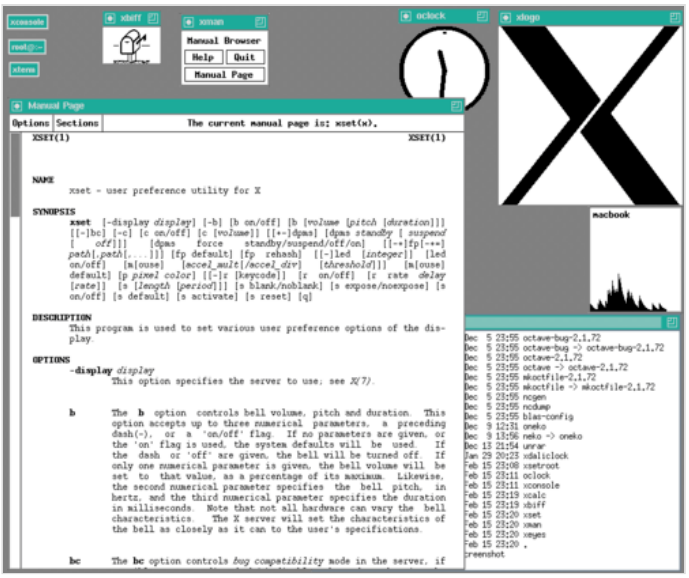
Shell (computing)

An operating system **shell** is a computer program that provides relatively broad and direct access to the system on which it runs. The term *shell* refers to how it is a relatively thin layer around an operating system.[1][2]

A shell is generally a command-line interface (CLI) program although some graphical user interface (GUI) programs are arguably classified as shells too.



A graphical interface similar to one from the late 1980s, which features a graphical window for a man page, a shaped window (oclock) as well as several iconified windows. In the lower right we can see a terminal emulator running a Unix shell, in which the user can type commands as if they were sitting at a terminal.

Overview

Operating systems provide various services to their users, including file management, process management (running and terminating applications), batch processing, and operating system monitoring and configuration.

Most operating system shells are not *direct* interfaces to the underlying kernel, even if a shell communicates with the user via peripheral devices attached to the computer directly. Shells are actually special applications that use the kernel API in just the same way as it is used by other application programs. A shell manages the user–system interaction by prompting users for input, interpreting their input, and then handling output from the underlying operating system (much like a read–eval–print loop, REPL).[3] Since the operating system shell is actually an application, it may easily be replaced with another similar application, for most operating systems.

In addition to shells running on local systems, there are different ways to make remote systems available to local users; such approaches are usually referred to as remote access or remote administration. Initially available on multi-user mainframes, which provided text-based UIs for each active user *simultaneously* by means of a text terminal connected to the mainframe via serial line or modem, remote access has extended to Unix-like systems and Microsoft Windows. On Unix-like systems, Secure Shell protocol (SSH) is usually used for text-based shells, while SSH tunneling can be used for X Window System–based graphical user interfaces (GUIs). On Microsoft Windows, Remote Desktop Protocol can be used to provide GUI remote access, since Windows Vista, PowerShell Remote, since Windows 10 build 1809 SSH[4] can also be used for text-based remote access via WMI, RPC, and WS-Management.[5]

Most operating system shells fall into one of two categories – command-line and graphical. Command-line shells provide a command-line interface (CLI) to the operating system, while graphical shells provide a graphical user interface (GUI). Other possibilities, although not so common, include a voice user

interface and various implementations of a text-based user interface (TUI) that are not CLI, such as text-based menu systems. The relative merits of CLI- and GUI-based shells are often debated. Many computer users use both depending on the task to be performed.

History

Early interactive systems provided a simple command-line interpreter as part of the resident monitor. This interpreter might be called by different names, such as `COMCON` on DEC TOPS-10 systems.[6] The interpreter would execute one of a number of predefined commands, one of which would be to run a user program. Common commands would log the user on and off the system, allocate, free, and manipulate devices and files, and query various pieces of information about the system or a user process.[7]

In 1964, for the Multics operating system, Louis Pouzin conceived the idea of "using commands somehow like a programming language," and coined the term *shell* to describe it.[9] In a 1965 document, the shell is defined as "a common procedure called automatically by the supervisor whenever a user types in some message at his console, at a time when he has no other process in active execution under console control. This procedure acts as an interface between console messages and subroutine [in the supervisor]."[10] This system was first implemented by Glenda Schroeder and an unnamed man from General Electric.[11]

> The purpose of such a procedure is to create a medium of exchange into which one could activate any procedure, *as if it were called from the inside of another program*. Hereafter, for simplification, we shall refer to that procedure as the "SHELL".
>
> Louis Pouzin, The SHELL: A Global Tool for Calling and Chaining Procedures in the System [8]

Multics also introduced the *active function*, a key concept in all later shells. This is defined as

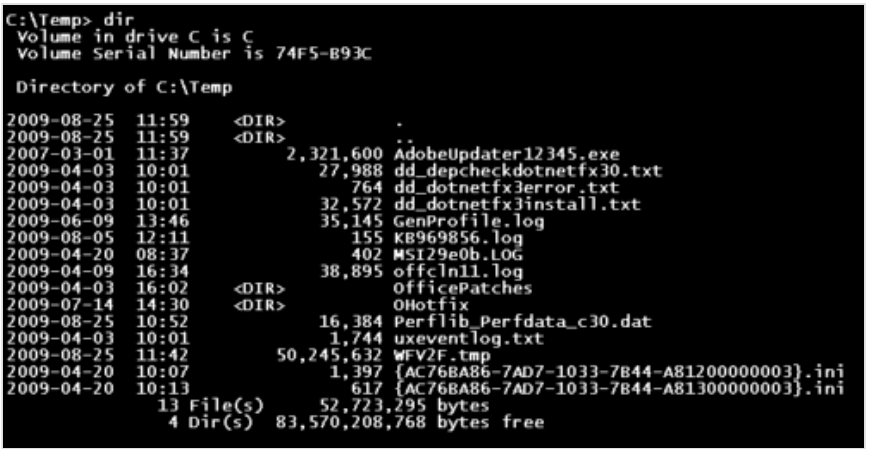
> a string... which is replaced by a character string return value before the command line containing it is executed. Active functions are often used... to implement command-language macros.[12]

In 1971, Ken Thompson developed the Thompson shell in the first version of Unix. While simpler than the Multics shell, it contained some innovative features, which have been carried forward in modern shells, including the use of < and > for input and output redirection.

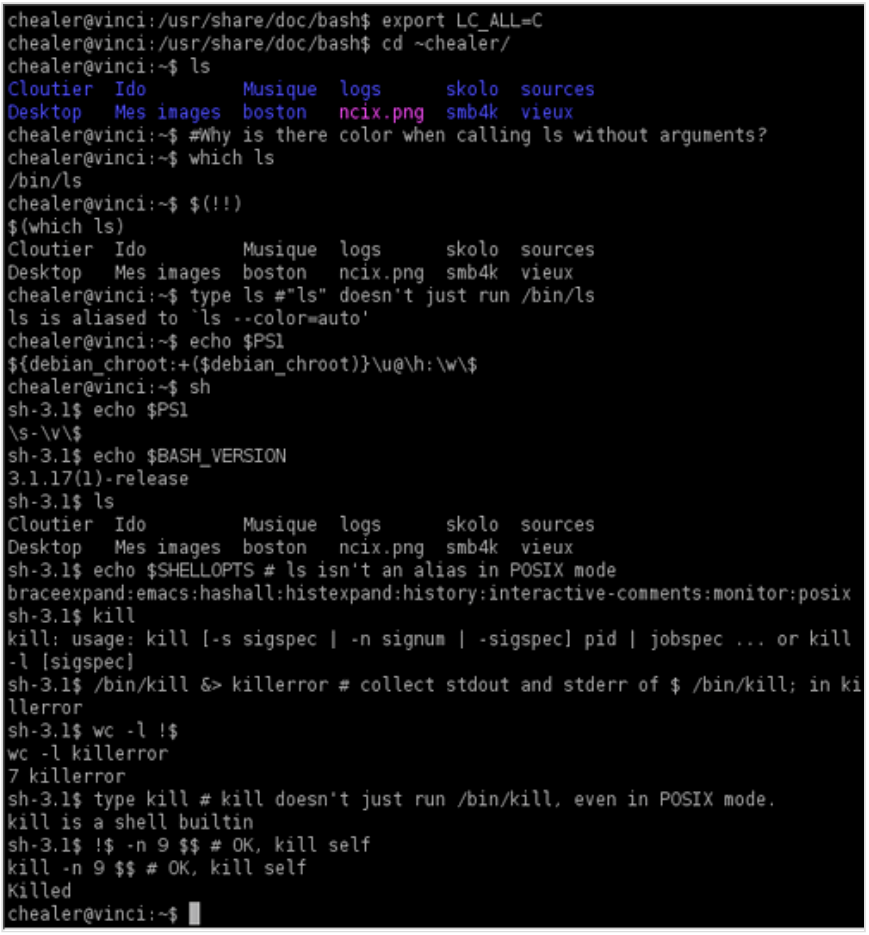
The graphical shell first appeared in Douglas Engelbart's NLS system, demonstrated in December, 1968 at the Fall Joint Computer Conference in San Francisco, in what has been called *The Mother of All Demos*. Engelbart's colleagues at Stanford Research Institute brought the concept to the Xerox Palo Alto Research Center (PARC), where it appeared on the Alto, introduced in 1973. From there the idea spread to Niklaus Wirth's Lilith in 1980, and the Apple Lisa in 1983, then became ubiquitous.

Command-line shells

A command-line interface (CLI) is an operating system shell that uses alphanumeric characters typed on a keyboard to provide instructions and data to the operating system, interactively. For example, a teletypewriter can send codes representing keystrokes to a command interpreter program running on the computer; the command interpreter parses the sequence of keystrokes and responds with an error message if it cannot recognize the sequence of characters, or it may carry out some other program action such as loading an application program, listing files, logging in a user and many others. Operating systems such as UNIX have a large variety of shell programs with different commands, syntax and capabilities, with the POSIX shell being a baseline. Some operating systems had only a single style of command interface; commodity operating systems such as MS-DOS came with a standard command interface (COMMAND.COM) but third-party interfaces were also often available, providing additional features or functions such as menuing or remote program execution.



Command Prompt, a CLI shell in Windows



Bash, a widely adopted Unix shell

Application programs may also implement a command-line interface. For example, in Unix-like systems, the telnet program has a number of commands for controlling a link to a remote computer system. Since the commands to the program are made of the same keystrokes as the data being sent to a remote computer, some means of distinguishing the two are

required. An escape sequence can be defined, using either a special local keystroke that is never passed on but always interpreted by the local system. The program becomes modal, switching between interpreting commands from the keyboard or passing keystrokes on as data to be processed.

A feature of many command-line shells is the ability to save sequences of commands for re-use. A data file can contain sequences of commands which the CLI can be made to follow as if typed in by a user. Special features in the CLI may apply when it is carrying out these stored instructions. Such batch files (script files) can be used repeatedly to automate routine operations such as initializing a set of programs when a system is restarted. Batch mode use of shells usually involves structures, conditionals, variables, and other elements of programming languages; some have the bare essentials needed for such a purpose, others are very sophisticated programming languages in and of themselves. Conversely, some programming languages can be used interactively from an operating system shell or in a purpose-built program.

Several command-line shells, such as Nushell, Xonsh, Bash (Unix shell), and Z shell, offer command-line completion, enabling the interpreter to expand commands based on a few characters input by the user.[13]

A command-line interpreter may offer a history function, so that the user can recall earlier commands issued to the system and repeat them, possibly with some editing. Since all commands to the operating system had to be typed by the user, short command names and compact systems for representing program options were common. Short names were sometimes hard for a user to recall, and early systems lacked the storage resources to provide a detailed on-line user instruction guide.

Graphical shells

A graphical user interface (GUI) provides means for manipulating programs graphically, by allowing for operations such as opening, closing, moving and resizing windows, as well as switching focus between windows. Graphical shells may be included with desktop environments or come separately, even as a set of loosely coupled utilities.

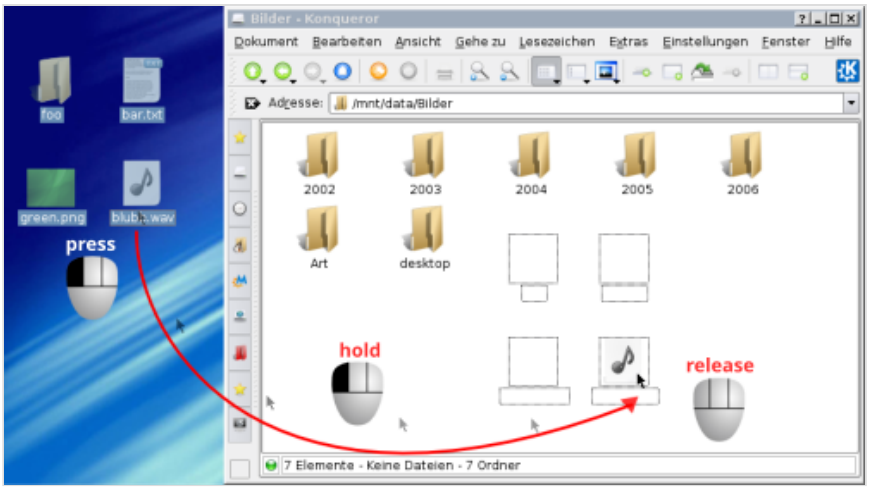
Most graphical user interfaces develop the metaphor of an "electronic desktop", where data files are represented as if they were paper documents on a desk, and application programs similarly have graphical representations instead of being invoked by command names.

Unix-like systems

Graphical shells typically build on top of a windowing system. In the case of X Window System or Wayland, the shell consists of an X window manager or a Wayland compositor, respectively, as well as of one or multiple programs providing the functionality to start installed applications, to manage open windows and virtual desktops, and often to support a widget engine.

In the case of macOS, Quartz Compositor acts as the windowing system, and the shell consists of the Finder,[14] the Dock,[14] SystemUIServer,[14] and Mission Control.[15]

Microsoft Windows



Drag and drop operation performed on a group of files between KDesktop and Konqueror in KDE

Modern versions of the Microsoft Windows operating system use the Windows shell as their shell. Windows Shell provides desktop environment, start menu, and task bar, as well as a graphical user interface for accessing the file management functions of the operating system. Older versions also include Program Manager, which was the shell for the 3.x series of Microsoft Windows, and which in fact shipped with later versions of Windows of both the 95 and NT types at least through Windows XP. The interfaces of Windows versions 1 and 2 were markedly different.

Desktop applications are also considered shells, as long as they use a third-party engine. Likewise, many individuals and developers dissatisfied with the interface of Windows Explorer have developed software that either alters the functioning and appearance of the shell or replaces it entirely. WindowBlinds by StarDock is a good example of the former sort of application. LiteStep and Emerge Desktop are good examples of the latter.

Interoperability programmes and purpose-designed software lets Windows users use equivalents of many of the various Unix-based GUIs discussed below, as well as Macintosh. An equivalent of the OS/2 Presentation Manager for version 3.0 can run some OS/2 programmes under some conditions using the OS/2 environmental subsystem in versions of Windows NT.

Invocation from a program

A shell can usually also be invoked from within a program through standard system functions like `system()`, `popen()` and `exec()` without any user interface being involved.

Beyond these standard functions, Windows for example provides the "Windows Shell API" which exposes a set of functions that programs can use to interact with the Windows shell. These shell functions are provided in DLL's (like shell32.dll and shlwapi.dll) and do not need the default Windows shell GUI i.e. explorer.exe to be running. Explorer.exe and its replacements generally make use of the Windows Shell API to provide users with a Windows shell GUI without having to implement themselves the Windows shell core functionalities (which are implemented and provided by shell32.dll and shlwapi.dll). But programs can use the Windows Shell API for other purposes and without providing users with a shell user interface.

Other uses

"Shell" is also used loosely to describe application software that is "built around" a particular component, such as web browsers and email clients, in analogy to the shells found in nature. Indeed, the (command-line) shell encapsulates the operating system *kernel*. These are also sometimes referred to as "wrappers".[2]

In expert systems, a shell is a piece of software that is an "empty" expert system without the knowledge base for any particular application.[16]

See also

- Comparison of command shells
- Human–computer interaction
- Internet Explorer shell – Programs based on the Internet Explorer browser engine
- Shell account – User account on a remote server
- Shell builtin – Computer function
- Superuser – Special user account used for system administration
- Unix shell – Command-line interpreter for Unix operating system
- Window manager – Type of system software
- Read–eval–print loop – Computer programming environment

References

1. "The Internet's fifth man" (https://www.economist.com/news/technology-quarterly/21590765-louis-pouzin-helped-create-internet-now-he-campaigning-ensure-its), Brain scan, *The Economist*, London: Economist Group, December 13, 2013, "Mr Pouzin created a program called RUNCOM that helped users automate tedious and repetitive commands. That program, which he described as a "shell" around the computer's whirring innards, gave inspiration—and a name—to an entire class of software tools, called command-line shells, that still lurk below the surface of modern operating systems."
2. Raymond, Eric S. (ed.). "shell" (http://www.catb.org/jargon/html/S/shell.html). *The Jargon File*.
3. "Operating system shells" (http://pic.dhe.ibm.com/infocenter/aix/v6r1/index.jsp?topic=%2Fcom.ibm.aix.baseadmn%2Fdoc%2Fbaseadmndita%2Fshells.htm). *AIX 6.1 Information Center*. IBM Corp. Retrieved September 16, 2012.
4. teocci. "How to SSH into Windows 10 or 11?" (https://gist.github.com/teocci/5a96568ab9bf93a592d7a1a237ebb6ea). *GitHub Gist*. Retrieved 2025-01-24.
5. Wheeler, Sean (14 October 2018). "Running Remote Commands" (https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/running-remote-commands?view=powershell-6). *Microsoft Docs*. Microsoft. Retrieved 30 June 2019. "You can run commands on one or hundreds of computers with a single PowerShell command. Windows PowerShell supports remote computing by using various technologies, including WMI, RPC, and WS-Management."
6. Digital Equipment Corporation (Nov 1980). *TOPS-10 MONITOR INTERNALS* (http://bitsavers.org/pdf/dec/pdp10/TOPS10_monitorInternalsCourse/EY-CD150-HO-006_monInternal.pdf) (PDF). pp. CMND-1 – CMND-16. Retrieved Mar 29, 2022.

7. Digital Equipment Corporation (Aug 1977). *DECSystem 10 Operating System Commands Manual* (http://bitsavers.org/pdf/dec/pdp10/TOPS10/AA-0916C-TB_DEC10_Operating_Systems_Command_Manual_Ver_6_03_Aug77.pdf) (PDF). Retrieved Mar 29, 2022.
8. Poizin, Louis. "The SHELL: A Global Tool for Calling and Chaining Procedures in the System" (https://people.csail.mit.edu/saltzer/Multics/Multics-Documents/MDN/MDN-4.pdf) (PDF).
9. Pouzin, Louis. "The Origin of the Shell" (https://multicians.org/shell.html). *multicians.org*. Retrieved Mar 29, 2022.
10. Pouzin, Louis. "The SHELL: A Global Tool for Calling and Chaining Procedures in the System" (https://people.csail.mit.edu/saltzer/Multics/Multics-Documents/MDN/MDN-4.pdf) (PDF). *MIT.edu*. Retrieved Mar 29, 2022.
11. Pouzin, Louis. "The Origin of the Shell" (https://multicians.org/shell.html). *multicians.org*. Retrieved Feb 12, 2024.
12. Honeywell, inc. (Feb 1983). *Multics Common Commands* (http://bitsavers.org/pdf/honeywell/multics/GB58-0_commonCmds_Feb83.pdf) (PDF). pp. 1-1 – 1-2. Retrieved Mar 29, 2022.
13. Xonsh Official Website (https://xon.sh/)
14. "The Life Cycle of a Daemon" (https://developer.apple.com/library/mac/documentation/macosx/conceptual/bpsystemstartup/chapters/Lifecycle.html). Apple Inc.
15. "Restart Mission Control in OS X Lion" (http://osxdaily.com/2011/11/23/restart-mission-control-in-os-x-lion/). OSXDaily. Nov 23, 2011.
16. *British Computer Society: The BCS glossary of ICT and computing terms* (https://books.google.com/books?id=g8Bds8ssYYgC&dq=%22shell+is+a+piece%22+%22expert+system%22&pg=PA135). Pearson Education. 2005. p. 135. ISBN 978-0-13-147957-9.
