

Bash (Unix shell)

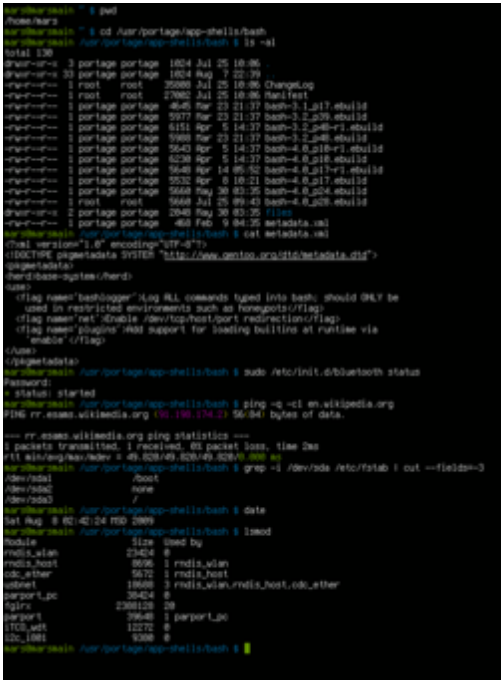
From Wikipedia, the free encyclopedia

Bash is a Unix shell written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell (sh).^{[3][4]} Released in 1989,^[5] it has been distributed widely as the shell for the GNU operating system and as a default shell on Linux and Mac OS X. It has been ported to Microsoft Windows and distributed with Cygwin and MinGW, to DOS by the DJGPP project, to Novell NetWare and to Android via various terminal emulation applications. In the late 1990s, many shells were in common use, of which bash was a minor player, unlike presently where bash has overwhelming favor.

Bash is a command processor, typically run in a text window, allowing the user to type commands which cause actions. Bash can also read commands from a file, called a script. Like all Unix shells, it supports filename wildcarding, piping, here documents, command substitution, variables and control structures for condition-testing and iteration.^[6] The keywords, syntax and other basic features of the language were all copied from sh. Other features, e.g., history, were copied from csh and ksh. Bash is a POSIX shell but with a number of extensions.

The name itself is an acronym, a pun, and a description. As an acronym, it stands for *Bourne-again shell*, referring to its objective as a free replacement for the Bourne shell.^[7] As a pun, it expressed that objective in a phrase that sounds similar to *born again*, a term for spiritual rebirth.^{[8][9]} The name is also

Bash



Screenshot of Bash and sh sessions demonstrating some features

Original author(s)	Brian Fox
Initial release	June 7, 1989
Stable release	4.3 / February 27, 2014 ^[1]
Written in	C
Operating system	Cross-platform
Platform	GNU
Available in	English, multilingual (gettext)
Type	Unix shell
License	GNU GPL v3+ ^[2]
Website	<div>www.gnu.org/software/bash/</div> <div>(https://www.gnu.org/software/bash/)</div>

descriptive of what it did, *bashing together* the features of sh, csh, and ksh.^[10]

Contents

- 1 History
- 2 Features
 - 2.1 Brace expansion
 - 2.2 Startup scripts
 - 2.2.1 Execution order of startup files
 - 2.2.1.1 When started as an interactive login shell
 - 2.2.1.2 When a login shell exits
 - 2.2.1.3 When started as an interactive shell (but not a login shell)
 - 2.2.2 Comparison with the Bourne shell and csh startup sequences
 - 2.2.2.1 Setting inheritable environment variables
 - 2.2.2.2 Aliases and Functions
 - 2.2.2.3 Commands performed only at login and logout
 - 2.2.2.4 Legacy-compatible Bash startup example
 - 2.2.3 Operating system issues in Bash startup
 - 2.3 Portability
 - 2.4 Keyboard shortcuts
 - 2.5 Processes management
- 3 See also
- 4 References
- 5 External links

History

Brian Fox began coding Bash on January 10, 1988^[11] after Richard Stallman became dissatisfied with the lack of progress being made by a prior developer.^[3] Stallman and the Free Software Foundation (FSF) considered a free shell that could run existing sh scripts so strategic to a completely free system built from BSD and GNU code that this was one of the few projects they funded themselves,

with Fox undertaking the work as an employee of FSF.^{[3][12]} Fox released Bash as a beta, version .99, on June 7, 1989^[5] and remained the primary maintainer until sometime between mid-1992^[13] and mid-1994,^[14] when he was laid off from FSF^[15] and his responsibility was transitioned to another early contributor, Chet Ramey.^{[16][17][18]}

Features

The Bash command syntax is a superset of the Bourne shell command syntax. The vast majority of Bourne shell scripts can be executed by Bash without modification, with the exception of Bourne shell scripts stumbling into fringe syntax behavior interpreted differently in Bash or attempting to run a system command matching a newer Bash builtin, etc. Bash command syntax includes ideas drawn from the Korn shell (ksh) and the C shell (csh) such as command line editing, command history, the directory stack, the \$RANDOM and \$PPID variables, and POSIX command substitution syntax \$(...). When used as an interactive command shell and pressing the tab key, Bash automatically uses command line completion to match partly typed program names, filenames and variable names. The Bash command-line completion system is very flexible and customizable, and is often packaged with functions that complete arguments and filenames for specific programs and tasks.

Bash's syntax has many extensions lacking in the Bourne shell. Bash can perform integer calculations without spawning external processes. It uses the ((...)) command and the \$(...) variable syntax for this purpose. Its syntax simplifies I/O redirection. For example, it can redirect standard output (stdout) and standard error (stderr) at the same time using the &> operator. This is simpler to type than the Bourne shell equivalent 'command > file 2>&1'. Bash supports process substitution using the <(command)syntax, which substitutes the output of (or input to) a command where a filename is normally used.

When using the 'function' keyword, Bash function declarations are not compatible with Bourne/Korn/POSIX scripts (the Korn shell has the same problem when using 'function'), but Bash accepts the same function declaration syntax as the Bourne and Korn shells, and is POSIX-conformant. Because of these and other differences, Bash shell scripts are rarely runnable under the Bourne or Korn shell interpreters unless deliberately written with that compatibility in mind, which is becoming less common as Linux becomes more widespread. But in POSIX mode,^[19] Bash conformance with POSIX is nearly perfect.

Bash supports here documents. Since version 2.05b Bash can redirect standard input (stdin) from a "here string" using the <<< operator.

Bash 3.0 supports in-process regular expression matching using a syntax

reminiscent of Perl.^[20]

Bash 4.0 introduced support for associative arrays.^{[19][21]} Associative arrays allow a fake support for multi-dimensional (indexed) arrays, in a similar way to AWK:

```
declare -A a      # declare an associative array 'a' faking a bi-dimensional indexed array
i=1; j=2         # initialize some indices
a[$i,$j]=5       # associate value "5" to key "$i,$j" (i.e. "1,2")
echo ${a[$i,$j]} # print the stored value at key "$i,$j"
```

Brace expansion

Brace expansion, also called alternation, is a feature copied from the C shell. It generates a set of alternative combinations. Generated results need not exist as files. The results of each expanded string are not sorted and left to right order is preserved:

```
echo a{p,c,d,b}e # ape ace ade abe
echo {a,b,c}{d,e,f} # ad ae af bd be bf cd ce cf
```

Brace expansions should not be used in portable shell scripts, because the Bourne shell will not produce the same output.

```
#!/bin/sh
# A traditional shell does not produce the same output
echo a{p,c,d,b}e # a{p,c,d,b}e
```

When brace expansion is combined with wildcards, the braces are expanded first, and then the resulting wildcards are substituted normally. Hence, a listing of JPEG and PNG images in the current directory could be obtained using:

```
ls *.{jpg,jpeg,png} # expands to *.jpg *.jpeg *.png - after which,
                   # the wildcards are processed
```

Startup scripts

When Bash starts it executes the commands in a variety of dot files. Though similar to Bash shell script commands, which have execute permission enabled and an [interpreter directive] like `#!/bin/bash`, the initialization files used by Bash require neither.

Execution order of startup files

When started as an interactive login shell

Bash reads and executes `/etc/profile` (if it exists). (Often this file calls `/etc/bash.bashrc`.)

After reading that file, it looks for `~/.bash_profile`, `~/.bash_login`, and `~/.profile` *in that order*, and reads and executes the first one that exists and is readable.

When a login shell exits

Bash reads and executes `~/.bash_logout` (if it exists).

When started as an interactive shell (but not a login shell)

Bash reads and executes `~/.bashrc` (if it exists). This may be inhibited by using the `--norc` option. The `--rcfile` file option will force Bash to read and execute commands from `file` instead of `~/.bashrc`.

Comparison with the Bourne shell and csh startup sequences

Elements of Bash were derived from the Bourne shell and csh, and allow limited startup file sharing with the Bourne shell and provide some startup features familiar to users of the csh.

Setting inheritable environment variables

In the Bourne shell, the `~/.profile` is used at login for setting environment variables which are then inherited by subprocesses. For Bash, the `~/.profile` can still be used in a compatible way, by executing it explicitly from the Bash-specific `~/.bash_profile` OR `~/.bash_login` with the line below. Bash-specific syntax can be kept out of the `~/.profile` to keep the latter compatible with the Bourne shell.

```
-----
```

```
. ~/.profile
```

Aliases and Functions

These two facilities, aliases from `csh` and the more general functions largely superseding them from Bourne shell, were not typically inheritable from the login shell and had to be redefined in each subshell spawned from the login shell. Although there is an `ENV` environment variable that could be applied to the problem, both `csh` and Bash support per-subshell startup files which address it directly. In Bash, the `~/.bashrc` is called for interactive subshells. If user-defined functions from the `~/.bashrc` are desired in the login shell as well, the `~/.bash_login` can include the line below after any setting up of environment variables:

```
. ~/.bashrc
```

Commands performed only at login and logout

The `csh` supports a `~/.login` file for purposes of tasks performed only during initial login, such as displaying system load, disk status, whether email has come in, logging the login time, etc. The Bourne shell can emulate this in the `~/.profile`, but doesn't predefine a file name. To achieve similar semantics to the `csh` model, the `~/.bash_profile` can contain the line below, after the environment setup and function setup:

```
. ~/.bash_login
```

Likewise, the `csh` has a `~/.logout` file run only when the login shell exits. The Bash equivalent is `~/.bash_logout`, and requires no special setup. In the Bourne shell, the `trap` built-in can be used to achieve a similar effect.

Legacy-compatible Bash startup example

The skeleton `~/.bash_profile` below is compatible with the Bourne shell and gives

semantics similar to `csh` for the `~/.bashrc` and `~/.bash_login`. The `[-r filename]` are tests to see if the *filename* exists and is readable, simply skipping the part after the `&&` if it's not.

```
[ -r ~/.profile ] && . ~/.profile          # set up environment, once, Bourne-sh syntax only.
if [ -n "$PS1" ] ; then                  # are we interactive?
[ -r ~/.bashrc ] && . ~/.bashrc           # tty/prompt/function setup for interactive shells.
[ -r ~/.bash_login ] && . ~/.bash_login   # any at-login tasks for login shell only.
fi
```

Operating system issues in Bash startup

Some versions of Unix and Linux contain Bash system startup scripts, generally under the `/etc` directories, which are called as part of standard Bash initialization but which themselves may read in other startup files in a different order than that in the documented Bash startup sequence. The default content of the root user's files may also have issues, as well as the skeleton files provided to new user accounts upon setup. The startup scripts that launch the X window system may also do surprising things with the user's Bash startup scripts in an attempt to set up the user's environment variables before launching the window manager, but are often addressable using a `~/.xsession` or `~/.xprofile` file to read the `~/.profile`, which provides the environment variables needed for the Bash shell windows spawned from the window manager, such as `xterm` or `Gnome Terminal`.

Portability

Invoking Bash with the `--posix` option or stating `set -o posix` in a script causes Bash to conform very closely to the POSIX 1003.2 standard.^[22] Bash shell scripts intended for portability should at least take into account the Bourne shell it intends to replace. Bash has certain features that the traditional Bourne shell lacks. Among these are:^[22]

- Certain extended invocation options
- Command substitution using `$()` notation (this feature is part of the POSIX 1003.2 standard though)
- Brace expansion
- Certain array operations, and associative arrays
- The double brackets extended test construct
- The double-parentheses arithmetic-evaluation construct
- Certain string manipulation operations

- Process substitution
- A Regular Expression matching operator
- Bash-specific builtins
- Coprocesses


Keyboard shortcuts

The following shortcuts work when using default (Emacs) key bindings.

Vi-bindings can be enabled by running `set -o vi`.^[23]

Note: For shortcuts involving `Alt`, you may be able to use `Esc` instead.

Note: Sometimes, you must use `Esc` instead of `Alt`, because the `Alt` shortcut conflicts with another shortcut. For example, in Trisquel 5.0 (a distribution of Linux), pressing `Alt + f` will not move the cursor forward one word, but will activate "File" in the menu of the terminal window.

- `Tab`  : Autocompletes from the cursor position.
- `Ctrl + a` : Moves the cursor to the line start (equivalent to the key `Home`).
- `Ctrl + b` : Moves the cursor back one character (equivalent to the key `←`).
- `Ctrl + c` : Sends the signal SIGINT to the current task, which aborts and closes it.
- `Ctrl + d`
 - Sends an EOF marker, which (unless disabled by an option) closes the current shell (equivalent to the command `exit`). (Only if there is no text on the current line)
 - If there is text on the current line, deletes the current character (then equivalent to the key `Delete`).
- `Ctrl + e` : (end) moves the cursor to the line end (equivalent to the key `End`).
- `Ctrl + f` : Moves the cursor forward one character (equivalent to the key `→`).
- `Ctrl + g` : Abort the research and restore the original line.
- `Ctrl + h` : Deletes the previous character (same as backspace).
- `Ctrl + i` : Equivalent to the tab key.
- `Ctrl + j` : Equivalent to the enter key.
- `Ctrl + k` : Clears the line content after the cursor and copies it into the clipboard.
- `Ctrl + l` : Clears the screen content (equivalent to the command `clear`).

- **Ctrl + n** : (next) recalls the next command (equivalent to the key **↓**).
- **Ctrl + o** : Executes the found command from history, and fetch the next line relative to the current line from the history for editing.
- **Ctrl + p** : (previous) recalls the prior command (equivalent to the key **↑**).
- **Ctrl + r** : (research) recalls the last command including the specified character(s). A second **Ctrl + r** recalls the next anterior command which corresponds to the research
- **Ctrl + s** : Go back to the next more recent command of the research (beware to not execute it from a terminal because this command also launches its XOFF). If you changed that XOFF setting, use **Ctrl + q** to return.
- **Ctrl + t** : Transpose the previous two characters.
- **Ctrl + u** : Clears the line content before the cursor and copies it into the clipboard.
- **Ctrl + v** : If the next input is also a control sequence, type it literally (e. g. * **Ctrl + v Ctrl + h** types "^H", a literal backspace.)
- **Ctrl + w** : Clears the word before the cursor and copies it into the clipboard.
- **Ctrl + x Ctrl + e** : Edits the current line in the \$EDITOR program, or vi if undefined.
- **Ctrl + x Ctrl + r** : Read in the contents of the inputrc file, and incorporate any bindings or variable assignments found there.
- **Ctrl + x Ctrl + u** : Incremental undo, separately remembered for each line.
- **Ctrl + x Ctrl + v** : Display version information about the current instance of bash.
- **Ctrl + x Ctrl + x** : Alternates the cursor with its old position. (C-x, because x has a crossing shape).
- **Ctrl + y** : (yank) adds the clipboard content from the cursor position.
- **Ctrl + z** : Sends the signal SIGTSTP to the current task, which suspends it. To execute it in background one can enter bg. To bring it back from background or suspension fg ['process name or job id'] (foreground) can be issued.
- **Ctrl + _** : Incremental undo, separately remembered for each line.
- **Alt + b** : (backward) moves the cursor backward one word.
- **Alt + c** : Capitalizes the character under the cursor and moves to the end of the word.

- **Alt + d** : Cuts the word after the cursor.
- **Alt + f** : (forward) moves the cursor forward one word.
- **Alt + l** : Lowers the case of every character from the cursor's position to the end of the current word.
- **Alt + r** : Cancels the changes and puts back the line as it was in the history.
- **Alt + u** : Capitalizes every character from the cursor's position to the end of the current word.
- **Alt + .** : Insert the last argument to the previous command (the last word of the previous history entry).

Processes management

The Bash shell has two modes of execution for commands: batch (Unix), and concurrent mode.

To execute commands in batch (i.e., in sequence) they must be separated by the character ";":

```
command1; command2
```

in this example, when command1 is finished, command2 is executed.

To have a concurrent execution of command1 and command2, they must be executed in the bash shell in the following way:

```
command1 & command2
```

In this case command1 is executed in background (symbol &), returning immediately the control to the shell that executes command2.

Summarizing:

- Normally a command is executed in foreground (fg). The control of the shell returns to the user after the command finishes its execution.
- with the symbol & after the command, it can be executed in background (bg). The shell is ready to execute other commands, concurrently to the first

command.

- A program in the running state and in foreground (fg) can be suspended pressing CTRL-Z
- A suspended program can be restarted in foreground using the command fg or background using the command bg.

See also

- Comparison of command shells

References

1. ^ "Bash-4.3 available for FTP" (<http://lists.gnu.org/archive/html/info-gnu/2014-02/msg00010.html>). February 27, 2014. Retrieved March 17, 2014.
2. ^ GNU Project. [www.gnu.org/software/bash (<https://www.gnu.org/software/bash>) "README file"]. "Bash is free software, distributed under the terms of the [GNU] General Public License as published by the Free Software Foundation, version 3 of the License (or any later version)."
3. ^ ^{**a**} ^{**b**} ^{**c**} Richard Stallman (forwarded with comments by Chet Ramey) (February 10, 1988). "GNU + BSD = ? (news:2362@mandrill.CWRU.Edu)". comp.unix.questions (news:comp.unix.questions). Web link (<https://groups.google.com/forum/#!original/comp.unix.questions/iNjWwkyroR8/yedr9yDWSuQJ>). "For a year and a half, the GNU shell was "just about done". The author made repeated promises to deliver what he had done, and never kept them. Finally I could no longer believe he would ever deliver anything. So Foundation staff member Brian Fox is now implementing an imitation of the Bourne shell.". Retrieved March 22, 2011.
4. ^ Hamilton, Naomi (May 30, 2008), "The A-Z of Programming Languages: BASH/Bourne-Again Shell" (http://www.computerworld.com.au/article/222764/a-z_programming_languages_bash_bourne-again_shell/?pp=2&fp=16&fpid=1), *Computerworld*: 2, retrieved March 21, 2011, "When Richard Stallman decided to create a full replacement for the then-encumbered Unix systems, he knew that he would eventually have to have replacements for all of the common utilities, especially the standard shell, and those replacements would have to have acceptable licensing."
5. ^ ^{**a**} ^{**b**} Brian Fox (forwarded by Leonard H. Tower Jr.) (June 8, 1989). "Bash is in beta release!". gnu.announce (news:gnu.announce). Web link (<http://groups.google.com/group/gnu.announce/msg/a509f48ffb298c35?hl=en>). Retrieved October 28, 2010.
6. ^ Bourne, S. R. (July–August 1978). "The UNIX Shell". *The Bell System Technical*

Journal (Short Hills, NJ: American Telephone and Telegraph Company) **57** (6): 1971–1990. ISSN 0005-8580 (<https://www.worldcat.org/issn/0005-8580>). "The lines between `<<!` and `!` are called a *here* document; they are read by the shell and made available as the standard input."

7. ^ C Programming (<http://www.ddj.com/cpp/184404693>) by Al Stevens, Dr. Dobb's Journal, July 1, 2001
8. ^ Richard Stallman (November 12, 2010). "About the GNU Project" (<https://www.gnu.org/gnu/thegnuproject.html>). Free Software Foundation. Archived (<https://web.archive.org/web/20110424064815/http://www.gnu.org/gnu/thegnuproject.html>) from the original on April 24, 2011. Retrieved March 13, 2011. "'Bourne Again Shell' is a play on the name 'Bourne Shell', which was the usual shell on Unix."
9. ^ Gattol, Markus (March 13, 2011), *Bourne-again Shell* (<http://www.markus-gattol.name/ws/bash.html>), retrieved March 13, 2011, "The name is a pun on the name of the Bourne shell (sh), an early and important Unix shell written by Stephen Bourne and distributed with Version 7 Unix circa 1978, and the concept of being "born again"."
10. ^ Ian Darwin (June 13, 1989). "at&t-free ksh (was: job control is a bug, not a feature)". comp.os.minix (news:comp.os.minix). Web link (<http://groups.google.com/group/comp.os.minix/msg/63c036d82ceca4d6?hl=en>). "Yup, the gnu project's Born Again Shell ("bash") is an attempt at bashing all the features of sh together with many of those from both csh and ksh.". Retrieved March 21, 2011.
11. ^ Brian Fox (August 29, 1996), *shell.c* (<http://ftp.gnu.org/gnu/bash/bash-1.14.7.tar.gz>), Free Software Foundation, "Birthdate: Sunday, January 10th, 1988. Initial author: Brian Fox"
12. ^ Richard Stallman (October 3, 2010). "About the GNU Project" (<https://www.gnu.org/gnu/thegnuproject.html>). Free Software Foundation. Archived (<https://web.archive.org/web/20110424064815/http://www.gnu.org/gnu/thegnuproject.html>) from the original on April 24, 2011. Retrieved March 21, 2011. "Free Software Foundation employees have written and maintained a number of GNU software packages. Two notable ones are the C library and the shell. ... We funded development of these programs because the GNU Project was not just about tools or a development environment. Our goal was a complete operating system, and these programs were needed for that goal."
13. ^ len (g...@prep.ai.mit.edu) (April 20, 1993). "January 1993 GNU's Bulletin (news:gnusenet930421bulletin@prep.ai.mit.edu)". gnu.announce (news:gnu.announce). Web link (<http://groups.google.com/group/gnu.misc.discuss/msg/4f42c739cd7e8bd8>). Retrieved October 28, 2010.

14. ^ Ramey, Chet (August 1, 1994). "Bash - the GNU shell (Reflections and Lessons Learned)" (<http://www.linuxjournal.com/article/2800#N0xa50890.0xb46380>). *Linux Journal*. Archived (<http://web.archive.org/web/20081205082152/http://www.linuxjournal.com/article/2800>) from the original on December 5, 2008. Retrieved November 13, 2008.
15. ^ Chet Ramey (October 31, 2010), *Dates in your Computerworld interview* (<http://www.scribd.com/doc/40556434/2010-10-31-Chet-Ramey-Early-Bash-Dates>), retrieved October 31, 2010
16. ^ Chet Ramey (June 12, 1989). "Bash 0.99 fixes & improvements". gnu.bash.bug (news:gnu.bash.bug). Web link (<http://groups.google.com/group/gnu.bash.bug/msg/1fc7b688f5d44438?hl=en>). Retrieved November 1, 2010.
17. ^ Chet Ramey (July 24, 1989). "Some bash-1.02 fixes". gnu.bash.bug (news:gnu.bash.bug). Web link (<http://groups.google.com/group/gnu.bash.bug/msg/072a03645663caea?hl=en>). Retrieved October 30, 2010.
18. ^ Brian Fox (March 2, 1990). "Availability of bash 1.05". gnu.bash.bug (news:gnu.bash.bug). Web link (<http://groups.google.com/group/gnu.bash.bug/msg/e6112ccc8866e2f4?hl=en>). Retrieved October 30, 2010.
19. ^ **a b** "6.11 Bash POSIX Mode" (https://www.gnu.org/software/bash/manual/html_node/Bash-POSIX-Mode.html), *The GNU Bash Reference Manual, for Bash, Version 4.1* (https://www.gnu.org/software/bash/manual/html_node/index.html), December 23, 2009, archived (https://web.archive.org/web/20101203065400/http://www.gnu.org/software/bash/manual/html_node/index.html) from the original on December 3, 2010, retrieved October 26, 2010
20. ^ The syntax matches that shown on the `regex(7)` (<http://www.tin.org/bin/man.cgi?section=7&topic=regex>) man page.
21. ^ "The shell provides associative array variables, with the appropriate support to create, delete, assign values to, and expand them." <http://tiswww.case.edu/php/chet/bash/NEWS>
22. ^ **a b** Mendel Cooper. "Portability Issues" (<http://tldp.org/LDP/abs/html/portabilityissues.html>). *The Linux Documentation Project*. ibiblio.org.
23. ^ "BASH Help - A Bash Tutorial" (http://www.hypexr.org/bash_tutorial.php#emacs). Hypexr.org. October 5, 2012. Retrieved July 21, 2013.

External links

- Official website (<https://www.gnu.org/software/bash/bash.html>)
- Bash Reference Manual (<https://www.gnu.org>)



Wikimedia Commons
has media related to
Bash.

/software/bash/manual/) (HTML

(https://www.gnu.org/software/bash/manual/html_node/index.html) PS

(<https://www.gnu.org/software/bash/manual>

/bash.ps.gz) PDF (<https://www.gnu.org/software/bash/manual/bash.pdf>)



Wikibooks has a book on the topic of:
Bash Shell Scripting

- Bash Guide for Beginners (<http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html>) article at The Linux Documentation Project
- Advanced Bash Scripting Guide (<http://tldp.org/LDP/abs/html/index.html>) article at The Linux Documentation Project
- Linux Shell Scripting Tutorial (LSST) v2.0 wiki (http://bash.cyberciti.biz/guide/Main_Page)
- "The Comprehensive List of bash Reference Documentation and Examples" (<http://www.bashcookbook.com/bashinfo/>)
- Useful Bash History Tips and Tricks (<http://spsneo.com/blog/2009/09/19/bash-history-tips-and-tricks/>)
- 2008 interview with GNU Bash's maintainer, Chet Ramey (<http://www.computerworld.com.au/index.php/id;1591223321;fp;16;fpid;1;pf;1>)
- Working with BASH environment variables (http://www.geeksworld.com/tutorials/operating_systems/linux/tips_and_tricks/working_bash_environment_variables_beginners_linux.php)
- Video Tutorial for Creating a Bash Script (<http://www.galatech.co.uk/index.php/component/content/article/37-blogs/100-how-to-create-a-bash-script>)
- Bash commands and examples (<http://www.shell-fu.org/lister.php?tag=bash>)
- Colorized Bash prompt (http://www.markus-gattol.name/ws/bash.html#colorized_shell_prompt) - how to set up a colorized Bash prompt based on the current connection method (SSH, telnet, etc.).
- jBash Project is a Java Parser for the Bourne Again Shell (<https://code.google.com/p/jbash/>)
- The 'official' channel FAQ for freenode's #bash channel is BashFAQ. (<http://mywiki.woledge.org/EnglishFrontPage>)
- BASHDB - Bash with a built-in debugger. (<http://bashdb.sourceforge.net/>)
- Bash Quick Reference card (<http://www.digilife.be/quickreferences>)

/QRC/Bash%20Quick%20Reference.pdf) <-- Dates from 1999 and version 2.02.0 of BASH. Very old.

- Beginner Linux Command Line Tutorial (<http://ryanstutorials.net/linuxtutorial>) - Covers the basics, all the way up to scripting.
- Bash tutorial (http://wiki.altervista.org/cs/bash_shell) - A quick tutorial about bash shell, with the explanation of the most useful commands. Slides provided.

Retrieved from "[http://en.wikipedia.org/w/index.php?title=Bash_\(Unix_shell\)&oldid=626068447](http://en.wikipedia.org/w/index.php?title=Bash_(Unix_shell)&oldid=626068447)"

Categories: Unix shells | Text-oriented programming languages

| Scripting languages | GNU Project software | Free software programmed in C

| Cross-platform free software | 1989 software

- This page was last modified on 18 September 2014 at 10:56.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.