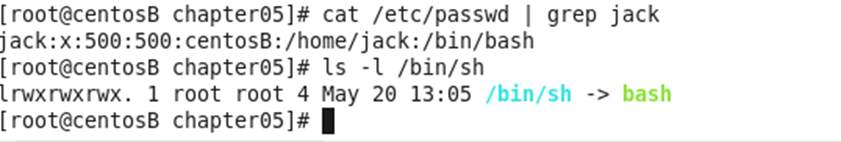
## 1.Shell类型

系统启动什么样的shell程序取决于你个人的用户ID配置。在/etc/passwd文件中，在用户ID记 录的第7个字段中列出了默认的shell程序。只要用户登录到某个虚拟控制台终端或是在GUI中启动 终端仿真器，默认的shell程序就会开始运行。

实例：用户Jack使用GNU bash shell作为自己的默认shell程序



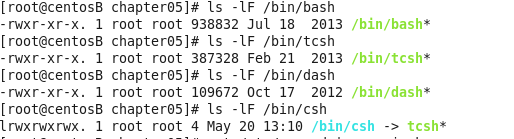
注意，用户christine默认的交互shell是/bin/bash，也就是bash shell。但是作为默认系统shell 的/bin/sh也被设置为bash shell, 但有时会不一样。

bash shell程序位于/bin目录内。从长列表中可以看出/bin/bash（bash shell）是一个可执行程序，如下所示。

CentOS发行版中还有其他一些shell程序。其中包括tcsh，它源自最初的C shell

另外还包括ash shell的Debian版：

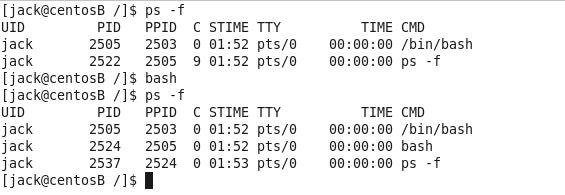
最后，C shell的软链接（参见第3章）指向的是tcsh shell，如下图所示



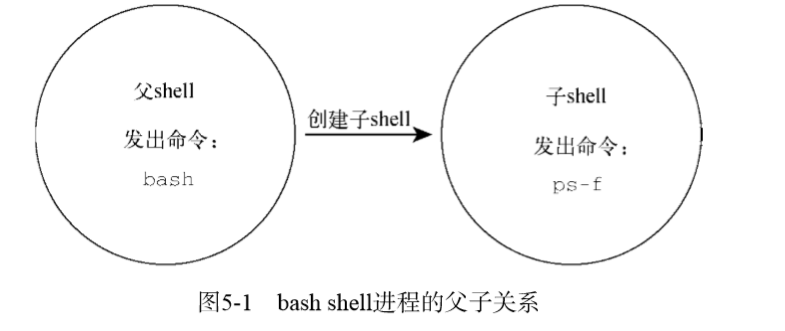
## 2.Shell的父子关系

用于登录某个虚拟控制器终端或在GUI中运行终端仿真器时所启动的默认的交互shell，是一 个父shell。本书到目前为止都是父shell提供CLI提示符，然后等待命令输入。 在CLI提示符后输入/bin/bash命令或其他等效的bash命令时，会创建一个新的shell程序。 这个shell程序被称为子shell（child shell）。子shell也拥有CLI提示符，同样会等待命令输入。

如下图所示：

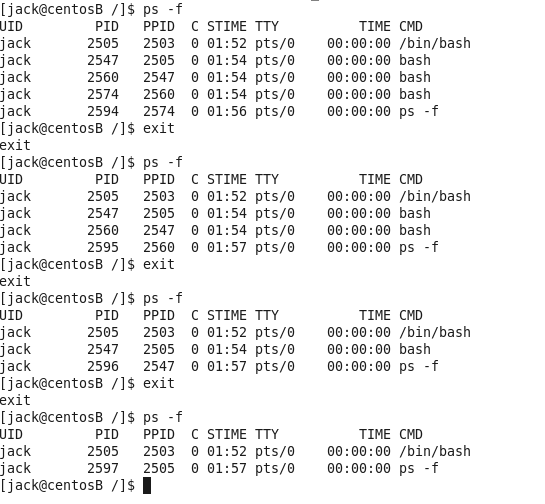
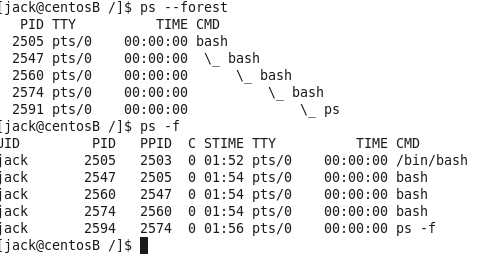
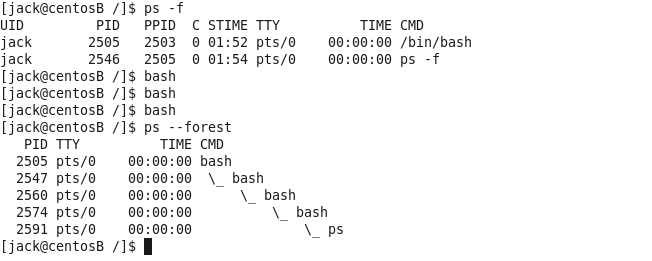


输入命令bash之后，一个子shell就出现了。第二个ps -f是在子shell中执行的。可以从显示 结果中看到有两个bash shell程序在运行。第一个bash shell程序，也就是父shell进程，其原始进程 ID是2505。第二个bash shell程序，即子shell进程，其PID是2524。注意，子shell的父进程ID（PPID） 是2505，指明了这个父shell进程就是该子shell的父进程。图5-1展示了这种关系。



在生成子shell进程时，只有部分父进程的环境被复制到子shell环境中。这会对包括变量在内的一些东西造成影响，我们会在变量笔记详细介绍中。 子shell（child shell，也叫subshell）可以从父shell中创建，也可以从另一个子shell中创建。如下图所示，

在下面的例子中，bash命令被输入了三次。这实际上创建了三个子shell。ps -forest命令 展示了这些子shell间的嵌套结构



### 2.1.进程列表

1.命令列表

你可以在一行中指定要依次运行的一系列命令。这可以通过命令列表来实现，只需要在命令 之间加入分号（;）即可，所有的命令依次执行，不存在任何问题。不过这并不是进程列表。命令列 表要想成为进程列表，这些命令必须包含在括号里。

尽管多出来的括号看起来没有什么太大的不同，但起到的效果确是非同寻常。括号的加入使命令列表变成了进程列表，生成了一个子shell来执行对应的命令。

**说明** 进程列表是一种命令分组（command grouping）。另一种命令分组是将命令放入花括号中， 并在命令列表尾部加上分号（;）。语法为{ command; }。使用花括号进行命令分组并不会像进程列表那样创建出子shell。

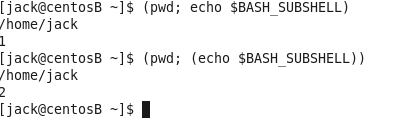
如何证明？

得借助一个使用了环境变量的命令。这个命令就是echo $BASH\_SUBSHELL。如果该命令返回0，就表明没有子shell。如果返回 1或者其他更大的数字，就表明存在子shell。

返回的数字就是子shell的个数。



括号的使用



注意，在第一个进程列表中，数字1表明了一个子shell，这个结果和预期的一样。但是在第 二个进程列表中，在命令echo $BASH\_SUBSHELL外面又多出了一对括号。这对括号在子shell中产生了另一个子shell来执行命令。因此数字2表明的就是这个子shell。 在shell脚本中，经常使用子shell进行多进程处理。但是采用子shell的成本不菲，会明显拖慢处理速度。在交互式的CLI shell会话中，子shell同样存在问题。它并非真正的多进程处理，因为 终端控制着子shell的I/O。

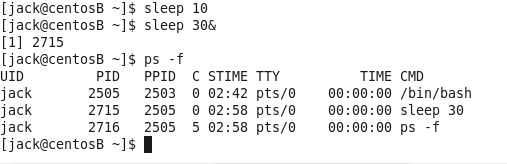
### 2.2.后台模式

在交互式shell中，一个高效的子shell用法就是使用后台模式，在后台模式中运行命令可以在处理命令的同时让出CLI，以供他用。

实例：

sleep命令接受一个参数，该参数是你希望进程等待（睡眠）的秒数。这个命令在脚本中常 用于引入一段时间的暂停。命令sleep 10会将会话暂停10秒钟，然后返回shell CLI提示符。

要想将命令置入后台模式，可以在命令末尾加上字符&。把sleep命令置入后台模式可以让我们利用ps命令来检查一下。



说明：sleep命令会在后台（&）睡眠10秒。当它被置入后台，在shell CLI提示符返回 之前，会出现两条信息。第一条信息是显示在方括号中的后台作业（background job）号（1）。 第二条是后台作业的进程ID（2715）。

ps命令用来显示各种进程。我们可以注意到命令sleep 30已经被列出来了。在第二列显 示的进程ID（PID）和命令进入后台时所显示的PID是一样的，都是2715

除了ps命令，你也可以使用jobs命令来显示后台作业信息。jobs命令可以显示出当前运行 在后台模式中的所有用户的进程（作业）。



jobs命令在方括号中显示出作业号（1）。它还显示了作业的当前状态（running）以及对

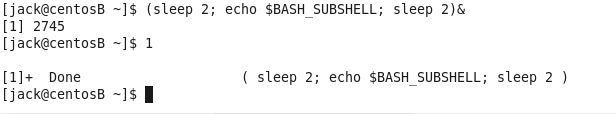
应的命令（sleep 30 &）。

利用jobs命令的-l（字母L的小写形式）选项，你还能够看到更多的相关信息。除了默认信息之外，-l选项还能够显示出命令的PID。一旦后台作业完成，就会显示出结束状态。

窍门： 需要提醒的是：后台作业的结束状态可未必会一直等待到合适的时候才现身。当作业结束状态突然出现在屏幕上的时候，你可别吃惊啊

### 2.3. 进程列表置入后台

把进程列表置入后台会产生一个作业号和进程ID，然后返回到提示符。不过奇怪的是表明单 一级子shell的数字1显示在了提示符的旁边！不要不知所措，只需要按一下回车键，就会得到另 一个提示符。 在CLI中运用子shell的创造性方法之一就是将进程列表置入后台模式。你既可以在子shell中 进行繁重的处理工作，同时也不会让子shell的I/O受制于终端



### 2.4.协程

协程可以同时做两件事。它在后台生成一个子shell，并在这个子shell中执行命令。

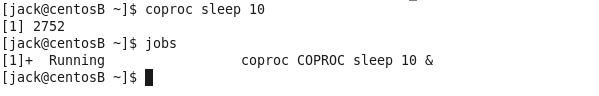
要进行协程处理，得使用coproc命令，还有要在子shell中执行的命令。除了会创建子shell

之外，协程基本上就是将命令置入后台模式。

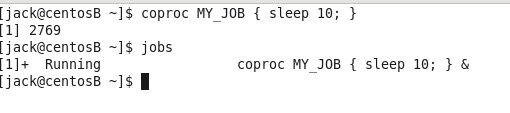
当输入coproc命令及其参数之后，你会发现启用了一个后台作业。屏幕上会显示出后台作

业号（1）以及进程ID（2752）。

jobs命令能够显示出协程的处理状态。



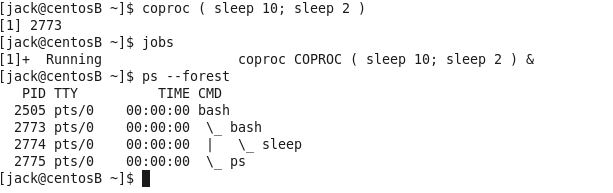
在上面的例子中可以看到在子shell中执行的后台命令是coproc COPROC sleep 10。COPROC 是coproc命令给进程起的名字。你可以使用命令的扩展语法自己设置这个名字。



通过使用扩展语法，协程的名字被设置成My\_Job。这里要注意的是，扩展语法写起来有点 麻烦。必须确保在第一个花括号（{）和命令名之间有一个空格。还必须保证命令以分号（;）结 尾。另外，分号和闭花括号（}）之间也得有一个空格。

说明 协程能够让你尽情发挥想象力，发送或接收来自子shell中进程的信息。只有在拥有多个协 程的时候才需要对协程进行命名，因为你得和它们进行通信。否则的话，让coproc命令 将其设置成默认的名字COPROC就行了。

将协程与进程列表结合起来产生嵌套的子shell。只需要输入进程列表，然后把命令coproc放在前面就行了。

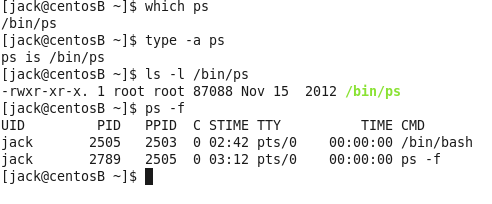


记住，生成子shell的成本不低，而且速度还慢。创建嵌套子shell更是火上浇油！ 在命令行中使用子shell能够获得灵活性和便利。要想获得这些优势，重要的是理解子shell的 行为方式。

## 3.内建命令与外部命令

### 1.外部命令

外部命令，有时候也被称为文件系统命令，是存在于bash shell之外的程序。它们并不是shell 程序的一部分。外部命令程序通常位于/bin、/usr/bin、/sbin或/usr/sbin中。 ps就是一个外部命令。你可以使用which和type命令找到它

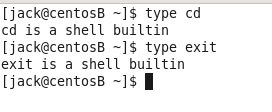
.

作为外部命令，ps命令执行时会创建出一个子进程。在这里，ps命令的PID是2789，父PID 是2505。作为父进程的bash shell的PID是2505

说明 就算衍生出子进程或是创建了子shell，你仍然可以通过发送信号与其沟通，这一点无论是 在命令行还是在脚本编写中都是极其有用的。发送信号（signaling）使得进程间可以通过 信号进行通信。信号及其发送会在第16章中讲到。

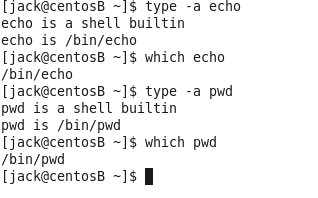
### 2.内建命令

内建命令和外部命令的区别在于前者不需要使用子进程来执行。它们已经和shell编译成了一 体，作为shell工具的组成部分存在。不需要借助外部程序文件来运行。 cd和exit命令都内建于bash shell。可以利用type命令来了解某个命令是否是内建的。



因为既不需要通过衍生出子进程来执行，也不需要打开程序文件，内建命令的执行速度要更 快，效率也更高

要注意，有些命令有多种实现。例如echo和pwd既有内建命令也有外部命令。两种实现略有不同。要查看命令的不同实现，使用type命令的-a选项



命令type -a显示出了每个命令的两种实现。注意，which命令只显示出了外部命令文件。

窍门 对于有多种实现的命令，如果想要使用其外部命令实现，直接指明对应的文件就可以了。 例如，要使用外部命令pwd，可以输入/bin/pwd。

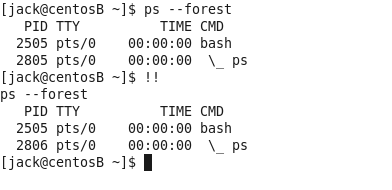
### 3.history命令

一个有用的内建命令是history命令。bash shell会跟踪你用过的命令。你可以唤回这些命令 并重新使用。 要查看最近用过的命令列表，可以输入不带选项的history命令。

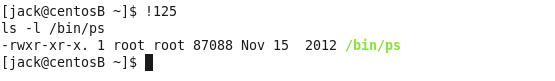
**窍门** 你可以设置保存在bash历史记录中的命令数。要想实现这一点，你需要修改名为HISTSIZE 的环境变量

你可以唤回并重用历史列表中最近的命令。这样能够节省时间和击键量。输入!!，然后按回 车键就能够唤出刚刚用过的那条命令来使用。

当输入!!时，bash首先会显示出从shell的历史记录中唤回的命令。然后执行该命令。 命令历史记录被保存在隐藏文件.bash\_history中，它位于用户的主目录中。这里要注意的是， bash命令的历史记录是先存放在内存中，当shell退出时才被写入到历史文件中。



编号为125的命令从命令历史记录中被取出。和执行最近的命令一样，bash shell首先显示出从 shell历史记录中唤回的命令，然后执行该命令。 使用bash shell命令历史记录能够大大地节省时间。利用内建的history命令能够做到的事情 远不止这里所描述的。可以通过输入man history来查看history命令的bash手册页面。



注意：错误的命令仍然会被记录

### 4.命令别名

alias命令是另一个shell的内建命令。命令别名允许你为常用的命令（及其参数）创建另一 个名称，从而将输入量减少到最低。

