Bash - ArchWiki

Archived: 2026-04-05 20:20:35 UTC

Bash (**B**ourne-**A**gain **SH**ell) is a command-line shell/programming language by the GNU Project. Its name alludes to its predecessor, the long-deprecated Bourne shell. Bash can be run on most Unix-like operating systems, including GNU/Linux.

Bash is the default command-line shell on Arch Linux.

Invocation

Bash behaviour can be altered depending on how it is invoked. Some descriptions of different modes follow.

If Bash is spawned by `login` in a TTY, by an SSH daemon, or similar means, it is considered a **login shell**. This mode can also be engaged using the `-l` / `--login` command line option.

Bash is considered an **interactive shell** when its standard input, output and error are connected to a terminal (for example, when run in a terminal emulator), and it is not started with the `-c` option or non-option arguments (for example, `bash script`). All interactive shells source `/etc/bash.bashrc` and `~/.bashrc`, while interactive *login* shells also source `/etc/profile` and `~/.bash_profile`.

Note In Arch `/bin/sh` (which used to be the Bourne shell executable) is symlinked to `bash`. If Bash is invoked with the name `sh`, it tries to mimic the startup behavior of historical versions of `sh`, including POSIX compatibility.

Configuration files

Bash will attempt to execute a set of startup files depending on how it was invoked. See the Bash Startup Files section of the GNU Bash manual for a complete description.

File	Description	Login shells (see note)	Interactive, *non-login* shells
`/etc/profile`	Sources application settings in `/etc/profile.d/*.sh` and `/etc/bash.bashrc`.	Yes	No
`~/.bash_profile`	Per-user, after `/etc/profile`. If this file does not exist, `~/.bash_login` and `~/.profile` are checked in that order. The skeleton file `/etc/skel/.bash_profile` also sources `~/.bashrc`.	Yes	No

`~/.bash_logout`	Per-user, after exit of a login shell.	Yes	No
`/etc/bash.bash_logout`	Depends on the `-DSYS_BASH_LOGOUT="/etc/bash.bash_logout"` compilation flag. After exit of a login shell.	Yes	No
`/etc/bash.bashrc`	Depends on the `-DSYS_BASHRC="/etc/bash.bashrc"` compilation flag. Sources `/usr/share/bash-completion/bash_completion`.	No	Yes
`~/.bashrc`	Per-user, after `/etc/bash.bashrc`.	No	Yes

Note

- Login shells can be non-interactive when called with the `--login` argument.
- While interactive, *non-login* shells do **not** source `~/.bash_profile`, they still inherit the environment from their parent process (which may be a login shell). See GregsWiki:ProcessManagement#On processes, environments and inheritance for details.

Shell and environment variables

The behavior of Bash and programs run by it can be influenced by a number of environment variables. Environment variables are used to store useful values such as command search directories, or which browser to use. When a new shell or script is launched it inherits its parent's variables, thus starting with an internal set of shell variables[1].

These shell variables in Bash can be exported in order to become environment variables:

```
VARIABLE=content
export VARIABLE
```

or with a shortcut

```
export VARIABLE=content
```

Environment variables are conventionally placed in `~/.profile` or `/etc/profile` so that other Bourne-compatible shells can use them.

See Environment variables for more general information.

Command line

Bash command line is managed by the separate library called Readline. Readline provides emacs and vi styles of shortcuts for interacting with the command line, i.e. moving back and forth on the word basis, deleting words etc.

It is also Readline's responsibility to manage history of input commands. Last, but not least, it allows you to create macros.

Tab completion

Tab completion is the option to auto-complete typed commands by pressing `Tab` (enabled by default).

Single-tab

It may require up to three tab-presses to show all possible completions for a command. To reduce the needed number of tab-presses, see Readline#Faster completion.

Common programs and options

By default, Bash only tab-completes commands, filenames, and variables. The package bash-completion extends this by adding more specialized tab completions for common commands and their options, which can be enabled by sourcing `/usr/share/bash-completion/bash_completion` (which has been already sourced in Arch's `/etc/bash.bashrc`). With bash-completion, normal completions (such as `ls file.*` `Tab` `Tab`) will behave differently; however, they can be re-enabled with `compopt -o bashdefault` *`program`* (see [2] and [3] for more detail).

Customize per-command

Note Using the `complete` builtin may cause conflicts with bash-completion.

By default, Bash only tab-completes file names following a command. You can change it to complete command names using `complete -c`:

~/.bashrc

```
complete -c man which
```

or complete command names and file names with `-cf`:

```
complete -cf sudo
```

See bash(1) § Programmable Completion for more completion options.

History

History completion

You can bind the up and down arrow keys to search through Bash's history (see: Readline#History and Readline Init File Syntax):

```
~/.bashrc
```

```
bind '"\e[A": history-search-backward'
bind '"\e[B": history-search-forward'
```

or to affect all readline programs:

```
~/.inputrc
```

```
"\e[A": history-search-backward
"\e[B": history-search-forward
```

History customization

The `HISTCONTROL` variable can prevent certain commands from being logged to the history.

To stop logging of consecutive identical commands:

```
~/.bashrc
```

```
export HISTCONTROL=ignoredups
```

To remove all but the last identical command:

```
~/.bashrc
```

```
export HISTCONTROL=erasedups
```

To avoid saving commands that start with a space:

```
~/.bashrc
```

```
export HISTCONTROL=ignorespace
```

To avoid saving consecutive identical commands, and commands that start with a space:

```
~/.bashrc
```

```
export HISTCONTROL=ignoreboth
```

To remove all but the last identical command, and commands that start with a space:

```
~/.bashrc
```

```
export HISTCONTROL="erasedups:ignorespace"
```

See bash(1) § HISTCONTROL for details.

Disable history

To disable the bash history only temporarily:

```
$ set +o history
```

The commands entered now are not logged to the `$HISTFILE`.

For example, now you can hash passwords with `printf secret | sha256sum`, or hide GPG usage like `gpg -eaF secret-pubkey.asc` and your secret is not written to disk.

To enable history:

```
$ set -o history
```

To disable all bash history:

```
~/.bashrc or /etc/profile
```

```
export HISTSIZE=0
```

... and just to make sure, destroy your old histfile forever:

```
$ wipe -i -l2 -x4 -p4 "$HISTFILE"
$ ln -sv /dev/null "$HISTFILE"
```

Mimic Zsh run-help ability

Zsh can invoke the manual for the command preceding the cursor by pressing `Alt+h`. A similar behaviour is obtained in Bash using this Readline bind:

```
~/.bashrc
```

```
run-help() { help "$READLINE_LINE" 2>/dev/null || man "$READLINE_LINE"; }
bind -m vi-insert -x '"\eh": run-help'
bind -m emacs -x     '"\eh": run-help'
```

This assumes are you using the (default) Emacs editing mode.

atuin replaces your existing shell history with an SQLite database, and records additional context for your commands. Additionally, it provides optional and fully encrypted synchronization of your history between machines, via an Atuin server.

Enable bash history timestamps (`export HISTTIMEFORMAT="%F %T "`) before syncing. Atuin works well with tools like blesh-git[AUR] and cmd-wrapped to provide an enhanced terminal experience across machines.

Aliases

alias is a command, which enables a replacement of a word with another string. It is often used for abbreviating a system command, or for adding default arguments to a regularly used command.

Personal aliases can be stored in `~/.bashrc` or any separate file sourced from `~/.bashrc` . System-wide aliases (which affect all users) belong in `/etc/bash.bashrc` . See [4] for example aliases.

For functions, see Bash/Functions.

Tips and tricks

Prompt customization

See Bash/Prompt customization.

Syntax highlighting and autosuggestions

ble.sh (Bash Line Editor), packed as blesh-git[AUR], is a command line editor written in pure Bash, which is an alternative to GNU Readline. It has many enhanced features like syntax highlighting, autosuggestions, menu-completion, abbreviations, Vim editing mode, and hook functions. Other interesting features include status line, history share, right prompt, transient prompt, and xterm title.

After installing it, source it in an interactive session.

```
~/.bashrc
```

```
source /usr/share/blesh/ble.sh
```

Configurations are explained in depth in the ~/.blerc file and at the wiki. The stable blesh[AUR] package is also available.

Command not found

pkgfile includes a "command not found" hook that will automatically search the official repositories, when entering an unrecognized command.

You need to source the hook to enable it, for example:

```
~/.bashrc
```

```
source /usr/share/doc/pkgfile/command-not-found.bash
```

Then attempting to run an unavailable command will show the following info:

```
$ abiword
```

```
abiword may be found in the following packages:
  extra/abiword 3.0.1-2 /usr/bin/abiword
```

Note The pkgfile database may need to be updated before this will work. See pkgfile#Installation for details.

Disable Ctrl+z in terminal

You can disable the `Ctrl+z` feature (pauses/closes your application) by wrapping your command like this:

```
#!/bin/bash
trap "" 20
adom
```

Now, when you accidentally press `Ctrl+z` in adom[AUR] instead of `Shift+z`, nothing will happen because `Ctrl+z` will be ignored.

Clear the screen after logging out

To clear the screen after logging out on a virtual terminal:

```
~/.bash_logout
```

```
clear
```

```
reset
```

Auto "cd" when entering just a path

Bash can automatically prepend `cd` when entering just a path in the shell. For example:

```
$ /etc
```

```
bash: /etc: Is a directory
```

But after adding one line into `.bashrc` file:

```
~/.bashrc
```

```
...
shopt -s autocd
...
```

You get:

```
[user@host ~]$ /etc
cd /etc
[user@host etc]$
```

Autojump

autojump-git[AUR] is a python script which allows navigating the file system by searching for strings in a database with the user's most-visited paths.

zoxide is an alternative which has additional features and performance improvements compared to the original `autojump` and can serve as a drop-in replacement for autojump.

Prevent overwrite of files

For the current session, to disallow existing regular files to be overwritten by redirection of shell output:

```
$ set -o noclobber
```

This is identical to `set -C`.

To make the changes persistent for your user:

```
~/.bashrc
```

```
...
set -o noclobber
```

To manually overwrite a file while `noclobber` is set:

```
$ echo "output" >| file.txt
```

Use directory stack to navigate

`pushd` and `popd` can be used to push or pop directories to a stack while switching to them. This can be useful for "replaying" your navigation history.

```
[user@host ~] pushd /tmp/dir1
[user@host /tmp/dir1] pushd /var/lib
[user@host /var/lib] popd
[user@host /tmp/dir1] popd
[user@host ~]
```

See bash(1) § DIRSTACK.

Shell environment detection

See Environment variables#Shell environment detection.

Troubleshooting

Line wrap on window resize

When resizing a terminal emulator, Bash may not receive the resize signal. This will cause typed text to not wrap correctly and overlap the prompt. The `checkwinsize` shell option checks the window size after each command and, if necessary, updates the values of `LINES` and `COLUMNS`.

```
~/.bashrc
```

```
shopt -s checkwinsize
```

Shell exits even if ignoreeof set

If you have set the `ignoreeof` option and you find that repeatedly hitting `ctrl-d` causes the shell to exit, it is because this option only allows 10 consecutive invocations of this keybinding (or 10 consecutive EOF characters,

to be precise), before exiting the shell.

To allow higher values, you have to use the IGNOREEOF variable.

For example:

```
export IGNOREEOF=100
```

Checking errors by analyzing scripts

The package shellcheck analyzes bash (and other shell) scripts, prints possible errors, and suggests better coding.

There is also the web site shellcheck.net of the same purpose, based on this program.

See also

- Bash Reference Manual, or `/usr/share/doc/bash/bashref.html`
- Readline Init File Syntax
- The Bourne-Again Shell - The third chapter of *The Architecture of Open Source Applications*
- PS1 generator - generate your .bashrc/PS1 bash prompt using an intuitive UI
- Even more useful .bashrc commands

Tutorials

- Greg's Wiki
- GregsWiki:BashGuide
- GregsWiki:BashFAQ
- Quote Tutorial

- An active and friendly IRC channel for Bash

Examples

- How to change the title of an xterm
