



Command-line completion

Command-line completion (also **tab completion**) is a common feature of [command-line interpreters](#), in which the program automatically fills in partially typed commands.

Command line interpreters are programs that allow a user to interact with the underlying operating system by typing commands at a command prompt using a command line interface (CLI), in contrast to pointing and clicking a mouse in a Graphical User Interface (GUI). Command-line completion allows the user to type the first few characters of a command,

program, or filename, and press a completion key (normally [Tab](#) ) to fill in the rest of the item. The user then presses [Return](#) or [Enter](#) to run the command or open the file.


Command-line completion is useful in several ways, as illustrated by the animation accompanying this article. Commonly accessed commands, especially ones with long names, require fewer keystrokes to reach. Commands with long or difficult to spell filenames can be entered by typing the first few characters and pressing a completion key, which completes the command or filename. In the case of multiple possible completions, some command-line interpreters, especially Unix shells, will list all possible completions beginning with those few characters. The user can type more characters and press [Tab](#)  again to see a new, narrowed-down list if the typed characters are still ambiguous, or else complete the command/filename with a trailing space. An alternate form of completion rotates through all matching results when the input is ambiguous.

Completable elements may include commands, arguments, file names and other entities, depending on the specific interpreter and its configuration. Command-line completion generally only works in [interactive mode](#). That is, it cannot be invoked to complete partially typed commands in [scripts](#) or [batch files](#), even if the completion is unambiguous. The name **tab completion** comes from the fact that command-line completion is often invoked by pressing the [tab](#) key.



Example of command-line completion in [Bash](#). 



Example of command-line completion in [PowerShell](#) with [Intellisense](#). 

Contents [\[hide\]](#)

- 1 [History](#)
- 2 [Example](#)
 - 2.1 [Prompting completion](#)
 - 2.2 [Rotating completion](#)
- 3 [Completion in different command line interfaces](#)
- 4 [See also](#)
- 5 [References](#)
- 6 [External links](#)
 - 6.1 [Unix shells](#)
 - 6.2 [Windows command interpreters](#)

History [\[edit \]](#)

Tab completion showed up early in computing history; one of the first examples appeared in the [Berkeley Timesharing System](#) for the [SDS 940](#), where if a typed string were ambiguous, the interpreter would do nothing, but if the string was *not* ambiguous, it would automatically complete it without any command from the user. This feature did not work well with the all too frequent [typos](#), and so was a mixed blessing. This feature was imitated by [Tenex](#)'s developers who made an important change: Tenex used "escape recognition", in which the interpreter would not attempt to autocomplete unless the [escape key](#) was struck (thus the name) by the user. The domain was also expanded from only program names on the Berkeley system to both program names and files on Tenex.^[1] The Tenex descendant [TOPS-20](#) moved command line completion from command interpreter to the operating system via the COMND JSYS system call, to make it available to other user applications.^[2] From there it was borrowed by Unix.

Example [\[edit \]](#)

To open the file `introduction-to-command-line-completion.html` with [Firefox](#) one would type:

```
firefox introduction-to-command-line-completion.html
```


This is a long command to type. Instead we can use command-line completion.

Prompting completion [\[edit \]](#)

The following example shows how command-line completion works in [Bash](#). Other command line shells may perform slightly differently.

First we type the first three letters of our command:

```
fir
```

Then we press `Tab`  and because the only command in our system that starts with "fir" is "firefox", it will be completed to:

```
firefox
```

Then we start typing the file name:

```
firefox i
```

But this time `introduction-to-command-line-completion.html` is not the only file in the current directory that starts with "i". The directory also contains files `introduction-to-bash.html` and `introduction-to-firefox.html`. The system can't decide which of these filenames we wanted to type, but it does know that the file must begin with "introduction-to-", so the command will be completed to:

```
firefox introduction-to-
```

Now we type "c":

```
firefox introduction-to-c
```

After pressing `Tab` it will be completed to the whole filename:

```
firefox introduction-to-command-line-completion.html
```

In short we typed:

```
firTab iTab cTab
```

This is just eight keystrokes, which is considerably less than 52 keystrokes we would have needed to type without using command-line completion.

Rotating completion [\[edit\]](#)

The following example shows how command-line completion works with rotating completion, such as Windows's [CMD](#) uses.

We follow the same procedure as for prompting completion until we have:

```
firefox i
```

We press `Tab` once, with the result:

```
firefox introduction-to-bash.html
```

We press `Tab` again, getting:

```
firefox introduction-to-command-line-completion.html
```

In short we typed:

```
firTab iTabTab
```

This is just seven keystrokes, comparable to prompting-style completion. This works best if we know what possibilities the interpreter will rotate through.

Completion in different command line interfaces [\[edit\]](#)

- [Unix shells](#), including [Bash](#) (the default shell in most [Linux distributions](#)) and [ksh](#) among many others, have a long-standing tradition of advanced and customizable completion capabilities.^{[\[3\]](#)}

- [Bash](#) programmable completion, `complete` and `compgen` commands^[4] have been available since the beta version of 2.04^[3] in 2000^[5] and offers at least Pathname and filename completion.
- For KornShell users, file name completion depends on the value of the EDITOR variable. If EDITOR is set to vi, you type part of the name, and then `Escape`, `\`. If EDITOR is set to [Emacs](#), you type part of the name, and then `Escape`, `Escape`.
- The [Z shell](#) (zsh) pioneered the support for fully programmable completion, allowing users to have the shell automatically complete the parameters of various commands unrelated to the shell itself, which is accomplished by priming the shell with definitions of all known switches as well as appropriate parameter types. This allows the user to e.g. type `tar xzf` `Tab` and have the shell complete only tarred gzip archives from the actual filesystem, skipping files which are incompatible with the input parameters. A modern zsh installation comes with completion definitions for over five hundred commands.
- [Tcsh](#) offers default file, command, and variable name completion activated using `Tab`. The 'complete' builtin command provides fully programmable completion. The source code comes with a 'complete.tcsh' file containing many examples of its completion syntax.
- [Windows PowerShell](#), the extensible command shell from Microsoft, which is based on [object-oriented programming](#) and the [Microsoft .NET](#) framework, provides powerful and customizable completion capabilities similar to those of traditional Unix shells.^{[6][7][citation needed]}
- The `cmd.exe` command processor of [Windows NT](#)-based systems supports basic completion. It is possible to use a separate key-binding for matching directory names only.
- `cmd.exe /F:ON` enables file and directory name completion characters (^F and ^D by default). Use `cmd.exe /?` for more information.
- [TweakUI](#) can be used to configure the keys used for file name and directory name completion.^[8]
- The [MS-DOS](#) command processor `COMMAND.COM` did not have command-line completion: pressing the tab key would just advance the [cursor](#). However, various enhanced shells for MS-DOS, such as [4DOS](#), the [FreeDOS](#) version of `COMMAND.COM`, or the Enhanced [DOSKEY.COM](#) feature Unix-style tab completion.
- [Far Manager](#) apart from its file management functions provides [command history](#) and line completion for Windows.

See also [[edit](#)]

- [Autocomplete](#)
- [Command-line interface](#)
- [Comparison of command shells](#)
- [Shell](#)

References [[edit](#)]

- ↑ "Origins and Development of TOPS-20" [↗](#). *www.opost.com*.
- ↑ DECSYSTEM-20 Assembly Language Guide [↗](#)
- ↑ ^{***a b***} "Working more productively with bash 2.x/3.x" [↗](#). *www.caliban.org*.
- ↑ "Bash Reference Manual" [↗](#). *tiswww.case.edu*.
- ↑ "Index of /gnu/bash" [↗](#). *ftp.swin.edu.au*.
- ↑ "The PowerShell Guy" [↗](#). *thepowershellguy.com*.
- ↑ "The PowerShell Guy" [↗](#). *thepowershellguy.com*.
- ↑ "Simon Peyton Jones at Microsoft Research" [↗](#).

External links [\[edit \]](#)

Unix shells [\[edit \]](#)

- [A Bash completion overview - "Working more productively with bash 2.x/3.x" by Ian Macdonald](#)
- [The zsh completion system, chapter from the Z Shell Manual](#)
- [Completion and listing from the TCSH Manual](#)

Windows command interpreters [\[edit \]](#)

(Be sure to check the "Applies to" section in each article)

- Windows Server 2003:
 1. [Directory name completion](#)
 2. [Filename completion](#)
- [Windows XP](#)
- [Windows 2000/NT 4](#)

Categories: [User interface techniques](#) | [Autocomplete](#)