UNIX操作系统第7版用户手册

第二卷　　程序开发工具　　上集

中国科学院成都计算机应用研究所情报室译

UNIX操作系统第7版　用户手册

第二卷　程序开发工具

UNIX操作系统用户手册

第二卷　程序开发工具

目　录

第一章　系统的程序设计工具

第二章　程序维护

第三章　UNIX维护和修改

第一章　系统的程序设计工具

1．　UNIX外壳介绍

S·R·Bourne（贝尔实验室）

摘要

外壳（shell）是一种命令程序设计语言，它提供与UNIX操作系统的接口。它的特点包括控制流原语，参数传递，变量和字符串替换。可以利用的结构如像while，if then else，case和for之类。在外壳和命令之间可以双向通信。由命令设置的代码可以被用来确定控制流，从命令来的标准输出可以用作外壳的输入。

外壳可以修改命令运行的那个环境。输入和输出可以改变到文件去的方向，还可以调用通过管道通信的进程。可以按照用户所规定的顺序查找文件系统中的目录，从而找到命令。可以从终端上读入命令，也可以从文件中读取命令，这样就可允许把命令过程存贮起来以便以后使用。

1．0引言

外壳既是命令语言，也是程序设计语言。它可以提供与UNIX操作系统的接口。这篇备忘录用例子介绍UNIX操作系统。第一段包括终端用户的大部份日常要求。有些熟悉UNIX的用户在读这一段时更为方便；举例说，可参阅“UNIX初步入门”。第二

段介绍那些主要打算在外壳过程中使用的外壳特点。这些特点包括控制流原语和外壳提供的字符串计值的变量。在阅读这一段时，如果具有程序设计语言的知识是有帮助的。最后一段介绍外壳的更先进的特点。

1.1 简单命令

简单命令由一个或多个字构成，这些字彼此由空格隔开。第一个字是要执行的那条命令的名称，其余的那些字作为传送给命令的变元。举例说，

谁

是打印用户注册的名称的命令。命令

ls—l

打印当前目录中的文件表。变元—l通知ls打印每个文件的状态信息，长度和建立这个文件的日期。

1.2 后台命令

为了执行一条命令，在正常情况下外壳建立一个新的进程，并等待这个进程完成。不等待进程完成也可以运行命令。举例说

cc pgm.c &

调用C语言的编译程序来编译文件pgm.c。尾部&是一个算符，它通知外壳不要等待命令完成。为了有助于保留这个进程的踪迹，在这个进程建立之后，外壳就报告它的进程号。利用ps命令可以得到当前正在进行的那些进程的清单。

1．3输入输出改向

大多数命令都在那些原先与终端相连接的标准输出上产生输出。这样的输出可以通过写入被送给文件。举例说

ls—l　>文件

符号>文件由外壳解释，并且不作为变元送给ls。如果不存在文件，那么外壳就要建立起这个文件；否则原先的文件内容由从ls来的输出代换。利用下面的符号可以把输出加到文件中

ls—l>文件

在这种情况中，如果不存在文件，也要建立起文件。

可以通过写入从文件中取得命令的标准输入，而不从终端上取得命令的标准输入。举例说

wc<文件

命令wc读取这条命令的标准输入（在这种情况中按文件改变方向）并打印找到的字符数，字数和行数。如果只需要行数，外么可以使用

wc—l<文件

1．4管道和筛选程序

通过写入以|指示的'管道'算符，就好像在ls—l|wc中一样，可以把一条命令的标准输出连接到另一条命令的标准输入。用这种方法连接起来的两条命令就构成一条管道，这种管道的总效果与下式一样

ls—l >文件；wc <文件

但不使用文件除外。代替两个进程是用一条管道来连接（参阅管道（2）），并且并联运行。管道是单向的，并且在没有什么内容可以读取时由暂停wc完成同步，如果管道是满的，则由暂停ls来完成同步。

筛选程序是一条命令，它读入这条命令的标准输入，用某种方式转换这条命令，并打印结果作为输出。一个这种筛选程序grep从它的输入中选取那些包含某些指定的字符串的行。举例说

ls | grep old

打印那些从包含有字符串old的ls来的输出行。举例说

who | sort

将打印按字母排序的注册用户表。

一条管道也可能由两条以上的命令组成，举例说

ls | grep old | wc—l

打印包含字符串old的当前目录中的文件名称号数。

1.5 文件名生成

许多命令都接受是文件名的变元。举例说

ls—l main.c

打印有关文件main.c的信息。

外壳可以提供一种产生与一种模式相匹配的文件名清单的机理。

举例说

ls—l *c

在以c结束的当前目录中产生像ls变元一样的全部文件名。字符*是一个模式，它可以与任何包括空字符串在内的字符串相匹配。一般来说，指定的模式如下。

*	与任何包括空字符串在内的字符串匹配
?	与任何单字符匹配
[...]	与任何包括起来的字符之一匹配。用一个负号隔开的一对字符将与词法上介于这对字符之间的任何字符相匹配。

举例说，

[a—z]*

匹配由字母a到z之间的一个字母开头的当前目录中的所有名称。

/usr/fred/test/?

与由单字符构成的/usr/fred/test目录中的全部名称。如果没有找到与指定模式相匹配的文件名，则模式像一个变元一样不改变地通过。在按照某个指定的模式保存打入和选择名称时这个机理都有用处。这个机理还可以用来寻找文件。举例说

echo/usr/fred/*/core

在/usr/fred的子目录中寻找并打印所有内存文件的名称。（echo是一条打印命令的由空格隔开的变元的标准UNIX命令）。

这最后一种特点可能是比较昂贵的，它需要对／usr/fred的全部子目录进行扫视。

在为模式给定的一般规则中有一个例外。在文名开头的字符'.'必须显式地匹配。

```
echo *
```

因此将回送当前目录中所有不以'.'开头的文件名。

```
echo .*
```

将回送所有那些以'.'开头的文件名。这样作可以避免由于粗心大意而造成名称'.'和'..'的匹配。'.'和'..'分别代表'当前目录'和'父目录'。（注意ＬＳ抑制文件'.'和'..'的信息）。

1.6加引号

有些字符，如像＜＞＊？｜&之类对外壳具有特殊含意，这些字符叫做元字符。在附录B中给出一张完整的元字符表。任何字符以／开头都用引号抬起来、并且失去了这个字符的特殊含意。\被省略，因此

```
echo \?
```

将回送一个单独的?，而

```
echo \\
```

将回送一个单独的\。为了允许长空字符串继续到一行以上，就不要管序列\新行。\便于把单个字符用引号抬起来。当一个以上的字符需要用引号抬起时，上述机理不大灵活而且容易出错。把字符

串包括在单引号之间可以引用一串字符。举例说

```
echo XX '* * * * ' X X
```

将回送

```
X X* * * * X X
```

用引号括起的字符串不能包含单引号，但可以包含保存的新行。这种加引号的机理最简单，并推荐供临时之用。

第三种加引号的机理利用双引号，这种机理也可以用来防止对一些元字符，但不是对全体元字符进行解释。详细介绍在后面的第3·4节中。

1·7提示

当从终端使用外壳时，外壳在读入一条命令之前将发出一个提示符。在没有明确指明时，这个提示符是'$'。它可以由说明来改变，举例说

```
ps1 = yesdear
```

把提示符置成字符串 yesdear。如果打入一个新行，又需要下一步的输入，那么外壳将发出提示符'>'。有时候由于错打引号就可能造成这种情况。如果不希望产生这种情况，那么一个中断（DEL）将使外壳回转去读另外一条命令。这个提示符可以由说明来改变，举例说

```
ps2 = more
```

1·8外壳和注册

按照Login（1）可以调用外壳来读取和执行在终端上打入的命令·如果用户的注册目录包含文件· profile，那么认为这个目录包含着命令，并且在从终端读入任何一条命令之前由外壳读入这个用户的注册目录。

1·9 小结

1· ls

打印当前目录中的文件名。

· ls > file

把输出从ls放入file中。

· ls | wc—l

打印当前目录中的文件号数·

· ls | grep old

打印那些包含字符串old在内的文件名

· ls | grep old | wc—l

打印那些文件名称包含字符串old的文件号数·

· cc pgm·c &

在后台运行cc。

2·0外壳过程

可以用外壳来读取和执行一个文件中包含的命令·举例说

sh file [args···]

调用从file来的命令·这种file叫做命令过程或外壳过程·可

以和调用一起供给变元，并在采用定位参数 $1，$2，···的文件中访问这些变元·举例说，如果文件 Wg 包括

```
Who  |  grep $1
```

则

```
Sh  Wg fred
```

等效于

```
who | grep fred
```

UNIX文件有三种独立的属性，读，写和执行·UNIX的键盘命令 chmod（1）可以用来使一个文件可以执行·举例说

```
chmod +X  Wg
```

将保证文件 Wg 有执行状态·按照这种情况，命令

```
Wg fred
```

等效于

```
sh wg fred
```

这样作就允许外壳过程和程序交换使用·在上述两者之一的情况下，需建立起一个新进程来运行命令·

就和提供的定位参数的名称一样，调用中的定位参数号也可以用作 $#·执行的文件名称被用作 $0·

特殊外壳参数 $*被用来代替除 $0 之外的所有定位参数·这种替换的典型用途是提供一些没有明确指定时的变元·如像在

```
nroff — T450 — ms $*
```

中、它予先直接把一些变元附加给那些已经给定了的变元中。

2·1控制统一for

外壳过程的通常用途是在变元（s1，s2，···）之间循环，每个变元执行一次命令。这种过程的一个例子是tel，tel查找包含下列形式的行的文件/usr/lib/te/nos

```
...
fred mho123
bert mho789
...
```

tel的正文是

```
for i
do grep $1 /usr/lib/telnos; done
```

命令

```
tel fred
```

打印那些包含字符串fred的/usr/lib/telnos中的行。

```
tel fred bert
```

打印那些包含后面跟着bert的fred的行。

for循环符号由外壳识别，并具有通用的形式

```
for name in W1 W2 ···
do command—list
done
```

命令表是一系列一条或多条简单的命令，这些命令由新行符或分号隔开。此外，像d o和d o n e这样的保留字只根据新行或分号识别。名称 name是外壳变量，每次依次执行跟在d o后面的命令表时把这个外壳变量置成字w l　w 2。。。。如果没有指出 i n　w l　w 2，则每个定位参数执行一次循环；也就是说假定是i n　S ＊。

另一个使用f o r循环的例子是建立一条命令，这条命令的正文是

```
for i do >Si;done
```

命令

```
create alpha beta
```

保证两个空文件alpha和beta存在，而且是空的。符号> fi le可以用于它自己的来建立或清除文件内容。还应注意在done之前需要有一个分号（或新行）。

2．2控制流—case

通过case符号可以提供多路分支。举例说

```
case S # in
  (1) cat >> $1 ;;
  (2) cat >> $2 <S1 ;;
   *) echo 'usage:append[form]to' ;;
esac
```

是一条附加的命令。当与一个变元一起调用，而这个变元像是

append file（附加文件）

时，S#是字符串|，而标准输入则用 cat 命令复制到文件结尾处。

append file1 file2

把file1（文件1）的内容附加到file2（文件2）中。如提供给append的变元数不是1或2，那么就打印一个信息来指示正当的用途。

case命令的一般形式是

```
case word in
    pattern)command—list
    ...
esac
```

外壳试图把字和每个模式相匹配，匹配的顺序是模式出现的顺序。如果找到了匹配，就执行有关的命令表，并且完成case命令的执行。由于*是与任何一个字符串相匹配的模式，所以*可以用在不指明的情况中。

应当注意：没有进行检查来保证只有一个模式匹配case变元。找到的第一个匹配规定要执行的命令集。在下面的例子中不执行跟在第二个*的命令。

```
case S# in
    *)...  ;;
```

```
        *)... ;;
    esac
```

另外一个使用case结构的例子是区分一个变元之间的不同形式。下列例子是一段CC命令。

```
for i
do case Si in
    -[ocs])... ;;
    -*)echo 'unknown flag Si ' ;;
    *C) /lib/C0 Si ... ;;
    *) echo 'unexpected argument Si';;
      esac
done
```

为了允许同样的命令与一个以上的模式相联系，case命令提供用一个|隔开的备用模式。举例说

```
case Si in
    -x|-y)...
esac
```

等效于

```
case Si in
    -[xy]) ...
esac
```