**Minishell Notes**

(from <https://www.gnu.org/software/bash/manual/bash.html#Basic-Installation>)

Other useful link: <https://42-cursus.gitbook.io/guide/rank-03/minishell>

* Bash is the shell, or command language interpreter, for the GNU operating system.
* A Unix shell is both a command interpreter and a programming language
* Shells may be used interactively or non-interactively.
  + In interactive mode, they accept input typed from the keyboard.
  + non-interactively, shells execute commands read from a file.
* Shell operation:
  + Reads input from file/string supplied as argument in terminal
  + Breaks down input into words and operators **(tokens)**
  + Parses tokens into simple and combined commands
  + Performs shell expansions on the tokens
    - expansions (~, $, +, -..(=arithmetic expansion), {},
  + filename expansions
    - (e.g. “*ls \*.txt* “-> .txt will be expanded by shell to match all files with \*.txt in current directory, before ls command is executed),
    - further filename expansions: ?, [], {} (braces also for “normal expansion, see above)

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

* Performs any necessary redirections
  + - Command input / output in Bash can be redirected by using redirection operators like '<', '>', '>>', '|', etc.
    - For example, you can redirect the output of a command to a file using '>'.
    - Bash will perform these redirections before executing the command.
* Removes the redirection operators and their operands from the argument list:
* Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

  Automatisch generierte Beschreibung
* Executes command
* Optionally waits for the command to complete and collects its exit status

built-in commands:

* echo, cd, pwd, export, unset, env, exit

executable programs (often referred to as file system commands):

* ls, mkdir, rm, mv, cp, touch, chmod etc.

How to separate between built-in and file system commands?:

type -t COMMAND in terminal  
-> shows either builtin or file (for executable program)

**Difference Parsing vs Execution:**

**Parsing**  
Parsing involves analyzing the sequence of tokens to understand their syntactic structure and to build a representation that makes it easier to execute the commands. This often involves creating an abstract syntax tree (AST) or another hierarchical structure that represents the command.  
**Execution**  
Once you have the parsed representation of the command, you can execute it by invoking the appropriate system calls (e.g., fork, exec, pipe, etc.) to run the commands.  
Example Steps from Tokens to Execution

1. **Lexing**: Convert the input command string into tokens.
2. **Parsing**: Convert the tokens into a structured representation (e.g., AST).
3. **Execution**: Traverse the structured representation and execute the commands.