Bash 命令行历史权威指南

首先，让我们回顾下一些基本的历史命令快捷键

你可能记得，Bash 提供了两种编辑命令的模式：emacs 与 vi, 很多快捷键在不同的编辑模式下是不同的。

切换模式：

```
$ set -o mode (vi/emacs)
```

假设你要执行如下命令：

```
$ echo foo bar baz
$ iptables -L -n -v -t nat
$ ... lots and lots more commands
$ echo foo foo foo
$ perl -wle 'print q/hello world/'
$ awk -F: '{print$1}' /etc/passwd
$
```

然后你想执行最后一条历史命令(awk -F ...).

你会想当然的按下键盘的上箭头并切安逸地陪伴你一生，但是真的需要把你的手移动哪么远吗？

如果在 Emacs 模式下你只用按下 Ctrl-P 就能够从历史记录中取到前一条命令（Ctrl-n 取下一条命令）

在 vi 模式下你只要按下 Ctrl-[(或 ESC)(用来切换到命令模式)与‘h’('j'取下一条命令)即可

这儿有一个等同有效的方式就是 Bash 的历史扩展机制－－事件引用符， 输入“!!”就可以达到以上等同的效果（详细后面说明）

现在，假设你不想通过再次重复输入就想执行以上第二条命令'iptables -L -n -v -t nat'

天真的用户会照常按下键盘上箭头直到找那所需的那条命令，但这并不是黑客们的玩转方式，黑客们喜欢快速有效的方式。忘掉键盘的上下右左键以及翻页那一块键吧，依我看，它们跟本用不着，而且离键盘主要工作区域太远了。（看来作者应该玩的是 vi）

在 emacs 模式下输入‘iptables’前几个字符，然后使用 Ctrl-r 就行了，比如:'ipt', 这将会正确地得到最后的以'iptables'起始的命令，再次输入 Ctrl-r 会得到更老输入的匹配，假如你错过了正确的结果，你要通过 Ctrl-s 就可以反向的搜索了（不要忘记默认的 Ctrl-s 操作会停止终端的输入，看起来像是假死了，记得按下 Ctrl-g 让终端恢复正常，具体设置，参见‘sttycommand’）。

在 vi 模式下使用 Ctrl-r 与 Ctrl-s 会有些不同。

通过使用‘Ctrl-[’或‘ESC’结合'/'来切换到命令模式，输入‘iptables’前几个字符（插入模式），如'ipt'，然后回车. Bash 将会匹配到最近的一条以'ipt'开始的历史记录，输入‘n’或者通过'/'再次操作会得到相关的上一条命令，使用 'N' or '?' 得到下一条命令。

通过事件引用符可以得到最最近以‘string’开头的历史命令

而使用'!iptables'就能扩展到最近的以‘iptables’开头的历史命令

另一种方式是使用 Bash 自身的‘history’命令与 grep 结合，然后再使用事件引用符‘!N’, N 指的是历史记录中的第几条命令。

例如：

```
$ history | grep 'ipt'
 2    iptables -L -n -v -t nat
$ !2     # will execute the iptables command
```

在 vi 编辑模式下，使用‘N’（命令行号）与‘G’， 就可以达到相同的效果，此例使用'2G'即可

列出与擦除历史命令

Bash 内建了历史命令('history')的查看与擦除

及以下例为例：

```
$ echo foo bar baz
$ iptables -L -n -v -t nat
$ ... lots and lots more commands
$ echo foo foo foo
$ perl -wle 'print q/hello world/'
$ awk -F: '{print$1}' /etc/passwd
$
```

输入‘history’将会得到所有的带行号的历史命令

```
 1    echo foo bar baz
 2    iptables -L -n -v -t nat
 ...  lots and lots more commands
 568  echo foo foo foo
 569  perl -wle 'print q/hello world/'
 570  awk -F: '{print$1}' /etc/passwd
```

输入‘history N’(N 为整数)，会得到最近的 N 条历史命令，如:'history 3', 结果如下：

```
 568  echo foo foo foo
 569  perl -wle 'print q/hello world/'
 570  awk -F: '{print$1}' /etc/passwd
```

‘history -c’将清除所以的历史记录，'history -d N'将会删除 N 条历史记录

默认在用户的主目录下会有一个名为'.bash_history'的历史记录文件

历史命令扩展

历史命令扩展的完善就是依赖所谓的事件引用符与词组引用符，事件引用符适用于先前取得的命令（事件），词组引用符适用于由事件而来的命令行。我们可以选择的就是：多样的修饰符可应用于已提取的参数。

事件引用符都是的以感叹号开头的特殊命令（也有以“^“开头的），它们可跟一个词组引用符和一个或多个修饰符。引用符与修饰符都是由':' 分隔开的

事件引用符

让我们看看下面的一组事件引用符的例子是如何运作的。

事件引用符“!!”用来取得前一条命令，例如：

```
$ echo foo bar baz
foo bar baz
$ !!
foo bar baz
```

这里的“!!”就执行了'echo foo bar baz'这条命令

事件引用符'!N'用于执行命令历史的第 N 条记录

假如你的历史记录输出如下：

```
 1    echo foo foo foo
 2    iptables -L -n -v -t nat
 ...  lots and lots more commands
 568  echo bar bar bar
 569  perl -wle 'print q/hello world/'
 570  awk -F: '{print$1}' /etc/passwd!
```

“!569”就会执行'perl ...' 这条命令，而'!1'就会执行'echo foo foo foo'

'!-N'就会执行当前的命令的计数前 N 条命令（中间的‘-’可以理解为减号），如下：

```
$ echo foo bar baz
foo bar baz
$ echo a b c d e
a b c d e
$ !-2
foo bar baz
```

‘!-2’执行上上一条命令，或者说是当前的命令的前 2 条命令

'!string'适用以'string'开头的历史命令。例如:

```
$ awk --help
$ perl --help
```

‘!p‘或者'!perl'或者'!per'都可以正确是执行到'perl -help'， 同理，'!a'亦可以执行那条以 awk 开头的命令

而'!?string?'就能匹配到包含'string'的命令，并不严格要求开头。

最有意思的事件引用符莫过于'^string1^string^'了，它将替换最后一条命令中的'string1'字符窜为'string2'，例如：

```
$ ehco foo bar baz
bash: ehco: command not found
$ ^ehco^echo^
foo bar baz
```

上面的‘^ehco^echo^’作用就是取得上一条历史记录并替换 ehco 为 echo，然后再执行

词组引用符与修饰符

事件引用符后面词组引用符（一个或多个）是用冒句分隔开的，它们适用于当前参照命令的部分或者所有的参数

例如：

```
$ echo a b c d e
a b c d e
$ echo !!:2
b
```

这里有一个最简单的词组引用符，':2'对应第 2 个参数（第 3 个单词），通常情况下':N'对应命令的第 N-1 个单词

词组引用符也接受一个范围值，例如：

```
$ echo a b c d e
a b c d e
$ echo !!:3-4
c d
```

符号用法也是多样的，例如，”:$”对应最后一个参数，”:^”对应第一个参数，”:*”对应所有的参数(等同':1-$'), 等等，详情看完整的参照表

修饰符可以改变单词引用符的行为，例如：

```
$ tar -xvzf software-1.0.tgz
software-1.0/file
...
$ cd !!:$:r
software-1.0$
```

'r'在这里就匹配了上一条命令的最后一个参数，'r'在此用作去掉后缀'.tgz'

‘h’用于匹配并删除文件路径中的文件名部分，返回文件路径的开头部分：

```
$ echo /usr/local/apache
/usr/local/apache
$ echo !!:$:h
/usr/local
```

'e'则只保留扩展名

```
$ ls -la /usr/src/software-4.2.messy-Extension
...
$ echo /usr/src/*!!:$:e
/usr/src/*.messy-Extension    # ls could have been used instead of echo
```

另一个有趣的修饰符是替换符':s/old/new/', 即替换单词'old'为'new'，'g'用于全局替换，例如：

```
$ ls /urs/local/software-4.2 /urs/local/software-4.3
/usr/bin/ls: /urs/local/software-4.2: No such file or directory
/usr/bin/ls: /urs/local/software-4.3: No such file or directory
$ !!:gs/urs/usr/
...
```

以上这个例子就是替换所有的”urs”为'usr'来达到我们想要的结果。

这里还有一些其它的修饰符，如'p'只是用来回显扩展过的命令，并不执行。详情看完整的参照表。

配置命令历史

Bash 允许用户自己配置命令历史文件，文件保存着命令以及记录号和一些选项

变量 HISTFILE, HISTFILESIZE, HISTIGNORE 和 HISTSIZEenvironment 对命令历史起着决定的作用。

HISTFILE, 正如词面意思，代表命令历史文件的路径

```
$ export HISTFILE=/home/pkrumins/todays_history
```

上面命令当保存命令历史文件路径为'/home/pkrumins/todays_history'

值设置为'/dev/null'或空将不保存历史记录

HISTFILESIZE 控制着历史文件最大记录数，例如：

```
$ export HISTFILESIZE=1000
```

这将保存 1000 条最近的历史命令

HISTSIZE 控制着当前会话（终端）的历史记录条数

```
$ export HISTSIZE=42
```

上面将保存当前会话最近 42 条历史命令

如果 HISTSIZE 变量的值小于 HISTFILESIZE 变量的值，只有限定数量的命令会被记录下来

HISTIGNORE 用来控制我们不想记录的命令，此变量是用冒号来分隔不同的模式，'&' 对匹配历史命令起来特殊的作用。

‘[ ]*’这个技巧可以用来忽略以空格开头的命令。

例如：

```
$ export HISTIGNORE="&:[ ]*:exit"
```

上面的命令就控制 bash 忽略复制命令，以及以空格开头，以及内容为‘exit’字符的命令

这儿有几个有关‘shopt’命令的选项。

'-s'参数让'shopt'命令的配置有效，而'-u'参数使其无效。

‘histappend’控制着历史列表如何写到命令历史记录中，设置这个选项将追加当前会话的历史命令到命令历史记录中，不设置（默认不设置，我的 ubuntu 10.04 默认却是打开的）每次将会重写命令历史记录文件。

打开：

```
$ shopt -s histappend
```

关闭：

```
$ shopt -u histappend
```

选项'histreedit'允许用户重编辑一条有误的历史替换命令

试想你已经输入如下：

```
$ echo foo bar baz
```

本意想通过'^baz^test^'替换'baz'为'test', 但是却输成了'^boo^test^', 这将因为不匹配而导致替换失败

如果将此选项打开，bash 将会保持你当前的'^boo^test^'错误的输入

最后，选项'histverify'允许用户去验证一条替换历史扩展命令

就拿前面的例子来说吧，假设你想通过'!!'再次'echo'命令，如果这个选项打开，bash 将不会立即执行这条'echo'命令，而是将命令显示好等待你去验证是否是你想要的结果。

调整命令提示

我的命令提示是这样的：

```
Wed Jan 30@07:07:03
pkrumins@catonmat:1002:2:~$
```

第一行显示日期与时间可以及时的追溯命令

第二行显示用户名，主机名，全局行号以及当前命令当号

全局行号可以让我更快捷地使用事件引用符

我的 PS1 参数，变量值如下：

```
PS1='\d@\t\n\u@\h:\!:\#:\w$ '
```

Bash 历史技巧表

在此下载（英文）

原文：http://www.catonmat.net/blog/the-definitive-guide-to-bash-command-line-history/

首次翻译技术文章，如有不明白之处，敬请谅解，欢迎各位来我的博客，提出您的意见
逍遥云的博客： http://hi.baidu.com/pallove