

Learning to program with F#

Jon Spurring

Department of Computer Science,
University of Copenhagen

August 30, 2018

Contents

1. Preface	6
2. Introduction	7
2.1. Learning how to solve problems by programming	7
2.2. How to solve problems	9
2.3. Approaches to programming	10
2.4. Why use F#	11
2.5. How to read this book	12
3. Executing F# code	13
3.1. Source code	13
3.2. Executing programs	14
4. Quick-start guide	18
5. Using F# as a calculator	26
5.1. Literals and basic types	26
5.2. Operators on basic types	33
5.3. Boolean arithmetic	36
5.4. Integer arithmetic	37
5.5. Floating point arithmetic	41
5.6. Char and string arithmetic	43
5.7. Programming intermezzo: Hand conversion between decimal and binary numbers	45
6. Values and functions	51
6.1. Value bindings	56
6.2. Function bindings	62
6.3. Operators	73
6.4. Do bindings	74
6.5. The Printf function	75
6.6. Reading from the console	79
6.7. Variables	81
6.8. Reference cells	84
6.9. Tuples	89
7. In-code documentation	95
8. Controlling program flow	103
8.1. While and for loops	103
8.2. Conditional expressions	110
8.3. Programming intermezzo: Automatic conversion of decimal to binary numbers	113

9. Organising code in libraries and application programs	116
9.1. Modules	116
9.2. Namespaces	121
9.3. Compiled libraries	124
10. Testing programs	129
10.1. White-box testing	133
10.2. Black-box testing	138
10.3. Debugging by tracing	142
11. Collections of data	153
11.1. Strings	153
11.1.1. String properties	154
11.1.2. String module	154
11.2. Lists	157
11.2.1. List properties	162
11.2.2. List module	162
11.3. Arrays	168
11.3.1. Array properties and methods	171
11.3.2. Array module	172
11.4. Multidimensional arrays	179
11.4.1. Array2D module	183
12. The imperative programming paradigm	187
12.1. Imperative design	188
13. Recursion	190
13.1. Recursive functions	190
13.2. The call stack and tail recursion	193
13.3. Mutual recursive functions	197
14. Programming with types	203
14.1. Type abbreviations	203
14.2. Enumerations	204
14.3. Discriminated Unions	205
14.4. Records	209
14.5. Structures	213
14.6. Variable types	215
15. Pattern matching	219
15.1. Wildcard pattern	223
15.2. Constant and literal patterns	224
15.3. Variable patterns	226
15.4. Guards	227
15.5. List patterns	228
15.6. Array, record, and discriminated union patterns	229
15.7. Disjunctive and conjunctive patterns	232
15.8. Active Pattern	234
15.9. Static and dynamic type pattern	238
16. Higher order functions	241
16.1. Function composition	244
16.2. Currying	245

17. The functional programming paradigm	247
17.1. Functional design	249
18. Handling Errors and Exceptions	251
18.1. Exceptions	251
18.2. Option types	265
18.3. Programming intermezzo: Sequential division of floats	267
19. Working with files	271
19.1. Command line arguments	272
19.2. Interacting with the console	274
19.3. Storing and retrieving data from a file	277
19.4. Working with files and directories.	284
19.5. Reading from the internet	284
19.6. Resource Management	287
19.7. Programming intermezzo: Ask user for existing file	289
20. Classes and objects	291
20.1. Constructors and members	292
20.2. Accessors	295
20.3. Objects are reference types	299
20.4. Static classes	301
20.5. Recursive members and classes	303
20.6. Function and operator overloading	304
20.7. Additional constructors	307
20.8. Interfacing with <code>printf</code> family	310
20.9. Programming intermezzo	311
21. Derived classes	317
21.1. Inheritance	317
21.2. Abstract class	322
21.3. Interfaces	325
21.4. Programming intermezzo: Chess	327
22. The object-oriented programming paradigm	343
22.1. Identification of objects, behaviors, and interactions by nouns-and-verbs . .	345
22.2. Class diagrams in the Unified Modelling Language	345
22.3. Programming intermezzo: designing a racing game	350
23. Graphical User Interfaces	356
23.1. Opening a window	357
23.2. Drawing geometric primitives	359
23.3. Programming intermezzo: Hilbert Curve	371
23.4. Handling events	378
23.5. Labels, buttons, and pop-up windows	382
23.6. Organising controls	387
24. The Event-driven programming paradigm	396
25. Where to go from here	397
A. The Console in Windows, MacOS X, and Linux	400
A.1. The Basics	400
A.2. Windows	400

Contents

A.3. MacOS X and Linux	404
B. Number systems on the computer	412
B.1. Binary numbers	412
B.2. IEEE 754 floating point standard	413
C. Commonly used character sets	417
C.1. ASCII	418
C.2. ISO/IEC 8859	419
C.3. Unicode	419
D. Common Language Infrastructure	424
E. Language Details	426
E.1. Arithmetic operators on basic types	426
E.2. Basic arithmetic functions	429
E.3. Precedence and associativity	431
Bibliography	433
Index	434

A | The Console in Windows, MacOS X, and Linux

Almost all popular operating systems are accessed through a user-friendly *graphical user interface (GUI)* that is designed to make typical tasks easy to learn to solve. As a computer programmer, you often need to access some of the functionalities of the computer, which, unfortunately, are sometimes complicated by this particular graphical user interface. The *console*, also called the *terminal* and the *Windows command line*, is the right hand of a programmer. The console is a simple program that allows you to complete text commands. Almost all the tasks that can be done with the graphical user interface can be done in the console and vice versa. Using the console, you will benefit from its direct control of the programs we write, and in your education, you will benefit from the fast and raw information you get through the console.

- graphical user interface
- GUI
- console
- terminal
- Windows command line

A.1. The Basics

When you open a *directory* or *folder* in your preferred operating system, the directory will have a location in the file system, whether from the console or through the operating system's graphical user interface. The console will almost always be associated with a particular directory or folder in the file system, and it is said that it is the directory that the console is in. The exact structure of file systems varies between Linux, MacOS X, and Windows, but common is that it is a hierarchical structure. This is illustrated in Figure A.1.

- directory
- folder

There are many predefined console commands, available in the console, and you can also make your own. In the following sections, we will review the most important commands in the three different operating systems. These are summarized in Table A.1.

A.2. Windows

In this section we will discuss the commands summarized in Table A.1. Windows 7 and earlier versions: To open the console, press **Start->Run** in the lower left corner, and then type **cmd** in the box. In Windows 8 and 10, you right-click on the windows icon, choose **Run** or equivalent in your local language, and type **cmd**. Alternatively, you can type **Windows-key + R**. Now you should open a console window with a prompt showing something like Listing A.1.

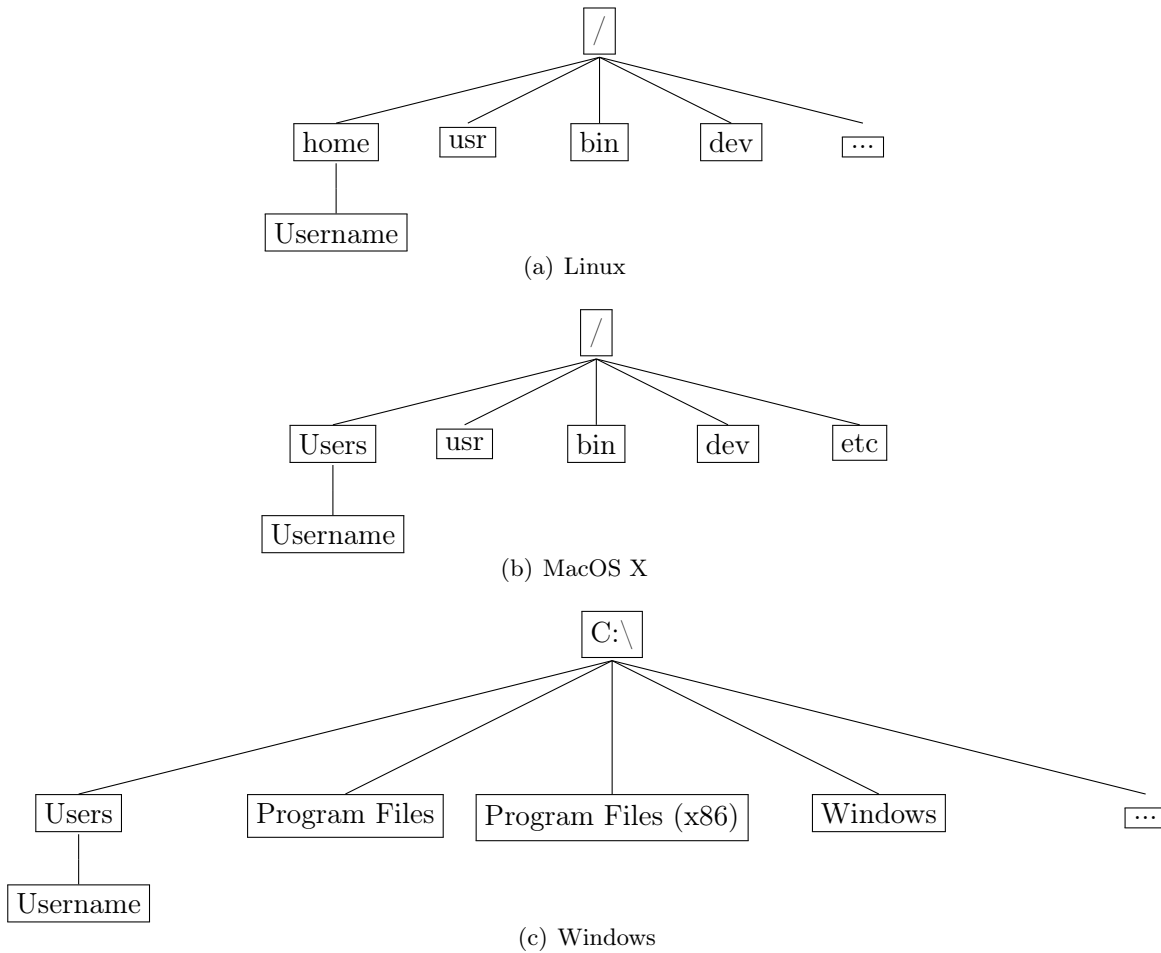


Figure A.1.: The top file hierarchy levels of common operating systems.

Windows	MacOS X/Linux	Description
<code>dir</code>	<code>ls</code>	Show content of present directory.
<code>cd <dir></code>	<code>cd <dir></code>	Change present directory to <code><dir></code> .
<code>mkdir <dir></code>	<code>mkdir <dir></code>	Create directory <code><dir></code> .
<code>rmdir <dir></code>	<code>rmdir <dir></code>	Delete <code><dir></code> (Warning: cannot be reverted).
<code>move <file> <file or dir></code>	<code>mv <file> <file or dir></code>	Move <code><fil></code> to <code><file or dir></code> .
<code>copy <file1> <file2></code>	<code>cp <file1> <file2></code>	Create a new file called <code><file2></code> as a copy of <code><file1></code> .
<code>del <file></code>	<code>rm <file></code>	delete <code><file></code> (Warning: cannot be reverted).
<code>echo <string or variable></code>	<code>echo <string or variable></code>	Write a string or content of a variable to screen.

Table A.1.: The most important console commands for Windows, MacOS X, and Linux.

Listing A.1: The Windows console.

```

1 Microsoft Windows [Version 6.1.7601]
2 Copyright (c) 2009 Microsoft Corporation. All rights reserved.
3
4 C:\Users\sporrington>

```

To see which files are in the directory, use *dir*, as shown in Listing A.2.

· *dir*

Listing A.2: Directory listing with dir.

```

1 C:\Users\sporrington>dir
2 Volume in drive C has no label.
3 Volume Serial Number is 94F0-31BD
4
5 Directory of C:\Users\sporrington
6
7 30-07-2015 15:23 <DIR> .
8 30-07-2015 15:23 <DIR> ..
9 30-07-2015 14:27 <DIR> Contacts
10 30-07-2015 14:27 <DIR> Desktop
11 30-07-2015 17:40 <DIR> Documents
12 30-07-2015 15:11 <DIR> Downloads
13 30-07-2015 14:28 <DIR> Favorites
14 30-07-2015 14:27 <DIR> Links
15 30-07-2015 14:27 <DIR> Music
16 30-07-2015 14:27 <DIR> Pictures
17 30-07-2015 14:27 <DIR> Saved Games
18 30-07-2015 17:27 <DIR> Searches
19 30-07-2015 14:27 <DIR> Videos
20          0 File(s)          0 bytes
21        13 Dir(s) 95.004.622.848 bytes free
22
23 C:\Users\sporrington>

```

We see that there are no files and thirteen directories (DIR). The columns tell from left to right: the date and time of their creation, the file size or if it is a folder, and the name file or directory name. The first two folders “.” and “..” are found in each folder and refer to this folder as well as the one above in the hierarchy. In this case, the folder “.” is an alias for C:\Users\sporrington and “..” for C:\Users.

Use *cd* to change directory, e.g., to Documents, as in Listing A.3.

· *cd*

Listing A.3: Change directory with cd.

```

1 C:\Users\sporrington>cd Documents
2
3 C:\Users\sporrington\Documents>

```

Note that some systems translate default filenames, so their names may be given different names in different languages in the graphical user interface as compared to the console.

You can use *mkdir* to create a new directory called, e.g., myFolder, as illustrated in List-

· *mkdir*

ing A.4.

Listing A.4: Creating a directory with mkdir.

```

1 C:\Users\sporrington\Documents>mkdir myFolder
2
3 C:\Users\sporrington\Documents>dir
4 Volume in drive C has no label.
5 Volume Serial Number is 94F0-31BD
6
7 Directory of C:\Users\sporrington\Documents
8
9 30-07-2015  19:17    <DIR>          .
10 30-07-2015  19:17    <DIR>          ..
11 30-07-2015  19:17    <DIR>          myFolder
12                0 File(s)                0 bytes
13                3 Dir(s)  94.656.638.976 bytes free
14
15 C:\Users\sporrington\Documents>

```

By using `dir` we inspect the result.

Files can be created by, e.g., *echo* and *redirection*, as demonstrated in Listing A.5.

· `echo`
· *redirection*

Listing A.5: Creating a file with echo and redirection.

```

1 C:\Users\sporrington\Documents>echo "Hi" > hi.txt
2
3 C:\Users\sporrington\Documents>dir
4 Volume in drive C has no label.
5 Volume Serial Number is 94F0-31BD
6
7 Directory of C:\Users\sporrington\Documents
8
9 30-07-2015  19:18    <DIR>          .
10 30-07-2015  19:18    <DIR>          ..
11 30-07-2015  19:17    <DIR>          myFolder
12 30-07-2015  19:18                8 hi.txt
13                1 File(s)                8 bytes
14                3 Dir(s)  94.656.634.880 bytes free
15
16 C:\Users\sporrington\Documents>

```

To move the file `hi.txt` to the directory `myFolder`, use *move*, as shown in Listing A.6.

· *move*

Listing A.6: Move a file with move.

```

1 C:\Users\sporrington\Documents>move hi.txt myFolder
2 1 file(s) moved.
3
4 C:\Users\sporrington\Documents>

```

Finally, use *del* to delete a file and *rmdir* to delete a directory, as shown in Listing A.7.

· `del`
· *rmdir*

Listing A.7: Delete files and directories with `del` and `rmdir`.

```

1 C:\Users\sporrington\Documents>cd myFolder
2
3 C:\Users\sporrington\Documents\myFolder>del hi.txt
4
5 C:\Users\sporrington\Documents\myFolder>cd ..
6
7 C:\Users\sporrington\Documents>rmdir myFolder
8
9 C:\Users\sporrington\Documents>dir
10 Volume in drive C has no label.
11 Volume Serial Number is 94F0-31BD
12
13 Directory of C:\Users\sporrington\Documents
14
15 30-07-2015  19:20    <DIR>          .
16 30-07-2015  19:20    <DIR>          ..
17                0 File(s)                0 bytes
18                2 Dir(s)  94.651.142.144 bytes free
19
20 C:\Users\sporrington\Documents>

```

The commands available from the console must be in its *search path*. The search path can be seen using `echo`, as shown in Listing A.8.

Listing A.8: Displaying the search path.

```

1 C:\Users\sporrington\Documents>echo %Path%
2 C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;
   C:\Windows\System32\WindowsPowerShell\v1.0\;"\Program
   Files\emacs-24.5\bin\"
3
4 C:\Users\sporrington\Documents>

```

The path can be changed using the Control panel in the graphical user interface. In Windows 7, choose the Control panel, choose **System and Security** → **System** → **Advanced system settings** → **Environment Variables**. In Windows 10, you can find this window by searching for “Environment” in the Control panel. In the window’s **System variables** box, double-click on **Path** and add or remove a path from the list. The search path is a list of paths separated by “;”. Beware, Windows uses the search path for many different tasks, so remove only paths that you are certain are not used for anything.

A useful feature of the console is that you can use the **tab**-key to cycle through filenames. E.g., if you write `cd` followed by a space and **tab** a couple of times, then the console will suggest to you the available directories.

A.3. MacOS X and Linux

MacOS X (OSX) and Linux are very similar, and both have the option of using *bash* as console. It is in the standard console on MacOS X and on many Linux distributions. A summary of the most important *bash* commands is shown in Table A.1. In MacOS X,

you find the console by opening **Finder** and navigating to **Applications** → **Utilities** → **Terminal**. In Linux, the console can be started by typing **Ctrl + Alt + T**. Some Linux distributions have other key-combinations such as **Super + T**.

Once opened, the console is shown in a window with content, as shown in Listing A.9.

Listing A.9: The MacOS console.

```
1 Last login: Thu Jul 30 11:52:07 on ttys000
2 FN11194:~ sporring$
```

“FN11194” is the name of the computer, the character `~` is used as an alias for the user’s home directory, and “sporring” is the username for the user presently logged onto the system. Use `ls` to see which files are present, as shown in Listing A.10.

· `ls`

Listing A.10: Display a directory content with `ls`.

```
1 FN11194:~ sporring$ ls
2 Applications      Documents      Library        Music          Public
3 Desktop           Downloads      Movies         Pictures
4 FN11194:~ sporring$
```

More details about the files are available by using flags to `ls` as demonstrated in Listing A.11.

Listing A.11: Display extra information about files using flags to `ls`.

```
1 FN11194:~ sporring$ ls -l
2 drwx----- 6 sporring staff 204 Jul 30 14:07 Applications
3 drwx-----+ 32 sporring staff 1088 Jul 30 14:34 Desktop
4 drwx-----+ 76 sporring staff 2584 Jul 2 15:53 Documents
5 drwx-----+ 4 sporring staff 136 Jul 30 14:35 Downloads
6 drwx-----@ 63 sporring staff 2142 Jul 30 14:07 Library
7 drwx-----+ 3 sporring staff 102 Jun 29 21:48 Movies
8 drwx-----+ 4 sporring staff 136 Jul 4 17:40 Music
9 drwx-----+ 3 sporring staff 102 Jun 29 21:48 Pictures
10 drwxr-xr-x+ 5 sporring staff 170 Jun 29 21:48 Public
11 FN11194:~ sporring$
```

The flag `-l` means long, and many other flags can be found by querying the built-in manual with `man ls`. The output is divided into columns, where the left column shows a number of codes: “d” stands for directory, and the set of three of optional “rwx” denote whether respectively the owner, the associated group of users, and anyone can respectively “r” - read, “w” - write, and “x” - execute the file. In all directories but the **Public** directory, only the owner can do any of the three. For directories, “x” means permission to enter. The second column can often be ignored, but shows how many links there are to the file or directory. Then follows the username of the owner, which in this case is **sporring**. The files are also associated with a group of users, and in this case, they all are associated with the group called **staff**. Then follows the file or directory size, the date of last change, and the file or directory name. There are always two hidden directories: “.” and “..”, where “.” is an alias for the present directory, and “..” for the directory above. Hidden files will be shown with the `-a` flag.

Use `cd` to change to the directory, for example to `Documents` as shown in Listing A.12. · `cd`

Listing A.12: Change directory with `cd`.

```
1 FN11194:~ sporring$ cd Documents/  
2 FN11194:Documents sporring$
```

Note that some graphical user interfaces translate standard filenames and directories to the local language, such that navigating using the graphical user interface will reveal other files and directories, which, however, are aliases.

You can create a new directory using `mkdir`, as demonstrated in Listing A.13. · `mkdir`

Listing A.13: Creating a directory using `mkdir`.

```
1 FN11194:Documents sporring$ mkdir myFolder  
2 FN11194:Documents sporring$ ls  
3 myFolder  
4 FN11194:tmp sporring$
```

A file can be created using `echo` and with *redirection*, as shown in Listing A.14. · `echo`
· *redirection*

Listing A.14: Creating a file with `echo` and redirection.

```
1 FN11194:Documents sporring$ echo "hi" > hi.txt  
2 FN11194:Documents sporring$ ls  
3 hi.txt          myFolder
```

To move the file `hi.txt` into `myFolder`, use `mv`. This is demonstrated in Listing A.15. · `mv`

Listing A.15: Moving files with `mv`.

```
1 FN11194:Documents sporring$ echo mv hi.txt myFolder/  
2 FN11194:Documents sporring$
```

To delete the file and the directory, use `rm` and `rmdir`, as shown in Listing A.16. · `rm`
· `rmdir`

Listing A.16: Deleting files and directories.

```
1 FN11194:Documents sporring$ cd myFolder/  
2 FN11194:myFolder sporring$ rm hi.txt  
3 FN11194:myFolder sporring$ cd ..  
4 FN11194:Documents sporring$ rmdir myFolder/  
5 FN11194:Documents sporring$ ls  
6 FN11194:Documents sporring$
```

Only commands found on the *search-path* are available in the console. The content of the · *search-path*
search-path is seen using the `echo` command, as demonstrated in Listing A.17.

Listing A.17: The content of the search-path.

```
1 FN11194:Documents sporring$ echo $PATH
2 /Applications/Maple
   17//Applications/PackageManager.app/Contents/MacOS/:
   /Applications/MATLAB_R2014b.app/bin/:/opt/local/bin:
   /opt/local/sbin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:
   /sbin:/opt/X11/bin:/Library/TeX/texbin
3 FN11194:Documents sporring$
```

The search-path can be changed by editing the setup file for Bash. On MacOS X it is called `~/.profile`, and on Linux it is either `~/.bash_profile` or `~/.bashrc`. Here new paths can be added by adding the following line: `export PATH=<new path>:<another new path>:$PATH`.

A useful feature of Bash is that the console can help you write commands. E.g., if you write `fs` followed by pressing the `tab`-key, and if `Mono` is in the search-path, then Bash will typically respond by completing the line as `fsharp`, and by further pressing the `tab`-key some times, Bash will show the list of options, typically `fshpari` and `fsharpc`. Also, most commands have an extensive manual which can be accessed using the `man` command. E.g., the manual for `rm` is retrieved by `man rm`.

Bibliography

- [1] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345—363, 1936.
- [2] Ole-Johan Dahl and Kristen Nygaard. SIMULA a language for programming and description of discrete event systems. introduction and user’s manual. Technical report, Norwegian Computing Center, 1967.
- [3] European Computer Manufacturers Association (ECMA). Standard ecma-335, common language infrastructure (cli). <http://www.ecma-international.org/publications/standards/Ecma-335.htm>.
- [4] International Organization for Standardization. Iso/iec 23271:2012, common language infrastructure (cli). <https://www.iso.org/standard/58046.html>.
- [5] Object Management Group. Uml version 2.0. <http://www.omg.org/spec/UML/2.0/>.
- [6] Programming Research Group. Specifications for the ibm mathematical formula translating system, fortran. Technical report, Applied Science Division, International Business Machines Corporation, 1954.
- [7] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, 1960.
- [8] X3: ASA Sectional Committee on Computers and Information Processing. American standard code for information interchange. Technical Report ASA X3.4-1963, American Standards Association (ASA), 1963. <http://worldpowersystems.com/projects/codes/X3.4-1963/>.
- [9] George Pólya. *How to solve it*. Princeton University Press, 1945.
- [10] Alan M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1936.

Index

`bash`, 404

`cd`, 402, 406

console, 400

`del`, 403

`dir`, 402

directory, 400

`echo`, 403, 406

folder, 400

graphical user interface, 400

GUI, 400

`ls`, 405

`mkdir`, 402, 406

`move`, 403

`mv`, 406

redirection, 403, 406

`rm`, 406

`rmdir`, 403, 406

search path, 404

search-path, 406

terminal, 400

Windows command line, 400