ddrescue-tui – A Terminal UI for ddrescue

Simon Moser

University of Freiburg, Freiburg, Germany

Abstract. This paper presents ddrescue-tui, a terminal user interface for the data recovery utility ddrescue. The tool allows users to visualize the status of disk blocks during the rescue process, as well as to start and control ddrescue from the terminal. The tool parses the map files generated by ddrescue and plots them as heat maps using matplotlib. It also uses the textual framework to create an interactive and user-friendly interface. The paper describes the implementation details, the functionalities, and the project management aspects of the tool. It concludes with some limitations and future directions for the tool.

Keywords: ddrescue · terminal · parsing · visualization

1 Introduction

`GNU ddrescue`[1] is a data recovery utility that allows the transfer of data from one file or block device (such as hard disks, CD-ROMs, etc.) to another. Unlike the eponymous tool `dd`, it focuses on the retrieval of undamaged parts in the event of read errors. The basic operation of ddrescue is fully automated, eliminating the need for manual error handling or program restart.

The **mapfile**, a unique feature of `ddrescue`, is a core feature for its efficiency. The mapfile allows data to be retrieved efficiently by reading only the necessary blocks. It also allows to interrupt the rescue process at any point and resume later at the same point.

`ddrescueview`[2] is a graphical interface for `ddrescue` mapfiles. It visualizes `ddrescue`'s mapfiles via a user-friendly GUI. The main view shows a grid, with the colour of each cell indicating the rescue status of the blocks it represents.

Both tools are used for data recovery, both in private and professional environments, for example in IT forensics[7, p. 130].

However, these tools are not always executed on a system with a graphical user interface. In forensic applications, for example, it is common to work mainly under Windows, as professional software is sometimes only supported there. However, as `ddrescue` requires a unix-like operating system, a small separate Linux is often used for this purpose, which in some cases does not offer a graphical user interface.

[1] GNU ddrescue: `https://www.gnu.org/software/ddrescue/`, hereinafter referred to as `ddrescue`, not to be confused with the older `dd_rescue`

[2] ddrescueview: `https://sourceforge.net/projects/ddrescueview/`

`ddrescue` itself is a command line application, so there is mainly a need for another application that visualises mapfiles on the terminal and thus provides more precise information on the progress of a backup. The implementation of this other application is described in the following sections, starting with the parsing of the mapfiles.

1.1 Writing process

In the writing process of this report, artificial neural networks were used for selected and individual purposes.

The online translation service DeepL, which according to the company is based on machine-learning methods such as Transformers and Convolutional Neural Networks, was used as a formulation aid.

LLMs, specifically ChatGPT with GPT-4, were inter alia used to simplify the manual creation of latex code such as tables and bibliography entries and for generating the abstract, however everything was still corrected and adjusted by hand.

2 Parsing mapfiles

2.1 Mapfile structure

Mapfiles created by `ddrescue` use a compact text format to store meta information about the data rescuing process and the information about the rescuing status of blocks[3]. A simple exemplary mapfile is shown below.

1.125

Listing 1: Example mapfile

```
# Mapfile. Created by GNU ddrescue version 1.27
# Command line: ddrescue /dev/sdb sdb.img sdb.map
# Start time:   2024-02-08 06:56:57
# Current time: 2024-02-08 06:56:59
# Copying non-tried blocks... Pass 1 (forwards)
# current_pos  current_status  current_pass
0x00970000     ?               1
#      pos        size  status
0x00000000  0x02200000  +
0x02200000  0x00000001  -
0x02200001  0x3F690000  ?
```

The mapfile consists of three main parts:

The **heading comments** comments provide metadata about the mapfile, as the version of ddrescue or ddrescuelog that created it, the command-line parameters used during the operation and the start time of the program. If it was created by ddrescue, it also includes the current save time and a copy of the status message from the screen (e.g., copying, trimming, finished).

The first non-comment line, the **status line**, contains the position being tried in the input file, a status character indicating the type of operation (e.g., copying, trimming, scraping) and a positive integer denoting the current pass in the current phase. The status line helps efficient resumption of copying or retrying phases.

The other non-comment lines each describe a **data block**, containing the starting position of the block in the input file, the size of the block in bytes and a status character indicating the block's state (e.g., copied, trimmed, scraped).

2.2 Parsing in Python

The Python function `parse_mapfile` implemented in `ddrescue_tui_parser.py` is designed to parse such mapfiles. It takes as input the path to the mapfile and the desired resolution for the output array. The function performs several steps of validation and parsing, using the module `numpy`[5]:

- It checks the validity of the input parameters. If the file path is not valid or the resolution parameters are not integers greater than 1, it returns with an error message.
- It opens the mapfile and reads it line by line, distinguishing between comment lines and data lines. The first line is used to check whether the opened file is actually a mapfile, otherwise the function returns an error message.
- It parses the comment lines to extract general information about the process, such as the version of ddrescue that created the mapfile, the command-line parameters used and the direction of the reading process (forwards or backwards).
- It parses the data lines to create custom `MapEntry` objects, each representing a block of data. The `MapEntry` object has a position, size, and status, which are extracted from the data line. The status converted to a numerical value, which can later be used for plotting the information.
- After parsing all lines of the mapfile, the function sorts the list of `MapEntry` objects by position to retrieve information about the total size from the last block of data. It then transforms this list into a 1D numpy array with the length of the desired plot resolution, with each element representing the status of multiple disk blocks. As the resolution of the plot is usually several orders of magnitude smaller than the number of blocks on the disk, some information is lost when the status is compressed in this way. The highest numerical status value determines which status a compressed block receives; the values were assigned in such a way that the most interesting information is retained in the author's opinion.
- Finally, the function reshapes the 1D numpy array into a 2D numpy array based on the provided resolution. It also updates the current position in the array based on the current position given in the mapfile.

The function returns the 2D numpy array and a dictionary containing the general information. This information will be used to visualize the status of different blocks of data in a disk imaging process and provide context about the process, as described in the following sections.

3 Plotting parsed block information

In the next step, the parsed 2D-array is plotted in `ddrescue_tui_plotter.py`. This code uses the module `matplotlib`[6] to generate a plot and `PIL`[2] to load the generated plot into the memory.

The plot is generated as a heat map with `matplotlib.pylot.imshow`. Each numerical value of the 2D array representing the rescueing status from the parsing step is assigned a colour value via the `COLORS` constant. The plot is set up without any decorators such as labels or ticks and then drawn in memory using the figure's `canvas.draw` function.

Using the figure's `canvas.get_renderer().buffer_rgba`, a binary representation of the plot is copied from memory and loaded using PIL's `Image.frombytes`. Finally, the remnants of the plot are cleaned up and the PIL image is returned for display. The following section described how the display was implemented in a terminal application.

4 Terminal User Interface (TUI)

The tool was developed as a terminal UI (TUI) as opposed to a graphical UI (GUI). In order to function well in any type of console, two different modes have been developed. The file `ddrescue_tui.py` offers the CLI entry point and defines its arguments using `argparse`. It decides which mode described in the following subsections is called, either from the command line parameter `noninteractive` or by checking whether stdin is a tty.

4.1 Non-interactive mode

The non-interactive mode (see fig. 1 on the facing page) only offers the function to visualise a map file.

The previously described functions for parsing and plotting are used to retrieve and plot image object.

This object is then printed to the terminal using the module `rich_pixels`. There are two methods in this module to print images, one from an image file and one from an image object. However, the method to print from an image object does not implement the possibility to resize the image, so it was necessary to use the protected method `Pixels._segments_from_image` which can handle the resizing.

The image is printed together with general status information and re-displayed at an interval specified in command line arguments using an infinite loop and `time.sleep`.

4.2 Interactive-mode

The interactive mode offers a range of additional functions. When called, you can specify whether a map file should be visualised (and at what initial interval) or

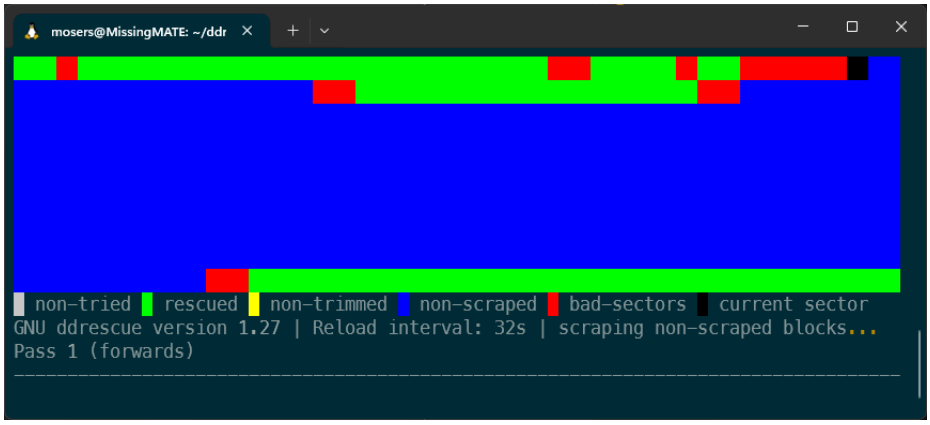

Fig. 1. Non-interactive output of a mapfile

whether `ddrescue` should be started (including all associated parameters, root permissions are required here). In addition, `ddrescue-tui` can also be started without further specifications.

The first tab of the interactive TUI (see fig. 2 on the next page) visualises a map file, shows some additional status information from it and allows you to change the reload interval during runtime.

The second tab (see fig. 3 on page 7) is only available if the programme was executed with root privileges. Here, `ddrescue` can be started with a form or the command of a running ddrescue is displayed here.

Last but not least, the TUI offers both a light and a dark mode, which can be switched at runtime.

The interactive TUI code is split in two additional files.

First of all, `ddrescue_tui_app.py` is called. It defines the main TUI application using the `textual` framework[3] and implements all top-level functions that need access to different parts of the application, like e.g. listeners for buttons. Also, the start of `ddrescue` is done here in the function `run_ddrescue`. The modules `queue` and `threading` are used to provide a non-blocking IO reading from stdout of the `ddrescue` process started with the modules `subprocess`.

Last but not least, `ddrescue_tui_widgets.py` contains customized widgets of the `textual` framework. The `StatusWidget` utilizes the frameworks reactive attributes, that trigger a redraw of a widget when the attribute changes, to display and update status information. The `PixelWidget` allows to display a simplified image on the console using the module `rich-pixels`[4] which is also developed by the `textual` developers. It uses the same mathod as described in section 4.1 on the preceding page to regularly draw the plot into the widget.

[3] textual: `https://textual.textualize.io/`
[4] rich-pixels: `https://github.com/darrenburns/rich-pixels`

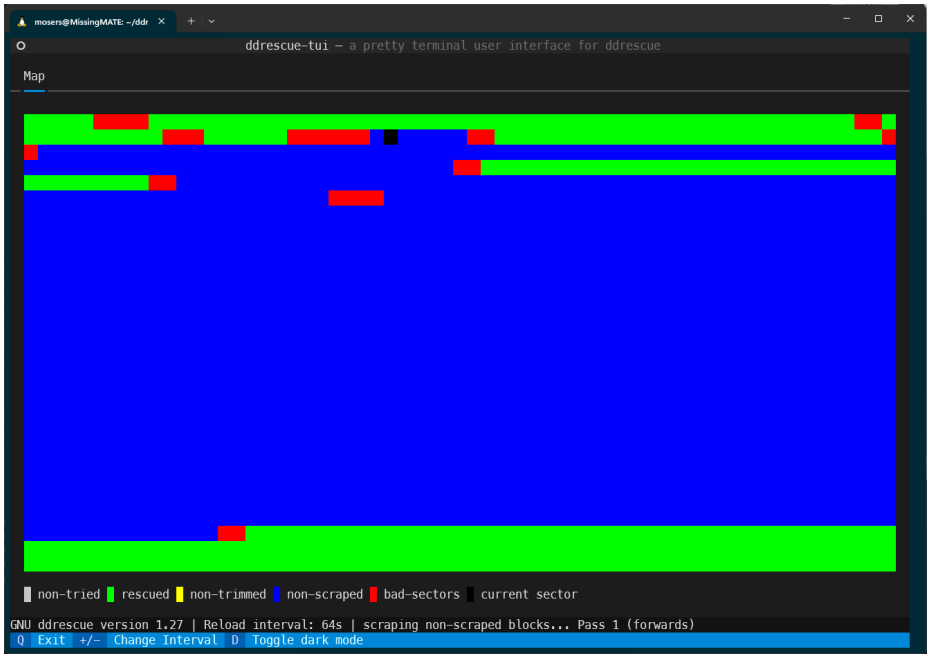

Fig. 2. Interactive output of a mapfile

5 Project management with CI/CD

For the management of the project, two GitHub Actions workflows[1, p. 8] are set up, `qa` and `build`. Every workflow runs on a container specified in the respective YAML file and defines one or more jobs that each consist out of steps that are either predefined (e.g. `actions/checkout@v4`[5] for checking out the repository) or just terminal commands (e.g. `pip install .` to install requirements).

The workflow `qa` is defined in `.github/workflows/qa.yml`. It is executed on every push event and starts two different jobs. The job `unittests` runs the unit tests stored in the directory `test` using different supported Python versions. The unit tests are written manually to assure that certain code units function as intended. The job `qodana` on the other hand triggers a run of the external static code analysis platform Qodana[6]. Qodana is checking the code for readability, maintainability and security issues. Additionally, checks for vulnerable libraries and the compatibility of the open source licences of the libraries used are activated. A badge in the README shows the status of the last workflow execution.

The workflow `build`, defined at `.github/workflows/build.yml`, is executed when a tag is pushed. It consists of a single job that installs all requirements and

[5] actions/checkout: `https://github.com/actions/checkout`
[6] Qodana: `https://www.jetbrains.com/qodana/`

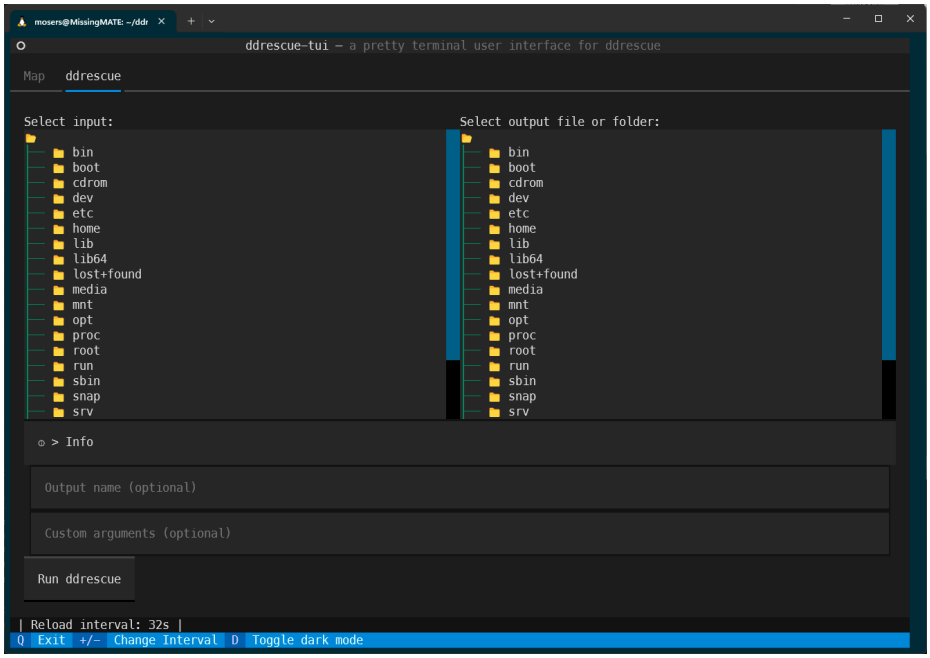

Fig. 3. Form to run ddrescue from the TUI

uses the module `build` to create a binary wheel. After successful build, a release is automatically created with the binary wheel and the source code attached.

Another GitHub feature that was tried out for the project was dependabot[4] which was configured in `/.github/dependabot.yml`. It checks for vulnerable libraries as well, creates issues for them and if possible provides pull requests with the update. The configuration includes the packaging ecosystem, the checking interval and ignored packages.

6 Conclusion

The project achieves the main goal of visualising `ddrescue` mapfiles on both interactive and non-interactive terminals. In addition, it provides further functionalities on interactive terminals, such as adjusting the update interval at runtime or starting ddrescue both as a command line parameter and via the UI.

This is also where most of the opportunities for improvement lie: `ddrescue` is executed in parallel to the TUI, which causes some difficulties with the live display of the stdout output in the TUI that have yet to be addressed.

Additionally, an option to load a (different) mapfile from inside of the UI would improve the usability

Furthermore, it would be preferable if the entire programme did not have to run with root permissions to execute `ddrescue`, but only this where it is necessary. However, a way to implement this still needs to be found.

Last but not least, there is also room for improvement in terms of user-friendliness, such as giving the option of installing via the Python Package Index with pip, offering more functionality as execution piped from a `ddrescue` run or colorless output or also just improving the documentation.

The plan here is to transfer the project from the course repository to a personal repository in order to make it publicly accessible and ensure further development.

References

1. Chaminda Chandrasekara, P.H.a.: Hands-on GitHub Actions: Implement CI/CD with GitHub Action Workflows for Your Applications. Apress (2021)
2. Clark, J.A.: Python Imaging Library Handbook. readthedocs.io (2024), `https://pillow.readthedocs.io/en/stable/handbook/`
3. Diaz, A.D.: GNU ddrescue Manual. GNU (2024), `https://www.gnu.org/software/ddrescue/manual/ddrescue_manual.html`, version 1.28
4. GitHub, I.: Keeping your supply chain secure with Dependabot (2024), `https://docs.github.com/en/code-security/dependabot`
5. Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., del R'ıo, J.F., Wiebe, M., Peterson, P., G'erard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E.: Array programming with NumPy. Nature **585**(7825), 357–362 (2020). `https://doi.org/10.1038/s41586-020-2649-2`
6. Hunter, J.D.: Matplotlib: A 2d graphics environment. Computing in science & engineering **9**(3), 90–95 (2007)
7. Pawlaszczyk, D.: Digitaler Tatort, Sicherung und Verfolgung digitaler Spuren, pp. 113–166. Springer (2017)

Topics	**Implementation**
Linux	The tool is developed for Linux, but apart from the peculiarities of different terminal emulators, there were no special problems that were dealt with
Text editor	Text editors and IDEs were used to develop the tool, but not in any special way.
Git	During development, the various possibilities of CI/CD with GitHub were explored, such as GitHub Actions.
Docker	Apart from GitHub Actions, containerization was not used.
Automation	The whole application is written in Python, especially the parsing of mapfiles which is described in section 2 on page 2.
matplotlib	Utilized for the generation of diagrams showing the rescuing status of ddrescue, as described in section 3 on page 4.
	The report was composed in , but there are no specific details to mention.
LLM	LLMs were used for selected individual purposes in the preparation of the report, see section 1.1 on page 2 for details.