Using the Backend CLI

Use the`backend`CLI`tangle`command to tangle code blocks from your`.md`files.

✅ Tangle

```
backend tangle --input-file-path <INPUT_FILE_PATH> --output-dir <OUTPUT_
```

🔧 Options

Option	Description
`--input-file-path`	Path to the input Markdown file.
`--output-dir`	Directory where the tangled file will be written.
`--target-block`	Name (tag) of the code block to tangle.
`-h`,`--help`	Show help message.
`-V`,`--version`	Show the CLI version.

📄 Example Input File

Here's a simple example of a Markdown file with named code blocks. You can find it under`test_data/test_file.md`:

~~~markdown
### Test file

This test defines a custom function in one block and uses it in the main

Define headers:

```c headers
#include <stdio.h>
```

Define a helper function in its own block:

```c helper
void greet(const char* name) {
    printf("Hello, %s!\n", name);
}
```

Define main block `main_block`:

```c use=[headers,helper] main_block
    greet("Tangle User");
```
~~~

🧵 Tangling a Block

To generate the full program by resolving all references, run:

```
cargo run -- tangle --input-file-path ./test_data/test_file.md --output-
```

This will create the file`main_block.c`inside`./test_data/`containing:

```c
#include <stdio.h>

void greet(const char* name) {
    printf("Hello, %s!\n", name);
}

int main() {
    greet("Tangle User");
    return 0;
}
```

📌 Tips

- Code block tags (like`headers`,`helper`, or`main_block`) must be unique within the file.
- You can import blocks using`use=[<BLOCK_TAG_1>,<BLOCK_TAG_2>]`.