

BASH: THE BOURNE-AGAIN SHELL

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

By: Asa Jenkins

WHAT IS BASH?

- **Bash** stands for “Bourne-Again Shell”, which is based on the **Bourne** shell and mostly compatible with its features.
- A **Shell** is a interface that allows the user to interact with other programs.
- **Bash** is the *de facto* shell on most GNU/Linux distributions.
- Think of **Bash** as a way to speak to your system. It allows you to perform basic operations like math, run applications, or tests.
- “**Bash** is a *Shell program*, and I tell it what to do.”

```
● ● ●
BASH(1)                                1. bash (less)                               BASH(1)

NAME
    bash - GNU Bourne-Again SHell

SYNOPSIS
    bash [options] [file]

COPYRIGHT
    Bash is Copyright (C) 1989–2005 by the Free Software Foundation,
    Inc.

DESCRIPTION
    Bash is an sh-compatible command language interpreter that exe-
    cutes commands read from the standard input or from a file. Bash
    also incorporates useful features from the Korn and C shells (ksh
    and csh).

    Bash is intended to be a conformant implementation of the Shell
    and Utilities portion of the IEEE POSIX specification (IEEE Stan-
    dard 1003.1). Bash can be configured to be POSIX-conformant by
    default.

OPTIONS
    In addition to the single-character shell options documented in
    the description of the set builtin command, bash interprets the
```

“man bash”

HISTORY

- Bash initial release was in **June 8, 1989**.
- **Brian J. Fox** is the original author of the **GNU Bash Shell**.
- Written for the **GNU Project** as a replacement for the **Bourne Shell**.
- The **GNU Project** is a free software, mass-collaboration project that aims to develop software for users to control the way they use their computing devices.



PROGRAMMING LANGUAGE OR SCRIPTING LANGUAGE?

- Bash not only is an excellent command line shell, but also a scripting language.
- **Shell Scripting** allows the use of the shell's abilities and to automate a lot of the tasks that would otherwise require a lot of commands.
- The difference between a scripting language and programming language Is programming languages are more powerful and a lot faster, and generally programming languages start off a source code and compiled down into an executable.
- Scripting languages on the other hand, also starts from source code, but is not compiled down to an executable. Rather an interpreter reads the instructions in the source file and executes the instructions.

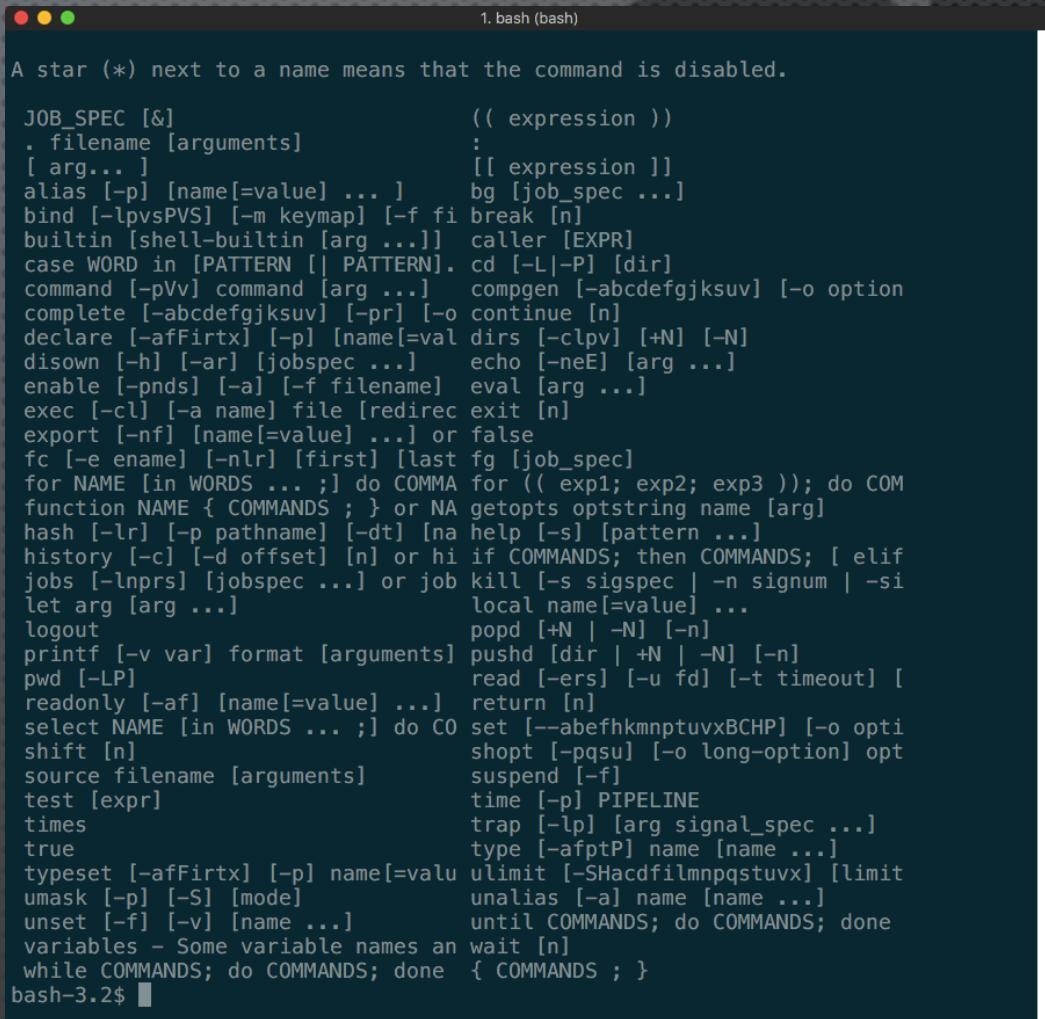
DESIGN

- Bash implements *Simple Commands* separated by spaces.
 - chmod u=rwx test1.txt
 - Sets test1.txt to readable/writeable/executable.
- You can pipe commands together using the (|) symbol.
 - ls -l | grep "\.docx\$"
 - Pipes the output of “ls -l” to “grep “\.\.docx\$”” which finds the occurrences with “.docx”.
- You can redirect output into files or redirect input from a file into a command/program/file.
- You can automate tasks using **scripts**.

```
asajenkins@Asas-MacBook-Air ~/Desktop/myWork
% vi hello-world.sh
1 # HEADER
2 #!/bin/bash
3
4 # VARIABLE
5 STRING="HELLO WORLD"
6
7 # PRINT VARIABLE
8 echo $STRING
asajenkins@Asas-MacBook-Air ~/Desktop/myWork
% chmod u=rwx hello-world.sh
asajenkins@Asas-MacBook-Air ~/Desktop/myWork
% ./hello-world.sh
HELLO WORLD
```

SHELL BUILT-IN COMMANDS

- Built-in commands are contained within the shell itself.
- When built-in commands are used in *simple commands*, the shell executes the command directly, without creating a new process.
- Built-in commands are necessary to implement functionality impossible or inconvenient to obtain with other utilities.



A star (*) next to a name means that the command is disabled.

```
JOB_SPEC [&] (( expression ))
. filename [arguments] :
[ arg... ] [[ expression ]]
alias [-p] [name[=value] ... ] bg [job_spec ...]
bind [-lpvsPVS] [-m keymap] [-f fi break [n]
builtin [shell-builtin [arg ...]] caller [EXPR]
case WORD in [PATTERN [| PATTERN]. cd [-L|-P] [dir]
command [-pVv] command [arg ...] compgen [-abcdefgjksuv] [-o option
complete [-abcdefgjksuv] [-pr] [-o continue [n]
declare [-afFirtx] [-p] [name[=val dirs [-clpv] [+N] [-N]
disown [-h] [-ar] [jobspec ...] echo [-neE] [arg ...
enable [-pnds] [-a] [-f filename] eval [arg ...
exec [-cl] [-a name] file [redirec exit [n]
export [-nf] [name[=value] ...] or false
fc [-e ename] [-nlr] [first] [last fg [job_spec]
for NAME [in WORDS ... ;] do COMMA for (( exp1; exp2; exp3 )); do COM
function NAME { COMMANDS ; } or NA getopt optstring name [arg]
hash [-lr] [-p pathname] [-dt] [na help [-s] [pattern ...
history [-c] [-d offset] [n] or hi if COMMANDS; then COMMANDS; [ elif
jobs [-lnprs] [jobspec ...] or job kill [-s sigspec | -n signum | -si
let arg [arg ...] local name[=value] ...
logout popd [+N | -N] [-n]
printf [-v var] format [arguments] pushd [dir | +N | -N] [-n]
pwd [-LP] read [-ers] [-u fd] [-t timeout] [
readonly [-af] [name[=value] ...] return [n]
select NAME [in WORDS ... ;] do CO set [--abefhkmnpptuvxBCHP] [-o opti
shift [n] shopt [-pqsu] [-o long-option] opt
source filename [arguments] suspend [-f]
test [expr] time [-p] PIPELINE
times trap [-lp] [arg signal_spec ...]
true type [-afptP] name [name ...]
typeset [-afFirtx] [-p] name[=valu ulimit [-SHacdfilmnpqstuvwxyz] [limit
umask [-p] [-S] [mode] unalias [-a] name [name ...
unset [-f] [-v] [name ...] until COMMANDS; do COMMANDS; done
variables - Some variable names an wait [n]
while COMMANDS; do COMMANDS; done { COMMANDS ; }
bash-3.2$
```

BUILT-IN COMMAND EXAMPLES

- Bourne Built-in commands
 - cd - Change directory
 - pwd - Print working directory
 - readonly - set something to read only or CONST
- Bash Built-in commands examples:
 - logout - logout current shell
 - echo - like cout a print statement
 - read - like cin a stdin
- Special Built-in commands:
 - hash - hash command maintains a hash table, which has the used command's path names
 - exec - Set current shell to program
 - eval - Evaluates several commands and arguments

SIMPLE COMMANDS

- Simple commands are usually followed by arguments specified by the user.
- These are referred to as simple commands, because they're usually the ones executed more frequently.
- The first word generally specifies a command which is followed by arguments.
 - chmod u=rwx test1.txt
 - ls -l
 - cd ..

PIPELINES

- A *pipeline* is the sequence of commands divided by (|) symbols.
- The output of each command is sent as input into the next command.
 - ls -l | grep "example"
 - more test1.txt | grep -i "hello"

REDIRECTION

- Command `> output_file`
 - Redirects command into a file.
- Command `< input_file`
 - Redirects input into a command/program/file
- Command `< input_file > output_file`
 - Redirects input into a command/program/file, and redirects the output of that into a file

LISTS

- A *list* is a sequence of one or more pipelines separated by one or more `;` , `&` , `&&` , or `||` .
- Optionally terminated by `;` , `&` , or newline.
 - command1 && command2
 - command1 || command2

ARITHMETIC EXPANSION

Operator	Meaning
VAR++ and VAR--	variable post-increment and post-decrement
++VAR and --VAR	variable pre-increment and pre-decrement
- and +	unary minus and plus
! and ~	logical and bitwise negation
**	exponentiation
*, / and %	multiplication, division, remainder
+ and -	addition, subtraction
<< and >>	left and right bitwise shifts
<=, >=, < and >	comparison operators
== and !=	equality and inequality
&	bitwise AND
^	bitwise exclusive OR
	bitwise OR
&&	logical AND
	logical OR
expr ? expr : expr	conditional evaluation
=, *=, /=, %=, +=, -=, <<=, >>=, &=, ^= and =	assignments
,	separator between expressions

SYNTAX

- Variables
 - Bash variables does not need a type.
 - Use 'declare' to initialize later.
 - Ex. declare -I number
 - Bash variables are mutable.
 - Important to note (=) symbol has no space between variable and message.
- Conditionals
 - If / then / elif / else / fi
 - Conditionals are similar in evaluation as in c++.
 - Conditionals are set by flags.
 - Example, -gt = greater than

```
bash-3.2$ egg="hello world"
bash-3.2$ echo $egg
hello world
```

```
if [ condition1 ]
then
    command1
    command2
    command3
elif [ condition2 ]
# Same as else if
then
    command4
    command5
else
    default-command
fi
```

SYNTAX CONT.

- Loops
 - For loop
 - While loop
 - Until loop
- Arrays
- Functions

```
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]; do
    echo The counter is $COUNTER
    let COUNTER=COUNTER+1
done
```

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]; do
    echo COUNTER $COUNTER
    let COUNTER-=1
done
```

```
#!/bin/bash
function quit {
    exit
}
function hello {
    echo Hello!
}
hello
quit
echo foo
```

```
## declare an array variable
declare -a arr=("element1" "element2" "element3")

## now loop through the above array
for i in "${arr[@]}"
do
    echo "$i"
    # or do whatever with individual element of the array
done

# You can access them using echo "${arr[0]}", "${arr[1]}" also
```

```
#!/bin/bash
for i in $( ls ); do
    echo item: $i
done
```

SO WHY USE BASH?

1. Very useful in day to day tasks.
2. Automates tasks.
3. Speeds up productivity.
4. Very handy for file manipulation.

FIN

