Shell History

The shell keeps a record of the commands you have previously executed. Bash keeps its history in memory for the current session and in the `~/.bash_history` file so that it can be recalled during future sessions. Other shells may use `~/.history`, `~/.zsh_history`, or other similarly named files. Having access to your shell history is extremely useful because it allows you to quickly repeat commands. This can save you time, save keystrokes, prevent you from making mistakes by running a previously known good command, and generally speed up your work flow.

`history` - Display a list of commands in the shell history.

`!N` - Repeat command line number `N`.

`!!` - Repeat the previous command line.

`!string` - Repeat the most recent command starting with "string."

```
$ history
1 ls
2 diff secret secret.bak
3 history
$ !1
ls
PerformanceReviews tpsreports
$ echo $SHELL
/bin/bash
$ !!
echo $SHELL
/bin/bash
$ !d
diff secret secret.bak
3c3
< pass: Abee!
---
> pass: bee
$
```

With the exclamation mark history expansion syntax you can rerun a command by number. In the above example the first command in the history was executed with `!1`. If you want to execute the second command you would execute `!2`. Another convenient shortcut is `!-N` which means execute the Nth previous command. If you want to execute the second to last command type `!-2`. Since `!!` repeats the most recent command, it is the same as `!-1`.

```
$ history
1 ls
2 diff secret secret.bak
3 history
$ !-2
diff secret secret.bak
3c3
< pass: Abee!
---
> pass: bee
$
```

By default bash retains 500 commands in your shell history. This is controlled by the `HISTSIZE` environment variable. If you want to increase this number add `export HISTSIZE=1000` or something similar to your personal initialization files.

`Ctrl-r` - Reverse search. Search for commands in your shell history.

You can search for commands in your history. For example, if you have the command `find /var/tmp -type f` in your shell history you could find it by typing `Ctrl-r fi Enter`. `Ctrl-r` initiates the reverse search and displays the search prompt, `fi` is the search string, and `Enter` executes the command that was found. You do not have to search for the start of the string. You could have very well searched for "var", "tmp", or "type."

```
$ find /var/tmp -type f
/var/tmp/file.txt
(reverse-i-search)`fi': find /var/tmp -type f
/var/tmp/file.txt
```

Tab Completion

Another way to increase your efficiency at the shell is by using tab completion. After you start typing a command you can hit the `Tab` key to invoke tab completion. Tab attempts to automatically complete partially typed commands. If there are multiple commands that begin with the string that precedes `Tab`, those commands will be displayed. You can continue to type and press `Tab` again. When there is only one possibility remaining, pressing the `Tab` key will complete the command.

`Tab` - Autocompletes commands and filenames.

```
$ # Typing who[Tab][Tab] results in:
$ who
who     whoami
$ # Typing whoa[Tab][Enter] results in:
$ whoami
bob
$
```

Tab completion not only works on commands, but it also works on files and directories. If you have files that start with a common prefix, `Tab` will expand the common component. For example, if you have two files named `file1.txt` and `file2.txt`, typing `cat f Tab` will expand the command line to `cat file`. You can then continue typing or press `Tab` twice to list the possible expansions. Typing `cat f Tab 2 Tab` will expand to `cat file2.txt`. After you experiment with tab completion it will soon become second nature.

```
$ # Typing cat f[Tab] results in:
$ cat file
$ # Typing: cat f[Tab][Tab][Tab] results in:
$ cat file
file1.txt  file2.txt
$ # Typing cat f[Tab] 2[Tab][Enter] results in:
$ cat file2.txt
This is file2!!!
$
```

Shell Command Line Editing

From time to time you will want to change something on your current command line. Maybe you noticed a spelling mistake at the front of the line or need to add an additional option to the current command. You may also find yourself wanting to recall a command from your shell history and modify it slightly to fit the current situation. Command line editing makes these types of activities possible.

Shells such as bash, ksh, tcsh, and zsh provide two command line editing modes. They are emacs, which is typically the default mode, and vi. Depending on the shell you can change editing modes by using the `set` or `bindkey` command. If you want to ensure your preferred mode is set upon login, add one of the two commands to your personal initialization files.

```
Shell  Emacs Mode    Vi Mode     Default mode
----   ------------  ----------  ------------
bash   set -o emacs  set -o vi   emacs
ksh    set -o emacs  set -o vi   none
tcsh   bindkey -e    bindkey -v  emacs
zsh    bindkey -e    bindkey -v  emacs
zsh    set -o emacs  set -o vi   emacs
```

Emacs Mode

As you would expect, in emacs command line editing mode you can use the key bindings found in the emacs editor. For example, to move to the beginning of the command line type `Ctrl-a`. To recall the previous command type `Ctrl-p`.

`Esc Esc` - Escape completion. Similar to tab completion.

`Ctrl-b` - Move cursor to the left (back)

`Ctrl-f` - Move cursor to the right (forward)

`Ctrl-p` - Up (Previous command line)

`Ctrl-n` - Down (Next command line)

`Ctrl-e` - Move to the end of the line

`Ctrl-a` - Move to the beginning of the line

`Ctrl-x Ctrl-e` - Edit the current command line in the editor defined by the $EDITOR environment variable.

See the section in this book on the emacs editor for more key bindings.

Vi Mode

When you are using vi command line editing mode you start in insert mode so you can quickly type commands. To enter command mode, press `Esc`. To move to the previous command, for example, type `Esc k`. To resume editing enter insert mode by pressing `i`, `I`, `a`, or `A`.

`Esc` - Enter command mode.

Key bindings in command mode:

`\` - Vi style file completion. Similar to tab completion.

`h` - Move cursor left

`k` - Up (Previous command line)

`j` - Down (Next command line)

`l` - Move cursor right

`$` - Move to the end of the line

`^` - Move to the beginning of the line

`i` - Enter insert mode.

`a` - Enter insert mode, append text at current location.

`A` - Enter insert mode, append text at end of line.

`I` - Enter insert edit mode, prepend text to start of line.

`v` - Edit the current command line in the editor defined by the $EDITOR environment variable.

See the section in this book on the vi editor for more key bindings.

Dealing with Long Shell Commands

The backslash (`\`) is the line continuation character. You have learned how to use the backslash to escape special characters like spaces. However, when a backslash is placed at the end of a line it is used as a line continuation character. This allows you to create command lines that are displayed as multiple lines but are executed as a single command line by the shell. You can use line continuation to make commands more readable and easier to understand.

```
$ echo "one two three"
one two three
$ echo "one \
> two \
> three"
one two three
$ echo "onetwothree"
onetwothree
$ echo "one\
> two\
> three"
onetwothree
$
```

Notice the greater-than symbol (`>`) in the above example. It is the secondary prompt string and can be customized by changing the PS2 environment variable. You learned previously how to change the primary prompt string with PS1 in the customizing the prompt section of this book.

```
$ PS2="line continued: "
$ echo "one \
line continued: two \
line continued: three"
one two three
$
```