Next: Introduction, Previous: (dir), Up: (dir) [Contents][Index]

Bash Features

This text is a brief description of the features that are present in the Bash shell (version 5.3, 18 May 2025). The Bash home page is http://www.gnu.org/software/bash/.

This is Edition 5.3, last updated 18 May 2025, of *The GNU Bash Reference Manual*, for `Bash`, Version 5.3.

Bash contains features that appear in other popular shells, and some features that only appear in Bash. Some of the shells that Bash has borrowed concepts from are the Bourne Shell (`sh`), the Korn Shell (`ksh`), and the C-shell (`csh` and its successor, `tcsh`). The following menu breaks the features up into categories, noting which features were inspired by other shells and which are specific to Bash.

This manual is meant as a brief introduction to features found in Bash. The Bash manual page should be used as the definitive reference on shell behavior.

Table of Contents

Next: Definitions, Up: Bash Features [Contents][Index]

1 Introduction

Next: What is a shell?, Up: Introduction [Contents][Index]

1.1 What is Bash?

Bash is the shell, or command language interpreter, for the GNU operating system. The name is an acronym for the ‘`Bourne-Again SHell`’, a pun on Stephen Bourne, the author of the direct ancestor of the current Unix shell `sh`, which appeared in the Seventh Edition Bell Labs Research version of Unix.

Bash is largely compatible with `sh` and incorporates useful features from the Korn shell `ksh` and the C shell `csh`. It is intended to be a conformant implementation of the IEEE POSIX Shell and Tools portion of the IEEE POSIX specification (IEEE Standard 1003.1). It offers functional improvements over `sh` for both interactive and programming use.

While the GNU operating system provides other shells, including a version of `csh`, Bash is the default shell. Like other GNU software, Bash is quite portable. It currently runs on nearly every version of Unix and a few other operating systems − independently-supported ports exist for Windows and other platforms.

Previous: What is Bash?, Up: Introduction [Contents][Index]

1.2 What is a shell?

At its base, a shell is simply a macro processor that executes commands. The term macro processor means functionality where text and symbols are expanded to create larger expressions.

A Unix shell is both a command interpreter and a programming language. As a command interpreter, the shell provides the user interface to the rich set of GNU utilities. The programming language features allow these utilities to be combined. Users can create files containing commands, and these become commands themselves. These new commands have the same status as system commands in directories such as `/bin`, allowing users or groups to establish custom environments to automate their common tasks.

Shells may be used interactively or non-interactively. In interactive mode, they accept input typed from the keyboard. When executing non-interactively, shells execute commands read from a file or a string.

A shell allows execution of GNU commands, both synchronously and asynchronously. The shell waits for synchronous commands to complete before accepting more input; asynchronous commands continue to execute in parallel with the shell while it reads and executes additional commands. The *redirection* constructs permit fine-grained control of the input and output of those commands. Moreover, the shell allows control over the contents of commands' environments.

Shells also provide a small set of built-in commands (*builtins*) implementing functionality impossible or inconvenient to obtain via separate utilities. For example, `cd`, `break`, `continue`, and `exec` cannot be implemented outside of the shell because they directly manipulate the shell itself. The `history`, `getopts`, `kill`, or `pwd` builtins, among others, could be implemented in separate utilities, but they are more convenient to use as builtin commands. All of the shell builtins are described in subsequent sections.

While executing commands is essential, most of the power (and complexity) of shells is due to their embedded programming languages. Like any high-level language, the shell provides variables, flow control constructs, quoting, and functions.

Shells offer features geared specifically for interactive use rather than to augment the programming language. These interactive features include job control, command line editing, command history and aliases. This manual

describes how Bash provides all of these features.

Next: Basic Shell Features, Previous: Introduction, Up: Bash Features

[Contents][Index]

2 Definitions

These definitions are used throughout the remainder of this manual.

POSIX

A family of open system standards based on Unix. Bash is primarily concerned with the Shell and Utilities portion of the POSIX 1003.1 standard.

blank

A space or tab character.

whitespace

A character belonging to the `space` character class in the current locale, or for which `isspace()` returns true.

builtin

A command that is implemented internally by the shell itself, rather than by an executable program somewhere in the file system.

control operator

A `token` that performs a control function. It is a `newline` or one of the following: '`||`', '`&&`', '`&`', '`;`', '`;;`', '`;&`', '`;;&`', '`|`', '`|&`', '`(`', or '`)`'.

exit status

The value returned by a command to its caller. The value is restricted to eight bits, so the maximum value is 255.

field

A unit of text that is the result of one of the shell expansions. After expansion, when executing a command, the resulting fields are used as the command name and arguments.

filename

A string of characters used to identify a file.

job

A set of processes comprising a pipeline, and any processes descended from it, that are all in the same process group.

job control

A mechanism by which users can selectively stop (suspend) and restart (resume) execution of processes.

metacharacter

A character that, when unquoted, separates words. A metacharacter is a `space`, `tab`, `newline`, or one of the following characters: '`|`', '`&`', '`;`', '`(`', '`)`', '`<`', or '`>`'.

name

A `word` consisting solely of letters, numbers, and underscores, and beginning with a letter or underscore. `Names` are used as shell variable and function names. Also referred to as an `identifier`.

operator

A `control operator` or a `redirection operator`. See Redirections, for a list of redirection operators. Operators contain at least one unquoted `metacharacter`.

process group

A collection of related processes each having the same process group ID.

process group ID

A unique identifier that represents a `process group` during its lifetime.

reserved word

A `word` that has a special meaning to the shell. Most reserved words introduce shell flow control constructs, such as `for` and `while`.

return status

A synonym for `exit status`.

signal

A mechanism by which a process may be notified by the kernel of an event occurring in the system.

special builtin

A shell builtin command that has been classified as special by the POSIX standard.

token

A sequence of characters considered a single unit by the shell. It is either a `word` or an `operator`.

word

A sequence of characters treated as a unit by the shell. Words may not include unquoted `metacharacters`.

Next: Shell Builtin Commands, Previous: Definitions, Up: Bash Features

[Contents][Index]

3 Basic Shell Features

Bash is an acronym for 'Bourne-Again SHell'. The Bourne shell is the traditional Unix shell originally written by Stephen Bourne. All of the Bourne shell builtin commands are available in Bash, and the rules for evaluation and quoting are taken from the POSIX specification for the 'standard' Unix shell.

This chapter briefly summarizes the shell's 'building blocks': commands, control structures, shell functions, shell *parameters*, shell expansions, *redirections*, which are a way to direct input and output from and to named files, and how the shell executes commands.