HopeCao

大天使

UID：9331
注册：2002-5-24
最后登录： 2008-02-05
帖子：1922
精华：33

可用积分：846
信誉积分：100
专家积分：0 (本版)

状态：...离线...

[个人空间] [短信] [博客]

1楼 发表于 2003-6-6 11:45 [报告]

bash

前言

本文译自《Slackware Linux Unleashed》（第三版）一书的 bash 一章，但做了一些必要的删节，并且有的地方根据实际情况作了较大的改动，必要的话请参考原文。对本文有任何问题的话请与我联系：con@nease.net

简介

本文将较为详细地介绍 Linux 下最常用的 shell，bash。bash(Bourne Again Shell) 是 Linux 的缺省 shell，并被大多数用户所使用，在本文中你将学会：

什么是shell

Linux上最常见的shell

命令补齐(Command-line completion)和通配

命令历史记录(Command history)和别名

重定向和管道

改变提示符

进程控制(Job control)

如何用户化配置你的bash shell

另外你还将看到一些最常用的bash命令和内存变量的用法，读完本文后你将能更快和更有效地使用bash。

Shells in a Nutshell

什么是shell？总之这个词好象总是出现在 Linux 里，但很多新用户并不清楚它的明确的含义。本节将给出一个确切的解释并说明为什么在Linux中它显得如此的重要。

什么是Shell?

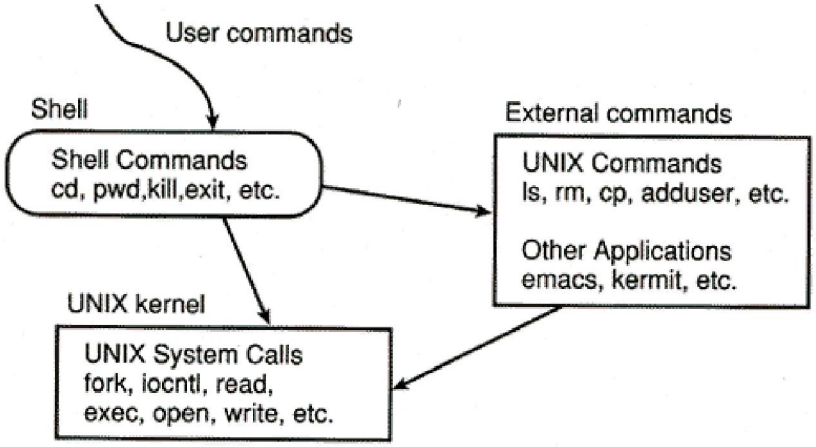
shell是你（用户）和Linux（或者更准确的说，是你和Linux内核）之间的接口程序。你在提示符下输入的每个命令都由shell先解释然后传给Linux内核。

注意：如果你熟悉m$-dos，你将认出这很象DOS 用户和 COMMAND.COM之间的关系。唯一的区别是command.com 的功能远不能和shell 相提并论。

---

shell 是一个命令语言解释器（command-language interpreter）。拥有自己内建的 shell 命令集。此外，shell也能被系统中其他有效的Linux 实用程序和应用程序（utilities and application programs）所调用。

不论何时你键入一个命令，它都被Linux shell所解释。一些命令，比如打印当前工作目录命令（pwd），是包含在Linux bash内部的（就象DOS的内部命令）。其他命令，比如拷贝命令（cp）和移动命令（rm），是存在于文件系统中某个目录下的单独的程序。而对用户来说，你不知道（或者可能不关心）一个命令是建立在shell内部还是一个单独的程序。



附图 展示了 shell 是如何执行命令解释的，shell 首先检查命令是否是内部命令，不是的话再检查是否是一个应用程序，这里的应用程序可以是Linux本身的实用程序，比如ls 和 rm，也可以是购买的商业程序，比如 xv，或者是公用软件（public domain software），就象 ghostview。然后shell试着在搜索路径里寻找这些应用程序。搜索路径是一个能找到可执行程序的目录列表。如果你键入的命令不是一个内部命令并且在路径里没有找到这个可执行文件，将会显示一条错误信息。而如果命令被成功的找到的话，shell的内部命令或应用程序将被分解为系统调用并传给Linux内核。

shell的另一个重要特性是它自身就是一个解释型的程序设计语言，shell 程序设计语言支持在高级语言里所能见到的绝大多数程序控制结构，比如循环，函数，变量和数组。shell 编程语言很易学，并且一旦掌握后它将成为你的得力工具。任何在提示符下能键入的命令也能放到一个可执行的shell程序里，这意味着用shell语言能简单地重复执行某一任务。

shell 如何启动

shell在你成功地登录进入系统后启动，并始终作为你与系统内核的交互手段直至你退出系统。你系统上的每位用户都有一个缺省的shell。每个用户的缺省shell在系统里的passwd文件里被指定，该文件的路径是/etc/passwd。passwd文件里还包含有其他东西：每个人的用户ID号，一个口令加密后的拷贝和用户登录后立即执行的程序，（注：为了加强安全性，现在的系统一般都把加密的口令放在另一个文件--shadow中，而passwd中存放口令的部分以一个x字符代替）虽然没有严格规定这个程序必须是某个Linux shell，但大多数情况下都如此。

最常用的shell

在Linux 和 UNIX系统里可以使用多种不同的shell可以使用。最常用的几种是 Bourne shell (sh), C shell (csh), 和 Korn shell (ksh)。三种shell 都有它们的优点和缺点。Bourne shell 的作者是 Steven Bourne。它是 UNIX 最初使用的shell 并且在每种 UNIX 上都可以使用。Bourne shell 在 shell 编程方面相当优秀，但在处理与用户的交互方面作得不如其他几种 shell。

C shell 由 Bill Joy 所写，它更多的考虑了用户界面的友好性。它支持象命令补齐（command-line completion）等一些 Bourne shell 所不支持的特性。普遍认为C shell 的编程接口做的不如 Bourne shell, 但 C shell 被很多 C 程序员使用因为 C shell的语法和 C语言的很相似，这也是C shell名称的由来。

Korn shell (ksh) 由 Dave Korn 所写。它集合了C shell 和 Bourne shell 的优点并且和 Bourne shell 完全兼容。

除了这些 shell 以外，许多其他的 shell 程序吸收了这些原来的 shell 程序的优点而成为新的 shell 。在 Linux 上常见的有 tcsh (csh 的扩展)，Bourne Again shell(bash, sh 的扩展), 和Public Domain Korn shell (pdksh, ksh 的扩展)。bash 是大多数Linux 系统的缺省 shell。

The Bourne Again Shell

Bourne Again shell (bash), 正如它的名字所暗示的，是 Bourne shell 的扩展。bash 与 Bourne shell 完全向后兼容，并且在 Bourne shell 的基础上增加和增强了很多特性。bash 也包含了很多 C 和 Korn shell 里的优点。bash 有很灵活和强大的编程接口，同时又有很友好的用户界面。

为什么要用 bash 来代替 sh 呢？Bourne shell 最大的缺点在于它处理用户的输入方面。在 Bourne shell 里键入命令会很麻烦，尤其当你键入很多相似的命令时。而 bash 准备了几种特性使命令的输入变得更容易。

命令补齐（Command-Line Completion）

通常你在 bash （或任何其他的 shell）下输入命令时你不必把命令输全 shell 就能判断出你所要输入的命令。例如，假定当前的工作目录包含以下的文件和子目录：

News/ bin/ games/ mail/ samplefile test/

如果你要进入 test 子目录，你将会输入以下的命令：

cd test

这个命令能够满足你的需要，但 bash还提供了稍微不同的方法来完成同样的事。因为 test 是当前目录里唯一以字母 t开头的子目录，bash在你只输入字母 t后就能判断出你要做什么了：

cd t

在你键入那个字母后，唯一的可能就是 test 。想让 bash 帮你结束命令的话，按下 Tab 键：

cd t<tab>;

当你这样做以后，bash 将帮你补齐命令并显示在屏幕上。但在你按下回车键以前命令并没有被执行，bash 会让你检验补齐的命令是否是你真正需要的。在输入象这样短的命令时你也许看不出它的价值所在，甚至在命令很短时还会简慢输入的速度，但是当你要输入的命令有点长时，你会发现这个特性是多么的美好。

但是当目录里有不止一个以字母 t 开头的文件时会发生什么情况呢？在你使用命令补齐时会有问题，让我们看看下面的情况，当前目录里有下列内容：

News/ bin/ mail/ samplefile test/ tools/ working/

现在这个目录里有两个以字母 t 开头的文件。假设你仍然想进入 test 子目录，如何使用命令补齐呢？如果你象先前那样键入：

cd t<tab>;

bash 将不知道你到底想进入哪个子目录，因为给出的信息并不唯一。如果你这样做了的话，bash 将发出一声蜂鸣提醒你没有足够的信息来补齐你的命令。蜂鸣之后bash 并不改变输入的命令，这将使你能在原来的基础上再输入更多的信息，在这个例子中你仅需再键入一个 e 并再按一下 Tab 键，这时 bash 就有足够的信息来完成你的命令了：

cd test

当你输入命令时不论何时按下 Tab 键，bash 都将尽其所能地试图补齐命令，不行的话会发出蜂鸣来提醒你需要更多的信息。你需要键入更多的字符，并再次按下 Tab 键，重复这个过程直至你期望的命令出现。

通配符

另一个使命令输入变得更简单的方法是在命令中使用通配符。bash 支持三种通配符：

CODE: [Copy to clipboard] [-]

*	匹配任何字符和任何数目的字符
?	匹配任何单字符
[...]	匹配任何包含在括号里的单字符

* 通配符的使用有些象命令补齐。例如，假设当前目录包含以下文件：

News/ bin/ games/ mail/ samplefile test/

如果你想进入 test 目录，你将键入 cd test, 或者你想用命令补齐：

cd t<tab>;

现在有第三种方法来做同样的事。因为仅有一个文件以字母 t 开头，你也能用 * 通配符来进入该目录。键入下列命令：

cd t*

* 匹配任何字符和任何数目的字符，所以 shell 将把 t* 替换为 test（当前目录里唯一和通配方案匹配的文件）。当前目录里只有一个文件以字母 t 开头的话这将是可靠的。但是如果当前目录里有不止一个文件以字母 t 开头，shell 将试着进入第一个符合匹配方案的目录，这个目录是以字母表排序的第一个目录，这个目录也许是也许不是你所期望的。

通配符 * 的一个更实际的用途是通配你要执行的命令中的多个名字相似的文件。例如，假设当前目录里包含以下文件：

ch1.doc ch2.doc ch3.doc chimp config mail/ test/ tools/

如果你需要打印所有扩展名是 .doc 的文件，你能使用象这样简化的命令：

lpr *.doc

在这个例子中，bash 将把 *.doc 替换为当前目录下所有文件名符合通配方案的文件。在 bash 进行了替换后，该命令将被处理为：

lpr ch1.doc ch2.doc ch3.doc

lpr 命令将以ch1.doc, ch2.doc, 和 ch3.doc为参数被调用。

注意： 除了以上给出的例子外，还有几种方法：

```
lpr *doc
lpr *oc
lpr *c
```

通配符 ? 除了只能匹配单个字符外，其他功能都与通配符 * 相同，如果用通配符 ? 来打印前面提到的那个目录里所有扩展名是 .doc 的文件的话，键入下面的命令：

lpr ch?.doc

通配符[...]能匹配括号中给出的字符或字符范围。同样以前面的目录为例，打印那个目录里所有扩展名是 .doc 的文件，你可以键入下列命令之一：

lpr ch[123].doc

或者：

lpr ch[1-3].doc

命令历史记录

bash 也支持命令历史记录。这意味着 bash 保留了一定数目的你先前已经在shell 里输入过的命令。这个数目取决于一个叫做HISTSIZE的变量。有关 HISTSIZE 的更多信息，请看本文后面的“bash 变量”一节。

bash 把你先前输入的命令文本保存在一个历史列表中。当你用你的帐号登录后历史列表将根据一个历史文件被初始化。历史文件的文件名被一个叫 HISTFILE 的 bash变量指定。历史文件的缺省名字是 .bash_history。这个文件通常在你的用户目录中。（注意该文件的文件名以一个句号开头，这意味着它是隐含的，仅当你带 -a 或 -A参数的 ls 命令列目录时才可见）

仅将先前的命令存在历史文件里是没有用的，所以 bash 提供了几种方法来调用它们。使用历史记录列表最简单的方法是用上方向键。按下上方向键后最后键入的命令将出现在命令行上。再按一下则倒数第二条命令会出现，以此类推。如果上翻多了的话也可以用向下的方向键来下翻。（和 DOS 实用程序doskey一样）如果需要的话，显示在命令行上的历史命令可以被编辑。

另一个使用命令历史文件的方法是用 bash 的内部命令 history 和 fc(fix 命令)命令来显示和编辑历史命令。history 命令能以两种不同的方法来调用。第一种是：

history [n]

当 history 命令没有参数时，整个历史命令列表的内容将被显示出来。下面是一个命令历史列表的例子：

CODE: [Copy to clipboard] [-]

```
1  mkdir /usr/games/pool
2  cp XpoolTable-1.2.linux.tar.z /usr/games/pool
3  cd /usr/games/pool/
4  ls
5  gunzip XpoolTable-1.2.linux.tar.z
6  tar -xf XpoolTable-1.2.linux.tar
7  ls
8  cd Xpool
9  ls
10  xinit
11  exit
12  which zip
13  zip
14  more readme
15  vi readme
16  exit
```

使用 n 参数的作用是仅有最后 n 个历史命令会被列出。例如，history 5 显示最后 5 个命令。

调用 history 命令的第二种方法用于修改命令历史列表文件的内容。命令的语法如下：

CODE: [Copy to clipboard] [-]

```
history [-r|w|a|n] [filename]
```

这种形式中，-r 选项告诉 history 命令读命令历史列表文件的内容并且把它们当作当前的命令历史列表。-w 选项将把当前的命令历史记录写入文件中并覆盖文件原来的内容。-a 选项把当前的命令历史记录追加到文件中。-n 选项将读取文件中的内容并加入到当前历史命令列表中。如果 filename 选项没有被指定，history 命令将用变量HISTFILE 的值来代替。

fc 命令能用两种方法来编辑历史命令。第一种使用下列语法：

CODE: [Copy to clipboard] [-]

```
fc [-e editor_name] [-n] [-l] [-r] [first] [last]
```

这里所有参数都是可选的。-e editor_name 选项用来指定用于编辑命令的文本编辑器。 first 和 last 选项用于选择列出历史命令的范围，既可以是数字也可以是字符串。-n 选项禁止列出命令的编号。-r 选项反向列出匹配的命令。-l 选项把匹配的命令行列在屏幕上（而不是在编辑器中）。如果-e editor_name参数没有被指定，则以变量FCEDIT的值来代替，如果该变量不存在的

话，则用变量EDITOR的值来代替，都不存在的话将使用vi编辑器。

别名

bash 的另一个使你的工作变得轻松的方法是命令别名。命令别名通常是其他命令的缩写，用来减少键盘输入。例如，你经常要键入如下的命令，你也许会倾向于为它建立一个别名来减少工作量：

```
cd /usr/X11/lib/X11/fvwm/sample-configs
```

假如为这个长命令建立一个名为goconfig的别名，在bash提示符下键入如下命令：

```
alias goconfig='cd /usr/X11/lib/X11/fvwm/sample-configs'
```

现在，除非你退出bash，键入goconfig将和原来的长命令有同样的作用。如果想取消别名，可以使用下面的命令：

```
unalias goconfig
```

这是一些很多用户认为有用的别名，你可以把它们写入你的.profile文件中提高工作效：

CODE: [Copy to clipboard] [-]

```
alias ll='ls -l'
    alias log='logout'
    alias ls='ls -F'
```

如果你是一名DOS用户并且习惯了DOS命令，你能下面的别名定义使你的 Linux 表现得象DOS 一样：

CODE: [Copy to clipboard] [-]

```
alias dir='ls'
    alias copy='cp'
    alias rename='mv'
    alias md='mkdir'
    alias rd='rmdir'
```

注意： 在定义别名时，等号的两头不能有空格，否则 shell 不能决定你需要做 什么。仅在你的命令中包含有空格或特殊字符时才需要引号。

如果你键入不带任何参数的alias命令，将显示所有已定义的别名，例如：

CODE: [Copy to clipboard] [-]

```
alias dir='ls'
alias ll='ls -l'
alias ls='ls -F'
alias md='mkdir'
alias net='term < /dev/modem >; /dev/modem 2>; /dev/null&'
alias rd='rmdir'
```

输入重定向

输入重定向用于改变一个命令的输入源。一些命令需要在命令行里输入足够的信息才能工作。比如 rm，你必须在命令行里告诉 rm 它你要删除的文件。另一些命令则需要更详细的输入，这些命令的输入可能是一个文件。比如命令 wc 统计输入给它的文件里的文件里的字符数，单词数和行数。如果你仅在命令行上键入 wc <enter>；，wc 将等待你告诉它要统计什么，这时 bash 就好象死了一样，你键入的每样东西都出现在屏幕上，但什么事也不会发生。这是因为 wc 命令正在为自己收集输入。如果你按下Ctrl-D，wc 命令的结果将被写在屏幕上。如果你输入一个文件名做参数，象下面的例子一样，wc 将返回文件所包含的字符数，单词数，和行数：

```
wc test
```

```
11 2 1
```

另一种把test文件内容传给 wc 命令的方法是重定向 wc 的输入。< 符号在bash里用于把当前命令的输入重定向为指定的文件。所以可以用下面的命令来把 wc 命令的输入重定向为 test 文件：

```
wc < test

11 2 1
```

输入重定向并不经常使用因为大多数命令都以参数的形式在命令行上指定输入文件的文件名。尽管如此，当你使用一个不接受文件名为输入参数的命令，而需要的输入又是在一个已存在的文件里时，你就能用输入重定向解决问题。

输出重定向

输出重定向比输入重定向更常用。输出重定向使你能把一个命令的输出重定向到一个文件里，而不是显示在屏幕上。

很多情况下都可以使用这种功能。例如，如果某个命令的输出很多，在屏幕上不能完全显示，你能把它重定向到一个文件中，稍后再用文本编辑器来打开这个文件；当你想保存一个命令的输出时也可以使用这种方法。还有，输出重定向可以用于把一个命令的输出当作另一个命令的输入时。（还有一种更简单的方法可以把一个命令的输出当作另一个命令的输入，就是使用管道，管道的使用将在本文的“管道”一节介绍）

输出重定向的使用与输入重定向很相似，但是输出重定向的符号是 >; 。

注意：记忆输入／输出重定向符号的最好方法是把<看作是一个漏斗，漏斗的小口　　指向需要输入的命令（因为需要接受输入的命令会在 <的左手边），而把>;　　当作一个大口指向有输出的命令的漏斗。

重定向举例，当你要把 ls 命令的输出保存为一个名为 directory.out 的文件时，你可以使用下面的命令：

CODE: [Copy to clipboard] [-]

```
ls >; directory.out
```

管道

管道可以把一系列命令连接起来。这意味着第一个命令的输出会通过管道传给第二个命令而作为第二个命令的输入，第二个命令的输出又会作为第三个命令的输入，以此类推。而管道行中最后一个命令的输出才会显示在屏幕上（如果命令行里使用了输出重定向的话，将会放进一个文件里）。

你能通过使用管道符 | 来建立一个管道行，下面的示例就是一个管道行：

CODE: [Copy to clipboard] [-]

```
cat sample.text | grep "High" | wc -l
```

这个管道将把 cat 命令（列出一个文件的内容）的输出送给grep命令。grep 命令在输入里查找单词 High，grep命令的输出则是所有包含单词 High的行，这个输出又被送给 wc命令。带 -l选项的 wc命令将统计输入里的行数。假设 sample.txt的内容如下：

CODE: [Copy to clipboard] [-]

```
Things to do today:

Low: Go grocery shopping

High: Return movie

High: Clear level 3 in Alien vs. Predator

Medium: Pick up clothes from dry cleaner
```

管道行将返回结果 2，指出你今天有两件很重要的事要做：

CODE: [Copy to clipboard] [-]

```
cat sample.text | grep "High" | wc -l

2
```

提示符

bash 有两级用户提示符。第一级是你经常看到的 bash 在等待命令输入时的提示符。缺省的一级提示符是字符$（如果是超级用户，则是#号）。你可以通过改变bash 的PS1变量的值来改变你的缺省提示符，例如：

CODE: [Copy to clipboard] [-]

```
PS1="Please enter a command"
```

把bash shell 的提示符该为指定的字符串。

当bash 期待输入更多的信息以完成命令时显示第二级提示符。缺省的第二级提示符是 >;。 果你要改变第二级提示符，可以通过设置PS2变量的值来实现：

CODE: [Copy to clipboard] [-]

```
PS2="I need more information"
```

另外你还可以用特殊的字符来定义你的提示符，下面的列表列出了最常用的特殊字符。

提示符特殊字符代码

CODE: [Copy to clipboard] [-]

```
字符      含义
\!      显示该命令的历史记录编号。
\#      显示当前命令的命令编号。

\$      显示$符作为提示符，如果用户是root的话，则显示#号。

\\      显示反斜杠。

\d      显示当前日期。

\h      显示主机名。

\n      打印新行。

\nnn    显示nnn的八进制值。

\s      显示当前运行的shell的名字。

\t      显示当前时间。

\u      显示当前用户的用户名。

\W      显示当前工作目录的名字。

\w      显示当前工作目录的路径。
```

这些特殊字符能组合成很多种有用的提示符方案（也可以组合为很奇异的方案），例如把 PS1 设为：

CODE: [Copy to clipboard] [-]

```
PS1="\t"
```

这导致提示符显示当前的时间，就象下面的显示一样（提示符后面将不会有空格）：

02:16:15

而下面的设置：

CODE: [Copy to clipboard] [-]

```
PS1=\t
```

将导致提示符变成下面的样子：

t

这显示了设置中引号的重要性，下面的提示符串：

CODE: [Copy to clipboard] [-]

```
PS1="\t\\ "
```

会使提示符看起来象这个样子：

02:16:30\

这种情况下，提示符后面会有一个空格，因为引号里有一个空格。

作业控制（Job Control）

作业控制能够控制当前正在运行的进程的行为。特别地，你能把一个正在运行的进程挂起，稍后再恢复它的运行。bash 保持

对所有已启动的进程的跟踪，你能在一个正在运行的进程的生命期内的任何时候把它挂起或是使它恢复运行。

按下 Ctrl-Z 使一个运行的进程挂起。bg 命令使一个被挂起的进程在后台恢复运行，反之 fg 命令使进程在前台恢复运行。这几个命令在当用户想在后台运行而意外的把它放到了前台时，经常被用到。当一个命令在前台被运行时，它会禁止用户与 shell 的交互，直到该命令结束。这通常不会造成麻烦，因为大多数命令很快就执行完了。如果你要运行的命令要花费很长的时间的话，我们通常会把它放到后台，以使我们能在前台继续输入其他命令。例如，你输入这个命令：

CODE: [Copy to clipboard] [-]

```
command find / -name "test" >; find.out
```

它将寻找整个文件系统中的名为test 的文件并把结果保存在一个叫fing.out的文件里。如果在前台运行的话，根据文件系统的大小，你的shell将有数秒甚至数分钟不能使用，你不想这样的话可以再输入以下面的内容：

control-z

bg

find 命令首先被挂起，再在后台继续被执行，并且你能马上回到bash下。

用户化配置bash

本文已经描述了许多用户化配置bash的方法。但知道现在为止，我们所做的改动都仅在当前运行的bash下才有效。一旦退出系统，所有的改动也随之消失了。为了保存这些用户化配置，你必须把它们保存到一个bash的初始化文件里。

你能把任何想每次进入cash都执行的命令放到初始化文件里。这个文件里最常见到的命令通常是alias和变量的初始化。bash 的初始化文件叫做 profile。每个使用bash的用户都有一个 .profile文件在他的用户目录里。bash在每次启动时都读取这个文件，并执行所有包含的命令。

下面的代码是缺省的.profile文件的内容。这个文件的位置在 /etc目录。如果你想设置自己的bash 的话把它拷到你的用户目录里（如果还没有的话）并命名为.profile。

注意： 有些setup程序会在建立用户时自动放一个.profile文件的拷贝在你的用 户目录里。但是并不是所有的都这么做，所以最好先检查一下你的用户目 录。记住所有以句点开头的文件都是隐含的，只有用ls -a或ls -A命令才能列出。

CODE: [Copy to clipboard] [-]

```
# commands common to all logins

export OPENWINHOME=/usr/openwin

export MINICOM="-c on"

export MANPATH=/usr/local/man:/usr/man/preformat:/usr/man:/X11/man:/usr/openwin /m
an

export HOSTNAME="`cat /etc/HOSTNAME`"

PATH="$PATH:/usr/X11/bin:$OPENWINHOME/bin:/usr/games:."

LESS=-MM

# I had problems using 'eval test' instead of 'TERM=', but you might want to # try
it anyway. I think with the right /etc/termcap it would work great.  # eval 'tset
-sQ "$TERM"'if [ "$TERM" = "" -o "$TERM" =
"unknown"]; then

TERM=linux

#PS1="'hostname':'pwd'# `

if [ "$SHELL" = "/bin/pdksh" -o "$SHELL" = "/bin/ksh" ]; then

PS1="! $"

elif [ "$SHELL" = "/bin/zsh" ]; then

PS1="%m:%~%# "

elif [ "$SHELL" = "/bin/ash" ]; then

PS1="$ "

else

PS1='\h:\w\$ `

fi

PS2='>; `

ignoreeof=10

export PATH DISPLAY LESS TERM PS1 PS2 ignoreeof

umask 022

# set up the color-ls environment variables:

if [ "$SHELL" = "/bin/zsh" l; then

  eval 'dircolors -z'

elif [ "$SHELL" = "/bin/ash" l; then

  eval 'dircolors -s'

else

  eval 'dircolors -b'

fi

echo

fortune

echo

export TAPE="/dev/nftape"
```

bash **命令概要**

这是几个最有用的bash**内部命令：**

alias: **设置**bash**别名。**

bash 还有许多命令，但这些是最常用的，想了解更详细的情况，请参考bash的手册--在提示符下键入 man bash。

bash 变量

这里是几个最有用的bash变量，包括变量名和简单描述。

CODE: [Copy to clipboard] [-]

EDITOR, FCEDIT: bsah fc 命令的缺省编辑器。

HISTFILE: 用于贮存历史命令的文件。

HISTSIZE: 历史命令列表的大小。

HOME: 当前用户的用户目录。

OLDPWD: 前一个工作目录。

PATH: bash寻找可执行文件的搜索路径。

PS1: 命令行的一级提示符。

PS2: 命令行的二级提示符。

PWD: 当前工作目录。

SECONDS: 当前shell开始后所流逝的秒数。

[/img]

您对本贴的看法：鲜花[0] 臭蛋[0]

_____

CODE: [Copy to clipboard]

```
while&#40;!Search&#40;MeiMei&#41;&#41;
    printf&#40;&quot;；） &quot;&#41;;
printf&#40;&quot;：） &quot;&#41;;
```


canhan
圣骑士

UID：61037
注册：2003-6-3
最后登录： 2008-06-24
帖子：109
精华：0

可用积分：94
信誉积分：100
专家积分：0 (本版)

状态：...离线...

[个人空间] [短信] [博客]

2楼 发表于 2003-6-6 11:47 **[报告]**

bash！

呵呵！还可以！

您对本贴的看法：鲜花[0] 臭蛋[0]

_____

上善若水，水善利万物而不争，居众人之所恶，故几于道。
居善地，心善渊，与善仁，言善信，政善治，动善时。夫唯不争，故无忧。

小皮
精灵

3楼 发表于 2003-6-6 11:50 **[报告]**

bash！

不错啊