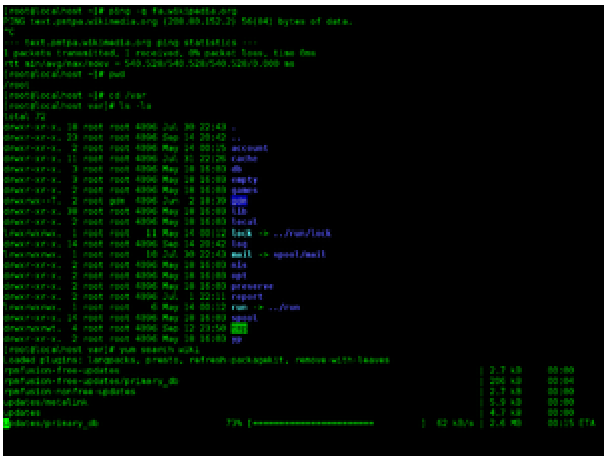
Wikipedia

Command-line interface

A **command-line interface** (**CLI**) processes commands to a computer program in the form of lines of text. The program which handles the interface is called a **command-line interpreter** or **command-line processor**. Operating systems implement a command-line interface in a shell for interactive access to operating system functions or services. Such access was primarily provided to users by computer terminals starting in the mid-1960s, and continued to be used throughout the 1970s and 1980s on VAX/VMS, Unix systems and personal computer systems including DOS, CP/M and Apple DOS.

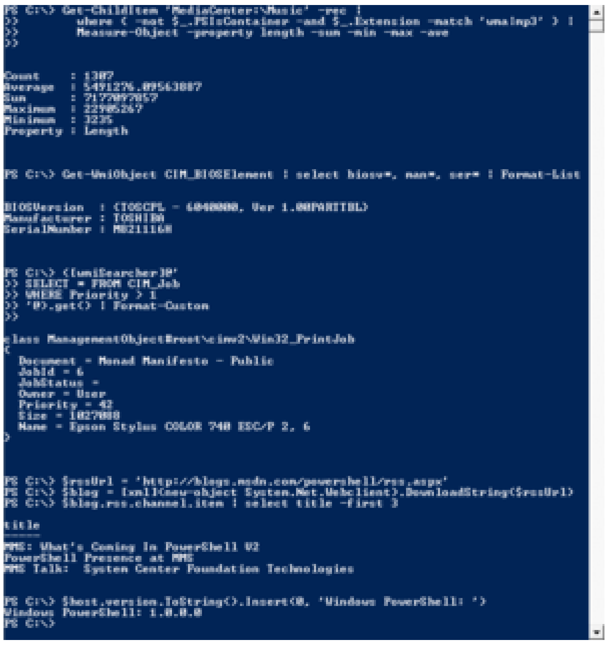


```
[root@localhost ~]# ping -q fa.wikipedia.org
PING text.pmtpa.wikimedia.org (208.80.152.2) 56(84) bytes of data.
^C
--- text.pmtpa.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 540.528/540.528/540.528/0.000 ms
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /var
[root@localhost var]# ls -la
total 72
drwxr-xr-x. 18 root root 4096 Jul 30 22:43 .
drwxr-xr-x. 23 root root 4096 Sep 14 20:42 ..
drwxr-xr-x.  2 root root 4096 May 14 00:15 account
drwxr-xr-x. 11 root root 4096 Jul 31 22:26 cache
drwxr-xr-x.  3 root root 4096 May 18 16:03 db
drwxr-xr-x.  3 root root 4096 May 18 16:03 empty
drwxr-xr-x.  2 root root 4096 May 18 16:03 games
drwxrwx--T.  2 root gdm  4096 Jun  2 18:39 gdm
drwxr-xr-x. 38 root root 4096 May 18 16:03 lib
drwxr-xr-x.  2 root root 4096 May 18 16:03 local
lrwxrwxrwx.  1 root root   11 May 14 00:12 lock -> ../run/lock
drwxr-xr-x. 14 root root 4096 Sep 14 20:42 log
lrwxrwxrwx.  1 root root   10 Jul 30 22:43 mail -> spool/mail
drwxr-xr-x.  2 root root 4096 May 18 16:03 nis
drwxr-xr-x.  2 root root 4096 May 18 16:03 opt
drwxr-xr-x.  2 root root 4096 May 18 16:03 preserve
drwxr-xr-x.  2 root root 4096 Jul  1 22:11 report
lrwxrwxrwx.  1 root root    6 May 14 00:12 run -> ../run
drwxr-xr-x. 14 root root 4096 May 18 16:03 spool
drwxrwxrwt.  4 root root 4096 Sep 12 23:50 tmp
drwxr-xr-x.  2 root root 4096 May 18 16:03 yp
[root@localhost var]# yum search wiki
Loaded plugins: langpacks, presto, refresh-packagekit, remove-with-leaves
rpmfusion-free-updates                                   | 2.7 kB     00:00
rpmfusion-free-updates/primary_db                        | 206 kB     00:04
rpmfusion-nonfree-updates                                | 2.7 kB     00:00
updates/metalink                                         | 5.9 kB     00:00
updates                                                  | 4.7 kB     00:00
updates/primary_db           73% [=================      ]  62 kB/s | 2.4 MB     00:15 ETA
```

Screenshot of a sample Bash session in GNOME Terminal 3, Fedora 15

Today, many users rely upon graphical user interfaces and menu-driven interactions. However, some programming and maintenance tasks may not have a graphical user interface and may still use a command line.

Alternatives to the command line interface include text-based user interface menus (for example, IBM AIX SMIT), keyboard shortcuts, and various desktop metaphors centered on the pointer (usually controlled with a mouse). Examples of this include the Microsoft Windows, DOS Shell, and Mouse Systems PowerPanel. Command-line interfaces are often implemented in terminal devices that are also capable of screen-oriented text-based user interfaces that use cursor addressing to place symbols on a display screen.



```
PS C:\> Get-ChildItem 'MediaCenter:\Music' -rec |
>>        where { -not $_.PSIsContainer -and $_.Extension -match 'wma|mp3' } |
>>        Measure-Object -property length -sum -min -max -ave
>>


Count    : 1307
Average  : 5491276.09563887
Sum      : 7177097857
Maximum  : 22905267
Minimum  : 3235
Property : Length



PS C:\> Get-WmiObject CIM_BIOSElement | select biosv*, man*, ser* | Format-List


BIOSVersion  : {TOSCPL - 6040000, Ver 1.00PARTTBL}
Manufacturer : TOSHIBA
SerialNumber : M821116H



PS C:\> ([wmiSearcher]@'
>> SELECT * FROM CIM_Job
>> WHERE Priority > 1
>> '@).get() | Format-Custom
>>

class ManagementObject#root\cimv2\Win32_PrintJob
{
  Document = Monad Manifesto - Public
  JobId = 6
  JobStatus =
  Owner = User
  Priority = 42
  Size = 1027088
  Name = Epson Stylus COLOR 740 ESC/P 2, 6
}



PS C:\> $rssUrl = 'http://blogs.msdn.com/powershell/rss.aspx'
PS C:\> $blog = [xml](new-object System.Net.WebClient).DownloadString($rssUrl)
PS C:\> $blog.rss.channel.item | select title -first 3

title
-----
MMS: What's Coming In PowerShell V2
PowerShell Presence at MMS
MMS Talk:  System Center Foundation Technologies


PS C:\> $host.version.ToString().Insert(0, 'Windows PowerShell: ')
Windows PowerShell: 1.0.0.0
PS C:\>
```

Screenshot of Windows PowerShell 1.0, running on Windows Vista

Programs with command-line interfaces are generally easier to automate via scripting.

Many software systems implement command-line interfaces for control and operation. This includes programming environments and utility programs.

Contents

Comparison to graphical user interfaces

Types

Operating system command-line interfaces

Application command-line interfaces

History

Usage

Anatomy of a shell CLI
- CLI and resource protection
- Command prompt
- Arguments
  - Command-line option
    - Abbreviating commands
    - Option conventions in DOS, Windows, OS/2
    - Option conventions in Unix-like systems
    - Option conventions in other systems
- Built-in usage help
- Command description syntax
- The space character

Command-line interpreter
- Early history
- Modern usage as an operating system shell

Scripting

Other command-line interfaces

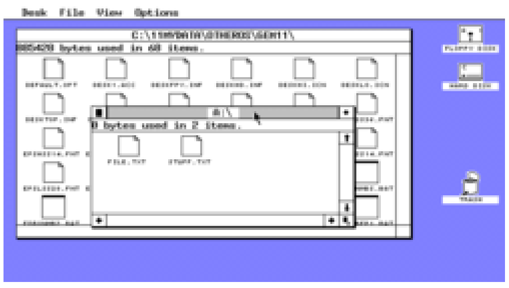
See also

Notes

References

External links

Comparison to graphical user interfaces



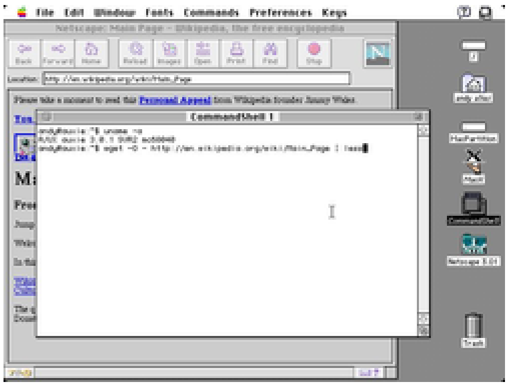
A graphical user interface with icons and windows (GEM 1.1 Desktop)

Compared with a graphical user interface, a command-line interface requires fewer system resources to implement. Since options to commands are given in a few characters in each command line, an experienced user may often find the options easier to access. Automation of repetitive tasks is simplified by line editing and history mechanisms for storing frequently used sequences; this may extend to a scripting language that can take parameters and variable options. A command-line history can be kept, allowing review or repetition of commands.

A command-line system may require paper or online manuals for the user's reference, although often a "help" option provides a concise review of the options of a command. The command-line environment may not provide graphical enhancements such as different fonts or extended edit windows found in a GUI. It may be difficult for a new user to become familiar with all the commands and options available, compared with the icons and drop-down menus of a graphical user interface, without repeated reference to manuals.

Types

Operating system command-line interfaces



```
andy@aucie ~$ uname -a
A/UX aucie 3.0.1 SVR2 mc68040
andy@aucie ~$ wget -O - http://en.wikipedia.org/wiki/Main_Page | less
```

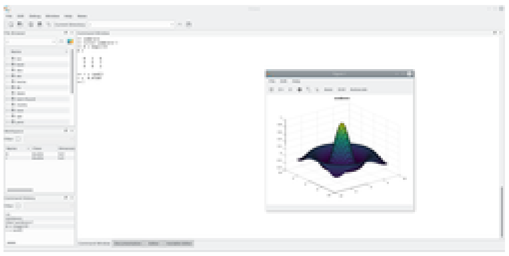
Apple Computer's CommandShell in A/UX 3.0.1

Operating system (OS) command-line interfaces are usually distinct programs supplied with the operating system. A program that implements such a text interface is often called a command-line interpreter, command processor or shell.

Examples of command-line interpreters include DEC's DIGITAL Command Language (DCL) in OpenVMS and RSX-11, the various Unix shells (sh, ksh, csh, tcsh, zsh, bash, etc.), CP/M's CCP, DOS' COMMAND.COM, as well as the OS/2 and the Windows CMD.EXE programs, the latter groups being based heavily on DEC's RSX-11 and RSTS CLIs. Under most operating systems, it is possible to replace the default shell program with alternatives; examples include 4DOS for DOS, 4OS2 for OS/2, and 4NT / Take Command for Windows.

Although the term 'shell' is often used to describe a command-line interpreter, strictly speaking, a 'shell' can be any program that constitutes the user-interface, including fully graphically oriented ones. For example, the default Windows GUI is a shell program named EXPLORER.EXE, as defined in the SHELL=EXPLORER.EXE line in the WIN.INI configuration file. These programs are shells, but not CLIs.

Application command-line interfaces


GNU Octave's GUI with command-line interface

Application programs (as opposed to operating systems) may also have command-line interfaces.

An application program may support none, any, or all of these three major types of command-line interface mechanisms:

- *Parameters*: Most operating systems support a means to pass additional information to a program when it is launched. When a program is launched from an OS command-line shell, additional text provided along with the program name is passed to the launched program.
- *Interactive command-line sessions*: After launch, a program may provide an operator with an independent means to enter commands in the form of text.
- *Inter-process communication*: Most operating systems support means of inter-process communication (for example, standard streams or named pipes). Command lines from client processes may be redirected to a CLI program by one of these methods.

Some applications support only a CLI, presenting a CLI prompt to the user and acting upon command lines as they are entered. Other programs support both a CLI and a GUI. In some cases, a GUI is simply a wrapper around a separate CLI executable file. In other cases, a program may provide a CLI as an optional alternative to its GUI. CLIs and GUIs often support different functionality. For example, all features of MATLAB, a numerical analysis computer program, are available via the CLI, whereas the MATLAB GUI exposes only a subset of features.

The early Sierra games, such as the first three *King's Quest* games (1984–1986), used commands from an internal command line to move the character around in the graphic window.

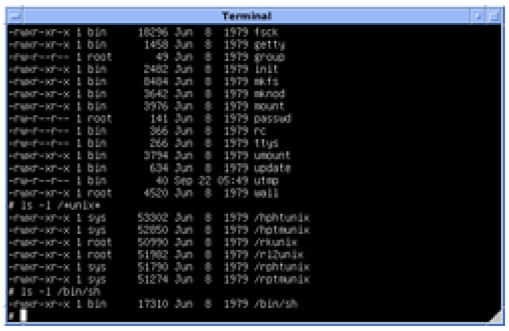
History

The command-line interface evolved from a form of dialog once conducted by humans over teleprinter (TTY) machines, in which human operators remotely exchanged information, usually one line of text at a time. Early computer systems often used teleprinter machines as the means of interaction with a human operator. The computer became one end of the human-to-human teleprinter model. So instead of a human communicating with another human over a teleprinter, a human communicated with a computer.

The mechanical teleprinter was replaced by a "glass tty", a keyboard and screen emulating the teleprinter. "Smart" terminals permitted additional functions, such as cursor movement over the entire screen, or local editing of data on the terminal for transmission to the computer. As the microcomputer revolution replaced the traditional – minicomputer + terminals – time sharing architecture, hardware terminals were replaced by terminal emulators — PC software that interpreted terminal signals sent through the PC's serial ports. These were typically used to interface an organization's new PC's with their existing mini- or mainframe computers, or to connect PC to PC. Some of these PCs were running Bulletin Board System software.

Early operating system CLIs were implemented as part of resident monitor programs, and could not easily be replaced. The first implementation of the shell as a replaceable component was part of the Multics time-sharing operating system.[1] In 1964, MIT Computation Center staff member Louis Pouzin developed the RUNCOM tool for executing command scripts while allowing argument substitution.[2] Pouzin coined the term "shell" to describe the technique of using commands like a programming language, and wrote a paper about how to implement the idea in the Multics operating system.[3] Pouzin returned to his native France in 1965, and the first Multics shell was developed by Glenda Schroeder.[2]

The first Unix shell, the V6 shell, was developed by Ken Thompson in 1971 at Bell Labs and was modeled after Schroeder's Multics shell.[4][5] The Bourne shell was introduced in 1977 as a replacement for the V6 shell. Although it is used as an interactive command interpreter, it was also intended as a scripting language and contains most of the features that are commonly considered to produce structured programs. The Bourne shell led to the development of the KornShell (ksh), Almquist shell (ash), and the popular Bourne-again shell (or Bash).[5]



```
-rwxr-xr-x 1 bin     18296 Jun  8  1979 fsck
-rwxr-xr-x 1 bin      1458 Jun  8  1979 getty
-rw-r--r-- 1 root       49 Jun  8  1979 group
-rwxr-xr-x 1 bin      2482 Jun  8  1979 init
-rwxr-xr-x 1 bin      8484 Jun  8  1979 mkfs
-rwxr-xr-x 1 bin      3642 Jun  8  1979 mknod
-rwxr-xr-x 1 bin      3976 Jun  8  1979 mount
-rw-r--r-- 1 root      141 Jun  8  1979 passwd
-rw-r--r-- 1 bin       366 Jun  8  1979 rc
-rw-r--r-- 1 bin       266 Jun  8  1979 ttys
-rwxr-xr-x 1 bin      3794 Jun  8  1979 umount
-rwxr-xr-x 1 bin       634 Jun  8  1979 update
-rw-r--r-- 1 bin        40 Sep 22 05:49 utmp
-rwxr-xr-x 1 root     4520 Jun  8  1979 wall
# ls -l /*unix*
-rwxr-xr-x 1 sys     53302 Jun  8  1979 /hphtunix
-rwxr-xr-x 1 sys     52850 Jun  8  1979 /hptmunix
-rwxr-xr-x 1 root    50990 Jun  8  1979 /rkunix
-rwxr-xr-x 1 root    51982 Jun  8  1979 /rl2unix
-rwxr-xr-x 1 sys     51790 Jun  8  1979 /rphtunix
-rwxr-xr-x 1 sys     51274 Jun  8  1979 /rptmunix
# ls -l /bin/sh
-rwxr-xr-x 1 bin     17310 Jun  8  1979 /bin/sh
# 
```

Bourne shell interaction on Version 7 Unix

Early microcomputers themselves were based on a command-line interface such as CP/M, DOS or AppleSoft BASIC. During the 1980s and 1990s, the introduction of the Apple Macintosh and of Microsoft Windows on PCs saw the command line interface as the primary user interface replaced by the Graphical User Interface. The command line remained available as an alternative user interface, often used by system administrators and other advanced users for system administration, computer programming and batch processing.

In November 2006, Microsoft released version 1.0 of Windows PowerShell (formerly codenamed *Monad*), which combined features of traditional Unix shells with their proprietary object-oriented .NET Framework. MinGW and Cygwin are open-source packages for Windows that offer a Unix-like CLI. Microsoft provides MKS Inc.'s ksh implementation *MKS Korn shell* for Windows through their Services for UNIX add-on.

Since 2001, the Macintosh operating system macOS has been based on a Unix-like operating system called Darwin. On these computers, users can access a Unix-like command-line interface by running the terminal emulator program called Terminal, which is found in the Utilities sub-folder of the Applications folder, or by remotely logging into the machine using ssh. Z shell is the default shell for macOS; bash, tcsh, and the KornShell are also provided. Before macOS Catalina, bash was the default.

Usage

A CLI is used whenever a large vocabulary of commands or queries, coupled with a wide (or arbitrary) range of options, can be entered more rapidly as text than with a pure GUI. This is typically the case with operating system command shells. CLIs are also used by systems with insufficient resources to support a graphical user interface. Some computer language systems (such as Python, Forth, LISP, Rexx, and many dialects of BASIC) provide an interactive command-line mode to allow for rapid evaluation of code.

CLIs are often used by programmers and system administrators, in engineering and scientific environments, and by technically advanced personal computer users. CLIs are also popular among people with visual disabilities since the commands and responses can be displayed using refreshable Braille displays.

Anatomy of a shell CLI

The general pattern of an OS command line interface[6][7] is:

```
Prompt command param1 param2 param3 … paramN
```

- Prompt — generated by the program to provide context for the client.
- Command — provided by the client. Commands are usually one of three classes:
  1. *Internal* commands are recognized and processed by the command line interpreter itself and not dependent upon any external executable file.
  2. *Included* commands run separate executable files generally considered part of the operating environment and always included with the OS.
  3. *External* commands run executable files that are not part of the basic OS, but added by other parties for specific purposes and applications.
- param1 …paramN — Optional parameters provided by the client. The format and meaning of the parameters depends upon the command issued. In the case of Included or External commands, the values of the parameters are delivered to the program (specified by the Command) as it is launched by the OS. Parameters may be either Arguments or Options.

In this example, the delimiters between command-line elements are whitespace characters and the end-of-line delimiter is the newline delimiter. This is a widely used (but not universal) convention for command-line interfaces.

A CLI can generally be considered as consisting of syntax and semantics. The *syntax* is the grammar that all commands must follow. In the case of operating systems, DOS and Unix each define their own set of rules that all commands must follow. In the case of embedded systems, each vendor, such as Nortel, Juniper Networks or Cisco Systems, defines their own proprietary set of rules that all commands within their CLI conform to. These rules also dictate how a user navigates through the system of commands. The *semantics* define what sort of operations are possible, on what sort of data these operations can be performed, and how the grammar represents these operations and data—the symbolic meaning in the syntax.

Two different CLIs may agree on either syntax or semantics, but it is only when they agree on both that they can be considered sufficiently similar to allow users to use both CLIs without needing to learn anything, as well as to enable re-use of scripts.

A simple CLI will display a prompt, accept a "command line" typed by the user terminated by the Enter key, then execute the specified command and provide textual display of results or error messages. Advanced CLIs will validate, interpret and parameter-expand the command line before executing the specified command, and optionally capture or redirect its output.

Unlike a button or menu item in a GUI, a command line is typically self-documenting, stating exactly what the user wants done. In addition, command lines usually include many defaults that can be changed to customize the results. Useful command lines can be saved by assigning a character string or alias to represent the full command, or several commands can be grouped to perform a more complex sequence – for instance, compile the program, install it, and run it — creating a single entity, called a command procedure or script which itself can be treated as a command. These advantages mean that a user must figure out a complex command or series of commands only once, because they can be saved, to be used again.

The commands given to a CLI shell are often in one of the following forms:

- `doSomething how toFiles`
- `doSomething how sourceFile destinationFile`
- `doSomething how < inputFile > outputFile`
- `doSomething how | doSomething how | doSomething how > outputFile`

where *doSomething* is, in effect, a verb, *how* an adverb (for example, should the command be executed "verbosely" or "quietly") and *toFiles* an object or objects (typically one or more files) on which the command should act. The `>` in the third example is a redirection operator, telling the command-line interpreter to send the output of the command not to its own standard output (the screen) but to the named file. This will overwrite the file. Using `>>` will redirect the output and append it to the file. Another redirection operator is the vertical bar (`|`), which creates a pipeline where the output of one command becomes the input to the next command.

CLI and resource protection

One can modify the set of available commands by modifying which paths appear in the PATH environment variable. Under Unix, commands also need be marked as executable files. The directories in the path variable are searched in the order they are given. By re-ordering the path, one can run e.g. \OS2\MDOS\E.EXE instead of \OS2\E.EXE, when the default is the opposite. Renaming of the executables also works: people often rename their favourite editor to EDIT, for example.

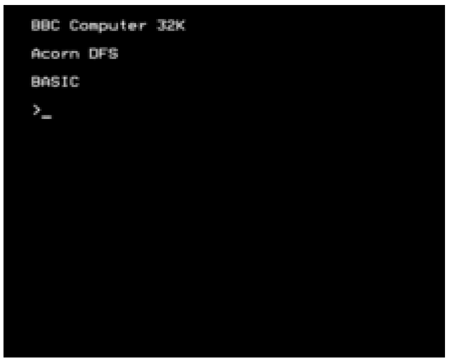
The command line allows one to restrict available commands, such as access to advanced internal commands. The Windows CMD.EXE does this. Often, shareware programs will limit the range of commands, including printing a command 'your administrator has disabled running batch files' from the prompt.

Some CLIs, such as those in network routers, have a hierarchy of modes, with a different set of commands supported in each mode. The set of commands are grouped by association with security, system, interface, etc. In these systems the user might traverse through a series of sub-modes. For example, if the CLI had two modes called *interface* and *system*, the user might use the command *interface* to enter the interface mode. At this point, commands from the system mode may not be accessible until the user exits the interface mode and enters the system mode.

Command prompt

A command prompt (or just *prompt*) is a sequence of (one or more) characters used in a command-line interface to indicate readiness to accept commands. It literally prompts the user to take action. A prompt usually ends with one of the characters `$`, `%`, `#`,[8][9] `:`, `>` or `-`[10] and often includes other information, such as the path of the current working directory and the hostname.

On many Unix and derivative systems, the prompt commonly ends in `$` or `%` if the user is a normal user, but in `#` if the user is a superuser ("root" in Unix terminology).



```
BBC Computer 32K
Acorn DFS
BASIC
>_
```

Prompt of a BBC Micro after switch-on or hard reset

End-users can often modify prompts. Depending on the environment, they may include colors, special characters, and other elements (like variables and functions for the current time, user, shell number or working directory) in order, for instance, to make the prompt more informative or visually pleasing, to distinguish sessions on various machines, or to indicate the current level of nesting of commands. On some systems, special tokens in the definition of the prompt can be used to cause external programs to be called by the command-line interpreter while displaying the prompt.

In DOS' COMMAND.COM and in Windows NT's cmd.exe users can modify the prompt by issuing a `PROMPT` command or by directly changing the value of the corresponding `%PROMPT%` environment variable. The default of most modern systems, the `C:\>` style is obtained, for instance, with `PROMPT $P$G`. The default of older DOS systems, `C>` is obtained by just `PROMPT`, although on some systems this produces the newer `C:\>` style, unless used on floppy drives A: or B:; on those systems `PROMPT $N$G` can be used to override the automatic default and explicitly switch to the older style.

Many Unix systems feature the `$PS1` variable (Prompt String 1),[11] although other variables also may affect the prompt (depending on the shell used). In the bash shell, a prompt of the form:

```
[time] user@host: work_dir $
```

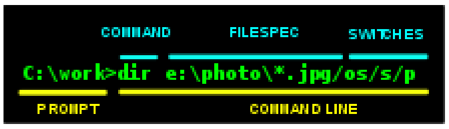
could be set by issuing the command

```
export PS1='[\t] \u@\H: \W $'
```

In zsh the `$RPROMPT` variable controls an optional "prompt" on the right-hand side of the display. It is not a real prompt in that the location of text entry does not change. It is used to display information on the same line as the prompt, but right-justified.

In RISC OS the command prompt is a `*` symbol, and thus (OS)CLI commands are often referred to as "star commands".[12] One can also access the same commands from other command lines (such as the BBC BASIC command line), by preceding the command with a `*`.

Arguments



```
C:\work>dir e:\photo\*.jpg/os/s/p
```

An MS-DOS command line, illustrating parsing into command and arguments

A **command-line argument** or **parameter** is an item of information provided to a program when it is started. A program can have many command-line arguments that identify sources or destinations of information, or that alter the operation of the program.

When a command processor is active a program is typically invoked by typing its name followed by command-line arguments (if any). For example, in Unix and Unix-like environments, an example of a command-line argument is:

```
rm file.s
```

"file.s" is a command-line argument which tells the program rm to remove the file "file.s".

Some programming languages, such as C, C++ and Java, allow a program to interpret the command-line arguments by handling them as string parameters in the main function. Other languages, such as Python, expose operating system specific API (functionality) through `sys` module, and in particular `sys.argv` for "command-line arguments".

In Unix-like operating systems, a single hyphen used in place of a file name is a special value specifying that a program should handle data coming from the standard input or send data to the standard output.

Command-line option

A **command-line option** or simply **option** (also known as a **flag** or **switch**) modifies the operation of a command; the effect is determined by the command's program. Options follow the command name on the command line, separated by spaces. A space before the first option is not always required, such as `Dir/?` and `DIR /?` in DOS, which have the same effect[10] of listing the DIR command's available options, whereas `dir --help` (in many versions of Unix) *does* require the option to be preceded by at least one space (and is case-sensitive).

The format of options varies widely between operating systems. In most cases the syntax is by convention rather than an operating system requirement; the entire command line is simply a string passed to a program, which can process it in any way the programmer wants, so long as the interpreter can tell where the command name ends and its arguments and options begin.

A few representative samples of command-line options, all relating to listing files in a directory, to illustrate some conventions:

Operating system	Command	Valid alternative	Notes
OpenVMS	`directory/owner`	`Dir /Owner`	instruct the *directory* command to also display the ownership of the files. *Note the Directory command name is not case sensitive, and can be abbreviated to as few letters as required to remain unique.*
Windows	`DIR/Q/O:S d*`	`dir /q d* /o:s`	display ownership of files whose names begin with "D", sorted by size, smallest first. *Note spaces around argument d* are required.*
Unix-like systems	`ls -lS D*`	`ls -S -l D*`	display in long format files and directories beginning with "D" (but not "d"), sorted by size (largest first). *Note spaces are required around all arguments and options, but some can be run together, e.g. **-lS** is the same as **-l -S**.*
Data General RDOS CLI	`list/e/s 04-26-80/b`	`List /S/E 4-26-80/B`	list every attribute for files created before 26 April 1980. *Note the /B at the end of the date argument is a **local switch**, that modifies the meaning of that argument, while /S and /E are **global switches**, i.e. apply to the whole command.*

Abbreviating commands

In Multics, command-line options and subsystem keywords may be abbreviated. This idea appears to derive from the PL/I programming language, with its shortened keywords (e.g., STRG for STRINGRANGE and DCL for DECLARE). For example, in the Multics "forum" subsystem, the *-long_subject* parameter can be abbreviated *-lgsj*. It is also common for Multics commands to be abbreviated, typically corresponding to the initial letters of the words that are strung together with underscores to form command names, such as the use of *did* for *delete_iacl_dir*.

In some other systems abbreviations are automatic, such as permitting enough of the first characters of a command name to uniquely identify it (such as `SU` as an abbreviation for `SUPERUSER`) while others may have some specific abbreviations pre-programmed (e.g. `MD` for `MKDIR` in COMMAND.COM) or user-defined via batch scripts and aliases (e.g. `alias md mkdir` in tcsh).

Option conventions in DOS, Windows, OS/2

On DOS, OS/2 and Windows, different programs called from their COMMAND.COM or CMD.EXE (or internal their commands) may use different syntax within the same operating system. For example:

- Options may be indicated by either of the "switch characters": `/`, `-`, or either may be allowed. See below.
- They may or may not be case-sensitive.
- Sometimes options and their arguments are run together, sometimes separated by whitespace, and sometimes by a character, typically `:` or `=`; thus `Prog -fFilename`, `Prog -f Filename`, `Prog -f:Filename`, `Prog -f=Filename`.
- Some programs allow single-character options to be combined;[10] others do not. The switch `-fA` may mean the same as `-f -A`,[10] or it may be incorrect, or it may even be a valid but different parameter.

In DOS, OS/2 and Windows, the forward slash (`/`) is most prevalent, although the hyphen-minus is also sometimes used. In many versions of DOS (MS-DOS/PC DOS 2.xx and higher, all versions of DR-DOS since 5.0, as well as PTS-DOS, Embedded DOS, FreeDOS and RxDOS) the **switch character** (sometimes abbreviated **switchar** or **switchchar**) to be used is defined by a value returned from a system call (INT 21h/AX=3700h). The default character returned by this API is `/`, but can be changed to a hyphen-minus on the above-mentioned systems, except for under Datalight ROM-DOS and MS-DOS/PC DOS 5.0 and higher, which always return `/` from this call (unless one of many available TSRs to reenable the SwitChar feature is loaded). In some of these systems (MS-DOS/PC DOS 2.xx, DOS Plus 2.1, DR-DOS 7.02 and higher, PTS-DOS, Embedded DOS, FreeDOS and RxDOS), the setting can also be pre-configured by a SWITCHAR directive in CONFIG.SYS. General Software's Embedded DOS provides a SWITCH command for the same purpose, whereas 4DOS allows the setting to be changed via `SETDOS /W:n`.[13] Under DR-DOS, if the setting has been changed from `/`, the first directory separator `\` in the display of the PROMPT parameter `$G` will change to a forward slash `/` (which is also a valid directory separator in DOS, FlexOS, 4680 OS, 4690 OS, OS/2 and Windows) thereby serving as a visual clue to indicate the change.[10] Also, the current setting is reflected also in the built-in help screens.[10] Some versions of DR-DOS COMMAND.COM also support a PROMPT token `$/` to display the current setting. COMMAND.COM since DR-DOS 7.02 also provides a pseudo-environment variable named `%/%` to allow portable batchjobs to be written.[14][15] Several external DR-DOS commands additionally support an environment variable `%SWITCHAR%` to override the system setting.

However, many programs are hardwired to use `/` only, rather than retrieving the switch setting before parsing command-line arguments. A very small number, mainly ports from Unix-like systems, are programmed to accept "-" even if the switch character is not set to it (for example `netstat` and `ping`, supplied with

Microsoft Windows, will accept the /? option to list available options, and yet the list will specify the "-" convention).

Option conventions in Unix-like systems

In Unix-like systems, the ASCII hyphen-minus begins options; the new (and GNU) convention is to use *two* hyphens then a word (e.g. `--create`) to identify the option's use while the old convention (and still available as an option for frequently-used options) is to use one hyphen then one letter (e.g., `-c`); if one hyphen is followed by two or more letters it may mean two options are being specified, or it may mean the second and subsequent letters are a parameter (such as filename or date) for the first option.

Two hyphen-minus characters without following letters (`--`) may indicate that the remaining arguments should not be treated as options, which is useful for example if a file name itself begins with a hyphen, or if further arguments are meant for an inner command (e.g., sudo). Double hyphen-minuses are also sometimes used to prefix "long options" where more descriptive option names are used. This is a common feature of GNU software. The *getopt* function and program, and the *getopts* command are usually used for parsing command-line options.

Unix command names, arguments and options are case-sensitive (except in a few examples, mainly where popular commands from other operating systems have been ported to Unix).

Option conventions in other systems

FlexOS, 4680 OS and 4690 OS use `-`.

CP/M typically used `[`.

Conversational Monitor System (CMS) uses a single left parenthesis to separate options at the end of the command from the other arguments. For example, in the following command the options indicate that the target file should be replaced if it exists, and the date and time of the source file should be retained on the copy: `COPY source file a target file b (REPLACE OLDDATE`

Data General's CLI under their RDOS, AOS, etc. operating systems, as well as the version of CLI that came with their Business Basic, uses only `/` as the switch character, is case-insensitive, and allows "local switches" on some arguments to control the way they are interpreted, such as `MAC/U LIB/S A B C $LPT/L` has the global option "U" to the macro assembler command to append user symbols, but two local switches, one to specify LIB should be skipped on pass 2 and the other to direct listing to the printer, $LPT.

Built-in usage help

One of the criticisms of a CLI is the lack of cues to the user as to the available actions. In contrast, GUIs usually inform the user of available actions with menus, icons, or other visual cues. To overcome this limitation, many CLI programs display a brief summary of its valid parameters, typically when invoked with no arguments or one of `?`, `-?`, `-h`, `-H`, `/?`, `/h`, `/H`, `/Help`, `-help`, or `--help`.[10][16][17]

However, entering a program name without parameters in the hope that it will display usage help can be hazardous, as programs and scripts for which command line arguments are optional will execute without further notice.

Although desirable at least for the help parameter, programs may not support all option lead-in characters exemplified above. Under DOS, where the default command-line option character can be changed from `/` to `-`, programs may query the SwitChar API in order to determine the current setting. So, if a program is not hardwired to support them all, a user may need to know the current setting even to be able to reliably request help. If the SwitChar has been changed to `-` and therefore the `/` character is accepted as alternative path delimiter also at the DOS command line, programs may misinterpret options like `/h` or `/H` as paths rather than help parameters.[10] However, if given as first or only parameter, most DOS programs will, by convention, accept it as request for help regardless of the current SwitChar setting.[10][13]

In some cases, different levels of help can be selected for a program. Some programs supporting this allow to give a verbosity level as an optional argument to the help parameter (as in `/H:1`, `/H:2`, etc.) or they give just a short help on help parameters with question mark and a longer help screen for the other help options.[18]

Depending on the program, additional or more specific help on accepted parameters is sometimes available by either providing the parameter in question as an argument to the help parameter or vice versa (as in `/H:W` or in `/W:?` (assuming `/W` would be another parameter supported by the program)).[19][20][17][16][18][nb 1]

In a similar fashion to the help parameter, but much less common, some programs provide additional information about themselves (like mode, status, version, author, license or contact information) when invoked with an "about" parameter like `-!`, `/!`, `-about`, or `--about`.[16]

Since the `?` and `!` characters typically also serve other purposes at the command line, they may not be available in all scenarios, therefore, they should not be the only options to access the corresponding help information.

If more detailed help is necessary than provided by a program's built-in internal help, many systems support a dedicated external "`help` *`command`*" command (or similar), which accepts a command name as calling parameter and will invoke an external help system.

The end of the HELP command output from RT-11SJ displayed on a VT100

In the DR-DOS family, typing `/?` or `/H` at the COMMAND.COM prompt instead of a command itself will display a dynamically generated list of available internal commands;[10] 4DOS and NDOS support the same feature by typing `?` at the prompt[13] (which is also accepted by newer versions of DR-DOS COMMAND.COM); internal commands can be individually disabled or reenabled via `SETDOS /I`.[13] In addition to this, some newer versions of DR-DOS COMMAND.COM also accept a `?%` command to display a list of available built-in pseudo-environment variables. Besides their purpose as quick help reference this can be used in batchjobs to query the facilities of the underlying command-line processor.[10]

Command description syntax

Built-in usage help and man pages commonly employ a small syntax to describe the valid command form:[21][22][23][nb 2]

- angle brackets for *required* parameters: `ping <hostname>`
- square brackets for *optional* parameters: `mkdir [-p] <dirname>`
- ellipses for *repeated* items: `cp <source1> [source2…] <dest>`

- vertical bars for *choice* of items: `netstat {-t|-u}`

Notice that these characters have different meanings than when used directly in the shell. Angle brackets may be omitted when confusing the parameter name with a literal string is not likely.

The space character

In many areas of computing, but particularly in the command line, the space character can cause problems as it has two distinct and incompatible functions: as part of a command or parameter, or as a parameter or name separator. Ambiguity can be prevented either by prohibiting embedded spaces in file and directory names in the first place (for example, by substituting them with underscores `_`), or by enclosing a name with embedded spaces between quote characters or using an escape character before the space, usually a backslash (`\`). For example

```
Long path/Long program name Parameter one Parameter two …
```

is ambiguous (is "program name" part of the program name, or two parameters?); however

```
Long_path/Long_program_name Parameter_one Parameter_two …,
LongPath/LongProgramName ParameterOne ParameterTwo …,
"Long path/Long program name" "Parameter one" "Parameter two" …
```

and

```
Long\ path/Long\ program\ name Parameter\ one Parameter\ two …
```

are not ambiguous. Unix-based operating systems minimize the use of embedded spaces to minimize the need for quotes. In Microsoft Windows, one often has to use quotes because embedded spaces (such as in directory names) are common.

Command-line interpreter

> Although most users think of the shell as an interactive command interpreter, it is really a programming language in which each statement runs a command. Because it must satisfy both the interactive and programming aspects of command execution, it is a strange language, shaped as much by history as by design.
>
> — Brian W. Kernighan & Rob Pike[24]

The term **command-line interpreter** (**CLI**) is applied to computer programs designed to interpret a sequence of lines of text which may be entered by a user, read from a file or another kind of data stream. The context of interpretation is usually one of a given operating system or programming language.

Command-line interpreters allow users to issue various commands in a very efficient (and often terse) way. This requires the user to know the names of the commands and their parameters, and the syntax of the language that is interpreted.

The Unix `#!` mechanism and OS/2 EXTPROC command facilitate the passing of batch files to external processors. One can use these mechanisms to write specific command processors for dedicated uses, and process external data files which reside in batch files.

Many graphical interfaces, such as the OS/2 Presentation Manager and early versions of Microsoft Windows use command-lines to call helper programs to open documents and programs. The commands are stored in the graphical shell or in files like the registry or the OS/2 `OS2USER.INI` file.

Early history

A Teletype Model 33 ASR teleprinter keyboard with punched tape reader and punch

The earliest computers did not support interactive input/output devices, often relying on sense switches and lights to communicate with the computer operator. This was adequate for batch systems that ran one program at a time, often with the programmer acting as operator. This also had the advantage of low overhead, since lights and switches could be tested and set with one machine instruction. Later a single system console was added to allow the operator to communicate with the system.

From the 1960s onwards, user interaction with computers was primarily by means of command-line interfaces, initially on machines like the Teletype Model 33 ASR, but then on early CRT-based computer terminals such as the VT52.

DEC VT52 terminal

All of these devices were purely text based, with no ability to display graphic or pictures.[nb 3] For business application programs, text-based menus were used, but for more general interaction the command line was the interface.

Around 1964 Louis Pouzin introduced the concept and the name *shell* in Multics, building on earlier, simpler facilities in the Compatible Time-Sharing System (CTSS).[25]

From the early 1970s the Unix operating system adapted the concept of a powerful command-line environment, and introduced the ability to *pipe* the output of one command in as input to another. Unix also had the capability to save and re-run strings of commands as "shell scripts" which acted like custom commands.

The command-line was also the main interface for the early home computers such as the Commodore PET, Apple II and BBC Micro – almost always in the form of a BASIC interpreter. When more powerful business oriented microcomputers arrived with CP/M and later DOS computers such as the IBM PC, the command-line began to borrow some of the syntax and features of the Unix shells such as globbing and piping of output.

The command-line was first seriously challenged by the PARC GUI approach used in the 1983 Apple Lisa and the 1984 Apple Macintosh. A few computer users used GUIs such as GEOS and Windows 3.1 but the majority of IBM PC users did not replace their COMMAND.COM shell with a GUI until Windows 95 was released in 1995.[26][27]

Modern usage as an operating system shell

While most non-expert computer users now use a GUI almost exclusively, more advanced users have access to powerful command-line environments:

- The default VAX/VMS command shell, using the DCL language, has been ported to Windows systems at least three times, including PC-DCL and Acceler8 DCL Lite. Unix command shells have been ported to VMS and DOS/Windows 95 and Windows NT types of operating systems. COMMAND.COM and Windows NT cmd.exe have been ported to Windows CE and presumably works on Microsoft Windows NT Embedded 4.0

- Windows Resource Kit and Windows Services for Unix include Korn and the Bourne shells along with a Perl interpreter (Services of Unix contains Active State ActivePerl in later versions and Interix for versions 1 and 2 and a shell compiled by Microsoft)
- IBM OS/2 (and derivatives such as eComStation and ArcaOS) has the cmd.exe processor. This copies the COMMAND.COM commands, with extensions to REXX.
- cmd.exe and COMMAND.COM are part of the Windows NT stream of operating systems.
- Yet another cmd.exe is a stripped-down shell for Windows CE 3.0.
- An MS-DOS type interpreter called PocketDOS has been ported to Windows CE machines; the most recent release is almost identical to MS-DOS 6.22 and can also run Windows 1, 2, and 3.0, QBasic and other development tools, 4NT and 4DOS. The latest release includes several shells, namely MS-DOS 6.22, PC DOS 7, DR DOS 3.xx, and others.
- Windows users have a CLI environment named Windows Command Prompt, which might use the CScript interface to alternate programs. PowerShell provides a command-line interface, but its applets are not written in Shell script. Implementations of the Unix shell are also available as part of the POSIX sub-system,[28] Cygwin, MKS Toolkit, UWIN, Hamilton C shell and other software packages. Available shells for these interoperability tools include csh, ksh, sh, bash, rsh, tclsh and less commonly zsh, psh
- COMMAND.COM (4DOS), Windows NT cmd.exe (4NT, TCC), and OS/2 cmd.exe (4OS2) and others based on them are enhanced shells which can be a replacement for the native shell or a means of enhancement of the default shell.
- Implementations of PHP have a shell for interactive use called php-cli.
- Standard Tcl/Tk has two interactive shells, Tclsh and Wish, the latter being the GUI version.
- Python, Ruby, Lua, XLNT, and other interpreters also have command shells for interactive use.
- FreeBSD uses tcsh as its default interactive shell for the superuser, and ash as default scripting shell.
- Apple macOS[nb 4] and many Linux distributions have the Bash implementation of the Unix shell. Early versions of macOS used tcsh as the default shell.
- Embedded Linux (and other embedded Unix-like) devices often use the Ash implementation of the Unix shell, as part of Busybox.
- Android uses the mksh shell,[29][30] which replaces a shell derived from ash[31] that was used in older Android versions, supplemented with commands from the separate *toolbox*[32] binary.
- Routers with Cisco IOS,[33] Junos[34] and many others are commonly configured from the command line.

Scripting

Most command-line interpreters support scripting, to various extents. (They are, after all, interpreters of an interpreted programming language, albeit in many cases the language is unique to the particular command-line interpreter.) They will interpret scripts (variously termed shell scripts or batch files) written in the language that they interpret. Some command-line interpreters also incorporate the interpreter engines of other languages, such as REXX, in addition to their own, allowing the executing of scripts, in those languages, directly within the command-line interpreter itself.

Conversely, scripting programming languages, in particular those with an eval function (such as REXX, Perl, Python, Ruby or Jython), can be used to implement command-line interpreters and filters. For a few operating systems, most notably DOS, such a command interpreter provides a more flexible command-line interface than the one supplied. In other cases, such a command interpreter can present a highly customised user interface employing the user interface and input/output facilities of the language.

Other command-line interfaces

The command line provides an interface between programs as well as the user. In this sense, a command line is an alternative to a dialog box. Editors and databases present a command line, in which alternate command processors might run. On the other hand, one might have options on the command line, which opens a dialog box. The latest version of 'Take Command' has this feature. DBase used a dialog box to construct command lines, which could be further edited before use.

Programs like BASIC, diskpart, Edlin, and QBASIC all provide command-line interfaces, some of which use the system shell. Basic is modeled on the default interface for 8-bit Intel computers. Calculators can be run as command-line or dialog interfaces.

Emacs provides a command-line interface in the form of its minibuffer. Commands and arguments can be entered using Emacs standard text editing support, and output is displayed in another buffer.

There are a number of text mode games, like *Adventure* or *King's Quest 1-3*, which relied on the user typing commands at the bottom of the screen. One controls the character by typing commands like 'get ring' or 'look'. The program returns a text which describes how the character sees it, or makes the action happen. The text adventure *The Hitchhiker's Guide to the Galaxy*, a piece of interactive fiction based on Douglas Adam's book of the same name, is a teletype-style command-line game.

The most notable of these interfaces is the standard streams interface, which allows the output of one command to be passed to the input of another. Text files can serve either purpose as well. This provides the interfaces of piping, filters and redirection. Under Unix, devices are files too, so the normal type of file for the shell used for stdin,stdout and stderr is a tty device file.

Another command-line interface allows a shell program to launch helper programs, either to launch documents or start a program. The command is processed internally by the shell, and then passed on to another program to launch the document. The graphical interface of Windows and OS/2 rely heavily on command-lines passed through to other programs – console or graphical, which then usually process the command line without presenting a user-console.

Programs like the OS/2 E editor and some other IBM editors, can process command-lines normally meant for the shell, the output being placed directly in the document window.

A web browser's URL input field can be used as a command line. It can be used to "launch" web apps, access browser configuration, as well as perform a search. Google, which has been called "the command line of the internet" will perform a domain-specific search when it detects search parameters in a known format.[35] This functionality is present whether the search is triggered from a browser field or on Google's website.

Many video games on the PC feature a command line interface often referred to as a console. It is typically used by the game developers during development and by mod developers for debugging purposes as well as for cheating or skipping parts of the game.

See also

- Comparison of command shells
- List of command-line interpreters
- Batch processing

- Batch file
- Console application
- Interpreter directive
- Read-eval-print loop
- Shell (computing)
- Scripting language
- Shell script
- clig
- Computer terminal
- Terminal emulator
- Run command
- Graphical user interface § Comparison to other interfaces
- In the Beginning… Was the Command Line

Notes

1. An example is the comprehensive internal help system of the DR-DOS 7.03 DEBUG command, which can be invoked via `??` at the debug prompt (rather than only the default `?` overview). Specific help pages can be selected via `?n` (where `n` is the number of the page). Additionally, help for specific commands can be displayed by specifying the command name after `?`, f.e. `?D` will invoke help for the various dump commands (like `D` etc.). Some of these features were already supported by the DR DOS 3.41 SID86 and GEMSID.
2. Conventions for describing commands on DOS-like operating systems. Notable difference: The Windows Server 2003 R2 documentation uses italic letters for "Information that the user must supply", while the Server 2008 documentation uses angle brackets. Italics can not be displayed by the internal "help" command while there is no problem with angle brackets.
3. With the exception of ASCII art.
4. Via *Finder, Applications, Utilities, Terminal*.

References

1. "Unix Shells" (http://linuxfinances.info/info/unixshells.html). "the notion of having a replaceable "command shell" rather than a "monitor" tightly integrated with the OS kernel tends to be attributed to Multics."
2. "The Origin of the Shell" (http://www.multicians.org/shell.html). *www.multicians.org*. Retrieved 2017-04-12.
3. Metz, Cade (2013-01-03). "Say Bonjour to the Internet's Long-Lost French Uncle" (https://www.wired.com/2013/01/louis_pouzin_internet_hall/). *Wired*. Retrieved 2017-07-31.
4. Mazières, David (Fall 2004). "MULTICS - The First Seven Years" (http://www.scs.stanford.edu/nyu/04fa/notes/l4d.txt). *Advanced Operating Systems*. Stanford Computer Science Department. Retrieved 2017-08-01.
5. Jones, M. (2011-12-06). "Evolution of shells in Linux" (https://www.ibm.com/developerworks/library/l-linux-shells/). *developerWorks*. IBM. Retrieved 2017-08-01.
6. "GNU BASH Reference" (https://www.gnu.org/software/bash/manual/bashref.html#Shell-Commands).
7. "Microsoft Windows Command Shell Overview" (https://technet.microsoft.com/en-us/library/bb490954.aspx).

8. *SID Users Guide* (http://www.cpm.z80.de/randyfiles/DRI/SID_ZSID.pdf) (PDF). Digital Research. 1978. 595-2549. Archived (https://web.archive.org/web/20191020124044/http://www.cpm.z80.de/randyfiles/DRI/SID_ZSID.pdf) (PDF) from the original on 2019-10-20. Retrieved 2020-02-06. (4+69 pages)
9. *SID-86 User's Guide for CP/M-86* (http://www.cpm.z80.de/manuals/SID86_User_Guide.txt) (2 ed.). Digital Research. August 1982 [March 1982]. SID86UG.WS4. Archived (https://web.archive.org/web/20191020123025/http://www.cpm.z80.de/manuals/SID86_User_Guide.txt) from the original on 2019-10-20. Retrieved 2020-02-06. [1] (https://web.archive.org/web/20200208055456/https://archive.computerhistory.org/resources/access/text/2016/12/102762507-05-01-acc.pdf) (NB. A retyped version of the manual by Emmanuel Roche with Q, SR, and Z commands added.)
10. Paul, Matthias R. (1997-07-30). *NWDOS-TIPs — Tips & Tricks rund um Novell DOS 7, mit Blick auf undokumentierte Details, Bugs und Workarounds* (http://www.antonis.de/dos/dos-tuts/mpdostip/html/nwdostip.htm). *MPDOSTIP*. Release 157 (in German) (3 ed.). Archived (https://web.archive.org/web/20170910194752/http://www.antonis.de/dos/dos-tuts/mpdostip/html/nwdostip.htm) from the original on 2017-09-10. Retrieved 2014-09-06. (NB. NWDOSTIP.TXT is a comprehensive work on Novell DOS 7 and OpenDOS 7.01, including the description of many undocumented features and internals. It is part of the author's yet larger MPDOSTIP.ZIP collection maintained up to 2001 and distributed on many sites at the time. The provided link points to a HTML-converted older version of the NWDOSTIP.TXT file.)
11. Parker, Steve (2011). "Chapter 11: Choosing and using shells". *Shell Scripting: Expert Recipes for Linux, Bash and more* (https://books.google.com/books?id=wWjqCF9HLfYC). Programmer to programmer. Indianapolis, USA: John Wiley & Sons. p. 262. ISBN 978-111816632-1. Retrieved 2017-03-23. "The shell has four different command prompts, called PS1, P52, P53, and PS4. PS stands for Prompt String."
12. *RISC OS 3 User Guide* (http://www.4corn.co.uk/_archive/docs/RISC%20OS%203%20User%20Guide%20(3.0)-opt.pdf) (PDF). Acorn Computers Limited. 1992-03-01. p. 125.
13. Brothers, Hardin; Rawson, Tom; Conn, Rex C.; Paul, Matthias R.; Dye, Charles E.; Georgiev, Luchezar I. (2002-02-27). *4DOS 8.00 online help*.
14. Paul, Matthias R. (1998-01-09). *DELTREE.BAT R1.01 Extended file and directory delete* (https://web.archive.org/web/20190408145354/http://www.lookas.net/ftp/incoming/darbui/Justas/DRDOS/DELTREE.BAT). Caldera, Inc. Archived from the original (http://www.lookas.net/ftp/incoming/darbui/Justas/DRDOS/DELTREE.BAT) on 2019-04-08. Retrieved 2019-04-08.
15. *DR-DOS 7.03 WHATSNEW.TXT — Changes from DR-DOS 7.02 to DR-DOS 7.03* (https://web.archive.org/web/20190408142232/http://www.lookas.net/ftp/incoming/darbui/Justas/DRDOS/WHATSNEW.TXT). Caldera, Inc. 1998-12-24. Archived from the original (http://www.lookas.net/ftp/incoming/darbui/Justas/DRDOS/WHATSNEW.TXT) on 2019-04-08. Retrieved 2019-04-08.
16. Paul, Matthias R. (2002-05-13). "[fd-dev] mkeyb" (https://marc.info/?l=freedos-dev&m=102133580113139&w=2). *freedos-dev*. Archived (https://archive.today/20180910213410/https://marc.info/?l=freedos-dev&m=102133580113139&w=2) from the original on 2018-09-10. Retrieved 2018-09-10. "[…] CPI /H […] CPI [@] [@] [/?|/Help[:topic]] [/!|/About] […] [?|&] […] /?, /Help Display this help screen or specific help for a topic (+) […] /!, /About Display the 'About' info screen […] /Cpifile (+) .CPI/.CP file name <EGA.CPI>; extension: <.CPI>; CPI.EXE=StdIn […] /Report Report file name <"=StdOut>; extension: <.RPT> […] /Style (+) Export <0>-6=BIN-raw/ROM/RAM/PSF0/1/SH/CHED; 7-12/13-18/19-24=ASM-hex/dec/bin/ip/il/p/l/mp/ml […] CPI /H:C […] Overview on codepage file parameter usage: […] CPI /H:S […] Overview on /Style parameters: […] ?, & Online edit mode (prompts for additional parameter input) […]"
17. Paul, Matthias R. (2002-01-09). "SID86" (https://groups.google.com/d/msg/comp.os.cpm/KG4R7ZNvHK8/U5LAkmjcxYgJ). Newsgroup: comp.os.cpm (news:comp.os.cpm). Retrieved 2018-04-08. "[…] Since the DR-DOS 7.03 DEBUG is still based on the old SID86.EXE, I suggest to run DEBUG 1.51 and enter the extended help system with ?? from the debug prompt. This will give you eight screens full of syntax and feature help. Some of these features were also supported by older issues. […]"

18. Paul, Matthias R.; Frinke, Axel C. (2006-01-16). *FreeKEYB - Advanced international DOS keyboard and console driver* (User Manual) (v7 preliminary ed.).
19. *CCI Multiuser DOS 7.22 GOLD Online Documentation*. Concurrent Controls, Inc. (CCI). 1997-02-10. HELP.HLP. (NB. The symbolic instruction debugger SID86 provides a short help screen on ? and comprehensive help on ??.)
20. Paul, Matthias R. (1997-05-24) [1991]. *DRDOSTIP.TXT — Tips und Tricks für DR DOS 3.41 - 5.0* (http://www.antonis.de/dos/dos-tuts/mpdostip/html/drdostip.htm). *MPDOSTIP* (in German) (47 ed.). Archived (https://web.archive.org/web/20161107125452/http://www.antonis.de/dos/dos-tuts/mpdostip/html/drdostip.htm) from the original on 2016-11-07. Retrieved 2016-11-07.
21. "The Open Group Base Specifications Issue 7, Chapter 12.1 Utility Argument Syntax" (http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap12.html). The Open Group. 2008. Retrieved 2013-04-07.`man-pages(7) (https://linux.die.net/man/7/man-pages)` – Linux Conventions and Miscellany Manual (NB. Conventions for describing commands on Unix-like operating systems.)
22. "Command shell overview" (https://technet.microsoft.com/en-us/library/cc737438.aspx). *Windows Server 2003 Product Help*. Microsoft. 2005-01-21. Retrieved 2013-04-07.
23. "Command-Line Syntax Key" (https://technet.microsoft.com/en-us/library/cc771080.aspx). *Windows Server 2008 R2 TechNet Library*. Microsoft. 2010-01-25. Retrieved 2013-04-07.
24. Kernighan, Brian W.; Pike, Rob (1984). *The UNIX Programming Environment* (https://archive.org/details/unixprogramminge0000kern). Englewood Cliffs: Prentice-Hall. ISBN 0-13-937699-2.
25. Pouzin, Louis. "The Origin of the Shell" (http://www.multicians.org/shell.html). *Multicians.org*. Retrieved 2013-09-22.
26. "Remembering Windows 95's launch 15 years later" (http://betanews.com/2010/08/24/remembering-windows-95-s-launch-15-years-later/).
27. "A history of Windows" (https://web.archive.org/web/20150301073112/http://windows.microsoft.com/en-US/windows/history). *windows.microsoft.com*. Archived from the original (http://windows.microsoft.com/en-US/windows/history#T1=era0) on 2015-03-01.
28. "Windows POSIX shell compatibility" (https://technet.microsoft.com/en-us/library/cc754351.aspx).
29. "master - platform/external/mksh - Git at Google" (https://android.googlesource.com/platform/external/mksh/+/master). *android.googlesource.com*. Retrieved 2018-03-18.
30. "Android adb shell - ash or ksh?" (https://stackoverflow.com/questions/11950131/android-adb-shell-ash-or-ksh). *stackoverflow.com*. Retrieved 2018-03-14.
31. "Android sh source" (https://archive.today/20121217165813/https://github.com/android/platform_system_core/tree/master/sh). Archived from the original (https://github.com/android/platform_system_core/tree/master/sh) on 2012-12-17.
32. "Android toolbox source" (https://github.com/android/platform_system_core/tree/master/toolbox).
33. "Cisco IOS Configuration Fundamentals Configuration Guide, Release 12.2" (http://www.cisco.com/c/en/us/td/docs/ios/12_2/configfun/configuration/guide/ffun_c/fcf001.html). Cisco. 2013-10-30. Using the Command-Line Interface. "The Cisco IOS command-line interface (CLI) is the primary user interface…"
34. "Command-Line Interface Overview" (http://www.juniper.net/techpubs/software/junos/junos55/swconfig55-getting-started/html/cli-overview-getting-started.html). *www.juniper.net*. Retrieved 2018-03-14.
35. "Google strange goodness" (http://www.knaster.com/2004/04/google_strange_.html).

External links

- The Roots of DOS (http://www.patersontech.com/Dos/Softalk/Softalk.html) David Hunter, *Softalk for the IBM Personal Computer* March 1983. Archived at Patersontech.com since 2000 (https://web.archive.org/web/20001003150623/http://www.patersontech.com/Dos/Softalk/Softalk.html).
- Command-Line Reference (https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands): Microsoft TechNet Database "Command-Line Reference"