mpy-REPL-Tool Documentation

Release 0.13

Chris Liechti

Sep 19, 2023

Contents

Contents:

CHAPTER 1

Getting Started

1.1 Installation

This tool requires Python 3.

```
python3 -m pip install mpy-repl-tool
python3 -m pip install "mpy-repl-tool[mount]"
```

Use the second line to support the `mount` command. On windows, use `py -3` instead of `python3`.

The source code is available at github.

1.2 Find a MicroPython board

```
# list serial ports
python3 -m there detect

# and optionally also test them for a running MicroPython
# (interrupts a running program on target)
python3 -m there detect --test
```

The following examples automatically pick the first USB-Serial adapter to communicate, add a `-p COMxy` option or set the `MPY_PORT` environment variable to choose a different one.

1.3 Usage examples

```
# run a file without copying it to the target's file system:
python3 -m there run examples/hello_world.py
```

(continues on next page)

(continued from previous page)

```
# get a file list
python3 -m there ls

# file listing with more details
python3 -m there ls -l

# read the contents of a file from the target
python3 -m there cat /flash/boot.py

# copy multiple files from computer to target
python3 -m there push *.py /flash

# copy main.py and library directory from computer to target, set RTC and
# reset to start. Note: that --set-rtc is not supported by all boards.
python3 -m there --reset-on-connect --set-rtc --reset push -r lib main.py /flash

# backup all the files from the board on the PC
python3 -m there pull -r / backup/
```

Adding a `-i` starts a serial terminal:

```
python3 -m there -i

# or after running an other action
python3 -m there -i run examples/hello_world.py
```

An few statements can be executed using `-c` and it can be combined with other options:

```
python3 -m there push xy.py / -c "import xy; xy.test()" -i
```

When FUSE is available on the system and `fusepy` was installed, it is also possible to browse the files in a file navigator/explorer:

```
mkdir mpy-board
python3 -m there mount mpy-board
```

See also *Getting mount to run on Windows*, it currently requires a hack to get it working there.

Connection to telnet REPLs such as the one provided by the WiPy is also possible:

```
python3 -m there -p socket://192.168.1.1:23 -u micro -w python -i
```

CHAPTER 2

Commandline

2.1 Overview

```
usage: there [-h] [-p PORT] [-b BAUDRATE] [--set-rtc]
             [--reset-on-connect] [-c COMMAND] [-i] [--reset] [-u USER]
             [-w PASSWORD] [-v] [--develop] [--timeit]
             ACTION ...

Do stuff via the MicroPython REPL

optional arguments:
  -h, --help            show this help message and exit

port settings:
  -p PORT, --port PORT  set the serial port
  -b BAUDRATE, --baudrate BAUDRATE
                        set the baud rate

operations before running action:
  --set-rtc             set the RTC to "now" before command is executed
  --reset-on-connect    do a soft reset as first operation (main.py will not
                        be executed)

operations after running action:
  -c COMMAND, --command COMMAND
                        execute given code on target
  -i, --interactive     drop to interactive shell at the end
  --reset               do a soft reset on the end

login:
  -u USER, --user USER  response to login prompt
  -w PASSWORD, --password PASSWORD
                        response to password prompt
```

(continues on next page)

(continued from previous page)

```
diagnostics:
  -v, --verbose         show diagnostic messages, repeat for more
  --develop             show tracebacks on errors (development of this tool)
  --timeit              measure command run time

subcommands:
  use "__main__.py ACTION --help" for more on each sub-command

  ACTION                sub-command help
    detect              help locating a board
    run                 execute file contents on target
    ls                  list files
    hash                hash files
    cat                 print contents of one file
    pull                file(s) to copy from target
    push                file(s) to copy onto target
    rm                  remove files from target
    df                  Show filesystem information
    mount               Make target files accessible via FUSE
    rtc                 Read the real time clock (RTC)
```

One or two `--verbose` flag print progress information on stderr for some actions, e.g. `push` and `pull` list deltas with one `-v` and all files with two. A third `--verbose` (or `-vvv`) also prints the data exchanged between PC and target.

The order of operation is as follows:

1) execute `--reset-on-connect`
2) execute action (`run`, `push` etc.)
3) run statements that are given with `--command`
4) execute –reset
5) start miniterm if `--interactive` is given

All of these steps can be combined or used on their own.

The environment variables `MPY_PORT`, `MPY_BAUDRATE`, `MPY_USER` and `MPY_PASSWORD` are used as defaults if the corresponding command line options are not given. And if those are not given, the default is `hwgrep://USB` and 115200 baud, and None for user and password.

`hwgrep://USB` picks a random USB-Serial adapter, works best if there is only one MicroPython board connected. Otherwise the detect action should be used to find the comport and use `--port` option or environment variable.

If `--user` and `--password` are given, it waits for a login and password prompt after connecting. This is useful when connecting to e.g. a WiPy via telnet.

2.2 Actions

2.2.1 `detect`

Help finding MicroPython boards.

By default it simply lists all serial ports. If `--test` is used, each of the ports is opened (with the given `--baudrate`) and tested for a Python prompt. If there is no response it runs in a timeout, so this option is quite a bit slower that just listing the ports.

```
usage: there detect [-h] [-t]

optional arguments:
  -h, --help            show this help message and exit
  -t, --test            open and test each port
```

2.2.2 run

Execute the contents of a (small) file on the target, without saving it to the targets file system.

The file contents is sent to the REPL. The execution time is limited (see `--timeout` option to change) unless `--interactive` is given, then miniterm is started immediately.

```
usage: there run [-h] [-t TIMEOUT] [FILE]

positional arguments:
  FILE                  load this file contents

optional arguments:
  -h, --help            show this help message and exit
  -t TIMEOUT, --timeout TIMEOUT
                        wait x seconds for completion
```

Note, larger files can be executed using `push` and `--command` combined.

2.2.3 ls

List files on the targets file system. With `--long` more details are shown such as the file size.

```
usage: there ls [-h] [-l] [-r] [PATH [PATH ...]]

positional arguments:
  PATH                  paths to list

optional arguments:
  -h, --help            show this help message and exit
  -l, --long            show more info
  -r, --recursive       list contents of directories
```

The file date (shown in `--long` format) is often not very useful as most MicroPython boards do not have a battery backed RTC running.

2.2.4 cat

Loads a file from the target and prints it contents to stdout (in binary mode).

```
usage: there cat [-h] PATH

positional arguments:
  PATH                  filename on target

optional arguments:
  -h, --help            show this help message and exit
```

2.2.5 rm

Remove files and/or directories on the target.

```
usage: there rm [-h] [-f] [-r] [--dry-run] PATH [PATH ...]

positional arguments:
  PATH                  filename on target

optional arguments:
  -h, --help            show this help message and exit
  -f, --force           delete anyway / no error if not existing
  -r, --recursive       remove directories recursively
  --dry-run             do not actually create anything on target
```

2.2.6 pull

Copies files and directories from the MicroPython board to the PC.

The remote path should be absolute (starting with `/`) and supports wildcards, e.g. `/*.py`. On POSIX systems it may be needed to escape wildcards to avoid local expansion (e.g. `/\*.py` or with quotes `"/*.py"`.

```
usage: there pull [-h] [-r] [--dry-run] REMOTE [REMOTE ...] LOCAL

positional arguments:
  REMOTE                one or more source files/directories
  LOCAL                 destination directory

optional arguments:
  -h, --help            show this help message and exit
  -r, --recursive       copy recursively
  --dry-run             do not actually create anything on target
```

2.2.7 push

Copies files and directories from the PC to the MicroPython board.

The remote path should be absolute (starting with `/`). When copying a single file, the remote path may be a directory or a path including filename. When copying multiple files it must be a directory. The local path supports wildcards, e.g. `*.py`.

```
usage: __main__.py push [-h] [-r] [--dry-run] [--force]
                        LOCAL [LOCAL ...] REMOTE

positional arguments:
  LOCAL            one or more source files/directories
  REMOTE           destination directory

optional arguments:
  -h, --help       show this help message and exit
  -r, --recursive  copy recursively
  --dry-run        do not actually create anything on target
  --force          write always, skip up-to-date check
```

Directories named `.git` or `__pycache__` are excluded.

By default files are first checked (SHA256) if they are already up to date and copying is not needed. This speeds up transfer substantially. With `--force`, this check will be skipped and the files are always transferred.

The action can also be combined with `--command` and `--interactive` to start the downloaded code and see its output.

2.2.8 mkdir

Create new directory.

```
usage: there mkdir [-h] [--parents] PATH [PATH ...]

positional arguments:
  PATH        filename on target

optional arguments:
  -h, --help  show this help message and exit
  --parents   create parents
```

2.2.9 hash

Generate and print a SHA256 hash for each file given.

```
usage: there hash [-h] [-r] [PATH [PATH ...]]

positional arguments:
  PATH             paths to list

optional arguments:
  -h, --help       show this help message and exit
  -r, --recursive  list contents of directories
```

2.2.10 df

Show file system info.

```
usage: theredf [-h] [PATH [PATH ...]]

positional arguments:
  PATH        remote path

optional arguments:
  -h, --help  show this help message and exit
```

2.2.11 mount

Mount the target as file system via FUSE.

```
usage: there mount [-h] [-e] MOUNTPOINT

positional arguments:
  MOUNTPOINT            local mount point, directory must exist
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -e, --explore         auto open file explorer at mount point
```

A virtual file system is created and attached to the given directory. It mirrors the contents of the MicroPython board. Operations such as creating, renaming, deleting are supported.

To improve performance, the mount command is caching data such as directory listings and stat file infos. The cache is set to be valid for 10 seconds.

2.2.12 `rtc`

Read and print the real time clock on baords that support `pyb.RTC()`:

```
usage: __main__.py rtc [-h] [--test]

optional arguments:
  -h, --help  show this help message and exit
  --test      test if the clock runs
```

The `--test` function reads the clock twice and check that it is running.

CHAPTER 3

Technical

3.1 REPL connection

`there.repl_connection` implements a Protocol for pySerial so that statements can be executed on a remote Python prompt (REPL). MicroPython has a special "machine mode" where it does not echo input and clearly marks the output and error response, so that it is easy to parse with a machine.

The class `there.repl_connection.MicroPythonRepl` provides two functions for remote code execution. `MpyPath` is an `pathlib.Path` like object that performs operations on remote files.

3.2 Sync functionality

The command line tool implements push and pull commands that sync files. The underlying logic is available in the sync module.

3.3 Mount Action

FUSE is a feature of the GNU/Linux kernel that allows to implement file system in user space programs. There are compatible libraries for MacOS and even for Windows.

`fuse_drive.py` implements an class for `fusepy`. It gets a connection which it's using to execute commands on the target.

See also *Getting mount to run on Windows*, it currently requires a hack to get it working there.

3.4 Miniterm-MPY

This project uses a modified version of pySerial's miniterm. This version handles the special keys on Windows and translates them to escape sequences. It also uses the Python module colorama to get support for receiving some escape

sequences.

Note: `colorama` does currently not support (or recognize, when split accross multiple writes) all escape sequences sent by MicroPython, so some quirks may be visible under Windows.

Note: An alternative to `colorama` is to get `ansi.sys` working.

CHAPTER 4

Appendix

4.1 Getting `mount` to run on Windows

Install https://github.com/dokan-dev/dokany/releases/tag/v1.0.1 (Tested with V1.0.1)

Patch fuse.py:

at the top, add an new `elif`:

```
if _system == 'Darwin':
    ...
elif _system == 'Windows':
    import os
    os.environ['PATH'] += r';C:\Program Files\Dokan\Dokan Library-1.0.1'
    _libfuse_path = find_library('dokanfuse1.dll')
else:
    ...
```

and line around 980:

```
elif _system == 'Linux':
```

to:

```
elif _system == 'Linux' or _system == 'Windows':
```

Now it is possible to use `py -3 -m there mount xxx` where xxx is an existing directory and the data is then visible in that directory.

4.2 License

Note: Individual files contain the following tag instead of the full license text.

    SPDX-License-Identifier: BSD-3-Clause

This enables machine processing of license information based on the SPDX License Identifiers that are here available: http://spdx.org/licenses/

CHAPTER 5

Indices and tables

- genindex
- modindex
- search