Author: Martin von Böhlen, ©2022

bash them all

What is bash

Cite from https://en.wikipedia.org/wiki/Bash_(Unix_shell)

Bash is a command processor that typically runs in a text window where the user types commands that cause actions. Bash can also read and execute commands from a file, called a shell script. Like most Unix shells, it supports filename globbing (wildcard matching), piping, here documents, command substitution, variables, and control structures for condition-testing and iteration. Bash is a POSIX-compliant shell, but with a number of extensions.

The shell's name is an acronym for Bourne Again Shell, a pun on the name of the Bourne shell that it replaces and the notion of being "born again".

Why use bash

Use bash to become compatible to the whole world of IT. Because bash is everywhere:

- Under *Windows* as part of tried and tested *Cygwin* (see https://www.cygwin.com) or even stand alone (https://itsfoss.com/install-bash-on-windows/) but only with *Windows Subsystem for Linux (WSL).*
- Under *OS-X* it was the default login shell since initial version. But has been replaced by zsh ever since. Though it remains available as an alternative to zsh.
- There is a bash for Android.
- Apple calls its bash for iOS “a-Shell” (see https://apps.apple.com/us/app/a-shell/id1473805438)Under Linux it is the default login shell for all major and most minor distributions.
- All surviving UN*X have bash.

How to use bash

Bash executes commands typed in by a user, or read in from a file. Commands usually have arguments. Arguments are separated by blanks. Arguments with a leading ‘-’ are called ‘options’. F.e.: *ls -l* calls a program called ‘ls’ with an option argument ‘-l’.

Each program called by bash inherits three files with file number from 0 to 2. These files are:

0. `stdin  = standard input`
1. `stdout = standard output`
2. `stderr = standard error`

The standard output of a program can be connected to the standard input of another program by using pipes (pipelines). Pipes are denoted by a vertical bar ‘|’. Thus, *ls | cat* would redirect the output of ‘ls’ into the input of ‘cat’. Note that often but not always a ‘-’ instead of a file name means stdout.

Bash resolves a thingy called ‘regular expressions’. For now it is sufficient to regard them as wildcards. F.e.: *cat *.sh* prints out the content of all files ending with ‘.sh’ in the current directory.

This is an important difference between bash and Windows! Under Windows’ *cmd* a program resolves its wild cards on its own. Under *bash* a program is called with already resolved arguments.

Useful bash commands

For every command there should be a manual entry. You can read it by typing *man <command>*. Sadly, manual pages are missing sometimes, depending on deployment. Then there should be at least a <command> --help option providing a terse help. Or in the most dire straits a *<command> -h* option.

The following commands should provide a working basis for beginners:

Command	**Efect**
cat	Print out files. W.o. parameters it echoes stdin to stdout.
cd	Change directory
chmod	Change access mode of file/dir
chown	Change ownership of file/dir
cmp	Compare files
cp	Copy files/dirs
date	Display/set date
grep	Get pattern matching lines from file
gzip	Compress/decompress files with Ziff-Lempel
ls	Show content of directories
mkdir	Make directory
mv	Move/rename files/dirs
passwd	New password for me
rm	Remove files/dirs
rmdir	Remove empty dir
sha1sum	Get checksum
sha256sum	Get checksum
sha512sum	Get checksum
sleep	Sleep at least n seconds (maybe more)
su	Switch user (mostly replaced by sudo)
sudo	Super user do
tar	Archive manager
tee	Put a tee in a pipe.
time	Measure command execution time
top	Show process list
touch	Update date of file (create file if n/a)
unzip	Unzip Windows Zip archive
vi	The only true editor
wc	Word count
wget	Snatch internet page/file
zcat	Uncompress and cat .gz