Bash, a command line interface for interacting with the operating system, was created in the 1980s. Other popular shells are zsh and fish.

1 Programming in Bash

1.1 Shebang

The shebang (`#!`) at the head of a script indicates an interpreter for execution, as in `#!/bin/bash`. Lines starting with a `#` (with the exception of shebang) are comments and thus won't be executed.

1.2 Quoting and literals

Single quotes '' preserve the literal value of characters enclosed within them. A single quote may not appear between single quotes, even when escaped, but may appear between **double quotes** `""`. They work similarly, with an exception that the shell expands any variables that appear within them.

1.3 Variables

Variable names are case sensitive. They can contain digits and underscores as well, but a name starting with a digit is not allowed. Example:

```
var="kind"
echo ${var}ness # kindness
```

Special variables:

1. `$0`: name of the script itself. `$1`, `$2`, `$3`, ...: the first, second, etc. argument. `shift` removes first argument and advances rest of them forward.
2. `$*` and `$@` denote all the positional parameters.
3. `$#`: the number of positional parameters
4. `$?`: most recently executed command exit status.
5. `$$`: the process ID of the shell.
6. `$!`: the process ID of the most recently executed command.

It reads a line of input from standard input or a file passed as an argument to its `-u` flag, and assigns it to a variable. In Linux based shells, like Bash, it is present as a shell built in function, and not as a separate executable file.[1]

To read a line of input:

★ **read** reads a line from the stdin and splits it into fields:
`n` returns after reading n characters,
`n` displays a prompt

1.4 Streams

There are always three default files open:

1. *stdin* (the keyboard, file descriptor 0),
2. *stdout* (the screen, file descriptor 1) and
3. *stderr* (error messages output, file descriptor 2).

These **streams** can be **redirected**:

- `cmd > file` redirects to a file (overwrites),
- `cmd >> file` appends instead,
- `m>n` (or `m>&n`) redirects a file descriptor to a file (or another file descriptor),
- `&>file` redirects both stdout and stderr to a file;
- `:> file` truncates file to zero length and
- `|` (pipe) serves as a command chaining tool.

1.5 Control flow statements

The one-line constructs `&&` and `||` work like logical and $\wedge$ and logical or $\vee$.

1.5.1 Conditionals

Here at least one statement must be specified inside every block, but one can use a single colon (`:`) as a null statement to avoid rewriting the code.

```
if condition; then
  commands
elif second_condition; then
  some_commands
else
  other_commands
fi

select word in "foo" "bar" "baz"
do
  echo "Your word is $word".
done
```

There is also a case instruction:

```
case $fruit in
  banana)
    echo "Bananas are awry."
    ;;
  orange|apple)
    echo "Ugh..."
    exit 1
    ;;
  *)
    echo "Unknown fruit!"
esac
```

1.5.2 Testing conditions

Remember that `test` command follows symbolic links (except for the `-h` test).

- **File tests**:
  1. `-e` file exists, `-s` file is nonempty.
  2. `-d` directory, `-f` regular file, `-h` symbolic link.
  3. `-b` block device, `-c` character device, `-p` named pipe, `-S` socket.
- **File permissions**:
  1. `-r` readable, `-w` writable, `-x` executable,
  2. `-u` setuid, `-g` setgid, `-k` sticky bit.
- **String tests**: `-z` empty, `-n` nonempty.
- **Arithmetic tests**: `-eq` $=$, `-ne` $\neq$, `-lt` $<$, `-gt` $>$, `-le` $\leq$, `-ge` $\geq$.

1.5.3 Loops

```
for var in "the first" "the second"; do
  echo "${var}"
done

for (( i = 1; i <= 10; i++ )); do
  echo "i = ${i}."
done # C-style

while read myline; do
  echo "It says ${myline}"
done < some_file
```

As Bash Guide for Beginners by M. Garrels says:

1. the `break` statement is used to exit the current loop before its normal ending.
2. the `continue` statement resumes iteration of an enclosing while, until, select or for loop.

1.6 Regular expressions

- POSIX character classes:
  - `[:alnum:]` = `[a-zA-Z0-9]`
  - `[:alpha:]` = `[a-zA-Z]`
  - `[:ascii:]` = `[\x00-\x7F]`
  - `[:blank:]` = `[ \t]`
  - `[:cntrl:]` = `[\x00-\x1F\x7F]`
  - `[:digit:]` = `[0-9]`
  - `[:graph:]` = `[\x21-\x7E]`
  - `[:lower:]` = `[a-z]`
  - `[:print:]` = `[\x20-\x7E]`
  - `[:space:]` = `[ \t\r\n\v\f]`
  - `[:word:]` = `[A-Za-z0-9_]`
  - `[:xdigit:]` = `[A-Fa-f0-9]`
- Repetitions:
  - `*`: 0 or more, `+`: 1 or more, `?`: 0 or 1,
  - `{a, b}`: at least a, at most b.
- Anchors:
  - `^`: start of line,
  - `$`: end of line,
  - `\<`: start of word,
  - `\>`: end of word.
- Other:
  - `one|two`: one or two,
  - `(one)`: a group,
  - `$n`: nth group,
  - `[abcd]`, `[a-d]`: ranges,
  - `[^abcd]`: negation (not `[abcd]`).

1.7 sed – stream editor

The default delimiters is `/`, but any other will work too. Some sed commmands:

1. to substitute (s) strings globally (g):
   `sed 's/foo/bar/g'`
2. to substitute only in lines containing `baz`:
   `sed '/baz/s/foo/bar/g'`
3. to substitute only in lines without `baz`:
   `sed '/baz/!s/foo/bar/g'`
4. to use regular expressions:
   `sed 's/[0-9]\+/(&)/g'`
5. to delete the first 10 lines:
   `sed '1-10d'`
6. to delete the last line:
   `sed '$d'`
7. to edit files in place:
   `sed -i 's/foo/bar/g' *.txt`

– `n` suppresses automatic printing of pattern space:

8. `sed -n '45,50p'` ... prints lines 45th to 50th.
9. `sed -n 'FLEX.\{65\}/p'` .. prints lines of 65 characters or more
10. `sed -n 'FLEX.\{65\}/!p'` . prints lines of 64 characters or less

1.8 awk – Aho, Weinberger, Kernighan

Awk is a pseudo-C interpretor. General form of its code:

```
BEGIN {initialization}
search pattern {actions}
END {final actions}
```

Examples of search patterns:

1. `/word[0-9]+/`: regular expressions,
2. `!/word[0-9]+/`: negations of these,
3. `$1 ~ /a/`: matches or does not (`!~`) lines with a,
4. `length($0) > 18`.

Important variables:

1. **FS**: field separator (tab and space by default),
2. **OFS**: output field separator,
3. **RS**: record separator (new line),
4. **NR**: number of the current record,
5. **NF**: number of fields in the current record.

2 Emacs shortcuts in Bash

1. `Ctrl A` moves to the start of the line,
2. `Ctrl E` moves to the end of the line,
3. `Ctrl U` deletes to the beginning of the line.
4. `Ctrl K` deletes to the end of the line.
5. `Ctrl W` deletes to the start of the word.
6. `Ctrl Y` pastes text from the clipboard.
7. `Ctrl L` clears the screen.
8. `Alt R` undoes all changes to the line.
9. `Ctrl R` searches incrementally up the history.
10. `Ctrl XE` invokes an editor to write complex command.

3 Unix utilities and shell builtins

3.1 File system

■ **cat** concatenates and prints files:
`A` shows all nonprinting characters,
`b` numbers nonempty output lines,
`n` numbers all output lines,
`s` suppresses repeated empty output lines.
■ **tac** does the same in reverse.
★ **rev** reverses lines characterwise.
■ **nl** numbers lines of files:
`s` adds "string" after line number,
`w` uses "number" columns for line numbers.

■ **chgrp** changes group ownership.
■ **chmod** changes permissions of a file:
`ugoa` permissions of the owner, group, other/all users,
`+-=` adds, removes or sets selected file mode bits,
`rwx` selects file mode bits: read/write/execute (4/2/1).
■ **chown** changes owner of a file.
★ **umask** sets file mode creation mask.
■ **touch** changes file timestamps:
`a` only the access time,
`m` only the modification time,
`t` uses custom stamp instead of current time,
`c` does not create files.

■ **shasum** prints or checks SHA message digests:
`a` algorithm: 1, 224, 256, 384, 512, 512224 or 512256,
`b` reads in binary mode,
`c` checks SHA sums read from the "files".
■ See also **cksum** (CRC checksums) and **md5sum**.
■ **wc** prints newline, word and byte counts (`lwc`):
`m` prints the character counts,
`L` prints the maximum display width.

■ **dd** converts and copies a file:
`if=` reads from a file instead of standard input,
`of=` writes to a file insteaed of standard output,
`bs=` up to "bytes" bytes at a time,
`count=` copies only "n" input blocks.

■ **cp** copies files and directories:
`b` makes a backup of each existing destination file,
`f` removes an existing destination file if needed,
`i` prompts before overwrite,
`n` does not overwrite existing files,
`L` always follows symlinks in "source",
`P` never follows symlinks in "source",
`p` preserves timestamps, mode, ownership,
`r` copies directories recursively,
`s` makes symbolic links instead,
`l` hard links files instead,
`t` copies all "source" arguments into "directory",
`T` treats "destination" as a normal file,
`u` copies only newer source files,
`v` explains what is being done.
■ **mv** moves (renames) files:
`b` makes a backup of each existing destination file,
`i` prompts before overwriting,
`f` does not prompt before overwriting,
`n` does not overwrite existing destination files.
`t` moves all "source" arguments into "directory",
`T` treats "destination" as a normal file,
`u` moves only newer source files,
`v` explains what is being done.
■ **rm** removes files or directories:
`f` never prompts,
`i` always prompts,
`r` removes directories and their contents.
■ See also **rmdir** (directories removal) and **shred**.
■ **mkdir** makes directories (`mkdir p`: with parents as needed, no error if existing).

■ **df** reports file system disk space usage:
`h` prints size in powers of 1024,
`i` list inode information instead of block usage,
`t` limits listing to file systems of given type,
`x` limits listing to file systems not of given type,
`T` prints file systems types.

■ **du** estimates file space usage:
`a` writes counts for all files, not just directories,
`c` produces a grand total,
`d` the depth at which summing should occur,
`h` prints sizes in human readable format,
`s` diplays only a total,
`X` excludes files that match pattern.

★ **file** determines file type.

★ **find** searches for files in a directory hierarchy.
1. Tests:
   - `-name` base of file name,
   - `-iname` case insensitive name,
   - `-group`, `-user` ownership
   - `-perm 755`, `-perm /u=x` permissions
   - `-size +5M -1G` size between 5MB and 1GB
   - `-amin -60` accessed in last hour
   - `-cmin`, `-mmin`: created, modified,
   - `-mtime +7` modified over a week ago
   - `-type d` directories only,
   - `-type f` files only,
   - `-empty` empty files or directories only,
2. Example (deletes files larger than 5 megabytes):
   - `find . -size +5M -exec rm -f \;`

★ **fsck** checks and repairs a Linux filesystem:
`a` automatically repairs (without any question!),
`t` specifies the type(s) of filesystem to be checked,
`A` tries to check all filesystems in one run,
`M` skips mounted filesystems,
`R` skips the root filesystem.

■ **ln** makes hard links between files (not directories; only in the same file system):
`s` makes symbolic links instead.

■ **ls** lists directory contents:
`a` does not ignore entries starting with dot,
`F` appends indicator to entries,
`h` prints human readable sizes,
`i` prints the index number of each file,
`l` prints permissions, number of hard links, owner, group, size, last-modified date as well,
`r` reverses order while sorting,
`R` lists subdirectories recursively,
`S` sorts by file size (largest first),
`t` sorts by modification time (newest first),
★ **tree** lists tree-like contents of directories.

★ **mount** mounts a filesystem.

■ **pwd** prints name of current directory.

★ **tar** stores and extracts files from a disk archive:
`c` creates a new archive,
`x` extracts files,
`t` lists the contents of an archive,
`v` verbosely lists files processed,
`j` bzip2 compression,
`z` uses zip/gzip (gz compression),
`f` uses archive file or device (???),
`k` does not replace existing files when extracting.

■ **pv** monitors the progress of data through a pipe.

■ **tee** duplicates pipe content:
`a` appends to the given files, does not overwrite,
`i` ignores interrupts.

★ Missing: **cmp**, **fuser**, **pax**, **type**.

3.2 Processes

■ **chroot** changes the root directory for the current running process and their children.

★ **at** schedules commands to be executed once, at a particular time in the future: it accepts times of the form `HH:MM`, `midnight`, `noon` or `teatime`; `MMDD[CC]YY`, `MM/DD/[CC]YY`, `DD.MM.[CC]YY` or `[CC]YY-MM-DD` (the specification of a date must follow the specification of the time of day). You can also give times like `now + 3 hours`.

★ **bg** resumes suspended jobs in the background.
★ **fg** resumes suspended jobs in the foreground.
★ **jobs** lists the active jobs.
★ **command &** runs command in the background.

★ **cron**: a daemon executing scheduled commands.
★ **crontab** maintain individual users' crontab files.

★ **kill** sends a `TERM` signal to a process.
★ **killall** kills processes by name.

★ **ps** reports a snapshot of the current processes:
`e` selects all processes,
`f` does full-format listing,
`C` selects processes by command name,
`p` selects processes by PID,
`u` selects processes by EUID or name.
★ **pstree** displays a tree of processes.

★ **nice** changes process priority.

★ **pgrep**, **pkill** looks up or signals processes based on name and other attributes.

★ **time** runs programs and summarizes system resource usage.

★ **top** displays linux processes.

3.3 User environment

★ **clear** clears the terminal screen.
★ **env** runs a program in a modified environment.
★ **exit** terminates the calling process.
★ **finger** looks up user information.
★ **history** displays the history list.
★ **mesg** displays messages from other users.

★ **passwd** changes user password:
`d` deletes an account's password (makes it empty),
`e` expires an account's password,
`n` minimum days to change password,
`w` warning days before password expire,
`x` maximum days a password remains valid.
★ **pwgen** generate pronounceable passwords:
`s` generates hard to memorize passwords,
`y` includes special characters,
`n` includes numbers,
`N` generates "num" passwords

★ **su** changes user ID or becomes superuser.
★ **sudo** executes a command as superuser:
`u` as a different user.

★ **hostname** shows/sets the host name:
`i` displays the network address.
★ **uname** prints system information:
`a` all information, in the following order:
`s` the kernel name,
`n` the network node hostname,
`r` the kernel release,
`v` the kernel version,
`m` the machine hardware name,
`p` the processor type,
`i` the hardware platform,
`o` the operating system.

★ **uptime**: how long has the system been running?

★ **wall** writes a message to all users,
★ **write** sends a message to another user.

★ **who** shows who is logged on,
★ **w** does the same and shows what they are doing,
★ **whoami** prints effective userid.

3.4 Text processing

★ **awk** is a pattern scanning / processing language.
★ **grep** prints lines matching a pattern:
`c` prints a count of matching lines instead,
`e` uses a "regexp" pattern,
`f` obtains patterns from a file,
`i` ignores case disctinctions,
`v` inverts the sense of matching,

`w` selects only lines containing matches that form whole words,
`n` prints line numbers as well,
`A` prints "num" lines of trailing content,
`B` prints "num" lines of leading content,
`C` prints "num" lines of both contents,
`R` reads all files under each directory.
★ **sed**: a stream editor filtering/transforming text.

■ **comm** compares two sorted files line by line.
■ **shuf** generates random permutations:
`e` treats each "arg" as an input line,
`i` treats each number .. through .. as an input line,
`n` outputs at most "count" lines,
`r` output lines can be repeated (with `-n`).
■ **sort** sorts lines of text files:
`c` checks for sorted input,
`f` folds lower case to upper case characters,
`g` compares general numerical values,
`h` compares human readable numbers,
`k` sorts via a key,
`n` compares string numerical values,
`r` reverses the results,
`s` stabilizes the sort.
■ **tsort** performs topological sort.
■ **uniq** omits repeated lines:
`c` prefixes lines by the number of occurences,
`d` only prints duplicate lines, one for each group,
`f` avoids comparing first fields,
`i` ignores differences in case,
`s` avoids comparing first characters,
`w` compares no more than n characters.

■ **cut** prints selected parts of lines:
`--complement` complements the selection,
`c` selects only these characters,
`d` uses "delim" instead of Tab for field delimeter,
`f` selects only these fields,
`s` does not print lines not containing delimeters.
■ **join** joins lines of two files on a common field.
■ **paste** merges lines of files.
`d` reuses characters from "list" instead of tabs,
`s` pastes one file at a time, not in parallel.
■ **tr** translates or deletes characters:
`c` uses the complement of "set1",
`d` deletes characters, does not translate,
`s` replaces each sequence of a repeated character that is listed in the last specified "set" with a single occurrence of that character.

★ **diff** compares files line by line:
`y` outputs in two columns,
`i` ignores case differences,
`w` ignores all white space.

★ **fmt** is a simple optimal text formatter,
★ **fold** wraps each line to fit in specified width.

■ **head** outputs the first (last) part of files:
`c` the first "num" bytes,
`n` the first "num" lines,
■ **tail** the last "num" bytes:
`c` the last "num" bytes,
`n` the last "num" lines,
`f` outputs appended data as the file grows,
`s` sleeps for "n" seconds between iterations.
■ **split** splits a file into pieces:
`a` generates suffixes of length "n" (default 2),
`b` puts "size" bytes per output file,
`d` uses numeric (not alphabetic) suffixes,
`l` puts "number" lines/records per output file,
`n` generates "chunks" output files.
■ See also: **csplit**.

★ **more** pages text too large to fit on one screen and allows scrolling down, but not up and therefore is deprecated.
★ **less** is an enhanced version of more:
`+F` monitors the tail of a file which is growing.

★ **vim** is an advanced text editor, too complex to be explained here. See also **emacs**.

★ **xargs** builds and executes command lines:
`0` takes care of filenames with spaces, backslashes.
`I` replaces occurrences of "string" with names read from standard input.

★ **yes** outputs a string repeatedly until killed.

3.5 Shell builtins

★ **alias** allows a string to be substituted for a word.
★ **cd** changes the shell working directory:
`-` to the previous directory.
★ **echo*** displays a line of text:
`e` enables interpretation of backslash escapes,
`n` does not output the trailing newline.
★ **test** checks file types and compares values.
★ **unset** unsets a shell variable, removing it from memory and the shell's exported environment.
★ **wait** waits for process to change state.

3.6 Networking

★ **curl** transfers a URL.
★ **dig** interrogates DNS name servers.
`x` performs a simplified reverse lookup.
★ **host** is a DNS lookup utility.
★ **ifconfig** configures a network interface.
★ **inetd** is a super-server daemon that provides Internet services.
★ **netcat**: arbitrary TCP and UDP connections and listens.
★ **netstat** prints network connections, routing tables, interface statistics, masquerade connections, and multicast memberships.
★ **nslookup** queries Internet name servers interactively.
★ **ping** tests the reachability of a host on an IP network by sending ICMP ECHO_REQUEST:
`c` stops after sending "count" packets,
`n` numeric output only, avoids to lookup symbolic names for host addresses.
★ **rdate** sets the system's date from a remote host.
★ **rlogin** starts a terminal session on a remote host.
★ **rsync** copies files fast (remote or local):
`a` in archive mode, equivalent to:
`g` preserves group,
`o` preserves owner (super-user only)
`p` preserves permissions,
`t` preserves modification times,
`l` copies symlinks as symlinks,
`b` make backups,
`c` skip based on checksum,
`n` performs a dry run without changes made,
`r` resursively,
`u` skip newer files on the receiver,
`v` increases verbosity,
`z` compresses file data during the transfer,
`--delete` deletes extraneous files from dest dirs.
★ **route** shows and manipulates the IP routing table.
★ **ssh** is an OpenSSH SSH client (remote login program).
`D` (bind address)
`p` (port)
`X` (X11 forwarding)
★ **traceroute** is a computer network diagnostic tool for displaying the route (path) and measuring transit delays of
★ **wget** is a non-interactive network downloader.
`A,R` specifies lists of file suffixes or patterns (when wildcard characters appear) to accept or reject,
`b` goes to background immediately after startup,
`c` continues getting a partially-downloaded file,
`m` turns on options suitable for mirroring: infinite recursion and time-stamping,
`np` does not ever ascend to the parent directory when retrieving recursively,
`U` identifies as "agent-string" to the HTTP server.
`w` waits the specified number of seconds between the retrievals (see also `-random-wait`).

3.7 Searching

★ **find** searches for files in a directory hierarchy.
★ **locate** finds files by names.
★ **updatedb** updates the file database used by locate.
★ **whatis** displays one-line manual page description.
★ **whereis** locates the binary, source, and manual page files for a command.

3.8 Hardware

★ **dmesg** prints/controls the kernel ring buffer.
★ **lsblk** lists block devices.
★ **lsof** lists open files.
★ **lsusb** listsq USB devices.

3.9 For programmers

■ **g++** compiles, assembles and links C++ files:
`o` writes the build output to a file named …

3.10 Miscellaneous

★ **bc** is an arbitrary precision calculator language.
1. `echo 'obase=16;255' | bc` prints `FF`,
2. `echo 'ibase=2;obase=A;10' | bc` prints `2`,
3. `scale=10` (after `bc -l`) sets working precision.

★ **dc** is a reverse-polish desk calculator. One of the oldest Unix utilities, predating even the invention of the C programming language.
★ **cal**, **ncal** displays a calendar.
`e` displays date of Easter,
`j` displays Julian days,
`m` displays the specified month,
`w` prints the numbers of the weeks,
`y` displays a calendar for the specified year,
`3` displays the previous, current and next month.
★ **date** prints or set the system date and time.
■ **seq** prints a sequence of numbers:
`w` equalizes width by padding with leading zeroes.
■ **sleep** delays for a specified amount of time.
★ **true**, **false** does nothing, (un)successfully.