

# 环境变量与文件查找

## 一、环境变量

### 1.变量

要解释环境变量，得先明白变量是什么，准确的说应该是 shell 变量，所谓变量就是计算机中用于记录一个值（不一定是数值，也可以是字符或字符串）的符号，而这些符号将用于不同的运算处理中。通常变量与值是一一对应的关系，可以通过表达式读取它的值赋值给其它变量，也可以直接指定数值赋值给任意变量。为了便于运算和处理，大部分的编程语言会区分变量的类型，用于分别记录数值、字符或者字符串等等数据类型。shell 中的变量也基本如此，有不同类型（但不专门指定类型名），可以参与运算，有作用域限定。

变量的作用域即变量的有效范围（比如一个函数中、一个源文件中或者全局范围），在该范围内只能有一个同名变量。一旦离开则该变量无效，如同不存在这个变量一般

在 Shell 中如何创建一个变量，如何给变量赋值和如何读取变量的值呢？这部分内容会在 [bash 脚本编程](#) 这门课中详细介绍，这里我简单举例说明一下：

使用 declare 命令创建一个变量名为 tmp 的变量

```
$ declare tmp
```

其实也可以不用 declare 预声明一个变量，直接即用即创建，这里只是告诉你 declare 的作用，这在创建其它指定类型的变量（如数组）时会用到

使用=号赋值运算符为变量 tmp 赋值为 shiyanlou

```
$ tmp=shiyanlou
```

读取变量的值，使用 echo 命令和\$符号（\$符号用于表示引用一个变量的值，初学者经常会忘记输入）

```
$ echo $tmp
```

```
shiyanlou:~/ $ declare tmp
shiyanlou:~/ $ tmp=shiyanlou
shiyanlou:~/ $ echo $tmp
shiyanlou
```

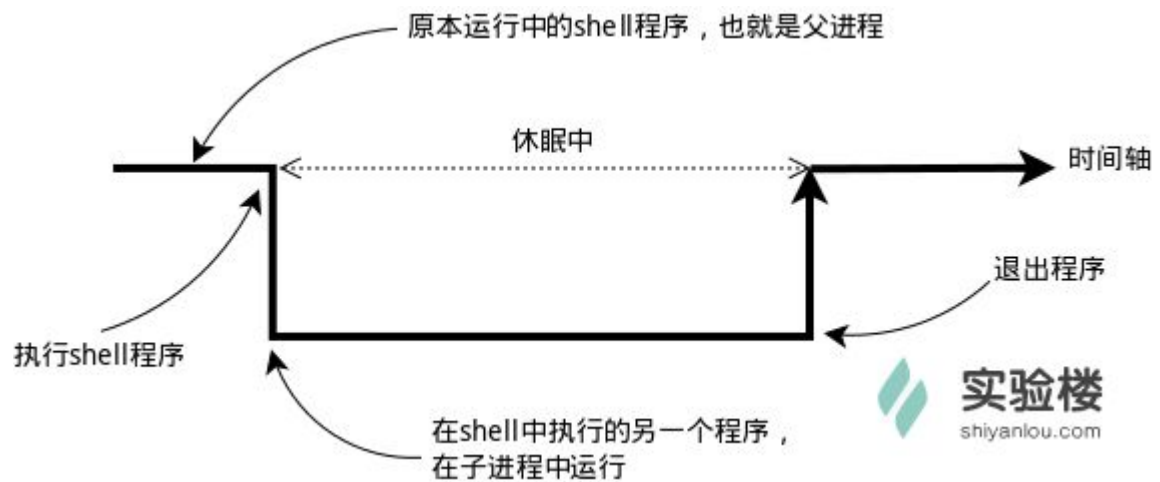


注意:关于变量名，并不是任何形式的变量名都是可用的，变量名只能是英文字母,数字或者下划线，且不能以数字作为开头

## 2.环境变量

简单理解了变量的概念，就很好解释环境变量了，环境变量就是作用域比自定义变量要大，如 shell 的环境变量作用于自身和它的子进程。在所有的 Unix 和类 Unix 系统中，每个进程都有其各自的环境变量设置，且默认情况下，当一个进程被创建时，处理创建过程中明确指定的话，它将继承其父进程的绝大部分环

境设置。shell 程序也作为一个进程运行在操作系统之上，而我们在 shell 中运行的大部分命令都将以 shell 的子进程的方式运行。



通常我们会涉及到的环境变量有三种：

- 当前 shell 进程私有用户自定义变量 ,如上面我们创建的 temp 变量 ,只在当前 shell 中有效
- shell 本身内建的变量
- 从自定义变量导出的环境变量

也有三个与上述三种环境变量相关的命令，set，env，export。这三个命令很相似，都可以用于打印相关环境变量,区别在于涉及的是不同范围的环境变量，详见下表：

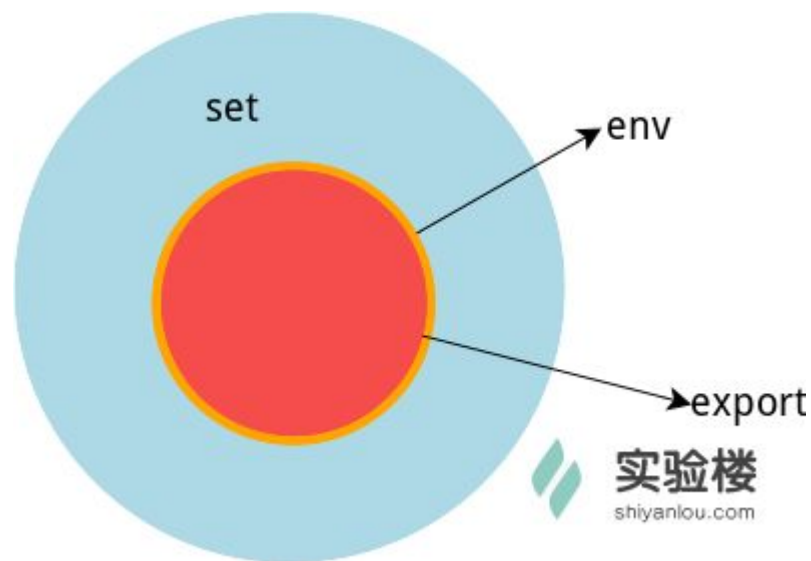
命令	说明
set	显示当前 shell 所有环境变量，包括其内建环境变量（与 shell 外观等相关），用户自定义变量及导出的环境变量

## 命令

## 说明

`env` 显示与当前用户相关的环境变量，还可以让命令在指定环境中运行

`export` 显示从 shell 中导出成环境变量的变量，也能通过它将自定义变量导出为环境变量



你可以更直观的使用 `vimdiff` 工具比较一下它们之间的差别：

```
$ temp=shiyancelou
$ export temp_env=shiyancelou
$ env|sort>env.txt
$ export|sort>export.txt
$ set|sort>set.txt
```

上述操作将命令输出通过管道|使用 `sort` 命令排序，再重定向到对象文本文件中

```
$ vimdiff env.txt export.txt set.txt
```

使用 `vimdiff` 工具比较导出的几个文件的内容

35 SESSION_MANAGER=local	36 SESSION_MANAGER=local	175 SESSION_MANAGER=local/556004ad71e8:@/tmp/.ICE-unix/
		176 _set
		177 setopt localoptions localtraps \${_comp
36 SHELL=/usr/bin/zsh	37 SHELL=/usr/bin/zsh	178 SHELL=/usr/bin/zsh
37 SHLVL=1	38 SHLVL=1	179 SHLVL=1
		180 SHORT_HOST=556004ad71e8
		181 signals=(EXIT HUP INT QUIT ILL TRAP ABRT BUS FPE KI
		182 SPROMPT='zsh: correct '\''%R'\'' to '\''%r'\'' [nya
		183 status=0
38 temp_env=shiyanolou	39 temp_env=shiyanolou	184 temp_env=shiyanolou
		185 temp=shiyanolou
		186 termcap
		187 terminfo
39 TERM=xterm	40 TERM=xterm	188 TERM=xterm
		189 TIMEFMT='%J %U user %S system %P cpu %*E total'
		190 TMPPREFIX=/tmp/zsh
		191 trap - ZERR
		192 TRY_BLOCK_ERROR=-1
		193 TTY=/dev/pts/0
		194 TTYIDLE=-1
		195 UID=1000
		196 userdirs
		197 usergroups
		198 USERNAME=shiyanolou
40 USER=shiyanolou	41 USER=shiyanolou	199 USER=shiyanolou
41 =/usr/bin/env		200 VENDOR=pc
42 VNCDESKTOP=X	42 VNCDESKTOP=X	201 VNCDESKTOP=X
		202 watch=()
		203 WATCH=''
		204 WATCHFMT='%n has %a %l from %m.'
		205 widgets
43 WINDOWID=50331654	43 WINDOWID=50331654	206 WINDOWID=50331654
		207 WORDCHARS=''
44 XDG_CONFIG_DIRS=/etc	44 XDG_CONFIG_DIRS=/etc	208 XDG_CONFIG_DIRS=/etc/xdg
45 XDG_CURRENT_DESKTOP=	45 XDG_CURRENT_DESKTOP=	209 XDG_CURRENT_DESKTOP=XFCE
46 XDG_DATA_DIRS=/usr/l	46 XDG_DATA_DIRS=/usr/l	210 XDG_DATA_DIRS=/usr/local/share:/usr/share
47 XDG_MENU_PREFIX=xfce	47 XDG_MENU_PREFIX=xfce	211 XDG_MENU_PREFIX=xfce-
48 XKL_XMODMAP_DISABLE=	48 XKL_XMODMAP_DISABLE=	212 XKL_XMODMAP_DISABLE=1
NORMAL env.txt <38:1	export.txt << 39:1	set.txt 80% < 184:1

env.txt


export.txt

set.txt



关于环境变量，可以简单的理解成在当前进程的子进程是否有效，有效则为环境变量，否则不是（有些人也将所有变量统称为环境变量，只是以全局环境变量和局部环境变量进行区分，我们只要理解它们的实质区别即可）。我们这里用 export 命令来体会一下，先在 shell 中设置一个变量 temp=shianlou，然后再新建一个子 shell 查看 temp 变量的值：

```
shianlou:~/ $ temp=shianlou
shianlou:~/ $ echo $temp
shianlou
shianlou:~/ $ zsh                                创建子shell,因为我们的环境中使用的是zsh，ubuntu默认shell为bash
shianlou:~/ $ echo $temp
shianlou:~/ $ exit                                值为空，表示该变量无效
shianlou:~/ $ export temp                          退出子shell
shianlou:~/ $ zsh                                导出变量temp为环境变量
shianlou:~/ $ echo $temp                          重新创建子shell
shianlou                                           设为环境变量后有效
```



注意：为了与普通变量区分，通常我们习惯将环境变量名设为大写

### 3.命令的查找路径与顺序

你可能很早之前就有疑问，我们在 shell 中输入一个命令，shell 是怎么知道在哪去找到这个命令然后执行的呢。这是通过环境变量 PATH 来进行搜索的，熟悉 windows 的用户可能知道 windows 中的也是有这么一个 path 环境变量。这个 PATH 里面就保存了 shell 中执行的命令的搜索路径

查看 PATH 环境变量的内容

```
$ echo $PATH
```

默认情况下你会看到如下输出

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

如果你还记得我们在 linux 目录结构那一节的内容,你就应该知道上面这些目录下放的是哪一类文件了。通常这一类目录下 放的都是可执行文件,当我们在 shell 中执行一个命令时,系统就会按照 PATH 中设定的路径按照顺序依次到目录中去查找,如果存在同名的命令,则执行先 找到的那个。下面我们将练习创建一个最简单的可执行 shell 脚本和一个使用 C 语言创建的"hello world"程序,如果这两部分内容你之前没有学习过,那么你可以在这里 [C 语言入门教程](#)和[高级 Bash 脚本编程指南](#)或者 [Linux Shell Scripting Tutorial \(LSST\) v2.0](#) 进行一个入门学习

创建一个 shell 脚本文件：

```
$ vim hello_shell.sh
```

在脚本中添加如下内容,保存并退出(注意不要省掉第一行,这不是注释哈,看到论坛有用户反应会有语法错误,就是因为没有了第一行)

```
#!/bin/zshfor ((i=0; i<10; i++));do  
  
    echo "hello shell"done  
  
exit 0
```

为文件添加可执行权限

```
$ chmod 755 hello_shell.sh
```

创建一个 c 语言"hello world"程序：

```
$ vim hello_world.c
```

```
#include <stdio.h>
```

```
int main(void){
```

```
    printf("hello world!\n");
```

```
    return 0;
```

```
}
```

使用 gcc 生成可执行文件：

```
$ gcc -o hello_world hello_world.c
```

**gcc 生成二进制文件默认具有可执行权限，不需要修改**

在 shiyanlou 家目录创建一个 mybin 目录，并将上述 hello\_shell.sh 和

hello\_world 文件移动到其中

```
$ mkdir mybin$ mv hello_shell.sh hello_world mybin/
```

现在你可以在 mybin 目录中分别运行你刚刚创建的两个程序

```
$ cd mybin$ ./hello_shell.sh$ ./hello_world
```



```
shiyanolou:~/ $ mkdir mybin
shiyanolou:~/ $ mv hello_shell.sh hello_world mybin
shiyanolou:~/ $ cd mybin
shiyanolou:mybin/ $ ./hello_world
hello world!
shiyanolou:mybin/ $ ./hello_shell.sh
hello shell
hello shell
hello shell
hello shell
hello shell
hello shell
hello shell
hello shell
hello shell
hello shell
```

然后你回到上一级目录，也就是 shiyanolou 家目录，你再想运行那两个程序，会发现提示命令找不到，除非你加上命令的完整路径，但是这样不是很麻烦嘛，如何做到想使用系统命令一样执行自己创建的脚本文件或者程序呢。那就要将命令所在路径添加到 PATH 环境变量了。

#### 4.添加自定义路径到"PATH"环境变量

在前面我们应该注意到 PATH 里面的路径是以：作为分割符，所以我们可以这样添加自定义路径

```
$ PATH=$PATH:/home/shiyanolou/mybin
```

**注意这里一定要使用绝对路径**

现在你就可以在其他任意目录执行那两个命令了。你可能会意识到这样还并没有很好的解决问题，因为我给 PATH 环境变量追加了一个路径，它也只是在当 前

shell 有效，我一旦退出终端，再打开就会发现又失效了。有没有方法让添加的环境变量全局有效又或者每次启动 shell 时自动执行上面添加自定义路径 到 PATH 的命令了。我只能说："嗯，年轻人，有想法，我很欣赏你，这样是完全可以的",下面我们就来说说后一种方式——让它自动执行。

在每个用户的家目录中有一个 shell 每次启动时会默认执行一个配置脚本，以初始化环境，包括添加一些用户自定义环境变量等等。zsh 的配置文件是.zshrc，相应 bash 的配置文件为.bashrc。它们在 etc 下还都有一个或多个全局的配置文件，不够我们一般只修改用户目录下的配置文件。

我们可以简单的使用下面命令直接添加内容到.zshrc 中

```
$ echo "PATH=$PATH:/home/shiyanlou/mybin" >> .zshrc
```

**上述命令中**>>表示将标准输出以追加的方式重定向到一个文件中，注意前面用到的>是以覆盖的方式重定向到一个文件中，使用的时候一定要注意分辨。在指定文件不存在的情况下都会创建新的文件

然后重新启动终端生效，现在你就可以随意发挥了

## 5.修改和删除已有变量

### 变量修改

变量的修改有以下几种方式:

变量设置方式	说明
--------	----

变量设置方式	说明
<code>\${变量名#匹配字符串}</code>	从头向后开始匹配，删除符合匹配字符串的最短数据
<code>\${变量名##匹配字符串}</code>	从头向后开始匹配，删除符合匹配字符串的最长数据
<code>\${变量名%匹配字符串}</code>	从尾向前开始匹配，删除符合匹配字符串的最短数据
<code>\${变量名%%匹配字符串}</code>	从尾向前开始匹配，删除符合匹配字符串的最长数据
<code>\${变量名/旧的字串/新的字符串}</code>	将符合旧字符串的第一个字符串替换为新的字符串
<code>\${变量名//旧的字串/新的字符串}</code>	将符合旧字符串的全部字符串替换为新的字符串

比如要修改我们前面添加到 PATH 的环境变量：

为了避免操作失误导致命令找不到，我们先将 PATH 赋值给一个新的自定义变量 path

```
$ path=$PATH
```

```
$ echo $path
```

```
$ path=${path%/home/shiyanlou/mybin}# 或使用通配符,*表示任意多个任意
```

字符

```
$ path=${path%*/mybin}
```

其他设置方式，你就自己操作体会吧。

## 变量删除

可以使用 unset 命令删除一个环境变量

```
$ unset temp
```

## 6.如何让环境变量立即生效

在上面我们在 shell 中修改了一个配置脚本文件之后（比如 zsh 的配置文件的 home 目录下的 .zshrc），每次都要退出终端重新打开甚至重启主机之后其才能生效，很是麻烦，我们可以使用 source 命令来让其立即生效。

如：

```
$ source .zshrc
```

source 命令还有一个别名就是 .，注意与表示当前路径的那个点区分开，虽然形式一样，但作用和使用方式一样，上面的命令如果替换成 . 的方式就应该是

```
$ . ./zshrc
```

注意第一个点后面有一个空格，而且后面的文件必须指定完整的绝对或相对路径名，source 则不需要

## 二、搜索文件

与搜索相关的命令常用的有如下几个 whereis,which,find,locate

whereis **简单快速**

\$whereis who

```
shiyanolou:~/ $ whereis who
who: /usr/bin/who /usr/bin/X11/who /usr/share/man/man1/who.1.gz
```

你会看到它找到了三个路径，两个可执行文件路径和一个 man 在线帮助文件所在路径，是不是很快，它快是因为它并没有从硬盘老老实实挨个去找，而是直接从数据库中查询。whereis 只能搜索二进制文件(-b)，man 帮助文件(-m)和源代码文件(-s)。如果想要获得更全面的搜索结果可以使用 locate 命令

locate **快而全**

通过"/var/lib/mlocate/mlocate.db"数据库查找，不过这个数据库也不是实时更新的，系统会使用定时任务每天自动执行 updatedb 命令更新一次，所以有时候你刚添加的文件，它可能会找不到，你就得自己执行一次 updatedb 命令（在我们的环境中必须先执行一次该命令）。它可以用来查找指定目录下的不同文件类型，如：

查找/etc 下所有以 sh 开头的文件

\$ locate /etc/sh

注意,它不只是在 **etc** 目录下查找并会自动递归子目录进行查找

查找/usr/share/下所有 jpg 文件

```
$ locate /usr/share/*.jpg
```

注意要添加\*号前面的反斜杠转义，否则会无法找到

如果想只统计数目可以加上-c 参数，-i 参数可以忽略大小写进行查找，whereis 的-b,-m，-s 同样可以是使用

which 小而精

which 本身是 shell 内建的一个命令，我们通常使用 which 来确定是否安装了某个指定的软件，因为它只从 PATH 环境变量指定的路径中去搜索命令

```
$ which man
```

find 精而细

find 应该是这几个命令中最强大的了，它不但可以通过文件类型、文件名进行查找而且可以根据文件的属性（如文件的时间戳，文件的权限等）进行搜索。find 命令强大到，要把它弄明白至少需要单独好几门课程才行，我们这里就只介绍一些常用的内容，希望深入学习的用户可以就多找“男人”多动手。

在指定目录下搜索指定文件名的文件

```
$ find /etc/ -name interfaces
```

注意 **find** 命令的路径是作为第一个参数的，基本命令格式为 **find [path] [option] [action]**

与时间相关的命令参数

参数	说明
----	----

-atime	最后访问时间
--------	--------

-ctime	创建时间
--------	------

-mtime	最后修改时间
--------	--------

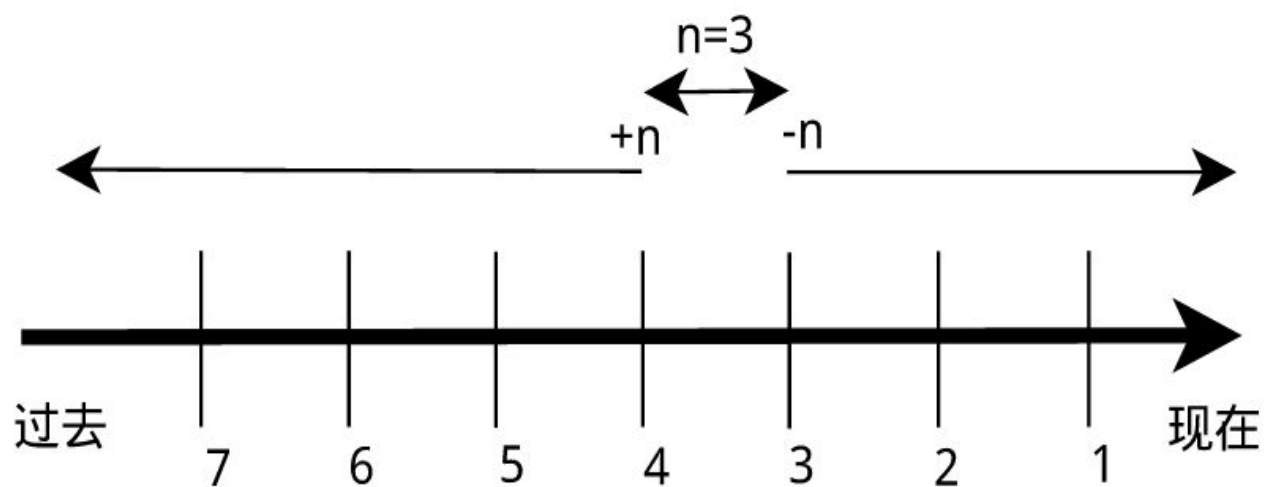
下面以-mtime 参数举例

-mtime n: n 为数字，表示为在 n 天之前的“一天之内”修改过的文件

-mtime +n: 列出在 n 天之前（不包含 n 天本身）被修改过的文件

-mtime -n: 列出在 n 天之前（包含 n 天本身）被修改过的文件

newer file: file 为一个已存在的文件，列出比 file 还要新的文件名



列出 home 目录中，当天（24 小时之内）有改动的文件



```
$ find ~ -mtime 0
```

列出比某个文件新的所有文件

```
# 姑且利用一下工程师配置环境时遗留的 test.c~文件吧-_-|| $ find ~ newer
```

```
Documents/test.c\~
```

### 三、linux 乐趣多

看过《黑客帝国》，觉得里面的矩阵世界的满屏代码的效果炫酷不，在 linux 里面你也可以轻松实现这样的效果，你只需要一个命令 `cmatrix`

同样先安装，不要指望 ubuntu 会将虽如此炫酷但却无用的命令预装在里面

```
$ sudo apt-get update;sudo apt-get install cmatrix
```