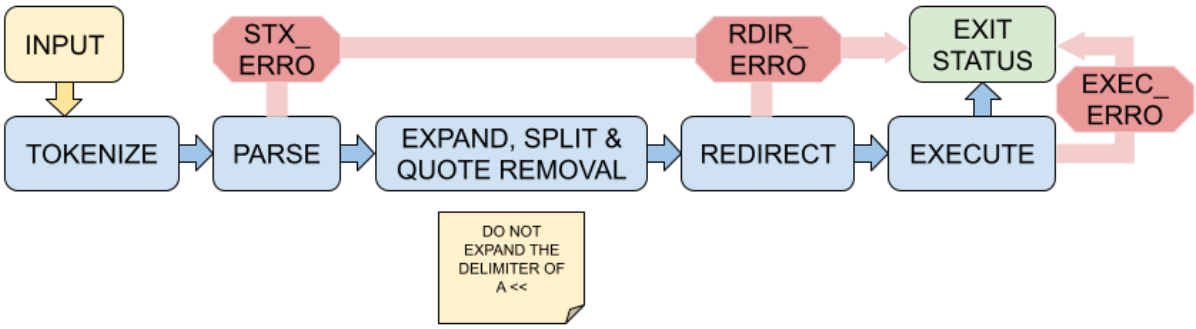
The minish manual

1. Syntax

When *minish* reads its input, it divides the input into words and operators, employing the quoting rules to select which meanings to assign various words and characters.

Minish then parses these tokens into commands and other constructs, removes the special meaning of certain words or characters, expands others, redirects input and output as needed, executes the specified command, waits for the command's exit status, and makes that exit status available for further inspection or processing.

1.1 Operation



Basically, *minish* does the following:

1. Reads its input from the user's terminal.
2. Breaks the input into *words*[1] and operators, obeying the quoting rules described in Quoting. These *tokens*[2] are separated by *metacharacters*[3].
3. Parses the tokens into commands.
4. Performs the various expansions, breaking the expanded tokens.
5. Performs any necessary redirections and removes the *redirection operators*[4] and their operands from the argument list.
6. Executes the command.
7. Optionally waits for the command to complete and collects its exit status.

1.2 Quoting

Quoting is used to remove the special meaning of certain characters or words to *minish*. Quoting can be used to disable special treatment for special characters and to prevent variable expansion.

Each of the shell *metacharacters* has special meaning to *minish* and must be quoted if it is to represent itself.

2. Commands

A command is just a sequence of *words*[1] separated by *metacharacters*[3], terminated by a *pipe*[5] or a newline. The first *word* generally specifies a command to be executed, with the rest of the *words* being that command's arguments. The command to be executed can be an executable filename or a builtin name. The return status of a command is its exit status as provided by the POSIX 1003.1 `waitpid` function, or 128+n if the command was terminated by signal n.

2.1 Pipelines

A pipeline is a sequence of one or more commands separated by a *pipe*[5] (`|`).

The format for a pipeline is:

```
command1 [ | commandN ]*
```

The standard output of *command1* is connected via a *pipe* to the standard input of *commandN*. This connection is performed before any redirections specified by the command.

3. Variables

A *variable* is an entity that stores values. It can be a *name*[6] or the special character `?`. A variable is set if it has been assigned a value. The null string is a valid value. Once a variable is set, it may be unset only by using the unset builtin command.

In *minish*, the only way a variable may be assigned to is by using the builtin `export`, which exports it to the current environment:

```
export name=[value]
```

If no *value* is given, the null string is assigned.

All values undergo variable expansion and quote removal.

4. Expansion and Quote Removal

After the command line has been split into *tokens*[2], the following steps are performed from left to right, in the following order.

1. variable expansion
2. word splitting
3. quote removal

4.1 Variable Expansion

The `$` character introduces variable expansion.

The environment variable *name*[6] to be expanded (or `?`) may be double-quoted including the leading `$` character to protect the variable to be expanded from characters immediately following it which could be interpreted as part of the *name*.

The basic form of variable expansion is *$variable*. The value of *variable* is substituted by its value.

4.2 Word Splitting

minish scans the results of variable expansion that did not occur within double quotes for word splitting.

minish treats each character of `<space>` or `<tab>` as a delimiter, and splits the results of the variable expansion into words using these characters as field terminators. Sequences of `<space>` and `<tab>` at the beginning and end of the results of the variable expansion are ignored.

Explicit null arguments (`""` or `''`) are retained and passed to commands as empty strings. **Unquoted implicit null arguments, resulting from the expansion of variables that have no values, are removed.** If a variable with no value is expanded within double quotes, a null argument results and is retained and passed to a command as an empty string. When a quoted null argument appears as part of a word whose expansion is non-null, the null argument is removed. That is, the word `-d''` becomes `-d` after word splitting and null argument removal.

Note that if no expansion occurs, no splitting is performed.

4.3 Quote Removal

All unquoted occurrences of the characters `'` and `"` that did not result from one of the above expansions are removed.

5. Redirections

Before a command is executed, its input and output may be *redirected* using a special notation interpreted by *minish*. Redirection allows commands' file descriptors to refer to different files, and can change the files the command reads from and writes to. Redirection may also be used to modify file descriptors in the current shell execution environment. The following *redirection operators*[4] may appear anywhere within a command. Redirections are processed in the order they appear, from left to right.

5.1 Redirecting Input

The format for redirecting input is:

```
<file
```

Redirection of input causes the file *file* to be opened for reading on the standard input.

5.2 Redirecting Output

The format for redirecting output is:

```
>file
```

Redirection of output causes the file *file* to be opened for writing on the standard output. If the file does not exist it is created; if it does exist it is truncated to zero size.

5.3 Appending Redirected Output

The format for appending output is:

```
>>file
```

Redirection of output in this fashion causes the file *file* to be opened for appending on the standard output. If the file does not exist it is created.

5.4 Here Documents

The format of here documents is:

```
<<word
        here_document
delimiter
```

This type of redirection instructs *minish* to read input from the current source until a line containing only *delimeter* (with no trailing spaces or tabs) is seen. All of the lines read up to that point are then used as the standard input for a command.

> ⚠ Warning
>
> No variable expansion is performed on word. If any part of *word* is quoted, the *delimiter* is the result of quote removal on *word*, and the lines in the *here_document* are not expanded. If *word* is unquoted, all lines of the *here_document* are subjected to variable expansion.

6. Executing Commands

6.1 Command Expansion

When a command is executed, *minish* performs the following expansions and redirections, from left to right, in the following order.

1. The words that the parser has marked as redirections are saved for later processing.
2. The words that are not redirections are expanded. If any words remain after expansion, the first word is taken to be the name of the command and the remaining words are the arguments.
3. Redirections are performed.

If no command name results, redirections are performed, but do not affect the current shell environment. A redirection error causes the command to exit with a non-zero status.

After a command has been split into *words*[1], if it results in a command and an optional list of arguments, the following actions are taken.

1. *minish* searches for it in the list of shell builtins. If a match is found, that builtin is invoked.
2. If the name is not a builtin, and contains no slashes, *minish* searches each element of `$PATH` for a directory containing an executable file by that name. If the search is unsuccessful, the shell prints an

error message and returns an exit status of 127.

3. If the search is successful, or if the command name contains one or more slashes, the shell executes the named program in a separate *execution environment*. Argument 0 is set to the name given, and the remaining arguments to the command are set to the arguments supplied, if any.
4. *minish* waits for the command to complete and collects its exit status.

6.3 Command Execution Environment

The shell has an *execution environment*, which consists of the following:

- Open files inherited by the shell at invocation.
- The current working directory as set by `cd` or inherited by the shell at invocation.
- Shell variables that are inherited from the shell's parent in the environment and the ones set by the `export` builtin.

When a command other than a builtin is to be executed, it is invoked in a separate execution environment that consists of the following:

- The shell's open files, plus any modifications specified by redirections to the command.
- The current working directory.
- The shell environment variables.

A command invoked in this separate environment cannot affect the shell's execution environment.

A *subshell* is a copy of the shell process.

Builtin commands that are invoked as part of a pipeline are also executed in a subshell environment. Changes made to the subshell environment cannot affect the shell's execution environment.

6.4 Environment

When a program is invoked it is given an array of strings called the *environment*. This is a list of name-value pairs, of the form `name=value`.

minish provides several ways to manipulate the environment. On invocation, the shell scans its own environment getting its variables. Executed commands inherit the environment. The `export` commands allow variables to be added from the environment. If the value of a variable in the environment is modified, the new value becomes part of the environment, replacing the old. The environment inherited by any executed command consists of the shell's initial environment, whose values may be modified in the shell, less any pairs removed by the `unset` commands, plus any additions via the `export` commands.

> **Note**
>
> It's important to note that ONLY if the command line contains a single command (no pipes) and that command is a builtin, it's redirections and execution will take place in the current shell environment (not in a subshell), thus affecting its environment.

6.5 Exit Status

The exit status of an executed command is the value returned by the `waitpid` system call or equivalent function. Exit statuses fall between 0 and 255. Exit statuses from shell builtins and compound commands are also limited to this range. Under certain circumstances, the shell will use special values to indicate specific failure modes.

For the shell's purposes, a command which exits with a zero exit status has succeeded. A non-zero exit status indicates failure. This seemingly counterintuitive scheme is used so there is one well-defined way to indicate success and a variety of ways to indicate various failure modes. When a command terminates on a fatal signal whose number is N, *minish* uses the value 128+N as the exit status.

If a command is not found, the child process created to execute it returns a status of 127. If a command is found but is not executable, the return status is 126.

If a command fails because of redirection, the exit status is greater than zero.

All of the *minish* builtins return an exit status of zero if they succeed and a non-zero status on failure. All builtins return an exit status of 2 to indicate incorrect usage, generally invalid options or missing arguments.

The exit status of the last command is available in the special variable `$?`.

6.6 Signals

minish ignores `SIGTERM` (so that `kill 0` does not kill the shell), and `SIGINT` is caught and handled. When *minish* receives a `SIGINT`, it breaks out of any executing loops. In all cases, *minish* ignores `SIGQUIT`. Non-builtin commands started by *minish* have signal handlers set to the values inherited by the shell from its parent. *minish* exits by default upon receipt of a `SIGHUP`. Before exiting, an interactive shell resends the `SIGHUP` to all jobs, running or stopped. When *minish* is waiting for a foreground command to complete, the shell receives keyboard-generated signals such as `SIGINT` (usually generated by `^C`) that users commonly intend to send to that command. This happens because the shell and the command are in the same process group as the terminal, and `^C` sends `SIGINT` to all processes in that process group.

The minish Loop

When the shell is running, it loops infinitely through the following stages:

1. *minish* displays the prompt `minish$` before reading each command line.
2. Readline is used to read commands from the user's terminal.
3. The command line is interpreted (tokenized, parsed, expanded, quotes removed).
4. The command line is redirected and executed.

Minish ignores `SIGTERM`.

> **Note**
>
> Expansion errors, redirection errors, a special builtin returning an error status and parser syntax errors will not cause the shell to exit.

History

A working history allows to retrieve and execute previously executed commands, by browsing them using the up-arrow and down-arrow keys.

Builtins

`echo [-n] [arg ...]`

> Output the args, separated by spaces, followed by a newline. The return status is always 0. If `-n` is specified, the trailing newline is suppressed.

`cd [dir]`

> Change the current directory to dir. The variable HOME is the default dir. The variable `CDPATH` defines the search path for the directory containing dir. Alternative directory names in `CDPATH` are separated by a colon (:). A null directory name in `CDPATH` is the same as the current directory, i.e., ".". If dir begins with a slash (/), then `CDPATH` is not used. If a non-empty directory name from `CDPATH` is used and the directory change is successful, the absolute pathname of the new working directory is written to the standard output. The return value is true if the directory was successfully changed; false otherwise.

`pwd`

> Print the absolute pathname of the current working directory. The return status is 0 unless an error occurs while reading the name of the current directory or an invalid option is supplied.

`export name=value ...`

> The value of the environment variable name is set to value. If the environment variable name doesn't exist it is created. If no value is given, the value will be set to "". export returns an exit status of 0 unless one of the names is not a valid shell variable name. The text after the '=' undergoes variable expansion and quote removal before being assigned to the variable.

`unset [name ...]`

> For each name, remove the corresponding variable. Each name refers to a shell variable. Read-only variables may not be unset. Each unset variable is removed from the environment passed to subsequent commands. The exit status is true unless a name is readonly.

`env`

> Prints the current environment.

`exit [N]`

> Prints `exit` followed by a newline before closing the shell returning the exit status N. If N is not defined, the exit status is that of the last command executed. The `exit` word is not printed if the output has been redirected or piped.

> **Note**
>
> Builtins return 0 if successful, and non-zero if an error occurs while they execute.

---

1. **word**: A sequence of characters treated as a unit by the shell (excluding operators). Words may not include unquoted metacharacters. ↩ ↩2 ↩3
2. **token**: A sequence of characters treated as a unit by the shell. Both operators and words are tokens. ↩ ↩2
3. **metacharacter**: A character that, when unquoted, separates words. A metacharacter is a space, tab or one of the following characters: `|`, `<`, or `>`. ↩ ↩2
4. **redirection operator**: A token that performs a redirection function. It is `<`, `>`, `<<` or `>>`. ↩ ↩2
5. **pipe**: A token that separates one command from the next, connecting the output of the former to the output of the latter. It can be also considered a *redirection operator*. It is `|`. ↩ ↩2
6. name: A word consisting only of alphanumeric characters and underscores, and beginning with an alphabetic character or an underscore. Names are used as shell variable names. ↩ ↩2