

This sheet "BASH 3.2" is assembled by kg with ©, August 15, 2007. Most recent version is here:

http://www.oreilly.de/catalog/bashtbger/chapter/bashref_english.pdf

Most relevant reference is bash.1 and bashref.info.

Remark: **Options** shown like this (option in option: [option]), **\$Variables** like this. Necessary spaces marked with `_`. "cmd" means "command".

COMMAND LINE OPTIONS

The most imported cmd line options:

- c *string* Execute string, then exit.
- i Behave like interactive shell.
- l, --login Start like login shell.
- r, --restricted Restricted shell (no cd, absolute cmd paths, etc).
- s Read cmds from standard input.
- D, --dump-po-strings, --dump-strings Dump lang. transl. strings.
- init-file *file*, --rcfile *file* Read given init *file*.
- noprofile, --norc Ignore all profiles.
- verbose Be verbose, print shell input lines as they are read.
- noediting Disable readline library if shell is interactive.
- posix Behaves like POSIX 1003.2 standard says.
- No more options follows.
- +O *shopt_option* Activate(-) or deactivate(+) *shopt_option*, see below.
- debugger Turn on extended debugging mode.
- version Show version information.
- help Display a usage message.

STARTUP

Bash reads and executes cmds from the first existing readable file:

Interactive login shell uses the first accessed file (in this order):

`/etc/profile`, `~/.bash_profile`, `~/.bash_login`, `~/.profile`

Interactive no login shells uses: `/etc/bash.bashrc`, `~/.bashrc`

Non-interactive shell (script):

```
if [ -n "$BASH_ENV" ]; then . "$BASH_ENV"; fi
```

In posix mode, interactive shells: expand `$ENV` and read `initfile`.

Remote shell while startup: `/etc/bash.bashrc`, `~/.bashrc`

Relative cmd line options:

- norc, --noprofile Disable reading profiles.
- rcfile *file*, --init-file *file* Use alternative init *file*.

Executed while exiting login shell: `~/.bash_logout`

Individual readline initialization file: `~/.inputrc`

COMPOUND COMMANDS

More than one cmd per line:

`cmd1 ; cmd2` (called "list")

Two special forms called "short circuit test":

`cmd1 && cmd2` Executes `cmd2` only if `cmd1` returns exit code 0.

`cmd1 || cmd2` Executes `cmd2` only if `cmd1` returns exit code NOT equal 0.

(*list*) Execute *list* in a subshell.

{ *list* ; } Execute *list* in current shell (called "group").

((*expression*)) Arithmetic evaluation of *expression*, like `let`.

[[*expression*]] Return status evaluation of *expression*, like `test`.

(*expression*) Return value of *expression*.

! *expression* True if *expression* is false.

expression1 && *expression2* True if both *expressions* are true.

expression1 || *expression2* True if one *expression* is true.

SHELLOPTS

controlled by `set`:

Syntax: `set --abefhkmnp tuvxBCHP -o option arg ...`

Current option set is in `$-`. Some options may be set with short form (`-a`) or also long form (`-o allexport`). Using `+` rather than `-` causes these options to be turned off. The options are:

- a, **allexport** Automatically mark variables and functions which are modified or created for export to the environment of subsequent cmds.
- b, **notify** Report status of background jobs immediately.
- e, **errexit** Exit immediately if simple cmd exits with a non-zero status.
- f, **noglob** Disables pathname expansion (eg `*` is not special).
- h, **hashall** Remember location of cmds as they are looked up for execution.
- k, **keyword** Place arguments as assignments in the environment.
- m, **monitor** Background processes in separate process group, print exit status.
- n, **noexec** Read cmds but do not execute them, ignored by interactive shells.
- p, **privileged** Privileged mode, `$ENV` and `$BASH_ENV` files are not processed, shell functions not inherited from the environment, and `$SHELLOPTS`, if it appears in the environment, is ignored. Turning this option off causes the effective user and group ids to be set to the real user and group IDs.
- t, **onecmd** Exit after reading and executing one cmd.
- u, **nounset** Unset variables treated as error while parameter expansion.
- v, **verbose** Print shell input lines as they are read.
- x, **xtrace** Display cmd and expanded arguments after expanding simple cmds and `for`, `case`, `select`, preceded by `$PS4`.
- B, **braceexpand** Perform brace expansion, on by default.
- C, **noclobber** Do not overwrite existing files with `>`, `>&`, and `<>`, only by `>|`.
- E, **errtrace** Inherit ERROR-trap by functions, cmd substitutions, in subshells.
- H, **histexpand** Enable ! history substitution, on by default in interactive shells.
- P, **physical** Do not follow symbolic links when executing `cd`.
- T, **functrace** Inherit DEBUG-trap by functions, cmd substitutions, subshells.
- If no arguments following, positional parameters are unset.
- End of options, all remaining args assigned to positional parameters.
- o *option-name* If no *option-name*, print current values. `+o` with no *option-name* shows cmds to recreate current options. Only long options:
 - emacs** Emacs-style cmd line editing, enabled by default.
 - history** Cmd history, default in interactive shells.
 - ignoreeof** Like `IGNOREEOF=10`.
 - pipefail** If set, the return value of a pipeline is the value of the last (right-most) cmd to exit with a non-zero status, or zero if all cmds in the pipeline exit successfully. Option is disabled by default.
 - posix** Activate posix mode.
 - vi** Use a vi-style cmd line editing interface.

SHELLOPTS

controlled by `shopt`:

`shopt -pqsu -o optname ...`

-s Enable (set) each *optname*.

-u Disable (unset) each *optname*.

-q No output (quiet mode); return status indicates whether *optname* is set or unset.

-o Restricts values of *optname* to those defined `-o` (set builtin).

Available `shopt options` are:

cdable_vars Interpret `cd` arguments as variable, if not directory.

cdspell Allow minor spelling errors in directory names.

checkhash Check existing of cmds found in hash table before execute.

checkwinsize Check window size (update `$LINES/$COLUMNS`) after cmd.

cmdhist Save multiple line cmd in one history entry.

dotglob Include filenames beginning with a "." in the results of pathname expansion.

execfail Non-interactive shell will not exit if unable to execute `exec` argument.

expand_aliases Aliases are expanded, default for interactive shells.

extdebug Enable debugger features. (`-F` option to declare displays source file name and line number. If cmds run by the DEBUG-trap returns non-zero, the next cmd is skipped (not executed). If returning `2`, and the shell is executing in a subroutine, a call to return is simulated.

extglob Enable extended pattern matching features.

extquote Quoting in `'string'` and `"string"` is performed within expansions, enabled by default.

failglob Failing match filenames treated as expansion error.

force_ignore Suffixes in `$IGNORE` causes words to be ignored when performing word completion, enabled by default.

gnu_errfmt Error messages are written in the standard GNU error format.

histappend Append history to file named by `$HISTFILE`, rather overwrite.

histredit If readline is used, history substitutions can be re-edited.

histverify With readline, lines loaded into editing buffer for modifications.

hostcomplete With readline, hostname completion for words with `@`, (default enabled).

huponexit Bash sends SIGHUP to all jobs when interactive login shell exits.

interactive_comments Ignore text after `#` in interactive shells, default.

lithist With `cmdhist`, multi-line cmds saved to history with embedded newlines.

login_shell Shell sets this option if it is started as a login shell.

mailwarn Message, if mailfile is accessed since last check.

no_empty_cmd_completion With readline, no `$PATH`-search for completions on empty lines.

nocaseglob Filename globbing is case-insensitive.

nocasematch Case-insensitive pattern matching in `case/[[[`.

nullglob Patterns which no match expand to null string, rather themselves.

progcomp Enable programmable completion facilities (default).

promptvars Parameter expansion, cmd substitution, arithmetic expansion, and quote removal on prompt string, enabled by default.

restricted_shell Shell is in restricted mode.

shift_verbose Prints error message if `shift` count exceeds `$#`.

sourcepath Uses `$PATH` for `.` and `source` to find file in argument, enabled by default.

xpg_echo Builtin `echo` expands backslash-escape sequences by default.

ALIASES

Aliases substitutes strings for words, if they are first word of simple cmd (lines) in interactive shells or with shell option `expand_aliases`; can not use parameters:

alias `-p` Shows all defined aliases.

alias name='definition' Creates alias `"name"` with `definition`.

unalias name Removes alias `name`.

SHELL FUNCTIONS

Uses parameters, local vars, may be recursive.

create: `function name () {compound-cmd;} redirection`

functions listed with `set`, `declare`, `typeset` (option `-f`)

declare local variables: `local` (options like `declare`)

export to subshell: `export -f`

Return (exit) status: `return status`

PARAMETERS

Expansions of special cmd line args (only for reference):

`$1, $2, $n...` Positional parameter number *n*.

`$0` Name of the shell or shell script.

`$*` Positional parameters, starting from one; in "double quotes" (`"$*"`): single word with all parameters (`"single word... parameters"`).

`$@` Positional parameters, starting from one; in double quotes: parameter expands to separate words (`"$1", "$2", "$3..."`).

`##` Number of positional parameters (decimal).

`$?` Status of most recently executed foreground pipeline.

`$-` Current option flags (specified upon invocation by `set`).

`$$` PID of the current shell.

`$!` Process ID of most recently background cmd.

`$_` Shell startup: absolute pathname of the shell or script subsequent: last argument to previous cmd after expansion.

`set -- list of words` (Re)assign positional parameters (`$1, $2, $3...`) to *list of words*, delimited by first element in `$IFS`.

`${n}` Refers positional parameter *n* or variable *name*.

Indirect Reference: `${ ! Variablename }` or `${ ! positional parameter }`

SHELL VARIABLES (Set by the shell):

\$BASH: Full file name (with path) to invoke this bash.
\$BASH_ARGC: Array, numbers of parameters in frame of call stack.
\$BASH_ARGV: Array, with parameters in current bash call stack.
\$BASH_COMMAND: Cmd currently being executed.
\$BASH_EXECUTION_STRING: Argument to `-c` invocation option.
\$BASH_LINENO array: Line numbers in source files of functions.
\$BASH_REMATCH: Array assigned by `=~` to `[[]]` cmd.
\$BASH_SOURCE: Array of source filenames.
\$BASH_SUBSHELL: Current subshell level.
\$BASH_VERSION: Array containing bash version information.
\$BASH_VERSION: Bash version string.
\$COMP_CWORD: Index in `{ $COMP_WORDS }` of word with cursor.
\$COMP_LINE: Cmd line number (only words/completions).
\$COMP_POINT: Cursor position index relative to beginning of cmd.
\$COMP_WORDBREAKS: Readline word separators.
\$COMP_WORDS: Array of the current cmd line.
\$DIRSTACK: Array of directory stack.
\$EUID: Effective user ID.
\$FUNCNAME: Array of all shell functions in execution call stack.
\$GROUPS: Array of groups of which user is member.
\$HISTCMD: History line number.
\$HOSTNAME: Name of current host.
\$HOSTTYPE: Set to type of machine.
\$LINENO: Current sequential line number within a script/function.
\$MACHTYPE: String describes the system type.
\$OLDPWD: Previous working directory (set by `cd`).
\$OPTARG: Last option argument processed by `getopts`.
\$OPTIND: Index of next argument to be processed by `getopts`.
\$OSTYPE: String that describes the operating system.
\$PIPESTATUS: List of exit status (array) of last foreground pipe.
\$PPID: Process ID of shell's parent.
\$PWD: Current working directory (set by `cd` cmd).
\$RANDOM: Random integer between 0 and 32767.
\$REPLY: Input set by `read` builtin.
\$SECONDS: Number of seconds since shell invocation.
\$SHELLOPTS: Colon-separated list of active shell options.
\$SHLVL: Current shell level.
\$UID: Current user ID.

SHELL VARIABLES (Used by the shell):

\$BASH_ENV: Init file when executing a shell script.
\$CDPATH: Search path for `cd` ("`:`"-separated list of directories).
\$COLUMNS: Terminal width when print select lists.
\$COMPREPLY: Array for possible completions.
\$EMACS: If value starts "`t`" bash disables line editing.
\$FCEDIT: default editor for `fc`.
\$FIGIGNORE: "`:`"-separated list of suffixes for filename completion.
\$GLOBIGNORE: "`:`"-separated list of patterns, ignored by pathname expansion.
\$HISTCONTROL: "`:`"-separated list of values for history saving.
\$HISTFILE: File for cmd history (default: `~/.bash_history`).
\$HISTFILESIZE: Maximum number of lines in history file.
\$HISTIGNORE: "`:`"-separated pattern list for cmd lines saved.
\$HISTSIZE: Number of cmds in history (default: 500).
\$HISTTIMEFORMAT: Format string for time stamps in history.
\$HOME: Home directory of current user.
\$HOSTFILE: File in `/etc/hosts`-format; to complete hostnames.
\$IFS: Internal Field Separators for word/line splitting, default: `space,tab,newline`.
\$IGNOREEOF: Number of EOF ignored before exits (default: 10).
\$INPUTRC: Filename for readline startupfile (def: `~/.inputrc`).

\$LANG: Locale category for category not selected `$LC_*` variable.
\$LC_ALL: Overrides `$LANG` and any other `$LC_*` variable.
\$LC_COLLATE: Collation order used for sorting.
\$LC_CTYPE: Interpretation/behavior of char classes in patterns.
\$LC_MESSAGES: Locale to translate doublequoted strings preceded by `$`.
\$LC_NUMERIC: Locale category for number format.
\$LINES: Column length for print select lists.
\$MAIL: File name, bash informs of the arrival of mail.
\$MAILCHECK: Time (in seconds, default 60) bash checks for mail.
\$MAILPATH: "`:`"-separated list of file names checked for mail.
\$OPTERR: if 1 (default), bash displays error messages by `getopts`.
\$PATH: Cmd search path ("`:`"-separated list of directories).
\$POSIPLY_CORRECT: If exits, bash starts in posix mode.
\$PROMPT_COMMAND: Cmds executed prior primary prompt.
\$PS1: Primary prompt string (default: `\s-\v\$_`).
\$PS2: Secondary prompt string (default: `'>_'`).
\$PS3: Prompt for select cmd (default: `'#?_'`).
\$PS4: Prompt during execution trace (default: `'+_'`).
\$SHELL: Path to shell in environment (default: user login shell).
\$TIMEFORMAT: Format string for timing information.
\$TMOUT: Default timeout for the `read` builtin.
\$TMPDIR: Directory for temporary files for the shell's use.
\$auto_resume: For job control: `exact` or `substring`.
\$command_not_found_handle: Function for unfound cmds.
\$histchars: Characters to control history behavior.

ARRAYS (index starts at zero)

create: `name[index]=value` or: `declare -a name[index]`.
assign: `name[index]=value` or `name=(value1 . . . valuen)`.
reference: `${name[index]}`. If double-quoted, `${name[*]}` expands to single word with the value of each array member (separated by the first character of `$IFS`), and `$name[@]` expands each element to separate word.
`unset` destroys arrays; `unset name[subscript]` destroys appointed elements.

REDIRECTION Creating new file descriptors: `exec descriptor file`

`<` standard input (file descriptor 0).
`>` standard output (file descriptor 1).
`&>` file standard output to file.
`2>` standard error (file descriptor 2).
`>&file` standard error to file.
`2>&1` standard output and standard error.
`n<file` redirection descriptor `n` from file.
`n>file` redirection descriptor `n` to file (allow overwrite), disabled by shell option `noclobber`.
`>|file` overwrites `noclobber`.
`n>>file` redirection descriptor `n` to file (append).
`n<<file` duplicate input file descriptor.
`n>&word` duplicate output file descriptor.
`n<&digit-` moves input file descriptor `digit` to descriptor `n`.
`n>&digit-` moves output file descriptor `digit` to descriptor `n`.
`n<>file` open descriptor `n` for reading and writing.
`<<-word` "here document".
`<<<word` "here strings".

Special redirectings:

`/dev/fd/fd` if `fd` is a valid integer, file descriptor `fd` is duplicated.
`/dev/stdin` file descriptor 0 is duplicated.
`/dev/stdout` file descriptor 1 is duplicated.
`/dev/stderr` file descriptor 2 is duplicated.
`/dev/tcp/host/port` open a TCP connection to socket.
`/dev/udp/host/port` open a UDP connection to socket.

QUOTING is used to prevent expansion of metachars:

`\` Escape character, next char is literal.
`' enclosed chars literal'` All enclosed chars are literal.
`" enclosed chars literal"` Chars literal, except: `$`, ``` (backtick), `\`, `!`.
`$' string'` Allows escape sequences (result is single-quoted, no dollar):

`\a` alert (bell)
`\b` backspace
`\e` escape character
`\f` form feed
`\n` new line
`\r` carriage return
`\t` horizontal tab
`\v` vertical tab
`\\` backslash
`\'` single quote
`\nnn` eight-bit character (octal value)
`\xHH` eight-bit character (hexadecimal value)
`\cx` control-x character

A double-quoted string preceded by a dollar sign (`$`) will cause the string to be translated according to the current locale. If current locale is `C` or `POSIX`, the dollar sign is ignored. If the string is translated and replaced, replacement is double-quoted.

EXPANSIONS Substitution of special chars { , } , ~ and parameters or variables in cmd lines before execution.

Parameter Expansion `${parameter}` Substitut *parameter* value (braces required for positional parameter with more than one digit or if followed by character or whitespace in parameter). Variablenames contains letters, numbers and `_`.

`!` as first char (like `! var`) for variable indirect expansion.
`${parameter:-default}` Substitut *default* if parameter unset or null; does not change *parameter*.
`${parameter:=default}` Assign *default*, if unset or null, assign default to *parameter* (change!).
`${parameter:?message}` Display message if null or unset.
`${parameter:+alt}` Use alt, if parameter null or unset; only test, no substitution!
`${parameter:offset:length}` Expand up to charlength (def: all), starts at offset.
`${!prefix*}`, `${!prefix@}` Expand variables whose names begin with *prefix*, separated by first char of `$IFS`.
`${!name[@]}`, `${!name[*]}` Array: expand to array indices assigned in *name*; no array: expands to 0 if *name* set, else null.
`${#parameter}` Expand to length in characters. If *parameter* is `*` or `@` expand to number of positional parameter. If *parameter* is an array: number of elements.
`${parameter#word}`, `${parameter##word}` pathname expansion, remove word from the left (`#`: minimum, `##` maximum).
`${parameter%word}`, `${parameter%%word}` pathname expansion, remove word from right (`%`: minimum, `%%` maximum).
`${parameter/pattern/string}`, `${parameter//pattern/string}` pathname expansion: replace pattern (only first: `/`, all: `//`) by string.

Tilde Expansion If word begins with an unquoted tilde character (`~`), all of the characters preceding the first unquoted slash (or all characters, if there is no unquoted slash) are considered a *tilde-prefix*.

`~tilde-prefix/` This expands to login name *tilde-prefix*.
`~+n` expands to `$PWD`; with *n* (number): `$DIRSTACK` index *n*.
`~-n` Expands to `$OLDPWD`; with *n* (number): `$DIRSTACK` index *n*.

Brace Expansion Evaluation from left to right: `a{d,c,b}e` expands to `ade ace abe`. A sequence expression takes the form `{x.y}`, where `x` and `y` are either integers or single characters (same type). When integers are supplied, the expression expands to each number between `x` and `y`, inclusive, when characters, to each character lexicographically between `x` and `y`, inclusive. Example: `mkdir ./ {old,new}` or `ls ./ {ucb/{ex,ed},lib/{ex?.?*,hw}}`

ARITHMETIC EXPANSION (Insert result of arithmetic in cmd line; do not use `$(expression)`). `$((expression))` Insert integer result of `expression`. Allowed Operators (shell variables also allowed):

`id++`, `id--` variable post-increment and post-decrement.
`++id`, `--id` variable pre-increment and pre-decrement.
`-`, `+` unary minus and plus.
`!`, `~` logical and bitwise negation.
`**` exponentiation: `$((left ** right))` means *left^{right}*.
`*`, `/`, `%` Multiplication, division, remainder.
`+`, `-` Addition, subtraction.
`<<`, `>>` Left and right bitwise shifts.
`<=`, `>=`, `<`, `>` Comparison; eg `$((3 <= 5))`: 1 if true else 0.
`==`, `!=` Test of equality and inequality: 1 if true else 0.
`&` Bitwise AND.
`^` Bitwise exclusive OR.
`|` Bitwise OR.
`&&` Logical AND.
`||` Logical OR.
`expr?expr:expr` Conditional operator.
`=`, `*`, `/`, `%`, `+`, `-`, `==`, `<<=`, `>>=`, `&=`, `^=`, `|=` assignment.
`expr1`, `expr2` comma.

Constants with a leading `0` interpreted octal, leading `0x` or `0X` denotes hexadecimal, otherwise decimal. Numbers can be in the form `base#n`, where arithmetic base is (decimal, default: `10`) between `2` and `64`, and `n` is a number in that base. Digits greater than `9` are represented by (in that order) lowercase letters, uppercase letters, `@`, and `_`.

WORD SPLITTING `$IFS` contains three word delimiters, default: space, tab, newline (in that order). Bash uses this for parameter expansion, word splitting in cmd lines and for read, to divide array elements, and in programmable completions.

PATHNAME EXPANSION Sorting is language controlled by variable `$LC_COLLATE`, `$GLOBIGNORE` contains ignore pattern, Option `dotglob` includes filenames beginning with `"."`.

PATTERN MATCHING

`*` Means any string, including null string. `?` is any single character, except `"."`.
`[...]` enclosed characters, hyphen denotes a range expression; `!` or `^` (at first): matches none enclosed characters. Option `extglob` enables extended patterns:
`?(pattern-list)` : Zero or one occurrence of patterns.
`*(pattern-list)` : Zero or more occurrences of patterns.
`+(pattern-list)` : One or more occurrences of patterns.
`@(pattern-list)` : One of the given patterns.
`!(pattern-list)` : Anything except one of the given patterns.

COMMAND SUBSTITUTION (Insert cmd output) in cmdline. `$(cmd)` (preferred) or backticks (old form): ``cmd``.

PROCESS SUBSTITUTION Inserte output of `cmd list`.
read: `< (cmd list)` write to: `> (cmd list)`.

CONDITIONAL EXPRESSIONS Used in tests or loops.
`[[expression]]` (preferred) or `[expression]` or `test expression`. True `expression` means 0, false 1. `!` inverse results. (No word splitting, no expansion of filenames, tilde, variables, parameters, no arithmetic expression, no command/process substitution, quote removal.) The file operators:

`-a file`: True if `file` exists.
`-b file`: True if `file` is block special.
`-c file`: True if `file` is character special.
`-d file`: True if `file` is a directory.
`-e file`: True if `file` exists.
`-f file`: True if `file` exists and is a regular file.
`-g file`: True if `file` is set-group-id.
`-h file`: True if `file` is a symbolic link.
`-k file`: True if `file` has its "sticky" bit set.
`-p file`: True if `file` is a named pipe (fifo).
`-r file`: True if `file` is readable by you.
`-s file`: True if `file` exists and is not empty.
`-S file`: True if `file` is a socket.
`-t fd`: True if `file descriptor` is opened on a terminal.
`-u file`: True if `file` is set-user-id.
`-w file`: True if `file` is writable by you.
`-x file`: True if `file` is executable by you.
`-o file`: True if `file` is effectively owned by you.
`-g file`: True if `file` is effectively owned by your group.
`-l file`: True if `file` is a symbolic link.
`-n file`: True if `file` has been modified since it was last read.
`-s file`: True if `file` exists and is a socket.
`file1 -nt file2` True if `file1` is newer or `file2` not exist.
`file1 -ot file2` True if `file1` is older or `file1` not exist.
`file1 -ef file2` True if `file1` and `file2` refer to the same device/inode.

String operators:

`-z string` True if `string` is empty.
`string` True if `string` is not empty.
`-n string` True if `string` is not empty.
`string1 = string2` True if `strings` are equal.
`string1 == string2` True if `strings` are equal.
`string1 != string2` True if the `strings` are not equal.
`string1 < string2` True if `string1` sorts before `string2`.
`string1 > string2` True if `string1` sorts after `string2`.

Other operators:

`-o optname` True if shell option `optname` (set by `set`) is enabled.
`number1 OP number2`

OP is: `-eq (=)`, `-ne (!=)`, `-lt (<)`, `-le (<=)`, `-gt (>)`, `-ge (>=)`.

BUILTIN COMMANDS Optional arguments typeset like `option`.

Try `help builtin` for online help.

`:` `arguments` No action, only expanding arguments, redirections.
`. script arguments` or `source script arguments` Execute script in current environment; return exit status of the last cmd.
`alias -p name [=value]` ... Show (`-p`) or define alias.
`unalias -a name` ... Remove/destroy (`-a`: all) alias.
`builtin shell-builtin arguments` Execute specified shell builtin.
`command -pVv cmd args...` Run builtin/cmd from `$PATH` with specified args.
`-p` uses default `$PATH`. `-v`, `-V` shows path of cmd.
`caller expr` Returns the context of any active subroutine call (shell/function/script executed with `.` or `source`).
`enable -adnps -f filename builtin` ... Enable/disable (`-n`) builtins. `-f file` loads new builtins (object file), `-d` delete previous loaded builtin, `-p` shows all builtins, `-a` with indicator. `-n` shows disabled builtins. `-s` uses POSIX output.
`eval arg` ... Read `args` and concatenate into a single cmd line.

`exec -cl -a name cmd [arguments]` Replaces current shell by `cmd` with specified arguments, open/close specified redirections. `-l` as first arg puts dash in `$0` like login. `-c` clears environment. `-a` makes `argv[0]` to `name`.
`exit n` Cause shell to exit with a status `n`.
`export -fn name [=word]` ... Export variable/function (`-f`) to environment for next cmds. `-n` removes property. `-p` prints exports. `--` disables further options.
`getopts optstring name args` Parse positional parameters. `:` in `optstring` marks options with args; at first disables option error messages.
`hash -lr -p path -dt name` Show/remove hashed paths. `-d` delete `name` from hash. `-r` removes all hashed paths. `-p` put `path` in hash. Use `-t` to print several `names`. Use `-l` for reusable output.
`help -s pattern` Display information about builtins. `-s`: only short usage synopsis.
`logout` Exit a login shell.
`let arg arg ...` Evaluate arithmetic expression (see above), return result.
`read -ers -ufd -pprompt -aarray -ttimeout ...` Read (standard) input, or `fd`. `-s` is used on terminals without echo. `-e` uses readline (see below) to read input, `-r` means raw. `-a` specifies array to read in (default: `$REPLY`), `-u` file descriptor read from. `-t` defines timeout.
`return n` Terminates function/script with exit status `n`.
`shift n` Rotate positional parameters `n` times left.
`suspend -f` Suspend shell until it receives a SIGCONT signal. `-f` forces suspend without message in login shells.
`times` Print user/system times for shell and processes.
`trap -lp [arg] sigspec ...` Catches signal and reacts with `arg`. Without `arg` or `- sigspec` reseted to original values. `-l` prints list of signal names and numbers. `-p` shows assigned `args` for `sigspec` (default all assigned).
`type -aftpP name name ...` Shows type of shell object. `-t`: short classification. `-p` prints whole filenames. `-P` forces `$PATH` search for `name`. `-a` shows all known paths for `name`. `-f` suppresses function lookup.
`unset -fv name ...` Remove variable or function. `-f`: only functions; `-v` only variables.
`echo -neE arg ...` Output args, separated by spaces. `-n`: no trailing newline. `-e` allows enhanced backslash-escape characters, see QUOTING, escape sequences. `-E` disables `-e`.
`printf -v var` Extended echo with `printf(1)` formats. `-v` puts output into the value of `var` rather than stdout.
`test expression` or `[expression]` Return a status of `0` or `1` depending on the evaluation `expression`.

Jobs Processes executed in background (`&`) called "jobs". No `jobspec` for last job.

`disown -ar -h jobspec ...` Remove jobs from the table. `-h`: mark job to ignore `$SIGHUP`. `-a` removes all jobs from table. `-r` removes only running jobs.
`bg jobspec ...` Resume suspended job in the background as `&`.
`fg jobspec` Resume job in the foreground, and make it current.
`jobs -lnprs jobspec ...` or `jobs -x cmd jobspec` List/replace active jobs. `-l` list jobs with PIDs, `-p` shows only PIDs. `-r` shows only running jobs, `-s` stopped jobs. `-n` shows jobs with changed status. `jobs -x` executes `cmd` after replacing jobspecs with PID of process group leader.
`kill -l -[s] sigspec [-n signum pid] job ...` Send signal pid/job. `-s` defines signal by name, `-n` by number. `-l` lists available signals with their numbers.
`wait n ...` Wait for process and return termination status `n`.

Loops, Tests, Cases Uses exit status for decision; `:` in `cmds` for no op.

`break n` Exit from loop level `n` in `for`, `while`, `until`, `select`.
`continue n` Resume the next `n`th iteration of loop.
`if cmds1; then cmds2; elif cmds3; then cmds4; ... else cmds5; fi` `cmds2` only executed if `cmds1` returns status 0, else `cmds5` are executed (if present). `elif` permits embedded tests (`cmds3...`).
`case word in pattern [|pattern]...) cmds ; ... esac` Tests `word` against `pattern`, executes `cmds` of first true match.

