PDF-XChange
Click to buy NOW!
www.docu-track.com

深入浅出
Linux工具与编程

- ✧ 这是一部有思想、有内容的书刊。
- ✧ 这是一部回答学什么、怎么学的书刊。
- ✧ 这是一部大学毕业生一次阅读、终生受益的书刊。
- ✧ 这是一部集实用性、典型性、模仿性案例的书刊。
- ✧ 这是一部很通俗易懂的书刊。
- ✧ 这是一部帮你突破技术玻璃纸、一通百通的书刊。
- ✧ 这是一部包含作者从业十年心得经验的书刊。
- ✧ 这是一部零起点Linux专家的速成培训教程。

PDF-XChange
Click to buy NOW!
www.docu-track.com

前言

作者在软件行业从业了十年，先后通过了国家软件水平等级考试的《高级程序员》级别和《系统分析师》级别。参加了大型行业软件如银行核心业务系统、前置系统、数据仓库、金卡工程、银行大小额现代化支付、中间件、支付宝银行端接口等一系列项目设计、开发、测试和实施工作。有十分深厚的软件编程经验，同时一直在多家企业负责新员工的培训工作，对企业员工培训有较多心得。新员工常碰到在大量繁多的技术面前，应该学什么、怎么学的问题；各种技术与工具知识点怎么分优先级和重点问题；技术玻璃纸难以突破，水平难以提高的问题等等。在企业里，员工怎样在时间有限、精力有限情况下掌握好实用和有用的技术，满足企业用人的需要？作者总结的经验是培训教材的导向必须注重目的性、思想性、实践性、典型性和实用性，以任务驱动式培训和以目标管理为方法，用图文和言简意赅的语言描述技术思想，用经典程序说明技术思想，用多个项目实战案例解释如何高水平运用技术思想。模仿是人们快速提高自身能力的捷径，本书的许多程序注重经典，采用实际编程使用的架构，可以作为实际编程模板进行模仿。本书的编排充分注重了易学习性、可模仿性和实战性，其中模板化编程案例和规范化练习教程可以让读者短时间把书本知识变成自身的能力。同时本书是一本技术思想深厚的书刊，书刊的许多内容来源于作者十年技术积累的总结，本书有些技术概念的概括来源于作者多年的思考和感悟，项目案例来源于作者从业的实际项目当中。我相信本书许多技术概念的经典概括一经技术人员说出，足以打动Linux行业的资深人士；本书的经典项目案例一经拿到项目中去，常常会让客户觉得你是该行业的技术专家。这是一部由长期从事一线开发的技术人员书写的书刊，书刊内容通俗易懂，力求让Linux技术变得简单，阅读本书能大大增加读者学习Linux技术的信心。本书把繁多Linux技术进行浓缩，能大大节约读者的学习时间和学习成本。本书注重对技术概念简要概括，更注重技术实现，许多章节力求让程序说话，有时细节决定成败，本书有一些技术细节的概括来源于作者多年工作经验的总结。没有理论，实践是盲目的；没有实践，理论是空洞的。本书力求用言简意赅的理论让读者能地道说出技术的精髓，用经典程序和项目案例使读者加深对技术理论的理解。本书用精炼的概念总结技术，用通俗易懂的语言说明技术，用精心的模板程序和项目案例实现技术。

本书特色：

1．零起点的企业级培训教程

零起点，只需大学计算机专业毕业生一般水平即可对本书进行阅读和练习。本书内容通俗易懂，图文并茂，注重知识点总结概括和分类。知识内容注重层层递进，以达到让读者在低起点向专家迈进的目的。有Linux从业基础部分—如Linux C语言程序设计，有Linux从业素质能力培养部分—如Linux命令及其工具、Linux C语言开发工具，有Linux编程专家水平能力训练部分—如Linux进程编程、Linux文件编程、Linux进程间通信、网络编程与XML编程。本书涵盖了Linux原理篇、命令篇、工具篇和程序篇。

专业就是“简单的事情”重复的做，做到专业就是把复杂的事情变得简单，方法为分类、分层、总结、模板化和流程化。而本书正是致力于这一目的，把复

PDF-XChange Click to buy NOW! www.docu-track.com

杂的技术简单明了呈现在读者面前，帮助读者历练成专业人士。

2．大量作者的企业级实训内容

本书的许多章节是作者企业级培训的实训内容，知识点注重了目标明确、言简意赅、分清主次、项目导向，以求达到简洁不简单的效果。本书知识点注重了理论、同时注重了知识点的实用性、可模仿性、案例性和典型性。本书属于企业级实训教程，以Linux行业从业素质能力培养为导向，以实际应用为目标，以简洁的理论和经典练习为过程，以期达到快速提高读者职业水平和职业能力的目的。本书采用Linux行业素质能力模型的训练方法，即将Linux从业知识点逐条列出并把知识点整合到规范化练习案例中，以达到让读者通过模仿练习快速把知识变为能力的目的。如本书Linux工具与命令章节读者按照练习即可达到Linux行业从业所需的中级水平，而Shell章节按照练习即可快速提高到Linux行业老员工的水平。Linux C开发工具章节做到了简洁明了，符合项目级开发需要而且通俗易懂，vi与vim概述告诉学员如何学习练习，gcc、Makefile、gdb讲解了实际项目开发所需的主要知识，抛弃了项目中没有使用到的大量细节内容，易学易掌握，使读者一天就能弄懂学会。本书这些企业级培训内容能帮助读者短时间内学到实用够用的Linux开发知识。

3．学什么，怎么学

技术是很难很快学好的，但可以快速学会常用和关键的技术，本书以实用论为导向，丢弃了项目开发中用不到的众多技术细节。

本书注重理论联系实际，作者把自己十年的项目经验整合到理论和练习中。作者将Linux从业的知识进行分类化、总结化和案例化，许多知识点都以实际工作所需知识为准，也是以作者所掌握主要和重点知识为准。

本书许多章节配有典型程序和规范化案例练习，学习好理论后按照案例练习即可达到技术的提升，练习后相当于潜移默化复制了老员工多年的经验。本书的内容编排告诉了学员学什么，内容的选取完全参照作者十年从业经验所用到的知识，言简意赅的图文讲解和规范化案例练习告诉学员怎么学。本书的编排和内容很注重学习的方法论，知识描述图文并茂，简洁明了，使学员在轻松学习氛围中，短时间内大幅提升自己的技术层次。

4．多个实用项目案例

本书包含了多个高水平的经典项目案例。如Shell章节备份脚本，C语言章节的实用日志库，Linux进程章节数据仓库多进程案例，网络章节的实用文件服务器和实用通信库，XML章节支付宝银行端接口项目的XPath库。这些项目案例有较大的实战参考意义。

5．XML章节填补市场空白

XML是软件行业经常使用的技术，经常应用在数据交换、web服务、内容管理、电子商务、配置脚本等许多方面。XML技术在软件开发中使用得越来越广泛，越来越流行。然而市面上的书刊缺少针对XML开发技术的讲解，本书进行了专门的总结，同时提供了十分丰富练习程序和经典的项目案例。

6．Linux专家速成培训教程

PDF-XChange Click to buy NOW! www.docu-track.com

时间是人类发展的空间，赢得时间就是赢得个人发展的空间。在个人的职业生涯中，一步领先常常可以做到步步领先。只要读者静下心来一个月对本书进行阅读，按照教材进行练习，读者就可以大大提高Linux工具及其编程的技术水平，从而让读者在从业生涯中做到终生受益，完全掌握本书内容即可达到Linux专家水平。可以说本书是一本通向Linux专家之路的速成教程。

由于作者水平有限，本书有所错漏在所难免，希望读者批评指正。

本书内容：

本书总共六篇，所有内容注重了理论联系实际和实战性。每篇的主要内容如下：

第一篇 Linux命令及其工具

本篇包括Linux操作系统介绍，Linux命令说明，Linux常见实用工具（正则表达式、find、sed、awk）说明及实例练习，Shell编程语法说明及编程实例。

第二篇 Linux C语言程序设计

本篇包括C语言基础、C语言控制结构、C语言函数、C语言数组、结构体及指针、C语言预编译、格式化I/O函数、字符串和内存操作函数、字符类型测试函数、字符串转换函数、Linux C语言开发工具(vi与vim编辑器、gcc、Makefile和gdb)。本篇多次运用堆栈表格对程序运行进行解释，这对于理解计算机语言运行机理非常重要。只有理解的才是最深刻的，理解其运行机理，可以触类旁通、一通百通，移植到理解C++语言和Java等语言。

第三篇 Linux 进程

本篇包括Linux编程基本概念、Linux进程、Linux线程、管道与信号、消息队列、信号量和共享内存。Linux进程章节中守护进程模板和数据仓库多进程处理案例可以应用到实际项目中。本篇Linux进程间通信程序范例是实际项目中精简的demo程序，程序模型和使用方法与实际项目中类似。

第四篇 Linux 文件

本篇包括Linux文件编程，该章节对文件函数进行了分类总结并提供了典型范例。

第五篇 网络编程

本篇包括网络知识基础、socket编程。socket编程章节包括TCP并发服务器案例、TCP迭代服务器案例、文件服务器案例、UDP服务器编程、UDP广播、UDP多播、Unix/Linux域套接字编程等。

第六篇 XML编程

本篇包括XML概念、XML语法、XPath语法、libxml编程、支付宝银行端接口XML项目案例。本篇是目前市面上唯一对Linux下XML编程进行全面总结的书刊，在实际项目开发中有较大的借鉴意义。

PDF-XChange Click to buy NOW! www.docu-track.com

目录

PDF-XChange Click to buy NOW! www.docu-track.com

PDF-XChange
Click to buy NOW!
www.docu-track.com

PDF-XChange
Click to buy NOW!
www.docu-track.com

第 1 篇 Linux 命令及其工具

- 第 1 章 Linux系统与命令
- 第 2 章 Linux常用实用工具（正则表达式、find、sed、awk）
- 第 3 章 Shell编程

学海聆听：

- ✧ 合抱之木，生于毫末；九层之台，起于垒土；千里之行，始于足下。
- ✧ 学而不思则罔，思而不学则殆。
- ✧ 知识来源于学习、思考、顿悟、实践和交流。
- ✧ 非学无以广才，非志无以成学。
- ✧ 临渊羡鱼，不如退而结网。
- ✧ 纸上得来终觉浅，绝知此事要躬行。
- ✧ 在模仿中成长，在创新中成功。
- ✧ 好的开始等于成功的一半。
- ✧ 梦想引领未来，行动成就精彩。
- ✧ 确定目标—支配行动—克服困难—实现目标。
- ✧ 工欲善其事，必先利其器。
- ✧ 博学之，审问之，慎思之，明辨之，笃行之。

第1章 Shell编程

Shell编程在Linux从业中经常要用到，而且是在Linux行业网上招聘中经常要求的技能，读者需要重点掌握。

Shell的中文意思是“外壳”，通俗地讲，Shell是一个交互编程接口。Shell是一个命令解释程序，可以用Shell编写各种脚本工具。解释型语言特点是对源文件进行边识别翻译边执行，不会直接生成二进制机器码，Shell就是一个解释性语言。Shell文件执行时交给Shell解释器进行解析，然后调用相应的系统命令进行执行。Shell作为一种命令语言解释器，内置了大量的命令集，涵盖了当前Linux中的所有命令。Shell脚本的执行流程是Shell解释器顺次读取每一行命令，识别成一条条Linux系统指令，然后调用Linux相应的命令接口生成执行结果。

Shell除了作为命令解释程序以外，还是一种高级程序设计语言，它有变量、关键字、有各种控制语句、支持函数模块、有自己的语法结构，利用Shell程序设计语言可以编写出功能很强但代码简单的程序。

Shell有Bourne（简称B） shell、Korn shell、C shell三种类型，三种Shell的功能大同小异，用得最多的还是B Shell。Shell脚本文件头可用#!/bin/sh说明脚本用哪一种Shell执行，#!表示使用哪一种解释器执行当前文本，/bin/sh是指B shell解释器。若文件头无#!说明用哪种解释器执行文本，系统会选择SHELL环境变量作为此文本的解释器。Shell的注释以#号开头，后面接注释文字。

1.1 Shell环境变量

1.1.1 环境变量说明

环境变量用于描述该用户的操作环境下特定意义的变量，或者说是通过设置环境变量来配置用户的操作环境。Linux中的环境变量包括：用户所使用的Shell类型、工作的主目录、登录方式等。

环境变量分为系统环境变量和用户自定义环境变量。系统环境变量为在系统中有特定意义的环境变量，而且在不定义时也存在，系统环境变量可以重新进行赋值。如PATH环境变量就是系统环境变量，表明用户搜索执行码时所用到路径。

Shell用户环境变量是每一个Linux用户定义在.profile或.bash_profile中生效的变量，同时还包括.bash_profile中包含执行脚本的环境变量。

环境变量包括定义和导出生效两部分，定义INFORMIXDIR环境变量如INFORMIXDIR=/usr/informix，导出生效如export INFORMIXDIR。只有导出生效的环境变量才能被引用，引用时变量需要前要加$符号。

环境变量定义和导出有如下两种格式：

```
① name=value; export name
② name=value
  export name
```

unset命令用来删除环境变量，如unset USERNAME是删除USERNAME变量。

环境变量使用alias进行变量重定义，重定义的环境变量不需要用export导出，定义方法如alias ygp='cd ~/public/ygp'。

1.1.2 用户常用的系统环境变量

表 3-1 列出了用户常用的系统环境变量，这些变量可以在.bash_profile中重新赋值，以便适应用户环境客户化的需要。

表 3-1 用户常用系统环境变量表

变 量 名	意 义
PWD	当前用户的工作目录
HOME	用户主目录的路径全名
LOG NAME	用户的登录名
SHELL	当前所使用的 Shell
PATH	搜索执行码的路径
PS1	命令行提示符
LANG	定义语言编码方式
EXINIT	保存 vi 编辑初始化设置选项 如设置行号，设置 tab 为 4 个空格，设置方法如下： EXINIT='set nu tab=4';export EXINIT

下面是env.sh脚本，用来输出环境变量。

```
#!/bin/sh
echo "PWD:"$PWD
echo "path:"$PATH
echo "Logname:"$LOGNAME
echo "Shell:"$SHELL
echo "HOME:"$HOME
```

```
$ chmod u+x env.sh      --增加执行权限
$ ./env.sh              --执行，执行结果如下：
```

```
PWD:/home/zfb
path:/home/zfb/extra/bin:/home/zfb/extra/bin:/usr/lib/jdk/bin:/usr/kerberos/bin
Logname:zfb
Shell:/bin/bash
HOME:/home/zfb
```

带格式的：项目符号和编号

1.1.3 用户登录脚本示例

1. 命令行提示符

下文约定脚本约定行首$为命令行的提示符。

```
$pwd
/home/zfb/public/ygp/shell
```

2. 用户.bash_profile脚本部分内容

下面是一个用户下的.bash_profile脚本，后文会对此脚本进行解释说明。

```
INFORMIXDIR=/usr/informix
PATH=$PATH:$INFORMIXDIR/bin:$HOME/bin:.
PATH=$PATH:/opt/subversion/bin.
PS1='linux开发':'$PWD>'
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi
alias rm='rm -i'
alias ygp='cd /home/zfb/public/ygp'
export PATH INFORMIXDIR
```

(1) 脚本具体说明如下

上面的示例中PATH、INFORMIXDIR都是环境变量，其中PATH是系统环境变量，INFORMIXDIR是用户自定义环境变量，环境变量必须用export命令导出才能生效。

环境变量可以递归赋值，上面示例中的PATH就是进行了递归赋值。

上面示例中HOME为系统环境变量，其路径值是由增加用户时进行指定的。

PATH指出此用户下执行程序的搜索路径，不同路径名用“：”分开，以“.”结束搜索路径。当我们敲入一执行码时./test系统会根据PATH路径去寻找该执行码。

.bash_profile可以包含其他可执行脚本，如上面.bashrc在文件在当前目录存在，则解释执行。

alias是对变量进行重定义的命令。

(2) .bash_profile文件何时执行？

① 用户登录时立即执行。

② 可以手工执行，在主目录下用. ./.bash_profile来进行重新执行。

带格式的：项目符号和编号

1.2 Shell的符号、变量及运行

1.2.1 Shell 中的符号及其含义

在Shell中，内置了许多特定符号，这些符号分别代表着特定的含义，下面是对这些符号的说明。

- *： 匹配 0 个和多个字符组成的串。
- ?： 匹配单个字符。
- []：匹配的字符范围或列表。例如：$ls [a-c]*，将列出以a-c范围内字符开头的所有文件，$ls [e,m,t]*将列出以e、m或t开头的所有文件。
- >：为重定向覆盖输出。
- <：为重定向输入。
- 》：为重定向添加。
- |：管道命令，左边的输出做右边的输入，如ls *.c|wc –l。
- $#：传送给命令Shell的参数序号。
- $-：在Shell启动或使用set命令时提供选项。
- $?：上一条命令执行后返回的值。
- $$：当前Shell的进程号。
- $!：上一个子进程的进程号。
- $@：所有的参数，每个都用双引号引起，以("$1""$2"...)的形式保存所有输入的命令行参数。
- $*：所有的参数，用一个双引号引起整体，以("$1 $2...")的形式保存所有输入的命令行参数。
- $n：位置参数值，n表示位置。
- $0：当前Shell名。
- $：引用某个变量。
- #：注释符号。
- &：后台命令。
- &&（布尔与）与条件符号：仅当其左边命令执行成功后，才执行其右边命令。
- ||（布尔或）或条件符号：仅当其左边命令执行不成功后，才执行其右边命令。
- !（布尔非）： 反转命令的退出状态值。
- ；：命令分隔符，在一个命令行中依次执行各个命令。
- “...”：双引号表示除\、$、’ 和”外，由双引号引起来的字符为普通字符。
- ‘...’：单引号引起来的字符均作为普通字符。
- `...` ：命令替换，倒引号引起来的字符串作为Shell命令执行。

- ~：表示主目录。
- .(内置句点)：执行命令。
- ..：表示上级目录。
- []（内置表达式）：计算算术表达式的值，相当于test。
- {}：用来封装函数体。
- \：表示转义字符。
- <<：重定向输入。
- <<-：重定向输入，输入去掉行首的tab键。

带格式的：项目符号和编号

1.2.2 ``反引号命令替换

``内部整体作为Linux命令执行输出，例 3-1 和例 3-2 说明了反引号的用法。

【例 3-1】命令行的使用

```
$wc -l `ls test.c`
7 test.c
```

【例 3-2】base.sh脚本用例

(1) 编写测试脚本base.sh

```
#!/bin/sh

#使用basename命令得到文件名
file=`basename $0`
echo "file name : $file"

#使用pwd命令得到当前路径
path=`pwd`

#将两个字符串连接起来
path=$path/$file
echo "full path : $path"
exit 0
```

(2) 增加脚本执行权限

```
$chmod u+x base.sh
```

(3) 使用./base.sh，执行结果如下：

```
file name : base.sh
full path : /home/zfb/public/ygp/shell/base.sh
```

带格式的：项目符号和编号

1.2.3 Shell 变量

1. 变量特点

Shell中变量特点如下：

① 无需定义，可直接使用。
② Shell大小写敏感。
③ $为Shell保留字符，变量被其他变量引用时前面需要加$。
④ 变量赋值“=”两边是没有空格的，不然会带来错误。
⑤ 如果在赋值语句中，右边没有任何信息，那么这个变量为一个空字符串；另外仅定义声名而没有赋值的变量，默认也是一个空字符串。
⑥ Shell只有两种类型，一种是整形数字，一种字符串。整形数字必须所有位都

PDF-XChange Click to buy NOW! www.docu-track.com

为数字，类型Shell解释器自动识别。

⑦ 如果一个变量中含空格、制表位、换行符，则要用双引号引起，不然会出错。

⑧ 字符串左右应加双引号“ ”。

⑨ Shell内置 9 个位置变量，从$1 到$9。

2. 引用变量三种方法

Shell中引用变量有如下三种方法：

① 使用双引号引用变量

```
“ $var ”
```

② 使用大括号引用变量

```
{$var}
```

③ 直接引用变量

```
$var
```

3. 用户变量赋值

(1) 用户变量赋值种类

用户变量赋值有如下 4 种方法：

① 直接赋值，赋值方法如下

```
user=meng   #字符串赋值
null=       #空串赋值
number=12345   #数字赋值
```

② 变量赋值，赋值方法如下

```
var1=$user
var2=$var1
```

③ read读入，赋值方法如下

```
read 变量 1［变量 2］
read var1 var2   #当输入abc def并打回车后，则变量var1 和vae2 就被分别赋值abc和def了
```

④ 参数置换方式为变量赋值，赋值方法如下

```
${变量：－字串} 如果变量被设定并非空，则返回是变量的值，否则是字串的值。
${变量：＋字串} 如果变量被设定并非空，则返回是字串的值，否则是变量的值（即空值）。
${变量：＝字串} 如果变量被设定并非空，则返回变量的值，否则是字串的值，同时变量被设成字串。
${变量：？字串} 如果变量被设定并非空，则返回变量的值，否则返回报错。
```

(2) 编写测试程序var.sh，测试观察变量变化效果

```
#!/bin/sh

var1=abc
var2=${var1:-"hello"}
echo "var1="$var1 "var2="$var2
var3=
var4=${var3:-"hello"}
echo "var3="$var3 "var4="$var4

var5=abc
var6=${var5:+"hello"}
echo "var5="$var5 "var6="$var6
var8=${var7:+"hello"}
echo "var7="$var7 "var8="$var8

var9=abc
var10=${var9:="hello"}
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
echo "var9="$var9 "var10="$var10
var11=
var12=${var11:="hello"}
echo "var11="$var11 "var12="$var12

var13=abc
var14=${var13:?"hello"}
echo "var13="$var13 "var14="$var14
var16=${var15:?"hello"}
echo "var15xx="$var15"

echo "game over"
```

增加执行权限并执行：

```
$chmod u+x var.sh
$./var.sh
```

执行结果如下：

```
var1=abc var2=abc
var3= var4=hello
var5=abc var6=hello
var7= var8=
var9=abc var10=abc
var11=hello var12=hello
var13=abc var14=abc
./var.sh: line 27: var15: hello
```

4. 位置变量

Shell脚本可以向脚本命令行传递参数。在Shell中，$0 表示执行的程序名，$1 到$9 是传递的命令行参数，Shell脚本最多能传递 9 个参数，$1~$9 称为Shell内置的位置变量。shift会让位置参数左移一位，即$2 变$1，$3 变$2。

(1) 编写测试程序arg.sh，测试位置变量变化效果

```
#!/bin/sh
echo NO.0 $0
echo NO.1 $1
echo NO.2 $2
echo NO.3 $3
echo NO.4 $4
echo NO.5 $5
echo NO.6 $6
echo NO.7 $7
echo NO.8 $8
echo NO.9 $9

shift
echo shifting
echo NO.0 $0
echo NO.1 $1
echo NO.2 $2
echo NO.3 $3
echo NO.4 $4
echo NO.5 $5
echo NO.6 $6
echo NO.7 $7
echo NO.8 $8
echo NO.9 $9
```

PDF-XChange Click to buy NOW! www.docu-track.com

(2) 增加脚本执行权限并执行

```
$ chmod u+x arg.sh
$./arg.sh arg1 arg2 arg3 arg4
```

(3) 执行结果如下

```
NO.0 ./arg.sh
NO.1 arg1
NO.2 arg2
NO.3 arg3
NO.4 arg4
NO.5
NO.6
NO.7
NO.8
NO.9
shifting
NO.0 ./arg.sh
NO.1 arg2
NO.2 arg3
NO.3 arg4
NO.4
NO.5
NO.6
NO.7
NO.8
NO.9
```

5. 字符串变量

字符串变量左右应加双引号，否则会报错。

(1) 编写测试程序str.sh，测试字符串变量变化效果

```
#!/bin/sh
string1=good morning   #没有双引号引起字符串，执行时会报错
string2="good morning"
echo "string1:"$string1
echo "string2:"$string2
unset string2   #清空变量的值
echo "string2:"$string2
```

(2) 执行 sh str.sh，执行结果如下：

```
str.sh: 3: morning: not found
string1:
string2:good morning
string2:
```

6. 表达式求值

expr命令用来对表达式进行求值，其操作符和运算符之前必须有空格隔开。在Shell脚本中，数学表达式直接运算需要用两对圆括号括起来。

【例 3-3】expr.sh脚本

(1) 编写测试程序expr.sh，测试表达式求值的效果

```
#!/bin/sh

expr 3 + 9
expr 9 % 2
expr 3 \* 2   #\表示转义
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
sum=$((3+2))
echo sum:$sum
mod=$(( 3 % 2 ))
echo mod:$mod
mul=$(( 3 * 2 ))
echo mul:$mul

a=3
c=$(($a +8))
echo c:$c
```

(2) 执行sh expr.sh，执行结果如下：

```
12
1
6
sum:5
mod:1
mul:6
c:11
```

【例 3-4】let.sh脚本

在Shell中，使用let内置命令可以完成对数值的运算。

(1) 编写测试程序let.sh，测试let命令的使用

```
#!/bin/bash
let a=11
let a=a+5
echo "11 + 5 = $a"

let "a <<= 3"       # let "a = a << 3"
echo "\"\$a\" (=16) left-shifted 3 places = $a"

let "a /= 4"        # let "a = a / 4"
echo "128 / 4 = $a"

let "a -= 5"        # let "a = a - 5"
echo "32 - 5 = $a"

let "a *=  10"      # let "a = a * 10"
echo "27 * 10 = $a"

let "a %= 8"        # let "a = a % 8"
echo "270 modulo 8 = $a  (270 / 8 = 33, remainder $a)"
exit 0
```

(2) 执行./let.sh，执行结果如下：

```
11 + 5 = 16
"$a" (=16) left-shifted 3 places = 128
128 / 4 = 32
32 - 5 = 27
27 * 10 = 270
270 modulo 8 = 6  (270 / 8 = 33, remainder 6)
```

带格式的：项目符号和编号

1.2.4 Shell 脚本执行

Shell命名规则一般是filename.sh，结尾以.sh表示文件类型。

Shell脚本有两种执行方式：一种为sh filename.sh，第二种执行方式是对

文件增加执行权限然后敲入执行码进行执行，如 chmod u+x filename.sh; ./filename1.sh。

带格式的：项目符号和编号

1.2.5 Shell 退出状态

1. Shell退出状态说明

图 3-1 画出了Shell脚本的执行流程及退出状态的保存方法。由于Shell解释器是创建子进程执行Shell脚本，因此Shell进程是Shell脚本的父进程，所以Shell可以得到子进程的状态，Shell脚本内命令同理。退出状态必须是十进制数，范围必须是 0 至 255，Shell脚本执行成功返回 0，报错返回非 0。

$? 是一个Shell中的内置变量，代表着最后一次运行进程的退出状态码。

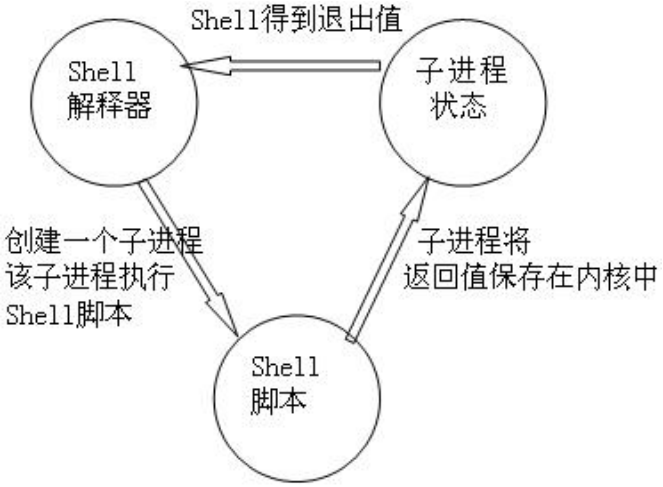

图 3-1 Shell退出状态图

2. Shell退出状态举例

下面的用例用来说明Shell的退出状态。

(1) 编写测试程序test.c

```
#include <stdio.h>
int main(void)
{
    printf("test program\n") ;
    return 35 ;
}
```

(2) 在Shell命令行中编译该程序

```
gcc test.c -o test
```

(3) 编辑exit.sh脚本，脚本内容如下

```
#!/bin/sh
#exit.sh 测试不同进程的退出状态码

./test #执行test程序
echo statas=$?

exit 45
```

(4) 执行exit.sh脚本

```
sh exit.sh
```

PDF-XChange Click to buy NOW! www.docu-track.com

(5) 执行结果如下：

```
test program
statas=35
```

(6) 查看最后一次退出状态

```
$ echo $?
45
```

带格式的：项目符号和编号

1.3 Shell的输入输出

1.3.1 Shell 的输入

Shell用来输入的指令是read函数，其格式说明如下：

```
read 变量1［变量2］
```

利用read函数可以交互式地为变量赋值，当然也可以通过制表符或空格为多个变量赋值，使用read函数读入变量的三种情况说明如下：

① 如果变量的个数多于输入串的字符个数，则依次赋值，剩下的变量取空值。

② 如果变量的个数等于输入串的字符个数，则一一对应赋值。

③ 如果变量的个数小于输入串的字符个数，则除依次赋值外，最后一个变量接纳剩下的所有字符串。

【例 3-5】read函数的例子

read.sh源代码如下：

```
#!/bin/sh
echo "input your name and age:"
read name age
echo " name is :"$name
echo "age is :"$age
```

上述代码的意思是从终端中输入两个值，分别赋值给name和age这两个变量，然后利用echo函数将其输出，执行结果如下：

```
$sh read.sh
input your name and age:
sky 3000    --键盘输入
name is :sky
age is :3000
```

带格式的：项目符号和编号

1.3.2 Shell 的输出

1. echo函数介绍

echo是Shell中实现文本和变量输出的函数，能够输出提示信息，显示执行结果和报告执行状态等。

echo函数后面的各参数之间以空格隔开，以换行符终止。如果数据之间需保留多个空格，则要用双引号把它们整个串给引起来，以便Shell对它们进行正确地操作。

echo函数中，还定义了一组转义字符，用于输出控制或打印无法显示的字符。在使用转义字符的时候，要加入“ -e ”选项。

2. 输出转义字符

表 3-2 列出了Shell中的转义字符并对其作用进行了说明，echo函数利用这些转义字符，可以打印出无法显示的字符。

表 3-2 转义字符及其作用说明表

转义字符	作　用
\a	响铃报警
\b	后退一个字符位置
\c	它出现在参数的最后位置。在它之前的参数被显示后，光标不换行，新的输出信息接在本行后
\e	转义字符
\f	换页
\n	显示换行
\r	回车
\t	制表符
\v	垂直制表符
\\	反斜线本身

带格式的：项目符号和编号

1.4 Shell测试条件

Shell提供两种测试条件的方式，利用test命令和利用方括号形式，其定义格式如下：

```
test –d $dir
[-d $dir]
```

这两种方式完全等价，即[表达式] 等价于test 表达式。

1. 条件测试分类

条件测试可以分为四类：字符串测试、数值测试、逻辑测试、文件属性测试。

2. 字符串测试

表 3-3 列出了对字符串测试操作的说明，字符串测试的作用是用于测试字符串操作的返回值。注意使用到=、!=、<、>这些符号时，两边需要加空格。

表 3-3 字符串测试表

参　数	作　用
-z s1	如果字符串 s1 的长度为 0，则测试条件为真
-n s1	如果字符串 s1 的长度大于 0，则测试条件为真
s1	如果字符串 s1 不是空字符串，则测试条件为真
s1 = s2	如果字符串 s1 等于字符串 s2，则测试条件为真
s1 != s2	如果字符串 s1 不等于字符串 s2，则测试条件为真
s1 < s2	如果按字符顺序字符串 s1 在字符串 s2 之前，则测试条件为真
s1 > s2	如果按字符顺序字符串 s2 在字符串 s1 之前，则测试条件为真

下面以test_str1.sh、test_str2.sh两个用例来说明字符串测试的使用方法。

【例 3-6】test_str1.sh用例

(1) 编写测试程序test_str1.sh

```
#!/bin/sh
echo please input name:
read name
if test $name
then
    echo "name is :"$name
else
    echo "name is null"
fi
```

PDF-XChange Click to buy NOW! www.docu-track.com

(2) 执行sh test_str1.sh，执行结果如下：

```
please input name:              --直接回车

name is null
```

【例 3-7】 test_str2.sh用例

(1) 编写测试程序test_str2.sh

```
#!/bin/sh

str1="happy"
str2="happy"
str3=

#测试str1 与str2 相等
test $str1 = $str2
echo $?

#测试str3 是否是空串
test -z $str3
echo $?

#测试str1 与str2 不相等
test $str1 != $str2
echo $?

#使用test另一种方式[]来实现上述三种判断
echo "using [ ] "

#测试str1 与str2 相等
[ $str1 = $str2 ]
echo $?

#测试str3 是否是空串
[ -z $str3 ]
echo $?

#测试str1 与str2 不相等
[ $str1 != $str2 ]
echo $?
```

(2) 执行sh test_str2.sh，执行结果如下：

```
0
0
1
using [ ]
0
0
1
```

请大家扩充Shell脚本功能，把所有参数使用一遍，进行举一反三，以下同。

3. 数值测试

表 3-4 列出了数值测试的使用方法，数值测试主要用于两个数值之间大小的比较。

表 3-4 数值测试表

参 数	作 用

PDF-XChange Click to buy NOW! www.docu-track.com

n1 -eq n2	如果整数 n1 等于 n2，则测试条件为真
n1 -ne n2	如果整数 n1 不等于 n2，则测试条件为真
n1 -lt n2	如果整数 n1 小于 n2，则测试条件为真
n1 -le n2	如果整数 n1 小于或等于 n2，则测试条件为真
n1 -gt n2	如果整数 n1 大于 n2，则测试条件为真
n1 -ge n2	如果整数 n1 大于或等于 n2，则测试条件为真

下面以test_number.sh用例来说明数值的使用方法。

【例 3-8】test_number.sh用例

(1) 编写测试程序test_number.sh

```
#!/bin/sh

a=1
b=3

test $a -eq $b
echo $?

test 6 -eq 6
echo $?

#使用[]方式完成上述功能
echo "using [ ] "
[ $a -eq $b ]
echo $?

[ 6 -eq 6 ]
echo $?
```

(2) 执行sh test_number.sh，执行结果如下：

```
1
0
using [ ]
1
0
```

4. 逻辑测试

表 3-5 列出了逻辑测试的使用方法。逻辑运算符的作用是用于逻辑语句的判断，也就是对“与”、“或”、“非”条件的判断。逻辑表达式中优先级的顺序是：“ () ”运算符>“ ！”运算符>“ -a ”运算符>“ -o ”运算符。

表 3-5 逻辑测试表

参　数	作　用
!	逻辑“非”，放在任意逻辑表达式之前，使原来为真的表达式变为假，使原来为假的表达式变为真
-a	逻辑“与”，放在两个逻辑表达式中间，只有两个表达式都为真，结果才为真，否则为假
-o	逻辑“或”，放在两个逻辑表达式中间，只有两个表达式都为假，结果才为假，否则为真
()	圆括号可以把一个逻辑表达式括起来，使之成为一个整体，优先得到运算

下面以test_log.sh用例来说明逻辑测试的使用方法。

【例 3-9】test_log.sh用例

(1) 编写测试脚本test_log.sh

PDF-XChange Click to buy NOW! www.docu-track.com

```
#!/bin/sh

[ -x $1 -a $0 ] #检查两个文件是否同时可执行，其中一个是Shell脚本本身
echo $?

[ -w $1 -o $0 ] #检查两个文件是否有一个可写，其中一个是Shell脚本本身
echo $?
```

(2) 增加脚本执行权限

```
$chmod +x test_log.sh
```

(3) 使用ls查看文件参数

```
$l test_log.sh test.c
-rw-rw-r--  1 zfb zfb  86 12月 23 10:11 test.c
-rwxrwxr-x  1 zfb zfb 164 12月 23 19:25 test_log.sh
```

(4) 执行./test_log.sh test.c，执行结果如下：

```
$./test_log.sh test.c
1
0
```

5. 文件属性测试

表 3-6 列出了文件属性测试的使用方法，文件属性测试用于测试文件类型。

表 3-6 文件属性测试表

参数	作　用
-r 文件名	若文件存在并且是用户可读的，则测试条件为真
-w 文件名	若文件存在并且是用户可写的，则测试条件为真
-x 文件名	若文件存在并且是用户可执行的，则测试条件为真
-f 文件名	若文件存在并且是普通文件，则测试条件为真
-d 文件名	若文件存在并且是目录文件，则测试条件为真
-p 文件名	若文件存在并且是命名的 FIFO 文件，则测试条件为真
-b 文件名	若文件存在并且是块设备文件，则测试条件为真
-c 文件名	若文件存在并且是字符设备文件，则测试条件为真
-s 文件名	若文件存在并且文件的长度大于 0，则测试条件为真
-t 文件描述字	若文件被打开并且文件描述字是与终端设备相关的，则测试条件为真。默认的“文件描述字”是 1

下面以test_file.sh用例来说明文件属性测试的使用方法。

【例 3-10】test_file.sh用例

(1) 编写测试脚本test_file.sh

```
#!/bin/sh

file=test.c

[ -r $file ] #测试读权限
echo $?
[ -w $file ] #测试写权限
echo $?
[ -x $file ] #测试执行权限
echo $?
[ -f $file ] #测试是否是文件
echo $?
[ -d $file ] #测试是否是目录
echo $?
```

PDF-XChange Click to buy NOW! www.docu-track.com

(2) 增加脚本执行权限

```
$chmod +x test_file.sh
```

(3) 使用ls查看test.c文件

```
$l test_log.sh test.c
-rw-rw-r--  1 zfb zfb  86 12 月 23 10:11 test.c
```

(4) 使用./test_file.sh查看执行结果

```
0
0
1
0
1
```

带格式的：项目符号和编号

1.5 Shell的流程控制结构

本节介绍Shell流程控制语句，Shell流程控制语句有if语句、case语句、while语句、until语句、for语句和跳转语句(break、continue、exit)，下文将介绍这些语句的语法形式和使用用法。

带格式的：项目符号和编号

1.5.1 if 语句

if语句条件返回值为 0 表示条件测试为真；如果条件命令执行不成功，其返回值不等于 0，条件测试就为假。

if语句的语法形式如下：

```
if   测试条件 1
then  命令或命令表
elif   测试条件 2
then 命令或命令表
else 命令或命令表
fi
```

其中，elif部分和else部分可以省去，一种结构可以演化为三种结构。下面以eq_str.sh、file_test.sh两个用例来说明if语句的使用方法。

【例 3-11】eq_str.sh用例

if语句条件为文件属性测试和字符串测试时需要使用 if [[文件属性测试或字符串测试]]这种格式。

(1) 编写测试脚本eq_str.sh

```
str1="happy new year"
str2="happy new year"


if [[ $str1 = $str2 ]]
then
        echo "they are equal"
fi
```

(2) 增加脚本执行权限

```
$chmod u+x eq_str.sh
```

(3) 执行 ./eq_str.sh，执行结果如下：

```
they are equal
```

【例 3-12】file_test.sh用例

PDF-XChange Click to buy NOW! www.docu-track.com

(1) 编写测试脚本file_test.sh

```
#!/bin/sh
if [ -f $1 ] #普通文件
then
    echo "regular file"
elif [ -d $1 ] #目录文件
then
    echo "dir file" #链接文件
elif [ -l $1 ]
then
    echo "symlink file"
fi
```

(2) 增加脚本执行权限

```
$chmod u+x file_test.sh
```

(3) 执行 ./file_test.sh，执行结果如下：

```
regular file
```

带格式的：项目符号和编号

1.5.2 case 语句

case语句是一种多重判断语句，类似于多个if...elif操作。case语句的执行原理，是将字符串与各个模式顺次匹配，若满足条件则执行，否则继续查找，如果没有匹配成功的，则不执行任何语句，直接退出。

使用case语句时，应注意以下事项：

① 每个模式匹配后的处理语句，是以“;;”两个分号进行结束。

② 模式串表达式应该有唯一性，不要出现几个模式串表达式能够相互转换的情况，这样不利于语句调试。

③ 一个模式表达式可以包含多个模式串，但要用“|”隔开，“|”在这里是“或”的关系。

case语句是一个基于模式匹配的多路分支结构，其一般语法形式如下：

```
case word in
pattern 1) 命令表1;;
pattern 2) 命令表2;;
    ...
     *)  缺省命令表;;
esac
```

下面以case.sh、case_menu.sh两个用例来说明case语句的使用方法。

【例 3-13】case.sh用例

(1) 编写测试脚本case.sh

```
#!/bin/sh
echo please input your name:
read name
case $name in
Tom)
    echo your name is tom ;;
Jim)
    echo your name is Jim ;;
*)
```

```
        echo "sorry we don't know your name" ;;
    esac
```

(2) 执行 sh case.sh，执行结果如下：

```
    please input your name:
    Jim  --输入
    your name is Jim
```

【例 3-14】case_menu.sh用例

(1) 编写脚本case_menu.sh

```
    #!/bin/sh
    echo "1 save"
    echo "2 load"
    echo "3 exit"
    echo  #输出一个换行
    echo "please input chioce"
    read chioce
    #快捷键如下
    #s--存储(save)
    #l--加载(load)
    #e--退出(exit)
    case $chioce in
        1 | S | s)
            echo "save";;
        2 | L | l)
            echo "load";;
        3 | $ | s)
            echo "exit";;
        *)   #其他的输入情况
            echo "invalid choice"
            exit 1;;
    esac
    exit 0
```

(2) 执行 sh case_menu.sh，执行结果如下：

```
    1 save
    2 load
    3 exit

    please input chioce
    s
    save
```

带格式的：项目符号和编号

1.5.3 while 语句

while语句的执行过程是：先测试条件语句是否为真，若为真，则执行循环体；当执行完当前命令后，再进行条件测试，直到条件结果为假，循环结束。这里的条件测试语句既可以是test语句，也可以是运行命令的返回值，若返回值大于 0，则表示条件为真，否则条件为假。

while语句的语法形式如下：

```
    while 测试条件
     do
         命令表
     done
```

下面以ls_file.sh、read_while.sh两个用例来说明while语句的使用方法。

【例 3-15】ls_file.sh用例

执行时每隔 1 秒显示test.c文件大小，使用ctrl+c结束脚本执行。

PDF-XChange Click to buy NOW! www.docu-track.com

```
#!/bin/sh
while true
do
    sleep 1 ;
    ls -l test.c
done
```

【例 3-16】read_while.sh用例

(1) 编写测试脚本read_while.sh

```
#!/bin/sh
type="";
echo input your type:
read type
while [ $type != "quit" ]
do
    echo "your input is :"$type
    echo input your type:
    read type
done
```

(2) 执行 sh read_while.sh，执行结果如下：

```
input your type:
why
your input is :why
input your type:
quit
```

带格式的：项目符号和编号

1.5.4 until 语句

until语句在形式上是while语句的一种变形。对于until语句中的条件测试语句，如果条件为假，则执行；否则不执行。

until语句的语法形式如下：

```
until 测试条件
 do
     命令表
 done
```

下面以until.sh用例来说明until语句的使用方法。

【例 3-17】until.sh用例

(1) 编写测试脚本until.sh

```
#!/bin/sh
type="";
echo input your type:
read type
until [ $type = "quit" ]
do
    echo "your input is :"$type
        echo input your type:
        read type
done
```

(2) 执行 sh until.sh，执行结果如下：

```
input your type:
what
your input is :what
input your type:
quit
```

1.5.5 for 语句

1. for语句统一语法形式

for语句是Shell中经常使用的循环语句，for语句统一的语法形式如下：

```
for  变量名 in  循环参数列表
do
     命令表
 done
```

上面统一语法形式，可以演化下面三种表现形式。

2. 数组作为循环参数

for语句循环参数是数组表时语法形式如下：

```
for  变量名 in  数组表
do
     命令表
 done
```

下面以for_array.sh用例说明for语句循环参数是数组表时的用法。

【例 3-18】for_array.sh用例

(1) 编写测试脚本for_array.sh

```
#!/bin/sh

for word in Hello to you
do
    echo $word
done

array="what who where"
for word in $array
do
    echo $word
done

file=`ls`
for word in $file
do
    echo $word
done
```

(2) 执行 sh for_array.sh，执行结果如下：

```
Hello
to
you
what
who
where
args.sh
case_menu.sh
```

3. 以正则表达式作为循环参数

for语句循环参数是正则表达式时语法形式如下：

```
for  变量 in  正则表达式
do
   命令表
done
```

下面以for_regular.sh用例说明for语句循环参数是正则表达式时的用法。

【例 3-19】for_regular.sh用例

(1) 编写测试脚本for_regular.sh

```
#!/bin/sh
for file in /dev/tty[1-3]
    do
    ls -l $file
done
```

(2) 执行 sh for_regular.sh，执行结果如下：

```
crw-------  1 root root 4, 1 12月 23 08:24 /dev/tty1
crw-------  1 root root 4, 2 12月 23 08:24 /dev/tty2
crw-------  1 root root 4, 3 12月 23 08:24 /dev/tty3
```

4. 位置参数方式作为循环参数

for语句循环参数是位置参数时语法形式如下：

```
for 变量 in  $*
do
   命令表
done
```

下面以for_args.sh用例说明for语句循环参数是位置参数时的用法。

【例 3-20】for_args.sh用例

(1) 编写测试脚本for_args.sh

```
#!/bin/sh
for word in $*
do
    echo $word
done
```

(2) 执行sh for_args.sh arg1 arg2 arg3，执行结果如下：

```
arg1
arg2
arg3
```

带格式的：项目符号和编号

1.5.6 跳转语句

1. break语句

break语句是一个退出循环的命令，主要用于多层循环的嵌套，它的一般使用形式为：

```
break [n]
```

其中，n用来表示跳出几层循环，默认值为 1，即退出本次循环。

2. continue语句

continue语句与break语句有相同之处，都是用于终止本次循环，区别在于break语句是退出整个循环，即不再执行剩下的循环操作，而continue语句是停止本次循环体的执行，转向循环体中的下一次循环。

continue语句的原型为：

```
continue [n]
```

其中，n用来表示跳出几次循环，默认值为 1。

3. exit语句

exit语句是退出正在执行的Shell脚本，可以主动指定返回值。

exit语句的原型为：

```
exit [n]
```

其中，n是主动设定的返回值；如果未显式给定n的值，则该值默认取最后一次命令的执行状态作为返回值。

带格式的：项目符号和编号

1.6 Shell数组

在Shell中可以使用数组来存储同类型的数值集合。一般Shell中支持一维数组，但不限定数组的具体大小，数组的使用方式是采用指定下标。数组中的下标往往是由 0 开始编号的，一般可以采用直接给出下标的方式，也可以通过算术表达式的方式指定下标。但不管采用哪种方式，都要记住一点，不能给出一个小于0，或大于数组已存元素个数的值，不然会产生越界的错误。

在数组的操作中，取值的一般方式是：

```
${数组名[下标值]}
```

相对应的数组的赋值操作的一般方式是：

```
数组名[下标值]=值
```

对于数组的赋值，可以采用单个元素逐一进行赋值，也可以采用一次性赋值的方式，但要注意，一次性赋值时值与值之间要用空格隔开，赋值方法如下：

```
数组名=(value1 value2 value3...)
```

在数组中可以使用*或@符号来代替下标，此时*或@就是上文的通配符。

下面以array.sh用例来说明Shell数组的使用方法。

【例 3-21】array.sh用例

(1) 编写测试脚本array.sh

```
name=(tom jim jane test1)
echo "name[0] is :"${name[0]}
echo "name[1] is :"${name[1]}
echo "name[2] is :"${name[2]}
echo "name[3] is :"${name[3]}
echo "name set  is :"${name[*]}
echo "name set  is :"${name[@]}
```

(2) 增加脚本执行权限

```
$chmod u+x array.sh
```

(3) 执行 ./array.sh，执行结果如下：

```
name[0] is :tom
name[1] is :jim
name[2] is :jane
name[3] is :test1
name set  is :tom jim jane test1
name set  is :tom jim jane test1
```

带格式的：项目符号和编号

1.7 Shell函数

函数代表一种模式化的设计思想，可以将一些常用的、内聚度较高的操作，封装成函数，在需要时进行调用。

在Shell中可以按照一定的规则，定义一组命令集，组成一个过程，这个过程有过程名，在执行这个命令集时，只需要通过使用这个过程名，就能实现执行

PDF-XChange Click to buy NOW! www.docu-track.com

过程中相关命令的功能，这个过程就叫函数。

函数其实是一个模块化的概念，按一定功能，设定一个模块，在使用时，只需指定模块名，便能达到操作的目的。Shell中的函数一次定义，可以多次使用。执行函数操作，并不需要创建新进程，而是在当前的Shell进程中运行。

1. Shell函数定义原型

Shell中函数定义方法如下：

```
function 函数名()
{
   语句
}
```

在这里，关键字function是可以不显式指定。在使用函数时，要注意函数应先定义再使用，调用函数时，只需指定函数名，不用加后面的()。

下面以show.sh用例来说明Shell函数的使用方法。

【例 3-22】show.sh用例

(1) 编写测试脚本show.sh

```
show()
{
        echo  $a $b $c
        echo  $1 $2 $3
}
a=111
b=222
c=333
d=444
f=555
e=666
echo "Function Begin"
show $d $e $f
echo "Function Finished"
```

(2) 执行sh show.sh，执行结果如下：

```
Function Begin
111 222 333
444 666 555
Function Finished
```

2. Shell函数的参数与返回值

① 变量传递的两种方法

变量传递有两种方法：其一为变量直接传递法，数量不限，如上文show.sh中变量a、b、c。其二为位置参数法，数量最多 9 个，如上文show.sh中$1、$2、$3。

② 函数返回值

函数执行到最后一条语句后就会退出，但也可以主动调用return语句来实现提前退出。return语句的使用方式相对简单，原型为：return n，其中n的值可以主动指定，若采用默认的方式，则退出值为最近一个命令的退出码。

带格式的：项目符号和编号

1.8 I/O重定向

1. Linux文件描述

“一切皆是文件”是 Unix/Linux的基本设计哲学之一。不仅普通的文件，

目录、字符设备、块设备、套接字等在Unix/Linux中都是以文件被对待。它们虽然类型不同，但是对其提供的却是同一套操作界面。

在Linux中每一个进程都由task_struct（参见图 12-4 进程task_struct文件结构）数据结构来定义。task_struct数据结构的files选项指向打开文件描述符。每一个进程默认打开三个文件，这三个文件对应的文件指针是files_struct数据结构中的fd[0]、fd[1]、fd[2]，对应文件描述符为 0、1、2，分别指向标准输入（键盘）、标准输出（屏幕）、标准错误（屏幕）。

在Linux系统，文件描述符(File Descriptor)可以用一个数字来表示。表 3-7 列出了进程默认打开的文件描述符表。

表 3-7　进程默认打开文件描述符表

文件描述符	名称	常用缩写	默认指向
0	标准输入	stdin	键盘
1	标准输出	stdout	屏幕
2	标准错误输出	stderr	屏幕

2. 基本I/O重定向

I/O重定向表示重新定位数据的流向，下面说明的是常见的I/O重定向方法。

cmd > file：把 stdout 重定向到file文件中。
cmd >> file：把 stdout 重定向到 file 文件中(追加)。
cmd 1> file：把 stdout 重定向到 file文件中。
cmd > file 2>&1：把 stdout 和 stderr 一起重定向到 file 文件中。
cmd 2> file：把 stderr 重定向到 file 文件中。
cmd 2>> file：把 stderr 重定向到 file 文件中(追加)。
cmd >> file 2>&1：把stdout和stderr 一起重定向到 file 文件中(追加)。
cmd <file>file2：cmd命令以file文件作为stdin，以file2 文件作为stdout。
cmd < file：以file文件作为 stdin。
cmd << delimiter：从stdin中读入，直至遇到delimiter分界符。
cmd <<- delimiter：从stdin中读入，直至遇到delimiter分界符，输入去掉行首tab键。

3. 高级I/O重定向

下面说明复杂I/O重定向的方法，较少使用，了解即可。

>&n：使用系统调用dup(2)复制文件描述符n，并把结果用作标准输出。
<&n：标准输入复制自文件描述符n。
<&-：关闭标准输入（键盘）。
>&-：关闭标准输出。
n<&-：表示将文件描述符n输入关闭。
n>&-：表示将文件描述符n输出关闭。
cmd 2>file：运行一个命令并把错误输出(文件描述符 2)定向到file。
cmd 2>&1：运行一个命令并把它的标准输出和输入合并(严格的说是通过复制文件描述符 1 来建立文件描述符 2 ，但效果通常是合并了两个流)。
exec 1>outfilename：打开文件outfilename作为stdout。
exec 2>errfilename：打开文件errfilename作为stderr。
exec 0<&-：关闭 fd0。
exec 1>&-：关闭 fd1。

exec 5>&-：关闭 fd5。

带格式的：项目符号和编号

1.9 Shell内建命令

1. 内建命令列表

表 3-8 列出了Shell的内建命令及其说明，Shell的内建命令是在Shell脚本中能使用的命令。

表 3-8 Shell内建命令列表

命令	含义
:	空命令，返回退出状态零
.	在当前进程的环境下执行程序
break	跳出最内层的循环
break [n]	循环控制命令
alias	为存在的命令列出并创建别名
bg	将一个作业放到后台
bind	显示当前键和函数的绑定，或将键和一个readline函数或宏绑定
echo [args]	显示用换行符终止的参数
enable	开启和关闭Shell内置命令
eval [args]	读参数作为Shell的输入，并执行产生的命令
exec command	执行命令来取代当前的Shell
exit [n]	以状态n退出Shell
export [var]	使var能被子Shell识别
fc	用于编辑历史命令的历史编辑命令
fg	将后台作业放到前台
getopts	解析并处理命令行选项
hash	控制内部哈希表以更快地搜索命令
help [command]	显示关于内置命令的帮助信息，如果指定命令，将显示该内置命令的详细帮助
history	显示带行号的历史清单
jobs	列出放在后台的作业
kill [-singal process]	发送信号给指定PID号或作业号的进程
getopts	用于Shell脚本以解析命令行并检查合法的选项
let	用来对算术表达式求值并将算术计算的结果赋给变量
local	用在函数中以限制变量在函数中的作用域
logout	退出登录Shell
popd	从目录栈中删除项
pushd	往目录栈中添加项
pwd	显示当前工作目录
read [var]	从标准输入读取一行到变量var
readonly [var]	使变量var只读。不能被复位
return [n]	从一个函数返回，n是返回的退出值
set	设置选项和位置参量
shift [n]	向左移动位置参量n次
stop pid	终止PID号进程的执行
suspend	暂停当前Shell的执行(如果是一个登录Shell就不暂停)
test	检查文件类型且测试条件表达式
times	为从该Shell运行的进程显示所累积的用户和系统时间
trap [arg] [n]	当Shell接收到信号n(0、1、2 或 15)时执行参数

PDF-XChange Click to buy NOW! www.docu-track.com

命令	含义
type [command]	打印命令的类型。例如，pwd是一个内置Shell命令
typeset	和declare一样。设置变量并给它们属性
ulimit	显示并设置进程资源限度
umask [octal digits]	设置创建文件时关于文件属主、属组和其他用户执行权限的掩码
unalias	删除别名
unset [name]	删除变量值或函数
wait [pid#n]	等待后台PID号为n的进程返回并报告终止状态
date	显示时间和时间设置

2. trap命令

trap命令用于指定在接收到信号后将要采取的行动，trap命令的一种常见用途是忽略某些信号或在脚本程序被信号中断时完成清理工作。历史上，Shell总是使用数字来代表信号；现在提倡使用信号的名字，使用信号名时需要省略SIG前缀。在命令提示符下输入命令trap –l可以查看信号编号及其关联的名称。

“信号”是指那些被异步发送到一个程序的事件。默认情况下，它们通常会终止一个程序的运行。

trap命令使用格式如下：

```
trap 'command' signal-list
```

其中trap命令的参数分为两部分，前一部分是接收到指定信号时将要采取的行动，后一部分是要处理的信号名。

（1）trap捕捉到信号之后，可以有三种反应方式

① 执行一段程序来处理这一信号。

② 接受信号的默认操作。

③ 忽视这一信号。

（2）trap对上面三种方式提供了三种基本形式

① 设置信号的处理方式，使用第一种形式

```
trap 'command' signal-list
trap "command" signal-list
```

② 恢复信号的默认操作，使用第二种形式

```
trap signal-list
```

③ 忽略信号，使用第三种形式

```
trap " " signal-list
```

在第一种形式trap命令中Shell接收到signal-list清单中数值相同的信号时，将执行引号中的命令串。

使用trap时有如下注意事项：

① 对信号 11(段违例)不能捕捉，因为Shell本身需要捕捉该信号去进行内存的转储。

② 在捕捉到signal-list中指定的信号并执行完相应的命令之后， 如果这些命令没有将Shell程序终止的话，Shell程序将继续执行收到信号时所执行的命令后面命令，这样将很容易导致Shell程序无法终止。

③ 在trap语句中，单引号和双引号是不同的，当Shell程序第一次碰到trap语句时，将把command中的命令扫描一遍，此时若command是用单引号引起来的话，那么Shell不会对command中的变量和命令进行替换。

表 3-9 列出了能被trap命令捕获的比较重要信号列表及其说明。

表 3-9 能够被捕获的比较重要信号列表

命令	含义
HUP(1)	挂起，通常因终端掉线或用户退出而引发
INT(2)	中断，通常因按下Ctrl+c 组合键而引发
QUIT(3)	退出，通常因按下Ctrl+\ 组合键而引发
ABRT(6)	中止，通常因某些严重的执行错误而引发
ALRM(14)	报警，通常用来处理超时
TERM(15)	终止，通常在系统关机时发送

通常需要忽略的信号有四个，即HUP、INT、QUIT、 TSTP，也就是信号 1、 2、3、24，使用下面的语句可以使这些信号被忽略。

```
trap "" 1 2 3 24 或 trap "" HUP INT QUIT TSTP
```

3. date命令

date命令的功能是显示和设置系统日期和时间。

date命令语法形式如下，查看时间时格式需要带+号。

```
date [选项] [+格式]
```

date常见的选项说明如下：

```
-d datestr, --date datestr 显示由datestr描述的日期。
-s datestr, --set datestr 设置datestr 描述的日期。
-u, --universal 显示或设置通用时间。
```

表 3-10 列出date设置和显示时间时格式选项的种类及其说明。这里需要说明的是，只有超级用户才能用date命令设置时间，一般用户只能用date查看时间。

表 3-10 date时间格式说明表

格式	说明
%H	小时（00..23）
%I	小时（01..12）
%k	小时（0..23）
%l	小时（1..12）
%M	分（00..59）
%p	显示出AM或PM
%r	时间（hh：mm：ss AM或PM），12 小时
%s	从 1970 年 1 月 1 日 00：00：00 到目前经历的秒数
%S	秒（00..59）
%T	时间（24 小时制）（hh:mm:ss）
%X	显示时间的格式（%H:%M:%S）
%Z	时区 日期域
%a	星期几的简称（ Sun..Sat）
%A	星期几的全称（ Sunday..Saturday）
%b	月的简称（Jan..Dec）
%B	月的全称（January..December）
%c	日期和时间（ Mon Nov 8 14：12：46 CST 1999）
%d	一个月的第几天（01..31）
%D	日期（mm / dd / yy）
%h	和%b选项相同
%j	一年的第几天（001..366）
%m	月（01..12）
%w	一个星期的第几天（0 代表星期天）
%W	一年的第几个星期（00..53，星期一为第一天）
%x	显示日期的格式（mm/dd/yy）
%y	年的最后两个数字（ 1999 则是 99）

PDF-XChange Click to buy NOW! www.docu-track.com

格式	说明
%Y	年（例如：1970，1996 等）

例 1：用指定的格式显示时间

```
$ date ‘+This date now is =>%x , time is now =>%X , thank you !'
This date now is =>11/12/99 , time is now =>17:53:01 , thank you !
```

例 2：用默认格式显示当前的时间

```
# date
Fri Nov 26 15：20：18 CST 1999
```

例 3：设置时间为下午 14 点 36 分

```
# date -s 14:36:00
Fri Nov 26 14：15：00 CST 1999
```

例 4：设置时间为 1999 年 11 月 28 号

```
# date -s 991128
Sun Nov 28 00：00：00 CST 1999
```

例 5：按格式显示下一天的时间

```
# date -d next-day +%Y%m%d
20060328
```

带格式的：项目符号和编号

1.10 实用Shell脚本

下面三个shell脚本是作者从业中经常使用到的脚本。file_mod.sh是去掉文件中的回车符，当Windows下的文本文件传到Linux下时，文件每行会多一个回车符，利用file_mod.sh可以批量修改文件，去掉文件尾的回车符。stopproc.sh脚本是根据进程名杀死一个或多个进程的脚本，此脚本相当于实现Linux下的pkill命令的功能。backup.sh脚本是实用数据备份脚本，这是一个技术含金量很高而且非常实用的脚本，此脚本实现了按需备份（排除不需要备份的文件）和自动ftp传输功能。

1. 去掉文件中回车符

【例 3-23】file_mod.sh用例

```
file="aa.txt aaa.txt"
echo $file

#  ^[表示ESC键，需要先输ctrl+v，再按ESC键
# ^M表示回车，需要先输ctrl+v，再按M键
for filename in $file
do
    echo $filename
    vi $filename<<-EOF
    :%s/aaa/AAA/g^M^[
    :wq^M^[
    EOF
done
```

2. 根据进程名kill进程

【例 3-24】stopproc.sh用例

```
#!/bin/sh

if [ $# -lt 1 ]
then
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
        echo "usage ./simstop.sh proc_name"
            exit 1
    fi

    PROCESS=`ps -ef|grep $1|grep -v grep|grep -v PPID|awk '{ print $2}'`
    for i in $PROCESS
    do
        echo "Kill the $1 process [ $i ]"
        kill  -9 $i
    done
```

3. 实用数据备份脚本

【例 3-25】backup.sh用例

```
    #!/bin/sh
    FAPWORKDIR=/home/bep

    cd ${FAPWORKDIR}
    DATE=`date +"%Y%m%d"`

    rm -f backup/bep.$DATE.tar  backup/exclude.list

    # 产生排除文件列表
    # *.tar *.Z *.gz *.rar *.o .* 等文件都不备份
    find -L include -name '*.tar' -o -name '*.Z' -o -name '*.gz' -o -name '*.rar' -o
     -name '.*' >> backup/exclude.list
    find -L src     -name '*.tar' -o -name '*.Z' -o -name '*.gz' -o -name '*.rar' -o
     -name '.*' -o -name '*.o' >> backup/exclude.list

    tar -chvf backup/bep.$DATE.tar -X backup/exclude.list .profile  include  src scr
    ipts lib
    compress -F backup/bep.$DATE.tar

    ftp -n 192.168.31.125 <<!
    user bepbak bepbak
    lcd backup
    cd 110_bak
    bin
    put bep.$DATE.tar.Z
    bye
    !

    rm -f backup/bep.$DATE.tar backup/bep.$DATE.tar.Z backup/exclude.list
```