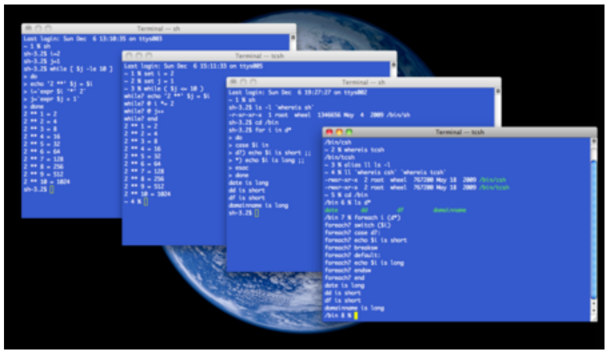
Wikipedia
The Free Encyclopedia

Unix shell

A **Unix shell** is a command-line interpreter or shell that provides a command line user interface for Unix-like operating systems. The shell is both an interactive command language and a scripting language, and is used by the operating system to control the execution of the system using shell scripts.[2]



```
Terminal — sh
Last login: Sun Dec  6 13:10:35 on ttys003
~ 1 % sh
sh-3.2$ i=2
sh-3.2$ j=1
sh-3.2$ while [ $j -le 10 ]
> do
> echo '2 **' $j = $i
> i=`expr $i '*' 2`
> j=`expr $j + 1`
> done
2 ** 1 = 2
2 ** 2 = 4
2 ** 3 = 8
2 ** 4 = 16
2 ** 5 = 32
2 ** 6 = 64
2 ** 7 = 128
2 ** 8 = 256
2 ** 9 = 512
2 ** 10 = 1024
sh-3.2$
```

```
Terminal — tcsh
Last login: Sun Dec  6 15:11:33 on ttys005
~ 1 % set i = 2
~ 2 % set j = 1
~ 3 % while ( $j <= 10 )
while? echo '2 **' $j = $i
while? @ i *= 2
while? @ j++
while? end
2 ** 1 = 2
2 ** 2 = 4
2 ** 3 = 8
2 ** 4 = 16
2 ** 5 = 32
2 ** 6 = 64
2 ** 7 = 128
2 ** 8 = 256
2 ** 9 = 512
2 ** 10 = 1024
~ 4 %
```

```
Terminal — sh
Last login: Sun Dec  6 19:27:27 on ttys002
~ 1 % sh
sh-3.2$ ls -l `whereis sh`
-r-xr-xr-x  1 root  wheel  1346656 May  4  2009 /bin/sh
sh-3.2$ cd /bin
sh-3.2$ for i in d*
> do
> case $i in
> d?) echo $i is short ;;
> *) echo $i is long ;;
> esac
> done
date is long
dd is short
df is short
domainname is long
sh-3.2$
```

```
Terminal — tcsh
/bin/csh
~ 2 % whereis tcsh
/bin/tcsh
~ 3 % alias ll ls -l
~ 4 % ll `whereis csh` `whereis tcsh`
-rwxr-xr-x  2 root  wheel  767200 May 18  2009 /bin/csh
-rwxr-xr-x  2 root  wheel  767200 May 18  2009 /bin/tcsh
~ 5 % cd /bin
/bin 6 % ls d*
date        dd          df          domainname
/bin 7 % foreach i (d*)
foreach? switch ($i)
foreach? case d?:
foreach? echo $i is short
foreach? breaksw
foreach? default:
foreach? echo $i is long
foreach? endsw
foreach? end
date is long
dd is short
df is short
domainname is long
/bin 8 %
```

tcsh and sh shell windows on a Mac OS X Leopard[1] desktop

Users typically interact with a Unix shell using a terminal emulator; however, direct operation via serial hardware connections or Secure Shell are common for server systems. All Unix shells provide filename wildcarding, piping, here documents, command substitution, variables and control structures for condition-testing and iteration.

Concept

Generally, a *shell* is a program that executes other programs in response to text commands. A sophisticated shell can also change the environment in which other programs execute by passing named variables, a parameter list, or an input source.

In Unix-like operating systems, users typically have many choices of command-line interpreters for interactive sessions. When a user logs into the system interactively, a shell program is automatically executed for the duration of the session. The type of shell, which may be customized for each user, is typically stored in the user's profile, for example in the local `passwd` file or in a distributed configuration system such as NIS or LDAP; however, the user may execute any other available shell interactively.

On operating systems with a windowing system, such as macOS and desktop Linux distributions, some users may never use the shell directly. On Unix systems, the shell has historically been the implementation language of system startup scripts, including the program that starts a windowing system, configures networking, and many other essential functions. However, some system vendors have replaced the traditional shell-based startup system (init) with different approaches, such as systemd.

Early shells

The first Unix shell was the Thompson shell, *sh*, written by Ken Thompson at Bell Labs and distributed with Versions 1 through 6 of Unix, from 1971 to 1975.[3] Though rudimentary by modern standards, it introduced many of the basic features common to all later Unix shells, including piping, simple control structures using `if` and `goto`, and filename wildcarding. Though not in current use, it is still available as part of some Ancient UNIX systems.

It was modeled after the Multics shell, developed in 1965 by American software engineer Glenda Schroeder. Schroeder's Multics shell was itself modeled after the RUNCOM program Louis Pouzin showed to the Multics Team. The "rc" suffix on some Unix configuration files (for example, ".vimrc"), is a remnant of the RUNCOM ancestry of Unix shells.[1][4]

The PWB shell or Mashey shell, *sh*, was an upward-compatible version of the Thompson shell, augmented by John Mashey and others and distributed with the Programmer's Workbench UNIX, circa 1975–1977. It focused on making shell programming practical, especially in large shared computing centers. It added shell variables (precursors of environment variables, including the search path mechanism that evolved into $PATH), user-executable shell scripts, and interrupt-handling. Control structures were extended from if/goto to if/then/else/endif, switch/breaksw/endsw, and while/end/break/continue. As shell programming became widespread, these external commands were incorporated into the shell itself for performance.

But the most widely distributed and influential of the early Unix shells were the Bourne shell and the C shell. Both shells have been used as the coding base and model for many derivative and work-alike shells with extended feature sets.[5]

Bourne shell

The Bourne shell, *sh*, was a new Unix shell by Stephen Bourne at Bell Labs.[6] Distributed as the shell for UNIX Version 7 in 1979, it introduced the rest of the basic features considered common to all the later Unix shells, including here documents, command substitution, more generic variables and more extensive builtin control structures. The language, including the use of a reversed keyword to mark the end of a block, was influenced by ALGOL 68.[7] Traditionally, the Bourne shell program name is `sh` and its path in the Unix file system hierarchy is `/bin/sh`. But a number of compatible work-alikes are also available with various improvements and additional features. On many systems, sh may be a symbolic link or hard link to one of these alternatives:

- Almquist shell (ash): written as a BSD-licensed replacement for the Bourne Shell; often used in resource-constrained environments. The sh of FreeBSD, NetBSD (and their derivatives) are based on ash that has been enhanced to be POSIX conformant.
  - Busybox: a set of Unix utilities for small and embedded systems, which includes 2 shells: ash, a derivative of the Almquist shell; and hush, an independent implementation of a Bourne shell.
  - Debian Almquist shell (dash): a modern replacement for ash in Debian and Ubuntu
- Bourne-Again shell (bash): written as part of the GNU Project to provide a superset of Bourne Shell functionality. This shell can be found installed and is the default interactive shell for users on most Linux systems.
- KornShell (ksh): written by David Korn based on the Bourne shell sources[8] while working at Bell Labs
- Public domain Korn shell (pdksh)
  - MirBSD Korn shell (mksh): a descendant of the OpenBSD /bin/ksh and pdksh, developed as part of MirOS BSD
- Z shell (zsh): a relatively modern shell that is backward compatible with bash. It's the default shell in Kali Linux since 2020.4 and macOS since 10.15 Catalina.

The POSIX standard specifies its standard shell as a strict subset of the Korn shell, an enhanced version of the Bourne shell. From a user's perspective the Bourne shell was immediately recognized when active by its characteristic default command line prompt character, the dollar sign ($).

C shell

The C shell, *csh*, was modeled on the C programming language, including the control structures and the expression grammar. It was written by Bill Joy as a graduate student at University of California, Berkeley, and was widely distributed with BSD Unix.[9]

The C shell also introduced many features for interactive work, including the history and editing mechanisms, aliases, directory stacks, tilde notation, cdpath, job control and path hashing. On many systems, csh may be a symbolic link or hard link to TENEX C shell (tcsh), an improved version of Joy's original version. Although the interactive features of csh have been copied to most other shells, the language structure has not been widely copied. The only work-alike is Hamilton C shell, written by Nicole Hamilton, first distributed on OS/2 in 1988 and on Windows since 1992.[10]

Configuration files

Shells read configuration files in various circumstances. These files usually contain commands for the shell and are executed when loaded; they are usually used to set important variables used to find executables, like $PATH, and others that control the behavior and appearance of the shell. The table in this section shows the configuration files for popular shells.[11]

Explanation:

- blank means a file is not read by a shell at all.
- "yes" means a file is always read by a shell upon startup.
- "login" means a file is read if the shell is a login shell.
- "n/login" means a file is read if the shell is not a login shell.
- "int." means a file is read if the shell is interactive.

Configuration file	sh	ksh	csh	tcsh	bash	zsh
`/etc/.login`			login	login		
`/etc/csh.cshrc`			yes	yes		
`/etc/csh.login`			login	login		
`~/.tcshrc`				yes		
`~/.cshrc`			yes	yes[a]		
`~/etc/ksh.kshrc`		int.				
`/etc/sh.shrc`	int.[b]					
`$ENV` (typically `~/.kshrc`)[12]	int.[c][d]	int.			int.[e]	
`~/.login`			login	login		
`~/.logout`			login	login		
`/etc/profile`	login	login			login	login[f]
`~/.profile`	login	login			login[g]	login[f]
`~/.bash_profile`					login[g]	
`~/.bash_login`					login[g]	
`~/.bash_logout`					login	
`~/.bashrc`					int.+n/login	
`/etc/zshenv`						yes
`/etc/zprofile`						login
`/etc/zshrc`						int.
`/etc/zlogin`						login
`/etc/zlogout`						login
`~/.zshenv`						yes
`~/.zprofile`						login
`~/.zshrc`						int.
`~/.zlogin`						login

a. only if `~/.tcshrc` not found
b. Newer versions of the Bourne Shell only
c. Available on systems that support the "User Portability Utilities option"; value of the variable must be an *absolute* path, and it is ignored "if the user's real and effective user IDs or real and effective group IDs are different."[13]
d. `$ENV` is `$HOME/.shrc` in newer versions of the Bourne Shell
e. Same behavior as `sh`, but only if invoked as `sh` (bash 2+) or, since bash 4.2, also if invoked *explicitly* in POSIX compatibility mode (with options `--posix` or `-o posix`).[14]
f. Only in sh/ksh compatibility mode (when invoked as bash, sh, ksh)

g. The first readable file in order of `~/.bash_profile`, `~/.bash_login` and `~/.profile`; and only `~/.profile` if invoked as `sh` or, as of at least Bash 4.2, if invoked *explicitly* in POSIX compatibility mode (with options `--posix` or `-o posix`)

Other shells

Variations on the Unix shell concept that don't derive from Bourne shell or C shell include the following:[15]

- es – A functional programming rc-compatible shell written in the mid-1990s.
- Friendly interactive shell (fish) – First released in 2005.
- PowerShell – An object-oriented shell developed originally for Windows OS and now available to macOS and Linux.
- Qshell – A shell on the IBM i operating system based on POSIX and X/Open standards.
- rc – The default shell on Plan 9 from Bell Labs and Version 10 Unix written by Tom Duff. Ports have been made to various Unix-like operating systems.
- scsh – A Scheme Shell.
- wish – A windowing shell for Tcl/Tk.

See also

- Comparison of command shells
- List of Unix commands
- Read–eval–print loop
- Restricted shell
- Shell (computing)
- Shell account
- Shell script
- Shell shoveling

References

1. Tom Van Vleck (1995-02-05). "Unix and Multics" (http://www.multicians.org/unix.html). Multicians.org. Retrieved 2012-08-14.
2. Bourne, Stephen R. (October 1983). "The Unix Shell" (https://archive.org/stream/byte-magazine-1983-10/1983_10_BYTE_08-10_UNIX#page/n187/mode/2up). *BYTE*. p. 187. Retrieved 30 January 2015.
3. "V6 Thompson Shell Port - History" (http://v6shell.org/history/). V6shell.org. Retrieved 2012-08-14.
4. Louis Pouzin (2000-11-25). "The Origin of the Shell" (http://www.multicians.org/shell.html). Multicians.org. Retrieved 2012-08-14.
5. Nikolai Bezroukov (2015-08-13). "Introduction to the Unix shell history" (https://web.archive.org/web/20220608181527/www.softpanorama.org/People/Shell_giants/introduction.shtml). Softpanorama. Archived from the original (http://www.softpanorama.org/People/Shell_giants/introduction.shtml) on 2022-06-08. Retrieved 2016-08-21.
6. Bourne, Stephen (2009-03-05). "The A-Z of Programming Languages: Bourne shell, or sh" (https://www2.computerworld.com.au/article/279011/a-z_programming_languages_bourne_shell_sh/) (Interview). Interviewed by Howard Dahdah. Computerworld. Retrieved 2022-08-16.

7. "*Re: Late Bloomers Revisited*" (https://groups.google.com/group/comp.lang.misc/msg/d58db4799c33e093?hl=en&dmode=source). Retrieved 20 September 2014.
8. Korn, David G. (October 26, 1994), "ksh - An Extensible High Level Language" (https://www.usenix.org/legacy/publications/library/proceedings/vhll/full_papers/korn.ksh.a), *Proceedings of the USENIX 1994 Very High Level Languages Symposium*, USENIX Association, retrieved February 5, 2015, "Instead of inventing a new script language, we built a form entry system by modifying the Bourne shell, adding built-in commands as necessary."
9. Harley Hahn, Harley Hahn's Guide to Unix and Linux: Unix/Linux Timeline (https://www.harley.com/unix-book/book/chapters/h.html).
10. "Hamilton C shell for Windows Release Notes 4.0" (http://hamiltonlabs.com/ReleaseNotes.htm). Retrieved 20 September 2014.
11. "Different UNIX Shells" (https://web.archive.org/web/20160403120601/www.unixnote.com/2010/05/different-unix-shell.html). unixnote.com. 2010. Archived from the original (http://www.unixnote.com/2010/05/different-unix-shell.html) on 2016-04-03. Retrieved 2016-08-21.
12. SCO Unix Group, SCO Unixware 7 documentation, 22 Apr 2004, retrieved 18 Oct 2012 (http://unix.harley.com/instructors/timeline.html).
13. "Shell Command Language" (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html#tag_18_05_03). *opengroup.org*. Retrieved 15 June 2015.
14. "Bash Reference Manual: Bash Startup Files" (https://www.gnu.org/software/bash/manual/html_node/Bash-Startup-Files.html). *gnu.org*. Retrieved 15 June 2015.
15. "FreeBSD Ports: Shells" (https://web.archive.org/web/20210112142623/www.freebsd.org/ports/shells.html). Freebsd.org. 2014-03-30. Archived from the original (http://www.freebsd.org/ports/shells.html) on 2021-01-12. Retrieved 2014-04-05.