Engineer

Bash

the GNU shell most scripts are written for

What it is

Bash, the Bourne-Again SHell, distributed as GNU Bash and started as the `bash` command, is a program that reads commands, typed at a prompt or saved in a file, and carries them out: it starts other programs, connects their input and output, and adds variables, loops, tests and functions so that a sequence of commands can make decisions. The name is a pun on Stephen Bourne, whose `sh` shipped with the Seventh Edition of Research Unix at Bell Labs. Its manual describes Bash as largely compatible with that shell, with features taken from the Korn shell and the C shell, and as intended to conform to the shell part of the POSIX standard. The current release is 5.3, and the 5.3 manual is dated May 2025.

A shell is easy to confuse with the window it runs in: the terminal draws characters and forwards keystrokes, and the shell is the program on the other side that interprets them. Bash runs interactively, with a prompt, history and tab completion, or non-interactively, reading a script from top to bottom unattended, which is where most of its traps live.

Bash is one of several programs that can fill the same role. POSIX defines a portable shell language, `sh` , and the program installed as `/bin/sh` varies by system. On Debian and Ubuntu it is dash, a small POSIX shell descended from the Almquist shell, and Debian Policy names bash, dash and ksh as the most common shells to use there. Zsh has been the default login shell for new accounts on macOS since version 10.15, although the Mac still carries a `/bin/bash` . A script picks its interpreter with its first line, the shebang: `#!/bin/sh` asks for whatever POSIX shell the system has, while `#!/bin/bash` or `#!/usr/bin/env bash` asks for Bash by name.

Use the concept in a decision

Put “What is Bash?” to work in a real decision. We can help relate the concept to the system you are building or buying. Day-to-day care of servers, mail, accounts and updates, so the things everybody relies on stay boring.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Ran 10 PostgreSQL and MS SQL servers on Ubuntu — 10 database servers deployed and kept healthy

Ran 10 PostgreSQL and MS SQL servers on Ubuntu Get in touch →

Why it matters

Bash is the connecting language of Unix servers. It is on almost every Linux machine, and the commands an engineer types at a prompt are already Bash, so a handful of them saved in a file becomes a script that runs on what the machine already has. For chaining existing programs, such as dumping a database, compressing the dump and copying it elsewhere, few languages are as brief, because pipes, redirection and exit statuses are its native vocabulary.

The same traits turn into costs as a script grows. Every value is a string, and arithmetic works only on integers, so anything with structure, such as JSON, dates or decimal amounts, has to be handed to another tool. Unquoted expansions are split into words and then matched against filenames, so a value containing a space or an asterisk changes the meaning of the command it lands in. Error handling is opt-in: by default a script carries on after a command fails, and the options that change that, `set -e` and `pipefail`, come with exceptions the manual needs several paragraphs to list. A workable rule is that once a script needs data structures, a parser or a test suite, it has become a program, and a language such as Python is cheaper to maintain.

Portability is a decision as well. Debian Policy requires `/bin/sh` scripts to keep to POSIX features plus a short list of extensions, and asks any script that needs more to name its shell on the first line, for example `#!/bin/bash`. It suggests `checkbashisms` or a run under a stricter shell such as posh to find violations, and advises `/bin/bash` when in doubt. The standard moves slowly: Bash has offered `pipefail` since version 3.0, and POSIX added it only in its 2024 edition.

A shell turns text into commands, which makes it a security boundary wherever outside text reaches it. Shellshock, recorded as CVE-2014-6271 in September 2014, showed how wide that boundary can be. Bash up to version 4.3 executed commands that trailed a function definition in an environment variable, so

anything that copied attacker-controlled text into the environment before starting Bash, including CGI scripts under Apache, the ForceCommand feature of OpenSSH and some DHCP clients, became a way to run code on the machine. The original fix was incomplete, and what remained received its own identifier, CVE-2014-7169. The everyday lesson is smaller: outside text reaches a command as an argument, never as part of a string handed to `eval` or `sh -c`.

What it looks like in practice

Across the company's work, Bash keeps the job the tech stack assigns it: Python for data work, automation and services, and Bash for the glue, written to be read.

In the six ETL and ELT pipelines that fed Google BigQuery from a data warehouse behind an IPsec VPN, the downloaded archives held Parquet, CSV and .bak files. Bash and Python scripts unpacked them and classified each file by type and column layout before bcp, psql and SSIS loaded them into staging schemas and gzip, Google Cloud Storage and the bq tool carried the results into BigQuery. With the unpacking and sorting scripted, a run that had taken three hours or more by hand finished in under 20 minutes, and the data could be synced every day instead of once a week.

The hundred automated critical data backups show the same division of labour. Barman handled the PostgreSQL side, Google Cloud held the offsite copies, and Bash and Python orchestrated and validated the whole regime, with retention policies keeping the copies current and integrity checks confirming that each one was good rather than merely present. That is the job Bash suits: a sequence of existing tools, where the script's value lies in checking what each step returned.

Where it goes wrong

A filename with a space becomes two arguments. An unquoted `$file` is split on whitespace and then expanded as a glob, so `report 2024.csv` reaches `rm` as two names and a stray `*` matches a whole directory. Quote every expansion, as in `"$file"`, pass argument lists on as `"$@"`, keep lists of names in arrays, and let ShellCheck find the rest.

A backup pipeline reports success after the dump has failed. A pipeline's exit status is that of its last command, so in `pg_dump app | gzip > app.sql.gz` a failed dump still leaves gzip succeeding on whatever little it received. Turn on `set -o`

`pipefail`, which makes the pipeline return the last non-zero status, and check the result with `gzip -t` or, better, a test restore.

A script with `set -e` carries on past a failed command. The option is ignored in the test of an `if` or `while`, in every part of an `&&` or `||` list except the last, and inside any function called from those places, so the dump below can fail and the script still prints its success message:

```
#!/usr/bin/env bash
set -euo pipefail

backup() {
  pg_dump app > /var/backups/app.sql
  gzip -f /var/backups/app.sql
}

if backup; then # set -e is ignored inside backup here
  echo "backup done"
fi
```

A related trap is `local out=$(cmd)`, which returns the status of `local` and hides a failure in `cmd`. Inside functions, follow each command that matters with `|| return 1`, and declare a variable before assigning to it on a separate line.

A script that works on one machine breaks on another. A file that starts with `#!/bin/sh` but uses `[[` tests or arrays runs under Bash on one system and under dash on the next. There `[[` is an unknown command, so an `if` built on it prints an error and takes its else branch, and an array assignment is a syntax error that ends the script. On a Mac, `/bin/bash` reports version 3.2.57, older than the associative arrays and `mapfile` that arrived in 4.0, so `declare -A` stops with an invalid-option error there. Name the shell the script really needs in the shebang, and test against the oldest version it has to run on.

A cleanup line deletes far more than it should. With `build_dir` unset, `rm -rf "$build_dir"/*` expands to `rm -rf /*`. `set -u` turns an unset variable into an error, and writing `"${build_dir:?}"` stops a non-interactive script whenever the variable is unset or holds an empty string.

Further reading

- GNU Bash Reference Manual — the 5.3 manual: expansions, quoting, the `set` builtin and the exact cases where `set -e` is ignored.
- Shell Command Language — the portable `sh` language in POSIX.1-2024, the limit for any `#!/bin/sh` script.
- Debian Policy: Scripts — what `/bin/sh` must support on Debian, when a script must ask for Bash by name, and the advice on `set -e`.
- Bash NEWS — the maintainer's release-by-release list of new features, for checking which version a construct needs.
- CVE-2014-6271 — the NVD record for Shellshock, with the attack paths it lists and the follow-up identifier.

Glossary

Choose for your actual constraints

Bring the deployment path, the operational risks and the people on call. We can scope a change with checks, a recovery route and clear ownership.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is Bash?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/bash/