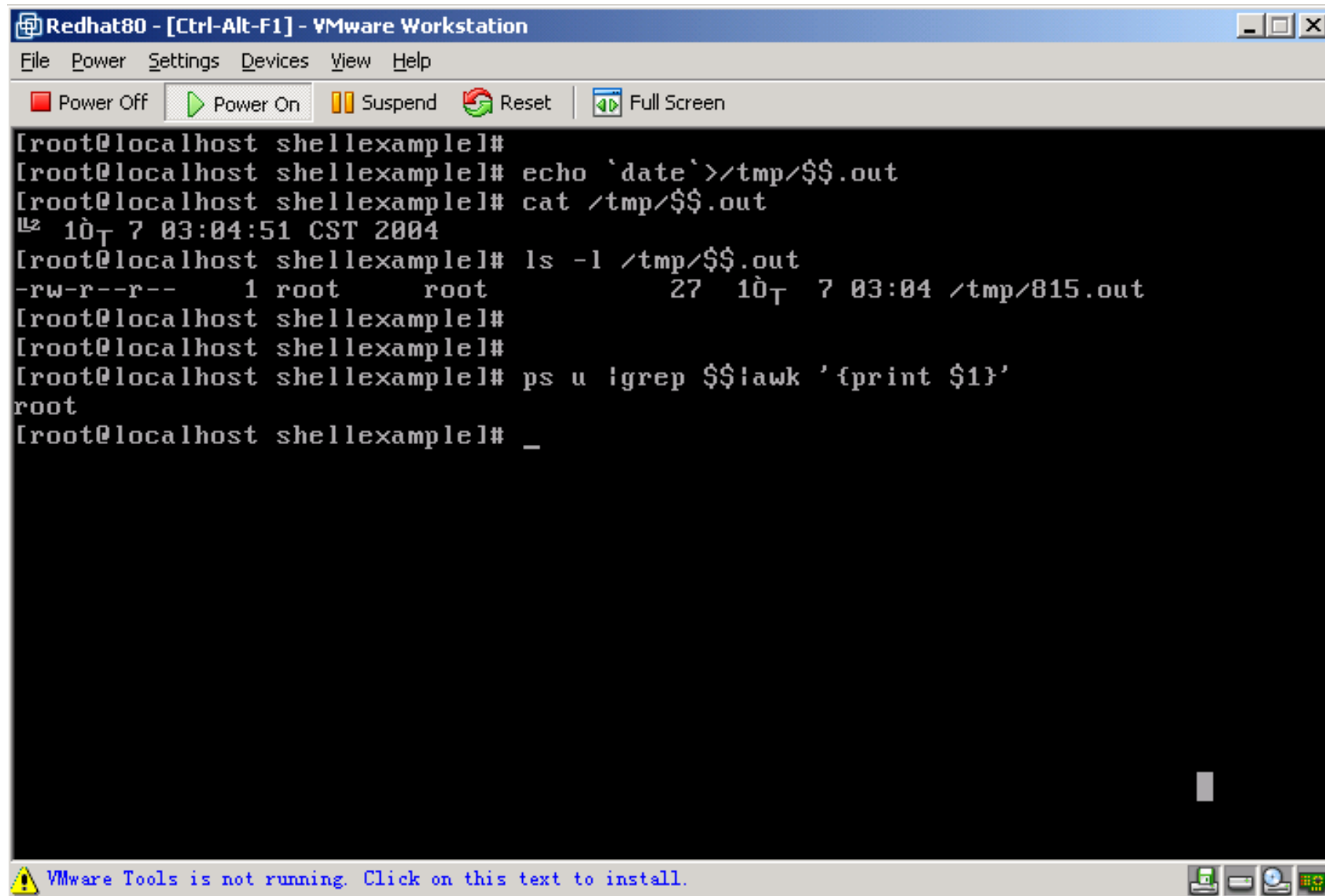
At the bottom of the window, there is a message: "VMware Tools is not running. Click on this text to install." and a row of icons.

# \$\$的用法

- ❖ \$\$变量用于保留当前进程的id号。使用它可以做以下事情：
- ❖ 1、创建唯一的文件名。
- ❖ 2、确定哪一个用户正在运行当前进程。
- ❖ 3、访问/**proc**目录结构，了解有关当前进程的信息。
- ❖ 注意：/**proc**目录是一个伪文件系统，用作与核心数据结构的接口。通常只有系统管理员才需要知道/**proc**目录的细节。



# \$\$的用法



Redhat80 - [Ctrl-Alt-F1] - VMware Workstation

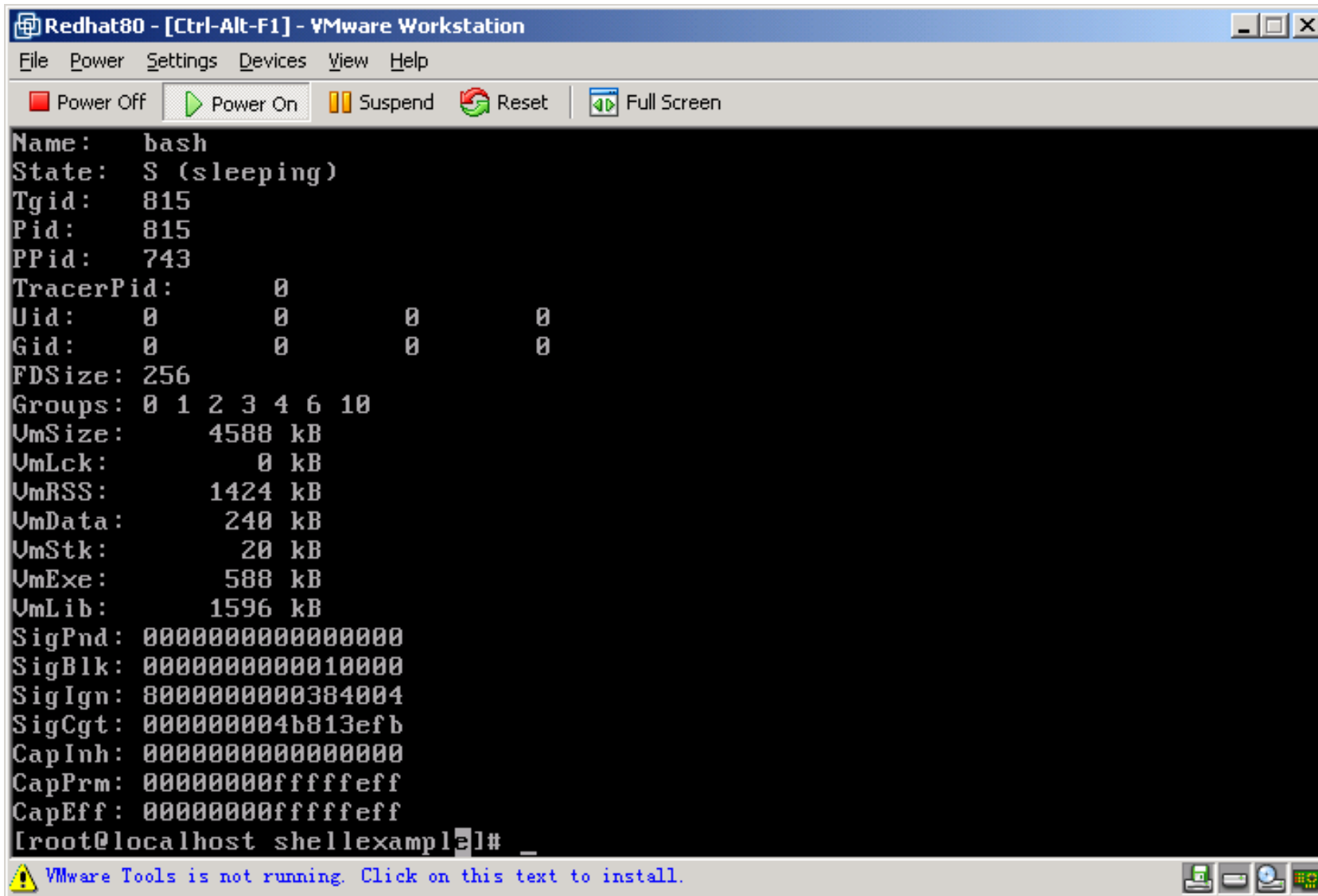
File Power Settings Devices View Help

Power Off Power On Suspend Reset Full Screen

```
[root@localhost shellexample]#  
[root@localhost shellexample]# echo `date`>/tmp/$$.out  
[root@localhost shellexample]# cat /tmp/$$.out  
Tue 10 7 03:04:51 CST 2004  
[root@localhost shellexample]# ls -l /tmp/$$.out  
-rw-r--r-- 1 root root 27 10 7 03:04 /tmp/815.out  
[root@localhost shellexample]#  
[root@localhost shellexample]#  
[root@localhost shellexample]# ps u |grep $$|awk '{print $1}'  
root  
[root@localhost shellexample]# _
```

VMware Tools is not running. Click on this text to install.

# \$\$的用法



```
Redhat80 - [Ctrl-Alt-F1] - VMware Workstation
File Power Settings Devices View Help
Power Off Power On Suspend Reset Full Screen

Name: bash
State: S (sleeping)
Tgid: 815
Pid: 815
PPid: 743
TracerPid: 0
Uid: 0 0 0 0
Gid: 0 0 0 0
FDSize: 256
Groups: 0 1 2 3 4 6 10
VmSize: 4588 kB
VmLck: 0 kB
VmRSS: 1424 kB
VmData: 240 kB
VmStk: 20 kB
VmExe: 588 kB
VmLib: 1596 kB
SigPnd: 0000000000000000
SigBlk: 0000000000010000
SigIgn: 8000000000384004
SigCgt: 000000004b813efb
CapInh: 0000000000000000
CapPrm: 00000000fffffeff
CapEff: 00000000fffffeff
[root@localhost shellexample]#
```

! VMware Tools is not running. Click on this text to install.

# 变量替换

- ❖ 如果你需要给变量添加一个后缀时，就需要用花括号把变量名括起来。
- ❖ `$word="large"`
- ❖ `$echo "He was a ${word}r man."`
- ❖ `He was a larger man.`
- ❖ 变量名前后的花括号告诉shell，变量名从何开始，到何结束。

# Shell的基本结构及观念

- ❖ **Script**是以行为单位，我们所写的**Shell**命令会被分解成一行一行来执行。而每一行可以是命令、注解、或是流程控制指令等。如果某一行尚未完成，可以在行末加上“\”，这个时候下一行的内容就会接到这一行的后面，成为同一行，如下：
- ❖ `echo The message is \`
- ❖ `too long so we have \`  
`to split it into \`  
`several lines`

# 变量替换

**Shell**在遇到未设置的变量时，将其值作为空串处理。而在实际应用中，对于未设置的变量，用户可以根据需要采用不同的处理方式，这可通过变量替换来实现。

变量替换提供了三种功能：

- ☆ 允许替换未设置变量的隐含值；
- 🕒 允许对未设置变量赋值；
- 🕒 在访问未设置变量时，提示出错信息。

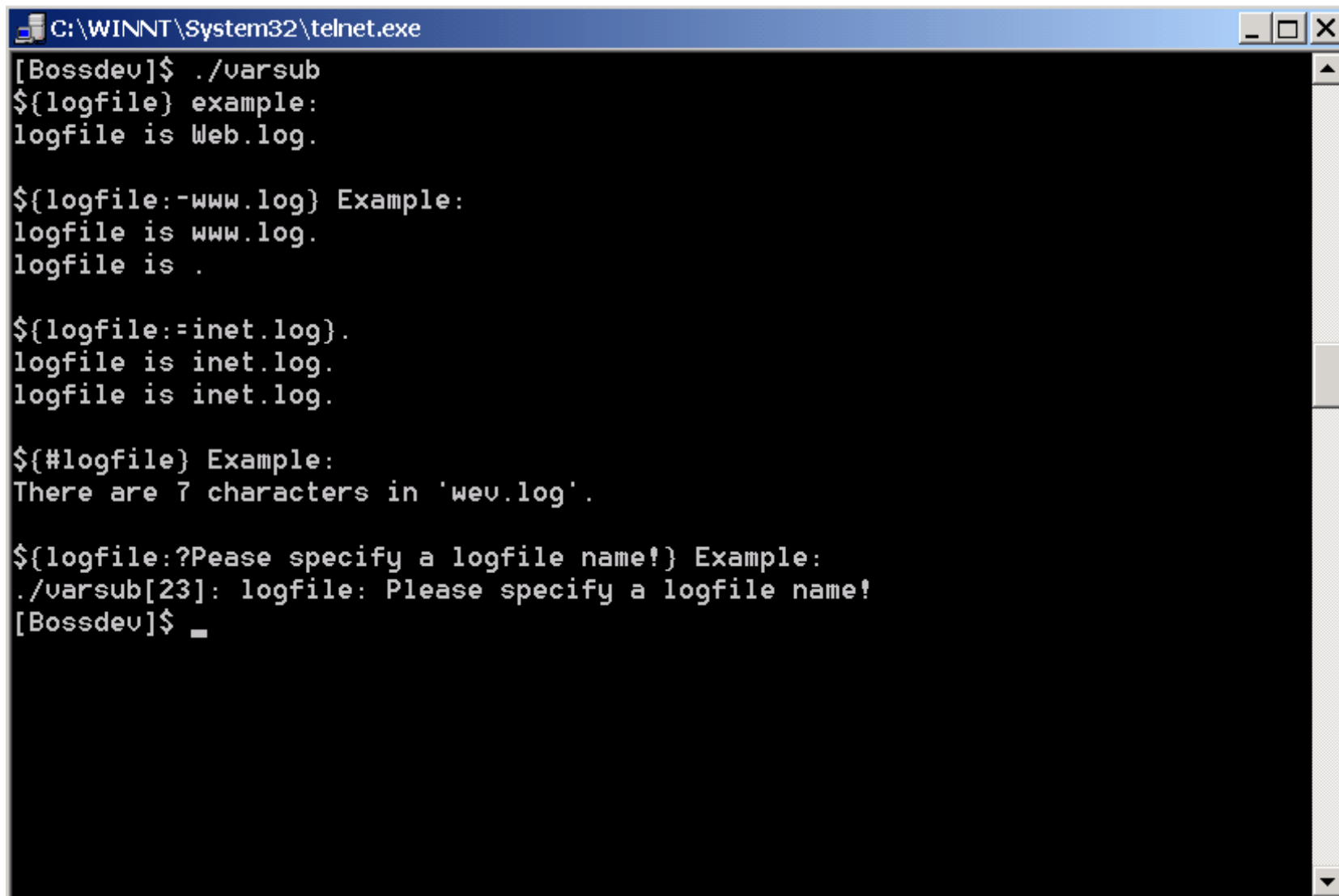
# 变量替换

- ❖ `${VARIABLE}` 基本变量替换。
- ❖ `${VARIABLE:-DEFAULT}` 如果VARIABLE没有值，则这种表示形式返回DEFAULT到值。
- ❖ `${VARIABLE:= DEFAULT}` 如果VARIABLE没有值，则这种表示形式返回DEFAULT到值。另外，如果VARIABLE没有设置，则把DEFAULT的值赋予它。
- ❖ `${VARIABLE:+VALUE}` 如果VARIABLE被设置，则这种表示形式返回VALUE；否则，返回一个空串。
- ❖ `${#VARIABLE}` 返回VARIABLE值得长度。如果VARIABLE为\*或@，则返回\$@所表示元素的个数。
- ❖ `${VARIABLE:?MESSAGE}` 如果VARIABLE没有值，则这种表示形式返回MESSAGE的值。Shell将显示VARIABLE的名字。

# 变量替换

```
C:\WINNT\System32\telnet.exe
[Bossdev]$ cat varsub
#!/usr/bin/sh
logfile="Web.log"
echo "\${logfile} example:"
echo "logfile is ${logfile}."
echo
unset logfile
echo "\${logfile:-www.log} Example:"
echo "logfile is ${logfile:-www.log}."
echo "logfile is $logfile."
echo
unset logfile
echo "\${logfile:=inet.log}."
echo "logfile is ${logfile:=inet.log}."
echo "logfile is $logfile."
echo
unset logfile
logfile="wev.log"
echo "\${#logfile} Example:"
echo "There are ${#logfile} characters in '$logfile'."
echo
unset logfile
echo "\${logfile:?Pease specify a logfile name!} Example:"
echo "logfile is ${logfile:?Please specify a logfile name!}"
echo "logfile is $logfile."
[Bossdev]$
```

# 变量替换



A screenshot of a Windows telnet window titled "C:\WINNT\System32\telnet.exe". The window shows a series of commands and their outputs demonstrating variable substitution. The prompt is "[Bossdev]\$". The commands and outputs are as follows:

```
[Bossdev]$ ./varsub
${logfile} example:
logfile is Web.log.

${logfile:-www.log} Example:
logfile is www.log.
logfile is .

${logfile:=inet.log}.
logfile is inet.log.
logfile is inet.log.

${#logfile} Example:
There are 7 characters in 'web.log'.

${logfile:?Please specify a logfile name!} Example:
./varsub[23]: logfile: Please specify a logfile name!
[Bossdev]$ _
```



# 数组变量

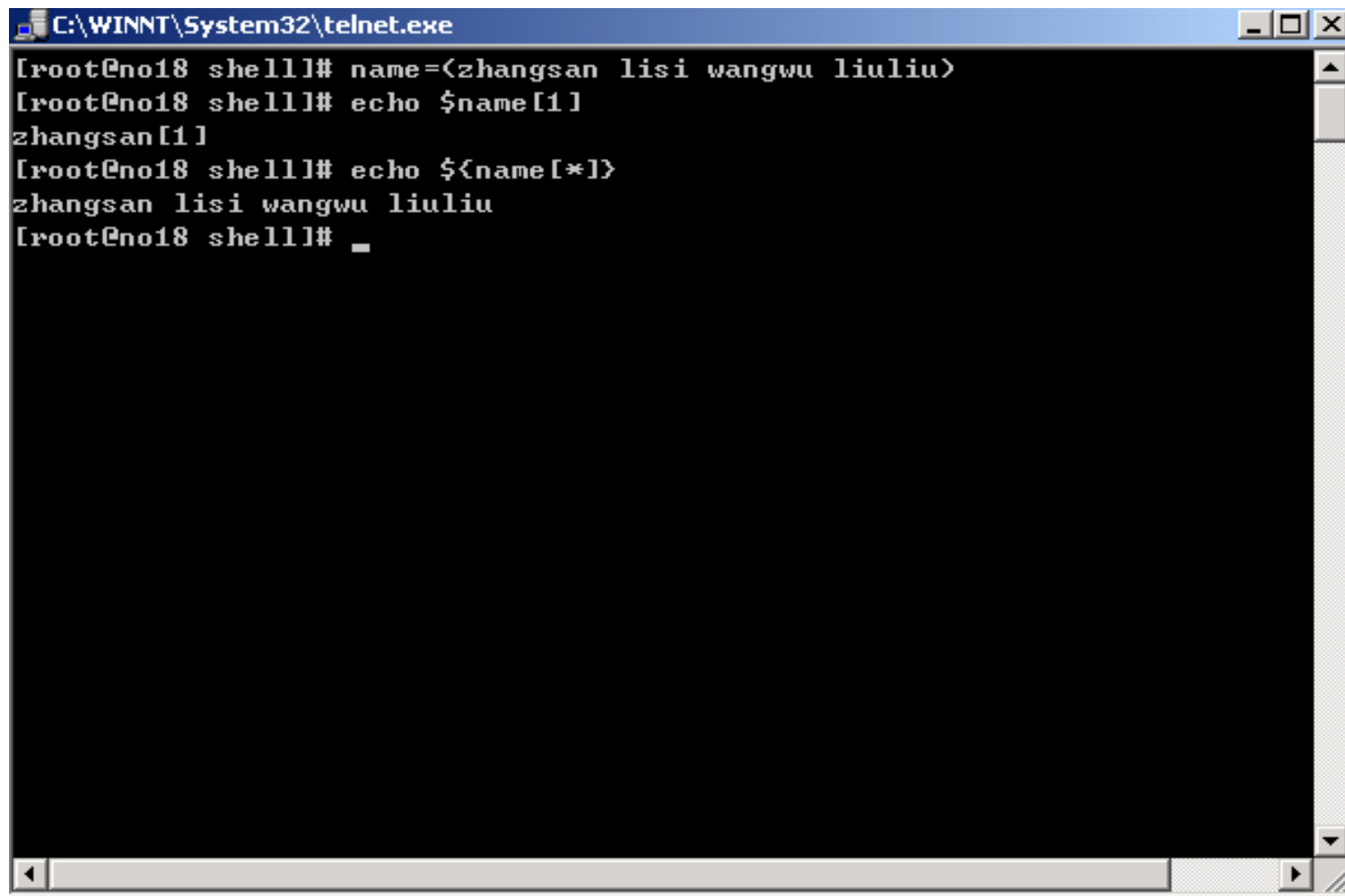
- ❖ 在Korn Shell(ksh)中，扩展了变量的功能，支持数组变量。
- ❖ 数组提供了一种将变量集合分组的方法。程序员不是为每个所要求的变量创建一个新名字，而是使用一个数组变量存储所有其它变量。
- ❖ 创建数组变量最简单的方法就是为数组中的每一个元素赋值。
- ❖ `name[index]=value`
- ❖ 数组初始化的第二种方式是一次设置多个元素。
- ❖ `set -A name value1 value2 value3 ... valuen`
- ❖ 在bash中，多个元素的设置方式如下：
- ❖ `name=(value1 value2 ...valuen)`
- ❖ ksh和bash都是使用从0开始的连续数组下标。

# 访问数组值

- ❖ 在为数组赋值后，可以对它们进行访问：
- ❖ `${name[index]}`
- ❖ 可以使用下列方式访问数组中的所有项：
- ❖ `${name[*]}`
- ❖ `${name[@]}`
- ❖ 如果数组中某一项所存储的值带有空格，则这种格式的数组访问方式就不能正常工作，需要使用第二种格式。

# 数组变量

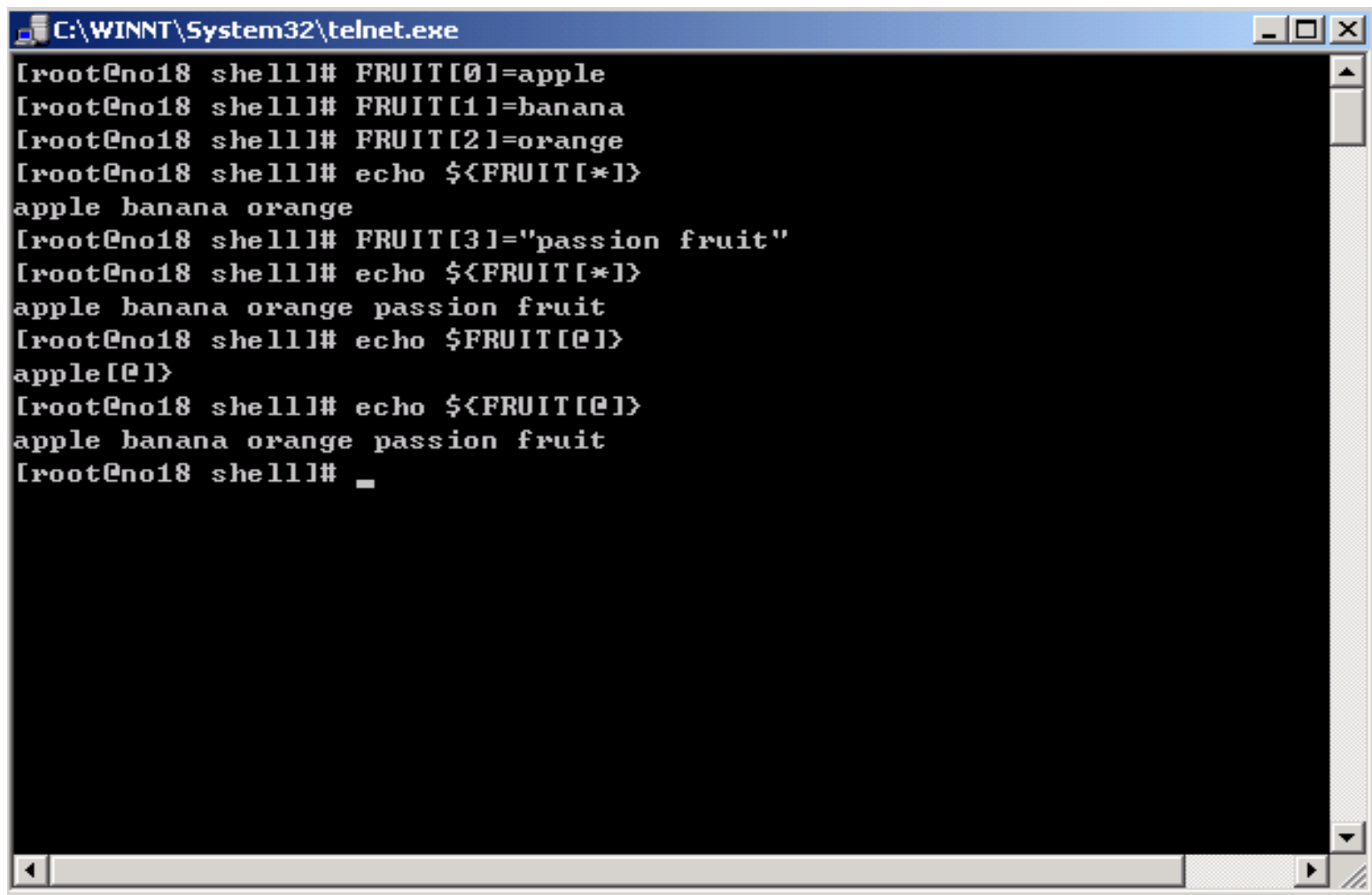
❖ 例：

A screenshot of a Windows command prompt window titled "C:\WINNT\System32\telnet.exe". The window shows a telnet session with a root user on a host named "no18". The user enters a command to assign an array of names to a variable named "name". Then, they use "echo" to print the second element of the array and the entire array. The output shows "zhangsan" for the second element and the full list of names for the entire array.

```
[root@no18 shell]# name=(zhangsan lisi wangwu liuliu)
[root@no18 shell]# echo ${name[1]}
zhangsan[1]
[root@no18 shell]# echo ${name[*]}
zhangsan lisi wangwu liuliu
[root@no18 shell]# _
```

# 访问数组值

❖ 例:

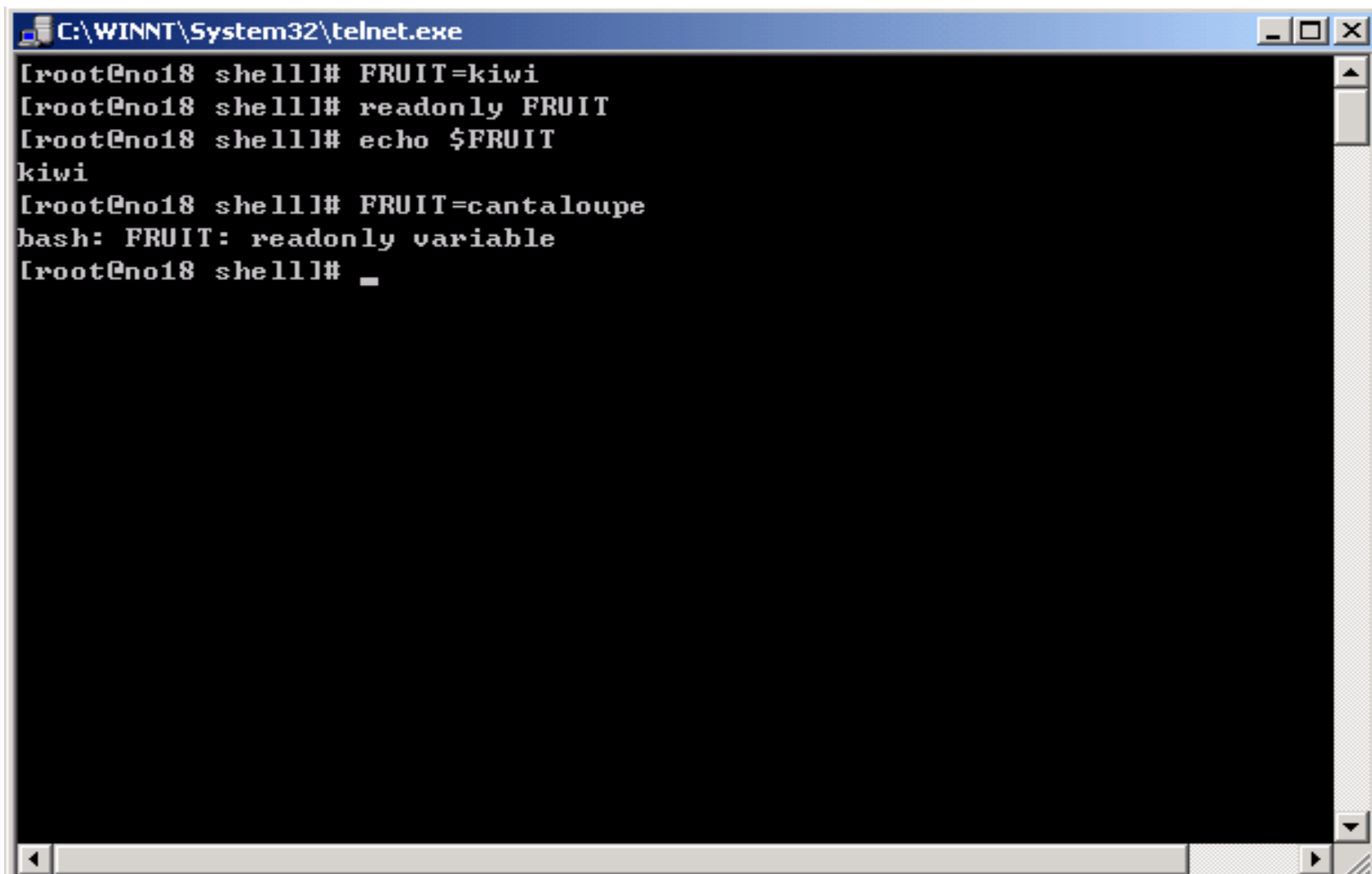


```
C:\WINNT\System32\telnet.exe
[root@no18 shell]# FRUIT[0]=apple
[root@no18 shell]# FRUIT[1]=banana
[root@no18 shell]# FRUIT[2]=orange
[root@no18 shell]# echo ${FRUIT[*]}
apple banana orange
[root@no18 shell]# FRUIT[3]="passion fruit"
[root@no18 shell]# echo ${FRUIT[*]}
apple banana orange passion fruit
[root@no18 shell]# echo ${FRUIT[@]}
apple[@]
[root@no18 shell]# echo ${FRUIT[@]}
apple banana orange passion fruit
[root@no18 shell]# _
```

# 只读变量

- ❖ 通过使用**readonly**命令，**Shell**提供了一种将变量标记为只读方式，当命令被标为只读时，它的值不可改变。
- ❖ 注意：该特点可以在脚本中保持关键变量不会被意外地覆盖掉。

# 只读变量

A screenshot of a Windows command prompt window titled "C:\WINNT\System32\telnet.exe". The window has a black background with white text. The text shows a series of commands and their outputs in a telnet session. The commands are: "FRUIT=kiwi", "readonly FRUIT", "echo \$FRUIT", "FRUIT=cantaloupe", and an empty line. The outputs are: "kiwi" and "bash: FRUIT: readonly variable". The prompt is "[root@no18 shell]#".

```
C:\WINNT\System32\telnet.exe
[root@no18 shell]# FRUIT=kiwi
[root@no18 shell]# readonly FRUIT
[root@no18 shell]# echo $FRUIT
kiwi
[root@no18 shell]# FRUIT=cantaloupe
bash: FRUIT: readonly variable
[root@no18 shell]#
```

# 执行Shell命令

- ❖ 在**Bourne Shell**中有五种方法执行一个**UNIX**命令，而这五种方式所产生的结果稍微有些不同。
- ❖ 1. 直接下命令
- ❖ 这个方式和在命令列中直接下命令的效果一样。
- ❖ 2. 使用**sh**命令
- ❖ **sh command**
- ❖ 这个文件必须是**Bourne Shell**的**Script**，但这个文件并不一定要设成可执行。除此之外和直接下命令的方式一样。

# 执行Shell命令

## ❖ 3. 使用 “.”命令

### ❖ . command

❖ 这时和使用sh命令相似，只不过它不像sh一般会产生新的process，相反地，它会在原有的process下完成工作。

## ❖ 4. 使用exec命令

### ❖ exec command

❖ 此时这个Script将会被所执行的命令所取代。当这个命令执行完毕之后，这个Script也会随之结束。

## ❖ 5. 使用命令替换

❖ 这是一个相当有用的方法。如果想要使某个命令的输出成为另一个命令的参数时，就一定要使用这个方法。我们将命令列於两个“`”号之间，而Shell会以这个命令执行后的输出结果代替这个命令以及两个“`”符号。



# 执行Shell命令

❖ `str='Current directory is `pwd`'`  
`echo $str`

结果如下：

Current directory is /users/cc/mgtsai 这个意思是 `pwd` 这个命令输出 “/users/cc/mgtsai”，而后整个字符串代替原来的 ``pwd`` 设定 `str` 变量，所以 `str` 变量的内容则会有 `pwd` 命令的输出。

❖ `number=`expr $number + 1``

这就是先前所提要作数值运算的方法，基本上 `expr` 命令只将运算式解，而后输出到标准输出上。如果要将某变量设定成其值，非得靠命令替换的方式不可。这个例子是将 `number` 变量的值加1 后再存回 `number` 变量。

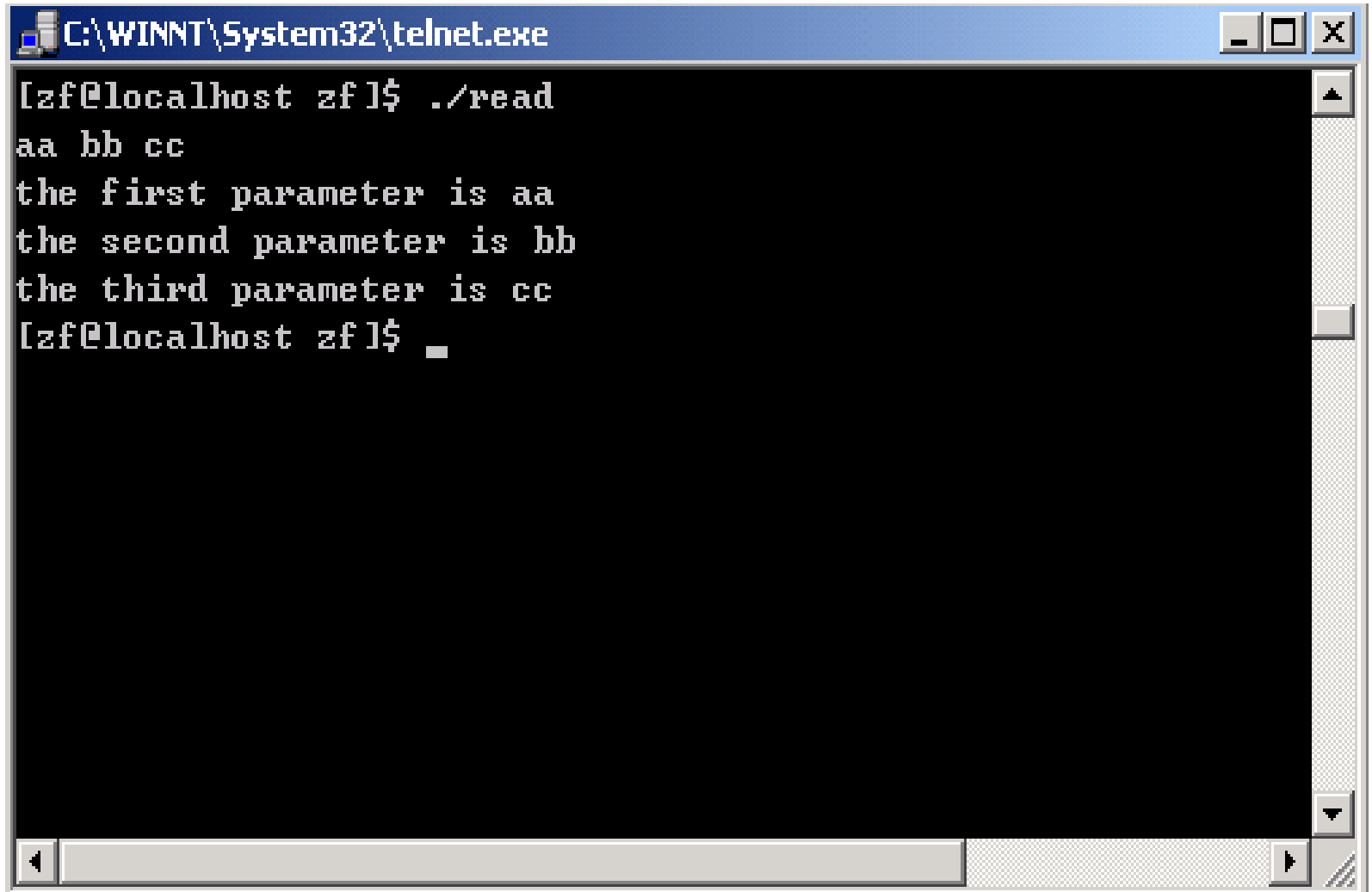
# Shell命令

- ❖ **Read命令**：从键盘读入数据，赋给变量
- ❖ 如：`read var1 var2 var3`
- ❖ 这时**read**会将一个字分给一个变量。如果输入的字比变量还多，最后一个变量会将剩下的字当成其值。如果输入的字比变量还少，则后面的变量会设成空字符串。

# Read命令

- ❖ Read 的例子:
- ❖ `#!/bin/sh`
- ❖ `read first second third`
- ❖ `echo "the first parameter is $first"`
- ❖ `echo "the second parameter is $second"`
- ❖ `echo "the third parameter is $third"`

# Read命令



```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./read
aa bb cc
the first parameter is aa
the second parameter is bb
the third parameter is cc
[zf@localhost zf]$ _
```

# Read命令



C:\WINNT\System32\telnet.exe

```
[Bossdev]$ cat read_02
#!/usr/bin/sh
echo "Currently logged in users are:"
who
echo
echo
echo "Enter the name of the user to whom you want to talk"
read NAME
echo "Initiating talk with $NAME"
talk $NAME
[Bossdev]$ _
```

# expr命令

- ❖ Shell提供了五种基本的算术运算符：
- ❖ +、-、\\*、/、%
- ❖ Shell变量通常按字符进行存放。为了进行整数的算术运算，必须使用**expr**命令，格式是：
- ❖ `expr n1 op n2`

# expr命令

- ❖ Shell变量的算术运算：
- ❖ expr命令：对整数型变量进行算术运算
- ❖ 如：  
`expr 3 + 5`  
`expr $var1 - 5`  
`expr $var1 / $var2`  
`expr $var3 \* 10`

注意：在运算符的前后必须留有空格。

# expr命令

- ❖ 多个算术表达式可以组合在一起：
- ❖ `expr 6 + 8 / 3`
- ❖ 运算符的优先级：
- ❖ 如果 要改变计算顺序，必须使用倒引号：
- ❖ `expr `expr 6 + 8` / 3`



# Shell命令

## ❖ Expr2程序的例子:

❖ `#!/bin/sh`

❖ `a=10`

❖ `b=20`

❖ `c=30`

❖ `value1=`expr $a + $b + $c``

❖ `echo "The value of value1 is $value1"`

❖ `value2=`expr $c / $b``

❖ `echo "The value of value2 is $value2"`

❖ `value3=`expr $c \* $b``


❖ `echo "The value of value3 is $value3"`

❖ `value4=`expr $a + $c / $b``

❖ `echo "The value of value4 is $value4"`

# Shell命令



 C:\WINNT\System32\telnet.exe

```
The value of value1 is 60  
The value of value2 is 1  
The value of value3 is 600  
The value of value4 is 11  
[Bossdev]$ _
```

# 复杂的Shell命令

❖ 复杂的运算:

**expr `expr 5+7` / \$var4**

❖ 将运算结果赋予变量:

**\$var4=`expr \$var1 / \$var2`**

# 条件与 test 命令

## ☞ 简单条件

在高级语言中判断条件依赖于运算的结果，而Shell语言依赖条件是命令执行的“出口状态”。

Shell命令的“出口状态”（\$?）：

成功：0 、 true

失败：x 、 false （x 为非0数值）

例：判断指定目录是否存在， 并显示相应信息。 # cat  
checkdir

```
test -d $1 && echo "$1 is a dictory"&& exit 0
```

```
echo "$1 is not a dictroy"
```

```
exit 1
```

# 条件与 test 命令（续1）

## 👉 test 命令

test 命令可用于对字符串、整数及文件进行各类测试。其命令格式如下：

**test expression**

或 **[ expression ]**      **注意** [ ] 中的空格)

**expression** 是测试的条件，计算结果：

为真，则返回“零”出口状态，

为假，否则返回“非零”出口状态。

例：判断当前上机用户人数是否多于10？

```
# test "`who | wc -l`" -gt 10
```

```
# echo $?
```

# 条件与 test 命令（续2）


## **test** 字符串测试表达式

expression	满足下列条件时返回真值
string1 = string2	string1与string2相同
string1 != string2	string1与string2不相同
string	string不为空串
-n string	string不为空串
-z string	string为空串

# 变量测试语句

- ❖ 假设变量**var**的值是“**hello**”:
- ❖ `$var = "hello"`
- ❖ `$var != "gello"`
- ❖ `$var != "hello"`

# 变量测试语句



C:\WINNT\System32\telnet.exe

```
[Bossdev]$ STR1=Hello  
[Bossdev]$ test $STR1 = Hello  
[Bossdev]$ echo $?  
0  
[Bossdev]$ test $STR1 != Hello  
[Bossdev]$ echo $?  
1  
[Bossdev]$
```



# 条件与 test 命令（续3）

例1：两个字符串进行比较

```
# user=smith  
# test "$user" = smith  
# echo $?  
0
```

例2：查找指定的文件或目录

```
# cat search  
test "$1" || { echo "err: no parameter" ; \  
               exit 1; }  
find . -name "$1" -print
```

## 条件与 test 命令（续4）

例2：带有空格的字符串比较

```
# month="January "  
# test "$month" = January  
# echo $?  
1  
# test $month = January  
# echo $?  
0
```

区别：**Shell**在处理变量时，遇到有双引号将保留其内容，而省略双引号时，将滤去空格。

# 条件与 test 命令（续7）

例3：带有空格的字符串比较

```
# a="testing string"
# test "$a" = "testing string"
# echo $?
0
# test $a = "testing string"
test: unknown operator string
```

Shell处理变量 `$a` 时，将其进行变量替换，然后将结果(`testing string`)传递给`test`，而`test`将`string`作为操作符来处理，因此出错。

## 条件与 test 命令（续8）

例4：带有空串（或未设置的字符串比较）

```
# name= " "  
# test "$name" = smith  
# echo $?  
1  
# test $name = smith  
test: argument expected
```

Shell处理变量"\$name"时，双引号将其括起的内容作为一个“位置持有者”来保留，并把该值传递给test，保证处理的正常执行。

# 条件与 test 命令（续9）

例4：带有空串的字符串比较

```
# blanks=" "  
# test $blanks  
# echo $?  
1  
# test "$blanks"  
# echo $?  
0
```

Shell处理变量\$blanks时，将空格滤去，使其变为空串传递给test；而双引号保留“位置持有者”的位置，其值为一个空格（空白符），传递给test。

# 条件与 test 命令（续10）

例4：带有算符的字符串比较

```
# symvar=="
```

```
# test -z "$symvar"
```

```
test: argument expected
```

出错的原因是 “=”运算符比 “-z”运算符的优先级要高，因此，**test** 命令期望在等号之后要有一个自变量。为避免上述问题的发生，可用下面命令形式替换：

```
# test x"$symvar" = x
```

```
# echo $?
```

```
1
```

# 条件与 test 命令（续11）

## ☞ **test**命令可用于整数比较

首先要搞清楚整数比较的两个概念：

☆ **Shell**并不区分放在**Shell**变量中的值的类型，就变量本身而言，它存放的仅仅是一组字符串，既**Shell**只有一种类型的变量——串变量。

🕒 当使用整数比较操作符时，是**test**命令来解释存放在变量中的整数值，而不是**Shell**。

# 条件与 test 命令（续12）

## **test** 整数测试表达式

expression	满足下列条件时返回真值
int1 -eq int2	两者为数值且int1等于int2
int1 -ge int2	两者为数值且int1大于或等于int2
int1 -gt int2	两者为数值且int1大于int2
int1 -le int2	两者为数值且int1小于或等于int2
int1 -lt int2	两者为数值且int1小于int2
int1 -ne int2	两者为数值且int1不等于int2



# 条件与 test 命令（续13）

例：

```
# x1= " 005 "  
# x2= " 10"  
# test " $x1 " = 5  
# echo $?
```

⇐ 按串方式比较

1

```
# test " $x1 " -eq 5  
# echo $?
```

⇐ 按数值方式比较

0

```
# test " $x2 " -eq 10  
# echo $?
```

0

# 条件与 test 命令（续14）

## ☞ test 中常用的文件测试表达式

expression	满足下列条件时返回真值
-r FileName	FileName存在且为用户可读
-w FileName	FileName存在且为用户可写
-x FileName	FileName存在且为用户可执行
-s FileName	FileName存在且其长度大于0
-d FileName	FileName为一个目录
-f FileName	FileName为一个普通文件

# 条件与 test 命令（续15）

例1：检查指定的文件是否存在并且可读

```
test -f /usr/fk/message
```

例2：检查指定的文件是否为目录

```
test -d /usr/src/local/sendmail
```

例3：检查指定的出错文件是否为空，如不空  
则列出该文件的内容。

```
test -s $errfile && cat $errfile
```

# 条件与 test 命令（续16）

## ☞ 表达式的逻辑运算

逻辑运算符包括：

**!** — 逻辑非单目运算符，可放置在任何其它 **test** 表达式之前，求得表达式运算结果得非值。

**-a** — 逻辑与运算符，执行两个表达式的逻辑与运算，并且仅当两者都为真时，才返回真值。

**-o** — 逻辑或运算符，执行两个表达式的逻辑或运算，并仅当两者之一为真时，就返回真值。

# 条件与 test 命令（续17）

## 👉 逻辑运算符的优先级

☆ 逻辑运算符优先级(由高到低) 的排列顺序如下：

( )    ↑    !    ↑    -a    ↑    -o

🕒 逻辑运算符优先级要比字符串操作符、数字比较操作符、文件操作符的优先级低。

# 条件与 test 命令（续18）

## ☞ 表达式的逻辑组合

expression	满足下列条件时返回真值
! expr	expr返回值为假( Not )
expr1 -a expr2	expr1和expr2同时为真 ( And )
expr1 -o expr2	expr1为真或expr2为真 ( Or )
\(expr\)	expr为真时 [注意]左右括号前要加转义符 \

# 条件与 test 命令（续19）

例1：当指定的文件不可读时为真。

```
test ! -r /usr/fk/message
```

例2：当指定的文件均存在，且message为可读、

\$mailfile 指定的文件为普通文件时，返回真。

```
test -r /usr/fk/message -a -f "$mailfile "
```

例3：当变量值大于等于0并且小于10时为真。

```
test "$count " -ge 0 -a "$count " -lt 10
```

例4：

```
test \( "$a" -eq 0 -o "$b" -gt 5 \) -a "$c" -le 8
```

# if 结构

## 👉 if 的简单结构

格式    **if command1**  
          **then**  
          **command2**  
          **command3**  
          **...**  
          **fi**



# if 结构（续1）

## 👉 if 的完整结构

格式

```
if command1
then
    command2
    command3
    ...
else
    command4
    command5
    ...
fi
```

# if 结构（续2）

## ☞ if 的连用结构

格式1

```
if command1
then
    commands
else
    if command2
    then
        commands
        :
    :
fi
fi
```

# if 结构（续3）

## 👉 if 的连用结构

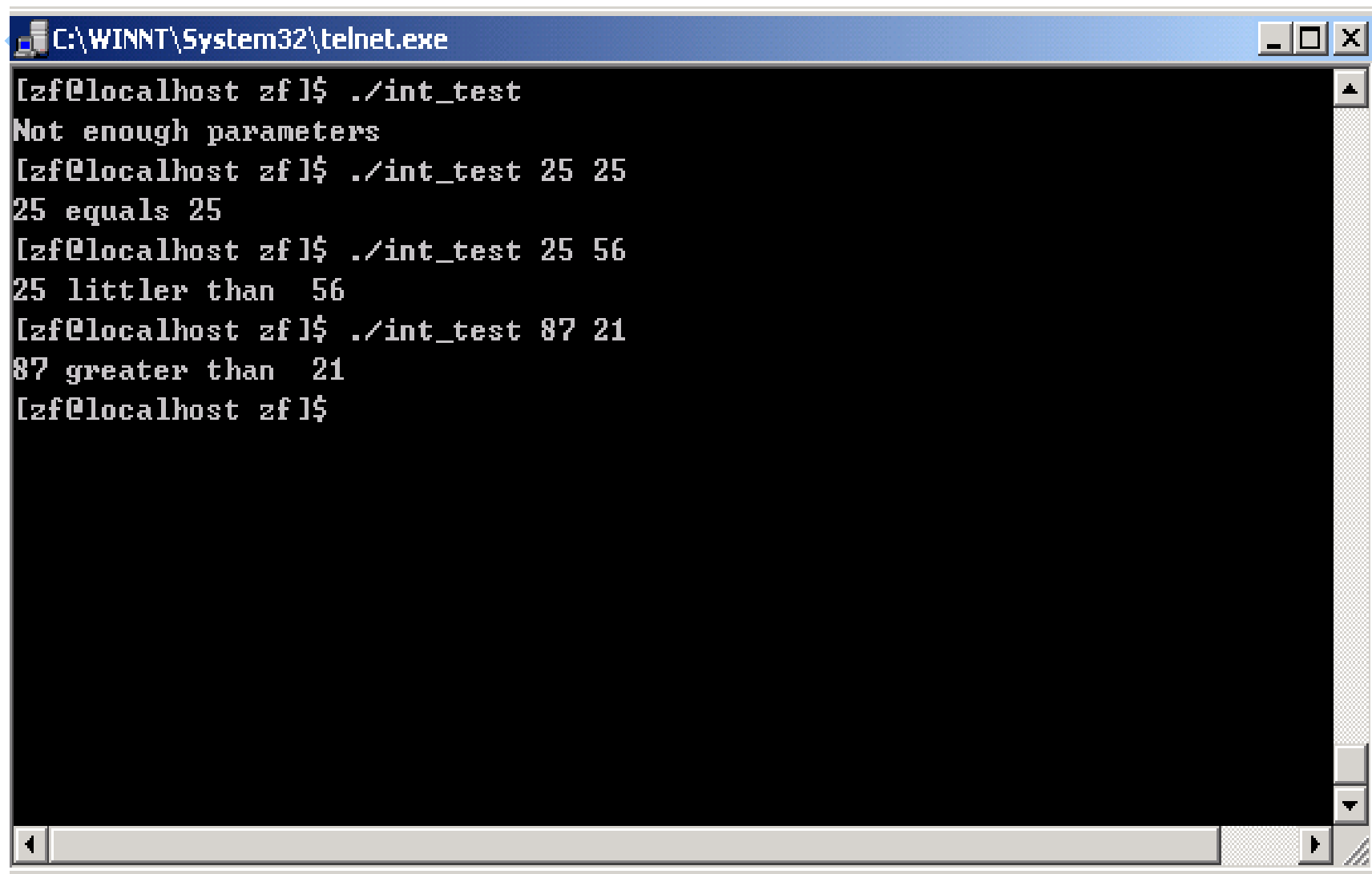
格式2

```
if command1
then
    commands
elif command2
then
    commands
    :
    :
    commands
else
    commands
fi
```

# 变量测试语句

- ❖ `#!/bin/sh`
- ❖ `if [ $# -ne 2 ]; then`
- ❖ `echo "Not enough parameters"`
- ❖ `exit 0`
- ❖ `fi`
- ❖ `if [ $1 -eq $2 ]; then`
- ❖ `echo "$1 equals $2"`
- ❖ `elif [ $1 -lt $2 ]; then`
- ❖ `echo "$1 littler than $2"`
- ❖ `elif [ $1 -gt $2 ]; then`
- ❖ `echo "$1 greater than $2"`
- ❖ `fi`

# 变量测试语句



```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./int_test
Not enough parameters
[zf@localhost zf]$ ./int_test 25 25
25 equals 25
[zf@localhost zf]$ ./int_test 25 56
25 littler than 56
[zf@localhost zf]$ ./int_test 87 21
87 greater than 21
[zf@localhost zf]$
```

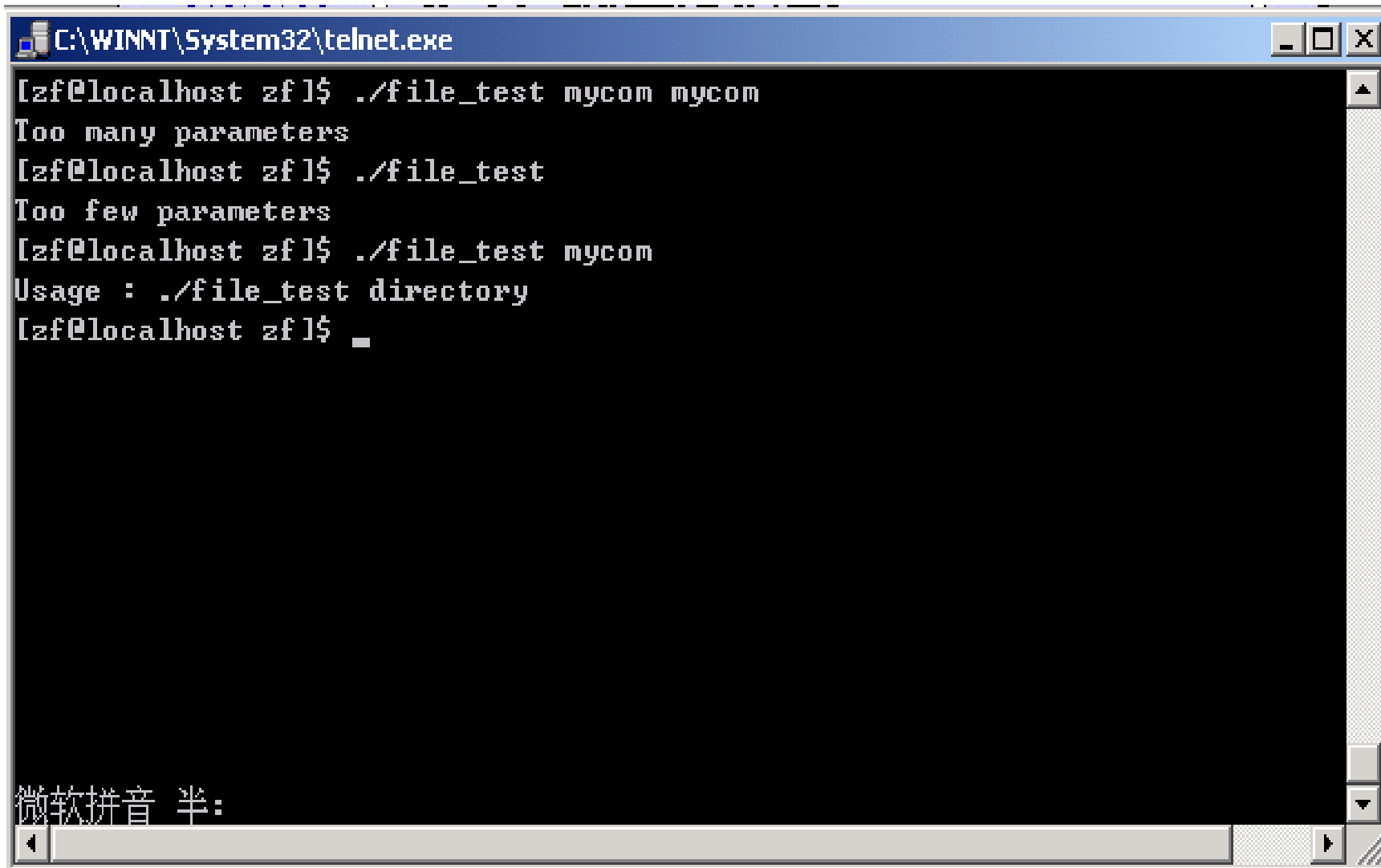
# 文件控制语句

```
❖ #! /bin/bash
❖
❖ if [ $# -gt 1 ]
❖ then
❖     echo "Too many parameters"
❖     exit 1
❖ fi
❖
❖ if [ $# -eq 0 ]
❖ then
❖     echo "Too few parameters"
❖     exit 1
```

# 文件控制语句

```
❖ fi
❖ if [ ! -d $1 ]
❖ then
❖     echo "Usage : $0 directory"
❖     exit 1
❖ fi
❖
❖ for i in $1/*
❖ do
❖     if [ -x $i -a ! -d $i ]
❖     then
❖         ls $i
❖     fi
❖ done
```

# 文件控制语句



```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./file_test mycom mycom
Too many parameters
[zf@localhost zf]$ ./file_test
Too few parameters
[zf@localhost zf]$ ./file_test mycom
Usage : ./file_test directory
[zf@localhost zf]$ _
```

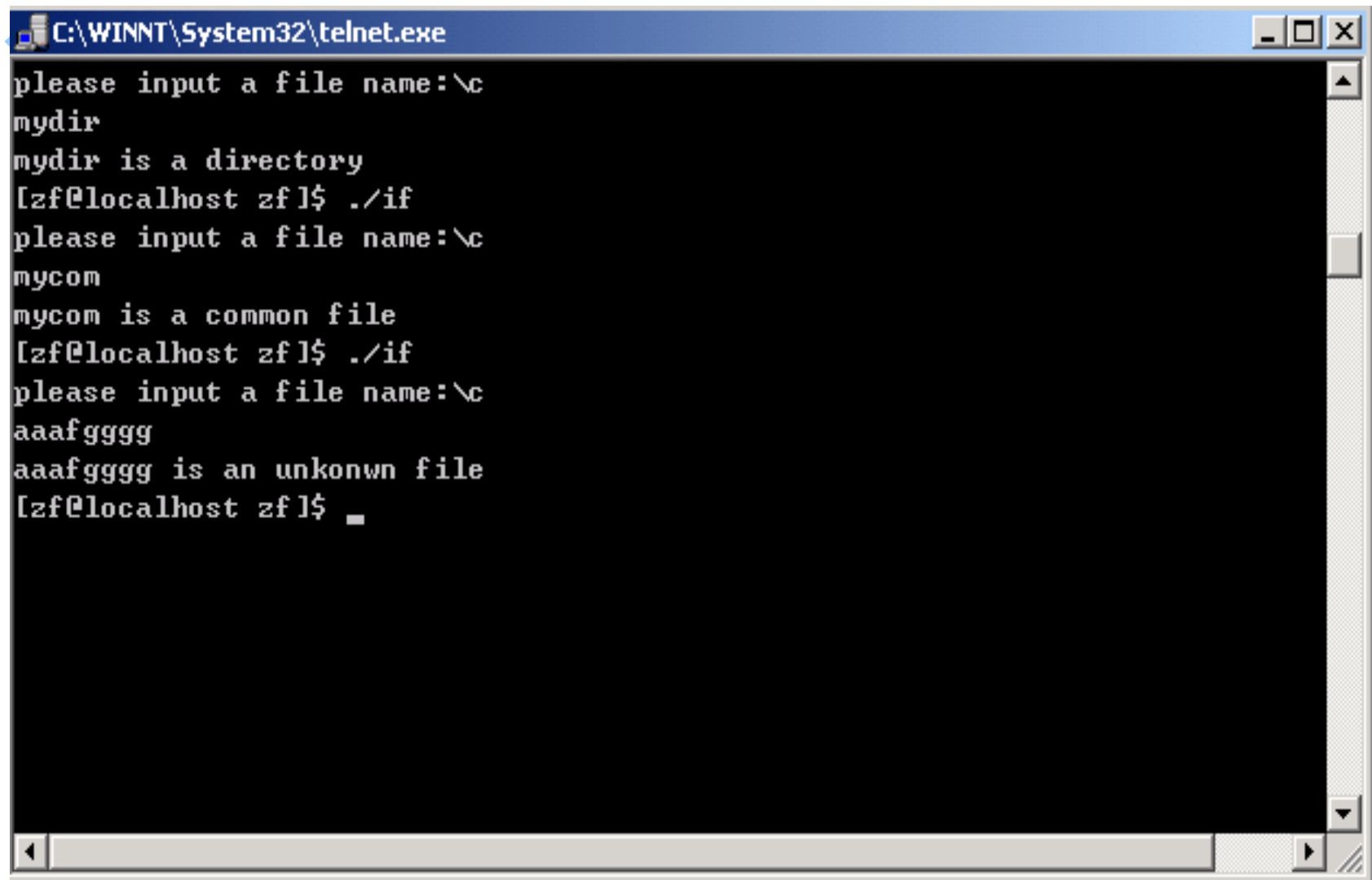
微软拼音 半:



# 流控制语句

- ❖ echo "please input a file name:\c"
- ❖ read file\_name
- ❖ if [ -d \$file\_name ]
- ❖ then
- ❖   echo "\$file\_name is a directory"
- ❖ elif [ -f \$file\_name ]
- ❖ then
- ❖   echo "\$file\_name is a common file"
- ❖ elif [ -c \$file\_name -o -b \$file\_name ]
- ❖ then
- ❖   echo "\$file\_name is a device file"
- ❖ else
- ❖   echo "\$file\_name is an unkonwn file"
- ❖ fi

# 流控制语句



```
C:\WINNT\System32\telnet.exe
please input a file name:\c
mydir
mydir is a directory
[zf@localhost zf]$ ./if
please input a file name:\c
mycom
mycom is a common file
[zf@localhost zf]$ ./if
please input a file name:\c
aaafgggg
aaafgggg is an unkonwn file
[zf@localhost zf]$ _
```

# case 结构

格式

```
case value in  
    pattern1) command11  
    ...  
    command1n;;  
    pattern2) command21  
    ...  
    command2n;;  
    ...  
    patternn) commandn1  
    ...  
    commandnn;;  
esac
```

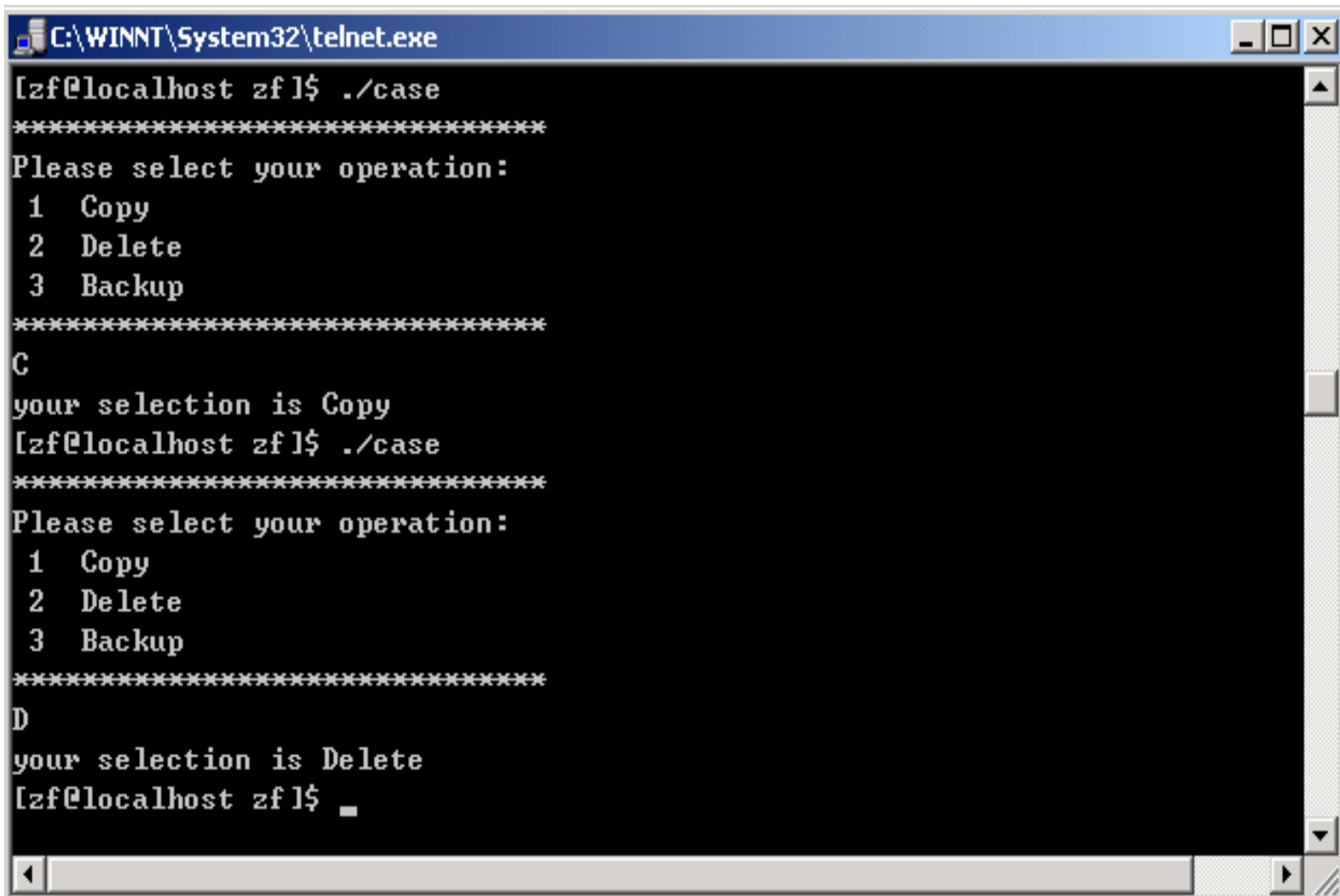
# 流控制语句

- ❖ `#!/bin/sh`
- ❖ `echo "*****"`
- ❖ `echo "Please select your operation:"`
- ❖ `echo " 1 Copy"`
- ❖ `echo " 2 Delete"`
- ❖ `echo " 3 Backup"`
- ❖ `echo "*****"`
- ❖ `read op`

# 流控制语句

```
❖ case $op in
❖     C)
❖         echo "your selection is Copy"
❖         ;;
❖     D)
❖         echo "your selection is Delete"
❖         ;;
❖     B)
❖         echo "your selection is Backup"
❖         ;;
❖     *)
❖         echo "invalid selection"
❖ esac
```

# 流控制语句

A screenshot of a Windows telnet session window. The title bar reads 'C:\WINNT\System32\telnet.exe'. The window contains a text-based menu program. The user enters './case' at the prompt '[zf@localhost zf]\$. The program displays a menu with three options: 1 Copy, 2 Delete, and 3 Backup. The user enters 'C', and the program responds 'your selection is Copy'. The user enters './case' again, and the menu is displayed. The user enters 'D', and the program responds 'your selection is Delete'. The prompt returns to '[zf@localhost zf]\$.

```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./case
*****
Please select your operation:
 1  Copy
 2  Delete
 3  Backup
*****
C
your selection is Copy
[zf@localhost zf]$ ./case
*****
Please select your operation:
 1  Copy
 2  Delete
 3  Backup
*****
D
your selection is Delete
[zf@localhost zf]$
```

# for 结构

格式

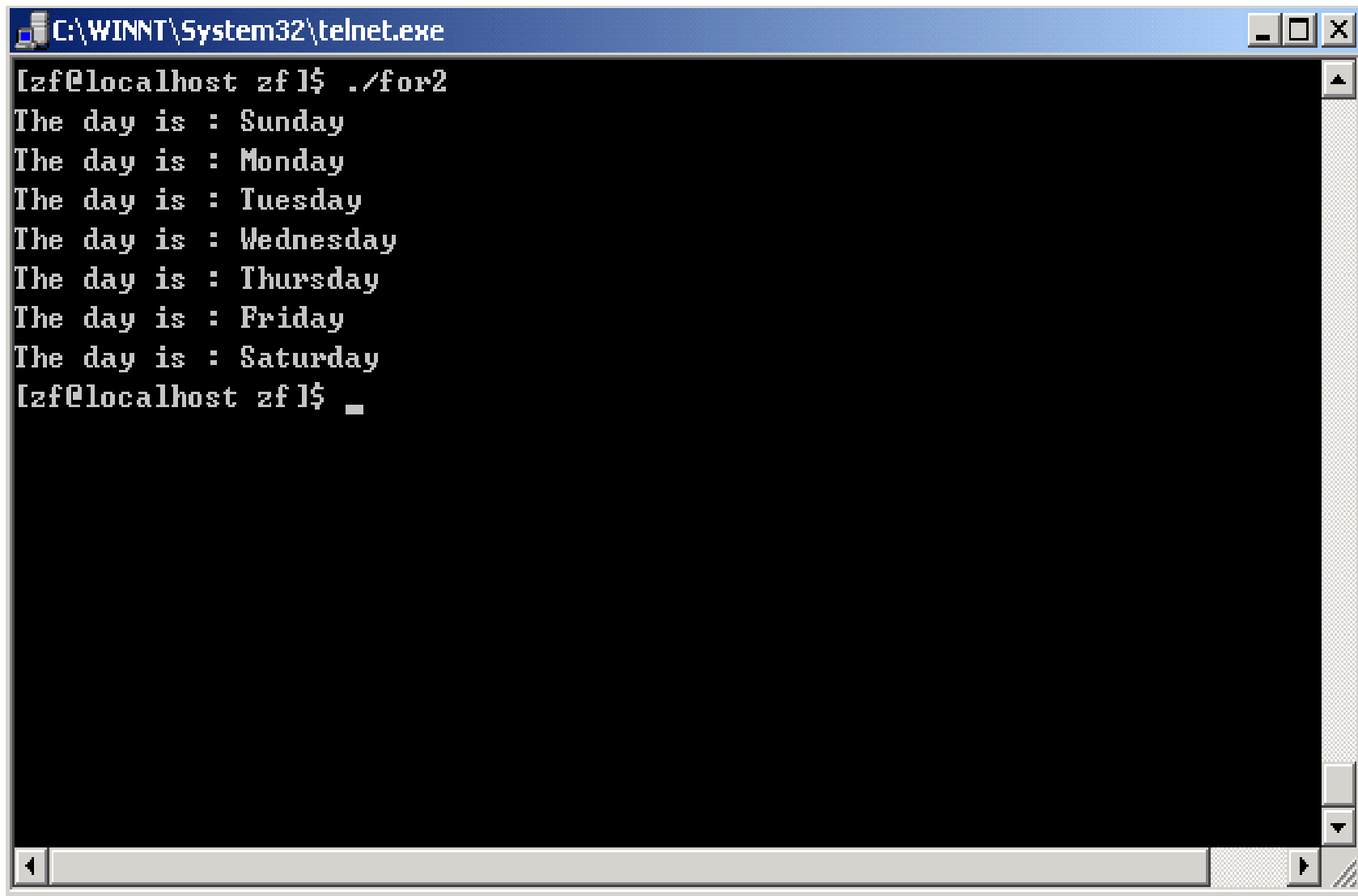
```
for variable in arg1 arg2 ... argn  
do  
    command  
    ...  
    command  
done
```

# 流控制语句

- ❖ 例子:
- ❖ `#!/bin/bash`
- ❖ `for DAY in Sunday Monday Tuesday  
Wednesday Thursday Friday Saturday`
- ❖ `do`
- ❖ `echo "The day is : $DAY"`
- ❖ `done`



# 流控制语句



```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$. /for2
The day is : Sunday
The day is : Monday
The day is : Tuesday
The day is : Wednesday
The day is : Thursday
The day is : Friday
The day is : Saturday
[zf@localhost zf]$ _
```

# 流控制语句

 C:\WINNT\System32\telnet.exe

```
[Bossdev]$ cat for2
#!/usr/bin/sh
echo "List of all executable files in home directory"
cd $HOME/test
for F in *
do
    if [ -x $F ]
    then
        ls -l $F
    fi
done
[Bossdev]$
```

# 流控制语句

C:\WINNT\System32\telnet.exe

[Bossdev]\$ ./for2

List of all executable files in home directory

-rwxr--r--	1	zhaofang	devgrp	170	1月	7日	15:12	assign_logfile
-rwxr--r--	1	zhaofang	devgrp	88	1月	7日	16:25	case_sensitive
-rwxr--r--	1	zhaofang	devgrp	670	1月	8日	15:10	copy

总数 8

-rwxr--r--	1	zhaofang	devgrp	180	1月	8日	15:15	example1
-rwxr--r--	1	zhaofang	devgrp	27	1月	7日	16:03	display_logfile
-rwxr--r--	1	zhaofang	devgrp	80	1月	7日	15:02	display_parameters
-rwxr--r--	1	zhaofang	devgrp	245	1月	7日	16:07	export_logfile
-rwxr--r--	1	zhaofang	devgrp	145	1月	8日	15:49	for2
-rwxr--r--	1	zhaofang	devgrp	180	1月	8日	14:47	read_02
-rwxr--r--	1	zhaofang	devgrp	280	1月	8日	14:23	script_oo
-rwxr--r--	1	zhaofang	devgrp	232	1月	8日	14:38	shift_00
-rwxr--r--	1	zhaofang	devgrp	63	1月	7日	16:35	testa
-rwxr--r--	1	zhaofang	devgrp	197	1月	8日	15:42	until

[Bossdev]\$

# while 结构

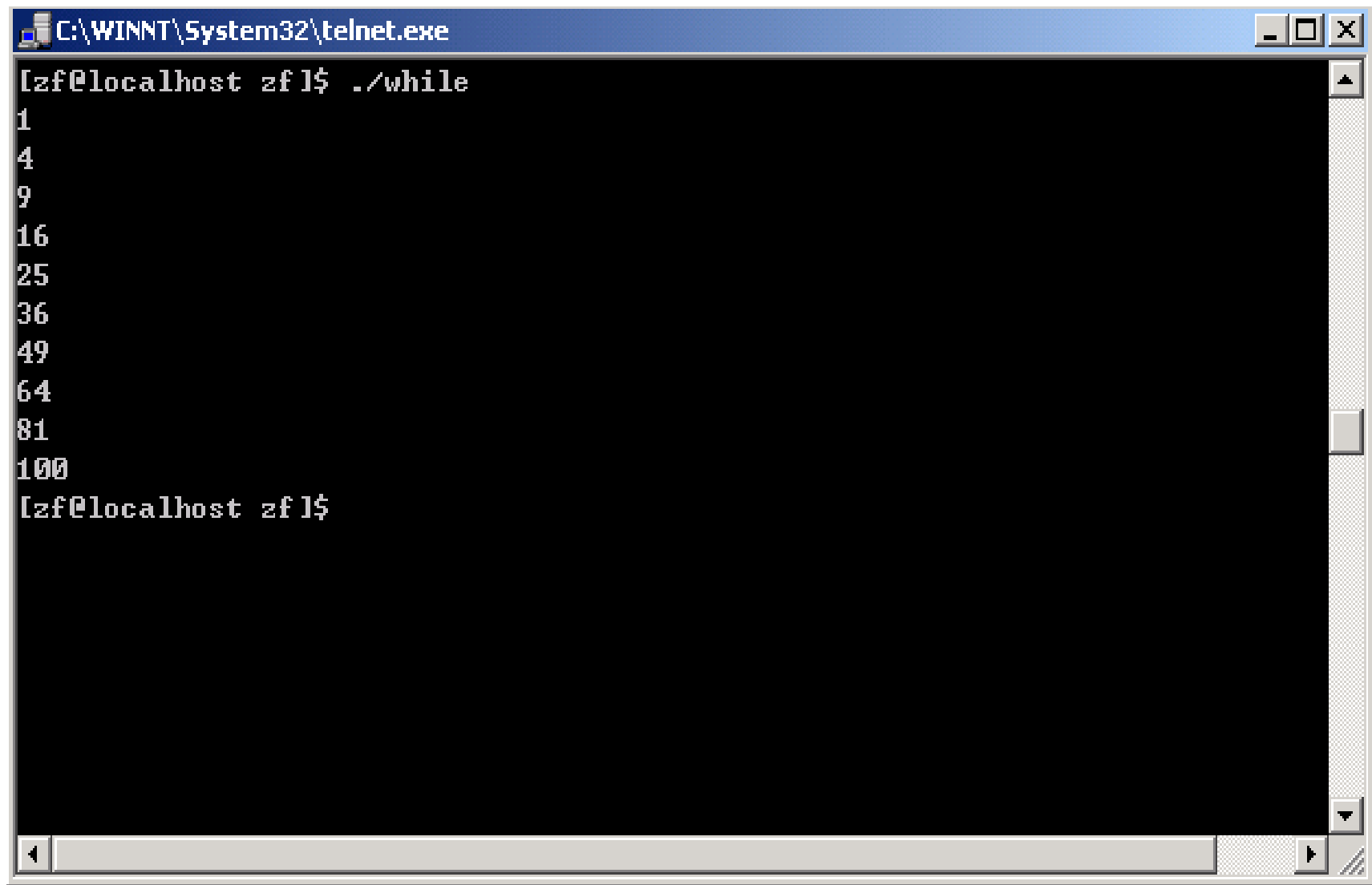
格式

```
while command  
do  
    command  
    ...  
    command  
done
```

# 流控制语句

- ❖ `#!/bin/sh`
- ❖ `num=1`
- ❖ `while [ $num -le 10 ]`
- ❖ `do`
- ❖ `square=`expr $num \* $num``
- ❖ `echo $square`
- ❖ `num=`expr $num + 1``
- ❖ `done`

# 流控制语句



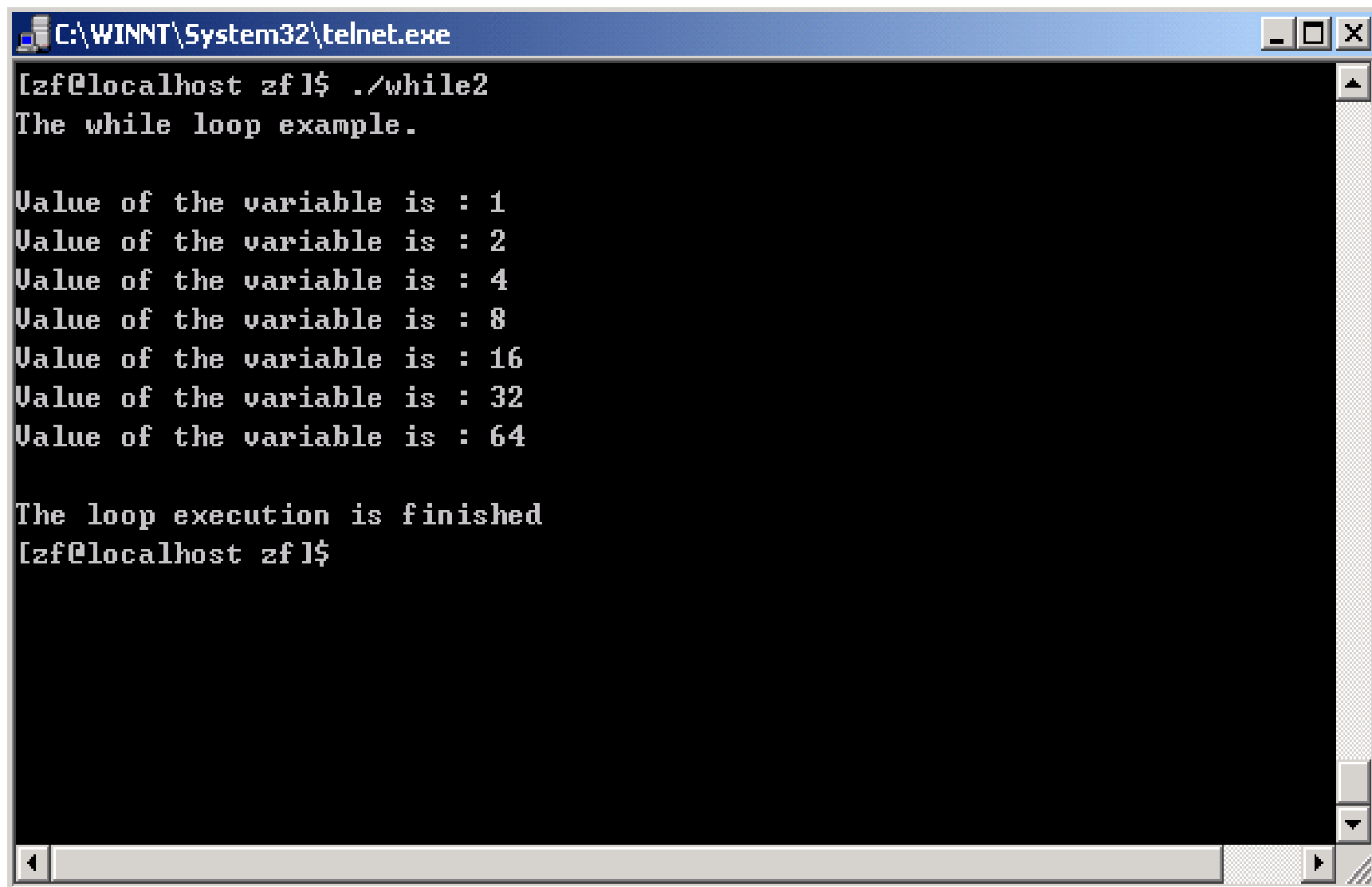
A screenshot of a Windows telnet window titled "C:\WINNT\System32\telnet.exe". The window has a black background with white text. The prompt is "[zf@localhost zf]\$". The user has entered the command "./while", which has started a loop printing the numbers 1, 4, 9, 16, 25, 36, 49, 64, 81, and 100. The prompt "[zf@localhost zf]\$" is visible again at the bottom, indicating the loop has finished. The window includes standard Windows controls (minimize, maximize, close) in the top right and a scrollbar on the right side.

```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./while
1
4
9
16
25
36
49
64
81
100
[zf@localhost zf]$
```

# 流控制语句

- ❖ echo "The while loop example."
- ❖ echo
- ❖ VAR1=1
- ❖ while ((VAR1<100))
- ❖ do
- ❖ echo "Value of the variable is : \$VAR1"
- ❖ ((VAR1=VAR1\*2))
- ❖ done
- ❖ echo
- ❖ echo "The loop execution is finished"

# 流控制语句



A screenshot of a Windows telnet window titled "C:\WINNT\System32\telnet.exe". The window has a black background with white text. The text shows a user at a [zf@localhost zf] prompt running the command `./while2`. The output indicates a while loop example where the value of a variable is printed in a series of lines: 1, 2, 4, 8, 16, 32, and 64. After the last line, it says "The loop execution is finished" and returns to the prompt. The window includes standard Windows window controls (minimize, maximize, close) in the top right and a scrollbar on the right side.

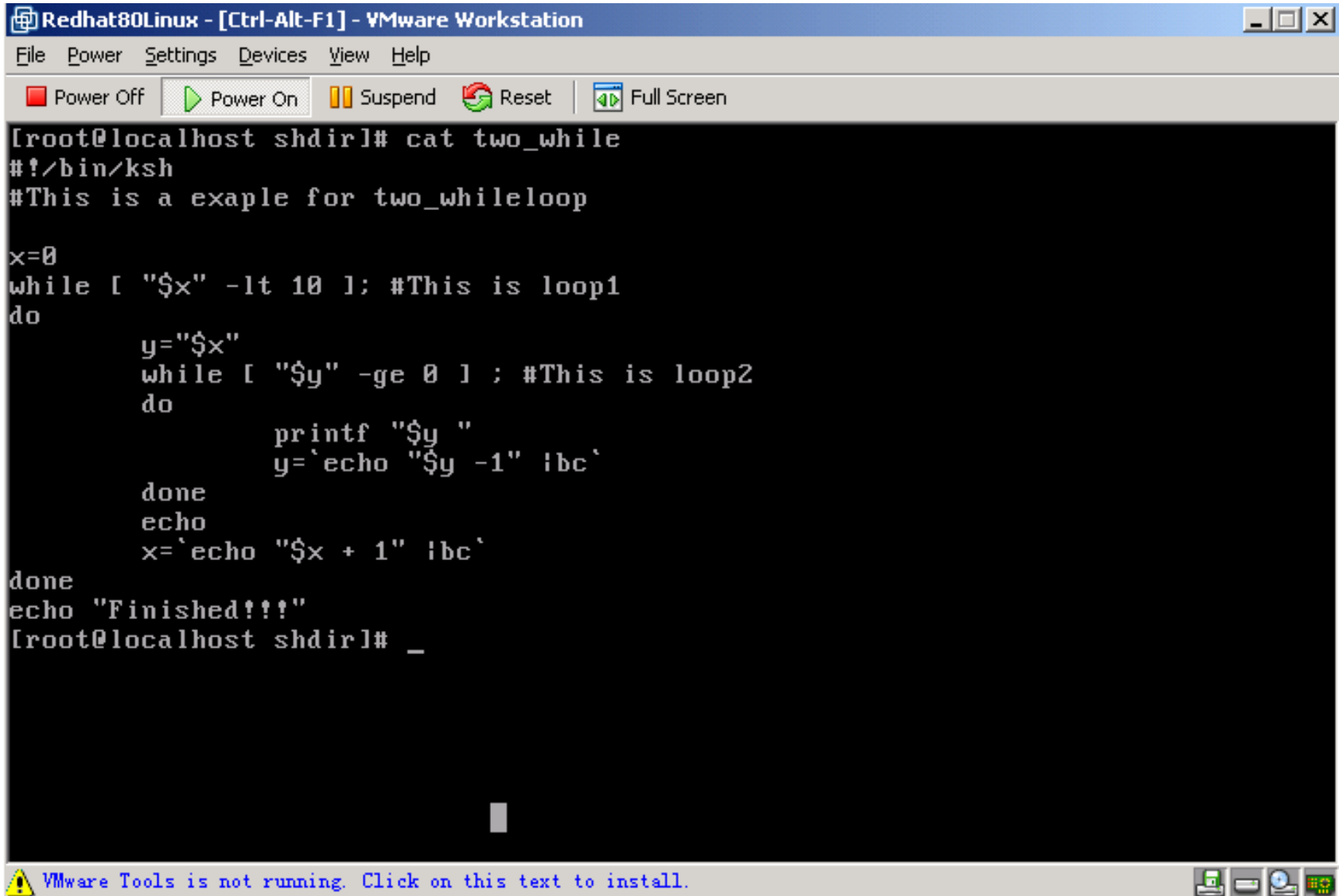
```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./while2
The while loop example.

Value of the variable is : 1
Value of the variable is : 2
Value of the variable is : 4
Value of the variable is : 8
Value of the variable is : 16
Value of the variable is : 32
Value of the variable is : 64

The loop execution is finished
[zf@localhost zf]$
```



# 流控制语句

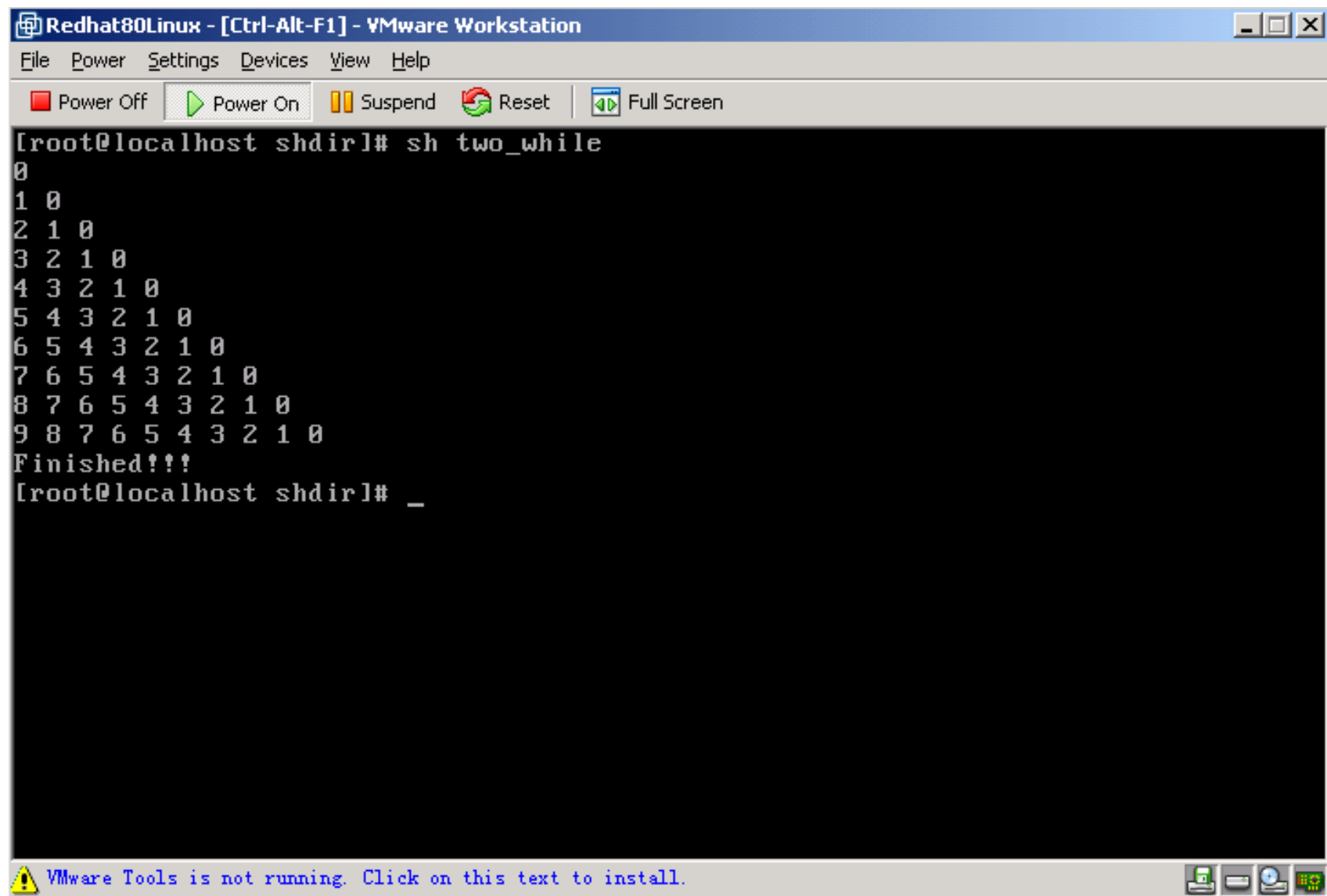


```
[root@localhost shdir]# cat two_while
#!/bin/ksh
#This is a exaple for two_whileloop

x=0
while [ "$x" -lt 10 ]; #This is loop1
do
    y="$x"
    while [ "$y" -ge 0 ]; #This is loop2
    do
        printf "$y "
        y=`echo "$y -1" |bc`
    done
    echo
    x=`echo "$x + 1" |bc`
done
echo "Finished!!!"
[root@localhost shdir]# _
```

! VMware Tools is not running. Click on this text to install.

# 流控制语句



```
Redhat80Linux - [Ctrl-Alt-F1] - VMware Workstation
File Power Settings Devices View Help
Power Off Power On Suspend Reset Full Screen

[root@localhost shdir]# sh two_while
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
6 5 4 3 2 1 0
7 6 5 4 3 2 1 0
8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
Finished!!!
[root@localhost shdir]# _
```

VMware Tools is not running. Click on this text to install.

# until 结构

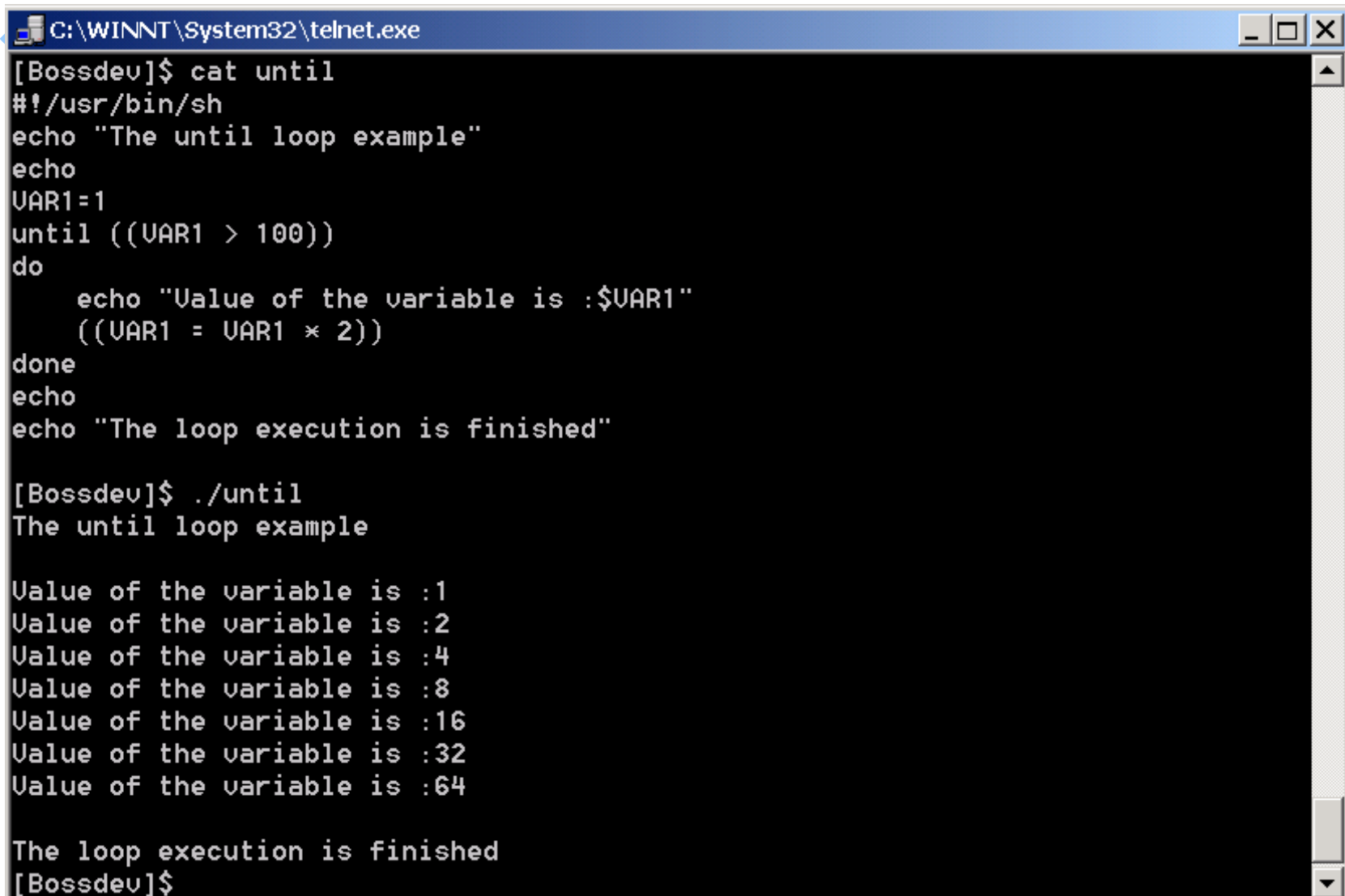
格式

```
until command  
do  
    command  
    ...  
    command  
done
```

# Until\_do\_done循环

- ❖ `until_do_done`循环和`while_do_done`循环类似。唯一不同的是只要测试条件保持为假时就继续执行。一旦条件为真，它就终止执行。
- ❖ `until condition`
- ❖ `do`
- ❖ `command block`
- ❖ `done`

# Until\_do\_done循环

A screenshot of a telnet window titled 'C:\WINNT\System32\telnet.exe'. The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The terminal text shows a user at the '[Bossdev]\$' prompt creating a file named 'until' with a shebang '#!/usr/bin/sh'. The file contains an 'until' loop that runs 'do' until the variable 'UAR1' is greater than 100. Inside the loop, it echoes the current value of 'UAR1' and doubles it. After the loop, it echoes a completion message. The user then runs './until', and the output shows the loop executing, with 'UAR1' values increasing from 1 to 64 in powers of 2. Finally, the loop completes and the user returns to the '[Bossdev]\$' prompt.

```
C:\WINNT\System32\telnet.exe
[Bossdev]$ cat until
#!/usr/bin/sh
echo "The until loop example"
echo
UAR1=1
until ((UAR1 > 100))
do
    echo "Value of the variable is :$UAR1"
    ((UAR1 = UAR1 * 2))
done
echo
echo "The loop execution is finished"

[Bossdev]$ ./until
The until loop example

Value of the variable is :1
Value of the variable is :2
Value of the variable is :4
Value of the variable is :8
Value of the variable is :16
Value of the variable is :32
Value of the variable is :64

The loop execution is finished
[Bossdev]$
```

# 流控制语句

- ❖ 跳出循环：**break**和**continue**
- ❖ **Break**: 跳出整个循环
- ❖ **Continue**: 跳过本次循环，进行下次循环

# 流控制语句

- ❖ end\_loop例子:
- ❖ #! /bin/sh
- ❖ while true
- ❖ do
- ❖     echo "\*\*\*\*\*"
- ❖     echo "Please select your operation:"
- ❖     echo " 1  Copy"
- ❖     echo " 2  Delete"
- ❖     echo " 3  Backup"
- ❖     echo " 4  Quit"
- ❖     echo "\*\*\*\*\*"
- ❖     read op

# 流控制语句

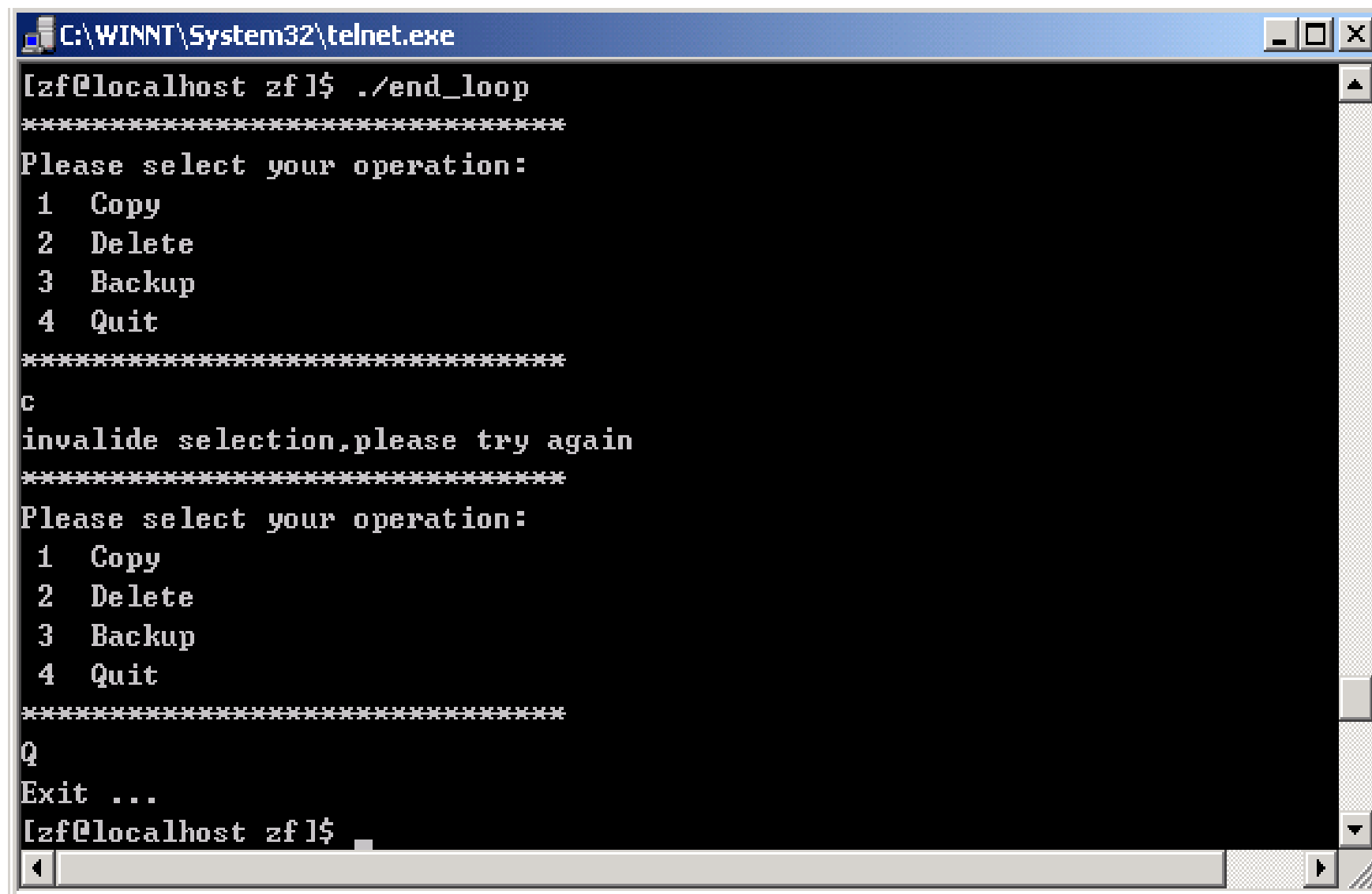
```
❖ case $op in
❖     C)
❖         echo "your selection is Copy"
❖         ;;
❖     D)
❖         echo "your selection is Delete"
❖         ;;
❖     B)
❖         echo "your selection is Backup"
❖         ;;
```



# 流控制语句

```
❖ Q)
❖     echo "Exit ..."
❖     break
❖     ;;
❖ *)
❖     echo "invalide selection,please try again"
❖     continue
❖ esac
❖ done
```

# 流控制语句



```
C:\WINNT\System32\telnet.exe

[zf@localhost zf]$ ./end_loop
*****
Please select your operation:
1  Copy
2  Delete
3  Backup
4  Quit
*****
c
invalide selection,please try again
*****
Please select your operation:
1  Copy
2  Delete
3  Backup
4  Quit
*****
Q
Exit ...
[zf@localhost zf]$
```

# Break语句的使用

C:\WINNT\System32\telnet.exe

```
[Bossdev]$ cat break_01
#!/usr/bin/sh
while true
do
    echo "Enter name of file to be displayed:\c"
    read FILE
    if [ ! -f $FILE ]
    then
        echo "This is not a regular file"
        break
    fi
    cat $FILE
done
echo "Good Bye!!"
[Bossdev]$ _
```

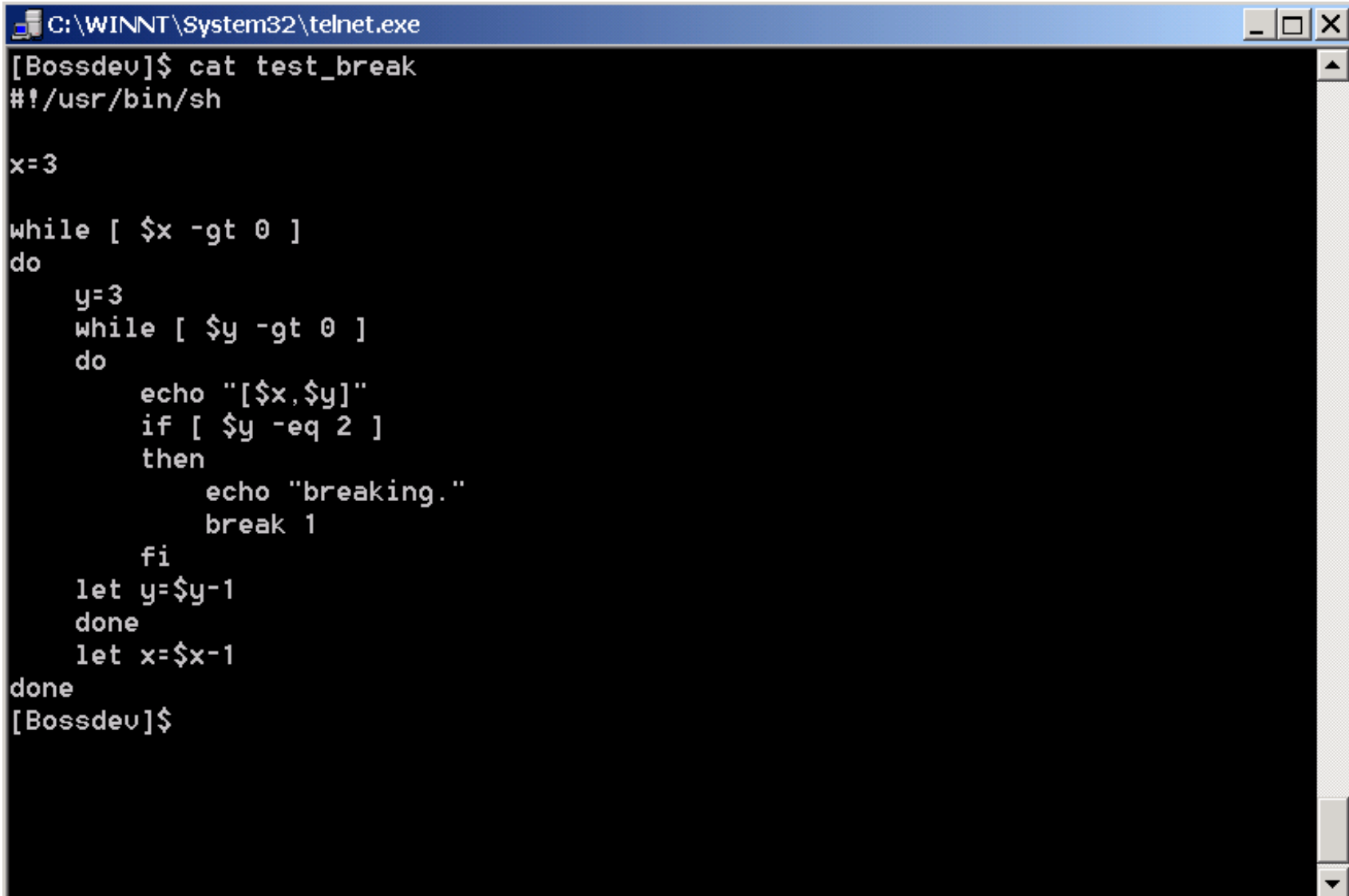
# Break语句的使用

C:\WINNT\System32\telnet.exe

```
[Bossdev]$ ./break_01
Enter name of file to be displayed:script_oo
#!/usr/bin/sh -x
#This is to show what a script looks like.
echo "Our first script"
echo "-----"
echo #This inserts an empty line in output.
echo "We are currently in the following directory"
pwd
echo
echo "This directory contains the following files"
ls

Enter name of file to be displayed:zhangsan
This is not a regular file
Good Bye!!
[Bossdev]$
```

# Break语句的使用

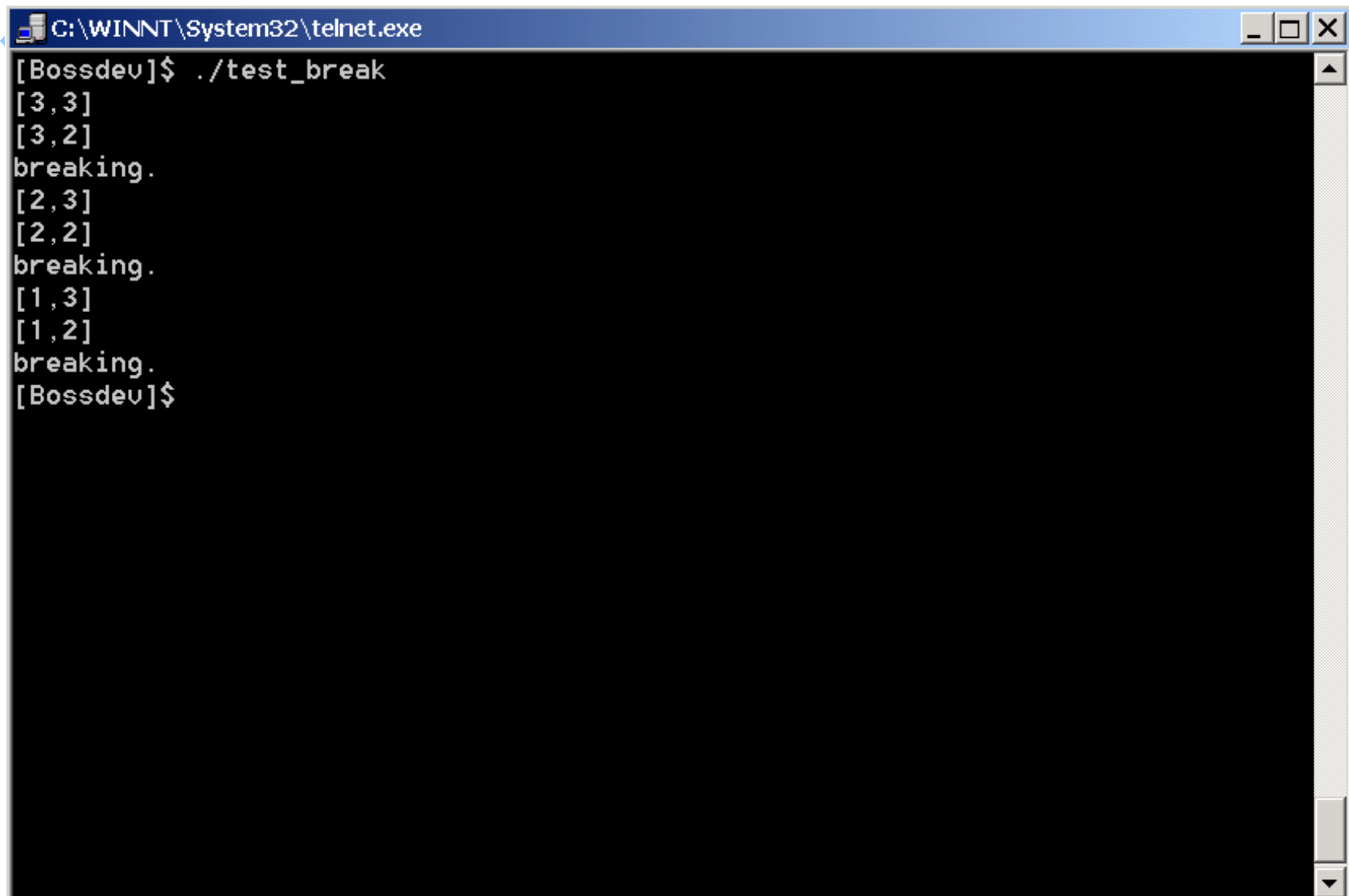


```
C:\WINNT\System32\telnet.exe
[Bossdev]$ cat test_break
#!/usr/bin/sh

x=3

while [ $x -gt 0 ]
do
    y=3
    while [ $y -gt 0 ]
    do
        echo "[$x,$y]"
        if [ $y -eq 2 ]
        then
            echo "breaking."
            break 1
        fi
        let y=$y-1
    done
    let x=$x-1
done
[Bossdev]$
```

# Break语句的使用



```
C:\WINNT\System32\telnet.exe
[Bossdev]$ ./test_break
[3,3]
[3,2]
breaking.
[2,3]
[2,2]
breaking.
[1,3]
[1,2]
breaking.
[Bossdev]$
```

# Continue语句的使用

```
C:\WINNT\System32\telnet.exe
[Bossdev]$ cat continue_01
#!/usr/bin/sh
while true
do
    echo "Enter name of file to be displayed: \c"
    read FILE
    if [ -f $FILE ]
    then
        cat $FILE
        continue
    fi
    echo "This is not a regular file"
    break
done
echo "Good Bye!!!"
[Bossdev]$
```

# Continue语句的使用

C:\WINNT\System32\telnet.exe

```
[Bossdev]$ ./continue_01
Enter name of file to be displayed: script_oo
#!/usr/bin/sh -x
#This is to show what a script looks like.
echo "Our first script"
echo "-----"
echo #This inserts an empty line in output.
echo "We are currently in the following directory"
pwd
echo
echo "This directory contains the following files"
ls

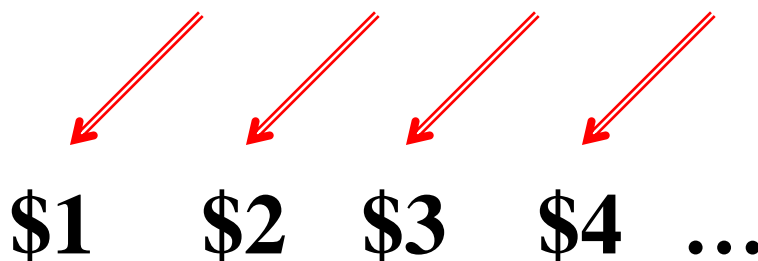
Enter name of file to be displayed: regular
This is not a regular file
Good Bye!!!
[Bossdev]$ _
```



# 流控制语句

- ❖ **Shift**指令：参数左移，每执行一次，参数 序列顺次左移一个位置，**\$#**的值减**1**，用于分别处理每个参数，移出去的参数 不再可用

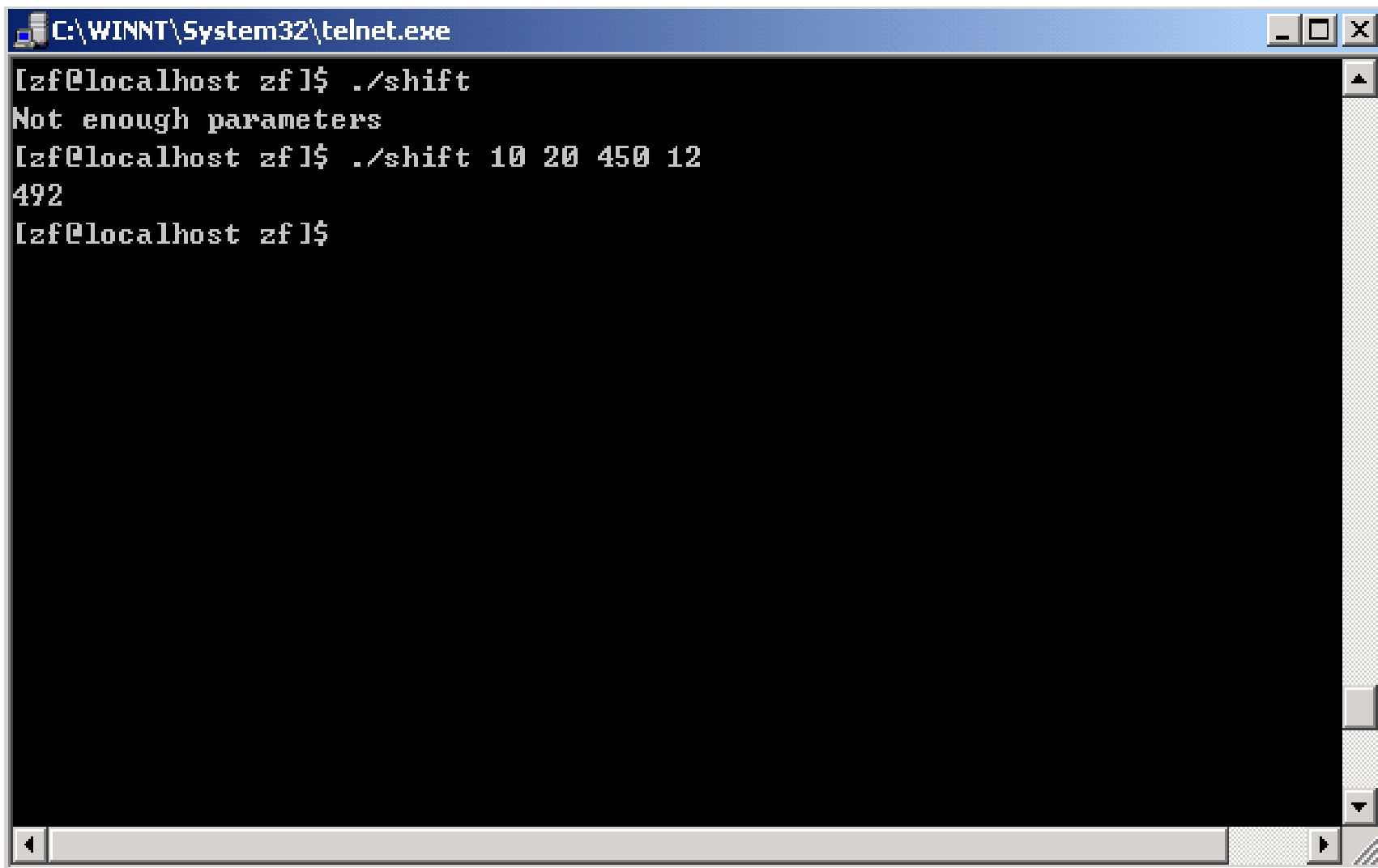
❖ 例如：    **\$1    \$2    \$3    \$4 ...**



# 流控制语句

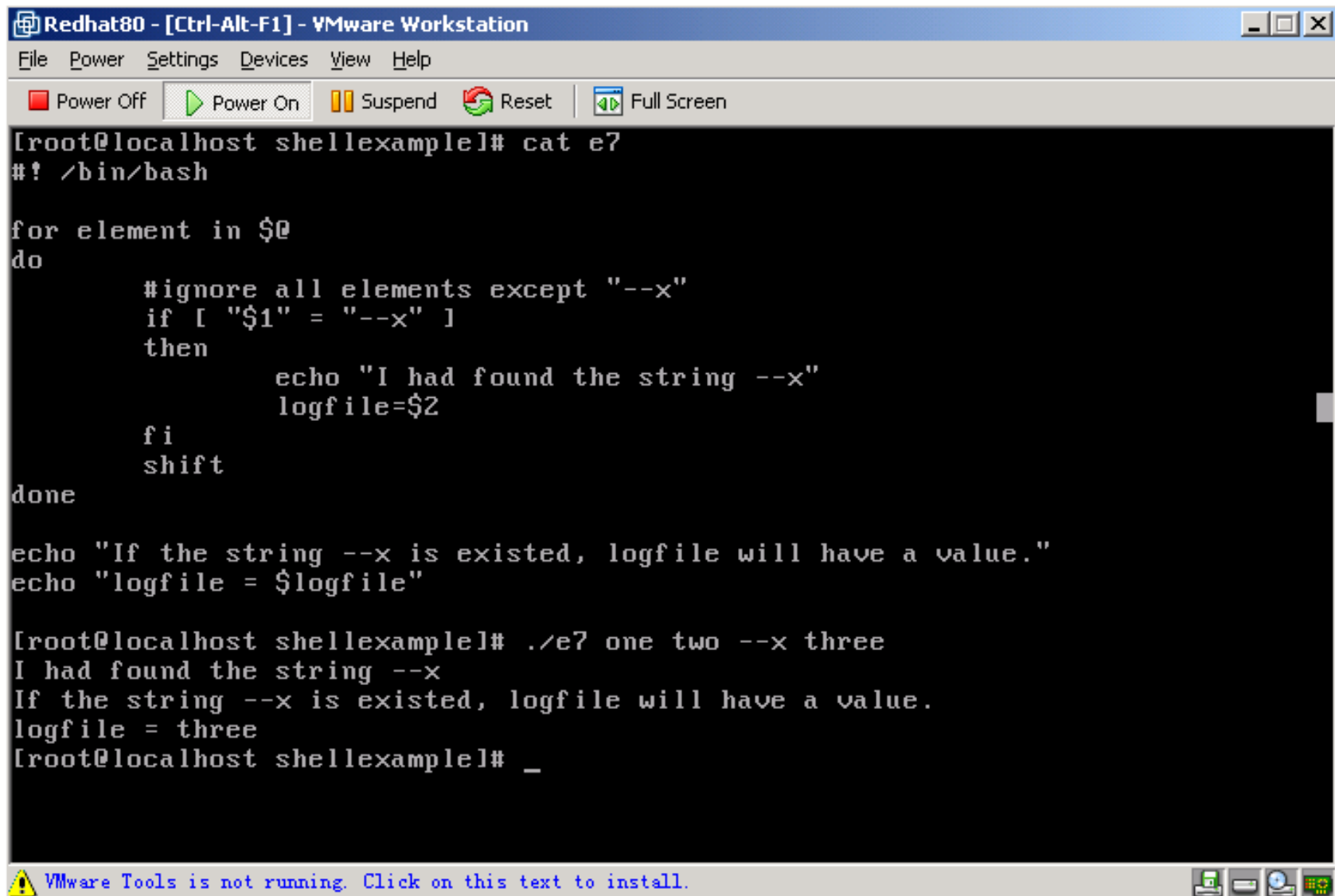
- ❖ **Shift**的例子:
- ❖ if [ \$# -le 0 ]
- ❖ then
- ❖ echo "Not enough parameters"
- ❖ exit 1
- ❖ fi
- ❖ sum=0
- ❖ while [ \$# -gt 0 ]
- ❖ do
- ❖ sum=`expr \$sum + \$1`
- ❖ shift
- ❖ done
- ❖ echo \$sum

# 流控制语句



```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./shift
Not enough parameters
[zf@localhost zf]$ ./shift 10 20 450 12
492
[zf@localhost zf]$
```

# shift命令的使用



```
Redhat80 - [Ctrl-Alt-F1] - VMware Workstation
File Power Settings Devices View Help
Power Off Power On Suspend Reset Full Screen

[root@localhost shellexample]# cat e7
#!/bin/bash

for element in $@
do
    #ignore all elements except "--x"
    if [ "$1" = "--x" ]
    then
        echo "I had found the string --x"
        logfile=$2
    fi
    shift
done

echo "If the string --x is existed, logfile will have a value."
echo "logfile = $logfile"

[root@localhost shellexample]# ./e7 one two --x three
I had found the string --x
If the string --x is existed, logfile will have a value.
logfile = three
[root@localhost shellexample]# _
```

! VMware Tools is not running. Click on this text to install.

# 流控制语句

## ❖ 函数的定义：

函数名 ( )

{

    命令序列

}

## ❖ 函数的调用：不带( )

函数名   参数**1**   参数**2** ...

# 流控制语句

## ❖ 函数中的变量:

变量均为全局变量，没有局部变量

## ❖ 函数中的参数: 调用函数时，可以传递参数，在函数中用**\$1**、**\$2...**来引用

# 流控制语句

- ❖ 函数function的例子:
- ❖ `abc=123`
- ❖ `echo $abc`
- ❖ `###` 定义函数
- ❖ `example1()`
- ❖ `{`
- ❖ `abc=456`
- ❖ `}`

# 流控制语句

- ❖ ### 调用函数
- ❖ example1
- ❖ echo \$abc
- ❖ ### 定义函数，使用参数
- ❖ example2()
- ❖ {
- ❖ echo \$1
- ❖ echo \$2
- ❖ }

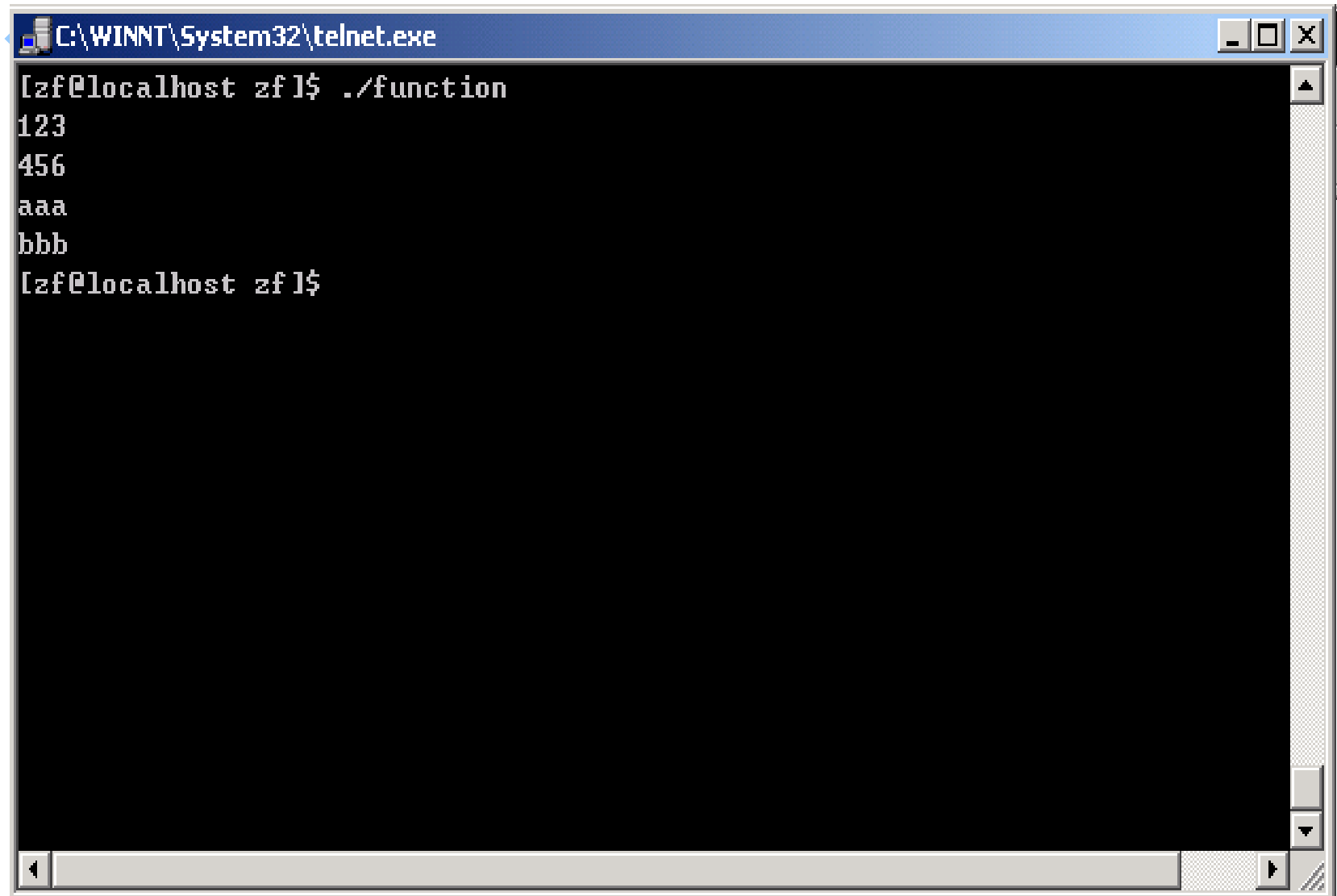


# 流控制语句

❖ ###调用函数,向它传递参数

❖ example2   aaa   bbb

# 流控制语句

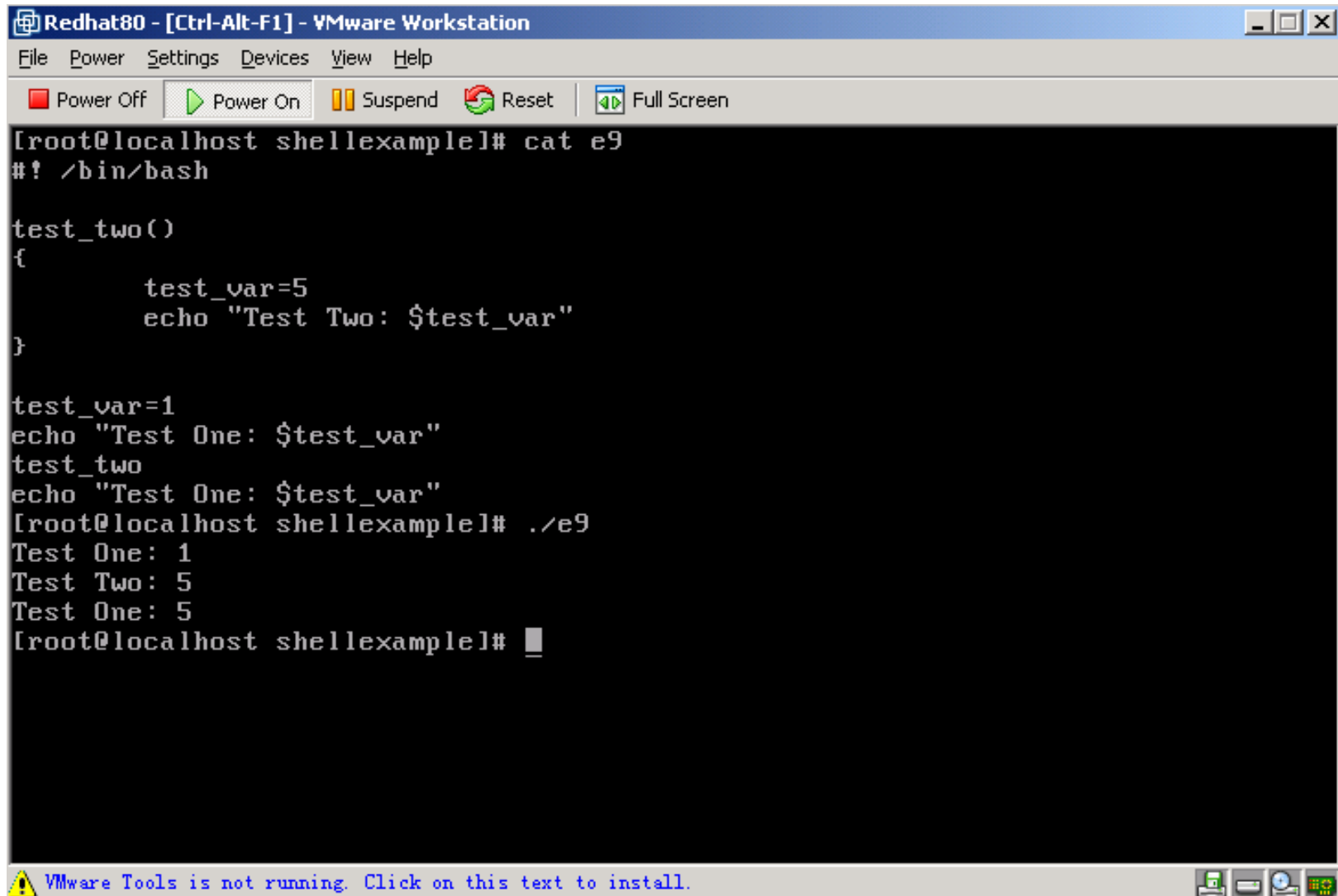


```
C:\WINNT\System32\telnet.exe
[zf@localhost zf]$ ./function
123
456
aaa
bbb
[zf@localhost zf]$
```

# Local命令的使用

- ❖ 函数的局部变量可以永久地改变全局变量，但如果使用命令**local**，可以对过程的变量的修改不影响其他过程。

# Local命令的使用



The screenshot shows a terminal window titled "Redhat80 - [Ctrl-Alt-F1] - VMware Workstation". The window has a menu bar with "File", "Power", "Settings", "Devices", "View", and "Help". Below the menu bar is a toolbar with buttons for "Power Off", "Power On", "Suspend", "Reset", and "Full Screen". The terminal content shows a root user at localhost in a shell named "shellexample". The user runs "cat e9" to display the contents of a file named "e9". The file contains a shell script with a function "test\_two()" that sets "test\_var" to 5 and prints "Test Two: \$test\_var". Below the function, "test\_var" is set to 1 and "echo 'Test One: \$test\_var'" is run. Then "test\_two" is called, which prints "Test One: 1" and "Test Two: 5". Finally, the user runs "./e9", which prints "Test One: 1", "Test Two: 5", and "Test One: 5". At the bottom of the window, there is a message: "VMware Tools is not running. Click on this text to install." with a yellow warning icon.

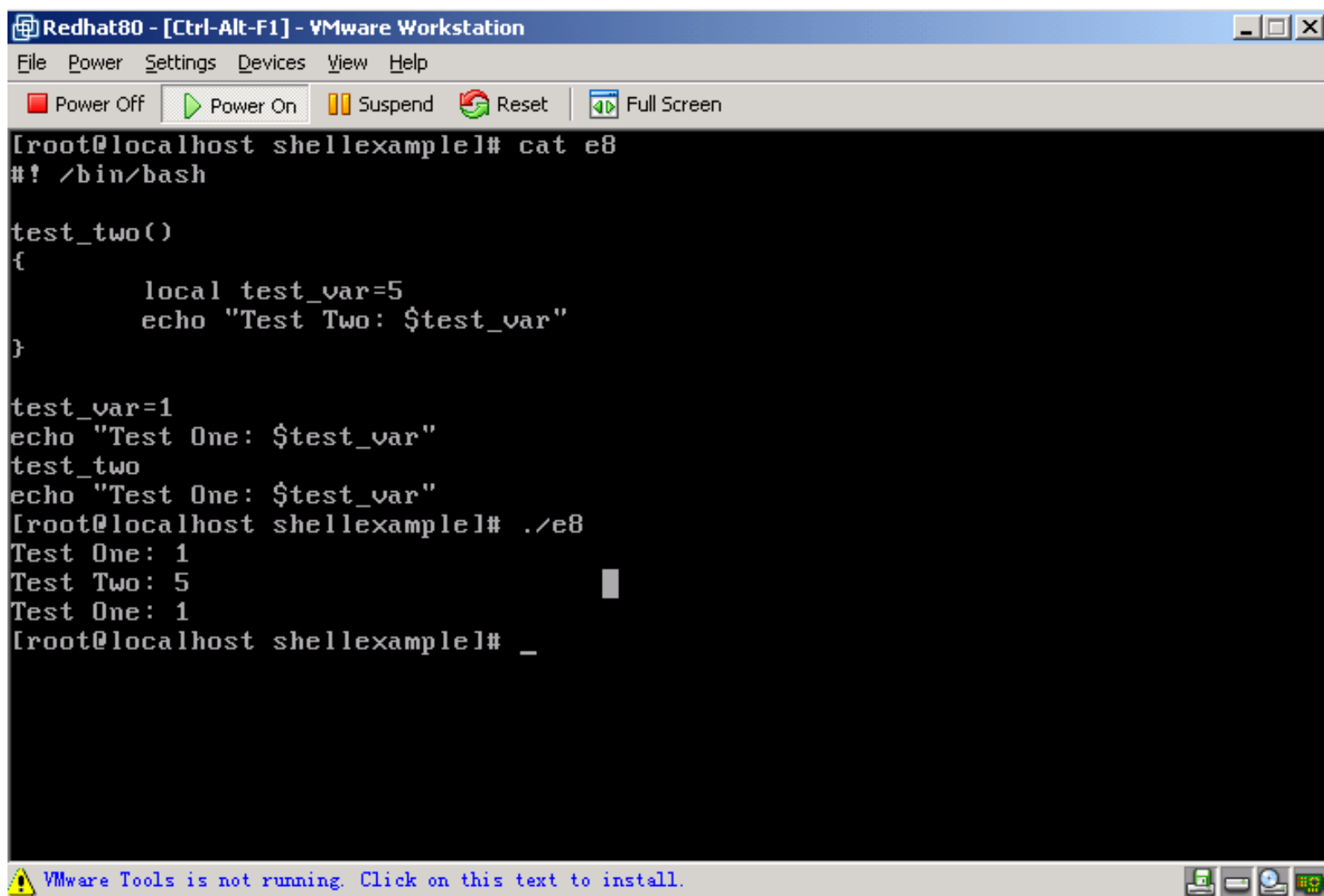
```
[root@localhost shellexample]# cat e9
#!/bin/bash

test_two()
{
    test_var=5
    echo "Test Two: $test_var"
}

test_var=1
echo "Test One: $test_var"
test_two
echo "Test One: $test_var"
[root@localhost shellexample]# ./e9
Test One: 1
Test Two: 5
Test One: 5
[root@localhost shellexample]#
```

VMware Tools is not running. Click on this text to install.

# Local命令的使用



The screenshot shows a terminal window titled "Redhat80 - [Ctrl-Alt-F1] - VMware Workstation". The window has a menu bar with "File", "Power", "Settings", "Devices", "View", and "Help". Below the menu bar is a toolbar with buttons for "Power Off", "Power On", "Suspend", "Reset", and "Full Screen". The terminal content shows a root user at localhost in a shell named "shellexample1". The user runs the command "cat e8", which displays the contents of a file named "e8". The file contains a function definition for "test\_two()", a variable assignment "test\_var=1", and two "echo" commands. The user then runs the command " ./e8", which executes the script and produces the output "Test One: 1", "Test Two: 5", and "Test One: 1". The terminal prompt returns to the root user. At the bottom of the window, there is a status bar with a warning icon and the text "VMware Tools is not running. Click on this text to install.".

```
Redhat80 - [Ctrl-Alt-F1] - VMware Workstation
File Power Settings Devices View Help
Power Off Power On Suspend Reset Full Screen

[root@localhost shellexample1]# cat e8
#!/bin/bash

test_two()
{
    local test_var=5
    echo "Test Two: $test_var"
}

test_var=1
echo "Test One: $test_var"
test_two
echo "Test One: $test_var"
[root@localhost shellexample1]# ./e8
Test One: 1
Test Two: 5
Test One: 1
[root@localhost shellexample1]# _
```

VMware Tools is not running. Click on this text to install.

# Thank You !

[zhaofang@email.buptsse.cn](mailto:zhaofang@email.buptsse.cn)

