

# The Dreaded Command Line

## Easier and More Powerful than Expected

Geoff Richards

<gbdirect>

<http://www.gbdirect.co.uk/>

---

Monday, 10th March 2003  
Presented for WYLUG

# 1. The Unix and Linux Command Line

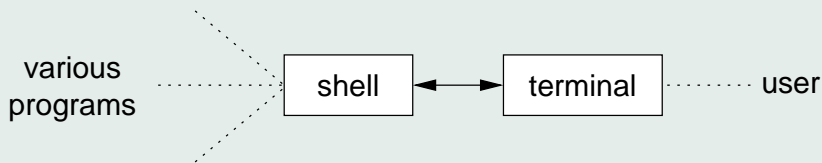
- Typed commands were first used for necessity
  - ◆ Only interface to the computer was a simple and slow terminal
- But the command line is still a powerful and efficient interface for many types of work

## 2. Pros and Cons

- The command line still has its place:
  - ◆ Typing commands can be quicker than getting to the same functionality with a mouse
  - ◆ Commands can be very powerful, in ways GUIs rarely are
  - ◆ Newer shells have many features to make commands easier to type
- But graphical interfaces are also useful:
  - ◆ The command line makes no sense for some things, like editing graphics
  - ◆ Easier to get into – you can see what options are open to you
- Both command line and graphical interfaces have a place in modern Unix and Linux

# 3. What's Involved

- The command line relies on three things:
  - ◆ The **terminal**, which displays the output of commands and echos what you type
  - ◆ The **shell**, which reads commands and works out what to do about them
  - ◆ The commands themselves – Unix and Linux have a vast array of small programs which can be run from a shell



## 4. The Terminal

- Lets you 'get at' the shell to work with it
- Usually a program which opens a window in a graphical environment, like xterm, konsole, gnome-terminal, . . .
- A program displays text on the terminal, and receives key-presses from it
  - ◆ Communication done with a simple protocol, using **escape sequences** for special things
- The shell you are using might be connected over a network
  - ◆ Things like rsh and (much better) ssh make this work
  - ◆ So the command line is just as powerful when used on a remote machine

# 5. The Shell

- The shell is your interface to the machine
- On Linux the most commonly used shell is **Bash**
  - ◆ The 'Bourne again shell' – a reincarnation of the original Unix shell by Steve Bourne
  - ◆ Compatible with the original, but has many new features
- The shell does three things every time you enter a command:
  - ◆ Reads lines of text from the user
  - ◆ Interprets commands and works out what to do with them
  - ◆ For most commands, runs one or more programs to do the work

## 6. Some Examples of Commands

- `$` represents the prompt – don't type it

- Ask a philosophical question:

```
$ whoami  
geoffr
```

- Copy a file:

```
$ cp report.txt report-backup.txt
```

- View a PostScript file:

```
$ gv unix_command_line.ps
```

- Output a message:

```
$ echo hello, world  
hello, world
```

## 7. Filename Completion

- Commands are often used to manipulate or examine files
  - ◆ So there's a lot of tedious filenames to type...
- The shell can do a lot of the typing for you:
  - ◆ Type the start of a filename
  - ◆ Type TAB
  - ◆ The shell will type the rest of the name, if there's only one file whose name starts like that

- For example, if the only file starting with *rep* is called *report.txt*:

```
$ less rep<TAB>
```

- If there are several files which Bash thinks you might mean, press TAB again to get a list of possible completions

## 8. Other Completion Goodies

- Completion also works for commands
  - ◆ Type the start of a program name and see what it completes to
  - ◆ Handy for finding programs when you were only guessing they existed
- As the shell completes filenames, it escapes special characters automatically
  - ◆ Makes it much easier to type filenames containing spaces and such
- After a `$` symbol, the shell completes variable names

## 9. History

- Another handy feature of modern shells is **history**
- The shell keeps a record of each command you type
- Pressing Up and Down keys scrolls through the history, retrieving each command
  - ◆ Emacs users may prefer to use Ctrl+P and Ctrl+N
- After going back to a command it can be edited and rerun
- It is possible to search back through the history
  - ◆ Type Ctrl+R, and then any part of the original command line
- The shell can save the history between logins

# 10. Globbing

- The shell can collect filenames for a command to use
  - ◆ Write a **glob** pattern, and the shell will find file's whose names match it
- For example, to delete all files in the current directory:

```
$ rm *
```

- To count words in all the files whose names end in `.txt`:

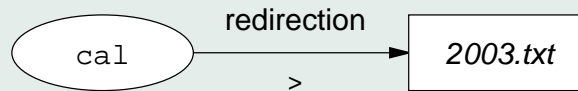
```
$ wc *.txt
```

- \* means “anything is allowed here”
- Other symbols can be used too:
  - ◆ ? means “any single character is allowed here”
  - ◆ [a-z] means “any lowercase letter can go here”

# 11. Redirection

- A command can be 'connected' to input and output files
  - ◆ The shell does this, setting it up before running the appropriate program
  - ◆ Done with the < and > symbols
- For example, to write a copy of this year's calendar to a file:

```
$ cal 2003 >2003.txt
```



- < is the same, but reads from the file

# 12. Piping

- Similar to redirection – but connects a program to another program
- Use the `|` symbol – often called the ‘pipe character’
- For example, pipe the output of `echo` into the program `rev` (which reverses each line of its input):

```
$ echo Happy Birthday | rev  
yadhtriB yppaH
```

- The output of the `echo` command is fed into the input of `rev`:



# 13. Shell Scripting

- Put a load of command lines into a file, and it's called a **shell script**
  - ◆ Same idea as a Windows 'batch file', but much more powerful
- Shell scripts can use looping, variables and other programming features
  - ◆ But these can also be used interactively
  - ◆ Useful for running the same command on many files, or creating a whole set of users in one go, or . . .
- If you end up typing the same sequence of commands over and over again, turn them into a shell script
  - ◆ But if you want to write a 'full size' program, learn Perl

## 14. So...

- Unix and Linux shells can be complicated and tricky, and are certainly feared by the uninitiated
  - ◆ But those who are practiced at them find them powerful and efficient
- Not as much typing is involved as you might think
  - ◆ Completion saves your fingers
  - ◆ History lets you go back and try again
- The shell is not for everyone, but well worth learning for anyone serious about Linux

<gbdirect>

[www.gbdirect.co.uk](http://www.gbdirect.co.uk)