《Linux命令行与shell脚本编程大全》重要知识点简单总结

第1章 初识Linux shell

内核：Linux系统核心，控制所有软硬件。具有四种功能：

1. 系统内存管理：管理可用物理内存+使用**交换空间**创建并管理虚拟内存。
2. 软件程序管理：运行中的程序称为进程，内核通过创建init进程来控制管理其他所有进程。现多使用systemd版的init进程。相关命令如下：

```
runlevel # 查看当前运行级
systemctl get-default # 查看当前默认目标（目标定义Linux系统特定运行状态）
```

3. 硬件设备管理：任何Linux系统需要与之通信的设备都必须在内核代码中加入其驱动程序。Linux将硬件设备视为特殊的文件，称为设备文件，分为字符设备文件、块设备文件和网络设备文件。
4. 文件系统管理：Linux内核支持通过不同类型的文件系统读写硬盘数据。

GNU实用工具：该工具中的命令行核心工具称为coreutils软件包，包含文件实用工具，文本实用工具和进程实用工具。

GNU/Linux shell：启动程序、管理文件、运行进程。所有Linux发行版默认的shell都是bash shell。

```
cat /etc/shells # 查看系统中安装的shell
chsh -s shell路径 # 更改默认shell
```

Linux内核、GNU工具、图形化桌面环境和应用软件是构成完整Linux系统所需的4个关键组件，具备这些组件的Linux系统包称为**发行版**。

第2章 走进shell

虚拟控制台可以通过ctrl+alt+功能键打开，在kali linux中为f1到f7，代表第1个到第7个控制台

```
tty # 查看当前虚拟控制台序号
setterm --inversescreen on # 反转shell背景色和前景色，变成白底黑字
setterm --background COLOR # 设置背景色
setterm --foreground COLOR # 设置前景色
```

第3-5章 bash shell命令

管理文件和目录

命令	描述	相关
man CMD	获取特定命令的帮助文档	Enter键逐行查看，空格键翻页，按q退出。 可选项-k：使用关键字查找 手册页有不同的节，从1到9，通常显示编号最低的节
pwd	获取当前工作目录	
ls	获取文件列表	可选项-F：区分文件和目录。目录名用“/”表示，可执行文件用“*”表示 可选项-a：列出所有文件，包括隐藏文件 可选项-R：递归选项，列出当前目录下的子目录的文件 可选项-l：产生长列表格式的输出 可选项-d：仅查看目录本身，不查看其中内容 可选项-i：查看指定文件的inode编号，该编号是文件的唯一标识 可选项-d：只显示目录，不显示目录内容 可选项-sh：以人类易读的格式显示文件大小 可选项-g：在文件长列表中不显示文件属主 可选项-o：在文件长列表中不显示文件属组 可以使用通配符匹配，常用的包括?、*、!、[]等
touch FILENAME	修改已有文件的日期和时间（或创建不存在的空文件，初始大小为0）	
cp SRC DES	复制文件到指定位置	可选项-i：强制shell询问是否需要覆盖已有文件 在参数为文件夹时需要在末尾添加“/” 可选项-R：递归复制整个目录的内容，DES参数中的文件夹如果不存在则自动创建
ln -s	创建软链接（符号链接）	软链接是一个独立的文件，该文件指向另一个文件，彼此内容并不相同；而硬链接创建的文件和源文件是同一个文件 只能对处于同一存储设备的文件创建硬链接，否则必须使用符号链接

命令	描述	相关
mv SRC DES	将文件和目录移动到指定位置或重新命名	mv只影响文件名，不会影响inode编号和时间戳 可选项-i：强制shell询问是否需要覆盖已有文件
rm FILENAME	删除指定文件	可选项-i：强制shell询问是否需要删除该文件 可选项-f：强制删除文件 可选项-r：删除目录中的文件，再删除目录本身 可选项-rf：直接删除指定目录及其所有内容
mkdir DIR	创建指定目录	可选项-p：批量创建目录和其下的子目录
rmdir DIR	删除指定目录	该命令默认只删除空目录，要删除非空目录须使用rm -r
file FILENAME	探测文件内部并判断文件类型	对软链接，该命令可以说明链接到的目标文件 对二进制可执行程序，该命令可以确定其编译时所面向的平台以及需要何种类型的库
cat FILENAME	显示文本文件中所有数据	可选项-n：给所有行（包括空行）加上行号 可选项-b：只给非空行加上行号
more FILENAME	显示文本文件中所有数据，但会在显示每页数据后暂停	
less FILENAME	显示文本文件中所有数据，但会在显示每页数据后暂停	该命令相较于more拥有更多特性，并且可以在完成整个文件的读取之前显示文件的内容
tail FILENAME	显示文件最后几行的内容，默认为10行	可选项-n：修改所显示的行数，形式为-n NUM或-NUM 可选项-f：允许在其他进程使用此文件时查看并动态修改文件内容，常用于实时监测系统日志
head FILENAME	显示文件开头几行的内容，默认为10行	可选项-n：修改所显示的行数，形式为-n NUM或-NUM
locate	查找文件	
updatedb	更新系统数据库	

命令	描述	相关
`which`	获取查找程序位置	只显示外部命令文件
`find`	高级递归查找文件	命令格式`find / -name foo -print`

进程、命令管理

命令	描述	相关
`ps`	获取静态进程列表	`ps`命令有3种类型：Unix风格、BSD风格和GNU风格 可选项`-ef`：查看系统中运行的所有进程（`-e`指定显示所有进程，`-f`扩充输出内容） 可选项`-l`：产生长格式输出 可选项`--forest`：以树状查看进程 可选项`-e`或`-A`：显示所有进程
`top`	获取动态进程列表	`top`命令运行时， 键入`f`：选择用于对输出进行排序的字段（默认为%CPU），同时也能查看各字段全称 键入`d`：修改轮询间隔 键入`q`：退出`top`
`kill PID`	向进程发送信号，默认值为15（TERM信号）	可选项`-s`：支持指定其他信号（用信号名或信号值） 可选项`-l`：获取所有信号和数字值 可选项`-#`：向特定进程发送特定信号
`killall PID/NAME`	向进程发送信号，可使用进程名称	
`pkill NAME`	使用程序名代替PID来终止进程	允许使用通配符
`echo $0`	显示当前shell的名称	如果该命令在shell脚本内运行，则会显示脚本名称
`echo $SHELLNAME_SUBSHELL`	显示是否生成子shell	其中的SHELLNAME需要替换成当前shell，例如当前为zsh，则命令为`echo $ZSH_SUBSHELL` 如果返回0则没有子shell，非0则存在子shell

命令	描述	相关
sleep SEC	希望进程睡眠相应秒数	可选项&：紧跟在SEC参数后，表示将命令置入后台模式
jobs	显示当前运行在后台模式中属于当前用户的所有进程（作业）	可选项-l：查看更多相关信息 如果有多个后台进程在运行，最近启动的作业在其作业号之后会有一个加号（+），在它之前启动则以减号（-）显示。带有加号的作业为默认作业
coproc CMD	在后台生成一个子shell并在其中执行命令	该命令会默认将进程命名为COPROC。 使用扩展语法可以自定义名称：coproc NAME { CMD; }
type CMD	查询某个命令是否为内建命令	可选项-a：查看同一种命令的不同实现
history	显示最近用过的命令列表	通过修改HISTSIZE环境变量可以设置保存在历史记录中的命令数量，默认为1000 可选项-a：在不退出shell的情况下强制将命令历史记录写入.SHELL_history文件 可选项-n：强制重新读取.SHELL_history文件，更新内存中的终端会话历史记录 可选项-c：清除命令历史，与-a结合使用可以清除.SHELL_history文件
!!	唤回并重用最近的命令	
!NUM	唤回并重用指定编号的命令	
alias NAME='CMD'	允许为常用命令及其参数创建别名	可选项-p：查看当前可用的别名
unalias NAME	删除指定别名	如果被删除的别名不是手动设置的，那么下次重新登录系统就会再次出现。可以通过修改环境文件永久删除某个别名
apropos CMD	获取特定命令的相关命令	

echo命令的可选项-n表示去掉换行符。

监测磁盘空间

命令	描述	相关
mount	挂载存储设备	可选项-t：指定文件系统类型并过滤输出结果 手动挂载命令：mount -t type device directory，但这种方法只能实现临时挂载。要强制Linux在启动时自动挂载文件系统，可以将其添加到/etc/fstab文件中 可选项-o：允许在挂载文件系统时添加一系列以逗号分隔的额外选项
umount [dir/device]	卸载存储设备	
df	查看所有已挂载磁盘的使用情况	可选项-t：指定文件系统类型并过滤输出结果 可选项-h：以人类易读格式显示磁盘空间
du	显示某个特定目录（默认是当前目录）的磁盘使用情况	可选项-c：显示所有已列出文件的总大小 可选项-h：按人类易读格式输出大小 可选项-s：输出每个参数的汇总信息

处理数据文件

命令	描述	相关
sort	依据默认语言的排序规则对数据进行排序	可选项-n：将数字按值排序 可选项-M：将数字按月份排序 可选项-t x -k NUM：-t指定分隔符，-k指定排序字段序号
grep STR FILENAME	在指定文件中逐行搜索字符串	可选项-v：反向搜索（不匹配指定模式的行） 可选项-n：显示匹配指定模式的行号 可选项-c：输出匹配指定模式的总行数 可选项-E STR1 -E STR2：指定多个匹配模式 可选项-r：递归搜索所有目录下的特定字符串 grep默认使用Unix风格正则表达式来匹配模式 egrep是grep的衍生，支持POSIX扩展正则表达式 fgrep支持将匹配模式指定为以换行符为分隔的一系列固定长度的字符串

命令	描述	相关
gzip	压缩指定文件	也可指定多个文件名或用通配符来一次性压缩多个文件
gunzip	解压指定文件	
tar	打包、压缩、归档文件，也可以将输出写入文件	可选项-cvf TAR NAME1 NAME2：创建归档文件（-c），包括指定的文件或目录 可选项-tf TAR：列出但不提取指定tar文件的内容 可选项-xvf TAR：从tar文件中提取内容 可选项-zxvf FILENAME.tgz：提取经gzip压缩过的tar文件

根驱动器是Linux安装的第一块硬盘，Linux使用根驱动器上一些特别的目录作为**挂载点**。挂载点是虚拟目录中分配给额外存储设备的目录，即使是其他物理驱动器中的文件和目录也会出现在这些挂载点目录中。

外部命令，有时也称文件系统命令，是存在于bash shell之外的程序，不属于shell程序的一部分，例如ps命令。执行外部命令会创建一个子进程，这种操作称为**衍生（forking）**。因此，外部命令系统开销较高。**内建命令**不需要使用子进程来执行，开销较低，例如cd和exit命令。有的命令既有内建命令也有外部命令，例如echo和pwd命令。针对两种实现的命令，使用其外部命令形式时输入其完整路径，使用其内建命令形式时直接输入命令。

第6章 Linux环境变量

命令	描述	相关
env或printenv	查看全局变量	printenv KEY命令可以显示个别环境变量的值，env没有这个用法，但env KEY=VALUE可以为新的环境变量设置值。两者都不会对变量结果进行排序
echo $KEY	查看某个环境变量的值	$KEY作用是引用该变量的值，并且它还可以作为其他命令的参数（类似C语言的指针），例如ls $KEY
set	显示特定进程的所有环境变量	同时按字母顺序对变量结果进行排序
key=value	创建仅对该shell进程可见的局部用户自定义变量	如果用于赋值的字符串包含空格，则必须用单引号或双引号来界定该字符串的起止。在变量名、等号和值之间没有空格

命令	描述	相关
`key=(value1 value2...)`	为某个环境变量设置多个值	要引用单个数组元素必须使用表示在其数组位置的索引，索引通常以0开始（但在zsh中以1开始） 显示整个数组变量可以用通配符*作为索引：`echo ${key[*]}` 改变某个索引的值：`key[index]=value` 删除数组中的某个值及其索引：`unset key[index]`，这个命令将值与其索引一并删去，因此该索引对应空值
`export key`	导出局部变量成为全局变量	可以将设置变量和导出变量合并为同一个命令：`export key=value`。子shell中导出的全局变量不能影响父shell中同名全局变量的值
`unset key`	删除已有的环境变量	在子进程中删除全局环境变量不会影响到父进程中的同名变量
`PATH=$PATH:path`	在PATH变量末尾添加新变量	

bash shell的惯例是所有环境变量均使用大写字母命名，用户自定义变量则用小写字母命名。

HISTFILESIZE和HISTSIZE变量的区别在于，前者是历史记录列表，位于内存中，在bash会话进行期间更新。后者是历史记录文件，位于硬盘上，通常是~/.bash_history，会话结束后，历史记录列表中的内容会被写入历史记录文件，写入命令数量受历史记录文件的限制。

PATH环境变量定义了用于查找命令和程序的目录。

创建自定义永久性全局变量或局部变量的方法是，在`/etc/profile.d`目录中创建一个以`.sh`结尾的文件，并把所有新的或修改过的全局环境变量设置都放在这个文件中。

保存个人用户永久性bash shell变量的最佳地点是`$HOME/.bashrc`文件，这适用于所有类型的shell进程。

数组变量可移植性低，在shell脚本编程时不常用。

第7章 Linux文件权限

用户管理

命令	描述	相关
`useradd`	创建用户并设置用户的`$HOME`结构	可选项`-D`：查看所使用的Linux发行版的系统默认值，使用`useradd -D -#`

命令	描述	相关
		还可更改默认值 可选项-m：创建$HOME目录
userdel	删除用户	默认只删除/etc/passwd和/etc/shadow文件中的用户信息，属于该账户的文件会被保留 可选项-r：删除用户的$HOME目录和邮件目录
usermod	修改用户账户信息	可选项-c：修改备注字段 可选项-e：修改过期日期 可选项-g：修改默认的登录组 可选项-G GROUP USERNAME：向组添加用户 可选项-l：修改用户账户的登录名 可选项-L：锁定账户，禁止用户登录 可选项-p：修改账号密码 可选项-U：解除锁定，恢复用户登录
passwd	修改用户密码	只有root用户才能修改其他用户的密码 可选项-e：强制用户下次登录时修改密码
chpasswd	修改一系列登录名和密码，自动加密并为用户账户设置密码	这些登录名-密码对以冒号分隔
chsh	修改默认的用户登录shell	必须用shell的全路径作为参数，不能只用shell名 可选项-s：指定新的shell名，若留空或不使用该可选项则指定当前登录shell
chfn	在/etc/passwd文件的备注字段中保存信息	使用finger命令查看Linux的用户信息
chage	管理用户账户的有效期	可选项-E：设置密码过期日期 可选项-I：设置密码过期多少天后锁定账户 可选项-m：设置更改密码的最小间隔天数 可选项-M：设置密码的最大有效天数
groupadd	新建组	
groupmod	修改已有组信息	可选项-g：修改已有组的UID 可选项-n：修改已有组的组名，保持UID不变
passwd	设置密码	

命令	描述	相关
passwd USERNAME	以root用户身份修改其他用户的密码	

Linux使用/etc/passwd来匹配登录名与对应的UID值，系统中各种运行服务进程都需要一个系统用户账户才能登录到Linux系统中。

/etc/shadow文件对Linux系统密码管理提供更多控制，只有root用户才能访问该文件。

组权限允许多个用户对系统对象共享一组权限。/etc/group文件包含系统中每个组的信息。组密码允许非组内成员使用密码临时成为该组成员。一个用户在/etc/passwd文件中指定某个组为主要组时，该用户不会作为该组成员再出现在/etc/group文件中。

文件权限

命令	描述	相关
umask (xxx)	显示和设置默认权限掩码	用全权限值（文件是666，目录和可执行文件是777）减去umask值得到实际的文件权限
chmod xxx FILENAME	修改文件权限	权限值设置也可使用第二种形式：[ugoa][+-=][rwxXstugo...]。 第一组字符中，u=user，g=group，o=others，a=all 第二组字符中，+表示增加权限，-表示移除权限，=表示设置权限 第三组字符中，X表示仅当对象是目录或者已有执行权限时才赋予执行权限，s表示在执行时设置SUID或SGID（其中SGID用于共享文件），t表示设置粘滞位，u表示设置属主权限，g表示设置属组权限，o表示设置其他用户权限 例如chmod o+r FILENAME表示为其他用户添加读取权限 可选项-R：以递归方式修改文件和目录的权限
chown NAME FILENAME	修改文件的属主	chown NAME.GROUP FILENAME修改文件的属主和属组； chown .GROUP FILENAME修改文件的属组 chown NAME. FILENAME在组名和用户登录名相同时可修改文件的属主和属组 可选项-R：递归修改子目录和文件的所属关系 可选项-h：修改文件的所有符号链接文件的所属关系 只有root用户能修改文件的属主。任何用户都可以修改文件的属组，但前提是该用户必须是原属组和新属组的成员
chgrp GROUP FILENAME	修改文件或目录的默认属组	

命令	描述	相关
`getfacl`	查看分配给文件或目录的ACL	ACL全称访问控制列表，允许指定包含多个用户或组的列表以及为其分配的权限。文件权限列末尾有加号（+）时表示该文件应用了ACL

`setfacl`	为用户或组分配权限	可选项`-m`：修改分配给文件或目录的权限
		可选项`-x`：删除特定权限
		语法：`u[ser]:uid:perms`
		`g[roup]:gid:perms`
		`o[ther]::perms`
		对于uid或gid，可以使用数字值或名称
		在规则定义前加上`d:`可设置目录的默认ACL

八进制的安全设置先获取rwx权限值，然后转为3位（bit）二进制值，用一个八进制值来表示。

第8章 管理文件系统

文件系统中，日志技术的替代选择是一种称作写时复制（copy-on-write, COW）的技术，通过快照兼顾了安全性和性能。

文件系统

命令	描述	相关
`fdisk`	在存储设备上创建和管理分区	只能处理最大2TB的硬盘，且不允许调整现有分区的大小，只能先删除现有分区再重建。该命令有自己的命令行可选项 可选项`d`：删除分区 可选项`n`：添加一个新分区 可选项`p`：显示当前分区表 可选项`q`：退出，不保存更改 可选项`w`：将分区表写入磁盘并退出
`gdisk`	采用GUID分区表创建和管理分区	`gdisk`与`fdisk`命令相似
`parted`	使用另一种命令行管理分区	GNU parted允许调整现有分区大小
`lsblk -f`	显示新近格式化过并挂载的分区	

命令	描述	相关
`fsck`	检查和修复大部分Linux文件系统类型	使用该命令前必须卸载设备 可选项`-t`：指定要检查的文件系统类型

一些发行版在创建好分区后不会通知Linux，此时需要使用`partprobe`命令或`hdparm`命令，或重启系统。

在将数据存储到分区之前必须使用某种文件系统对其进行格式化，以便Linux能够使用分区。随后将其挂载到虚拟目录中的某个挂载点，以便在新分区中存储数据。

逻辑卷管理器LVM

LVM的作用是在无须重建整个文件系统的前提下管理磁盘空间。它允许将多个分区组合在一起，作为单个分区（逻辑卷）进行处理。LVM主要由3个部分组成：**物理卷（PV），卷组（VG）和逻辑卷（LV）**。

命令	描述	相关
`pvcreate`	指定一个未使用的磁盘分区（或整个驱动器）由LVM使用	
`vgcreate`	将PV加入存储池，后者用于构建各种逻辑卷	被指定为PV的分区只能属于单个VG，但不同于该分区的其他分区可以属于其他VG
`lvcreate`	构建逻辑卷LV	LV由VG的存储空间块（physical extents）PE组成，可以挂载并作为普通磁盘分区使用。LV只能从一个指定的VG创建，但多个LV可以共享单个VG 可选项`-L`：设置LV的大小
`lvdisplay`	显示LV相关信息	LV需要用绝对路径表示 也可用`lvs`或`lvscan`命令
`vgextend`	将PV加入VG	
`vgreduce`	从VG中删除PV	
`lvextend`	扩大LV的容量	
`lvreduce`	收缩LV的容量	
`lvm help`	显示所有的LVM命令	

第9章 安装软件

软件包管理（基于Debian系统）

命令	描述	相关
`apt list`	显示仓库中所有可用的软件包	可选项`--installed`：仅输出已安装在系统中的软件包
`apt show PKG`	显示特定软件包的详细信息	不会指明软件包是否已经安装
`apt search`	查找特定的软件包	该命令自带通配符效果，不需要添加通配符 可选项`--name-only`：只搜索软件包名称
`apt update`	更新软件包数据库	
`apt upgrade`	更新系统中所有软件包	该命令不会删除任何软件包。命令`apt full-upgrade`表示先删除某个软件包再升级
`apt upgrade PKG`	更新特定软件包	
`apt install PKG`	下载特定软件包	
`apt remove PKG`	移除特定软件包	该命令会保留数据和配置文件。命令`apt purge`删除软件包以及所有的相关的数据和配置文件
`apt autoremove`	移除不再需要的软件包	
`dpkg`	管理软件包	
`dpkg -l`	获取所有已安装软件包列表	
`dpkg -L PKG`	列出与特定软件包相关的所有文件	
`dpkg --search FILENAME`	找出特定文件所属的软件包	文件要使用绝对路径

命令	描述	相关
`dpkg -i PACKAGENAME`	下载特定.deb软件包	
`dpkg -r PACKAGENAME`	删除特定软件包	
`dpkg -c FILENAME`	查看软件包内容（FILENAME是完整名称）	
`dpkg --force-install`	强制安装软件包	

软件包存储在称为仓库（repository）的服务器上，仓库位置保存在`/etc/apt/sources.list`。在该文件中，指定仓库源的格式为：`deb(or deb-src address distribution_name PKG_type_list)`。`deb`或`deb-src`指定软件包的类型，`deb`表明这是一个已编译程序的仓库源，而`deb-src`表明这是一个源代码的仓库源。`address`是软件仓库的网址，`distribution_name`是该软件仓库的发行版的版本名称。`PKG_type_list`表明仓库里有什么类型的软件包，可能不止一个单词。

软件包管理（基于Red Hat系统）

命令	描述	相关
`dnf list installed`	列出系统中已安装的软件包	可以使用`> FILENAME`运算符将输出结果重定向到文件中
`dnf provides FILENAME`	找出安装该文件的软件包	
`dnf install PKG`	安装特定软件包	
`dnf list upgrades`	查看已安装软件包的所有可用更新	
`dnf upgrade`	升级系统中所有软件包	添加`PKG`参数表示升级特定软件包。命令`dnf upgrade-minimal`将软件包升级到最新的bug修复版或安全补丁版，而不是最新的最高版本

命令	描述	相关
`dnf remove PKG`	卸载软件包	
`dnf clean all`	处理损坏的依赖关系	也可使用`dnf repoquery --deplist PKG`命令，该命令显示软件包的所有依赖关系以及哪些软件包提供了这些依赖
`dnf repolist`	查看当前拉取软件的仓库	`dnf`仓库的定义存在于配置文件`/etc/dnf/dnf.conf`和`/etc/yum.repos.d`目录中的单独文件

应用程序容器：snap和flatpak

应用程序容器能够提供应用程序运行所需的全部文件，使得开发人员可以将容器作为单个软件包分发，保证能够在任何Linux系统中运行。

命令	描述	相关
`snap version`	检查snap是否正在运行	首次安装snap时需要使用命令`systemctl enable snapd.socket`和`systemctl start snapd.socket`启动snap服务
`snap list`	查看当前已安装的snap应用程序列表	
`snap list NAME`	在snap仓库中搜索指定程序	
`snap info NAME`	查看snap应用程序（简称snap）的详细信息	
`snap install`	安装新的snap	安装时必须有root权限
`snap remove NAME`	删除snap	命令`snap disable`可以禁用snap，`snap enable`可以恢复snap
`flatpak list`	列出已安装的应用程序容器	
`flatpak search NAME`	搜索指定应用程序	

命令	描述	相关
`flatpak install NAME`	安装指定应用程序	
`flatpak uninstall`	删除应用程序容器	

源代码形式的软件包通常以tarball形式发布，可以用`tar`命令行命令创建和解包tarball。

第10章 文本编辑器

此处只列出vim。

vim编辑器快捷键

快捷键	描述	相关
`h`	左移一个字符	命令模式
`j`	下移一行	命令模式
`k`	上移一行	命令模式
`l`	右移一个字符	命令模式
`PageDown`或`Ctrl+F`	下翻一屏	命令模式
`PageUp`或`Ctrl+B`	上翻一屏	命令模式
`G`	移到缓冲区最后一行	命令模式
`NUM G`	移到缓冲区中的第NUM行	命令模式
`gg`	移到缓冲区的第一行	命令模式
`:`	进入vim的Ex模式	Ex模式提供一个交互式命令行，可以输入额外的命令来控制vim的操作
`q`	如果未修改缓冲区数据则退出vim	Ex模式

快捷键	描述	相关
q!	不保存修改并退出vim	Ex模式
w FILENAME	将文件另存为其他名称	Ex模式
wq	保存修改并退出	Ex模式

第11章 构建基础脚本

在CLI中，可以在多个命令之间使用分号分隔符，使之一起运行。

在创建shell脚本文件时，必须在文件的第一行指定要使用的shell，格式如：`#!/bin/bash`。可以在文件的各行输入命令，每行末尾加一个换行符。可以使用`sh FILENAME`直接运行shell脚本。脚本中，可以在环境变量名前加上`$`或使用`${variable}`的形式来引用这些环境变量。

在shell脚本中，可以使用**反引号**`` `CMD` ``或`$(CMD)`格式将命令输出赋给变量。在`date`命令后添加`+%y%m%d`格式会将日期显示为两位数的年、月、日组合。该方法可以用于提取日期信息，生成日志文件名等。

脚本重定向命令

命令	描述	相关
>	将命令输出发送至文件	
>>	在文件末尾追加数据	
<	将文件内容作为输入发送至命令	
<<	内联输入重定向符	无须使用文件进行重定向，只需在命令行中指定用于输入重定向的数据
wc	统计数据中的文本	默认输出3个值，分别代表文本的行数、文本的单词数和文本的字节数
|	将左侧的输出作为输入发送到右侧	
;	一次性依次运行多个命令	
{CMD;}	运行命令分组，尾部以分号结尾	

命令	描述	相关
()	将命令列表成为进程列表（命令分组的一种）	

管理文件和目录

命令	描述	相关
man COMMAND	获取特定命令的帮助文档	Enter键逐行查看，空格键翻页，按q退出。 可选项-k：使用关键字查找 手册页有不同的节，从1到9，通常显示编号最低的节
pwd	获取当前工作目录	
ls	获取文件列表	可选项-F：区分文件和目录。目录名用“/”表示，可执行文件用“*”表示 可选项-a：列出所有文件，包括隐藏文件 可选项-R：递归选项，列出当前目录下的子目录的文件 可选项-l：产生长列表格式的输出 可选项-d：仅查看目录本身，不查看其中内容 可选项-i：查看指定文件的inode编号，该编号是文件的唯一标识 可以使用通配符匹配，常用的包括?、*、!、[]等
touch FILENAME	修改已有文件的日期和时间（或创建不存在的空文件，初始大小为0）	
cp SRC DES	复制文件到指定位置	可选项-i：强制shell询问是否需要覆盖已有文件 在参数为文件夹时需要在末尾添加“/” 可选项-R：递归复制整个目录的内容，DES参数中的文件夹如果不存在则自动创建
ln -s	创建软链接（符号链接）	软链接是一个独立的文件，该文件指向另一个文件，彼此内容并不相同；而硬链接创建的文件和源文件是同一个文件 只能对处于同一存储设备的文件创建硬链接，否则必须使用符号链接

命令	描述	相关
mv SRC DES	将文件和目录移动到指定位置或重新命名	mv只影响文件名，不会影响inode编号和时间戳 可选项-i：强制shell询问是否需要覆盖已有文件
rm FILENAME	删除指定文件	可选项-i：强制shell询问是否需要删除该文件 可选项-f：强制删除文件 可选项-r：删除目录中的文件，再删除目录本身 可选项-rf：直接删除指定目录及其所有内容
mkdir DIR	创建指定目录	可选项-p：批量创建目录和其下的子目录
rmdir DIR	删除指定目录	该命令默认只删除空目录，要删除非空目录须使用rm -r
file FILENAME	探测文件内部并判断文件类型	对软链接，该命令可以说明链接到的目标文件 对二进制可执行程序，该命令可以确定其编译时所面向的平台以及需要何种类型的库
cat FILENAME	显示文本文件中所有数据	可选项-n：给所有行（包括空行）加上行号 可选项-b：只给非空行加上行号
more FILENAME	显示文本文件中所有数据，但会在显示每页数据后暂停	
less FILENAME	显示文本文件中所有数据，但会在显示每页数据后暂停	该命令相较于more拥有更多特性，并且可以在完成整个文件的读取之前显示文件的内容
tail FILENAME	显示文件最后几行的内容，默认为10行	可选项-n：修改所显示的行数，形式为-n NUM或-NUM 可选项-f：允许在其他进程使用此文件时查看并动态修改文件内容，常用于实时监测系统日志
head FILENAME	显示文件开头几行的内容，默认为10行	可选项-n：修改所显示的行数，形式为-n NUM或-NUM
locate	查找文件	
updatedb	更新系统数据库	

命令	描述	相关
`which`	获取查找程序位置	只显示外部命令文件
`find`	高级递归查找文件（find / -name foo -print）	

进程、命令管理

命令	描述	相关
`ps`	获取静态进程列表	`ps`命令有3种类型：Unix风格、BSD风格和GNU风格 可选项`-ef`：查看系统中运行的所有进程（`-e`指定显示所有进程，`-f`扩充输出内容） 可选项`-l`：产生长格式输出 可选项`--forest`：以树状查看进程
`top`	获取动态进程列表	`top`命令运行时， 键入`f`：选择用于对输出进行排序的字段（默认为%CPU），同时也能查看各字段全称 键入`d`：修改轮询间隔 键入`q`：退出`top`
`kill PID`	向进程发送信号，默认值为15（TERM信号）	可选项`-s`：支持指定其他信号（用信号名或信号值） 可选项`-l`：获取所有信号和数字值 可选项`-#`：向特定进程发送特定信号
`killall PID/NAME`	向进程发送信号，可使用进程名称	
`pkill NAME`	使用程序名代替PID来终止进程	允许使用通配符
`echo $0`	显示当前shell的名称	如果该命令在shell脚本内运行，则会显示脚本名称
`echo $SHELLNAME_SUBSHELL`	显示是否生成子shell	其中的SHELLNAME需要替换成当前shell，例如当前为zsh，则命令为`echo $ZSH_SUBSHELL` 如果返回0则没有子shell，非0则存在子shell

命令	描述	相关
`sleep SEC`	希望进程睡眠相应秒数	可选项&：紧跟在SEC参数后，表示将命令置入后台模式
`jobs`	显示当前运行在后台模式中属于当前用户的所有进程（作业）	可选项-l：列出进程PID和作业号 可选项-n：只列出上次shell发出通知后状态发生改变的作业 可选项-p：只列出作业的PID 可选项-r：只列出运行中的作业 可选项-s：只列出已停止的作业 如果有多个后台进程在运行，最近启动的作业在其作业号之后会有一个加号（+），在它之前启动则以减号（-）显示。带有加号的是默认作业，只能存在一个；带减号的会在当前默认作业结束后成为下一个默认作业，也只能存在一个。
`coproc CMD`	在后台生成一个子shell并在其中执行命令	该命令会默认将进程命名为COPROC。使用扩展语法可以自定义名称：`coproc NAME { CMD; }`
`type CMD`	查询某个命令是否为内建命令	可选项-a：查看同一种命令的不同实现
`history`	显示最近用过的命令列表	通过修改HISTSIZE环境变量可以设置保存在历史记录中的命令数量，默认为1000 可选项-a：在不退出shell的情况下强制将命令历史记录写入.SHELL_history文件 可选项-n：强制重新读取.SHELL_history文件，更新内存中的终端会话历史记录 可选项-c：清除命令历史，与-a结合使用可以清除.SHELL_history文件
`!!`	唤回并重用最近的命令	
`!NUM`	唤回并重用指定编号的命令	
`alias NAME='CMD'`	允许为常用命令及其参数创建别名	可选项-p：查看当前可用的别名
`unalias NAME`	删除指定别名	如果被删除的别名不是手动设置的，那么下次重新登录系统就会再次出现。可以通过修改环境文件永久删除某个别名

命令	描述	相关
`apropos CMD`	获取特定命令的相关命令	

echo命令的可选项-n表示去掉换行符。

执行数学运算

命令	描述	相关
`expr`	在命令行中执行数学运算	早期命令，只能识别少量算术运算符和字符串运算符
`$[expression]`	用更简单的方法执行数学运算	同样在shell脚本内适用，但该方法不支持浮点运算
`bc`	访问bash计算器	支持浮点运算 可选项`-q`：不显示欢迎信息 可选项`scale=NUM`：设置小数保留位数，默认为0 在脚本中使用bc的格式是：`$(echo "options; expression" | bc)`
`echo $?`	查看程序退出状态码	为0表示成功结束，为正整数表示因错误结束。其中，127表示无效命令，1表示无效参数
`exit NUM`	指定一个退出状态码	也可使用变量作为参数，但最大值只能是255，当值超过255时shell自动通过模运算得到最终状态码

第12-13章 结构化命令

1. if-then语句

格式如下：

```
if CMD
then
    CMDs
fi
```

或是

```
if CMD; then
    CMDs
fi
```

在bash shell中，if语句后的命令如果成功执行，即退出状态码为0，则位于then部分的命令就会执行。fi语句表示if-then语句到此结束。

2. if-then-else和if-then-elif-then语句

格式如下：

```
if CMD
then
    CMDs
else
    CMDs
fi
```

当if语句中的命令返回非0状态码时，执行else语句中的命令。

还可使用if-then-elif-then语句将两个if-then语句相结合，但不能再使用else语句。如果elif语句返回0，则执行对应的then语句。格式如下：

```
if CMD
then
    CMDs
elif CMD2
then
    CMDs
fi
```

3. test命令和[]

test命令可以在if-then或if-then-else语句中测试不同的条件，如果test命令列出的条件成立，则返回退出状态码0。格式如下：

```
if test condition
then
    CMDs
else
    CMDs
fi
```

如果condition部分为空，则test返回非0状态码并执行else语句。

使用[]同样可以定义测试条件，格式为if [condition]。必须注意空格。

(1) 数值比较

bash shell中的条件测试只能用于处理**整数**。

参数	描述	相关
n1 `-eq` n2	检查n1是否等于n2	eq: equal to
n1 `-ge` n2	检查n1是否大于或等于n2	ge: greater than or equal to
n1 `-gt` n2	检查n1是否大于n2	gt: greater than
n1 `-le` n2	检查n1是否小于或等于n2	le: less than or equal to
n1 `-lt` n2	检查n1是否小于n2	lt: less than
n1 `-ne` n2	检查n1是否不等于n2	ne: not equal to

(2) 字符串比较

比较字符串实质上是比较字符的Unicode编码值。编码值更小的字符比编码值更大的字符小。由于<和>会被shell视为重定向符，因此必须转义。例如：str1 \> str2。另外，`sort`命令使用系统的语言环境设置中定义的顺序排序。对于英语，在字符串排序时，大写字母被视为大于小写字母，因此排在小写字母后。这点与`test`命令相反。

运算符	描述
str1 = str2	检查str1是否和str2相同
str1 != str2	检查str1是否和str2不同
str1 \< str2	检查str1是否小于str2
str1 \> str2	检查str1是否大于str2
`-n` str1	检查str1长度是否不为0
`-z` str1	检查str1长度是否为0

在检查字符串长度是否为0时，未定义的变量长度同样会被视为0。

(3) 文件比较

命令	描述
-d file	检查file是否存在且为目录
-e file	检查file是否存在
-f file	检查file是否存在且为文件
-r file	检查file是否存在且可读
-s file	检查file是否存在且非空
-w file	检查file是否存在且可写
-x file	检查file是否存在且可执行
-O file	检查file是否存在且属当前用户所有
-G file	检查file是否存在且默认组与当前用户相同
file1 -nt file2	检查file1是否比file2新（创建时间更晚）
file1 -ot file2	检查file1是否比file2旧（创建时间更早）

4. 复合条件测试

if-then语句允许使用布尔逻辑将测试条件组合起来。格式为[condition1] && [condition2]（表示AND，有0则0）和[condition1] || [condition2]（表示OR，有1则1）。

5. if-then高级特性

shell提供了可在if-then语句使用的3个高级特性。

1. 单括号，格式if (CMD)。允许在if语句中使用子shell。
2. 双括号，格式((CMD))。允许在比较过程中使用高级数学表达式，包括自增++、自减--、逻辑求反!、位求反~、幂运算**、左移<<、右移>>、位布尔&，|，逻辑布尔&&，||。双括号中的大于号不用转义。
3. 双方括号，格式[[expression]]。允许进行模式匹配，使用通配符或正则表达式来匹配字符串。

6. case命令

case命令采用列表格式将variable的值与pattern的值相比较，匹配则执行相应命令。格式如下：

```
case variable in
pattern1 | pattern2) CMDs1;;
pattern3) CMDs2;;
*) default CMDss;;
esac
```

7. for命令

for命令用于创建遍历一系列值的循环，格式如下：

```
for var in list
do
    CMDs
done
```

或是

```
for var in list; do
    CMDs
done
```

在迭代结束后，其中变量var的值在脚本剩余部分仍然有效。使用for state in $(cat $file)还可以直接读取文件中的值进行遍历，文本中的值应当各占一行。

IFS是内部字段分隔符，默认为空格、制表符、换行符。脚本语句IFS=''可将IFS的值临时设置为自定义的值。指定多个分隔符只需在赋值语句中将这些字符写在一起。在设置新值前，最好先保存其原值，例如使用语句IFS_OLD=$IFS。

for命令还能模仿C语言的for语句，格式：for ((var assignment; condition; iteration process))，例如for ((a = 1; a < 10; a++))。这种语句也可以迭代使用多个变量，循环会单独处理每个不同的变量。

可以在for循环的末尾，即done语句后添加处理其输出的语句，例如重定向到文件、排序等。

8. while命令

while的原理同C语言等编程语言，返回状态码为0时执行命令，不为0时结束循环。格式如下：

```
while test CMD
do
    CMDs
done
```

shell的while语句可以定义多个测试命令，只有最后一个测试命令的退出状态码会被用于决定是否结束循环。

for和while循环可以处理CSV文件，只需将IFS设为相应分隔符，将文件内容作为输入置于语句末，在语句开始使用read -r para1 para2命令即可。

9. until命令

until命令与while完全相反，只有返回状态码不为0时才会执行命令，为0时结束循环。格式如下：

```
until test CMD
do
    CMDs
done
```

until语句可以定义多个测试命令，只有最后一个测试命令的退出状态码会被用于决定是否结束循环。

10. break命令

跳出当前正在执行的循环。在嵌套循环中，内层的break命令只会跳出内层循环。使用命令break n可以指定要跳出的循环层级，默认n为1，即跳出当前循环。如果n=2则break会停止下一级的外层循环。

11. continue命令

continue命令可以提前中止某次循环，但不会结束整个循环。使用命令continue n可以指定要继续的循环层级。

第14章 处理用户输入

1. 位置参数和特殊变量

命令行参数允许运行脚本时在命令行中添加数据。shell会将所有命令行参数都指派给位置参数，其名称都是标准数字。$0对应脚本名，$1对应第一个命令行参数，以此类推直到$9。第10个参数开始需要在数字周围添加花括号，例如${10}。basename命令返回不带路径的脚本名，如basename $0。

shell加入的特殊变量$#含有脚本运行时携带的命令行参数的个数。但用在花括号中时必须换成!，即${!#}，表示最后一个位置变量。当命令行没有参数时，$#的值为0。（注：似乎在kali linux中不生效）

$*变量会将所有的命令行参数视为一个单词。$@变量将所有的命令行参数视为同一字符串中的多个独立的单词，可使用for命令遍历。

2. shift命令

shift命令根据命令行参数的相对位置进行移动，默认将每个位置的变量值向左移动1个位置，即$3的值移入$2，$2的值移入$1等，但$1的值会被删除（无法再次恢复），而$0的值不变。格式shift n表示一次性移动多个位置。

3. getopt和getopts命令

getopt命令可以接受一系列任意形式的命令行选项和参数，并自动将其转换成适当的格式。格式：getopt optSTR paras。optSTR字段定义有效的命令行选项字母，以及哪些选项字母需要参数值，在每个需要参数值的选项字母后加一个冒号，并自动插入双连字符来分隔命令行中额外的参数。例如：getopt ab:cd -a -b BValue -cd test1 test2。使用可选项-q可忽略错误消息。

getopts命令是getopt的扩展版本，每次只处理一个检测到的命令行参数，在处理完所有参数后退出并返回一个大于0的退出状态码。格式：getopts optSTR variable。在optSTR中最开头添加冒号可以不显示错误消息。getopts命令要哦用到两个环境变量，如getopts :ab:c opt。其中opt，或OPTARG用于保存选项所带的参数值。在解析命令行选项，如-a，-b时，getopts会移除起始的连字符，所以在case判断语句中不用连字符。还可以不在选项与参数之间添加空格，直接写在一起。optSTR中未定义的选项字母会以问号形式传给脚本。处理完选项后，可以使用shift命令和OPTIND值来移动参数，如shift $ [$OPTIND - 1]。

某些Linux选项已经具有标准含义，添加这些选项可以更具用户友好性。

4. read命令

read命令从标准输入（键盘）或另一个文件描述符中接受输入。获取输入后将数据存入变量。格式read var(s)。

可选项-p允许直接指定提示符，如read -p MSG var(s)，将两个语句合为一句。可指定多个变量，也可不指定变量。不指定变量时，read会将接收到的所有数据都放进特殊变量REPLY中。

可选项-t指定输入计时器，单位为秒。超时后read命令返回非0退出状态码。如read -t seconds -p MSG var。

可选项-s避免read输入的数据出现在屏幕上。

（注：在某些shell中不支持read命令的交互选项，此时需要使用bash命令。）

read命令可以用来读取文件，一次读一行。通常方法是用cat命令获取内容后通过管道传给含有read命令的while命令。

第15章 呈现数据

1. 3个默认文件描述符

Linux用文件描述符来标识整个文件对象。bash shell保留了3个文件描述符（0、1和2），分别对应STDIN（标准输入），STDOUT（标准输出）和STDERR（标准错误）。STDIN对终端来说就是键盘，也可以使用重定向符<将输入重定向到指定的文件。常用命令如cat。STDOUT对终端来说就是终端显示器，也可以使用重定向符>将输出重定向到指定的文件，或是使用>>将数据追加到某个文件。但STDOUT并不会将错误信息重定向到文件。STDERR代表shell的标准错误输出，在默认情况下与STDOUT指向的位置一样，但可以对其进行重定向。一种是只重定向错误，使用重定向符>即可。要注意>必须紧挨着STDERR的文件描述符2，即2>。另一种是重定向错误消息和正常输出。如果要分别重定向到不同文件，则要使用**两个重定向符**，并在其之前放上相应的描述符，如`ls -la test badtest 2>err_file 1>stdout_file`。如果只需要重定向到同一个文件，则使用重定向符&>。

将一条自定义的错误消息重定向到STDERR可以使用>&2。

exec命令会启动一个新shell并将STDOUT文件描述符重定向到指定文件，将脚本中送往STDOUT的所有输出重定向，如`exec 2>testerror`。同样可以重定向STDIN到其他位置，如`exec 0<testfile`，通常和read命令相配合。

2. 6个替代性文件描述符

这些描述符序号从3到8。可以用exec命令分配用于输出的文件描述符，一旦将其指向文件就会一直有效，直到重新分配。如`exec 3>test`。或追加性的`exec 3>>test`。在后续语句中，应当有语句包含>&3运算符，以将该语句输出到指定文件。

恢复已重定向的文件描述符的方法是，现将另一个文件描述符分配给标准文件描述符，反之亦可，如`exec 3>&1`。然后再利用该文件描述符恢复，如`exec 1>&3`。

还可以使用同一个文件描述符兼做输入和输出，从而对文件进行读和写，如`exec 3<> test`。

使用&-可以关闭文件描述符，如`exec 3>&-`。如果在关闭文件描述符后重新打开该输出文件，则shell会用一个新文件来替换已有文件。这意味着如果输出数据，它就会覆盖已有文件。

3. lsof命令

lsof命令会列出整个Linux系统打开的所有文件描述符，包括所有后台进程和登录用户打开的文件。可选项-p允许指定进程ID（PID），可选项-d允许指定要显示的文件描述符编号，多个编号之间以逗号分隔。特殊变量$$可以获取进程的当前PID。可选项-a可用于另外两个选项的结果执行AND运算。

4. null文件

将STDERR重定向到一个名为null文件（格式>/dev/null）的特殊文件可以将脚本作为后台进程运行，并且不显示脚本输出。重定向到null文件的任何数据都会被丢弃。将null文件作为输入重定向，即cat /dev/null > file可以用来快速清除现有文件中的数据，例如日志文件。

5. /tmp目录

/tmp是存放临时文件的目录，任何用户都有权读写其中的文件。mktemp命令可以直接在/tmp目录中创建唯一的临时文件，该文件不使用umask值。

命令mktemp filenameXXX在本地目录创建一个文件，其中末尾的X数量至少要3个以上且前面没有空格，用于随机替换为同等数量的字符以保证文件名唯一性。filename可以包含任意文本字符。该命令只返回文件名

可选项-t强制在系统临时目录中创建文件。该命令返回完整路径。

可选项-d创建一个临时目录，格式mktemp -dt filenameXXX在/tmp文件中创建一个文件夹。

6. tee命令

格式tee filename。该命令将来自STDIN的数据同时送往STDOUT和所指定的文件名。可以配合管道符号来重定向命令输出。tee命令会在每次使用时覆盖指定文件的原先内容。可选项-a将数据追加到指定文件。

第16章 脚本控制

1. 常用信号

Linux常用信号如下：

信号	值	描述
1	SIGHUP	挂起（hang up）进程
2	SIGINT	中断（interrupt）进程（在shell中按下Ctrl+C组合键可生成）
3	SIGQUIT	停止（stop）进程
9	SIGKILL	无条件终止（terminate）进程（使用kill -9命令发送）
15	SIGTERM	尽可能终止进程
18	SIGCONT	继续运行停止的进程

信号	值	描述
19	SIGSTOP	无条件停止，但不终止进程
20	SIGTSTP	停止或暂停（pause），但不终止进程（在shell中按下Ctrl+Z组合键可生成，随后可用ps查看）

bash shell默认忽略任何收到的SIGQUIT（3）信号和SIGTERM（15）信号，但会处理收到的所有SIGHUP（1）和SIGINT（2）信号。shell会将这些信号传给shell脚本处理，而shell脚本的默认行为是忽略这些信号，因为可能不利于脚本运行。在有已停止脚本的情况下退出shell需要输入两次exit命令。第一次提示还有已停止的脚本，第二次直接退出。

2. trap命令

trap命令指定shell脚本需要侦测并拦截的Linux信号。如果脚本收到了其中列出的信号，则该信号不再由shell处理，而是由本地处理。格式trap CMDs signals。如果脚本中的命令被信号中断，则使用带指定命令的trap未必能使其继续执行。因此通常使用带空操作的命令trap "" signals来使脚本完全忽略信号。捕获脚本退出的命令为trap CMDs EXIT，提前退出也可捕获。

在脚本不同位置使用选项不同的-trap命令可以进行不同的信号捕获处理。例如，循环前的trap和循环后的trap根据脚本运行阶段的不同输出结果也不同。在交互式shell中使用可选项-p可以查看被捕获的信号，如果值为空，则表示按照默认方式处理信号。

使用trap -- signals或trap - signal可以移除设置的信号捕获并恢复默认行为。

3. 后台运行脚本

在脚本名之后加上&会将脚本作为一个独立的后台进程运行，这种方法可以运行多个后台作业。在终端会话中启动的后台进程与终端相关联，终端会话退出则后台进程也会退出。

使用nohup命令能阻断发给特定进程的SIGHUP信号，解除终端与进程之间的关联。这样即使退出终端会话，进程也不会退出。格式nohup CMD。使用该命令后，进程不再同STDOUT和STDERR相绑定，这些消息将重定向到一个名为nohup.out的文件中。这个文件一般在当前工作目录或$HOME中创建。如果nohup运行了多个命令，这些命令的输出都会被追加到同一个nohup.out文件中。使用jobs命令可以处理这些作业。使用kill -9 PID可以删除已停止的作业。

使用bg命令以后台模式重启已停止的作业。存在多个作业时，需要在bg命令后加上作业号，如bg 2。

使用fg命令以前台模式重启已停止的作业，需要加上作业号。

4. 调整优先级

优先级的取值范围是-20到+19，其中-20优先级最高。默认情况下shell以优先级0启动所有进程。

使用nice命令可以在启动命令时设置其调度优先级。可选项-n让命令以更低的优先级运行。ps中的ni一列显示优先级。但只有root用户才能提高作业的优先级。

使用renice命令可以修改已运行命令的优先级。格式renice -n NI -p PID。

5. at命令

命令at指定Linux系统何时运行脚本。其守护进程atd在后台运行，默认每60秒检查一次一个特殊目录（通常是/var/spool/at或/var/spool/cron/atjobs）以获取at命令提交的作业。at命令格式为at [-f filename] time。如果指定的时间已经过去，那么会在第二天的同一时刻运行作业。

at命令可用的时间和日期格式参考/usr/share/doc/at/timespec文件。该命令会将作业提交到作业队列，作业队列通常用小写字母a~z和大写字母A~Z指代，一共有52种。batch命令（/usr/bin/batch）会调用at命令将作业提交到b队列中。at命令默认将作业放入a队列。可选项-q指定其他的队列，z队列相对于其他队列少占CPU。

Linux默认会将at命令的STDOUT和STDERR发送到用户email地址，可以在脚本中对其重定向。可选项-M禁止作业发送邮件。

atq命令可以查看系统中有哪些作业在等待。atrm命令+作业号可以指定删除等待中的作业。但只能删除自己提交的作业。

6. cron程序

cron程序可以调度需要定期执行的作业。该程序从cron时间表中获知已安排执行的作业。cron时间表的格式为：minutepasthour hourofday dayofmonth month dayofweek CMD，允许使用特定值、取值范围或通配符（*）来指定各个字段。例如，每天10:15执行该程序则设置为15 10 * * * CMD（24小时制）。可以使用三字符的文本值（mon, tue, wed, thu, fri, sat, sun）或数值（0或7代表周日，6代表周六）指定dayofweek字段。

dayofmonth指月份中的日期值（1~31）。通过设置if-then语句，使用date命令检查明天的日期是不是某个月份的第一天（01），可以将命令设在每个月的最后一天执行。例如：00 12 28-31 * * if ["$(date +%d -d tomorrow)" = 01]; then CMD; fi。该脚本在每天中午12点检查当天是不是当月的最后一天。另一种方法是将CMD替换成一个包含if-then语句的控制脚本，在可能是每月最后一天的时候运行。命令列表必须指定完整路径。

crontab程序可以处理cron时间表。可选项-l列出已有的cron时间表。默认情况下，用户的cron时间表文件并不存在，可以用可选项-e向内添加字段。

还可使用4个预配置的cron脚本目录：/etc/cron.hourly、/etc/cron.daily、/etc/cron.monthly、/etc/cron.weekly和/etc/cron.yearly。

7. anacron程序

anacron程序会自动判断作业是否由于系统关闭等原因错过了设置的运行时间，并尽快运行该作业。也就是说，原计划在Linux系统关闭期间运行的作业在开机后自动运行。该程序只处理位于cron目录的程序（不会运行/etc/cron.hourly目录的脚本，因为时间需求少于一天），它通过时间戳来判断作业是否在正确的计划间隔内运行。每个cron目录的时间戳文件存放于/var/spool/anacron。

anacron程序使用自己的时间表（/etc/anacrontab）来检查作业目录。anacron时间表的格式为：period delay identifier CMD。period字段定义作业运行频率（以天为单位），用于检查作业的时间戳文件。delay字段指定在系统启动后到开始运行的等待分钟数。identifier字段是非空字段，值包括cron.daily等。CMD字段包含一个run-parts --report程和一个cron脚本目录名。

8. 使用新shell启动脚本

当用户登录bash shell时会运行的启动文件通常包括$HOME/.bash_profile，$HOME/.bash_login和$HOME/.profile。而$HOME/.bashrc是由非登录shell运行的文件。

source命令也可以运行shell脚本，但不会创建子shell。

第17章 创建函数

bash shell中创建函数的语法有两种。第一种是使用关键字function，格式：

```
function name {
    CMDs
}
```

第二种的格式：

```
name() {
    CMDs
}
```

只需要写出函数名即可调用函数。

1. 退出状态码

shell中的函数被视为一个小型脚本，运行结束时会返回一个退出状态。有3种方法能生成退出状态码，使用$?命令可以查看最后执行的命令的退出状态码。默认情况下，函数的退出状态码是函数中最后一个命令返回的退出状态码。但由于函数中命令执行失败也会继续向下执行，所以这种方法无法获知之前的命令是否成功执行。

return命令能以特定的整数值退出状态码退出函数。这个整数必须介于0~255，否则会出错。

可以将函数的输出赋给变量，例如result=$(dbl)，其中dbl是个函数。这样就可以突破return命令整数范围的限制，还可以返回浮点值和字符串。

2. 参数与变量

向函数传递参数的格式为func $var value，必须在shell脚本内预设，函数只通过位置变量访问参数，无法直接获取脚本的命令行参数。要在函数中使用脚本命令行参数，必须在调用函数时手动传入，例如func $1 $2。

默认情况下，在函数外定义的任何变量都是全局变量。在函数内部使用local关键字可将变量声明为局部变量。

3. 数组

向函数传入数组时，必须先将数组变量拆解成多个数组元素，然后将这些元素作为函数参数传递。最后在函数内部将所有参数重新组合成一个新的数组变量。样例如下：

```
function test {
    local newarray
    newarray=('echo "$@"')
    # 处理该数组
    ...
}

myarray=(1 2 3 4 5)
test ${myarray[*]}
```

函数局部变量newarray接收myarray中的所有数组元素并进一步处理。

函数返回数组时方式类似。先用echo语句按正确顺序输出数组的各个元素，然后脚本再将数组元素重组成一个新的数组变量。

4. 函数递归

函数可以递归调用，典型例子为阶乘。

```
function factorial {
    if [ $1 -eq 1 ]
    then
        echo 1
    else
        local temp=$[ $1 - 1 ]
        local result=`factorial $temp`
        echo $[ $result * $1 ]
    fi
}
```

5. 库

库可以使得同一段代码可以在多个脚本中使用。可以使用source命令的别名**点号操作符**.使脚本可以调用库中的函数。例如，库文件名为myfuncs，使用命令. ./myfuncs即可调用该库。

6. 命令行函数

在shell中定义函数就可以在整个系统的任意目录中使用它。

有两种方法在命令行中直接定义函数。第一种是采用单行方式定义，如$ function name{ CMDs; }，必须在每条命令后加分号。这种单行定义可以在函数体中跨行。例如：

```
$ funtion name{ CMD1;
CMD2; }
```

另一种是采用多行方式定义函数，无需在命令末尾放置分号，格式如下：

```
$ funtion name {
> CMD
> }
```

在命令行创建函数时如果函数名与另一个命令（如内建命令）相同，则函数会覆盖原来的命令。且退出shell后，函数也会消失。

可以在.bashrc文件中定义函数，使得每次启动shell都能调用。在shell脚本中可使用source命令或其别名.将库文件中的函数添加到.bashrc脚本中。例如：

```
if [ -r /etc/bashrc ] then
    . /etc/bashrc
fi

. /home/rich/libraries/myfuncs
```

这些函数还会传给子进程。

7. GNU shtool shell脚本函数库

这个库提供了一下简单现成的shell脚本函数。格式shtool [options] [function [options] [args]]。可以在命令行和脚本中直接使用shtool函数。

第18章 图形化桌面环境中的脚本编程

1. 文本菜单

case命令用来创建shell文本菜单，它根据用户的选项来执行相应的命令。其默认字符*可以用来处理所有不正确的菜单项。在创建菜单之前通常使用clear命令清除屏幕上的文本，然后使用echo命令显示菜单。可选项-e允许显示如制表符和换行符之类的非可打印字符。组合可选项-en可去掉结尾的换行符。最后使用read命令获取用户输入，可选项-n限制只读取一个字符。

通常为还没有实现的函数创建一个桩函数，它允许在实现某个函数的同时，菜单仍能正常运作。

select命令只需要一个命令就可以创建菜单，然后获取输入并自动处理。格式如下：

```
select variable in list
do
    CMDs
done
```

list参数是由空格分隔的菜单项列表，该列表构成了整个菜单。select命令将每个列表项显示成一个带编号的菜单项，然后显示一个由PS3环境变量定义的特殊提示符（PS3=xxx），指示用户做出选择。示例代码：

```
#!/bin/bash

function diskspace {
        clear
        df -k
}

function whoseon {
        clear
        who
}

function memusage {
        clear
        cat /proc/meminfo
}

PS3="Enter option:"
select option in "Display disk space" "Display logged on users"
"Display memory usage" "Exit program"
do
        case $option in
        "Exit program")
                break ;;
```

```
        "Display disk space")
                diskspace ;;
        "Display logged on users")
                whoseon ;;
        "Display memory usage")
                memusage ;;
        *)
                clear
                echo "Sorry, wrong selection" ;;
        esac
done
clear
```

2. 文本窗口部件

dialog软件包能够用ANSI转义控制字符，在文本环境中创建标准的窗口对话框，格式 `dialog --widget paras`。其中paras定义部件窗口的大小以及部件需要的文本，其部件表（--widget）可选项如下：

部件名	描述
calendar	提供可选择日期的日历
checklist	显示多个条目，其中每个条目都可以打开或关闭
form	构建一个带有标签以及文本字段（可以填写内容）的表单
fselect	提供一个文件选择窗口来浏览选择文件
gauge	显示一个进度条，指明已完成的百分比
infobox	显示一条消息，但不用等待回应
inputbox	显示一个文本框，以输入文本
inputmenu	提供一个可编辑的菜单
menu	显示一系列可供选择的菜单项
msgbox	显示一条消息，要求用户点选OK按钮
pause	显示一个进度条，指明暂停期间的状态
passwordbox	显示一个文本框，但会隐藏输入的文本
passwordform	显示一个带标签和隐藏文本字段的表单

部件名	描述
radiolist	提供一组菜单项，但只能选择其中一个
tailbox	用tail命令在滚动窗口中显示文件的内容
tailboxbg	跟tailbox一样，但运行在后台
textbox	在滚动窗口中显示文件的内容
timebox	提供一个选择小时、分钟和秒数的窗口
yesno	提供一条带有Yes按钮和No按钮的简单消息

每个dialog部件都提供了两种输出形式：STDERR和退出状态码（使用$?变量）。如果用户选择Yes或OK按钮则返回退出状态码0，如果选择Cancel或No按钮则返回退出状态码1。如果部件返回数据，则dialog命令将数据发送给STDERR，可以进行重定向。dialog有大量定制命令选项，例如--title，--backtitle等。可选项--form允许将多个文本框组合到单个窗口中，以此输入多个数据项。格式dialog --form text height width formheight [label y x item y x flen ilen]。

(1) msgbox部件

格式dialog [--title Title] --msgbox _text height width_。

(2) yesno部件

格式dialog [--title Title] --yesno _text height width_。

(3) inputbox部件

格式dialog --inputbox _text height width_ 2>file。

(4) textbox部件

格式dialog --textbox _file height width_。

(5) menu部件

格式dialog --menu _text height width total menus 2> file，其中total定义了一次在窗口中显示的菜单项总数。在菜单参数中需要提供标号和文本，如1 text 2 text。

(6) fselect部件

格式dialog --title Title --fselect path height width 2> file。使用空格键选定文件，将其加入文本框中。

在脚本中使用dialog命令，需要注意两条规则。一是如果有Cancel或No按钮，要检查dialog命令的退出状态码。二是重定向STDERR来获取输出值。

3. 图形化窗口部件

kdialog和zenity软件包分别为KDE桌面和GNOME桌面提供了图形化窗口部件。

(1) KDE环境

kdialog命令使用命令行选项指定具体使用哪种类型的窗口部件。格式：kdialog display-options window-options args。其中，kdialog窗口选项如下：

选项	描述
--checklist title [tag item status]	带有状态的多选列表菜单，可以表明选项是否被选定
--error text	错误消息框
--inputbox text [init]	输入文本框，init为默认值
--menu title [tag item]	带有标题的菜单选择框以及用标号标识的选项列表
--msgbox text	显示指定文本的简单消息框
--password text	隐藏用户输入的密码文本框
--radiolist title [tag item status]	带有状态的单选列表菜单，可以表明选项是否被选定
--separate-output	为多选列表和单选列表菜单返回按行分开的列表项
--sorry text	“对不起”消息框
--textbox file [width][height]	显示文件内容的文本框，可指定宽高
--title title	指定标题
--warningyesno text	含有Yes和No按钮的警告消息框

选项	描述
--warningcontinuecancel text	含有Continue和Cancel按钮的警告消息框
--warningyesnocancel text	含有Yes、No和Cancel按钮的警告消息框
--yesno text	含有Yes和No按钮的提问框
--yesnocancel text	含有Yes、No和Cancel按钮的提问框

checklist和radiolist部件允许在列表中定义单独的选项以及它们是否默认选定，如kdialog --checklist title 1 text [on off] 2 text [on off]。选择OK按钮后，kdialog会将所选列表项的标号发送到STDOUT。要注意，kdialog命令的菜单重定向不使用2>而使用>。

(2) GNOME环境

GNOME环境支持gdialog和zenity软件包。zenity更加常见，其窗口部件如下：

选项	描述
--calendar	显示整月日历
--entry	显示文本输入对话窗口
--error	显示错误消息对话窗口
--file-selection	显示完整的路径和文件名对话窗口
--info	显示信息对话窗口
--list	显示多选列表或单选列表对话窗口
--notification	显示通知图标
--progress	显示进度条对话窗口
--question	显示yes/no对话窗口
--scale	显示一个带有滑动条的比例尺对话窗口，可以选择一定范围内的数值
--text-info	显示含有文本的文本框

选项	描述
--warning	显示警告对话窗口

zenity不支持菜单对话窗口。

第19章 初识sed和gawk

sed编辑器和gawk编辑器用来在shell脚本中处理文本数据。

1. sed编辑器

sed编辑器是流编辑器，它根据事先设计好的一组规则编辑数据流。格式为sed options script file。

(1) 可选项

可选项-e CMDs允许处理输入时加入额外的sed命令，除最后一条命令外的所有命令末尾以分号结尾，或是使用sed -e '命令格式输入多行命令，此时不用分号结尾，但最后要以'结尾。

可选项-f允许处理输入时将file中指定的命令添加到已有的命令中，不用在每条命令后加分号，文件扩展名可以设为.sed以免与shell脚本混淆。

可选项-F打印出当前正在处理的文件名，且不受-n的影响。在F前加上1，即-1F表示只显示一次。

可选项-n表示不产生命令输出，使用p(print)命令完成输出。script指定应用于流数据中的单个命令，配合可选项命令的数量和形式可不同。默认情况下，sed编辑器会将指定的命令应用于STDIN输入流中，因此可配合管道使用。例如，使用echo text | sed s/old_STR/new_STR使用替换（s）命令替换字符串。sed编辑器默认不会修改文本文件的数据，只是将修改后的数据发送到STDOUT。

可选项-s告知sed将目录内的各个文件作为单独的流。

(2) s命令：替换

s命令默认只替换每行中出现的第一处匹配文本，要替换每行所有的匹配文本，必须使用替换标志。格式s/pattern/replacement/flags。

flags有4种选项：数字n指明替换第n处匹配；g指明替换所有匹配；p指明打印出替换后的行；w file将替换的结果写入文件，正常输出会被保存在STDOUT中。

sed编辑器允许使用自定义字符作为替代命令的替代分隔符，例如使用感叹号为sed s!/bin/bash!/bin/csh /etc/passwd。

(3) 行寻址

只想将sed编辑器中的命令应用于特定的某一行或某些行必须使用**行寻址**，有两种形式：以数字形式表示的**行区间**和匹配**行内文本**的模式。格式为[address]CMD，也可以将针对特定地址的多个命令分组：

```
address {
    CMD1
    CMD2
    CMD3
}
```

在使用数字形式的行寻址时可以用行号来引用文本流中的特定行，这个行地址既可以是单个行号，也可以是用起始行号、逗号以及结尾行号指定的行区间。如sed '2,3s/dog/cat/' file。结尾行号使用$符号将命令应用于从某行开始到结尾的所有行。

sed编辑器还允许指定文本模式来过滤出命令所应用的行，格式/pattern/CMD，如sed '/rich/s/bash/csh/' /etc/passwd表示只修改用户rich的默认shell。此外，sed编辑器还引入了正则表达式来创建匹配效果更好的模式。

在单行中执行多条命令，可以用花括号将其组合在一起。

(4) d命令：删除

d命令可以删除文本流中的特定行，如果没有加寻址模式则会删除所有文本行。如sed '3d' file表示删除file中的第3行，sed '/number 1/d' file表示删除file中包含number 1字段的文本行。

还可以使用两个文本模式，例如sed '/1/,/3/d' file来删除某个区间内的行。但这种方法风险很高。第一个模式会“启用”行删除功能，第二个模式会“关闭”行删除功能。在启用第一个模式后，如果后续没有匹配到第二个模式，则会删除后续所有行。

(5) i命令和a命令：插入和附加

i命令（插入命令）在指定行**前**增加一行，a命令（附加命令）会在指定行**后**增加一行。这两条命令不能在单个命令行中使用，必须指定是将行插入还是附加到另一行。格式如下：

```
sed '[address]CMD\
new line'
```

new line的文本是指定的插入或附加文本。也可使用管道符号，如：echo text2 | sed '[i a]\text2'。

要向数据流内部插入或附加数据，必须用地址告诉sed编辑器希望数据出现在什么位置。用这些命令时只能指定一个行地址，不能用行区间。要插入或附加多行文本，必须在要插入或附加的每行新文本末尾使用反斜线\。例如：

```
sed '3i\
This is an inserted line.\
This is another inserted line.
' data.txt
```

(6) c命令：修改

c命令（修改命令）允许修改数据流中整行文本的内容，可用数字和文本模式寻址。该命令同样必须单独指定一行，如：

```
sed '2c\
This is a changed line of text.
' data.txt
```

使用地址区间时，c命令会用指定的一行文本替换数据流中这些区间内的文本，即该区间内只剩下替换后的一行。

(7) y命令：转换

y命令（转换命令）是唯一可以处理单个字符的sed编辑器命令，格式为[address]y/inchars/outchars。该命令对inchars和outchars中的字符进行一对一映射，如果长度不同则报错。

y命令是全局命令，对文本行中匹配到的所有指定字符进行转换，不考虑字符出现的位置。

(8) 三个打印命令：p、=、l

p命令打印输出的一行，常用于打印包含匹配文本模式的行或部分行（使用区间）。使用sed -n可以抑制其他行的输出。格式如sed -n '2,3p' file。

=命令打印文本行在数据流中的行号。行号由数据流中的换行符决定。格式sed '=' file。

l命令（列出命令）可以打印数据流中的文本和不可打印字符。格式如sed -n 'l' file。

(9) w命令：写入

w命令（写入命令）用来向文件写入行。格式：[address]w filename，地址可以是sed支持的任意类型的寻址方式。

(10) r命令：读取

r命令（读取命令）允许将一条独立文件中的数据插入数据流。格式[address]r filename。该命令只能指定单个行号或文本模式地址，不能指定地址区间。sed编辑器会将文件内容插入指定地址之后。在末尾添加文本使用$符号。可以与d命令配合使用替换占位文本。

2. gawk编辑器

gawk较sed更为高级，它提供了一种编程语言。格式gawk options program file。options选项如下：

选项	描述
-F fs	指定行中划分数据字段的字段分隔符
-f file	从指定文件中读取gawk脚本代码
-v var=value	定义gawk脚本中的变量及其默认值
-L [keyword]	指定gawk的兼容模式或警告级别

gawk脚本用花括号定义，还要放在单引号中，如gawk '{program}'。在脚本运行时，它会一直等待来自STDIN的文本。使用Ctrl+D组合键来生成EOF字符，以此终止gawk程序。

在处理文本文件的数据时，gawk会自动为每一行的各个数据元素分配一个**数据字段变量**。默认情况下，$0代表整个文本行，$n代表第n个数据字段（例如第n个单词）。文本行中的数据字段通过字段分隔符来划分，默认是任意的空白字符。可选项-F指定其他的分隔符，例如，-F:指定冒号为分隔符。

gawk编程语言允许将多条命令组合成一个常规的脚本，只需在命令之间加入分号。也可以用次提示符一次一行输入脚本。按下Ctrl+D组合键可退出。gawk脚本文件扩展名可设置为.gawk，在文件中可一行一条命令，不用在末尾添加分号。并且在gawk脚本中，引用变量值时无须像shell脚本那样使用美元符号。

在命令花括号之前添加关键字BEGIN可以强制gawk在**读取数据前**执行BEGIN关键字后指定的脚本，但要在之前使用另一个花括号来定义实际要执行的脚本。如gawk 'BEGIN {program} {2nd program}' file。在2nd program后添加END关键字使gawk在处理完数据后执行后续脚本。

第20章 正则表达式

Linux中最常用的是POSIX基础正则表达式（BRE）引擎和POSIX扩展正则表达式（ERE）引擎。sed编辑器使用BRE引擎，gawk使用ERE引擎。

1. BRE模式

脱字符^指定位于数据流中文本行行首的模式，美元符号$定义行尾锚点。将这两个锚点组合在一起成为^$过滤出数据留中的空行，可以使用sed的d命令删除这些空行。例如，sed '/^$/d' file。

点号字符.匹配除换行符之外的任意单个字符。方括号[]用于定义字符组，可匹配其中任一字符。在字符组开头添加脱字符即可匹配字符组中没有的字符，称为排除型字符组，如[^ch]。在字符组中使用连字符可以指定匹配字符区间，如[0-9]或[a-z]。还可指定多个不连续的区间，如[a-ch-m]匹配a~c和h~m的字符。

BRE特殊字符组如下：

选项	描述
[[:alpha:]]	匹配任意字母字符
[[:alnum:]]	匹配任意字母数字字符
[[:blank:]]	匹配空格或制表符
[[:digit:]]	匹配0~9中的数字
[[:lower:]]	匹配小写字母字符a~z
[[:print:]]	匹配任意可打印字符
[[:punct:]]	匹配标点符号
[[:space:]]	匹配任意空白字符
[[:upper:]]	匹配任意大写字母字符A~Z

在某一字符后放置星号*表明该字符必须在匹配模式的文本中出现0次或多次。将点号和星号相结合组成.*，能够匹配任意数量的任意字符。也能用于字符组，如b[ae]*t，表示字符组元素出现0次或多次时可以匹配。

2. ERE模式

问号?表明前面的字符可以出现0次或1次，但不会匹配出现2次及以上的字符，可以与字符组相结合。

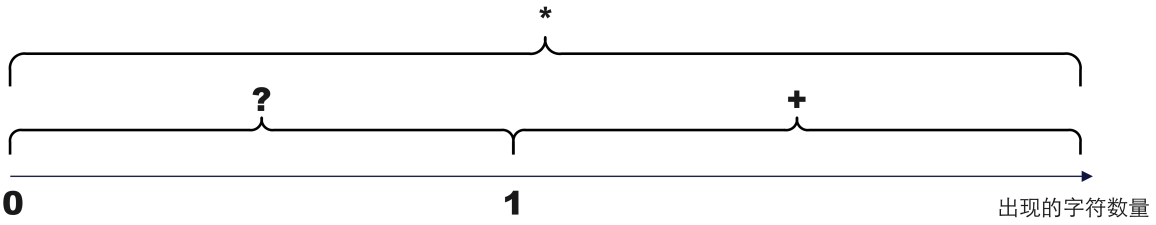
加号+表明前面的字符可以出现1次或多次，但必须至少出现1次，可以与字符组相结合。

星号、问号、加号的区别如下：

符号	描述
*	文本在匹配模式中出现0次或多次（>=0）
?	文本在匹配模式中出现0次或1次（0 or 1）

符号	描述
+	文本在匹配模式中出现1次或多次（>=1）

花括号{}允许为正则表达式指定具体的可重复次数，称为区间。有两种格式可以指定区间。①{m}表示文本恰好出现m次；②{m,n}表示文本至少出现m次，至多出现n次。如下图：



默认情况下，gawk不识别正则表达式区间，必须指定可选项--re-interval。如gawk --re-interval '/be{1,2}t/ CMD'。花括号同样适合字符组。

竖线符号|允许在检查数据流时，以逻辑OR方式指定正则表达式引擎要使用的两个或多个模式。格式expr1|expr2|...，正则表达式和竖线符号之间不能有空格。

圆括号()会对正则表达式进行分组，使得每一组被视为一个整体，可以像对普通字符一样对该组应用特殊字符。

第21章 sed进阶

1. 多行文本

sed编辑器提供了3个可用于处理多行文本的特殊命令：N命令、D命令和p和P命令。

(1) next命令

next命令包括单行n和多行N两种形式。

单行n命令让sed编辑器移动到文本的下一行，也叫将数据流中的下一行移入sed编辑器的工作空间（称为**模式空间**）。如命令sed '/text/{n ; d}' file表示在匹配到text后移动到其下一行，并删除该行。这种匹配是全局范围匹配，即如果有多行文本中包括text，则这些文本的下一行都会被删除。

多行N命令将下一行添加到模式空间中已有文本之后。这样的结果是数据流中的两行文本合并到同一个模式空间中。文本行之间仍用换行符分隔，但sed编辑器现在会将两行文本当成一行来处理。如命令/First/{ N ; s/\n/ / } file表示将包含First文本的一行与下一行合并，然后用s替换命令将换行符替换成空格。

如果N命令要匹配的文本在最后一行，则不会被成功匹配。解决方法是将单行编辑命令放到N命令前，将多行编辑命令放到N命令后，如：

```
sed '
s/text1 text2/newtext1 newtext2/
N
s/text1\ntext2/newtext1\nnewtext2/
' file
```

(2) D命令

单行删除d命令如果和N命令一起使用，则会将匹配到的两行都删掉。而D命令只会删除模式空间中的第一行。如命令`sed '/^$/{ N ; /Header/D} file'`表示只删除包含Header文本的行前的第一个空行。D命令在删除模式空间中的第一行后，会强制sed编辑器返回到脚本的起始处，对当前模式空间中的内容重新执行此命令。配合N命令可以单步扫过整个模式空间，对多行进行匹配。

(3) p和P命令

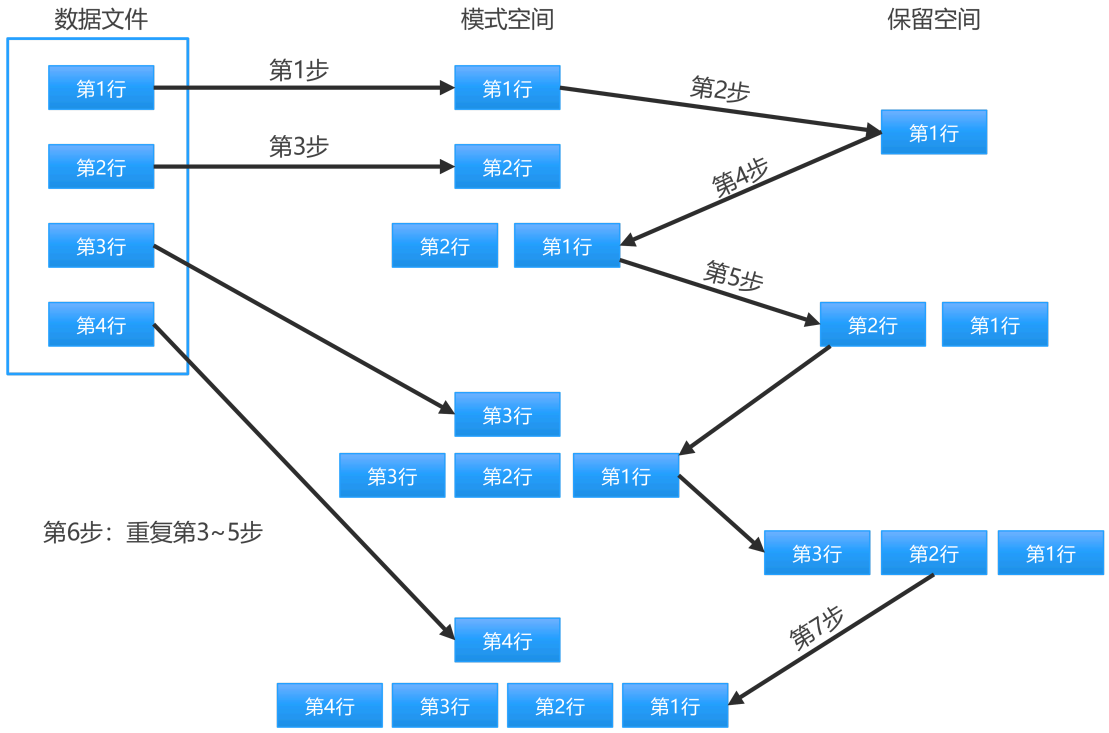
多行打印p或P命令只打印模式空间中的第一行。

2. 保留空间

模式空间在sed编辑器执行命令时保存待检查的文本，sed编辑器还有保留空间缓冲区。处理模式空间中的某些行时，可以用保留临时保存部分行。相关命令有5个：

命令	描述
h	将模式空间复制到保留空间
H	将模式空间附加到保留空间
g	将保留空间复制到模式空间
G	将保留空间附加到模式空间
x	交换模式空间的和保留空间的内容

这些命令可以将文本从模式空间复制到保留空间，以便清空模式空间，载入其他要处理的字符串。可用于反转各行文本，图示如下：



也可以直接用tac命令倒序显示文本文件。

3. 排除命令!

感叹号!命令让原本会起作用的命令失效，如sed -n '/Header/!p' file会打印出除Header字段所在行的所有行。

4. 分支命令b

分支b命令允许基于地址、地址模式或地址区间排除一整段命令，格式[address]b [label]。address决定哪些行会触发分支命令，label定义要跳转到的位置，并执行相应脚本行的脚本命令。如果缺失则跳过触发分支命令的行，继续处理余下的文本行。例如sed '{2,3b ; s/old_text/new_text/}'。label的格式是:labelNUM，最多可以有7个字符。样例如下：

```
sed '{/First/b jump1 ;
s/Line/Replacement
:jump1
s/Line/Jump Replacement/}
' data.txt
```

可以为分支命令指定一个地址模式，如果模式不匹配就不会再跳转。如/,/b label表示只会在行中有逗号的情况下跳转。

5. 测试命令t

t命令根据先前替换s命令的结果跳转到某个label处，而不是根据address进行跳转。如果s命令成功匹配并完成了替换，t命令就会跳转到指定标签，否则不会跳转。格式[address]t [label]。在没有指定label的情况下，如果测试成功，sed就会跳转到脚本结尾。

t命令还能够起到if-then语句的作用，即在两个替换选项中选择一个执行，且无须指定label。例如sed '{s/First/Matched/ ; t s/Line/Replacement}'。

6. &符号

&符号可以代表s命令中的匹配模式，可以代表任何匹配到的文本。例如以下代码：

```
echo "The cat sleeps in his hat." |
sed 's/.at/"&"/g'
```

输出为：The "cat" sleeps in his "hat".。

7. 反向引用

sed编辑器使用圆括号来定义替换模式中的子模式，随后使用特殊的字符组合来引用每个子模式匹配到的文本。这称作反向引用。反向引用由反斜线和数字组成，数字表明子模式的序号，如\1、\2。

在s命令使用圆括号时必须使用转义字符，如sed 's/\(text\) old_text/\1 new_text'。反向引用允许将某个子模式匹配到的文本作为替换内容，可以用于将一个单词替换一个短语，例如：echo "That furry hat is pretty." | sed 's/furry \(.at\)/\1/'使用匹配到的hat替换furry hat。也可以在两个或多个子模式间插入文本，如：

```
echo "1234567" | sed '{
:start
s/\(.*[0-9]\)\([0-9]\{3\}\)/\1,\2/
t start
}'
```

上述代码输出结果为1,234,567。

8. 在脚本中使用sed

在shell脚本中使用命令行参数，如$1，这个脚本就成为了一个**包装器**，可以运行其他脚本或参数。使用$()将sed编辑器命令的输出重定向到一个变量中。

9. 实用sed工具

如果文本文件内存在没有空行的多行文本，使用命令sed 'G' file可以加倍行间距。原因是启动sed编辑器时，保留空间中会有一个空行。使用G命令会将其附加到已有行之后。进一步使用sed '$!G' file能确保脚本不会将空行附加到数据流的最后一行之后。

如果文本文件内存在可能含有空行的多行文本，需要先使用/^$/d命令删除数据流中的所有空行，然后再用G命令在每行之后插入新的空行。

使用=命令可以显示数据流中行的行号，但会单独显示一排。解决方法是使用管道符号将该结果传给另一个sed编辑器脚本，由后者使用N命令合并这两行，并使用-s命令替换换行符，最终输出结果中没有包含间隔。例如sed '=' file | sed 'N; s/\n/ /'。

联合使用N、D命令和$符号可以用滚动窗口显示数据流末尾的若干行。N命令将下一行文本附加到模式空间中已有文本行之后，到达特定行数后使用$符号检查是否已经到达尾部，如果不是，就继续向模式空间增加行，同时使用D命令循环删除已有行。代码如下（设显示最后10行）：

```
sed '{
:start
$q ; N ; 11,$D
b start
}' data.txt
```

3个用来**删除数据中不需要的空行**的脚本如下：

删除连续的空行：方法是用地址区间来检查数据流。关键在于创建包含一个非空行和一个空行的地址区间，如果sed编辑器遇到了这个区间，它不会删除行。但对于不属于该区间的行（两个或更多的空行），则执行删除操作。命令为/./,/^$/!d。

删除开头的空行：命令为/./,$!d。该命令从含有字符的行开始，一直到数据结束，期间的所有行都不会被删除。

删除结尾的空行：命令如下：

```
sed '{
:start
/^\n*$/{$d; N; b start}
}'
```

该地址模式匹配只含一个换行符的行，如果这个行是最后一行，则删除命令将其删除。如果不是最后一行，则N命令将下一行附加到它后面，然后分支命令跳到循环起始位置重新开始。

删除HTML标签的命令为s/<[^>]*>//g。这个命令使sed编辑器忽略任何嵌入原始标签中的大于号。还可以使用D命令删除多余的空行。如：sed 's/<[^>]*>//g ; /^$/d' file。

第22章 gawk进阶

1. 使用变量

(1) 数据字段变量

gawk的数据字段变量（见19章）是一种内建变量。数据字段由字段分隔符划定。gawk会把文本文件中每一个空行都视为记录分隔符。可以控制gawk对输入数据和输出数据中字段和记录的处理方式的5个内建变量如下：

变量	描述
FIELDWIDTHS	由空格分隔的一列数字，定义了每个数据字段的确切宽度
FS	输入字段分隔符
RS	输入记录分隔符，默认为\n
OFS	输出字段分隔符，默认为1个空格
ORS	输出记录分隔符，默认为\n

FIELDWIDTHS用于没有使用字段分隔符，而是将数据放置在记录的特定列的情况。设置该变量后，gawk会忽略FS变量，并根据提供的字段宽度来计算字段，例如：`gawk 'BEGIN{FIELDWIDTHS="3 5 2 5"} {print $1,$2,$3,$4} file'`。这种方法并不适用于变长的数据字段。

gawk不会将程序脚本视为命令的一部分。例如：`gawk 'BEGIN{print AGRC,ARGV[1]}' data1`，其中gawk命令为ARGV[0]，data1为ARGV[1]。ARGC的值是命令行参数的数量，此处为2。

(2) 更多变量

ENVIRON变量使用关联数组来提取shell环境变量，关联数组用文本（而非数值作为数组索引）。例如：`gawk BEGIN{print ENVIRON["HOME"]}`，其中HOME为索引，提取HOME环境变量的值。

FNR、NF和NR变量可以用来在gawk脚本中跟踪数据字段和记录。NF用来引用记录中最后一个数据字段，如`{print $1, $NF}`。NF变量含有数据文件中最后一个字段的编号，可以在前面加上$，将其用作字段变量。

FNR变量包含**当前**数据文件中已处理过的记录数，NR变量包含**所有**数据文件中已处理过的记录总数。

(3) 自定义变量

在脚本中给变量赋值使用赋值语句，引用时不添加$符号。也可以通过gawk命令行为脚本中的变量赋值。可选项-v允许在BEGIN部分之前设定变量，必须放在脚本代码之前，如`gawk -v n=3 -f file data`。

2. 处理数组

gawk编程语言使用关联数组提供数组功能，允许用数字或字符串来引用数组元素，类似于Python的字典。赋值格式为：`var[index] = element`。遍历关联数组可以使用`for`语句的特殊形式：`for (var in array) { statements }`，该语句在每次循环时将关联数组`array`的下一个索引赋给变量`var`，然后执行一遍`statements`。

从关联数组中删除数组元素使用命令`delete array[index]`，此处的`index`为索引值。

3. 使用模式

(1) 正则表达式

gawk可以使用正则表达式（BRE或ERE）来筛选脚本要作用于数据流中的哪些行，它必须出现在与其对应脚本的左花括号前。如`gawk 'BEGIN{FS=","} /11/{print $1}' file`。正则表达式默认匹配包含字段分隔符。

匹配操作符~能将正则表达式限制在记录的特定数据字段。可以指定匹配操作符、数据字段变量以及要匹配的正则表达式，如`$1 ~ /^data/`。该表达式会过滤出第一个数据字段以文本`data`开头的所有记录。常用于在文件中搜索特定的数据元素。例如：`gawk -F: '$1 ~ /rich/{print $1,$NF}' /etc/passwd`。也可以结合`!`，构成`!~`来排除正则表达式的匹配。

(2) 数学表达式

还可以在匹配模式中使用数学表达式。如`gawk -F: '$4 == 0{print $1}' /etc/passwd`。也可以对文本数据使用表达式，但必须完全匹配。

(3) `if`语句

gawk编程语言内支持标准格式的`if-then-else`语句，但`if`语句的条件必须放在括号内。格式`if (condition) statement`。如果要执行多条语句，则放入花括号内。如：

```
gawk '{
if (condtion){
statements
} else {
statements
}
}' file
```

在单行中使用`else`字句格式：`if (condition) statement1; else statement2`。

(4) `while`语句和`do-while`语句

`while`语句格式：

```
while (condition){
statements
}
```

gawk同样支持在while中使用break语句和continue语句。

do-while语句格式：

```
do {
statements
} while (condition)
```

(5) for语句

gawk支持C风格的for循环：for (variable assignment; condition; iteration process)。

4. 格式化打印

格式化打印命令为printf，与C语言相同，允许指定具体如何显示数据的指令。格式printf "format STR", var1, var2。其中，format STR会用文本元素和格式说明符来具体指定如何呈现格式化输出。格式说明符的格式为%[modifier]control-letter。control-letter是一个单字符代码，用于指明显示什么类型的数据，modifier定义了可选的格式化特性。这些控制字母与C语言基本相同，例如，%d表示显示整数值。需要在printf命令末尾手动添加换行符，以便生成新行。代码示例：

```
gawk 'BEGIN{
x = 10 * 100
printf "The answer is: %e\n", x   # %e表示用科学计数法显示数字
}'
```

输出结果为The answer is: 1.000000e+03。

另外，还有3种修饰符可以进一步控制输出。width指定输出字段的最小宽度，prec指定浮点数中小数点右侧的位数或字符串中显示的最大字符数，-指明格式化空间中的数据采用左对齐而非右对齐（printf默认使用右对齐将数据放入格式化空间）。

5. 数学函数

gawk提供了部分用来执行常见数学运算的函数，这些函数能够处理的数值通常有一个区间，超出这个区间就会报错。表格如下：

函数	描述
atan2(x, y)	x/y的反正切，x和y以弧度为单位
cos(x)	x的余弦，x以弧度为单位

函数	描述
exp(x)	x的指数
int(x)	x的指数部分，取靠近0一侧的值
log(x)	x的自然对数
rand()	比0大且比1小的随机浮点值
sin(x)	x的正弦，x以弧度为单位
sqrt(x)	x的平方根
srand(x)	为计算随机数指定一个种子值

int()函数会生成一个值的整数部分，相当于其他编程语言的floor()函数。

rand()函数返回一个0到1之间（不包括0和1）的随机数，可以配合int()函数生成更大的随机数。如int(10 * rand())。

and(v1, v2)函数对v1和v2执行按位AND运算。

compl(val)函数对val执行补运算。

lshift(val, count)函数将val左移count位。

or(v1, v2)函数对v1和v2执行按位OR运算。

rshift(val, count)函数将val右移count位。

xor(v1, v2)函数对v1和v2执行按位XOR运算。

6. 字符串函数

函数	描述
asort(s, [d])	将数组s按照数组元素值排序。索引会被替换成表示新顺序的连续数字。如果指定d，则排序后的数组会被保存在数组d中
asorti(s, [d])	将数组s按索引排序。生成的数组会将索引作为数组元素值，用连续数字索引表明排序顺序。如果指定d，则排序后的数组会被保存在数组d中
gensub(r, s, h [, t])	针对变量$0或目标字符串t（如果指定）来匹配正则表达式r。如果h是一个以g或G开头的字符串，就用s替换匹配的文本。如果h是一个数字，则表示要替换r的第h处匹配

函数	描述
gsub(r, s [, t])	针对变量$0或目标字符串t（如果指定）来匹配正则表达式r。如果成功匹配，则将所有成功匹配的地方替换成字符串s
index(s, t)	返回字符串t在字符串s中的索引位置；如果没找到，则返回0
length([s])	返回字符串s的长度，如果没有指定则返回$0的长度
match(s, r [, a])	返回正则表达式r在字符串s中匹配位置的索引。如果指定了数组a，则将s的匹配部分保存在该数组中
split(s, a [, r])	将s以FS（字段分隔符）或正则表达式r（如果指定）分割并放入数组a中，返回分割后的字段总数
sprintf(format, variables)	用提供的format和variables返回一个类似于printf输出的字符串
sub(r, s [, t])	在变量$0或目标字符串t中查找匹配正则表达式r的部分。如果成功匹配，则用字符串s替换第一处匹配
substr(s, i [, n])	返回s中从索引r开始、长度为n的子串。如果未提供n，则返回s中剩下的部分
tolower(s)	将s中的所有字符都转换成小写
toupper(s)	将s中的所有字符都转换成大写

示例代码：

```
gawk 'BEGIN{FS=","}{
split($0, var)
print var[1], var[5]
}' data
```

7. 时间函数

函数	描述
mktime(datespec)	将一个按YYYY MM DD HH MM SS [DST]格式指定的日期转换成时间戳
strftime(format [, timestamp])	将当前时间的时间戳或timestamp（如果提供）转化为格式化日期（采用shell命令date的格式）

函数	描述
systime()	返回当前时间的时间戳

时间戳是自1970-01-01 00:00:00 UTC到现在，以秒为单位的计数，通常称为纪元时（epoch time）。

示例代码：

```
gawk 'BEGIN {
date = systime()
day = strftime("%A, %B %d, %Y", date)
print day
}'
```

8. 自定义函数

在gawk脚本中自定义函数必须使用function关键字：

```
function name([variables]){
statements
}
```

可在函数中使用return语句返回一个值，或是将函数的返回值赋给一个变量。

自定义函数必须出现在所有代码块之前，包括BEGIN代码块。例如：

```
gawk '
function name(){
statements
}
BEGIN{CMDs}{
statements
}
' file
```

可以将所有自定义的gawk函数放入一个文件，创建一个函数库。在gawk命令行中使用可选项-f就可以使用该文件。但-f不能和内联gawk脚本一起使用，因此需要使用多个-f。例如gawk -f funclib -f file，其中funclib为函数库文件。

第23章 使用其他shell

命令cat /etc/passwd | grep username可以查看账户使用的默认交互式shell。

命令ls -al /bin/sh可以查看默认的系统shell。

1. dash shell

dash shell命令行选项如下：

选项	描述
-a	导出分配给shell的所有变量
-c	从特定的命令字符串中读取命令
-e	如果是非交互式shell，就在未经测试的命令失败时立即退出
-f	显示路径通配符
-n	如果是非交互式shell，就读取命令但不执行
-u	在尝试扩展一个未设置过的变量时，将错误消息写入STDERR
-v	在读取输入时将输入写出到STDERR
-x	在执行命令时将每个命令写入STDERR
-I	在交互式模式下，忽略输入中的EOF字符
-i	强制shell运行在交互式模式下
-m	启用作业控制（在交互式模式下默认开启）
-s	从STDIN读取命令（在没有指定文件参数时的默认行为）
-E	启用Emacs命令行编辑器
-V	启用vi命令行编辑器

dash变量和bash变量的一个差异是，dash shell不支持数组。

在dash中执行数学运算需要使用双圆括号，不支持bash中的方括号。且dash只能识别=符号，不能识别==符号。

在dash中不能使用function关键字定义函数，只能使用函数名。

2. zsh shell

选项	描述
-c	只执行指定的命令，然后退出
-i	作为交互式shell启动，提供命令行提示符
-s	强制shell从STDIN读取命令
-o	指定命令行选项

zsh shell内建命令最重要的特性是模块。模块的格式是zsh/xxx。命令zmodload不添加参数会显示zsh shell中当前已安装的模块。接模块名可以添加新模块，如zmodload zsh/net/tcp。将zmodload命令放入$HOME/.zshrc文件中可以使zsh启动时自动加载常用的模块。

zsh shell数学运算支持双圆括号和let命令。使用let命令时应该在算式前后加上双引号以使用空格，如let value=" 4 * 5.1 / 3.2"。双圆括号可以放在算式两边或整个赋值表达式两边，如((value = 4 * 5.1))。要保证结果是浮点数，需要在指定数值时指定小数部分，如value=10.。

除bash shell结构化语法外，zsh还提供了repeat结构化命令。格式如下：

```
repeat param
do
    CMDs
done
```

param参数必须是一个数值，或是能计算出一个值的数学运算。这个值就是repeat命令要执行以下命令的次数。

zsh shell还支持使用function命令或函数名加圆括号的形式来创建自定义函数，如funcname(){CMDs}。

第24章 编写简单的脚本实用工具

1. 备份

Linux使用tar命令归档并备份数据。可选项-z会使用gzip将tar归档的文件压缩为tarball文件，其扩展名应该设置为.tar.gz或.tgz。可以在一个配置文件（实质是文本文件）中添加所有希望归档的目录的绝对路径进行批量归档。

为了让脚本读取配置文件，可以使用exec命令来重定向STDIN到该文件，然后使用while循环中的read命令读取各行目录。该while循环要将目录名加入归档列表，还要检查目录是否存在。

对于多个目录的备份，可以创建一个集中的归档仓库目录，并赋予某些用户访问权限。可以创建一个用户组，并将用户组的属组设置为该目录。为避免用户组的用户删除他人的归档文件，可以添加目录的粘滞位-t。

(1) 按日归档的脚本

创建一个按日归档的脚本示例如下，文件名为Daily_Archive.sh。

```
#!/bin/bash

today=$(date +%y%m%d)   # 获取今日日期
backupFile=archive$today.tar.gz # 设置归档文件名
config_file=/archive/Files_To_Backup.txt    # 指向含有待归档目录信息的归档配置文件
destination=/archive/$backupfile    # 追加归档文件的完整路径名

if [ -f $config_file ]  # 确定配置文件是否存在，如果不存在则报错
then
    echo
else
    echo "$config_file does not exist"
    exit
fi

file_no=1   # 从配置文件的第1行开始
exec 0< $config_file    # 重定向STDIN
read file_name  # 读取第一行记录

while [ $? -eq 0 ]  # 只要read命令仍然能够读取记录，则返回的退出状态码为0
do
    if [ -f $file_name -o -d $file_name ]   # 确定文件或目录是否存在
    then
        file_list="$file_list $file_name"   # 如果存在，则将名称添加到列表中
    else
        echo "$file_name does not exist."   # 不存在则报错
        echo "It is listed on line $file_no of the config file."
    fi

    file_no=$[$file_no + 1] # 跳到下一行
    read file_name  # 读取下一行
done

echo "Starting archive..."
echo

tar -czf $destination $file_list 2> /dev/null   # 归档文件到目标路径，并把警告信息丢弃

```

```
echo "Archive completed"
echo "Resulting archive file is: $destination"
echo

exit
```

在运行上述脚本时，首先需要更改脚本文件的权限。文件属主必须拥有可执行权限x才能运行脚本：chmod u+x Daily_Archive.sh。该脚本文件也可使用anacron命令定期运行。

(2) 按小时归档的脚本

创建归档目录层级结构能更直观地进行归档。可以在父目录下创建月份文件夹，然后在各个月份文件夹中再创建日文件夹。文件名为Hourly_Archive.sh，前半部分脚本如下：

```
#!/bin/bash

config_file=/archive/hourly/Files_To_Backup.txt

basedest=/archive/hourly    # 父目录

day=$(date +%d)
month=$(date +%m)
time=$(date +%k%M)

mkdir -p $basedest/$month/$day  # -p允许在单个命令中同时创建目录和子目录，且即使已经存在同名目录也不会报错

destination=$basedest/$month/$day/archive$time.tar.gz
```

后续主要代码与Daily_Archive.sh相同。

2. 删除账户

删除本地账户有4个步骤：获取正确的待删除用户账户名，“杀死”系统中正在运行的属于该账户的进程，确认系统中属于该账户的所有文件，删除该用户账户。

3. 系统监控

系统账户用于提供服务或执行特殊任务。一般这类账户需要再/etc/passwd文件中有对应记录，但禁止登录系统（root例外）。防止有人使用这些账户登录的方法是，将其默认shell设置为/bin/false、/usr/sbin/nologin或/sbin/nologin。可以使用cut命令获取/etc/passwd文件中所有账户的默认shell。

以超级用户权限使用chattr命令可以设置或取消不可变属性。使用lsattr命令可以查看属性是否设置成功。

第25章 井井有条

为新的脚本项目设置Git环境有几个步骤。首先在本地创建一个子目录，然后在目录中使用git init命令初始化.git/子目录。随后使用git config命令将个人信息添加到Git的全局仓库配置文件中，可以使用可选项--get查看信息。Git全局配置信息应用于系统中的所有Git项目，保存在主目录的gitconfig文件中，本地配置信息应用于工作目录中的特定Git项目，保存在working-directory/.git/config文件中。查看这些文件中的各种配置信息可使用git config --list命令。

之后要使用Git提交文件。创建好脚本之后，使用git add命令将其添加到暂存区（索引），随后使用git status命令查看脚本是否成功添加。使用git add.命令可以将当前工作目录的所有脚本同时添加到暂存区（索引）。这个命令会排除.gitignore文件中包含的文件或目录名。暂存区的索引文件是.git/index，对其使用file命令，则其类型显示为Git index。

然后使用git commit命令将项目提交至本地仓库。可以使用可选项-m添加注释，注释保存在COMMIT_EDITMSG文件中。再使用git status查看提交情况。如果这是一个新的脚本项目，则需要创建一个Markdown文件，命名为README.md，其内容描述该仓库的相关信息。

在向远程仓库推送项目之前，需要先使用git remote add origin _URL_命令来配置远程仓库地址，然后使用git remote -v命令检查远程仓库地址的状态。可使用git branch -m name重命名主分支，再用git branch --show-current命令查看分支的当前名称。用命令git push -u origin name将脚本复制到远程仓库。使用git clone命令将整个脚本项目从远程仓库复制到自己的本地系统。git log命令可以显示项目历史。