Bash简介

SHELL

shell 简介

Shell 是一个命令解释器, 是人与操作系统之间的桥梁。 我们平时无论任何操作,最终都要操作硬件,比如输入一个字符 “ a ”, 那么信号 首先会从键盘传递到主板,通过主板总线传递到内存,CPU,显卡等,最终经过显卡的运 算完成后在屏幕的某个位置, 显示一个特定字体的字符 “ a ”, 这一整个过程可以说是 不断的和硬件打交道了,但是如果让人去发送这些硬件操作码显然不适合, 因为这不是人干 的事,所以我们有了操作系统,操作系统通过加载一定的硬件驱动,从而控制硬件,操作硬件,那剩下的事就是如何和操作系统通信了,对于普通的系统管理员来说,这也是一件非常困难的事,为了方便人和操作系统沟通, 我们开发了 shell 。

Shell 可以将我们平时运行的一些指令解释给操作系统执行, 方便管理员操作系统。而 Shell 的脚本其实是一种命令的堆积,我们将所有需要执行的命令, 以从上至下的方 式写在一个文件当中, 交给 shell 去自动解释执行。

shell 历史

在 AT&T 的 Dennis Ritchie 和 Ken Thompson 设计 UNIXTM 的时候, 他们想要为 用户创建一种与他们的新系统交流的方法。那时的操作系统带有命令解释器。命令解释器接受用户的命令,然后解释它们,因而计算机可以使用这些命令。 但是 Ritchie 和 Thompson 想要的不只是这些功能,他们想提供比当时的命令解释器 具备更优异功能的工具。这导致了 Bourne shell (通称为 sh)的开发, 由 S.R. Bourne 创建。自从 Bourne shell 的创建, 其它 shell 也被一一开发, 如 C shell (csh) 和 Korn shell (ksh) 。

当自由软件基金会想寻求一种免费的 shell , 开发者们开始致力于 Bourne shell 以及当 时其它 shell中某些很受欢迎的功能背后的语言。 这个开发结果是 Bourne Again Shell , 或称 bash 。虽然你的Red Hat Linux 包括几 种不同的 shell , bash 是为互动用户提供的默认 shell 。

常见的 shell

- Bourne shell 即 sh : AT&T 贝尔实验室编写的一个交换式的命令解释器。
- C Shell : Bill Joy 于 20 世纪 80 年代早期开发。为了让用户更容易的使用, 他把语法结构变成了 C语言风格。它新增了命令历史、别名、文件名替换、作业控制等功能。
- korn shell (ksh) 是一个 Unix shell 。它由贝尔实验室的 David Korn 在二十世纪八十 年代早期编写。它完全向上兼容 Bourne shell 并包含了 C shell 的很多特性。
- Bourne-Again SHell : bash 是一个为 GNU 项目编写的 Unix shell 。它的名字是一 系列缩写:Bourne-Again SHell — 这是关于 Bourne shell (sh)的一个双关语(Bourne again / bornagain) 。 Bourne shell 是一个早期的重要 shell , 由 Stephen Bourne 在 1978 年前后编写,并同 Version 7 Unix 一起发布。 bash 则在 1987 年由 Brian Fox 创造。 在 1990 年, ChetRamey 成为了主要的维护者。 bash 是大多数 Linux 系统以及 Mac OS X v10.4 默认的 shell ,它能运行于大多数 Unix 风格的操作系统之上, 甚至被移植到了 Microsoft Windows 上的 Cygwin 和MSYS 系统中, 以实现 windows 的 POSIX 虚拟接口。此外, 它也被 DJGPP 项目移植到了 MS-DOS 上。
- POSIX shell : POSIX shell 与 Korn shell 非常的相似, 当前提供 POSIX shell 的最大卖主是Hewlett-Packard 。

为什么 **Shell**

- 解决重复操作的作业。
- 节约时间 , 提高工作效率。
- 功能强大,使用简单

/etc/shells

`/etc/shells` 文件记录了目前系统中注册过的 shell,每一个用户可以使用一种 shell,可以通过`usermod -s` 来修改用户 shell,也可以通过`chsh` 来更改 shell

```
[root@redhat6 tmp]# chsh carol
Changing shell for carol.
New shell [/bin/csh]: /bin/sh
Shell changed.
[root@redhat6 tmp]# su - carol
-sh-4.1$ tail /etc/passwd -n1
carol:x:60000:60000:carol:/home/carol:/bin/sh
-sh-4.1$
```

bash 功能

history 功能

history 命令用于显示指定数目的指令命令,读取历史命令文件中的目录到历史命令缓冲区和将历史命令缓冲区中的目录写入命令文件。 该命令单独使用时,仅显示历史命令,在命令行中,可以使用符号!执行指定序号的历史命令。

例如,要执行第 2 个历史命令,则输入`!2`。

历史命令是被保存在内存中的,当退出或者登录 shell 时,会自动保存或读取。在内存中,历史命令仅能够存储 1000 条历史命令,该数量是由环境变量 `HISTSIZE` 进行控制。

HISTSIZE 变量 --变量:反复调用的值,以$符号引用

定义方式:histsize 数字 临时生效

/etc/profile 里永久定义

保存位置:~/.bash_history

```
语法 history(选项)(参数)

选项 -c :清空当前历史命令;
-a :将历史命令缓冲区中命令写入历史命令文件中;
-r :将历史命令文件中的命令读入当前历史命令缓冲区;
-w:将当前历史命令缓冲区命令写入历史命令文件中。

参数 打印最近的 n 条历史命令

实例

!8 使用第八个命令
!Ser 使用该字符串开头的最近的命令
!!调用上一条命令
!$ 调用上一个命令的参数 = alt+.
Ctrl+R 搜索最近包含字符串的命令
```

```
history
    命令用法
    history
        -c  清空缓存
        -r      将~/.bash_history历史命令读入缓存

        方便的调用    ! num
                !! 调用最近一次的命令
                !关键字


    实战  1.你是黑客，窃取别人的历史命令
        ~/.bash_history
        2.为了防止黑客，你怎么防止一些关键命令被窃取，比如配置用户密码，或者数据库密码等
    [root@rhel7 ~]# echo "uplooking123"|passwd --stdin superman
    Changing password for user superman.
    passwd: all authentication tokens updated successfully.


        1）清缓冲
        2）删除文件

        [root@rhel7 ~]# export HISTCONTROL=ignorespace
        [root@rhel7 ~]# echo 1
        1
        [root@rhel7 ~]# echo 2
        2
        [root@rhel7 ~]#  echo 3
        3
        [root@rhel7 ~]# history
        1  HISTCONTROL=ignorespace
        2  exprot HISTCONTROL=ignorespace
        3  export HISTCONTROL=ignorespace
        4  echo 1
        5  echo 2
        6  history
```

别名 **alias**

alias 命令用来设置指令的别名。我们可以使用该命令可以将一些较长的命令进行简化。使用 alias 时,用户必须使用单引号" 将原来的命令引起来,防止特殊字符导致错误。 alias 命令的作用只局限于该次登入的操作。若要每次登入都能够使用这些命令别名,则可将相应的 alias 命令存放到 bash 的初始化文件/etc/bashrc 中。

```
语法 alias(选项)(参数)

选项
-p :打印已经设置的命令别名。

参数 命令别名设置:定义命令别名,格式为“ 命令别名=‘ 实际命令 ’” 。

实例 alias 新的命令=' 原命令 -选项/参数'
alias grep=’grep --color=auto’
```

- ~/.bashrc 针对单个用户生效
- /etc/bashrc 针对全局生效

实战

1. 我的工作路径在/tmp/justice/superman/dir1/，每天登陆的服务器都很累，因为要天天输入cd /tmp/justice/superman/dir1/,有什么办法能够轻松一点呢？让我少输入一点命令呢？
2. 取消我刚刚设置的别名unalias

tab 键补全功能

单击补全命令,如果没办法补全,双击可以显示以该字符串开始的文件名都有哪些。(或者命令都有哪些)

快捷键

- Ctrl +A 跳行首
- Ctrl +L 清屏
- Ctrl +e 跳行尾
- Ctrl +c 中断
- Ctrl +R 搜索
- ctrl +d logout

用户登录流程

学习用户登录流程的意义:

我们之前有提到过几个变量,比如 histsize、比如 umask 等。这些功能都可以是针对于某一个用户去设置的。那么我们是怎么知道具体要把这些相关配置信息写入哪一个文件呢?哪一个文件是用于全局的变量的设置的,哪一个文件针对单个用户生效,哪一个文件里面的配置字段会覆盖之前的配置字段呢?这就要用到我们的用户登录流程了。

五个配置文件的顺序:

我们用户在登录的时候,会调用一系列的文件,这些文件包括

- /etc/profile
- /etc/profile.d 目录下的所有文件
- /.bash_profile
- ~/.bashrc
- /etc/bashrc

这是我们系统登录过程中用到的五个文件,我们一一来查看。

/etc/profile

/etc/profile 里面包含了配置字段 include /etc/profile.d 。这个 include 代表扩展配置文件,当读取该文件的时候,会读取相应的一些扩展配置文件。

我们通常不会把所有的东西都往一个文件里面去写,一方面是由于全往一个文件里面去写会导致文件特别大,另一方面,不方便我们的修改文件的灵活度。我们要从一个大篇幅的配置文件中找到某一行,非常的累,那么我们就可以将配置文件也分门别类的去写。比如说我创建了一个新的程序,这个程序需要一些变量的设置,我就可以将这些变量写成一个新的文件,放在/etc/profile.d 目录下,那么当我读取 profile 的时候一样能够读到该文件,当我不需要那个程序了,我也不用去找这个配置字段了,直接把对应的配置文件删除就可以了吧。

~/.bash_profile

接下来回去读~/.bash_profile 文件,由于是家目录下的,这是属于用户个人本身的配置文件,也就是说,写在这个配置文件当中的变量,只针对对应的用户生效。该文件里写的内容一部分是去读取~/.bashrc 文件,一部分写了 path 变量的相关内容。

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
. ~/.bashrc
fi
# User specific environment and startup programs
PATH=$PATH:$HOME/bin
export PATH
```

PATH 变量

PATH 变量是我们系统中的相关命令的路径。PATH 的作用,简化我们的命令写法。

我们执行的命令一般都在 PATH 指定的路径中,比如说我们的 useradd 命令,实际上执行的是/usr/sbin/useradd ,再比如说 touch 命令,实际上执行的是/bin/touch 命令。

```
[root@mastera0 ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@mastera0 ~]# which useradd
/usr/sbin/useradd
[root@mastera0 ~]# which touch
/usr/bin/touch
[root@mastera0 ~]# vim pathtest
[root@mastera0 ~]# ll pathtest
-rw-r--r--. 1 root root 26 Jun 24 16:58 pathtest
[root@mastera0 ~]# chmod +x pathtest
[root@mastera0 ~]# ll pathtest
-rwxr-xr-x. 1 root root 26 Jun 24 16:58 pathtest
[root@mastera0 ~]# pathtest
-bash: pathtest: command not found
[root@mastera0 ~]# ./pathtest
this is a PATH-test!
[root@mastera0 ~]# cp pathtest /usr/local/bin
[root@mastera0 ~]# pathtest
this is a PATH-test!
[root@mastera0 ~]# which pathtest
/usr/local/bin/pathtest
```

那么 PATH 也有一个读取顺序,优先会从第一个位置开始找,比如说我们去/usr/local/sbin 下面创建一个同名的可执行文件 pathtest,内容如下:

```
[root@mastera0 ~]# vim pathtest
[root@mastera0 ~]# cp pathtest /usr/local/sbin
[root@mastera0 ~]# pathtest
this is a PATH-test!
[root@mastera0 ~]# ./pathtest
New! I am /usr/local/sbin this is a PATH-test!
[root@mastera0 ~]# pathtest
this is a PATH-test!
[root@mastera0 ~]# export PATH
[root@mastera0 ~]# pathtest
New! I am /usr/local/sbin this is a PATH-test!
```

这个就是 PATH 的读取顺序,由于第二个 pathtest 命令的路径/usr/local/sbin 优先于第一个

pathtest 命令的路径/usr/local/bin,所以执行了第二个 pathtest即/usr/local/sbin/pathtest。然后我们把第二个 pathtest 删除 `rm -rf /usr/local/sbin/pathtest`,重新 export 一下PATH,export 是设置环境变量的意思。之后我们讲到脚本会去说

```
[root@mastera0 ~]# rm -rf /usr/local/sbin/pathtest
[root@mastera0 ~]# export PATH
[root@mastera0 ~]# pathtest
this is a PATH-test!
```

那么这时候再来看一下执行结果是不是又回来了。

最后,读完这两个~/.bash_profile 和~/.bashrc 以后,系统最后再去读/etc/bashrc。

这是我们整个的流程。先后读取顺序分别是

1. /etc/profile
2. /etc/profile.d/*
3. ~/.bash_profile
4. ~/.bashrc
5. /etc/bashrc

记住,后面写的变量会覆盖之前的变量值,所以我们想让哪些变量生效,就一定要清楚写在什么位置。

比如说,我希望只能够给 student 用户使用的别名应该写到~/.bashrc 目录下

我希望让全局生效的参数可以放置在/etc/profile 目录下。

`Su` 和 `su -`的区别:这就是用户登录流程要注意的事情,系统里面有个很典型的例子,就是 su

`Su` 代表切换用户。用法是 `su - 用户名`,或者`su` 直接跟上用户名。前者叫标准切换,后者叫非标准切换。

非标准切换有些变量是拿不到的。所以你可以看到 su 之后,我们的位置还在切换之前的位置,而我们 `su -` 的位置已经切换到用户的家目录了。

那么是哪些变量没拿到呢?

我们来做个小实验。

演示 给 5 个配置文件的头部分别写上 `echo "xxx (配置文件名) start"`,尾部写上 `echo "xxxx(配置文件名) stop"`

`Man bash` 可以告诉你登录流程的相关内容。

PS1 变量

/etc/bashrc 中有一个变量 PS1,我们一起来看看他的作用。

`man bash` 后搜索 ps1 关键词

```
\d :代表日期,格式为 weekday month date,例如:"Mon Aug 1"
\H :完整的主机名称。例如:我的机器名称为:fc4.linux,则这个名称就是 fc4.linux
\h :仅取主机的第一个名字,如上例,则为 fc4,.linux 则被省略
\t :显示时间为 24 小时格式,如:HH:MM:SS
\T :显示时间为 12 小时格式
\A :显示时间为 24 小时格式:HH:MM
\u :当前用户的账号名称
\v :BASH 的版本信息
\w :完整的工作目录名称。家目录会以 ~代替
\W :利用 basename 取得工作目录名称,所以只会列出最后一个目录
 :下达的第几个命令
\$ :提示字符,如果是 root 时,提示符为:# ,普通用户则为:$
```

the在 PS1 中设置字符序列颜色的格式为:[\e[F;Bm]

- F为字体颜色,编号 30~37;
- B为背景色,编号 40~47 。
- 取消设置:[\e[m]

颜色表

```
前景 背景 颜色
30 40 黑色
31 41 红色
32 42 绿色
33 43 黄色
34 44 蓝色
35 45 紫红色
36 46 青蓝色
37 47 白色

代码
意义
0 OFF
1 高亮显示
4 underline
7 反白显示
8 不可见
```

举例:

PS1='[[\e[32m]###[\e[31m]\u@[\e[36m]\h \w]$[\e[m]'

- [\e[32m] 设置为绿色

- ## 显示现在运行的是第几条命令

- [\e[31m] 设置为红色
- \u@ 当前用户的账号名称@
- [\e[36m] 青蓝色
- \h \w
- \h 仅取主机的第一个名字,\w 是说:显示完整的路径,但是不知到为什么家他显示~而不是绝对路径。
- [\e[m] 使用来关闭颜色设置的。

要是你没有这个的话;那么,你的命令提示符,包括你通过命令提示符输出的东西都是和最后一次的颜色设置相同(除了一些有特殊意义的文件)。

这个配置写到哪里呢???

如果只想让当前用户生效,那么应该在用户的根目录下的 ".bashrc",在里头的最后一行加上,然后保存。然后 source .bashrc 或者 ". .bashrc" 或者注销一下。如果想让所有用户生效,该放哪里?思考题

Bash简介课后作业

1. 要求 student 用户登录的时候获取 umask 值为 044, 并且该 值永久生效。
2. 要求所有普通用户登录时, 最后获取到的 HISTSIZE 值为 500, 而 root 用户获取到的 HISTSIZE 的值为 1000 。
3. 梳理一下五个文件的登录顺序。