Introduction to bioinformatics

Basic Unix commands

A CRITICAL GUIDE

Version: 1 August 2018

Basic Unix commands

Overview

This Critical Guide briefly introduces the Unix Operating System, and provides a subset of some of the most helpful and commonly used commands, including those that allow various types of search, navigation and file manipulation. Several keystroke short-cuts are also explained, which help to make the routine use of Unix commands more efficient.

Teaching Goals & Learning Outcomes

This Guide showcases some of the simplest, most frequently used commands to help new users to understand and gain confidence in using the Unix Operating System. On reading the Guide, you'll be able to use a range of commands to:

- manipulate files, directories and processes;
- navigate directory structures and explore their contents;
- search for files, and search and compare file contents;
- direct command outputs into files or into other commands; and
- explain what many simple commands mean and how they're used.

1 Introduction

For many users, a computer's **Operating System** (OS) is a black box that conceals the technical wizardry controlling its **hardware** and **software**. The most ubiquitous OS is Microsoft Windows. This gained widespread popularity because its **Graphical User Interface** (GUI) shielded users from the difficulty of having to 'talk' to computers through the obscure **command-line** interfaces characteristic of early computer systems. Probably the most popular of these was the Unix OS: modular by design, this OS had a degree of portability that facilitated its propagation to many platforms.

In the early days of bioinformatics, many of the newly developed software tools didn't have simple, easy-to-use GUIs; unfortunately, many still don't! In consequence, users are obliged to interact with them directly through the command line. Furthermore, as the life sciences are becoming increasingly data driven, more and more researchers need to be able to write simple scripts in order to manage their data. For many, this means having to become familiar with Unix. Although this may sound like a daunting task, it's actually possible to get a long way with just a few basic commands.

This Guide isn't intended to be a comprehensive introduction to Unix (there are many excellent books for this purpose); rather, it offers a quick start for new users, to outline some of the most helpful Unix commands, to gain familiarity with various types of file manipulation and especially with navigation between files within **directory** structures.

2 About this Guide

Throughout the text, key terms – rendered in **bold** type – are defined in boxes. Dummy file- or directory names are *italicised*, while valid Unix commands are both bold and ***italicised***.

Part of the barrier for new users is that Unix commands can appear rather opaque, as they take the form of shorthand 'contractions' or acronyms that describe or stand for the commands they represent (*e.g.*, ***ls*** for list, ***mv*** for move, ***rm*** for remove, ***cp*** for copy, and so on). Generally, a command's default behaviour or output may be modified by adding different 'qualifiers'. Qualifiers are preceded by a **-** (minus) sign, and may be used individually or concatenated in groups (*e.g.*, ***-t*** or ***-Ftr***).

The following sections list many frequently used commands and some of the qualifiers that modify their behaviour, explaining what they do and giving examples of their use. Additional information is provided in supplementary boxes. Exercises are provided to give opportunities to practice the use of some of the most basic commands, and to help you understand how they behave and how to modify their outputs.

KEY TERMS

Command line: the means of interacting with a computer system via text commands (command lines) typed directly at a keyboard

Directory: a file that catalogues sets of files in a computer system; also referred to as 'folders', directories are denoted by the / sign

Executable file: a file that performs operations on a computer; the executable instructions are encoded, so are machine- not human-readable; executable files are usually denoted by the * symbol

Graphical User Interface (GUI): software that facilitates users' interactions with a computer system via easy-to-use graphical icons

Hardware: the physical components (the machines, wiring, *etc.*) of computer systems

Operating System: the software that controls a computer's hardware & software resources, & provides its program processes

Software: the programs, procedures & algorithms that instruct computers what tasks to perform & how to perform them

Symbolic link: a file that points to another file; symbolic links are usually denoted by the @ sign

3 The commands

The following is a subset of available Unix commands, and a limited set of examples of their use. Much more information can be found in the general command manual, which can be accessed via the command line:

man displays the manual entry for a specified command, one screen at a time

man ls
displays information on the **ls** command:
<spacebar> scrolls through the manual file one page at a time
b displays the previous page
j (or the *<enter>* key) moves to the next line
k moves to the previous line
g returns to the first line
G moves to the last line
q exits the file-pager at any point

3.1 Listing files and viewing file contents

ls displays the contents of the current directory, listed in (case-sensitive) alphabetical order

ls -F
gives further information, explicitly identifying which are files, which are directories, **symbolic links**, **executable files**, *etc.*

ls -l
gives more lengthy information, including details of file permissions, ownership, size, time the file was last modified, *etc.*

ls -t
lists the directory contents in the time order in which the files were modified, starting from the most recent

ls -r
lists the directory contents in reverse order

ls -a
lists all the contents of the current directory, including those hidden by the Unix OS

ls *.txt
lists all text files in the current directory

ls diffdir/
lists all files in a different directory, called *diffdir*

Additional commands, symbols & short-cuts

***** : asterisk is the 'wild-card' symbol, which can be used as a short-cut for 'all file names' or 'all file types', or part of a file name or file type

<up>: the <up> key scrolls up through all previous commands (which is helpful for editing any previous command in-line)

<down>: the <down> key scrolls back down through recalled commands

!! : two shrieks recall & execute the previous command (note: any previous command can be recalled & executed – *e.g.*, **!15** recalls & executes command 15)

history: lists all the previous commands in the current session (so, any previous command may be recalled & executed)

<tab>: the <tab> key auto-completes (so saves fully typing) file names

. : full-stop is a short-cut denoting the current directory

cat reads a text file and displays the full contents of the file on-screen; for multiple files, the contents are concatenated

cat myfile.txt
displays the content of the file *myfile.txt* onscreen

cat myfile.txt myotherfile.txt
displays the contents of both *myfile.txt* and of *myotherfile.txt* onscreen

more reads a text file and displays its content one screen at a time. The pager short-cuts shown for **man** can help navigate the file

more myfile.txt
displays the content of *myfile.txt* one screen at a time

less text-file reader, similar to **more**, but allowing both forward and backward navigation. See **man** for pager short-cuts

less myfile.txt
displays the content of *myfile.txt* one screen at a time

head reads a text file and displays its first 10 lines onscreen

head myfile.txt
displays the first 10 lines of *myfile.txt*

head -25 myfile.txt
displays the first 25 lines of *myfile.txt*

tail reads a text file and displays its last 10 lines onscreen

tail myfile.txt
displays the last 10 lines of *myfile.txt*

tail -5 myfile.txt
displays the last 5 lines of *myfile.txt*

EXERCISES

1 List the files in your home directory in reverse time order, showing full information about the file permissions, ownership, *etc.*, & showing explicitly which are files, directories, *etc.*

2 From your home directory, list the files on your Desktop.

3.2 Copying, moving and removing files

cp copies files and directories, and allows them to be re-named

cp myfile.txt myfilecopy.txt
copies *myfile.txt* to a new file called *myfilecopy.txt*

cp myfile.txt diffdir/myfilecopy.txt
copies *myfile.txt* to the new file, *myfilecopy.txt*, and places it in a different directory, *diffdir*

cp myfile.txt diffdir/
copies *myfile.txt* to the *diffdir* directory without changing its name

cp diffdir/myfile.txt .
copies *myfile.txt* from the *diffdir* directory to the current directory (**.**) without changing its name

cp diffdir/myfile.txt ./myfilecopy.txt
copies *myfile.txt* from *diffdir* to the current directory (**.**), changing its name to *myfilecopy.txt*

mv moves files and directories to different locations in the system's directory structure, and also allows them to be re-named (**mv** is a short-cut, equivalent to **cp** followed by **rm**)

mv myfile.txt diffdir/
moves *myfile.txt* to a different directory, *diffdir*

mv diffdir/myfile.txt .
moves *myfile.txt* from the *diffdir* directory to the current directory

mv myfile.txt myfilecopy.txt
re-names *myfile.txt* to *myfilecopy.txt* within the current directory

mv myfile.txt diffdir/myfilecopy.txt
moves *myfile.txt* to the *diffdir* directory and re-names it to *myfilecopy.txt*

mv diffdir/myfilecopy.txt ./myfile.txt
moves *myfilecopy.txt* from the *diffdir* directory to the current directory and re-names it to *myfile.txt*

rm removes or deletes files

rm *myfile.txt*
removes *myfile.txt* from the current directory

rm *diffdir/myfile.txt*
removes *myfile.txt* from a different directory, *diffdir*

rm **.txt*
removes all text files from the current directory

rm -i **.txt*
gives an interactive dialogue when removing all text files

rm *diffdir/*.txt*
removes all text files from the *diffdir* directory

EXERCISES

1 In your home directory, identify a file that isn't a directory. Copy it to a new file name (*e.g.*, *mynewfile*). Copy *mynewfile* to another new file (*e.g.*, *anothernewfile*) on your Desktop. List the contents of your home & Desktop directories to ensure the files exist.

2 Move *mynewfile* to your Desktop. List the contents of your Desktop to ensure that the move was successful.

3 Delete *mynewfile* & *anothernewfile* from your Desktop. List the contents of your Desktop directory to check that the removal was successful.

3.3 Searching for and comparing file contents

grep searches for a specified string of text within a text file

grep *gremlin myfile.txt*
searches for occurrences of the word gremlin in *myfile.txt* and displays the lines on which the word occurs

grep -n *gremlin myfile.txt*
searches for the word gremlin within *myfile.txt* and displays the lines and line numbers on which the word occurs

grep -i *gremlin myfile.txt*
performs a case-insensitive search for the word gremlin within *myfile.txt*

grep *gremlin *.txt*
searches for gremlin in all text files in the current directory

diff compares contents of text files and displays the differences

diff *myfile.txt mynewfile.txt*
displays differences between the contents of *myfile.txt* and *mynewfile.txt* within the current directory

diff *myfile.txt diffdir/mynewfile.txt*
displays content differences between *myfile.txt* and *mynewfile.txt* from a different directory, *diffdir*

diff -b *myfile.txt mynewfile.txt*
displays content differences between *myfile.txt* and *mynewfile.txt*, ignoring whitespace between words

diff -B *myfile.txt mynewfile.txt*
displays differences between the contents of *myfile.txt* and *mynewfile.txt*, ignoring blank lines

wc displays onscreen the number of lines, words and characters contained within a specified text file

wc *myfile.txt*
displays the number of lines, words and characters contained within *myfile.txt*

wc -l *myfile.txt*
displays the number of lines contained in *myfile.txt*

wc -w *myfile.txt*
displays the number of words contained in *myfile.txt*

wc -c *myfile.txt*
displays the number of characters within *myfile.txt*

3.4 Searching for files

find searches the directory structure for a specified file

find . -name *myfile.txt*
searches, from the current directory, all sub-directories for instances of *myfile.txt* (use of **/** rather than **.** searches the *entire* directory structure, but this isn't recommended)

find *diffdir* **-name** *myfile.txt*
searches the *diffdir* directory for *myfile.txt*

find . -name *"*.txt"*
searches, from the current directory, all sub-directories for all text files (note the use of quote marks)

EXERCISES

1 From your home directory, find a text file in your directory structure.

2 If the file isn't in your home directory, note its location & the path to its parent directory.

3 From your home directory, search for any word within the file.

4 From your home directory, display the word count of this file.

3.5 Directing command outputs

> re-directs the output of a command from the screen and saves it to a specified file

ls **.txt* **>** *mytextfiles.txt*
lists all the text files in the current directory and saves the list to a new text file, *mytextfiles.txt*

cat *myfile.txt myotherfile.txt* **>** *bothfiles.txt*
concatenates the contents of *myfile.txt* and *myotherfile.txt* to a new file, *bothfiles.txt*

grep *gremlin myfile.txt* **>** *gremlins.txt*
searches for occurrences of the word gremlin in *myfile.txt*, saving the result to a new file, *gremlins.txt*

find . -name *myfile.txt* **>** *findmyfile.txt*
searches, from the current directory, all sub-directories for instances of *myfile.txt* and saves the result to a new file, *findmyfile.txt*

>> appends the contents of a specified file to another file

cat *anotherfile.txt* **>>** *bothfiles.txt*

| sends ('pipes') the output of one command into another command, and displays the result onscreen

grep *gremlin myfile.txt* **|** *wc*
searches for occurrences of the word gremlin within *myfile.txt*, pipes the output into the ***wc*** command, which displays the number of lines, words and characters it contains

EXERCISES

1 Recall your previous *grep* command via the up-arrow key. Re-direct the output to a new file in your home directory (*wordfile.txt*).

2 Recall your previous *grep* command. Pipe the output to ***wc***. On how many lines in the specified text file did your word occur?

3 Recall your previous *grep* command. Edit the command line (use the *<left>* and *<right>* arrows) to search for a different word, & direct the output to another new file (*e.g.*, *wordfile2.txt*)

4 Combine the contents of *wordfile.txt* & *wordfile2.txt* into a new file, *wordfile3.txt*. Word-count *wordfile3.txt* using *wc*. How many lines are there in the combined file?

5 Delete *wordfile*.txt* from your home directory.

3.6 Changing file permissions

chmod changes the 'mode' or access permissions of specified files and directories, determining who is allowed to read (**r**), write (**w**) and execute (**x**) them, whether the owner (**u**), a group of users (**g**), or others (**o**) who are neither in the group nor the file's owner. Various modifiers are used to specify how to change the file permissions: *e.g.*, '**+**' adds a specified permission, '**–**' removes a permission, and '**=**' equalises permissions between specified users

chmod go-w *myfile.txt*
removes 'write permissions' from groups and others, preventing them from altering *myfile.txt*

chmod g+rwx *myfile.txt*
adds 'read, write and execute permissions' for the group, allowing them to read, alter and execute *myfile.txt*

chmod ug=rw *myfile.txt*
sets the permissions for both the owner and the group to read and alter *myfile.txt*

How file permissions are displayed

The beginning of this Guide introduced the 'list' (**ls**) command. One of its qualifiers (**-l**) was seen to provide more lengthy information about the files in a directory, including details of its access permissions, presented in the form: **rwxrwxrwx**. The **rwx** components designate the 'read', 'write' and 'execute' permissions respectively for the owner (**u**), the group (**g**) and other users (**o**).

#	Permission	rwx
7	read, write, execute	rwx
6	read and write	rw-
5	read and execute	r-x
4	read only	r--
3	write and execute	-wx
2	write only	-w-
1	execute only	--x
0	none	---

The table shows all permutations of file permissions, from full 'read, write and execute' to 'none'; numerical short-cuts for each state are also shown. With this notation, it becomes evident that **rwxrwxrwx** gives read, write and execute permissions to the owner, group and other users, while **rwxr-xr-x** is more limiting, removing write permission from the group and from other users.

Where '**d**' appears before the **rwxrwxrwx** notation, this denotes the permission status of a directory rather than an ordinary file.

EXERCISES

1 What notation shows a file owner has read, write & execute permissions, while group & other users have read-only permission?
2 Write the same notation for a directory.
3 What notation shows a file owner has read, write & execute permissions, while the group has read & execute permissions, but other users have execute-only permission.
4 Write the numerical notation equivalent to **rwxr-xr-x**

Up to this point, the Guide has been dealing with files and, while it's introduced the concept of 'directories' (also commonly known as folders), it hasn't yet listed commands that allow navigation around a directory hierarchy. Figure 1 illustrates part of a typical hierarchical structure of computer file systems, starting with the 'Home' directory and its familiar directories and files, drilling down from the Desktop to its various sub-directories, and thence to their files and sub-directories.

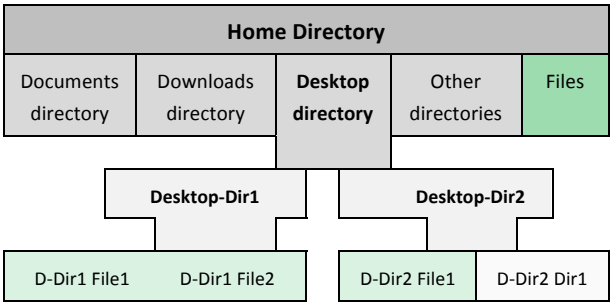

Figure 1 The hierarchical nature of a typical directory structure. *The Home directory includes various files & familiar directories (Documents, Downloads, Desktop, etc.). Moving 'down a level', we see that the Desktop directory contains two more directories (Desktop-Dir1, Desktop-Dir2). Of these, Directory1 contains two files, & Directory2 contains a single file & an additional directory (which may itself contain further files & directories).*

The section that follows takes a closer look at a range of commands that allow navigation through such directory hierarchies, and allow creation and removal of directories.

3.7 Directories

cd changes the current directory, allowing navigation through the directory structure; used without a qualifier, **cd** returns to the home directory from anywhere in the directory structure

cd *diffdir*
moves from the current directory into a directory (or folder) it contains, *diffdir* (this can be thought of as moving 'down a level' in the directory hierarchy – see Figure 1)

cd *diffdir/anotherdiffdir*
moves from the current directory into a directory, *anotherdiffdir*, contained within *diffdir* (this is effectively moving 'down two levels' in the directory hierarchy)

cd ..
moves from the current directory back to its parent directory (*i.e.*, moves 'up a level' in the directory hierarchy)

cd ../..
moves from the current to the parent of its parent directory (*i.e.*, moves 'up two levels' in the directory hierarchy)

cd *../diffdir2*
moves from the current to the sibling directory, *diffdir2*, in the parent directory (*i.e.*, 'one level up' and 'one level down' – this is like moving between folders on a Desktop)

cd *../../diffdir3*
moves from the current directory to the *diffdir3* directory in the parent directory (*i.e.*, 'two levels up', 'one level down')

pwd prints the current working directory, showing your current location

mkdir makes a new directory within the current directory

mkdir *diffdir4*
makes the new directory, *diffdir4*, in the current directory

rmdir removes a specified directory from the current directory. Before a directory can be deleted, all its files must be removed, including those hidden by the Unix OS (**ls -la** will reveal whether any of these exist, which can then be deleted)

rmdir *diffdir4*
removes the *diffdir4* directory from the current directory

EXERCISES

1 Draw a directory structure that's compatible with the fictional directory structure used in the `cd` examples here.
2 In your home directory, make a new directory, `newdir` & copy a text file into it. Now try to remove `newdir`.
3 Delete the text file from `newdir`. Now try to remove the directory.

3.8 Processes

`<ctrl>c` kills a current process or job

`<ctrl>z` suspends a current process or job, which can then either be moved to the background or resumed in the foreground, using the relevant background and foreground commands

`bg` moves the current process or job to run in the background

`fg` resumes the current process or job in the foreground

`fg 5`
moves process number 5 in the jobs table to the foreground

`jobs` lists all background jobs in a job table, together with their job number and job state (*i.e.*, whether suspended or running)

`jobs -l`
lists all background jobs and includes their process IDs (PID)

`ps` displays a header line, beneath which is listed information about all current processes and jobs, including their PIDs

`ps -e`
displays information about other users' processes

`ps -f`
displays the user ID (UID), the PID, the process start time, elapsed CPU time and the associated command

`kill` kills a specified process or job

`kill 5342`
kills the process with PID 5342

Miscellaneous commands

`date` displays the current date and time

`exit` leaves the current shell – it is equivalent to logging out of the current window (or `<ctrl>d`)

`passwd` invokes a program that allows you to change your password (if you invoke the program and decide not to change your password, exit using `<ctrl>d`)

`tar` creates files into or extracts files from an archive file

`tar -cvf tarfile.tar *.txt`
creates the archive file, `tarfile.tar`, listing all the text files in the current directory

`tar -xvf tarfile.tar`
extracts a list of all the files archived in `tarfile.tar`

`who` reveals which users are logged into the computer system

4 References & further reading

1 **List of Unix commands available from Wikipedia: en.wikipedia.org/wiki/List_of_Unix_commands**
2 **The Open Group UNIX V7 Product Standard: https://publications.opengroup.org/x1201**
3 Siever E *et al.* (2009) **Linux in a Nutshell.** O-Reilly Media Inc. ISBN: 978-0-596-15448-6.

TAKE HOMES

1 The Unix OS has a command-line interface;
2 Unix commands are rendered as short acronyms;
3 The behaviour, & hence output, of Unix commands may be modified using a variety of qualifiers;
4 A set of simple commands allows files & directories to be manipulated (viewing & analysing their contents; copying, moving & removing them; changing their ownership & access permissions; *etc.*);
5 A set of commands can be used to suspend or terminate processes &/or to run them either as foreground or background jobs;
6 Other commands allow navigation 'up' & 'down' a computer system's directory (folder) hierarchy;
7 There are commands to search for specific or generic file names or file types; other commands allow searches for particular text strings (words/phrases) within files, or allow file contents to be compared;
8 Some commands allow command outputs to be re-directed from the screen into files or into other commands;
9 A range of keyboard 'short-cuts' can be used to save much laborious typing: of these, '.' (denoting the home directory), '..' (allowing navigation up directory structures), '*' (denoting any text), '!' (allowing recall of previous commands), & <tab> (for auto-completing file & directory names) are amongst the most useful.

5 Acknowledgements & funding

GOBLET Critical Guides marry ideas from the Higher Apprenticeship specification for college-level students in England (www.contentextra.com/lifesciences/unit12/unit12home.aspx) with the EMBnet Quick Guide concept. The original Unix Quick Guide was written by Aoife McLysaght and Andrew Lloyd, and subsequently modified by Laurent Falquet for distribution by EMBnet in 2003: www.embnet.org/shared/quickguides/18-guideUNIX.pdf.

This Guide was developed with the support of a donation from EMBnet to the GOBLET Foundation.

Design concepts and the Guide's front-cover image were contributed by CREACTIVE.

6 Licensing & availability

The Guide is freely available for download via the GOBLET portal (www.mygoblet.org) and EMBnet website (www.embnet.org).

7 Disclaimer

About the organisations

GOBLET

GOBLET (Global Organisation for Bioinformatics Learning, Education & Training) was established in 2012 to unite, inspire and equip bioinformatics trainers worldwide; its mission, to cultivate the global bioinformatics trainer community, set standards and provide high-quality resources to support learning, education and training.

GOBLET's ethos embraces:

- ***inclusivity**: welcoming all relevant organisations & people*
- ***sharing**: expertise, best practices, materials, resources*
- ***openness**: using Creative Commons Licences*
- ***innovation**: welcoming imaginative ideas & approaches*
- ***tolerance**: transcending national, political, cultural, social & disciplinary boundaries*

Further information about GOBLET and its Training Portal can be found at **www.mygoblet.org** and in the following references:

- Attwood *et al.* (2015) **GOBLET: the Global Organisation for Bioinformatics Learning, Education & Training.** *PLoS Comput. Biol.*, **11**(5), e1004281.
- Corpas *et al.* (2014) **The GOBLET training portal: a global repository of bioinformatics training materials, courses & trainers.** *Bioinformatics*, **31**(1), 140-142.

GOBLET is a not-for-profit foundation, legally registered in the Netherlands: CMBI Radboud University, Nijmegen Medical Centre, Geert Grooteplein 26-28, 6581 GB Nijmegen. For general enquiries, contact **info@mygoblet.org**.

EMBnet

EMBnet, the Global Bioinformatics Network, is a not-for-profit organisation, founded in 1988 as a network of institutions, to establish and maintain bioinformatics services across Europe. As the network grew, its reach expanded beyond European borders, creating an international membership to support and deliver bioinformatics services across the life sciences: **www.embnet.org**.

Since its establishment, a focus of EMBnet's work has been bioinformatics Education and Training (E&T), and the network therefore has a long track record in delivering tutorials and courses worldwide. Perceiving a need to unite and galvanise international E&T activities, EMBnet was one of the principal founders of GOBLET. For more information and general enquiries, contact **info@embnet.org**.

CREACTIVE

CREACTIVE, by Antonio Santovito, specialises in communication and Web marketing, helping its customers to create and manage their online presence: **www.gocreactive.com**.

About the author

Teresa K Attwood (orcid.org/0000-0003-2409-4235)

Teresa (Terri) Attwood is a Professor of Bioinformatics with more than 25 years' experience teaching introductory bioinformatics, in undergraduate and postgraduate degree programmes, and in *ad hoc* courses, workshops and summer schools, in the UK and abroad.

With primary expertise in protein sequence analysis, she created the PRINTS protein family database and co-founded InterPro (her particular interest is in the analysis of G protein-coupled receptors). She has also been involved in the development of software tools for protein sequence analysis, and for improving links between research data and the scientific literature (most notably, Utopia Documents).

She wrote the first introductory bioinformatics text-book; her third book was published in 2016:

- Attwood TK & Parry-Smith DJ. (1999) ***Introduction to Bioinformatics.*** Prentice Hall.
- Higgs P & Attwood TK. (2005) ***Bioinformatics & Molecular Evolution.*** Wiley-Blackwell.
- Attwood TK, Pettifer SR & Thorne D. (2016) ***Bioinformatics challenges at the interface of biology and computer science: Mind the Gap.*** Wiley-Blackwell.

Affiliation

School of Computer Science, The University of Manchester, Oxford Road, Manchester M13 9PL (UK).