Shell Commands

❓ Also known as Internal Command or Built-in Command.

A Shell Command is integrated into the Shell itself. These commands are executed by the Shell without requiring an external program.

Shell Commands don’t involve launching a separate process.

File Management

- `cd`: change the current directory
- `dirs`: display the directory stack
- `popd`: restore the previous directory from the stack
- `pushd`: save the current directory and switch to a new one
- `pwd`: print the current working directory

Shell Environment

❓ The Shell Environment is the context within which a Shell operates, where various settings and configurations define how the Shell interacts with the system and the user. It consists of Environment Variables, Functions, Aliases, and Settings that influence Shell behavior, command execution, and user interaction. This environment is inherited by any programs or scripts launched from the Shell, making it important for consistent and predictable behavior.

❓ Shell Environment Variables are dynamic variables that can affect the behavior of processes running in the Shell.

- `alias`: create an alias for a command
- `unalias`: remove an alias
- `export`: set environment variables for child processes
- `set`: set Shell options or positional parameters
- `unset`: remove Shell variables or functions
- `declare`: declare variables and assign attributes
- `readonly`: mark variables as read-only
- `local`: define a local variable in a function
- `let`: perform arithmetic operations
- `type`: display information about a command (whether is built-in or external)

Command Execution and Job Control

❓ A Job is a process that is started by the Shell. Jobs can either run in the Foreground or in the Background. The Shell keeps track of all Jobs that it initiates and allows the user to manage them (e.g. pausing, resuming, or terminating them).

- `exec`: replace the current Shell process with a specified command
- `exit`: exit the current Shell session
- `eval`: evaluate a string as a command
- `source`: read and execute commands from a file
- `.`: same as “source”
- `jobs`: list the current jobs running in the background
- `fg`: bring a background job to the foreground
- `bg`: resume a job in the background
- `kill`: send a signal to a process
- `wait`: wait for a background process to complete
- `disown`: remove jobs from the Shell’s job table

Control Structures

Except `break`, these are described as Shell Keywords:

- `if`: conditionally execute commands
- `then`: used after “if” to define what to do when the condition is true
- `else`: used in “if” to define an alternative action if the condition is false
- `elif`: else if

- `fi` : end of “if” block
- `for` : loop through a list of values
- `while` : execute a block of code while a condition is true
- `until` : execute a block of code until a condition becomes true
- `case` : execute code based on a pattern
- `select` : create a menu of options
- `break` : exit a loop
- `continue` : skip the current iteration of a loop

I/O Redirection and FDs

- `echo` : display a line of text
- `read` : read a line of input from the user
- `printf` : format and print data
- `test` : evaluate an expression
- `true` : return a true value
- `false` : return a false value
- `exec` : redirect FDs or replace the Shell with a command

Shell Scripting and Functions

- `function` : define a Shell function
- `return` : exit a function and return a value
- `trap` : execute a command when the Shell receives a signal
- `shift` : shift positional parameters
- `getopts` : parse optional parameters for options

Shell Options

- `shopt`: set and unset shell options
- `ulimit`: limit the use of system resources
- `umask`: set the file mode creation mask
- `hash`: remember or forget the locations of commands
- `history`: display or manipulate the Shell history
- `help`: display help information about built-in commands

Debugging and Error Handling

- `set -e`: exit immediately if a command fails
- `set -u`: treat unset variables as an error
- `set -x`: print commands and their arguments as they are executed
- `set -o`: set shell options, e.g. “set -o pipefail”