Merly Mentor

User's Manual

ver. 0.0.21 — © 2023 Merly, Inc.

Executive Summary

Software developers spend around 70% of their time trying to understand code. Developers spend this time on a variety of tasks such as: identifying defects, finding technical debt, or generally trying to understand the code for feature extension. We created Merly Mentor to help programmers learn about code for each of the prior mentioned reasons. In this guide, we walk through Mentor to help provide an intuition of how it can be helpful and how it has historically been used.

Your code is never sent to us. Your privacy is our #1 priority.

Contents

1 What is Merly Mentor? 5

2 Installing the Merly Installer 6
2.1 Quick Start . . . 6
2.2 Using Merly Installer: . . . 7
2.3 Updating Mentor with the MerlyInstaller . . . 7
2.4 Requirements, Prerequisites, and Dependencies . . . 8

3 Installing Merly Mentor 9
3.1 Linux and macOS . . . 9
3.2 Windows . . . 9

4 First Steps 10
4.1 Modes of Operation . . . 10

5 Interactive Merly Mentor Execution 11
5.1 *Interactive* Mentor Inference On a File . . . 11
5.2 *Interactive* Mentor Inference On a Directory . . . 11

6 Non-Interactive Merly Mentor Execution 12
6.1 *Non-interactive* Mentor Inference on a File . . . 12
6.2 *Non-interactive* Mentor Inference on a Directory . . . 12

7 Using The Mentor Service **13**
7.1 Launch Configurations . . . 13
7.1.1 Monitoring the Mentor as a Service Daemon . . . 13
7.1.2 Advanced Daemon Configuration via the CLI . . . 14
7.2 Navigating and Logging In . . . 14
7.2.1 Homepage Elements . . . 15
7.3 Mentor as a Service Setup: Adding a Repository . . . 15
7.3.1 New Repositories . . . 16
7.3.2 Preparing the Repository and Running Inference . . . 16
7.4 Analyzing the Results . . . 18
7.4.1 Browsing by Instances and Data Files . . . 18
7.4.2 Browsing From the Performance Graph . . . 19
7.5 Configuring Mentor as a Service . . . 19
7.5.1 Configuration . . . 19
7.5.2 Tags and Filtering . . . 21
7.6 Repo Issues and Quick Actions . . . 22
7.6.1 Users . . . 23
7.6.2 Adding Users . . . 23
7.6.3 Users List . . . 24
7.6.4 Editing Users . . . 24
7.6.5 Deleting Users . . . 24
7.6.6 Sessions . . . 24
7.6.7 Languages . . . 24
7.6.8 Jobs . . . 24
7.6.9 Licenses . . . 25

8 Using Mentor With GitHub Actions **26**
8.1 Setting Up The Environment . . . 26
8.2 Creating the Workflow . . . 27
8.3 Running Mentor With GitHub Actions . . . 29
8.4 Exploring Results . . . 31

9 Advanced Usage in Mentor's Command Line Interface (CLI) Modes **32**
9.1 Command Line Argument: Repo Command . . . 32
9.1.1 General Principals and Usage . . . 32
9.1.2 Filtering . . . 32
9.1.3 Repo Sub-commands . . . 32
9.1.4 Flags and Configuration . . . 33
9.2 Additional Command Line Arguments . . . 33
9.2.1 Version . . . 33
9.2.2 Help . . . 33
9.2.3 Quiet . . . 34
9.2.4 No-color . . . 34
9.2.5 Exclude Files and/or Folders in Mentor CLI Modes . . . 35
9.2.6 Directory . . . 35
9.2.7 Language . . . 35
9.3 Copy/Paste Functionality in MentorCLI Modes . . . 36

10 Mentor Configuration Using JSON Files **37**
10.1 Colors . . . 37
10.1.1 Models Path . . . 38
10.1.2 Log Path . . . 38
10.1.3 Training Settings . . . 39

11 Exploring Mentor's Inference Results **40**

11.1 Understanding the User Interface . . . 40
11.1.1 Sort Criteria . . . 41
11.1.2 Class Filter . . . 41
11.1.3 Cost Filter . . . 41
11.1.4 Displayed Items . . . 41
11.1.5 Hide/Show Known Good . . . 41
11.1.6 Anomaly Identification . . . 41
11.1.7 Cost . . . 42
11.1.8 Complexity . . . 42
11.1.9 Source Code Location . . . 42
11.1.10 Anomaly/Expression Count . . . 42
11.1.11 Walking Through Code . . . 42
11.2 Classifying Expressions as Known Bad or Known Good . . . 42
11.3 Recommendations . . . 43

12 Basic Interactive Commands and Views For the Mentor CLI Modes 45
12.1 Code View . . . 46
12.2 Anomalies View . . . 46
12.3 Notables View . . . 47
12.4 Files View . . . 48
12.5 Expressions View . . . 49
12.6 Details View . . . 50
12.7 Known View . . . 51
12.8 Help Pop-up . . . 52

13 Sorting/Filtering Inference Results From the Mentor CLI Modes 53
13.1 Sort Criteria . . . 53
13.2 Class Filter . . . 53
13.2.1 Class Complexity . . . 53
13.3 Cost Filter . . . 55
13.4 Displayed Items Filter . . . 55
13.5 Hide/Show Known Good Filter . . . 55

14 Inference Comparison 56
14.1 Generating Multiple Inference Instances . . . 56
14.2 Git Repositories . . . 56
14.2.1 Using a Git Commit's SHA (–commit flag) . . . 56
14.2.2 Using a Number to Determine Inference Instances (–p flag) . . . 56
14.3 Non-Git Repositories . . . 57
14.4 Comparing Results Across Multiple Inference Runs . . . 59
14.4.1 Viewing New Anomalies in Merly Mentor . . . 59
14.4.2 Viewing New Notables in Merly Mentor . . . 60
14.5 Comparing Inference Results in the Post-Inference Web View (Static HTML) . . . 61

15 Post-Inference Web View (Static HTML Output) 66
15.1 Accessing the Post-Inference Web Files . . . 66
15.2 Index page . . . 67
15.3 Files page . . . 68
15.4 Anomalies page . . . 71
15.5 Notables page . . . 73
15.6 Known Good page . . . 75
15.7 Known Bad page . . . 77
15.8 Copy and Paste . . . 80

16 Mentor Plans and Feature Set 82

16.1 Basic 82
16.2 Mentor Professional 82
16.3 Mentor Professional+ 82
16.4 Mentor Enterprise 82
16.5 Mentor Enterprise+ 82
16.6 Mentor Limitless 83

17 Release Notes **84**

1 What is Merly Mentor?

At the highest level, Merly Mentor can be thought of as a code-based reasoning system. More specifically, Merly Mentor (also known as Mentor) is a self-supervised system which utilizes federated learning and an iterative multi-tiered code abstraction model to help reason about the semantics of code.

An overview of Mentor's system design is shown in Figure 1. Its distributed learning framework enables it to learn from billions of lines of code in a single day on commodity hardware. In learning from such an abundance of code, Mentor can learn both good and bad code syntax, interesting and uninteresting (copied) patterns, semantics, and even recommend patches for defects it finds. Once trained, Mentor's model can be used for a variety of tasks such as: *(i)* detecting potential technical debt or defects and recommending fixes in existing code, *(ii)* grading the quality of an existing repository, and *(iii)* guiding programmers through the important aspects of any code repository, to name a few.

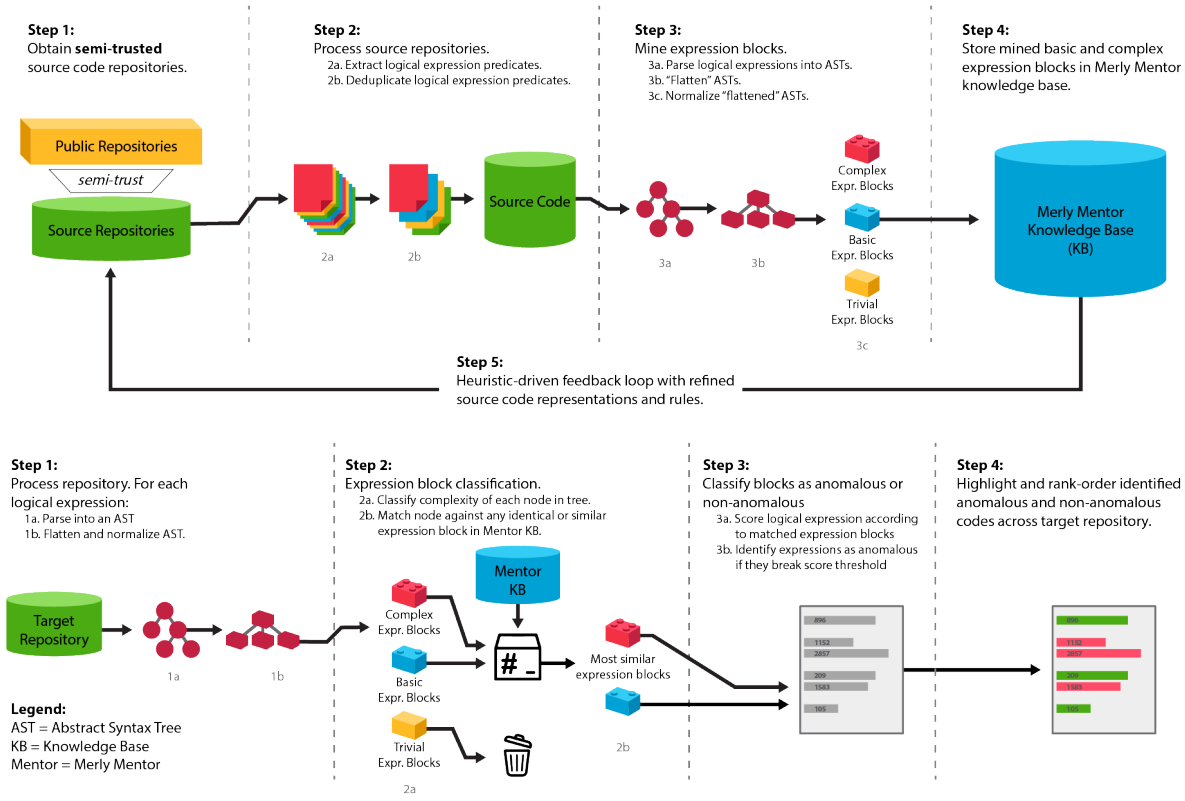


Figure 1: System Overview of Merly Mentor.

At its core, Merly Mentor is a machine programming (MP) system that learns how to identify anomalous code fragments in source code by training on billions of lines of source code across dozens of programming languages. These anomalous code fragments, also known as anomalies, are often latent defects in the existing code that programmers have failed to identify or correct. Mentor helps programmers find these anomalies and correct them, thereby improving the overall quality of the existing software. For this version of Mentor, we only include Mentor's ability to perform inference (i.e., detect good or bad patterns) on code. In subsequent releases, we may also include the ability to train new models on other code bases, including users' own proprietary ones.

2 Installing the Merly Installer

The MerlyInstaller program will not only install and register your application, but keep it up to date. In addition, the Merly Installer both installs and keeps other components, like models, up to date.

2.1 Quick Start

To get started using Merly Mentor, follow the instructions also found on the Merly.ai website By downloading Mentor, you agree to Merly's End User License Agreement (EULA).

For macOS and Linux, run the following commands in a terminal:

```
$ mkdir mentor
$ cd mentor
$ wget raw.githubusercontent.com/merly-ai/MPCC-Universal/main/install.sh
$ sh install.sh && rm install.sh
```

For Windows, run the following command using cmd.exe (not PowerShell) to start installation:

```
$ mkdir mentor
$ cd mentor
$ curl -SLO -# github.com/merly-ai/MP-CodeCheckBin-Windows/raw/main/bin/latest/MerlyInstaller.exe
$ MerlyInstaller.exe install
```

This script will download the MerlyInstaller program binary (which has a variety of functions described below) and its language models. To verify this, you will see a series "SUCCESS" messages in your terminal window during the install. You can also type `ls` ( ) or `dir` ( ) to view files in the */mentor* directory. After an installation, you should see two application binaries, the MerlyInstaller and MerlyMentor. With the installation complete, Mentor is ready to analyze a repository. To view instructions on running Merly Mentor's inference, go to the Section on Executing Merly Mentor. To learn more about the Merly Installer, continue to the next section.

2.2 Using Merly Installer:

In your /mentor directory, running the command `./MerlyInstaller usage` (🍎🐧) or `MerlyInstaller.exe usage` (🪟) will show the following options:

```
  -v | --version        | show software version
  -h | --help           | show this help
  -q | --quiet          | quiet
  -p | --product        | default: MerlyMentor (MerlyMentor | MerlyInstaller)
                        |   applies to 'install', 'updateproduct', and 'productinfo' only
  -m | --model          | default: all (CPP | C_Sharp | Go | Java | JavaScript | PHP | Python | Rust | VHDL)
                        |   applies to 'updatemodel', and 'modelinfo' only
  -k | --key            | Product Key (otherwise registration will generate a trial key)
                        |   applies to 'install' only
  --staging             | use the staging versions instead of production

>> Commands:
* Install              | download and install the given product
* UpdateProduct        | updating the given product to the latest version
* UpdateAll            | updating all products and models
* UpdateModel          | updating the given model to the latest version
* ProductInfo          | show current product information
* ModelInfo            | show current model information
* AllInfo              | show information for all components
```

Examples for using Merly Installer Depending on what your objectives are, there are a variety of ways to install and configure your installation. Here are some examples.

To download and install Merly Mentor with a product key:

🍎🐧 `$ ./MerlyInstaller -k ABCD-0123-QWER-9876 -p MerlyMentor install`

🪟 `$ MerlyInstaller.exe -k ABCD-0123-QWER-9876 -p MerlyMentor install`

To download and install Merly Mentor without registering a key (using it unlicensed for testing purposes):

🍎🐧 `$ ./MerlyInstaller install`

🪟 `$ MerlyInstaller.exe install`

To check Merly Mentor application version and models against latest releases and download/update if needed:

🍎🐧 `$ ./MerlyInstaller updateproduct`

🪟 `$ MerlyInstaller.exe updateproduct`

Note that you can specify a product (either the Installer or Mentor) with the `-p` flag, i.e. `... -p MerlyMentor`.

2.3 Updating Mentor with the MerlyInstaller

Mentor is constantly being updated. To update your copy of Mentor to the latest version including the latest programming language models it supports, run the Merly Installer (installed in the above installation steps) in the following way:

🍎🐧 `$ ./MerlyInstaller updateall`

🪟 `$ MerlyInstaller.exe updateall`

If you would like to update only certain components, please use the `usage` command from the Merly Installer as shown below for more details on how to update only the components you are interested in:

 `$ ./MerlyInstaller usage`

 `$ MerlyInstaller.exe usage`

2.4 Requirements, Prerequisites, and Dependencies

The Merly Installer will download the application binaries as well as models, but there are a number of other requirements and programs that you will need in order to run inference with Merly Mentor.

- Operating Systems
  - **macOS Version** Merly Mentor is designed to work on macOS Monterey (version 12) and above.
  - **Windows** Merly Mentor is designed to work on Windows 10 and Windows 11 systems.
- Runtimes
  - **Microsoft Visual C++ Redistributable** To avoid of missing runtime issues (like missing DLLs, i.e. *msvcp140.dll* or *vcruntime140.dll*), be sure that the MSVC++ redistributable is installed.

    Link: The Microsoft Visual C++ Redistributable download page
- Source Control
  - **Git** Git is a free and open source distributed version control system widely used in the industry. Mentor uses some git commands when processing and manipulating repositories behind the scenes.

    Link: Git's download page

3 Installing Merly Mentor

Below we list Merly Mentor's installation instructions for the currently supported operating systems (OSes). If you previously installed Mentor without a product key, but now have one, it is safe to run the MerlyInstaller again to register Mentor with a product key. A product key is a 16 character string, separated by hyphens each four characters and is generated when you register Mentor on Merly's website (e.g., ABCD-0123-QWER-9876). If you encounter any trouble with these steps, please contact `support@merly.ai` for assistance.

3.1 Linux and macOS

If you have not registered with a product key, execute the following command from the CLI, where `<key>` is your product key:

```
$ ./MerlyInstaller -k <key> install
```

3.2 Windows

Launch `cmd.exe` (do not use PowerShell as the below cURL command will not work). Navigate to your user preferred installation directory (e.g., `cd C:\Users\dev\tools\`). Then execute the following commands where `<key>` is your product key:

```
$ mkdir mentor
```

```
$ cd mentor
```

```
$ curl -LO github.com/merly-ai/MP-CodeCheckBin-Windows/raw/main/bin/latest/MerlyInstaller.exe
```

```
$ MerlyInstaller -k <key> install
```

4 First Steps

Now that Mentor is installed, inference can be preformed on a code repository. To read how to run it with basic configuration options, click or tap here to move directly to Section 5: Interactive Merly Mentor Execution.

4.1 Modes of Operation

The Mentor ecosystem employs a robust suite of options. Mentor can be run in what could be thought of as *modes*: from a command line interface (CLI), as a system daemon called the Mentor Service (which is viewed and controlled in a browser with a graphical user interface), and integration with GitHub Actions.

- Command Line Modes
  - **Interactive**: see verbose output as well as Code View when the process completes; allows for immediate interaction with issues; produces static HTML output.
  - **Non-interactive**: minimal output as well as a graceful exit to terminal when the process completes; still produces static HTML output.
- Graphical Modes
  - **Mentor Service**: accessed via the `localhost` on a web browser, with the same functionality, but with a GUI and dynamically rendered HTML pages.
  - **GitHub Actions**: using GitHub Actions and Workflows, have Mentor run when triggered by any number of git actions.

Each of these *modes* have various flags, options, and parameters–further explained in this manual–which can be used to configure how inference should be run on repositories, which Mentor saves on a per-repository basis.

Figure 2: A new Merly Mentor install, system folder, and Mentor Service in a browser.

5 Interactive Merly Mentor Execution

Merly Mentor can be executed on a command line interface (CLI) in one of two modes: interactive and non-interactive. A default interactive inference will perform the action, generate output, and then display results in a "code view," which can then be browsed, investigated, sorted, and more.

You must first have Mentor installed to perform the operations in this section. If you have not yet installed Mentor, refer to Section 3. For launching Mentor in a non-interactive (i.e. logging, quiet, or other headless) mode, refer to Section 6. Lastly, refer to the https:\merly.ai website to determine which type of inference your current product license supports. In this section we describe how to launch Mentor in an **interactive** fashion.

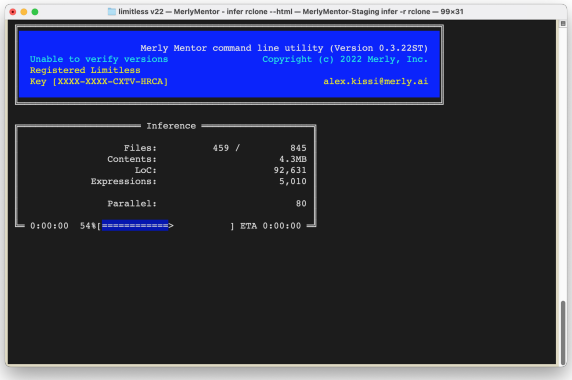


Figure 3: Merly Mentor inference running.

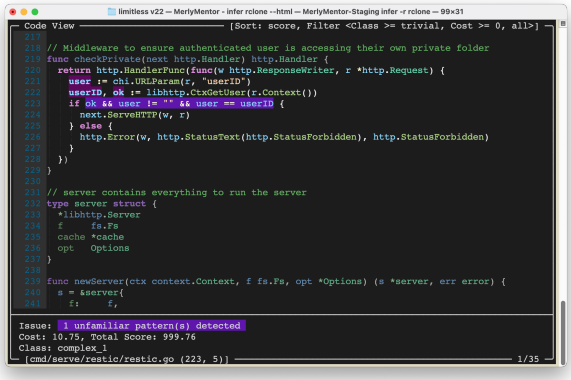


Figure 4: Post-inference results.

5.1 Interactive Mentor Inference On a File

To run *interactive* inference on a source file, from the command line interface (CLI), type the following (where `[source code file]` is the source code file you want to analyze):

```
$ ./MerlyMentor infer -D [source code file]
```

```
$ MerlyMentor.exe infer -D [source code file]
```

5.2 Interactive Mentor Inference On a Directory

To run *interactive* inference on a source folder, from the command line interface (CLI), type the following (where `[source code folder]` is a directory that contains the folder of code you want to analyze):

```
$ ./MerlyMentor infer -D [source code folder]
```

```
$ MerlyMentor.exe infer -D [source code folder]
```

When run successfully, Mentor will begin showing information that looks similar to Figure 4. This shows the progress of Mentor extracting the code DNA from the training model. When Mentor has loaded its trained model and processed the code DNA, it begins inference analysis on all source code that it finds in the files of the directory (or subdirectories) you have supplied when launching it. Figure 3 shows an example of Mentor's inference progress in analyzing a code repository, how much work it has completed, and how much work is remaining.

When inference analysis has completed, the *Code View* screen will appear (more detail and explanation shown in Figure 28), which will allow a user to analyze the inference results as discussed in the next section.

6 Non-Interactive Merly Mentor Execution

In this section, we describe how to launch Mentor non-interactively and discuss the number of output files it generates. One reason to launch Mentor in non-interactive mode is so that the output Mentor generates can be used as input to downstream software development or other machine learning stages.

If you believe an important output is missing from Mentor's output file generation, please let us know by email at support@merly.ai.

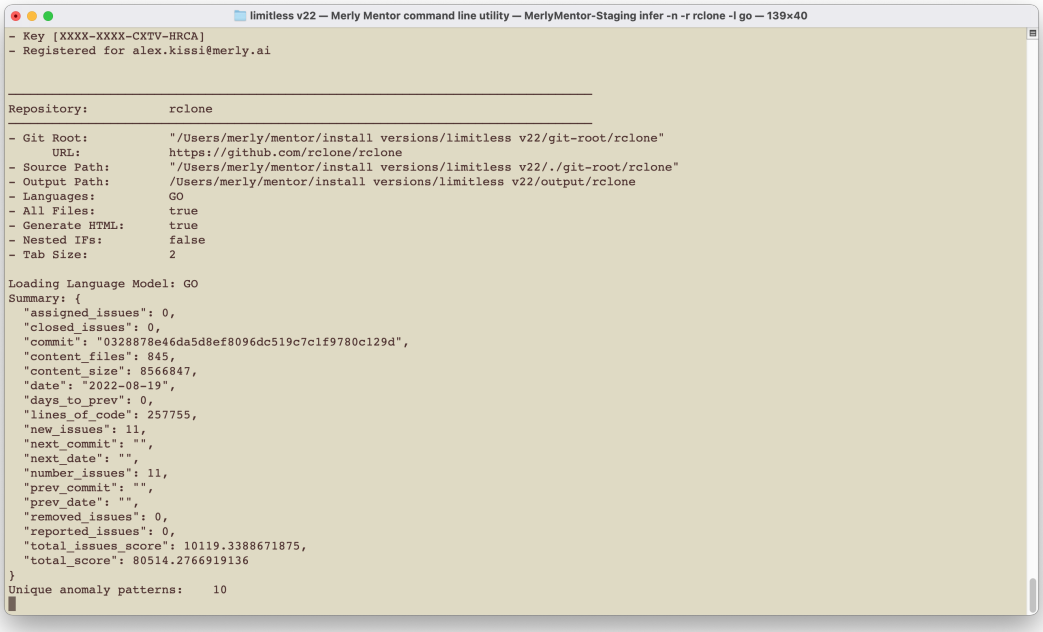


Figure 5: Merly Mentor in Non-Interactive Mode (also known as \quiet" mode).

6.1 Non-interactive Mentor Inference on a File

To run *non-interactive* inference on a source file, using the `-n` option from the command line interface (CLI), type the following (where `[source code file]` is the source code file you want to analyze):

```
$ ./MerlyMentor infer -n -D [source code file]
```

```
$ MerlyMentor.exe infer -n -D [source code file]
```

6.2 Non-interactive Mentor Inference on a Directory

To run *non-interactive* inference on a source folder, using the `-n` option from the command line interface (CLI), type the following (where `[source code folder]` is a directory that contains the folder of code you want to analyze):

```
$ ./MerlyMentor infer -n -D [source code folder]
```

```
$ MerlyMentor.exe infer -n -D [source code folder]
```

7 Using The Mentor Service

In addition to running Mentor using a command-line interface (CLI) in two *modes*, interactive and non-interactive, two methods with a graphical user interface are available. One of those is the Mentor Service. The Mentor Service is a daemon that is run on the local machine and adds a browser-based front-end to the existing back-end framework. The standard interface and various views (e.g. code, anomaly, etc.) are still a part of the system.

Once the service is started, a web browser can be used to access it by navigating to `localhost:4200` (assuming default configuration).

To see how the service will work, run Merly Mentor, passing the `service` command (`./MerlyMentor service` , and usage for the service will be shown:

```
MM service Commands:
  status        Report status of MM service
  start         Start MM service
  stop          Stop MM service
```

Figure 6: Sub-commands for the Mentor service.

7.1 Launch Configurations

There are two methods to launch the daemon, each behaving slightly differently, but accomplishing the same thing.

To launch as a *foreground* process:

`$ ./MerlyMentor daemon`

`$ MerlyMentor.exe daemon`

To launch the daemon in the *background*:

`$ ./MerlyMentor service start`

`$ MerlyMentor.exe service start`

7.1.1 Monitoring the Mentor as a Service Daemon

If Mentor as a Service is launched as a foreground process, any tasks, Hypertext Transfer Protocol (HTTP) requests, and notices are shown as output. If the service was launched as a background process, the output will be logged, but not shown in the terminal. Examples of commands used to manage the daemon are:

- Outputs current status, process identifier (PID), and port of the daemon if it is running, or indicates a background service not running:

  `$ ./MerlyMentor service status`

  `$ MerlyMentor.exe service status`

- Kills the currently running daemon process:

  `$ ./MerlyMentor service stop`

  `$ MerlyMentor.exe service stop`

- Starts a new background service process (if daemon is already running, a new process will not be started, rather a status report will be output):

  `$ ./MerlyMentor service start`

  `$ MerlyMentor.exe service start`

7.1.2 Advanced Daemon Configuration via the CLI

Advanced options are available. Using the following parameters when launching the MerlyMentor daemon, an IP address and port can be specified:

- `$ --daemon-ip <ip address>`: IP to be used for daemon
- `$ --daemon-port <port>`: Port to be used for daemon

And running "service status" would show output, e.g.:

```
$ MM service running with PID=21834, Port=4200
```

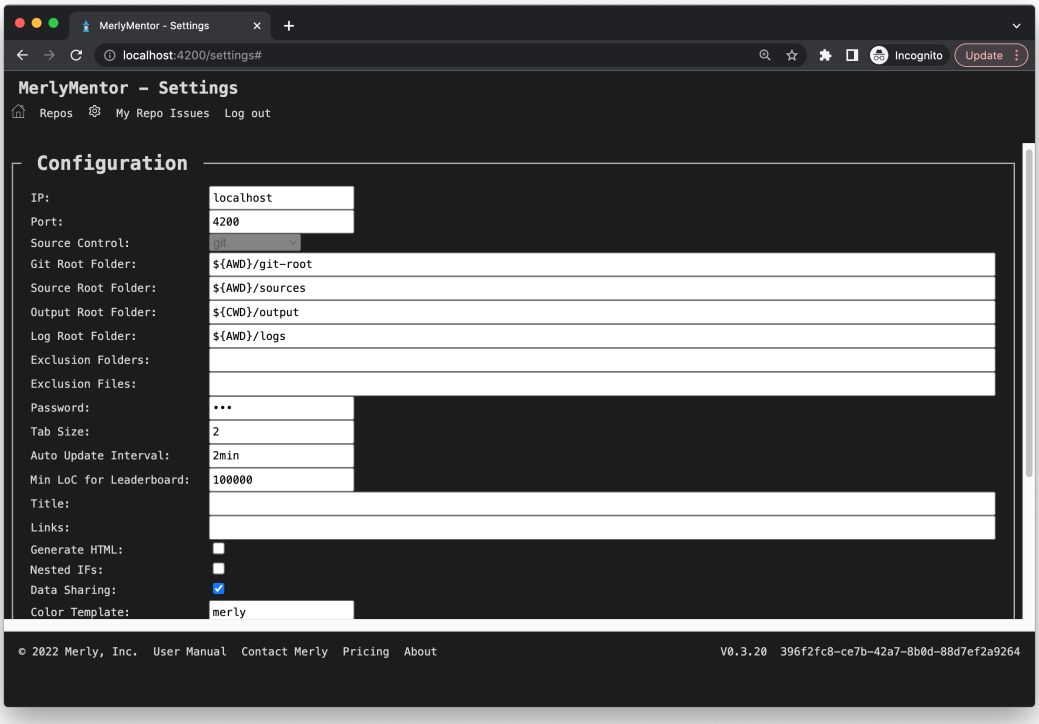


Figure 7: Managing Mentor as a Service from the web GUI.

7.2 Navigating and Logging In

Once the Mentor Service daemon is running on your system, the web GUI can be accessed. Using a web browser, navigate to the set address. The default is (`localhost:4200`). On the top menu bar, under the title bar is a link to the homepage (the 🏠 icon) as well as a `Sign In` button.

Clicking or tapping the sign in button opens a dialog box in the center of the page. The username is the assigned email address, and the password is what the service administrator generated and provided, or the user themselves set after first logon.

If a user has forgotten a password, please request a password reset from the Merly Mentor service administrator. For more on user management, see Section 7.6.1 (Mentor Service: Users).

7.2.1 Homepage Elements

The (configurable) title is at the top, and below that, the main navigation menu. Once a user is logged in, the options change to include links to Repos, Settings, My Repo Issues, and Log out. Hovering over the logout button will display the name of the logged in user. The final menu below that is a configurable navigation bar. This is where any links you add in the configuration will be available.

The main window of the Mentor Service's page will contain things such as leaderboards, repo performance graphs, issues lists, expression details, actionable items, configuration, and more.

In the footer, there are additional links: the User Manual, Merly contact information, and other ways to learn more about what Merly is doing. On the far right, you'll see the current version of Merly Mentor running the service, as well as the email address of the logged in user.

7.3 Mentor as a Service Setup: Adding a Repository

In the CLI, running inference by targeting a repository's directory will add it to the `repo list`. If you've previously run inference, those repositories will be visible on the homepage. If you are looking to add a new repository via the web GUI, this section will describe the process to add it. Doing so will allow inference to be performed.

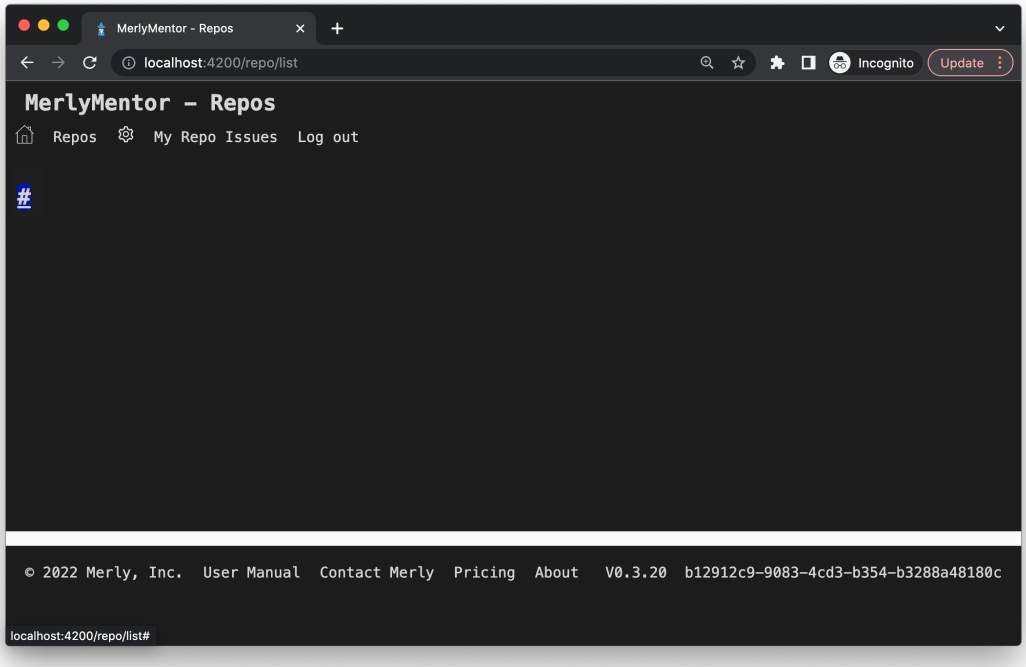


Figure 8: Adding a repository with the Mentor as a Service web GUI. Clicking or tapping the pound sign (**#**) will open a new dialog box.

1. In the menu bar at the top: click or tap "Repos." This page not only shows repositories currently being monitored by the Mentor Service, and allows users to add repositories with the "**#**" button in the top left corner of the page (See Figure 7). If there are repos already added to the service, this button will be *below* the performance graphs.

2. A context menu will open, where project and repository information can be configured.

7.3.1 New Repositories

In the newly opened dialog, fill in the following fields:

- `Name`: labels the repository; this will be displayed as the *section* title on the `/Repos` page performance graph
- `Description`: a short description of the project
- `Icon Link`: image to represent the project–used in various places and views
- `Project Link`: link to the project's website
- `Password`: the default password to be used when a new repo is added (UI in place–feature not yet implemented)
- `Source control`: method of version control (default and only option currently is git, with more (like Perforce) to be added in the future)
- `Git URL`: url to the project details
- `Git Path`: local path to source
- `Git Remote`: reference to the git remote (in most cases, "origin")
- `Git Branch`: reference to the git branch to be analyzed
- `Source Root`: local path to the project folder
- `Exclusion Folders/Files`: in the form of a comma separated list–will be split into parts with ',' as split char folder. Exclusions are for the 'directory' part of any file, and "?" matches any single char, and "*" matches any range of characters. Exclusions for the "file" setting are for the filenames specifically, therefore forward- and back-slashes should not be used for that pattern (as it will return no matches), and the wildcards "?" and "*" are treated the same as for folders
- `Output Root`: local path to the output of Mentor's analysis
- `Auto Update Interval`: sets the time in a format before inference will again–minimum is currently 1 day. Formats supported: 0-9m|min, 0-9h|hr|hrs, 0-9d|day|days
- `Tab Size`: number of spaces per tab (default is 2)
- `Tags`: configurable per repo, a comma separated list of tags used to be able to sort and organize repos on the homepage list
- `Generate HTML`: if checked, upon running inference, generates static HTML pages under the output path for selected repo
- `Nested IFs`: an experiential feature that is turned off by default, used to also analyze nested IFs as a single expression

7.3.2 Preparing the Repository and Running Inference

On the repos page, the newly added repository can be accessed. A new graph table and title of the repo will be displayed. Clicking or tapping on the title of the repo will display a detailed view. The **git clone now** button will initialize the analysis and clone the repository locally, based off of the previously configured path. Note: paths will show as "missing" until an inference action is taken. If the directories don't exist, they will be created automatically.

Repo modification actions such as editing, deleting, running or rerunning inference, cloning, and fetching need a *lead* or higher access level.

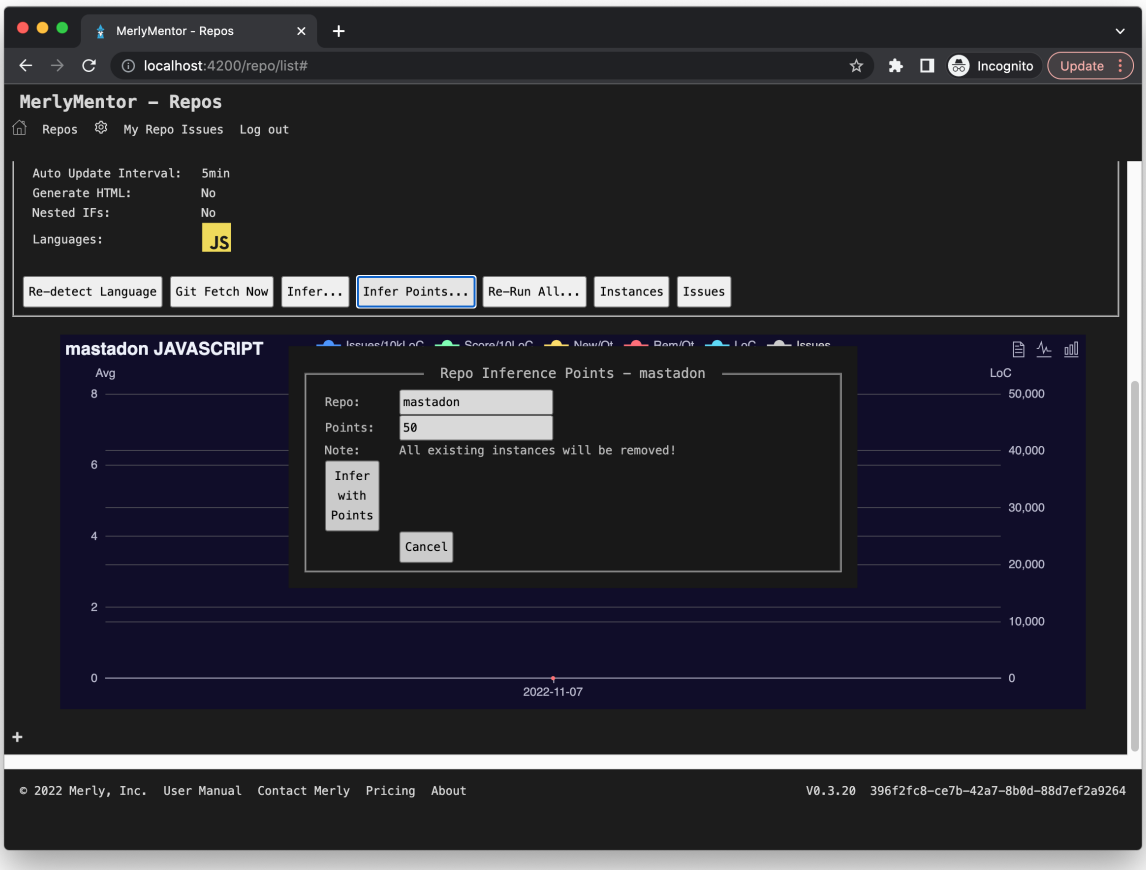


Figure 9: A newly added repository in the Mentor Service, before inference.

After adding the repository information, to run inference and generate results:

1. Clone the repository
2. Set number of inference points and let Mentor run inference

Running Inference

- **Git Clone**: clones the repository
- **Git Fetch**: performs a `git fetch` function on your tracked repository
- **Infer**: allows an inference to be run on specified commit by unique identifier (commonly called a git SHA) or branch
- **Infer Points**: sets quantity of inference points to run, and then runs inference* (note: existing inference data will be removed)

*The first inference point will be the earliest commit and the last is the most recent. An algorithm divides all remaining points in between as specified and matches them to a date that fits closely within a margin.

7.4 Analyzing the Results

Analyzing the results can be performed in a number of ways. Exploring and analyzing inference results, including the various views are described in the inference section.

After Running Inference After cloning the repository and an inference action is taken, more options become available:

- **Re-Run All**: runs inference again for all determined points
- **Instances**: below the performance graph, a more detailed view of which commit (by date and SHA), size, lines of code, and number of issues.
- **Issues**: numbered list of all identified repo issues. Note that Mentor uses this term in various places to refer to both anomalies and notable expressions (more on these in later sections).

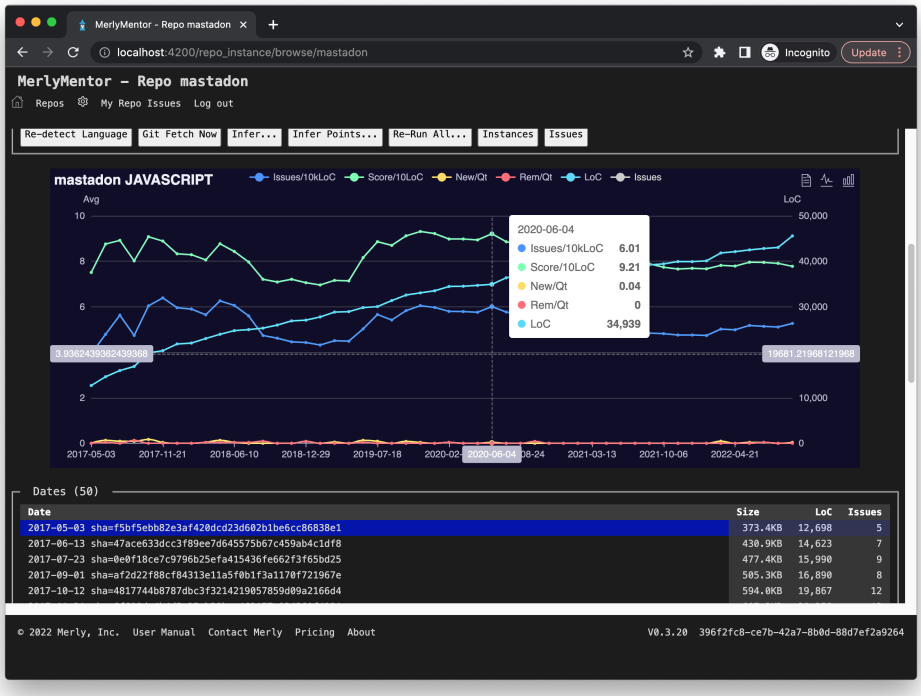


Figure 10: Mentor as a Service's repo detail view, with performance graph and dates of inference in a table below (the \Instances" button).

7.4.1 Browsing by Instances and Data Files

In a repository's menu, the `instances` button will display the date of each inference point, along with the Secure Hash Algorithm (SHA) hash, size, number of lines of code, and issues. Clicking or tapping a line will allow browsing of that specific SHA, along with even more details: files, issues, and links to the data files.

- Exploring Mentor Platform's Inference Results
- Understanding the code view
- Details on anomalies view
- Notables
- Files view

- Understanding the expressions view
- Code snippet details

The data files can be used for further automated processing and can be accessed from the output path: `/output/<PROJECT>/<DATE>/.mentor/` .

JSON files that are found within are:

- summary.json
- info.csv
- files.json
- expressions.json
- details.json

7.4.2 Browsing From the Performance Graph

Hovering over the performance graph displayed under a repository's name shows the performance over time as well as some at-a-glance metrics. Clicking or tapping at any point will display that date's inference.

7.5 Configuring Mentor as a Service

In a web browser, after navigating to the Mentor as a Service home page, the Settings menu will become available by clicking or tapping on the gear icon (⚙) in the menu bar at the top of the page. Further sub-menus will expand by clicking or tapping the title of that section.

7.5.1 Configuration

Advanced daemon configuration options are as follows:

- `Port` : specified internet port used to connect to the Mentor Service
- `Source Control` : method of version control (default and only option is git; P4 not yet implemented)
- `SSL Cert` : a path to a .pem file (i.e. one that has been split from a PFX file)
- `SSL Key` : a path to a .key file (i.e. one that has been split from a PFX file)
- `Git/Source Root Folder, Output Folder, etc.` : source directories for various Mentor processes and output
- `Exclusion` : excludes files or folders as designated, and can be provided in the form of a comma separated list–will be split into parts with ',' as split char. Folder Exclusions are for the 'directory' part of any file. A "?" matches any single character, and "*" matches any range of characters. Exclusions for the "file" setting are for the filenames specifically, therefore forward and back-slashes should not be used for that pattern (as it will return no matches), and the wildcards "?" and "*" are treated the same as for folders
- `Password` : the default password to be used when a new repo is added (UI in place–feature not yet implemented)
- `Tab Size` : number of spaces per tab (default is 2)
- `Auto Update Interval` : sets the time in a format before inference will again–minimum is currently 1 day. Formats supported: 0-9m|min, 0-9h|hr|hrs, 0-9d|day|days
- `Min LoC for Leaderboard` : sets the threshold for the number of lines of code in a project that will be displayed on the home page's Leaderboard. This value sets the point that above which will display "large" from "medium" on the homepage view

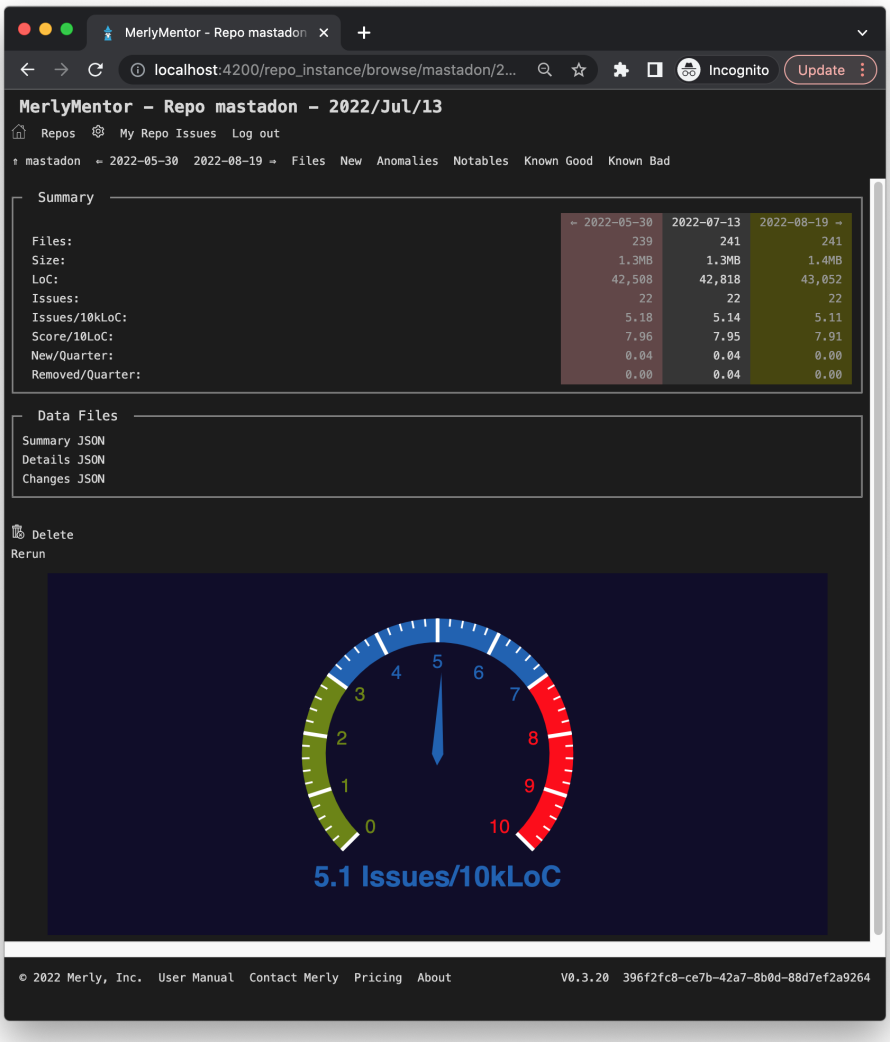


Figure 11: A report summary of a selected inference date.

- `Medium LoC for Leaderboard`: sets an additional threshold for the number of lines of code in a project that will be displayed on the home page's Leaderboard. This value sets the point that below which will split "medium" from "small" on the homepage view
- `Git Lag Log Threshold`: internal git diagnostic setting; a number (in seconds) that will log various git commands
- `Title`: sets a page title for this instance the Mentor Serviceas seen on the homepage next to the MerlyMentor menu bar
- `Links`: This allows custom text and links to be displayed under the first line of the menu bar; the syntax is similar to markdown, the format being: `[display text](http-link)[text2](http-link2)`. Note: full path including "https://" required
- `Generate HTML`: if checked, upon running inference, generates static HTML pages for the repo under the output path

- `All Files`: if checked, the files view will display all files inferred, not just ones with Issues (anomalies, notables, uniques)
- `Nested IFs`: an experiential feature that is turned off by default, used to also analyze nested IFs as a single expression
- `Data Sharing`: check to allow generic usage data–all that is sent back is a number with the type of action
- `Enable Debug Logs`: check to enable debug logs (verbose output for commands), found in the `/logs` subfolder
- `Service URL`: used as the public accessible URL to the service. If set, welcome email and password reset options become available. (additional configuration beyond the scope of the current mentor is required.)
- `Using Merly SMTP Server`: if checked, email notifications (such as new user creation) will generate and come from Merly's server
- `SMTP Host`: URL of the smtp server
- `SMTP Port`: port of the smtp server
- `SMTP User`: which mailbox authorizes the email notification. Format: valid email address
- `SMTP Password`: password for the SMTP server
- `SMTP Sender`: name of the sender (i.e. Merly Inc. No-reply)
- `SMTP From`: which mailbox the email comes from. Format: valid email address
- `SMTP Reply To`: which mailbox will be addressed when replying. Format: valid email address
- `Color Template`: this is for a future feature to allow user customization for color templates

Additional page elements:

- **Save**: saves current configuration settings
- **Clear Cache**: the cache is an internal feature used to speed up the system, and clearing this can be used to debug odd behavior; this button will clear the cache–which re-generated upon first view

Note on SSL settings: These settings are needed if one wants the Merly Mentor service to natively be only available via SSL. If the SSL has a password, Merly Mentor will ask for it while starting, via openssl.

settings.json In a hidden folder in the same directory, ".mentor" contains "settings.json," a collection of all the name/value pairs of the daemon's configuration.

7.5.2 Tags and Filtering

Each repo supports the use of tags for organization and sorting. These are configurable per repo and utilize the `Links` portion of the Configuration. Both the `/leaderboard` and `/repo/list` pages can be filtered by appending `?tag=tag1` at the end of the URL.

Additional arguments can be passed by appending an `&`, such as name and language to filter it even further. In this example, the link will filter all the added repos and only display ones matching the tag "tag1" and repos containing "ex" in their name, but also which have been identified as being C project:
`localhost:4000/repo/list?tag=tag1&name=ex&language=c`

Custom links for these filters can be set in the Configuration. See that section for more information.

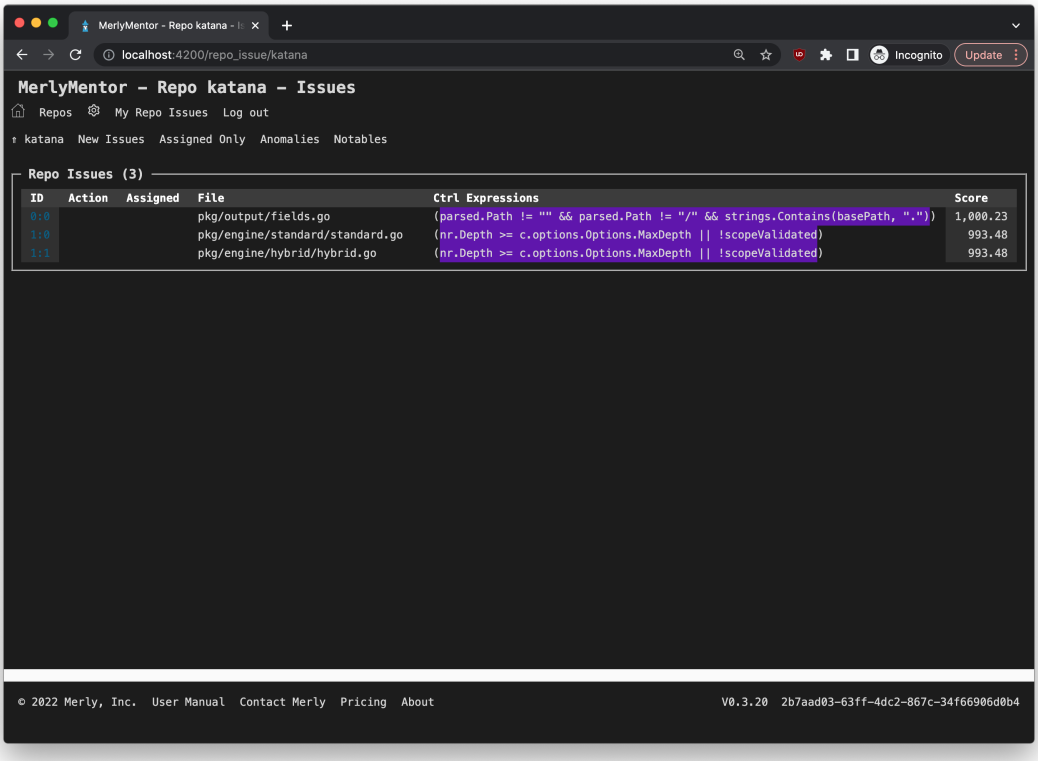


Figure 12: Issues view, showing ID as the first column

7.6 Repo Issues and Quick Actions

In many views (code, anomaly, issues, etc.), a list of "Repo Issues" are listed. In the leftmost column, each of those are assigned an ID. Hovering over the ID number displays quick actions in the dialog. In the hover menu, clicking or tapping the edit icon (✎) will allow you to assign a task, categorize it, and comment. Once the fields are filled out, click or tap the save icon (💾) in the top right corner of the hover box. These fields are drop down menus (which require another click or tap), or short response form (like for the Comment field).

In the Expression Details box, there are user-definable fields (see Figure 13):

- `Action`: options are "Ignore," "Investigate," "Simplify," "Fix," "Potential Crash," "Potential Security Issue"
- `Comment`: allows for a comment to be made
- `Category`: options are "Technical debt," "Logical error," "Memory issue," or "Security vulnerability"
- `Reporter`: user to report (from list of all created users)
- `Assigned`: assigned user (from list of all created users)

In addition to the *Quick Action* hover menu, clicking or tapping the ID number will being up the "Expression Info" and details page. Contained within is the information, such as filename and line number, as well as first date it's seen in the repo. A score is issued, along with a class and cost (see Section 13.1 for more information), along with a code snippet. From here, the same kind of categories, notes, and assignments can be made.

A note about permissions: a *developer* cannot remove a repo issue by selecting an empty action or empty assigned user.

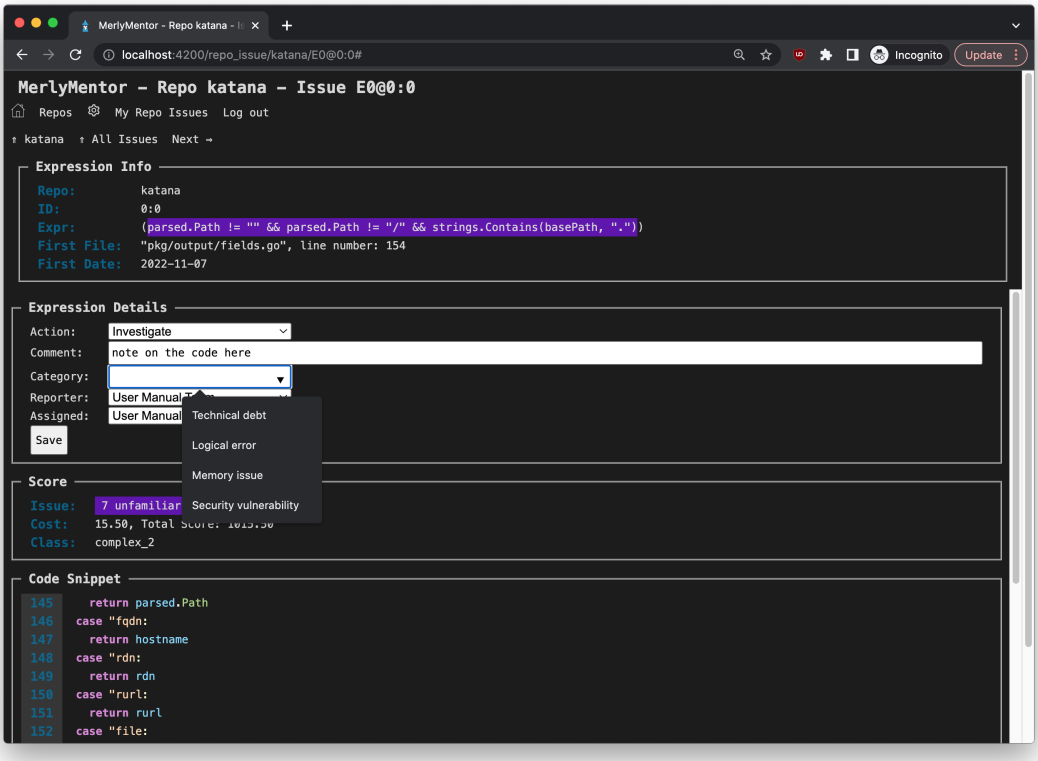


Figure 13: Expression Details page. See the section on Repo Issues and Quick Actions

7.6.1 Users

Managing Users via the command line (CLI) The first and default user of the service is the email address of the person to which the product is registered. In the terminal, running `./MerlyMentor -v` will output the user. The default password is blank.

On the command line, running `./MerlyMentor user` (/) / `MerlyMentor.exe user` ( ) will display the current users as well as sub-commands, which are as follows:

- `list`: print a list of current users, email addresses, and roles
- `add`: adds a user; must pass arguments `--user-name <name>` and `--user-email <email address>` (note that for the name argument, escape any spaces with a backslash)
- `remove`: removes a user; passing the argument `--user-name <name>` (note that like in the `add user` command, for the name argument, escape any spaces with a backslash)

Managing Users via the web GUI In addition to the command line (CLI), users can be managed via the web GUI. User modifications require an access level of *owner* or *admin*.

7.6.2 Adding Users

1. To add a user, log in as the first user, click or tap the gear icon ( ) in the menu bar
2. Navigate to the "Users" section. A current list of users will expand.
3. Click or tap the plus icon ( ) icon for a context box with new user information
4. Fields:
    - `Name`: the name of the user, e.g. "John Doe"

- `Email`: the full email address, e.g. "john@doe.com." Mentor will indicate if the email address has been used before.
- `Role`: drop down menu allows selection of three roles: administrator, restricted, or developer

The `Send welcome email` button will then create the user and send them a welcome email with a unique link to log in for the first time. During this first log-in, the user will have a chance to set their password, or use a generated one.

7.6.3 Users List

Under the *Users* configuration section, there is a table with a unique generated ID, the role the user was registered with or assigned, and online status (which indicates if a welcome email has been sent, if they're inactive, or if a user is active and online).

7.6.4 Editing Users

Here in the same Settings submenu, navigate to "Users" and expand the section. In the information table, you can view user status, role, emails, names, and unique IDs. Click or tap on an ID to edit a user. The same fields appear as when creating a user. A user can change their password, but when editing another user, if a user has a password set, it will appear as "***". Setting or erasing it will subsequently set or erase it. If you need to re-generate the password, you can click or tap that button. Additionally, to have the user reset their own password, use the `Reset Password` button to send them an email similar to the welcome email which will let them set a new password.

7.6.5 Deleting Users

In the same Users submenu, after opening a user to edit, click the Delete (🗑) icon at the bottom of the context window. **This will delete the user immediately–there is no confirmation dialog.**

users.json In the `.mentor` folder, a file called "users.json" can be found, and contains a collection of all the name/value pairs of the user information, but also some additional info which can be useful for customization and information, such as the user's color template, git token, associated customerIds, universally unique identifiers (UUID), and hashed passwords.

7.6.6 Sessions

This submenu shows various session information about users connected to Mentor as a Service, including a unique session identifier (ID), username, time of last ping, and time of last action. The "ping" represents the last time the client checked the service; the web-client ensures that the current session is still valid and was not "stolen" by someone else. "Score" shows that user's specified anomaly filter threshold (function not yet implemented).

7.6.7 Languages

A list of currently "loaded" language models, which are automatically loaded when needed for Mentor to perform an action.

7.6.8 Jobs

A cursory view of various jobs can be seen on this submenu. Reported information includes the type of job being performed such as start and stop time, as well as state (`running` or `completed`). Such jobs shown could be language operations (loading a model into memory for instance), a repo function (such as cloning or fetching), or even an inference (where the SHA is shown).

More verbose and numerous logs can be found in the `/logs` folder in the application's working directory (AWD). Info-level logs are in *command*`-info.log`. Debug-level logs are in *command*`-debug.log`. Warning-level logs are in *command*`-warning.log`, as well as reported in both debug and info-levels.

7.6.9 Licenses

Any license that is below the Limitless tier only allows for a single seat. This means that if multiple users access Mentor as a Service at the same time, more than one key needs to be added for additional users to utilize the Service.

Adding a License To add a new license, input the 16-digit key from the Merly.ai team, noting that the `Host` and `Customer` fields are pre-filled by default. Click or tap `Download` to validate the key, and when the `License` box populates, click or tap `Apply`.

keys.json In addition to the web GUI, the license data that populates the box after validation is appended to the `keys.json` file found in the `.mentor` folder.

8 Using Mentor With GitHub Actions

Another mode of operation with a graphical user interface (GUI) is GitHub Actions. Here, we use Mentor directly in your software development repository workflow. Using this feature, you can use predefined triggers to call and run Mentor as an automatic part of your continuious integration/continuous deployment (CI/CD) process. Running Mentor with a GitHub Action (GHA) will produce results that can be viewed directly within GitHub as part of your CI/CD process.

This section will describe the set up process for GitHub Actions, how to configure the workflow, and provide an example workflow on how to get started.

GitHub Actions Documentation Using Mentor via GitHub Actions is user configurable. For the full set of documentation and instructions on how to create and customize GHAs, refer to the GitHub Action documentation here.

8.1 Setting Up The Environment

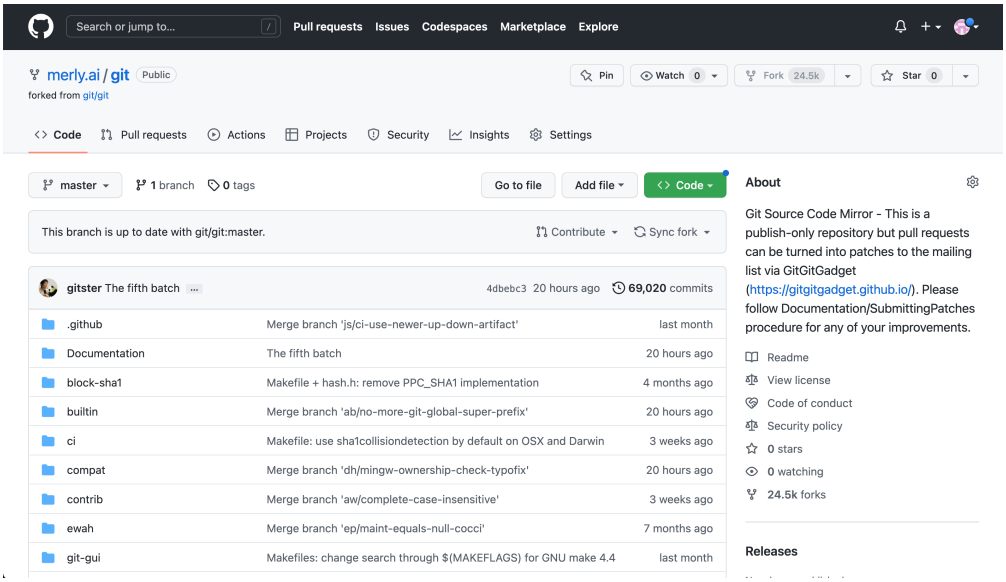


Figure 14: A GitHub repository's project page. In the navbar, \Actions" and \Settings" can be found.

On the repository's main page, click or tap the "⚙ Settings" tab in the navigation bar to bring up repository settings. Here, you will set a secret environment variable which the triggered GitHub Action will use and pass into the workflow.

In the left-hand navigation panel, look for "Secrets" and then "Actions." Once you open this menu up, click or tap the green "New Repository Secret" button in the top right (as seen in Figure 16).

Set the key/value pair of this new secret to:

- Name: `MM_KEY`
- Secret: `your Merly Mentor license key` (e.g., `ABCD-0123-QWER-9876`)

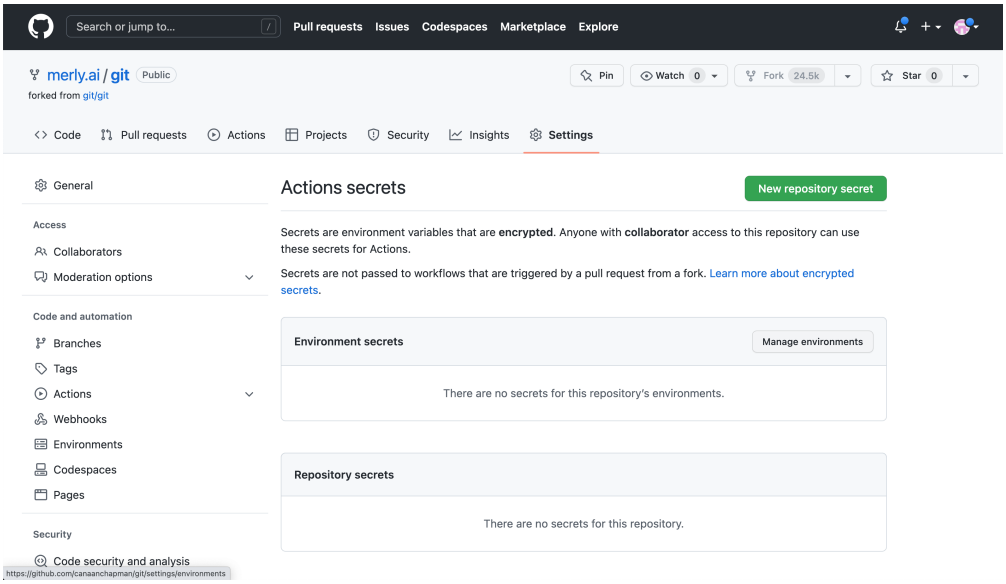


Figure 15: The settings page found under the left-side navigation menu Security > Secrets and Variables > Actions.

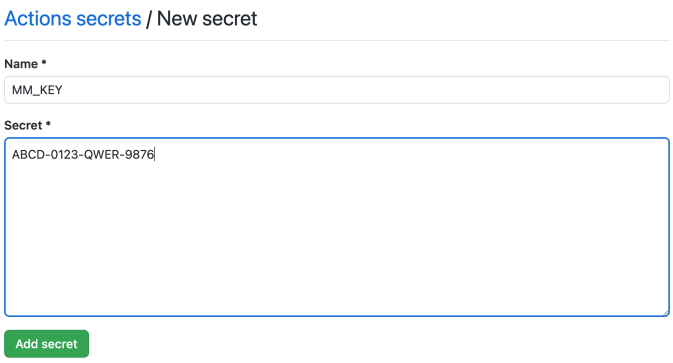


Figure 16: Adding your Merly Mentor license key as an environmental secret to be passed into the Action's container.

To save this secret environment value, click or tap the green "Add secret" button below the input boxes. You will see a notification "Repository secret added" which means it was saved successfully. You can verify this by seeing that "`MM_KEY`" has been added to your Repository secrets section.

Now that you've set your key, Mentor will take this variable and use it as part of the workflow.

8.2 Creating the Workflow

Now again from a project's repository main page on GitHub, click or tap the "▶ Actions" button in the navigation bar.

If you are working in a *forked repository*, you may have to acknowledge a dialog box from GitHub before you view or run any existing workflows. For these forked repos, you can click or tap the appropriate response in acknowledgment.

Notice that workflows are stored in the `<your repository>/.github/workflows/` directory. This is the place from which all workflows for GHA must be stored to run.

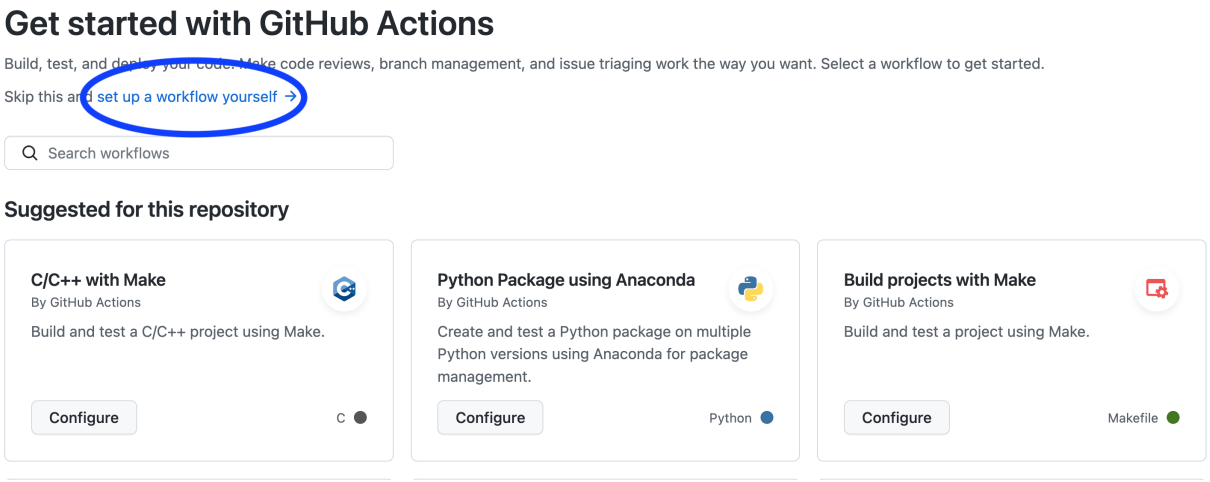


Figure 17: GitHub's repo \Actions" page with the button to set up a new workflow circled.

Once the new action page is open, GitHub will ask you to "Choose a workflow." Navigate to the link to "set up a workflow yourself" where a template will open and a file editor appears.

Start with code that specifies what will trigger the Action. Copy and paste this snippet into the file editor and then give your workflow a name.

```
name: MerlyMentor

on:
  push:
    branches: [ "master" ]
  pull_request:
    branches: [ "master" ]
```

In our example, the workflow has been named "MerlyMentor". Once the workflow is named, the "on:" section defines when the trigger will call the workflow to run.

For the full set of documentation and instructions on how to create and customize GHAs for your own needs, refer to the GitHub Action documentation here.

Next, you will describe the environment and jobs in which the containerized version of Merly Mentor will run.

```
env:
  MM_Install: ../../MM-Install
  MM_KEY: ${{ secrets.MM_KEY }}
```

Here is the environment set up, where the container will pass the `MM_KEY` secret you set up in the first step. Then, copy and paste this snippet into the editor to tell GHA that you want to use the latest Ubuntu container using the "runs-on" command. In our example below, the first thing this workflow does is to get the code using one of GitHub's standard library actions, "checkout@v3" with a fetch-depth of 0. This indicates retrieving all history for all branches and tags.

The container's environment is now set up, and when triggered, will check out the code. Continuing in our example, the inference Merly Mentor performs is run in two back-to-back jobs:

```
jobs:
  inference:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3
      with:
```

```
        fetch-depth: '0'

    - name: install MerlyMentor
      run: |
        echo "::add-mask::${{ secrets.MM_KEY }}"
        mkdir "$MM_Install"
        cd "$MM_Install"
        curl -fsSL https://raw.githubusercontent.com/merly-ai/MPCC-Universal/main/install.sh | bash -s -- -q -k "$MM_KEY"

    - name: Run MerlyMentor
      run: |
        echo "::add-mask::${{ secrets.MM_KEY }}"
        cd "$MM_Install"
        ./MerlyMentor --github-actions -D ${{github.workspace}} check
```

First, the named job "install MerlyMentor" will run the same commands a manual installation would run to get the binaries and models downloaded, as well as passing in your license key.

Next, the job "Run MerlyMentor" will move into the installation directory and run Mentor with the `command` and the `--github-actions` flag, which formats the output in a specific way GitHub can use to overlay annotations on the repository.

Figure 18: What the finished workflow may look like.

Put all of these together in the editor and save this .yml file by clicking or tapping the green "Start commit" button on the right side of the page.

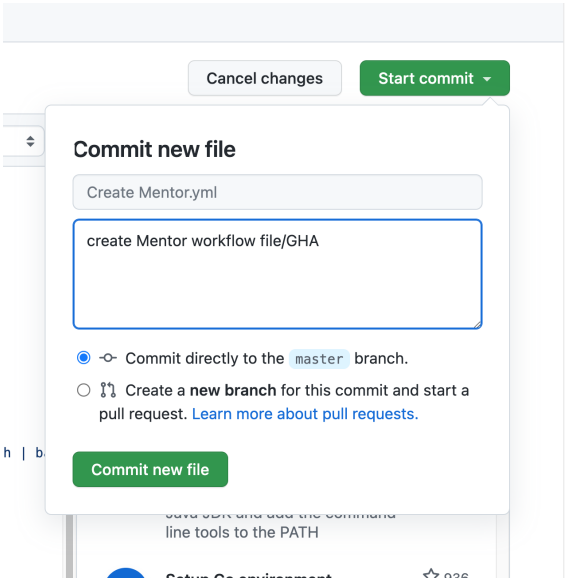


Figure 19: To save your workflow, you must commit your change by clicking or tapping the green button.

8.3 Running Mentor With GitHub Actions

When running Mentor as a GitHub Action, no manual activation is required. The setup you just performed is enough to have inference run and generate output as you update the repository.

In the example script, the workflow is called on any `git push` or `git pull`. This can be configured to your preferences (refer to the GitHub Actions documentation).

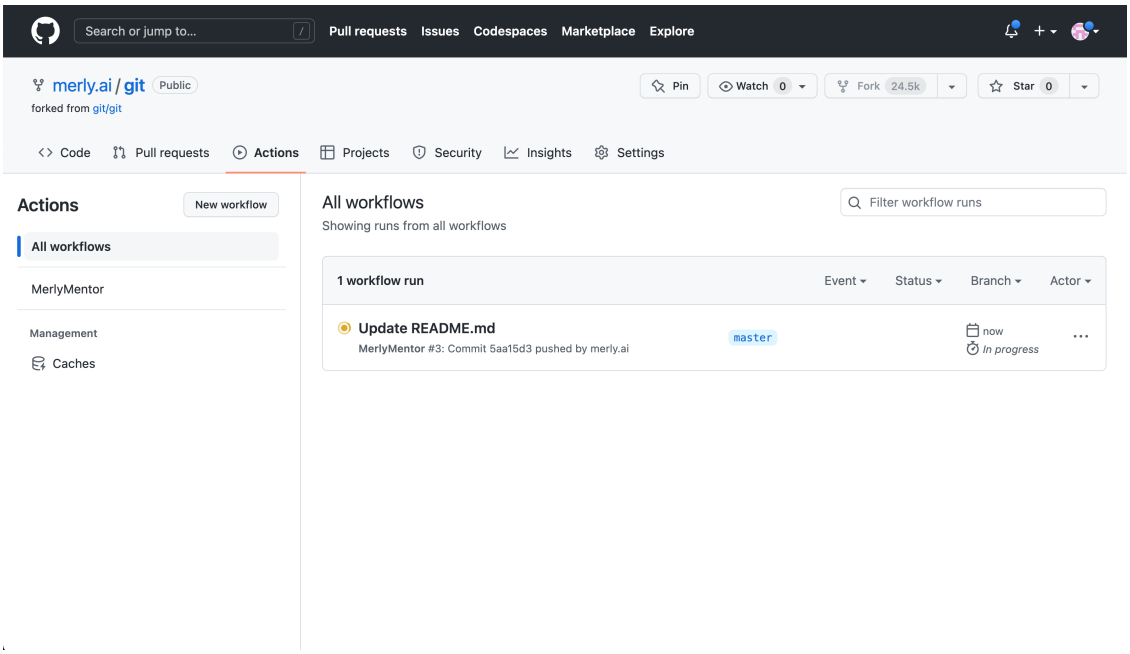


Figure 20: Actions page showing a recently triggered workflow running.

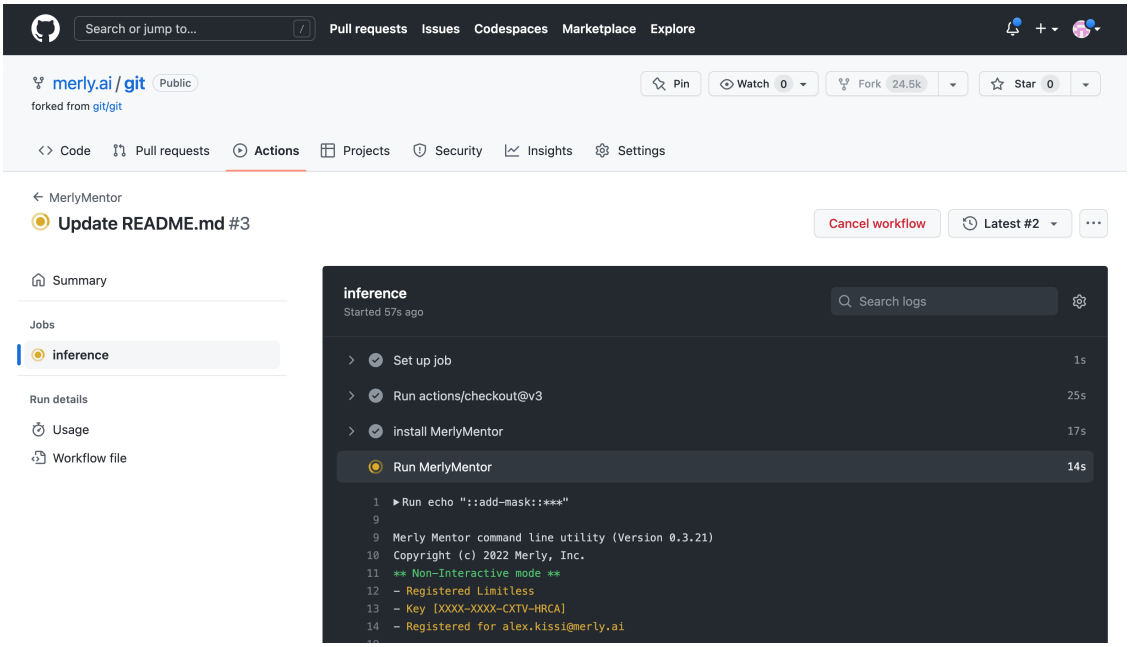


Figure 21: Monitoring the details of Merly Mentor running as a GitHub Action from within the Jobs page.

Because the system is effectively already configured and ready to trigger, you can make a test run to verify everything was set up correctly by simply making an edit in the repository and pushing a change. By navigating to your Actions, and viewing running jobs, you can view each step of the containerized process, as seen in Figures 20 and 21.

8.4 Exploring Results

When Merly Mentor is run as a GitHub action, a special flag is used, making the default output both readable and usable by GitHub. Because of this, once a workflow is complete, the job will end and the container is shut down. Results can be viewed in the Actions tab. This tab can be found under "All workflow runs", as seen in Figure 22.

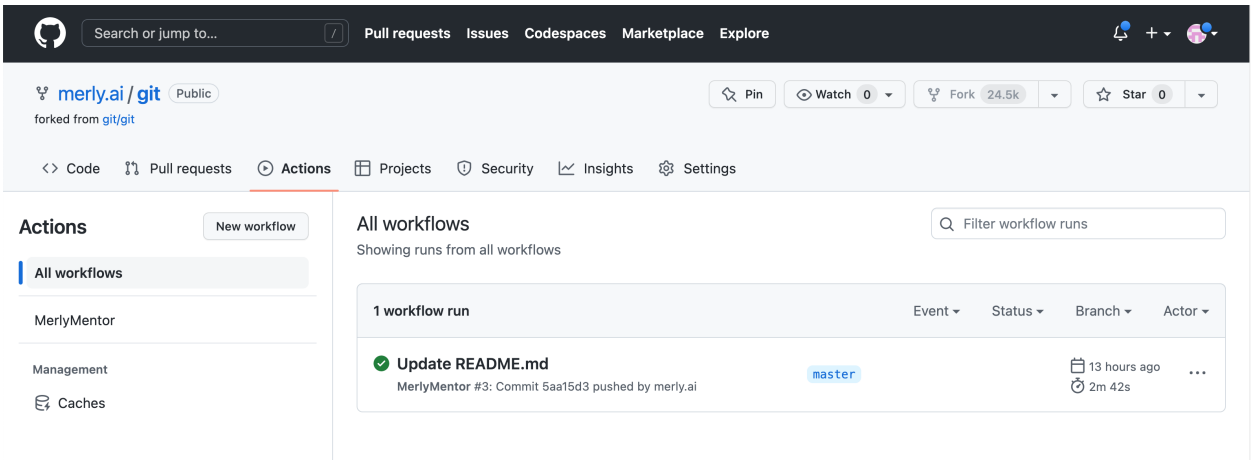


Figure 22: Workflow results and reporting page from a recently run Mentor job.

9 Advanced Usage in Mentor's Command Line Interface (CLI) Modes

In this section, we describe how to use some of the advanced features of Merly Mentor, using various command line arguments, flags, and subcommands to manipulate how it operates.

9.1 Command Line Argument: Repo Command

The repo command is one of several usage functions which essentially controls identifying and taking an action on your code.

9.1.1 General Principals and Usage

Running Merly Mentor with the repo command and a sub-command will manipulate configurations or invoke certain actions. To view what usage options and sub-commands are available, type the following into your operating system's (OS) command line or terminal:

```
$ ./MerlyMentor repo <sub-command>
```

```
$ MerlyMentor.exe repo <sub-command>
```

9.1.2 Filtering

To aid in viewing and managing various repositories in CLI modes of operation, filters to functions can be applied by name or language for the `list` and `details` sub-command. To filter a repository by name, pass the `-r` flag with the repo name. To filter by language, pass the `-l` flag with the desired language (see `MerlyMentor usage` for language arguments).

These filters can be combined to produce narrowed results:

- `repo -r git detail` : returns a detailed view of repos with "git" in the name
- `repo -r a list` : returns all repos with the letter "a" in the name
- `repo --html git detail` : returns a detailed view, listing repos with "git" in the name that are set to generate static HTML
- `repo -l typescript list` : returns a list of all repos using Typescript
- `repo -l c -l go list` : returns a list of all repos with the languages GO and C
- `repo -r git -l C list` : returns a list of repositories named "git" written in C
- `repo -r git -l C -u merly.ai list` : returns a list of repositories named "git" written in C where merly.ai is also present in the URL field

9.1.3 Repo Sub-commands

- `list` : displays list of all repos being tracked and languages, source control method, number of instances inferenced, and which branch it is tracking
- `detail` : more comprehensive than `list` , contains some of the same information, in addition to the paths to the git-root folder, source and output paths, languages found in the repository, and and other flags and configurations. See the Flags and Configuration subsection below.
- `remove` : takes the `-r` argument to specify the repo to remove. This does not remove *local files* of cloned repos.
- `clone` : preforms a "git clone" on named repo

- `fetch`: preforms a "git fetch" on named repo
- `files`: lists files to be inferenced taking into account currently excluded files and folders (which you can view viewed in `repo detail`). Note: performing an –exclude-files or –exclude-folders `update` command will overwrite existing exclusions, NOT append.
- `update`: updates repo configurations and flags without running inference. Changes can be viewed with `list` and `detail`.

9.1.4 Flags and Configuration

The repo command can be used to set or update various configurations stored on a per repository basis. When inference is run, any flags will set that repository's configs. Note that inference does not have to be run yet, though. The `update` sub-command will take any passed arguments and update the settings without running Merly Mentor.

List of flags and configurations:

- `-x | --exclude-file <file>`: excludes files matching given pattern (*, and ? supported)
- `-X | --exclude-folder folder` excludes folders matching given pattern (*, and ? supported)
- `--html | --no-html`: sets generation of static HTML (in specified output folder); default: true, stored per repo
- `--tab-size <number>`: number of spaces per tab; default: 2
- `--all-files | --issues-only`: affects rendering of "Files" page on static HTML. Setting "all files" to true will show all files in the report, regardless of if one contains an anomalous expression. Setting it to "issues only" will only show files with anomalies in the generated HTML. Does not affect "files view", as filtering can be performed.

9.2 Additional Command Line Arguments

Merly Mentor has several command line arguments so that you can tailor it to your preferences. Remember that you can use the command line command `MerlyMentor usage` to review the available command line arguments, subcommands, and flags.

9.2.1 Version

Use the command line argument `-v` to determine the version of Mentor that is currently installed. For example:

```
$ ./MerlyMentor -v
```

```
$ MerlyMentor.exe -v
```

9.2.2 Help

Use the command line argument `-h` to view the list of command line arguments. For example:

```
$ ./MerlyMentor -h
```

```
$ MerlyMentor.exe -h
```

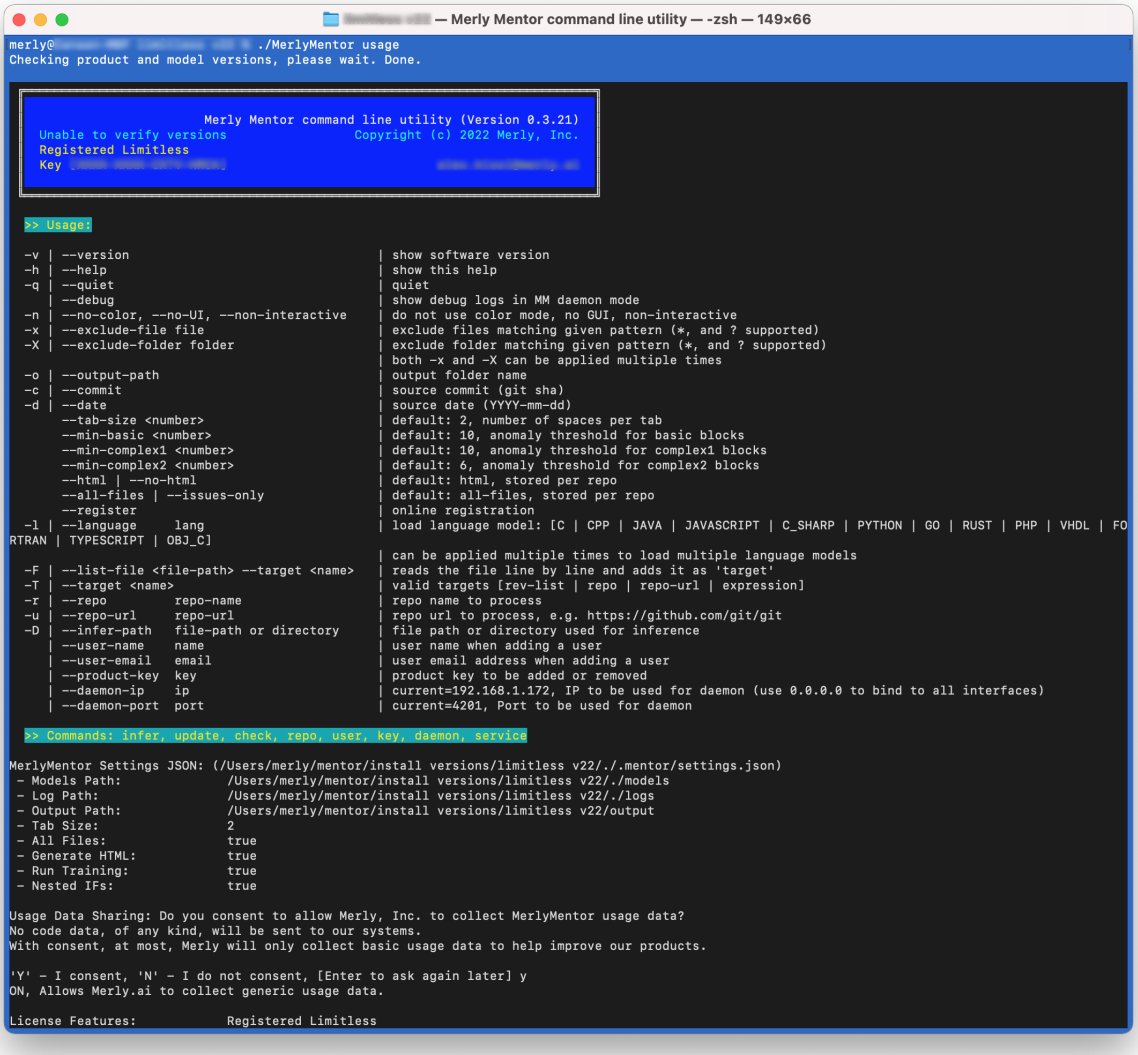


Figure 23: Use the argument `usage` to see a full list of command line arguments.

9.2.3 Quiet

Use the command line argument `-q` to run Mentor in "quiet" (no header) mode. For example:

```
$ ./MerlyMentor infer -D [source code file] -q
```

```
$ MerlyMentor.exe infer -D [source code file]-q
```

9.2.4 No-color

Use the command line argument `-n` to run Mentor in no-color mode. For example:

```
$ ./MerlyMentor infer -D [source code file] -n
```

```
$ MerlyMentor.exe infer -D [source code file]-n
```

9.2.5 Exclude Files and/or Folders in Mentor CLI Modes

If you wish to omit files or folders from Merly Mentor's inference, you can use Mentor's file and folder exclusion command. This feature is run when using the inference command or changing settings via the `repo update` command.

(Note: For how to exclude files and folders in the Mentor Service, see that section in the Mentor Service Configuration).

Use the command line argument `-x [file]` to exclude *files* matching the given pattern, `*` and `?` supported. For example:

```
$ ./MerlyMentor infer -D [source code folder] -x [file to exclude]
```

```
$ MerlyMentor.exe infer -D [source code file] -x [file to exclude]
```

Use the command line argument `-X [folder]` to exclude *folders* matching the given pattern, `*` and `?` supported. For example:

```
$ ./MerlyMentor infer -D [source code folder] -X [folder to exclude]
```

```
$ MerlyMentor.exe infer -D [source code file] -X [folder to exclude]
```

You can string multiple of these arguments together to exclude specific sets of files and directories. In the CLI, you do not need to separate these with a comma, rather, just continue to invoke the string of arguments, like this:

```
$ ...  -X src/staging/ -x /proc/secret.js -x /src/prod/internal/directory.js ...
```

Mentor's file and directory exclusion command uses wildcards. The following are some example usages:

- `$ -x *api*` - (lowercase x, excludes all files with *api* in the name)
- `$ -X */out/*` (uppercase X, excludes all folders that contain a folder *out* in the name)
- `$ -x */internal` (lowercase x, excludes all files with *internal* as the last folder name)

9.2.6 Directory

Use the command line argument `-D` to specify the directory used for inference. For example:

```
$ ./MerlyMentor infer -D [directory path]
```

```
$ MerlyMentor.exe infer -D [directory path]
```

9.2.7 Language

Use the command line flag `-l` to choose a specific programming language that Mentor will use as the model to run inference against.

Options are:

- C
- CPP
- PYTHON
- JAVA
- C_SHARP
- JAVASCRIPT
- GO
- RUST
- PHP
- VHDL
- FORTRAN
- TYPESCRIPT
- UNKNOWN

The default is UNKNOWN, but Merly Mentor will automatically detect the language and load the appropriate models. For example, the following could be used to specify C++:

```
$ ./MerlyMentor infer -D [directory path] -l CPP
```

```
$ MerlyMentor.exe infer -D [directory path] -l CPP
```

9.3 Copy/Paste Functionality in MentorCLI Modes

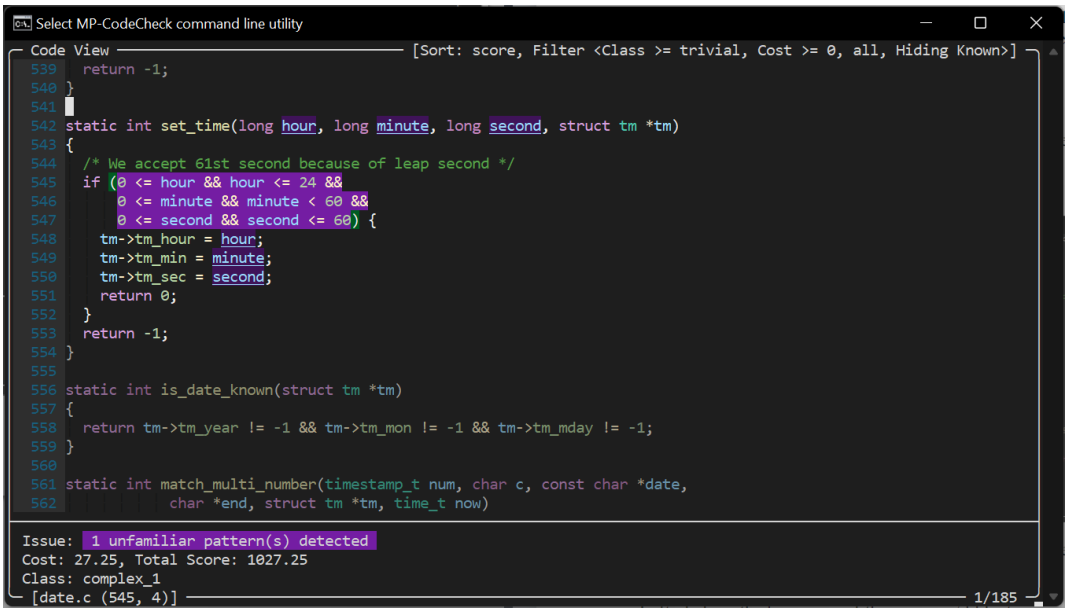


Figure 24: A white cursor appears to mark the start of your copy area.

Left-click on the beginning of the area you would like to copy. You will see a white cursor where the copy will begin (Figure 24). Next, hold down the left mouse button and drag the cursor over the area you would like to copy, as shown in Figure 25). You can also hold down the Shift key and use the cursor keys (`up`, `down`, `left`, and `right`) to select the area you would like to copy. When you have the selection highlighted, you can either right-click on your mouse or press the `Enter` key on your keyboard to complete the copy.

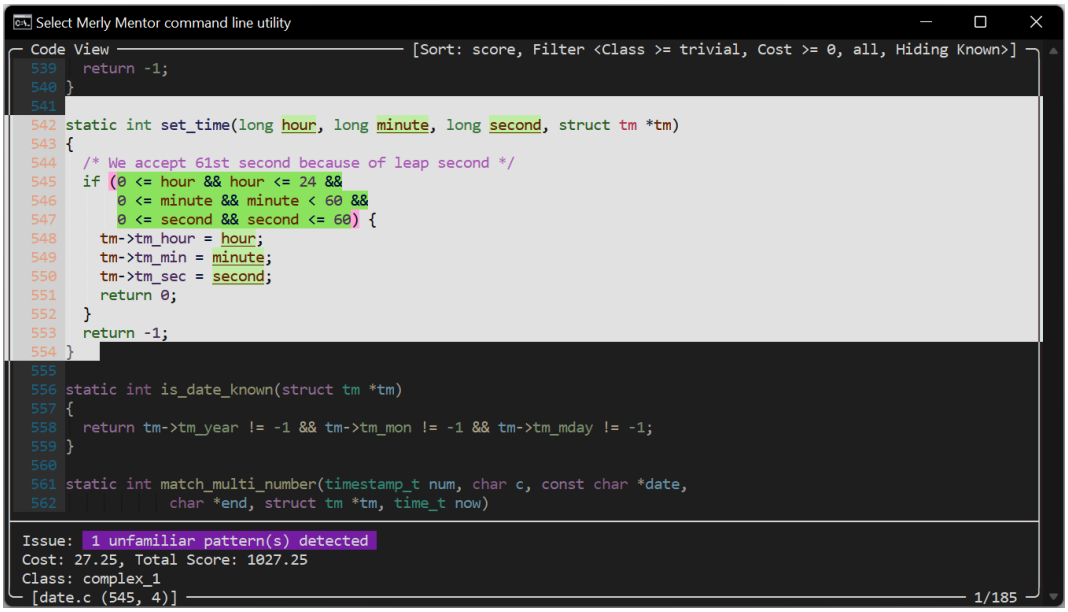


Figure 25: Highlighted text is ready to be copied.

10 Mentor Configuration Using JSON Files

For users who wish to *further* customize their Merly Mentor experience, a JSON file is available to configure Mentor to fit your preferences.

The JSON file is located in the application's working directory, as a hidden folder: `$AWD/.mentor/`

You can use any text editor to modify the colors, log file locations, users, keys, and various other settings. When installed, the initial set up is generated in the following files:

- keys.json - license keys and metadata
- repos.json - repo directories and settings
- sessions.json - metadata referenced for inferencing
- settings.json - various Mentor settings
- state.json - metadata and statistics
- users.json - users, emails, UUIDs, etc.

10.1 Colors

These are stored in the JSON file in hexadecimal (HEX) RGB; simply use your favorite color picker to find the hex value of a color, and change the value of the associated item.

For example, you can set `anomaly_background` to RGB `ab852e` to change the highlight color of the anomalous expressions to dark orange:

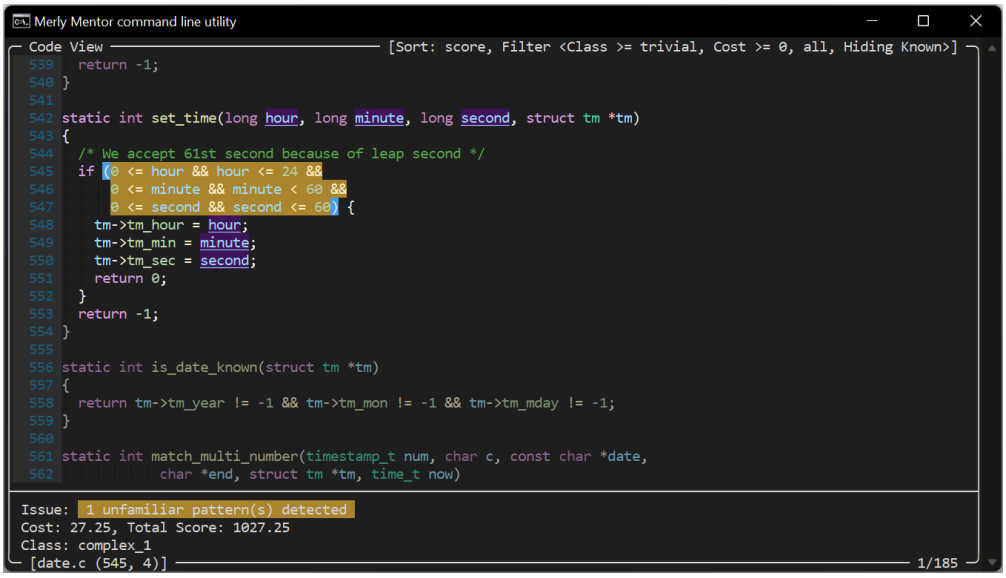


Figure 26: An example of changing the anomaly highlight color in Merly Mentor's Code View.

Or, set `highlight_background` to RGB `4a9de0` to change the highlight color of the non-anomalous expressions to light blue:

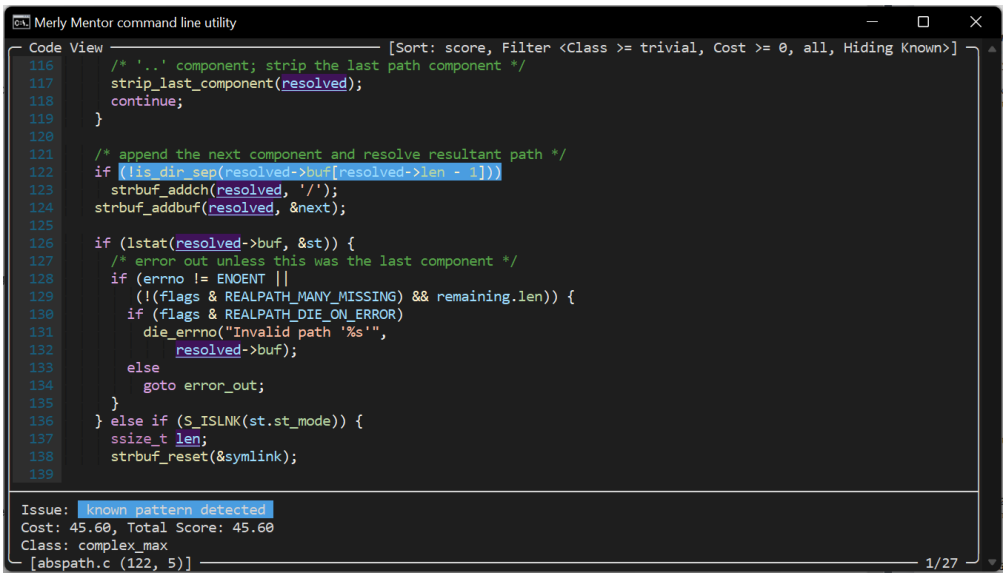


Figure 27: An example of changing the expression highlight color in Mentor's Code View.

10.1.1 Models Path

If you keep various installs of Merly Mentor, or want to have a shared directory of the downloaded models, the models path can be modified in the JSON file, and is initially set to:

`$AWD/models/`

10.1.2 Log Path

The log path can be modified in the JSON file, and is initially set to:

`$AWD/logs/`

10.1.3 Training Settings

Mentor adapts to the code you run inference against. These settings control exactly how Mentor does this.

- `\run_training"` – Determines whether or not training should be run before inference on the source code (defaults to true).
- `\filter"` – Determines if items (such as nested_ifs) are extracted (defaults to true).

11 Exploring Mentor's Inference Results

After inference analysis is performed via the command line (CLI) or other method such as the Mentor Serviceor through a GitHub Action, Mentor will allow a user to interface with the results and explore what Mentor calls "Issues," which are really a number of things, like anomalous expressions, notable expressions, and unique expressions.

11.1 Understanding the User Interface

The most common and main view includes source code with an expression highlighted. We call this screen the *Code View*, which will be described in more detail in Views section of this manual. Figure 28 provides an example of an anomalous code example found by Mentor.

Note: this section deals largely with the CLI mode of running Merly Mentor, but the concepts can be taken and applied to the Mentor Service's web GUI.

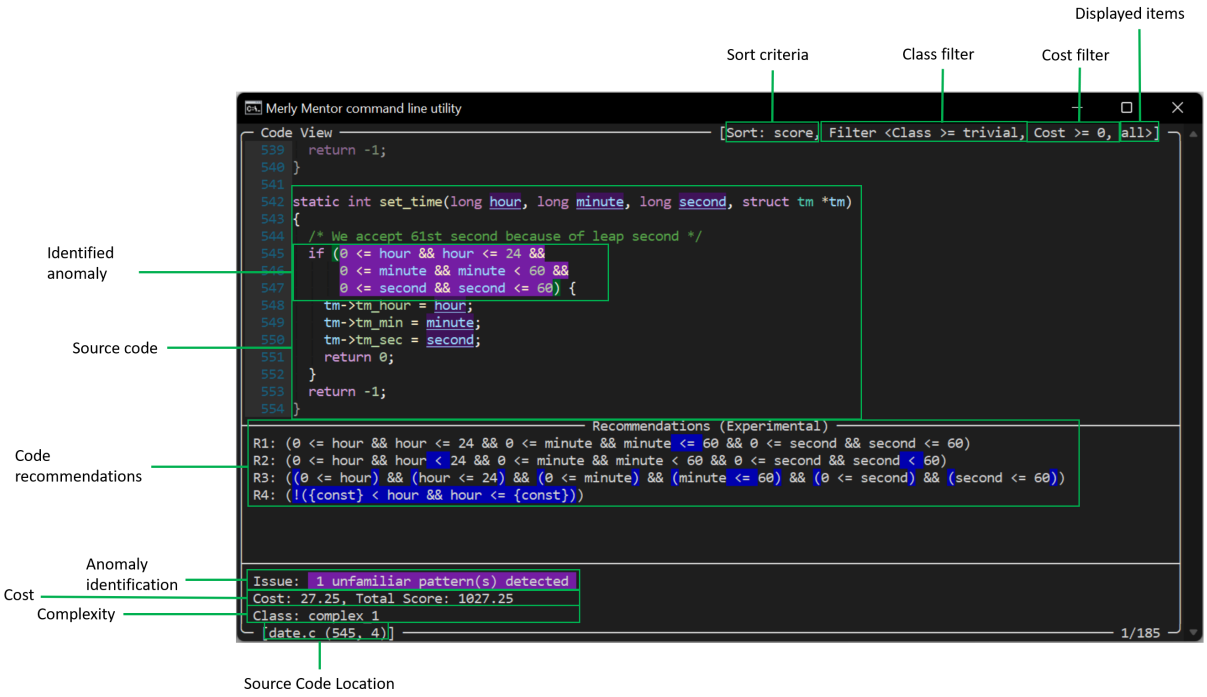


Figure 28: An Anomalous example of results, shown in Code View in Merly Mentor's interactive CLI mode of operation.

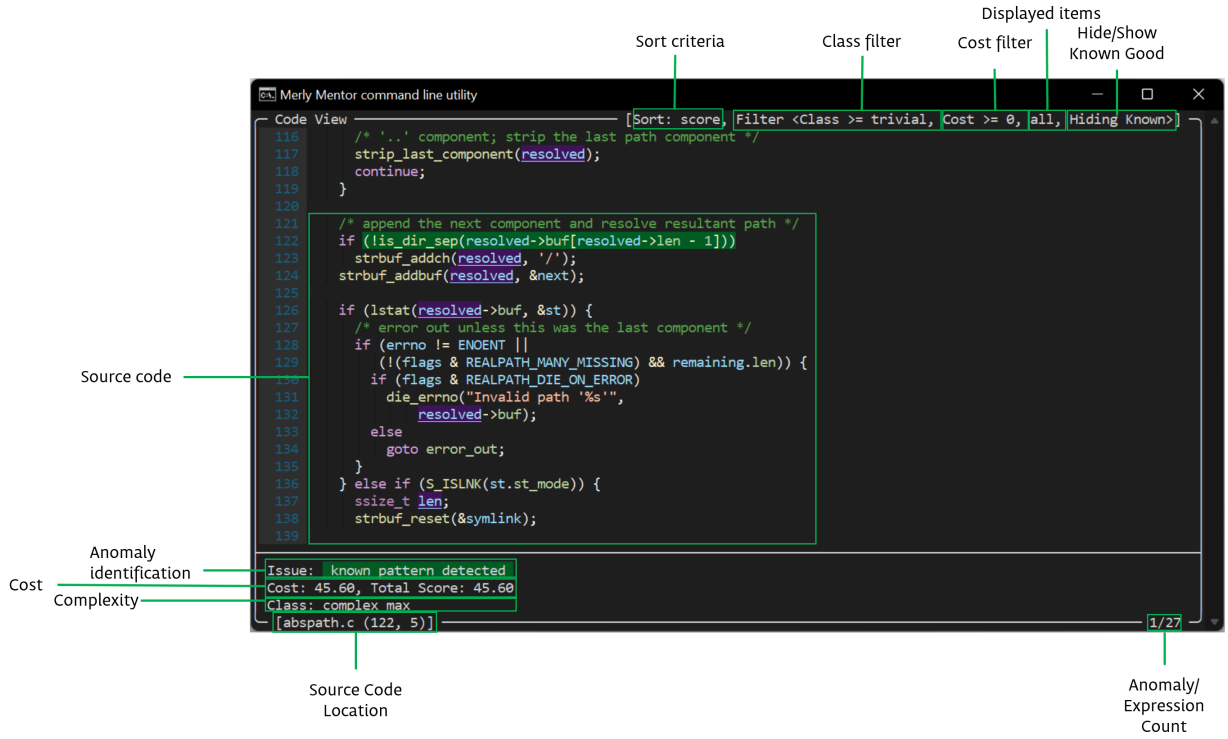


Figure 29: A Non-Anomalous example of results, shown in Code View in Merly Mentor's interactive CLI mode of operation.

11.1.1 Sort Criteria

This refers to how Mentor is sorting the list of expressions it has found. This can be via score (a numeric value assigned by anomaly identification and complexity), or location (sequential code order).

11.1.2 Class Filter

This refers to which class of complexity is being filtered in the current view. This can be set from a minimum value of trivial to a high value of max complexity.

11.1.3 Cost Filter

This refers to a "mental cost" of an expression. This filter can be set from a minimum value of 0 to a maximum value of 2,000.

11.1.4 Displayed Items

This refers to which items Mentor is displaying. It can be set to all expressions, or only anomalous expressions.

11.1.5 Hide/Show Known Good

This refers to whether or not Mentor displays expressions that have been marked by the user as Known Good.

11.1.6 Anomaly Identification

This displays whether or not Mentor has identified the current expression as an anomaly. Non-anomalous expressions will be classified as "known pattern detected" and highlighted in green. Anomalous expressions will be classified as "unfamiliar pattern(s) detected" and will be highlighted in purple.

11.1.7 Cost

This displays the "mental cost" of the current expression.

11.1.8 Complexity

This displays the class of complexity of the current expression.

11.1.9 Source Code Location

This displays the file location of the source code under review.

11.1.10 Anomaly/Expression Count

This displays the count of the highlighted expression, as well as the total expressions found in this file. If the user toggles the filter to show only anomalies, this will display the count of highlighted anomaly, and the total anomalies found in the current file.

11.1.11 Walking Through Code

You can move forwards and backwards through the expressions by using the `left arrow key` and `right arrow key`, and can move by pages through the code (by location) using the `Page Up` and `Page Down` keys on your keyboard. You can also scroll up and down through the code by hold the `Control` key while pressing the up or the `down arrow key`, respectively.

11.2 Classifying Expressions as Known Bad or Known Good

One of the key features of Merly Mentor is the ability to classify expressions as known, whether those known expressions are *good* or *bad*. This classification will allow Mentor to better understand the code under review and identify (or not identify, depending on the classification) other like expressions. This section describes how this functionality works.

First, navigate to an expression that you would like to classify. You can see various ways to view expressions and anomalies in Section 12, and also various ways to sort or filter inference results in Section 13. Once you have the expression identified and highlighted, you can classify it one of the following four ways:

- `G` key - This will mark the expression as a **known good pattern**. This tells Mentor that any pattern in the current repository that matches this one should not be considered an anomaly.
- `P` key - This will mark the expression as a **known bad pattern**. This tells Mentor that any pattern in the current repository that matches this one should be considered an anomaly.
- `H` key - This will mark the expression as a **known good unique expression**. This tells Mentor to ignore the specific instance of this expression and treat it as non-anomalous, but not to ignore (or treat as non-anomalous) any other matches to this pattern in the repository.
- `O` key - This will mark the expression as a **known bad unique expression**. This tells Mentor to recognize the specific instance of this expression as an anomaly, but not to use this pattern to identify other expressions as anomalies in the repository.

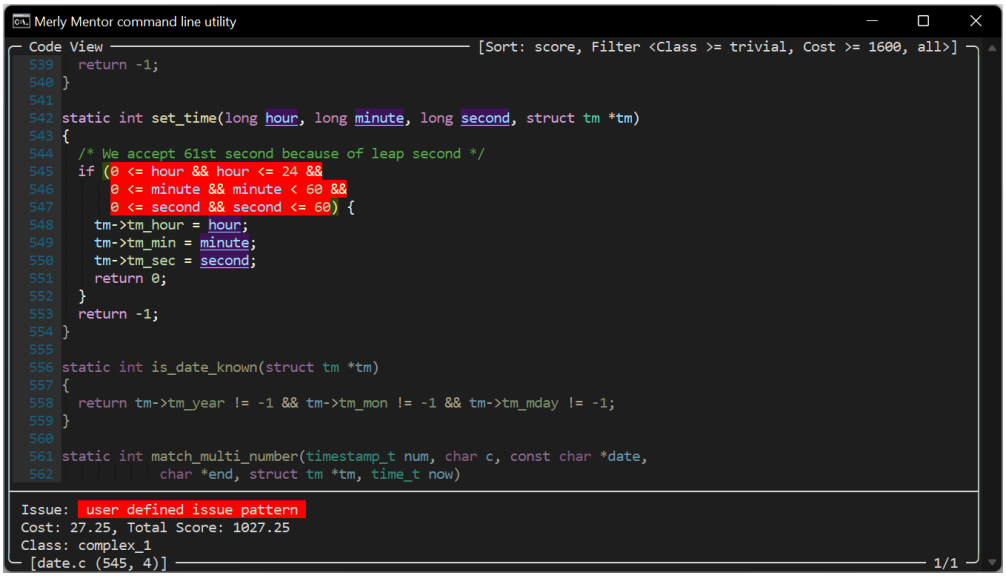


Figure 30: An expression marked as a known bad by the user.

Once you have classified expressions as known, you may want to see a list of all of the expressions that have been classified in this fashion. In order to see all of the known expressions, press the `K` key. You can then cycle through the good or bad known expressions by pressing the `b` key.

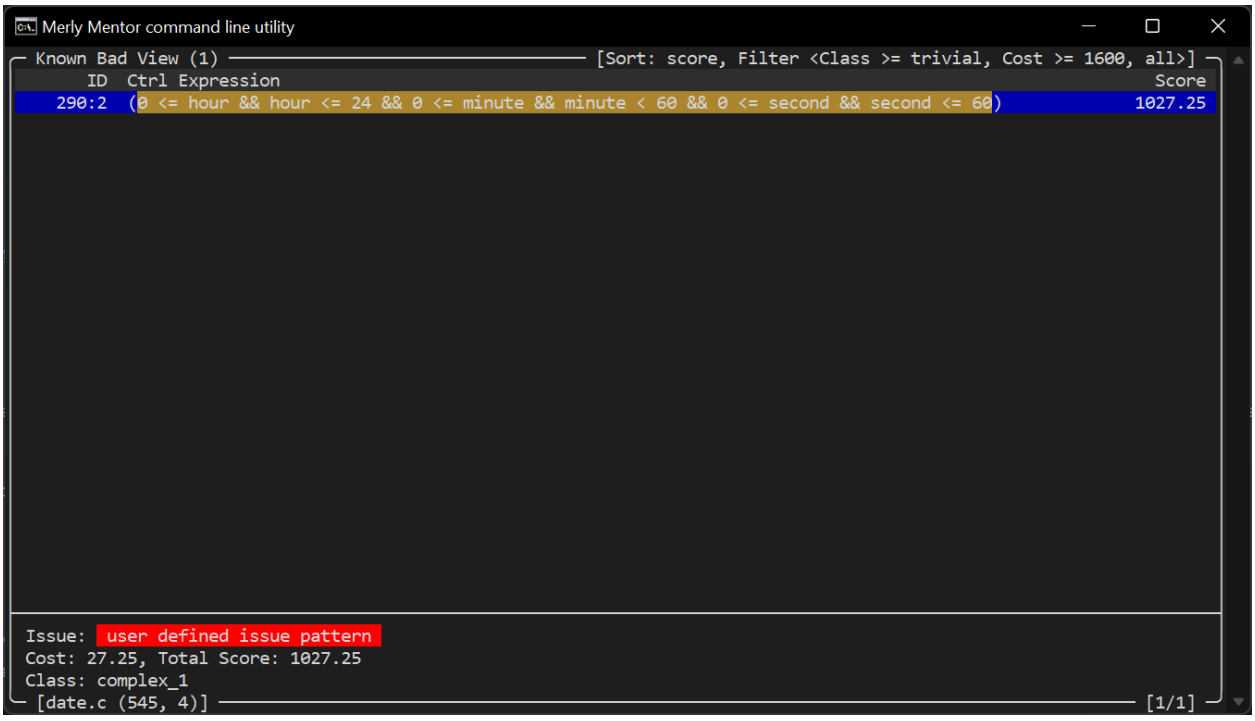


Figure 31: Mentor displaying a list of expressions marked as known bad by the user.

11.3 Recommendations

In addition to identifying anomalies, Merly Mentor can provide recommended changes to the anomalies it identifies. To use this feature, select an anomaly (see Section 12 for more details) and then press `r`. Mentor will then

provide a recommendation of a possible change (or changes). This feature is still experimental and results may vary.

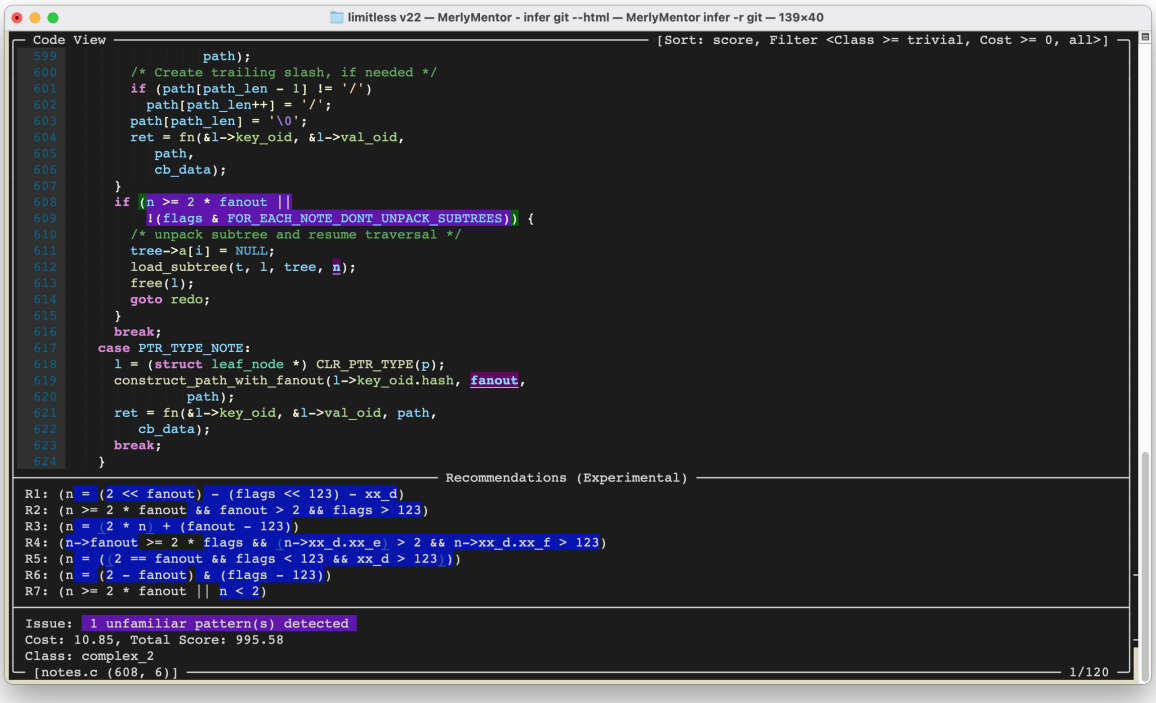


Figure 32: An example of Merly Mentor providing recommended code changes.

In Mentor's (Experimental) Recommendation list, it contains a partial recommendation that, if applied, would correct the original, incorrect code.

12 Basic Interactive Commands and Views For the Mentor CLI Modes

In Mentor, there are a number of supported keyboard and mouse commands. In this section we describe those keystrokes and explain mouse behavior. Perhaps the most important initial command to remember is the *help* command which can be launched by pressing the character `h` on your keyboard. The help command lists all of the keyboard commands, so if you ever find yourself not remembering a keyboard command, just press `h` and Mentor will launch the keyboard shortcut commands. A screenshot of the help dialog box is shown in Figure 41.

Additionally, a dynamically generated list of options specific to the user's current page can be accessed by selecting `t`. This feature is intended to help users navigate within Mentor more easily; providing them with just the most commonly used commands in whatever view they are currently in.

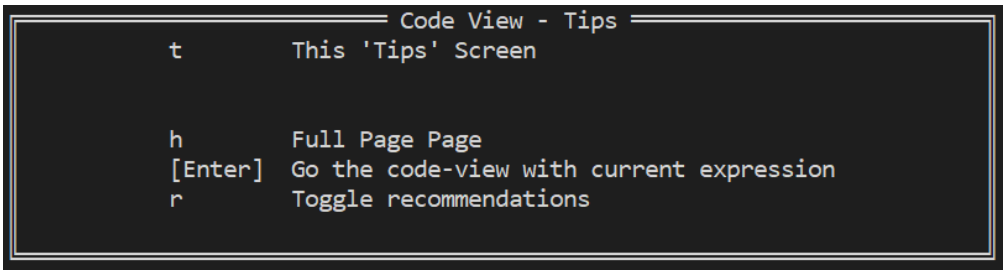


Figure 33: Merly Mentor's tips popup.

In addition to commands, there are several screens (referred to as *views* in this manual) that users can utilize to help them gain deeper insights into specific anomalies, general anomaly information, anomalies by file, anomalies per file, and so forth.

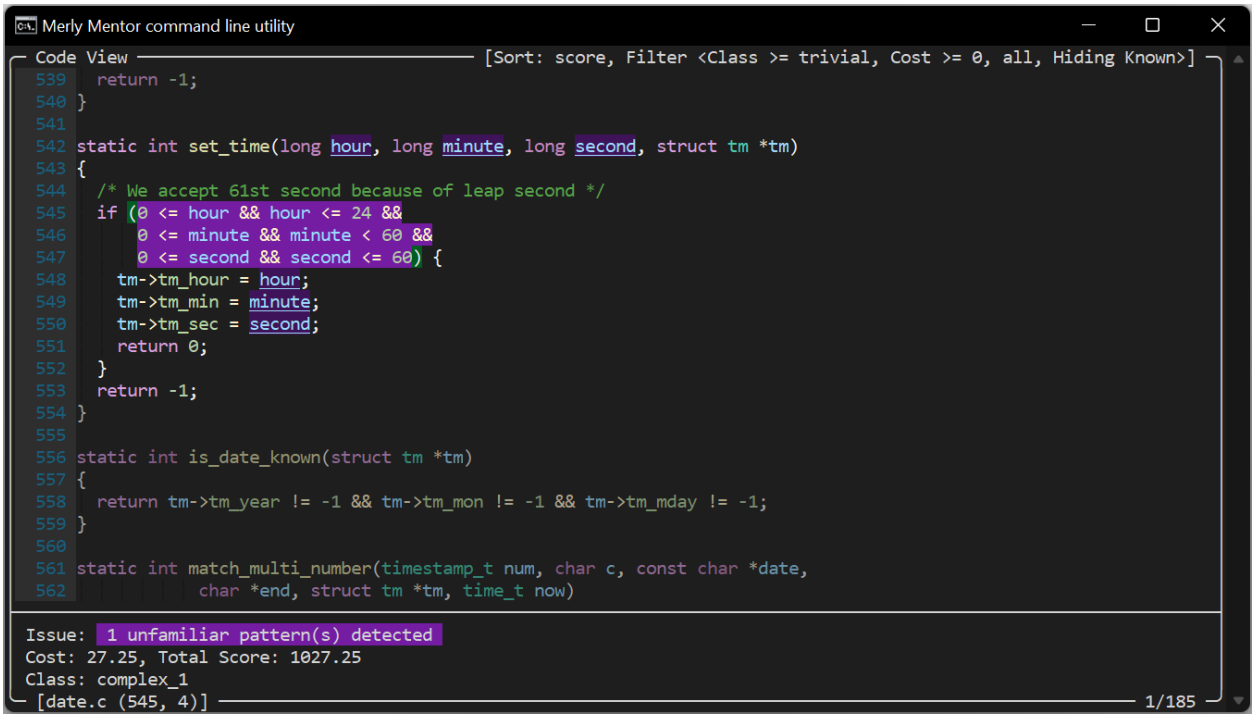


Figure 34: Merly Mentor's Code View.

12.1 Code View

The Code View is the default view that Mentor enters when launched. It displays some of the surrounding contextual code for a given piece of code that Mentor has analyzed and flagged. In the Code View, a user can iterate over all of the segments of code that Mentor has analyzed using a variety of keyboard and mouse commands (see Section 11 for more details). The Code View has several utilities, perhaps the most important is to help users understand the surrounding context of code for a given analyzed code segment to help them determine if an action should (or should not) be taken for that code fragment.

The intuition behind the Code View is that if a code fragment is deemed interesting (or uninteresting) a developer will likely need to understand the surrounding code context. Code View attempts to provide such surrounding context for the user. Often times what is shown on the screen may be insufficient context to deeply reason about the code. To help resolve this, a user can scroll up and down using the keyboard or mouse (more details in Section 11).

There are generally two types of labels that Mentor assigns to code it has analyzed: **unfamiliar pattern** (e.g., potentially anomalous, which could be a *defect* or *technical debt*), or **familiar pattern** (e.g., code that is unlikely to be the source of technical debt or a defect). By default Mentor color codes unfamiliar and familiar patterns differently so users can visually discern the difference. Users can change these colors as discussed in Section 10.

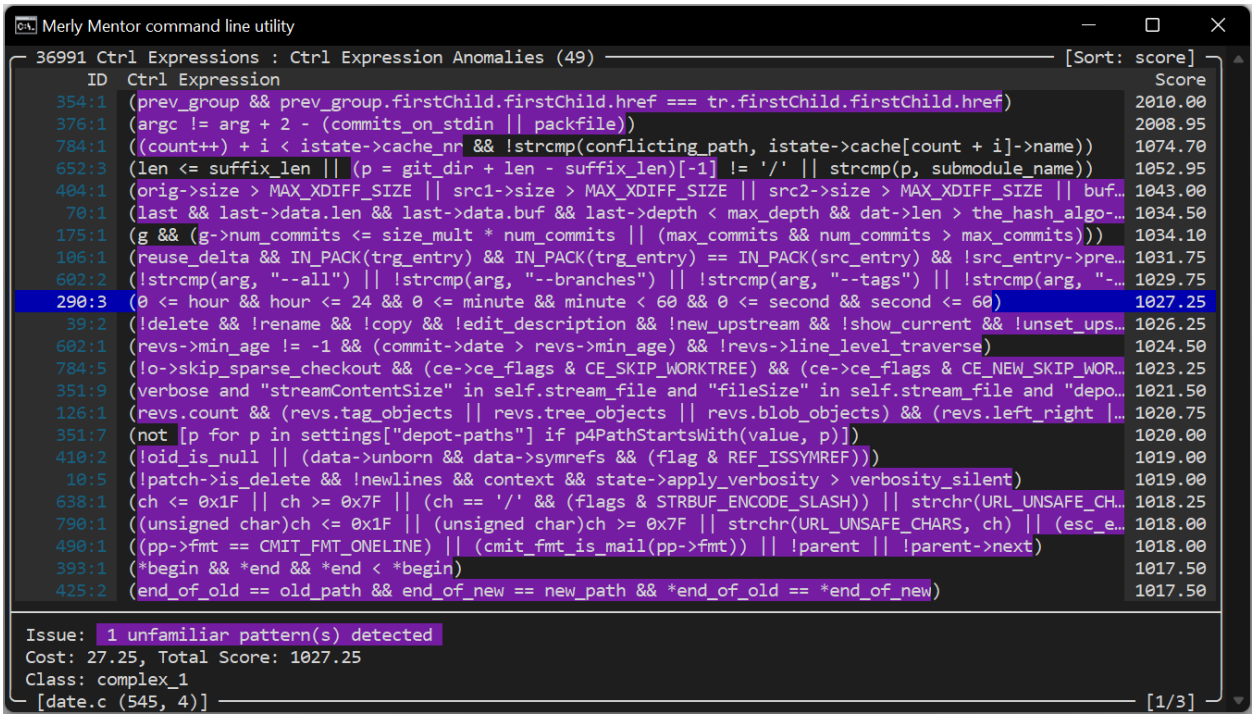


Figure 35: Merly Mentor's Anomalies View.

12.2 Anomalies View

The Anomalies View can be accessed by pressing the `a` key. This view filters out code fragments identified as non-anomalous by Mentor, and only displays code fragments (across all files) that Mentor identifies as anomalous.

The intuition behind the Anomalies View is that a developer may be interested in looking only at each of the code fragments that Mentor identified as anomalies, instead a full list that includes non-anomalous code fragments. This can allow the developer to focus on code fragmentsthat may need further review for possible updating. Press `Enter` with an anomaly highlighted to switch back to the Code View of that specific anomaly.

Figure 36: Merly Mentor's Notable Expressions View.

12.3 Notables View

The Notables View can be accessed pressing `A`. This view provides a list of code fragments identified as notable by Merly Mentor.

The intuition behind the Notables View is that a developer may be interested in code fragments that potentially have a high *mental cost* associated with understanding them. These may be code fragments that are complex and/or harder to read. The developers of Merly Mentor believe that these types of code fragments generally have a tendency to be more *dangerous* in nature, meaning a higher likelihood of being responsible for technical debt and/or the root cause of future defects if/when this code fragment is called. The developers of Mentor believe that reviewing these code fragments and simplifying them where possible will lead to better understanding of the code, and potential proactive reduction of future defects.

The Notable Expressions View uses a scoring algorithm that lists the code fragments from most complex to least complex. The user can scroll through the list in the same fashion as other views, using the `up arrow`, `down arrow`, `Page Up`, and `Page Down`.

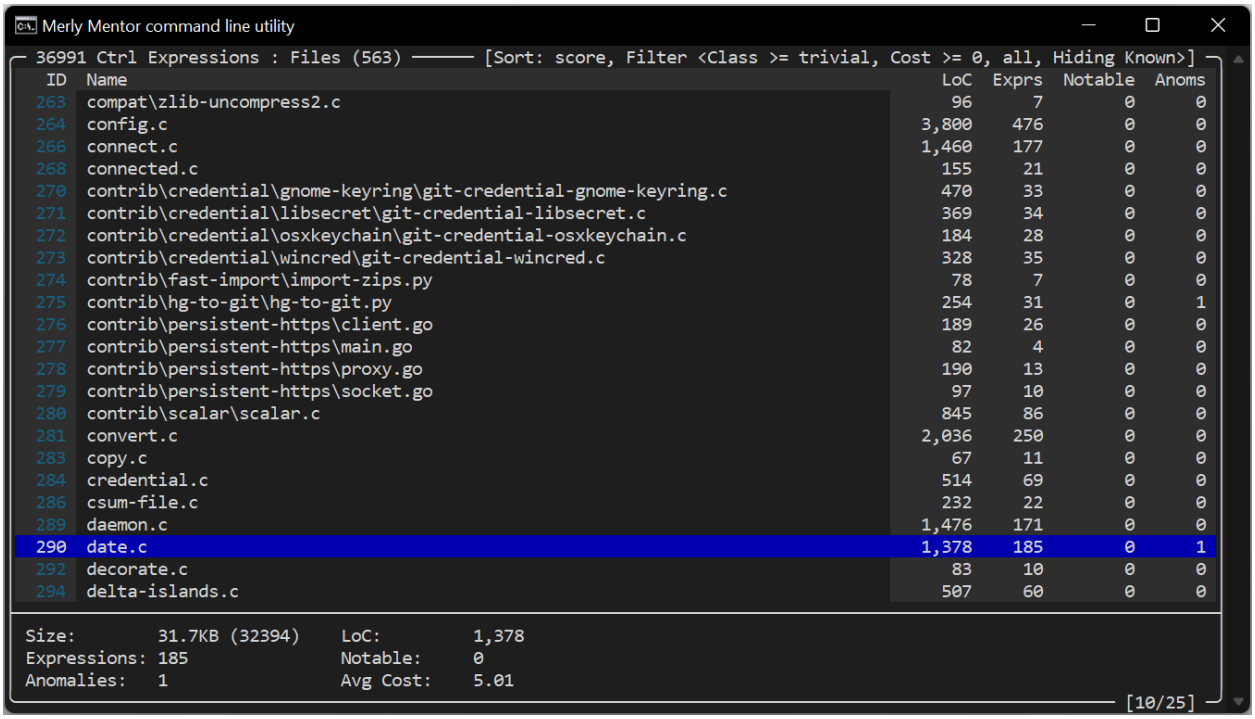


Figure 37: Merly Mentor's Files View.

12.4 Files View

Depending on the amount of code being reviewed, it may be beneficial to target a specific file or set of files to review. Mentor has a way to review a specific file called the Files View. Press `f` to switch to the Files View. This view shows all of the source code files, with the total number of expressions found in each file. You can move up and down the list using the `up` and `down` arrows, or the `Page Up` and `Page Down` keys on your keyboard.

Press `Enter` with a file highlighted to switch back to the Code View of the code fragments within that specific file.

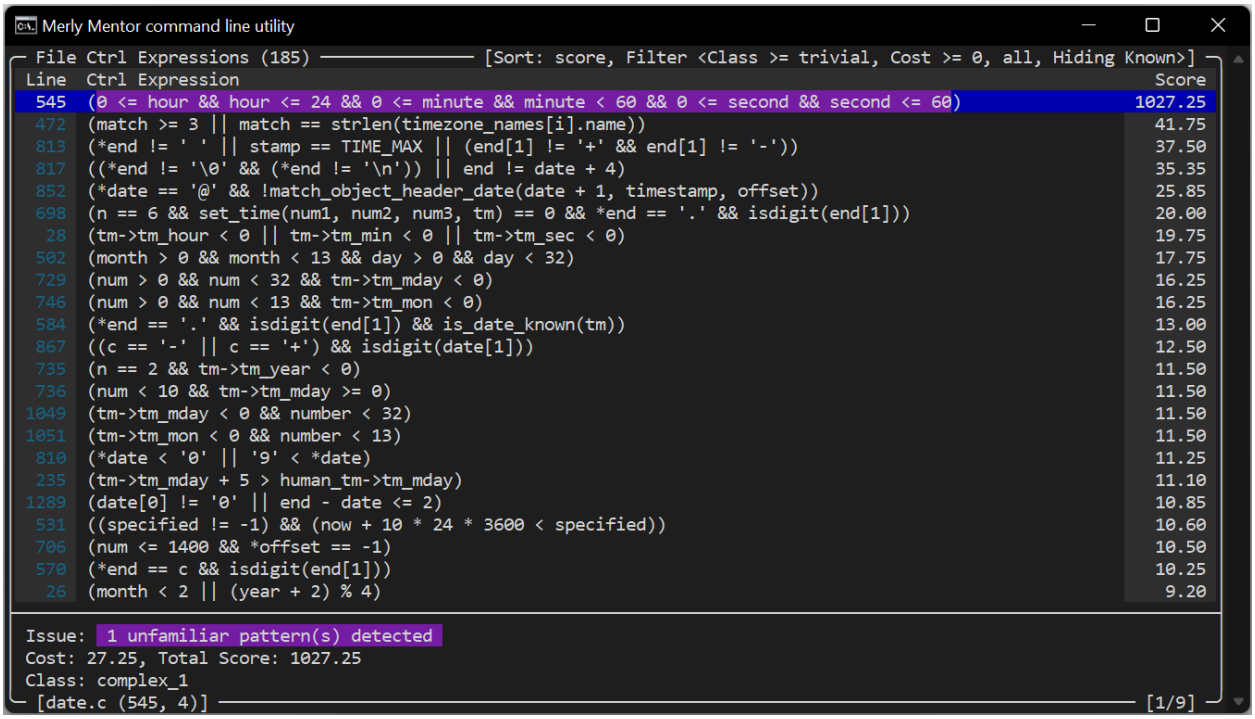


Figure 38: Merly Mentor's Expressions View.

12.5 Expressions View

If there's an extensive amount of code to review, it can be easier to review the code by the code fragments Mentor found. The Expressions View allows the user to do this. Press `e` to switch to the Expressions View. This view shows all of the code fragments in the current file, sorted by score. Using this view will allow you to see the anomalies in the current file, as they are listed at the top of the view. You can move up and down the list using the `up` and `down` arrows, or the `Page Up` and `Page Down` keys. Also note that you can toggle the sort between code location and score by pressing the `s` key.

Press `Enter` with a code fragment highlighted to switch back to the Code View with that specific code fragment highlighted.

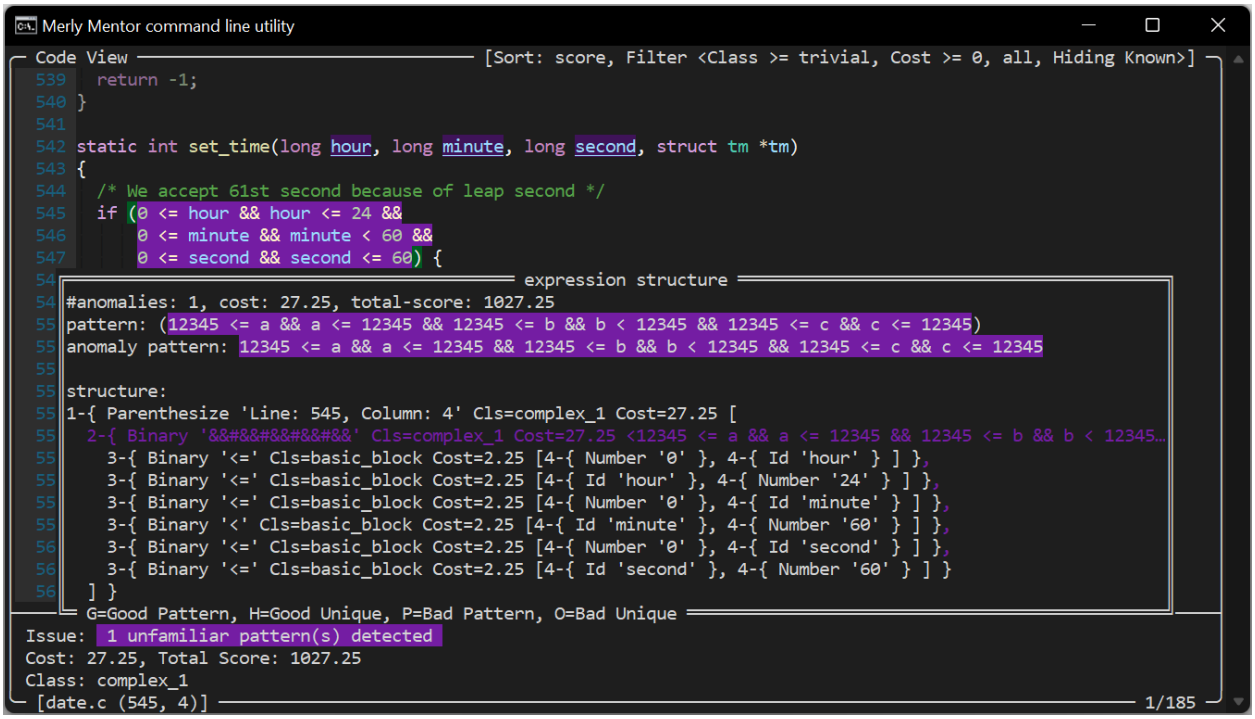


Figure 39: Merly Mentor's Details View.

12.6 Details View

Once you identify a code fragment that piques your interest, you may want to review it in more detail. You can do this via the Details view. Press `d` to switch to the Details view. This view shows the detail of the currently selected code fragment (anomalous or non-anomalous). The detail lets you know how many anomalies Mentor identified within the code fragment, as well as the cost and the total score. Drilling into these details can provide additional information about the code fragment, helping to determine if and where the code might need to be updated.

Press `d` to return to Code View.

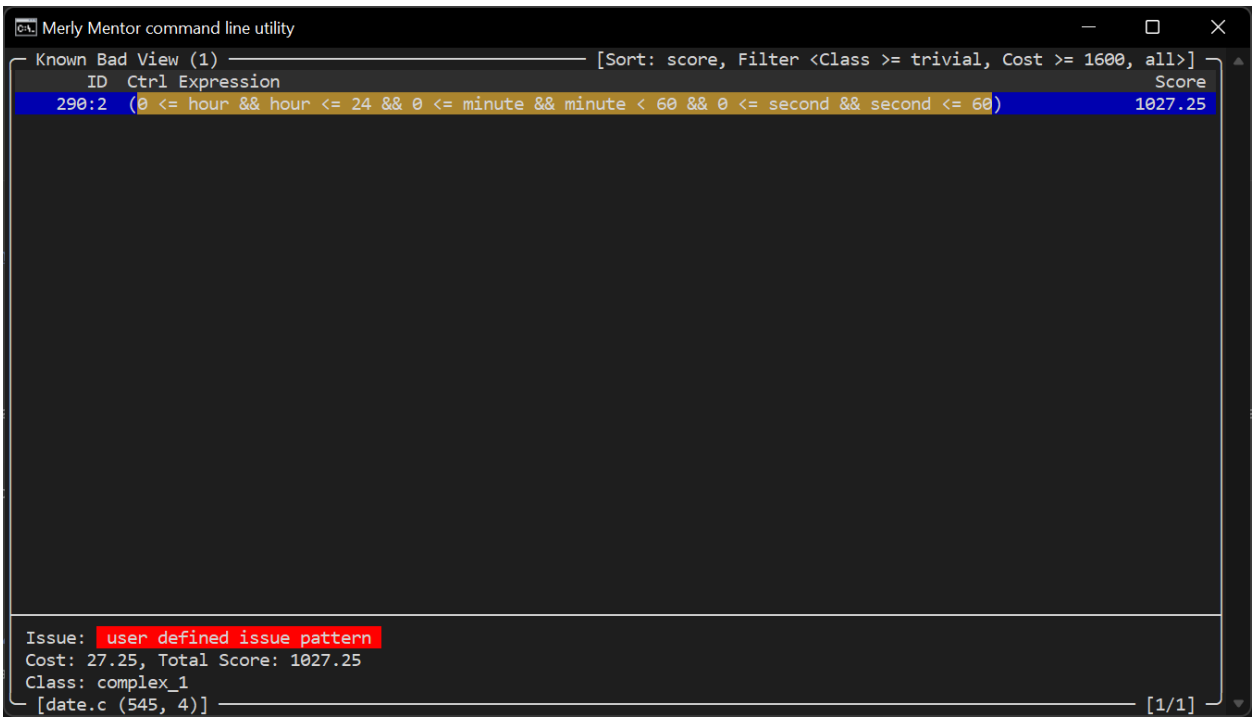


Figure 40: Merly Mentor's Known View.

12.7 Known View

For users that have identified known good or bad expressions (`G` : Learn Good Pattern, `H` : Learn Good Unique, `P` : Known Bad Pattern, `O` : Learn Bad Unique, `R` : Remove Known Entry), the list of identified items can be large and difficult to reason about without showing all items marked in this way. Pressing the `K` key will show the user a list of all expressions in the current code identified as known good. This can be toggled to showing identified known bad expressions by pressing the `b` key.

Press `ESC` to return to Code View.

(Note: for full details on classifying expressions and why you would or would not want to do this can be found here: Section on Classifying)

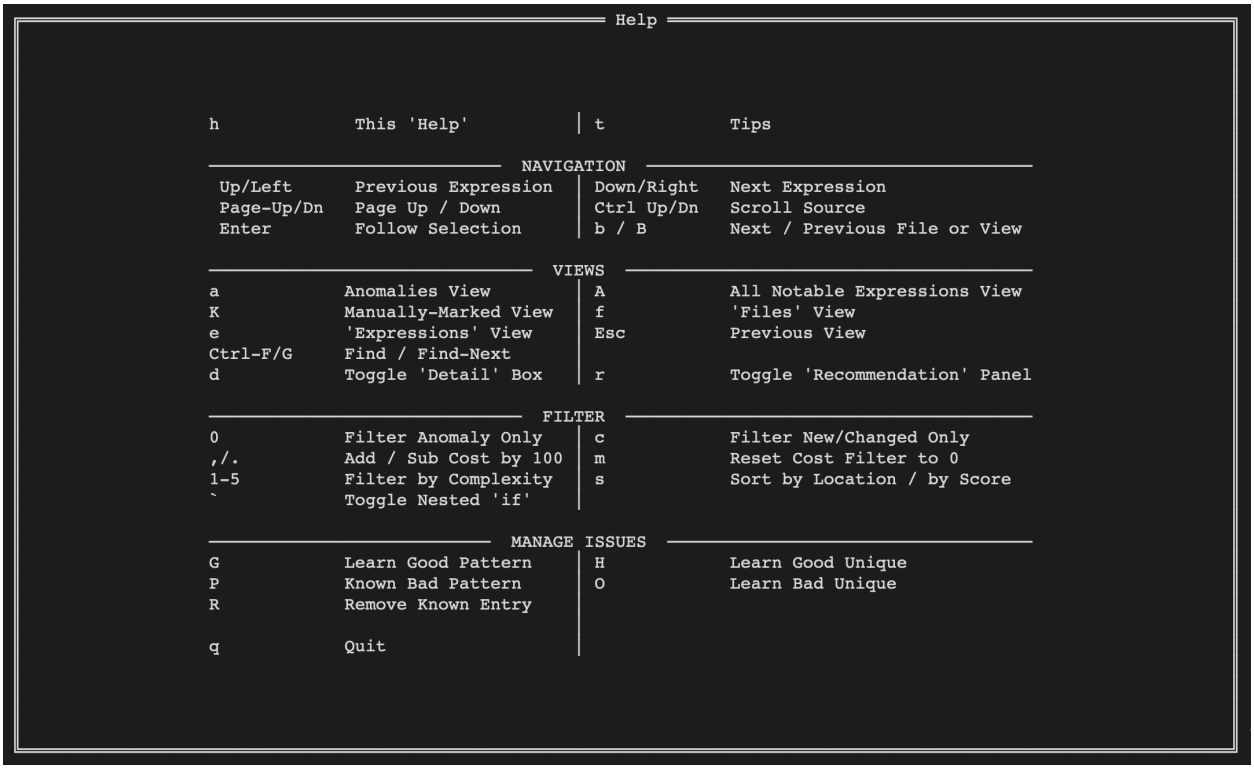


Figure 41: Merly Mentor's Help Dialog Box.

12.8 Help Pop-up

In addition to the above views, you can press the `h` key in any view to bring up the help screen which will show you all of the hotkeys and their functions.

13 Sorting/Filtering Inference Results From the Mentor CLI Modes

In Mentor, there are a number of ways to sort or filter inference results. Looking at inference results sorted or filtered in different ways can oftentimes provide insights or reveal patterns that could be difficult to visualize or reason about otherwise.

The following subsections list the ways Mentor's inference results on source code data can be sorted and/or filtered.

13.1 Sort Criteria

Inference results can be sorted in two distinct ways: by score, or by location. *Score* refers to a numeric value assigned to anomalies based upon complexity, and *location* refers to the sequential order of the code under review. Initially, the results are sorted by score, but this can be toggled between score and location by pressing the `s` key.

Options:

- Score (numeric value assigned by anomaly identification and complexity)
- Location (sequential code order)

Default: Score

Toggle: `s` key

13.2 Class Filter

Inference results can also be filtered by the level of complexity. This is called the class filter. This filter is initially set to trivial, which is the minimum level of complexity. However, this can be increased to filter out increasingly higher levels of complexity. There are 5 levels of increasing complexity levels defined by Mentor: *trivial* (the minimum level), *basic*, *complex 1*, *complex 2*, and the highest level of complexity, *complex max*.

13.2.1 Class Complexity

When using the class filter, it can be helpful to have an understanding of the intuition behind the different complexity levels. The developers of Mentor generally think about complexity in one of two ways: building blocks, or a tree structure.

For the building blocks example, it can be helpful to envision the lowest level of complexity as a single building block. Each level of complexity includes more connected building blocks, adding to the complexity of the structure.

Likewise, for the tree structure, additional levels of complexity add branches and nodes to the tree, which makes the structure more complex.

Figure 42 includes a visual aid to help you reason about the complexity tiers, and determine how to set the filter in a way that makes the most sense for you and your organization.

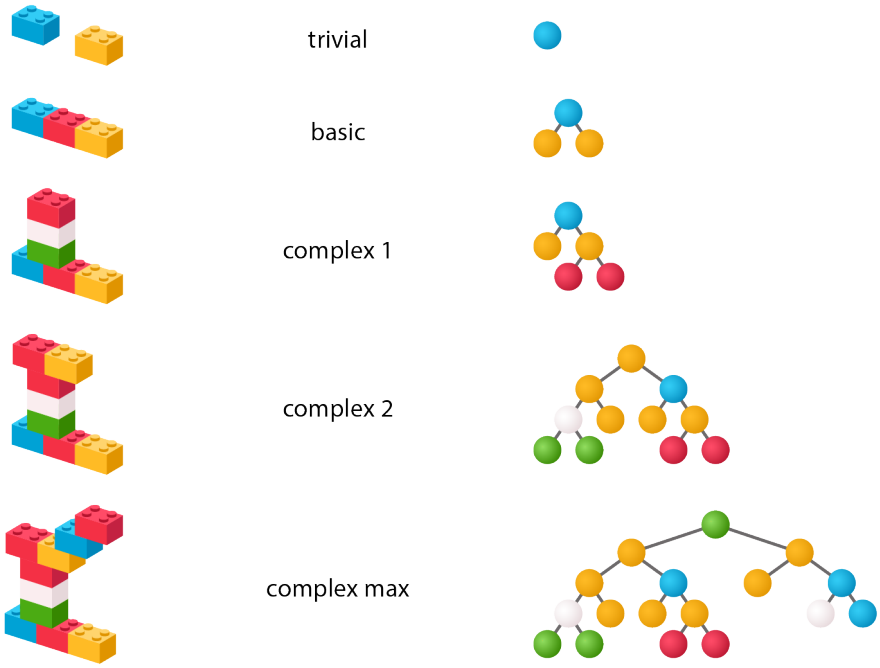


Figure 42: It can be helpful to reason about Merly Mentor's class complexity from a building block or tree structure perspective.

These filters can be accessed directly by pressing keys associated with the complexity levels:

- `1` key - Trivial (minimum)
- `2` key - Basic
- `3` key - Complex 1
- `4` key - Complex 2
- `5` key - Complex Max

Options:

- Trivial (minimum)
- Basic
- Complex 1
- Complex 2
- Max

Default: Trivial

Adjust: `1`, `2`, `3`, `4`, `5` keys

13.3 Cost Filter

Inference results can also be filtered by *cost*. Cost refers to the *mental cost* associated with reviewing and reasoning about a particular line of code. The higher the cost score, the more difficult it is to reason about the code under review. This filter is initially set to 0, and can be scaled up to a maximum value of 2,000. You can scale the cost filter by pressing the `,` key to decrease the filter or the `.` key to increase the filter. You can reset the filter to 0 by pressing the `m` key.

Options:

- 0 (minimum) to 2,000

Default: 0

Adjust: `,` to decrease, `.` to increase, `m` to reset to 0 (minimum)

13.4 Displayed Items Filter

You can also change the items Mentor displays in the inference results. Initially, all expressions are displayed, but you can filter the results to only display anomalies by pressing the `0` key. You can toggle back to showing all anomalies by pressing the same `0` key.

Options:

- All expressions
- Anomalies Only

Default: All expressions

Toggle: `0` key

13.5 Hide/Show Known Good Filter

For users that have identified known good patterns and would like to view them, this can be done by pressing the `9` key. Doing so will show the user patterns in the current code identified as known good. This can be toggled back to hiding known good instances by pressing the `9` key again.

Options:

- Hide Known Good
- Show Known Good

Default: Hide Known Good

Toggle: `9` key

14 Inference Comparison

If Merly Mentor has run inference on a code repository at two or more distinct points in time, Mentor can provide a comparison and show differences in the repository over those points in time. This can be helpful when trying to determine when an anomaly or notable expression was introduced, or to review any two revisions to each other as a way to reason about code quality as it has progressed over time. In this section, this functionality is explained.

14.1 Generating Multiple Inference Instances

The first step in utilizing Mentor's comparison functionality is ensuring that there are at least two separate instances of inference data that Mentor can compare against.

The easiest way to do this is imply to run inference on the repository at regular intervals. Currently, the smallest level of granularity is daily. Each day inference is run, Mentor will create a folder in the current directory that houses all of the inference data. Mentor can then use these folders to run inference comparisons.

Additionally, if you have access to the code repository at specific historical points in time, you can have Mentor create these inference folders retroactively and then compare them.

14.2 Git Repositories

Because Mentor is integrated with Git, running inference against a specific Git revision is straightforward. Using one of two commands to point to specific points in time, you can choose a specific point for inference.

14.2.1 Using a Git Commit's SHA (–commit flag)

The first way to choose a specific point in time, rather, a specific commit, the `--commit` argument is passed when performing inference. Note: using the Mentor command `usage` shows you can also use the shortcut `-c` to invoke a specific commit.

If you know the specific Git commit's Simple Hashing Algorithm (SHA) you would like to run inference against, you can use that SHA in concert with the `-c` argument on the command line when running inference. For example, you could run the following:

```
$ ./merlymentor infer -r git -c 966ff64a3042f879bca83e2376128cf5f39737a7
```

In this example, `966ff64a3042f879bca83e2376128cf5f39737a7` is the SHA that Merly Mentor will run inference against.

Mentor recognizes the standard Git references for revisions, so alternately, you could also run:

```
$ ./merlymentor infer -r git -c HEAD
```

This command would have Merly Mentor run inference against HEAD.

Repeating this process on at least two separate revisions would provide Merly Mentor with the data to run an inference comparison.

14.2.2 Using a Number to Determine Inference Instances (–p flag)

Another way to create multiple inference instances from a Git repository is to make use of the `-p` argument. This is command is especially helpful if you are interested in generating multiple inference instances over time and do not necessarily need to target any specific revision date*.

To use this feature, invoke the `-p` argument with a number of commits you'd like Mentor to execute inference on. For example, you could run:

```
$ ./merlymentor infer -n -r git -p 10
```

Explanation: This invokes Mentor and runs inference in (-n)on-interactive mode. The `-r git` flag and value specifies the repo git*, and then the `-p 10` flag and value provides Mentor the search depth. This would generate ten inference folders for comparison, as seen in Figure 43.*

*The first inference point will be the earliest commit; the second, the latest. An algorithm divides any number of other points as indicated and matches them to a date that fits closely within a margin.

Note that the `-p` argument must be run in non-interactive (`-n`) mode in order to fully build the inference output folders.

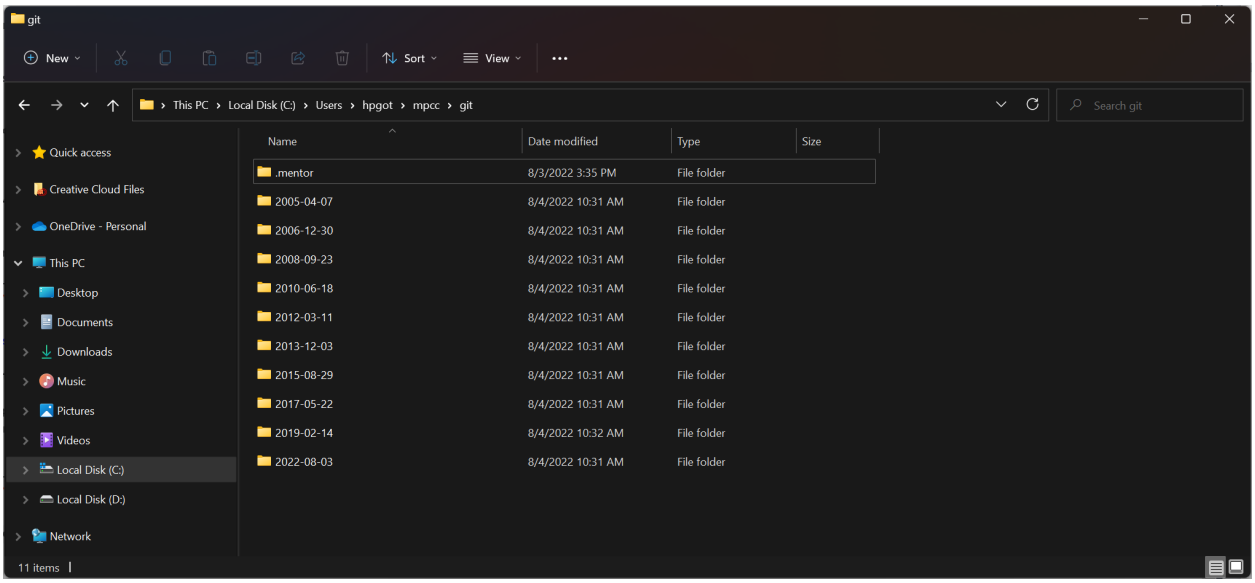


Figure 43: Multiple inference folders can be created by Merly Mentor using the -p argument.

14.3 Non-Git Repositories

If you are using a code repository tool other than Git (like P4), it is still possible to create multiple inference folders for other code repository tools. If you have access to code repositories at specific moments in time, you can tell Mentor to create folders for those revisions.

In order to create the inference folders correctly, it is important to have three pieces of information in advance of running inference:

- the *location* of the code repository
- a *desired folder name* to house the individual inference runs
- the *date* of check-in for the repository you are running inference against

Knowing these items will allow you to place the inference results in the proper folders. Now, look at how we would structure an inference command using these pieces of information:

1. First, use the location of the code repository with the `-D` (directory) argument.
2. Next, use the `--output-name` (`-o`) argument for the main folder that will house the inference folders.
3. Finally, use the `--date` (`-d`) argument for the date folder within the main folder (in the format of `yyyy-mm-dd`)

Specifically, the command would be structured as follows:

```
$ ./merlymentor infer -n -D <source code path> -o <output name> -d <date>
```

A full command example would look like this:

```
$ ./merlymentor infer -n -D /users/user/src/git -o git -d 2022-08-03
```

Running this command would run inference on the repository located at `/users/user/src/git`, creating (or using) a folder called `git`, and creating a 2022-08-03 sub-folder for the results from this specific inference run. See Figure 44 for an example output that would be generated from this command.

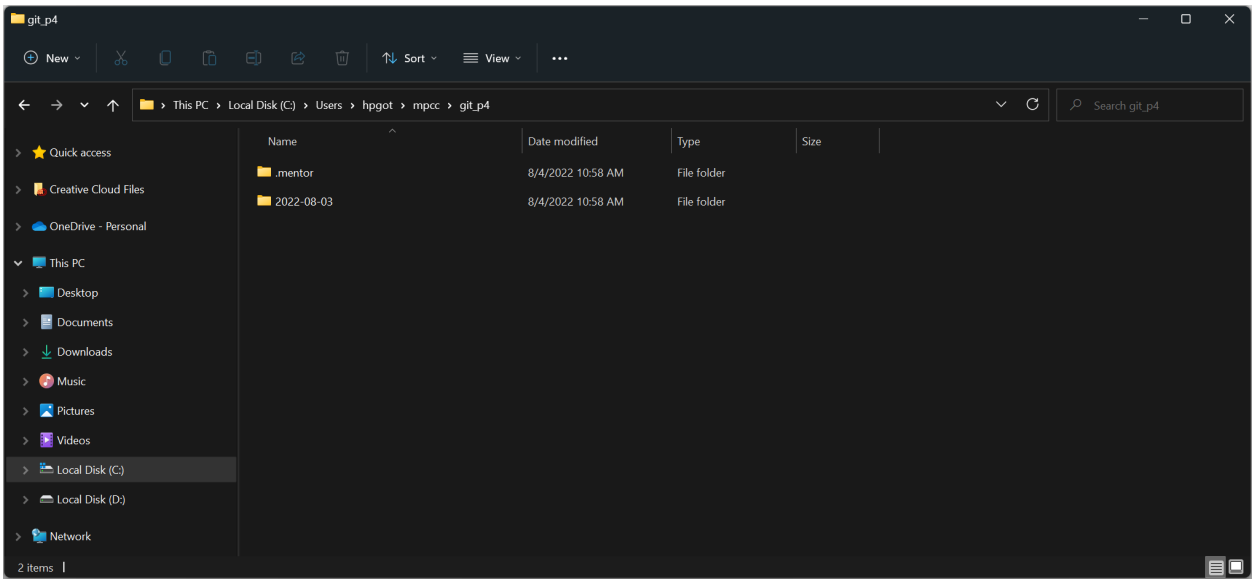


Figure 44: A date sub-folder created by Merly Mentor using the -o and -d arguments.

Running this command multiple times for different repositories would create multiple inference instances for Mentor to compare. For example, you could run the following commands in succession:

```
$ ./merlymentor infer -n -D /users/user/src/git -o git -d 2022-08-03
```

```
$ ./merlymentor infer -n -D /users/user/src/git -o git -d 2022-07-27
```

```
$ ./merlymentor infer -n -D /users/user/src/git -o git -d 2022-07-19
```

This would create the inference folders *like* shown in Figure 45.

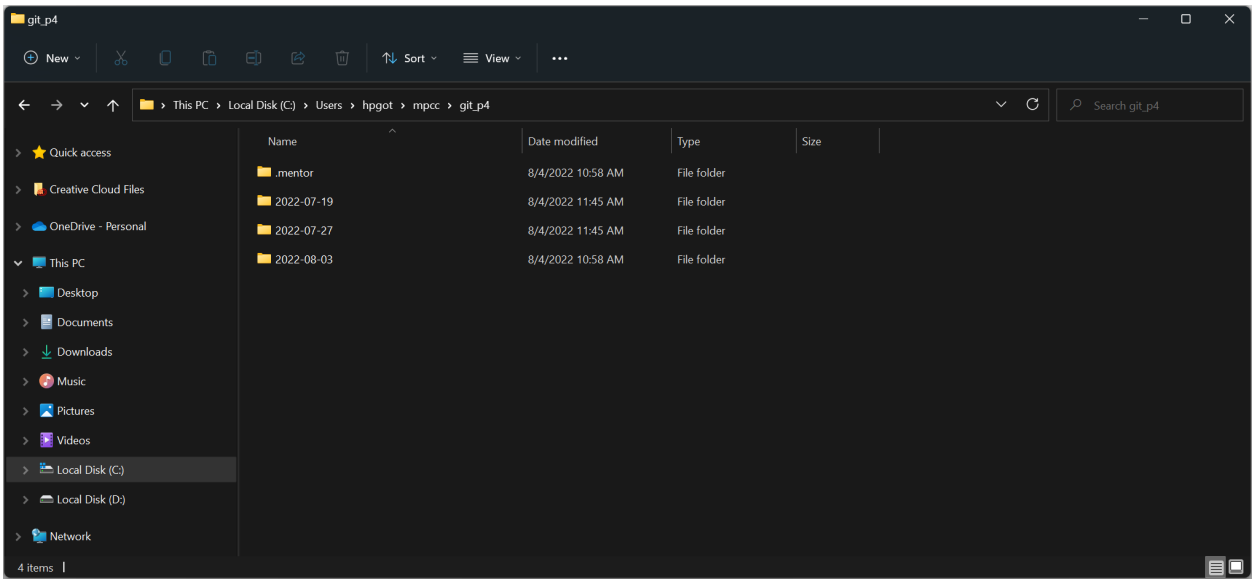


Figure 45: A series of date sub-folders created by Merly Mentor using the `-o` and `-d` arguments.

14.4 Comparing Results Across Multiple Inference Runs

Now that you have multiple folders with inference data, we will look at how Mentor can show this data and compare one repository revision to another **from within the interactive view on the command line interface (CLI).**

Be sure to have at least one previous inference folder created from an earlier version of the source code. See previous section (14.1) for details. Also, be sure to have run inference on the latest revision of the source code repository.

14.4.1 Viewing New Anomalies in Merly Mentor

The first way to do this is within Mentor using the `c` command. After running inference, go to the Anomalies view by pressing `a`. Doing so will list all of the anomalies that Mentor has identified.

From the Anomalies View, press `c` to see only the anomalies introduced with the latest revision. The displayed items indicator at the upper right changes from `all` to `New` when you toggle the `c` key.

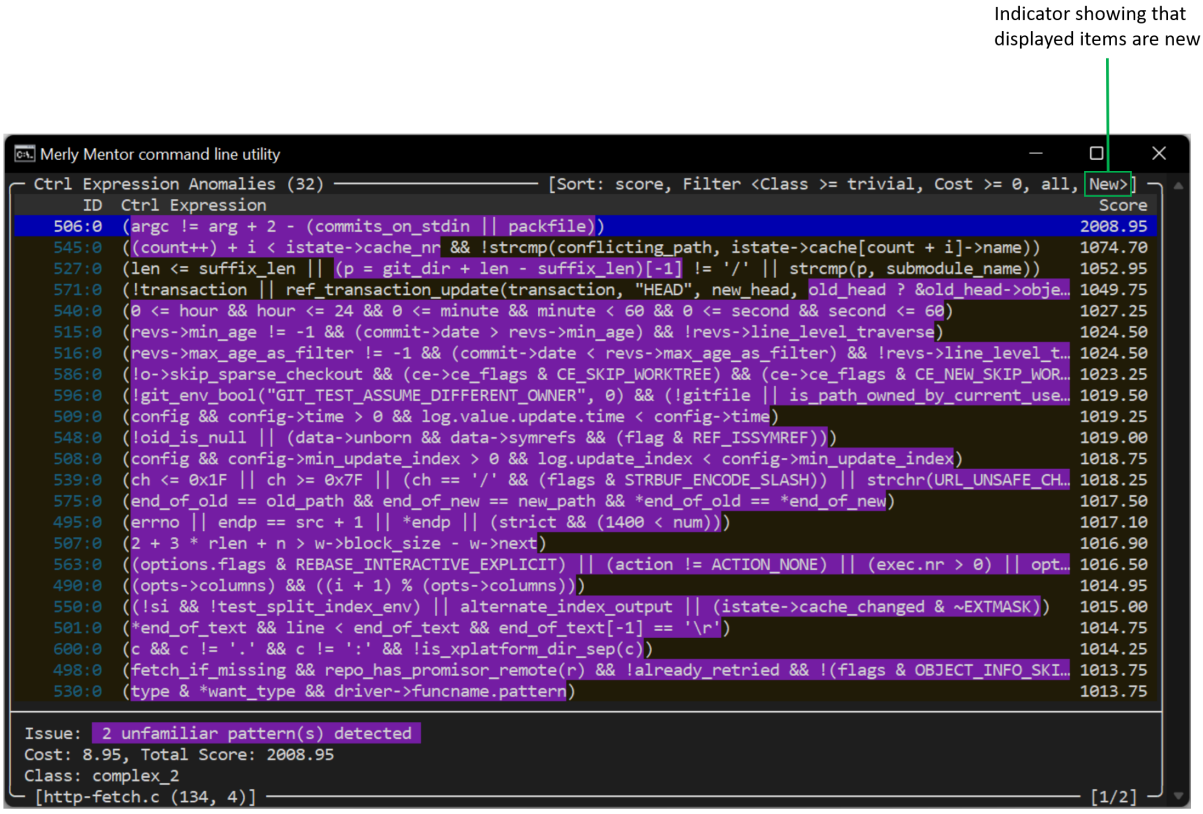


Figure 46: Merly Mentor showing a list of anomalies that were introduced with the latest revision of the code repository.

14.4.2 Viewing New Notables in Merly Mentor

This process can also be completed for notable expressions. After running inference on the latest revision, go to the Notables View by pressing `A`. Doing so will list all of the notables that Mentor has identified. From the Notables View, press `c` to enable the filter and only display the notables introduced with the latest revision. The displayed items indicator at the upper right changes from `all` to `New` when you toggle the `c` key.

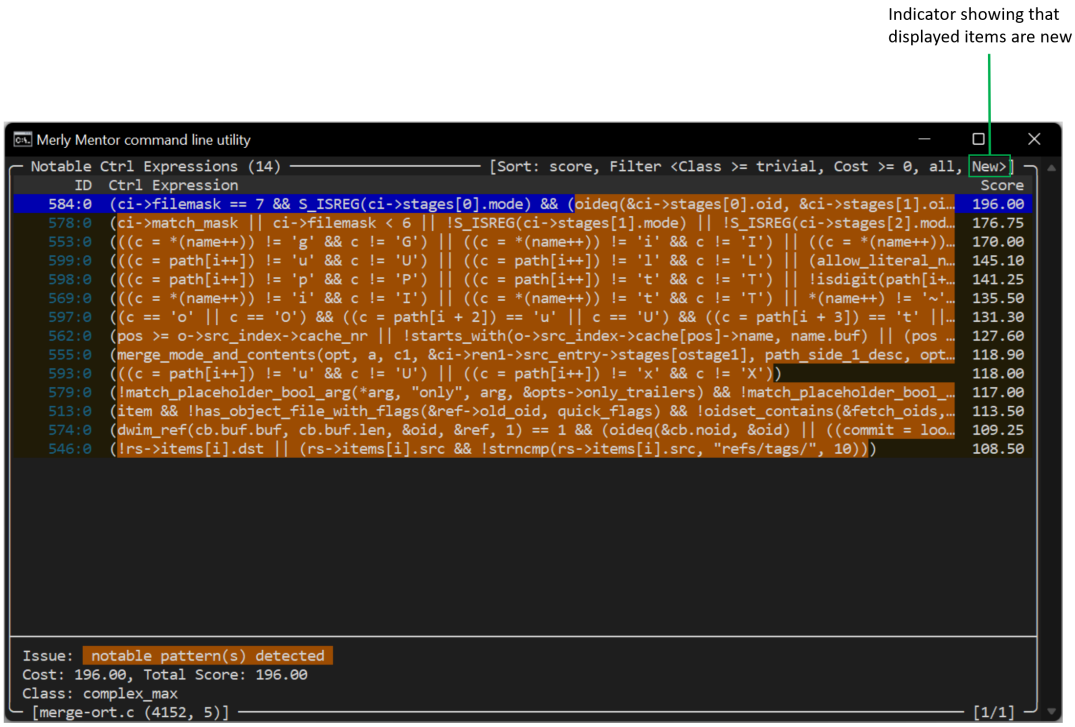


Figure 47: Merly Mentor showing a list of notables that were introduced with the latest revision of the code repository.

14.5 Comparing Inference Results in the Post-Inference Web View (Static HTML)

While you can view new anomalies and notables in Merly Mentor you can also compare inference results in the Post-Inference Web View, also known as *static HTML* output. Note that the Post-Inference Web View is an advanced feature found in the Limitless tier of Merly Mentor. In this section, we will review how to compare inference results in the Post-Inference Web View. For a full overview of the Post-Inference Web View, see Section 15.

First, be sure to have at least two previous inference folders created from the source code. See previous section (14.1) for details. Next, navigate to one of the date folders created within the main repository folder, and open the `index.html` page within it. See Figure 48 for more details.

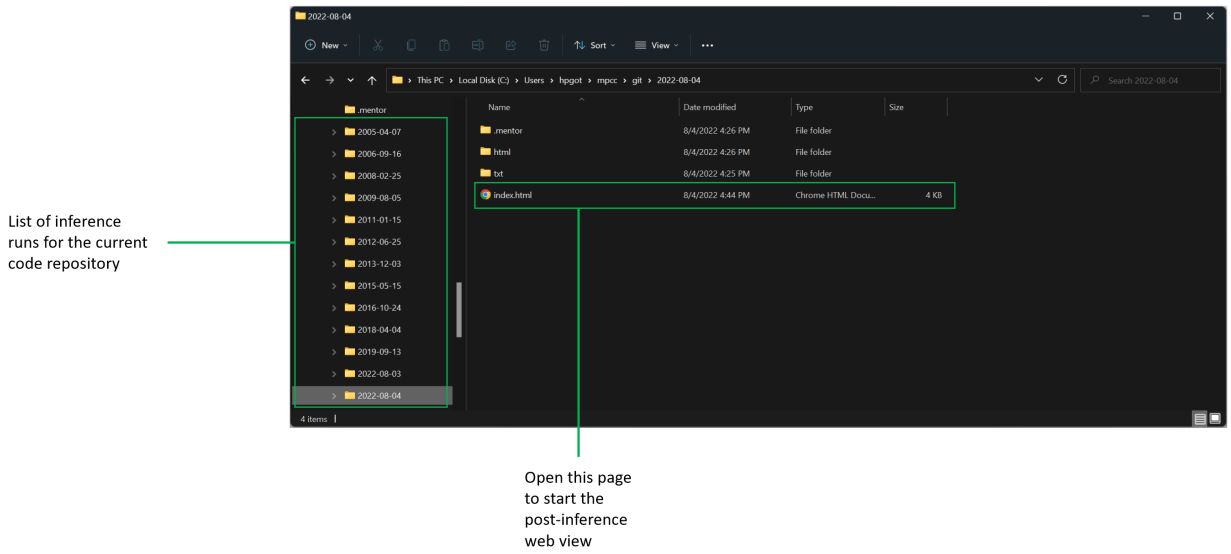


Figure 48: To start post-inference web view, navigate to the most recent date folder within the repository folder and open the index.html file.

You can navigate through the repository's instances by date by clicking on the arrow buttons in the navigation bar, or by pressing (and/or holding) the left and right arrow keys on your keyboard. If you have at least three instances of a repository in the output folder, the summary will display three repositories at a time, unless you are at the very beginning or very end of the repository list, in which case it will display two.

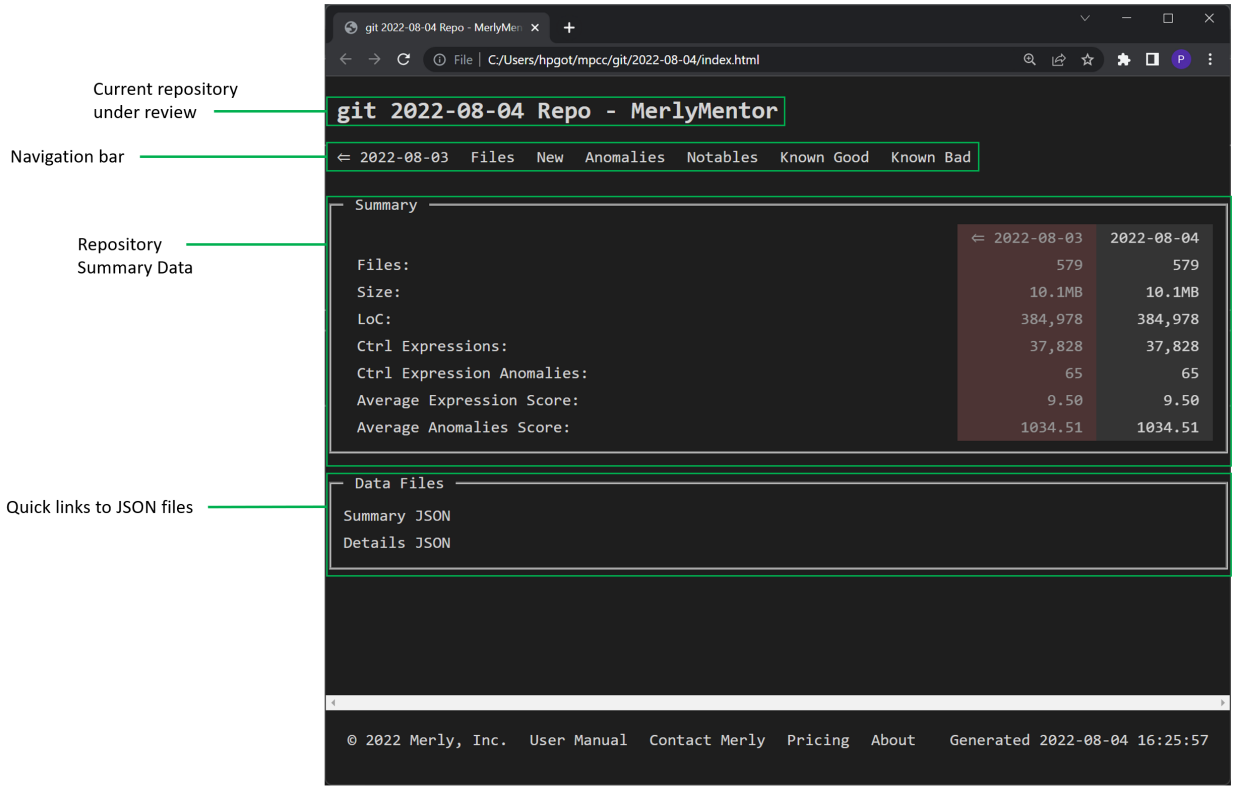


Figure 49: Merly Mentor displaying output generated from a CLI mode of Merly Mentor operation: the initial page of the post-inference web view.

Clicking on the `Files` link on the web view will take the user to a screen that shows the differences in the files between the previous revision and the current one. Pressing the left and right keys will move through the different revisions of the source code for comparison.

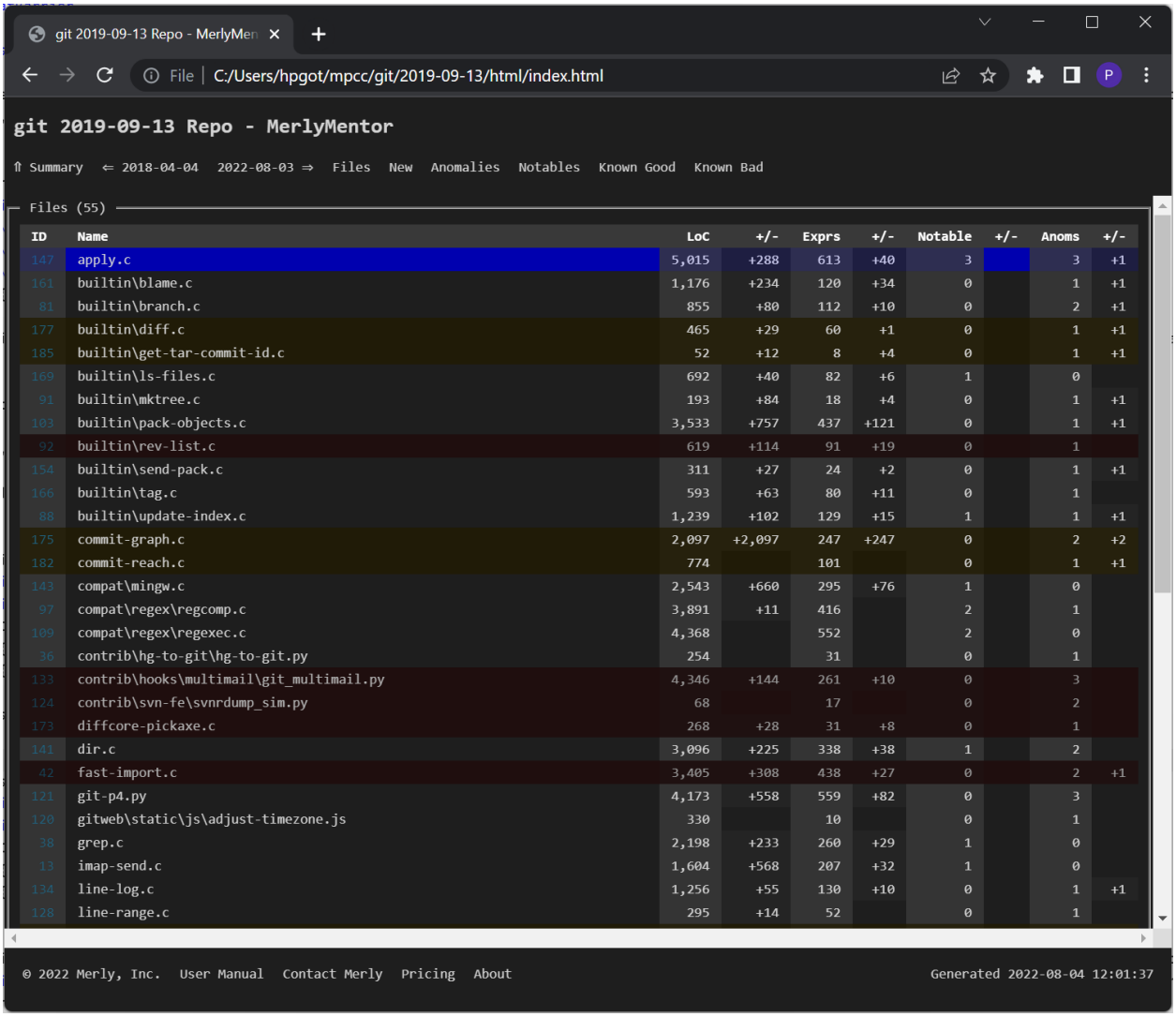


Figure 50: The file view on the post-inference Static HTML *website* displaying the differences in files between 2 inference runs. This is generated from a Mentor CLI mode of operation.

Click on the `New` link to view each of the new anomalies and notable expressions that were introduced with the current revision. Pressing the left and right keys will move through the different revisions so you can review the new anomalies and notables that were introduced with that specific revision.

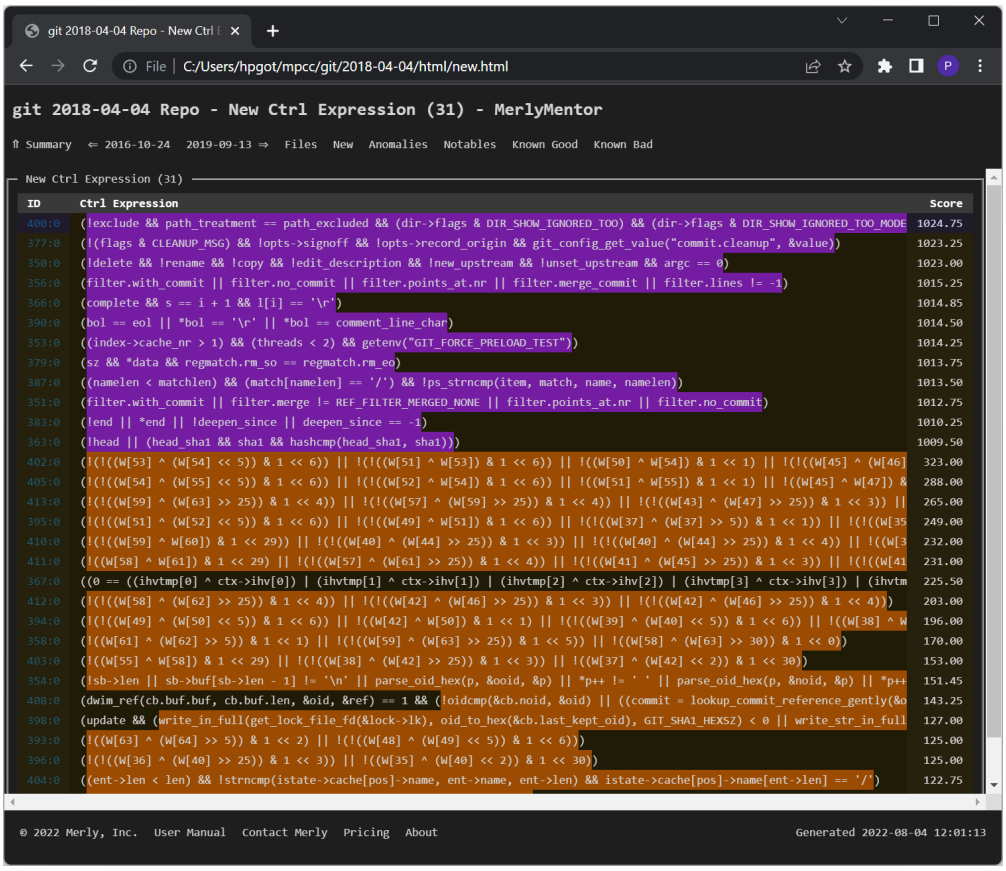


Figure 51: The New view on the post-inference website displaying the new anomalies and notables introduced with this revision.

15 Post-Inference Web View (Static HTML Output)

Post-Inference Web View (also called *static HTML*) is an advanced feature found in the Limitless subscription tier of Merly Mentor. This feature enables users to view and collaborate around the results of Mentor's inference in a portable browser-supported fashion after Merly Mentor has performed inference on a file or a repository.

For users familiar with the Mentor Service, it will be immediately familar, but this mode of output differs in several ways. For example, the Mentor Service is a system daemon, while Static HTML output is a collection of complete websites based on discreet inference dates. It does not update automatically, but will generate again on the next Mentor inference run.

In particular, Mentor's Post-Inference Web View generates inference results in *portable* and browser-based files, essentially taking the form of a fully interactive website per source repository. The intuition behind this feature is to improve Mentor's cohesiveness in team collaborations. With this feature, one team member can run inference on a repository and generate a full website of results which can be analyzed by the rest of the development team from a variety of computing mediums (e.g., desktops, laptops, tablets, cell phones, etc.) and locations (on-prem, off-prem, internet-hosted, etc.). In this section, we describe how to access and navigate through this feature.

Note: this feature is only available in Mentor Limitless subscription tiers.

The most efficient way to generate Mentor's Post-Inference Web View for a specified source code repository is to run Mentor in *non-interactive* inference mode. To do this, use the `-n` option from the command line interface (CLI), specifying the URL of a repo using the `-u` flag, the directory using the `-D`, or the repo from your `repo list` with `-r`:

Examples:

```
$ ./MerlyMentor infer -n -u [URL to repository]
```

```
$ MerlyMentor.exe infer -n -D [source code folder]
```

```
$ ./MerlyMentor infer -n -r [repo name from list]
```

Once Mentor's inference is complete, a fully-interactive website is constructed in the `/output` folder in the application working directory for the given source repository. Moreover, this inference step can be repeated an unlimited number of times against an unlimited number of repositories or instances of a repository.

This enables users to holistically analyze and characterize source repositories over any period of time. In the following subsections, we explain Mentor's inference-generated website's file and folder structure, including how to navigate the results using the various web views and pages.

To see an example of Mentor's web view for several open-source repositories, please visit the following webpage: https://merly-ai.github.io/.

15.1 Accessing the Post-Inference Web Files

In the Enterprise+ and above tiers of Mentor, each time Mentor performs inference on a repository, several output files and folders are generated in Mentor's working directory. These generated output files contain Mentor's inference results. We include a summary of the generated files and folders to help users navigate and understand them:

- **Project folder:** Mentor creates a folder within the working directory using the lowest sub-directory of the file you are running inference against.
- **Subfolder:** Mentor creates a subfolder within the project folder using a name that corresponds to the current date.

- **Sub-subfolders:** Mentor creates two subfolders within the date subfolder, one named `html` and one named `txt`
- **JSON files:** Mentor creates a hidden folder called `.mentor` where several JSON files with various metadata are stored:
  - details.json
  - expressions.json
  - files.json
  - summary.json
- **HTML file:** Mentor creates an HTML file within the date sub-folder, named index.html

If you would prefer to specify the name of the folder in which Mentor's web files are created, you can use the `-o` argument when running inference. For example, the command:

```
$ ./MerlyMentor infer -n -r git -o git-renamed-output-repo-folder
```

...would create a folder within `/output` named `git-renamed-output-repo-folder`. To access the Post-Inference Web View static HTML files, navigate to and open the `index.html` file within the date subfolder that you're interested in.

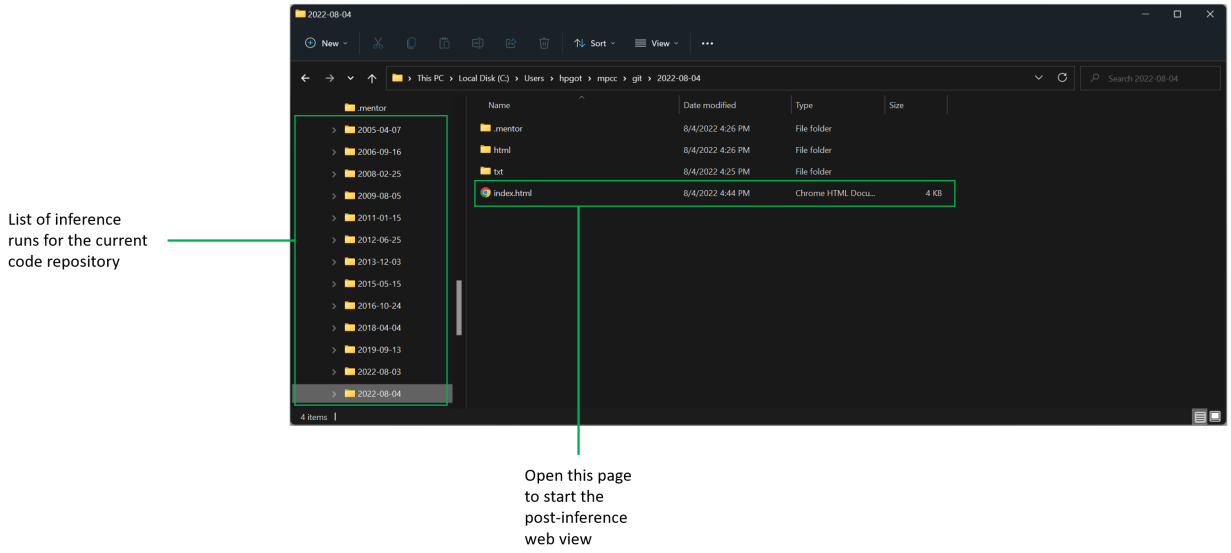


Figure 52: In Mentor's working directory, navigate to and open the index.html file to launch Mentor's generated website for the specified repository.

15.2 Index page

The index page will display a navigation bar to access different parts of your code repository, a summary of key aspects of your repository, and quick links to access the JSON files that Mentor creates after running inference.

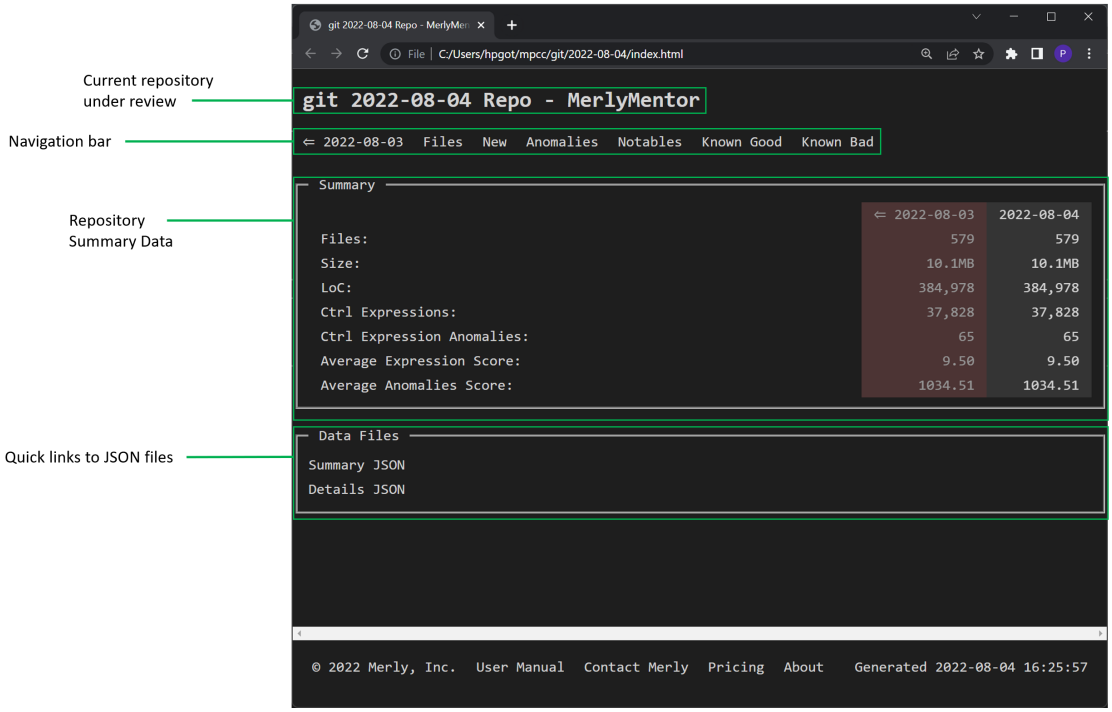


Figure 53: The initial view when opening the index.html page created by Merly Mentor.

15.3 Files page

Clicking on the Files link will take the user to a list of all of the files found in the repository. This view replicates the Files View found earlier in this manual's interactive mode.

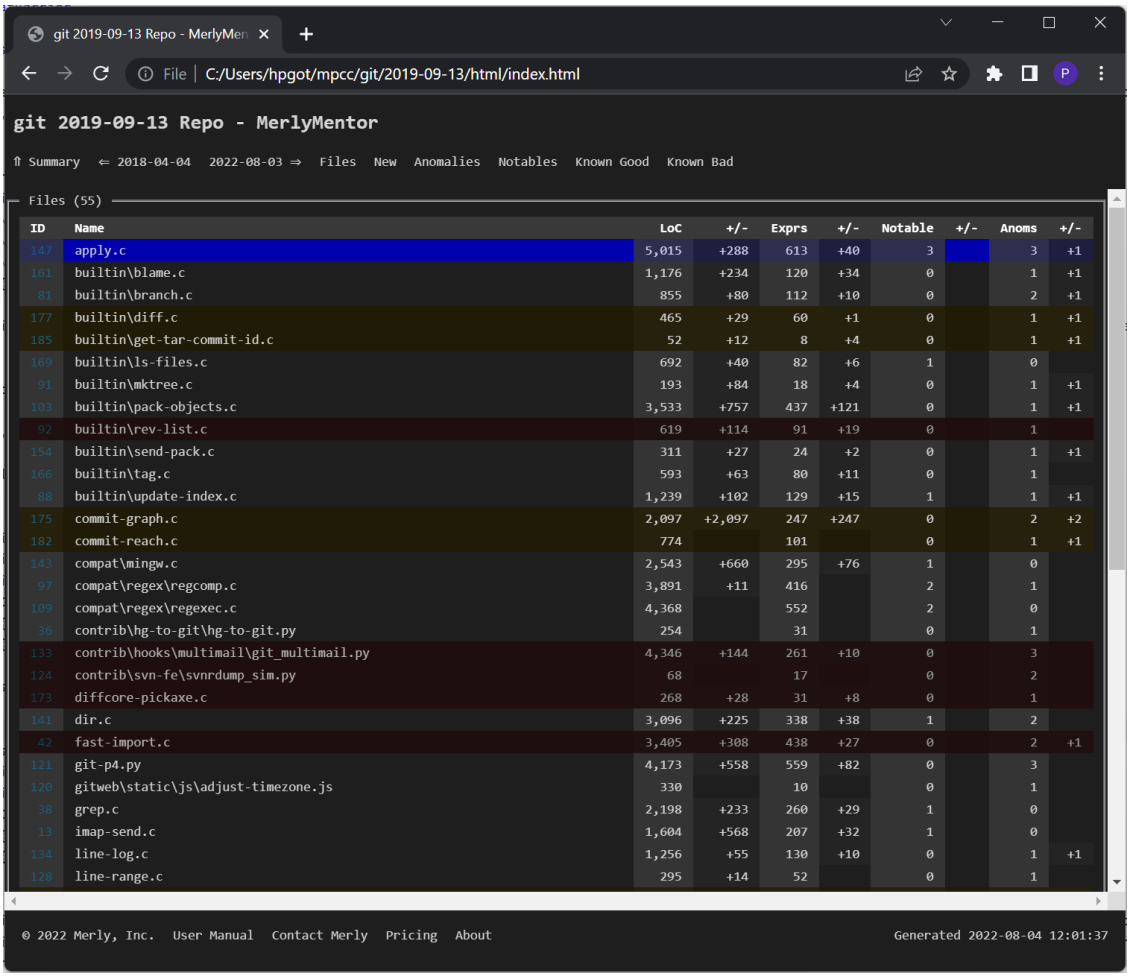


Figure 54: The files page shows a list of the files found in the repository, with each file's summary data.

On the right side of the Files listing view, Mentor displays the lines of code per source file, the total number of expressions, the notables found in the file, as well as the anomalies found in the file. The user can click or tap on any file to see its contents for a deeper analysis.

The default behavior of Mentor generates static HTML with only files with anomalous expressions listed. Two flags which can control this behavior when running an inference or repo update command:

- `--all-files`: shows all files in the inferenced repository
- `--issues-only`: shows only the files with anomalies

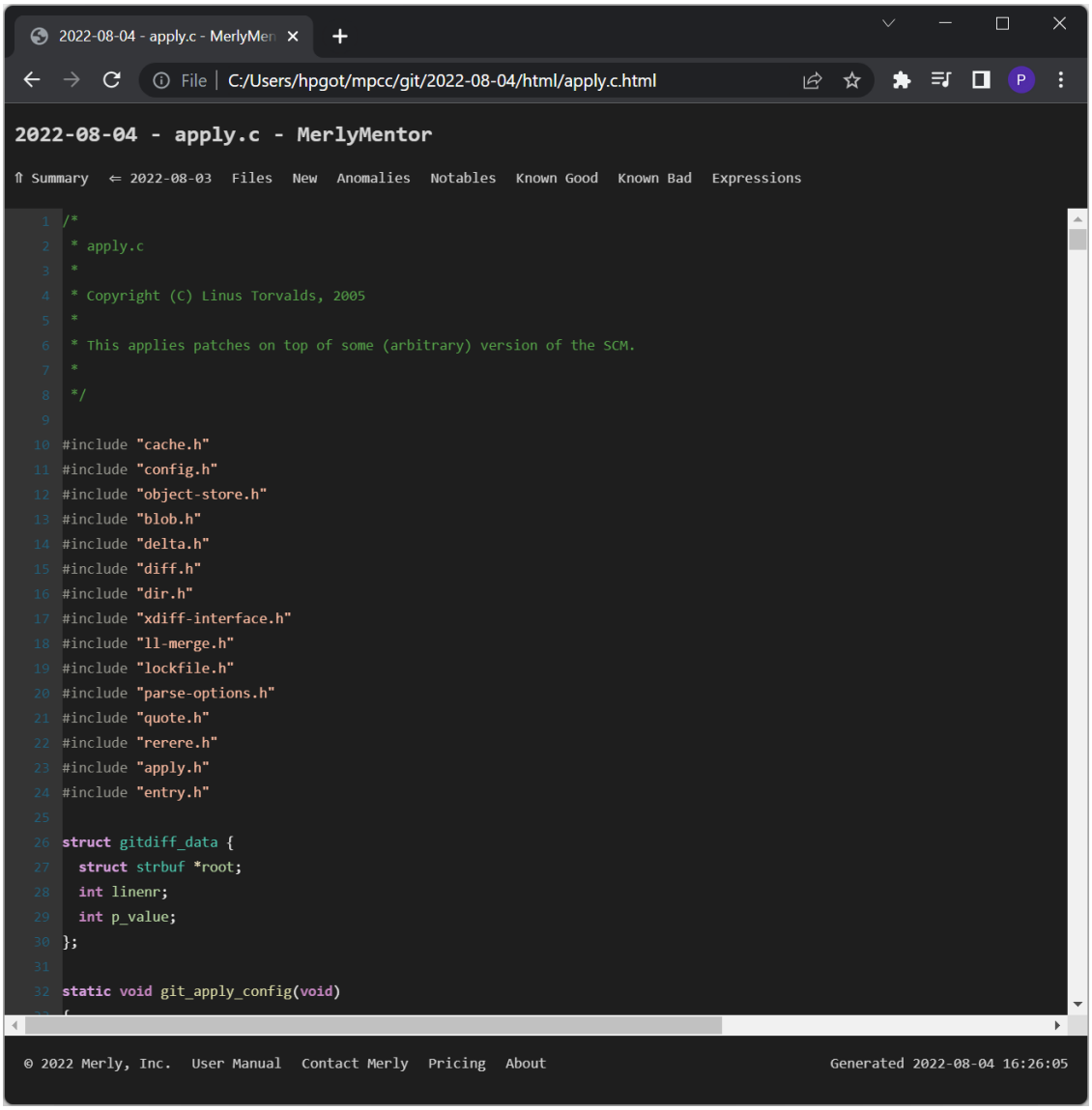


Figure 55: Clicking on a file from the files detail page takes the user to the code contained in that file.

15.4 Anomalies page

Clicking on the Anomalies link will take the user to a list of all of the anomalies found in the repository.

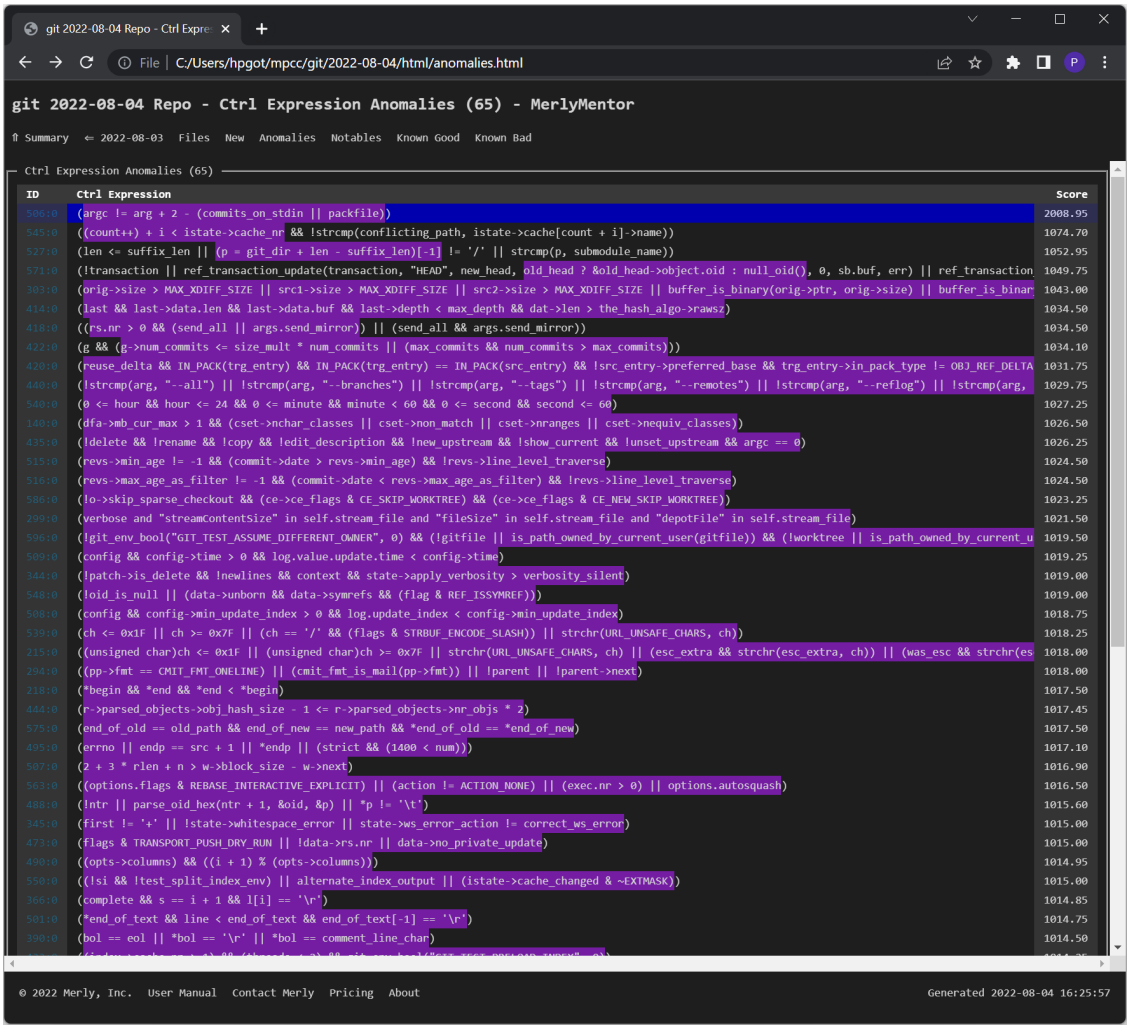
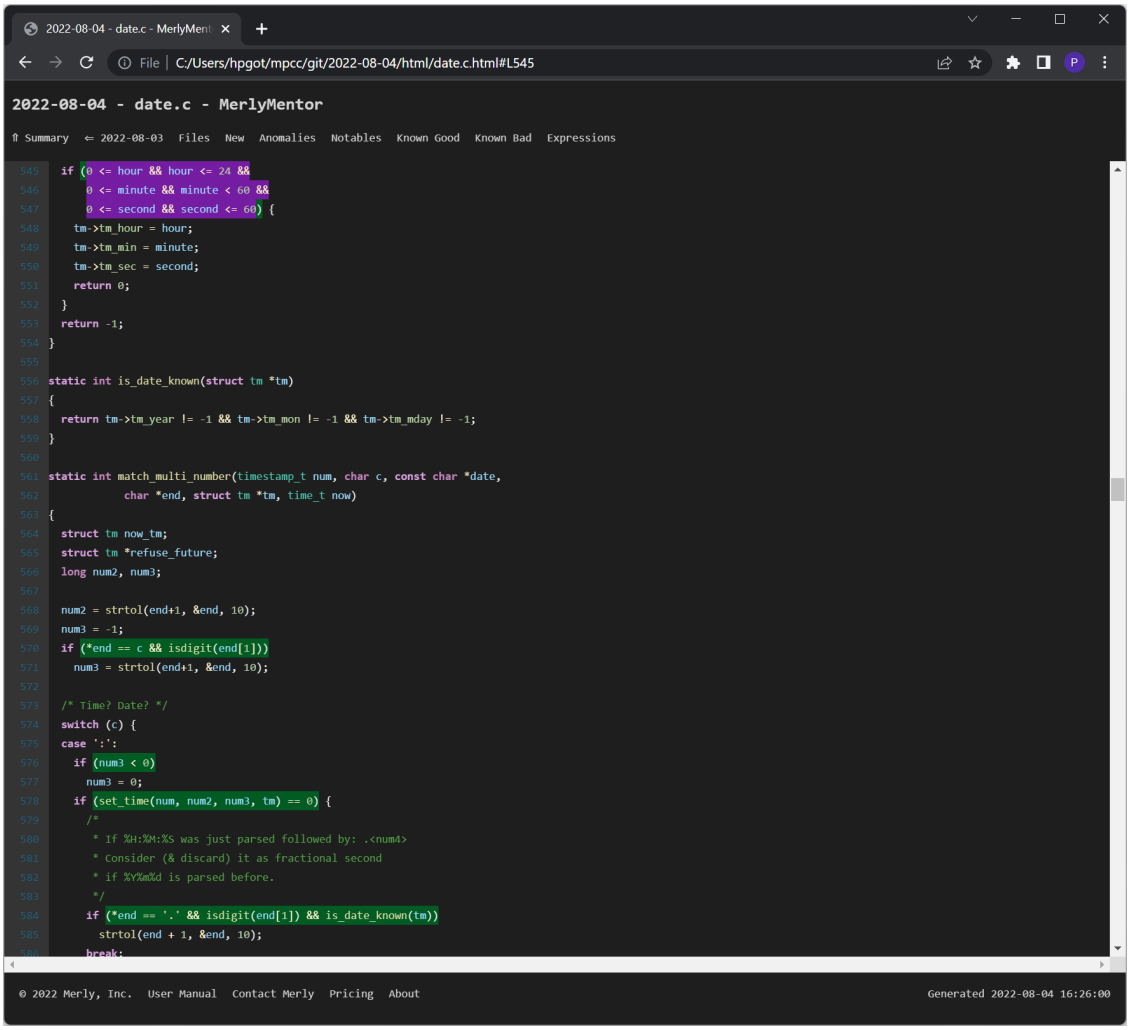


Figure 56: The anomalies page shows a list of the anomalies found in the repository, along with each item's assigned score.

The user can review the code surrounding the anomaly and try to determine context by clicking or tapping on the desired anomaly.




```
  if (0 <= hour && hour <= 24 &&
      0 <= minute && minute < 60 &&
      0 <= second && second <= 60) {
    tm->tm_hour = hour;
    tm->tm_min = minute;
    tm->tm_sec = second;
    return 0;
  }
  return -1;
}

static int is_date_known(struct tm *tm)
{
  return tm->tm_year != -1 && tm->tm_mon != -1 && tm->tm_mday != -1;
}

static int match_multi_number(timestamp_t num, char c, const char *date,
           char *end, struct tm *tm, time_t now)
{
  struct tm now_tm;
  struct tm *refuse_future;
  long num2, num3;

  num2 = strtol(end+1, &end, 10);
  num3 = -1;
  if (*end == c && isdigit(end[1]))
    num3 = strtol(end+1, &end, 10);

  /* Time? Date? */
  switch (c) {
  case ':':
    if (num3 < 0)
      num3 = 0;
    if (set_time(num, num2, num3, tm) == 0) {
      /*
       * If %H:%M:%S was just parsed followed by: .<num4>
       * Consider (& discard) it as fractional second
       * if %Y%m%d is parsed before.
       */
      if (*end == '.' && isdigit(end[1]) && is_date_known(tm))
        strtol(end + 1, &end, 10);
      break;
```

Figure 57: Clicking on an anomaly from the anomalies detail page takes the user to the code containing the anomaly.

The user can review Merly Mentor's recommendations by clicking on the anomaly within the code.

For more information about Merly Mentor's recommendations, please see Section 11.3.

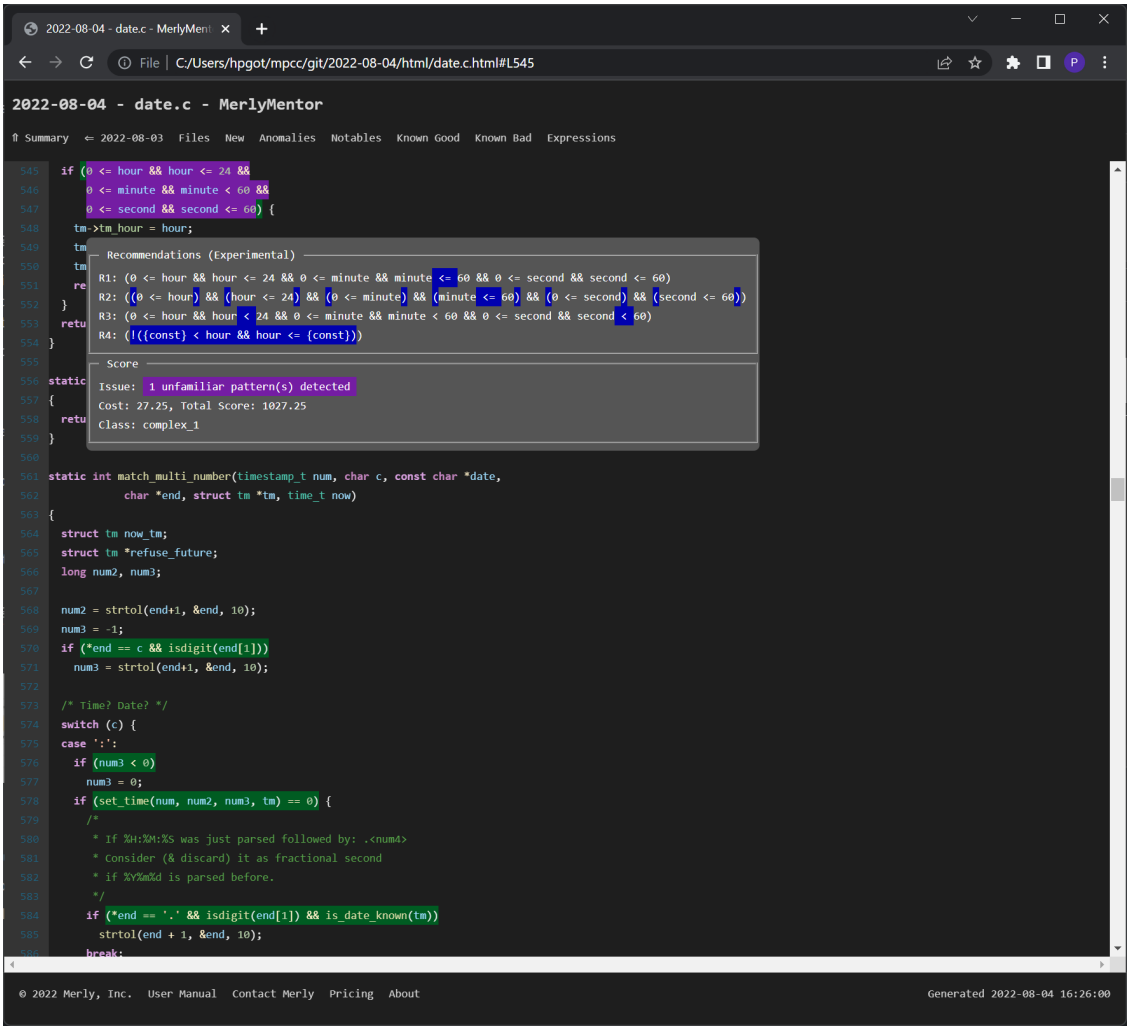


Figure 58: Clicking on an anomaly from within the code displays Mentor's code recommendations.

15.5 Notables page

Clicking on the Notables link will take the user to a list of all of the notables found in the repository. Notables are parts of code that Mentor identifies as potentially more complex and harder to understand. For more on Notables, please see Section 12.3

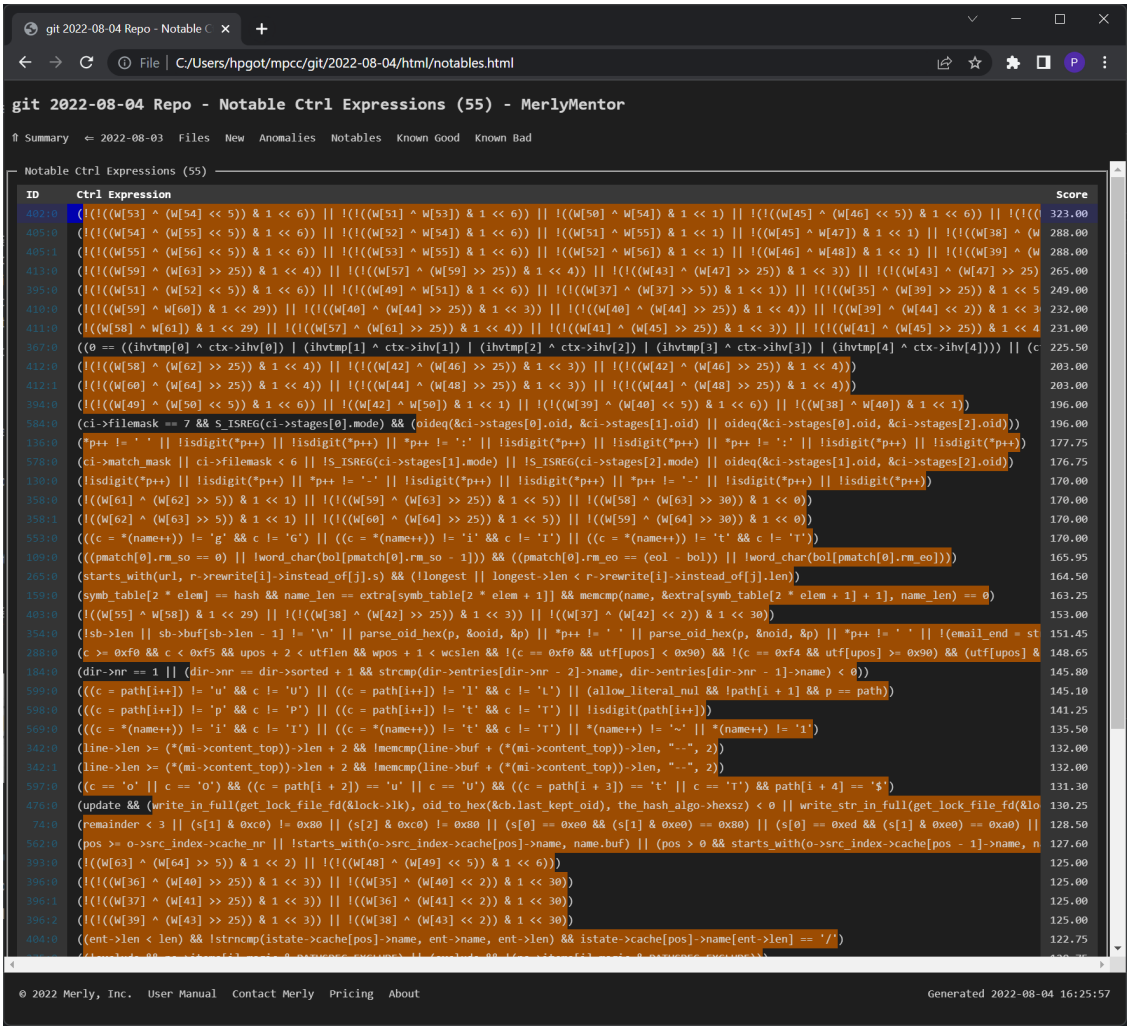


Figure 59: The notables page shows a list of the notable expressions found in the repository, along with each item's assigned score.

The user can review the code surrounding the notable and try to determine context by clicking on the desired notable.

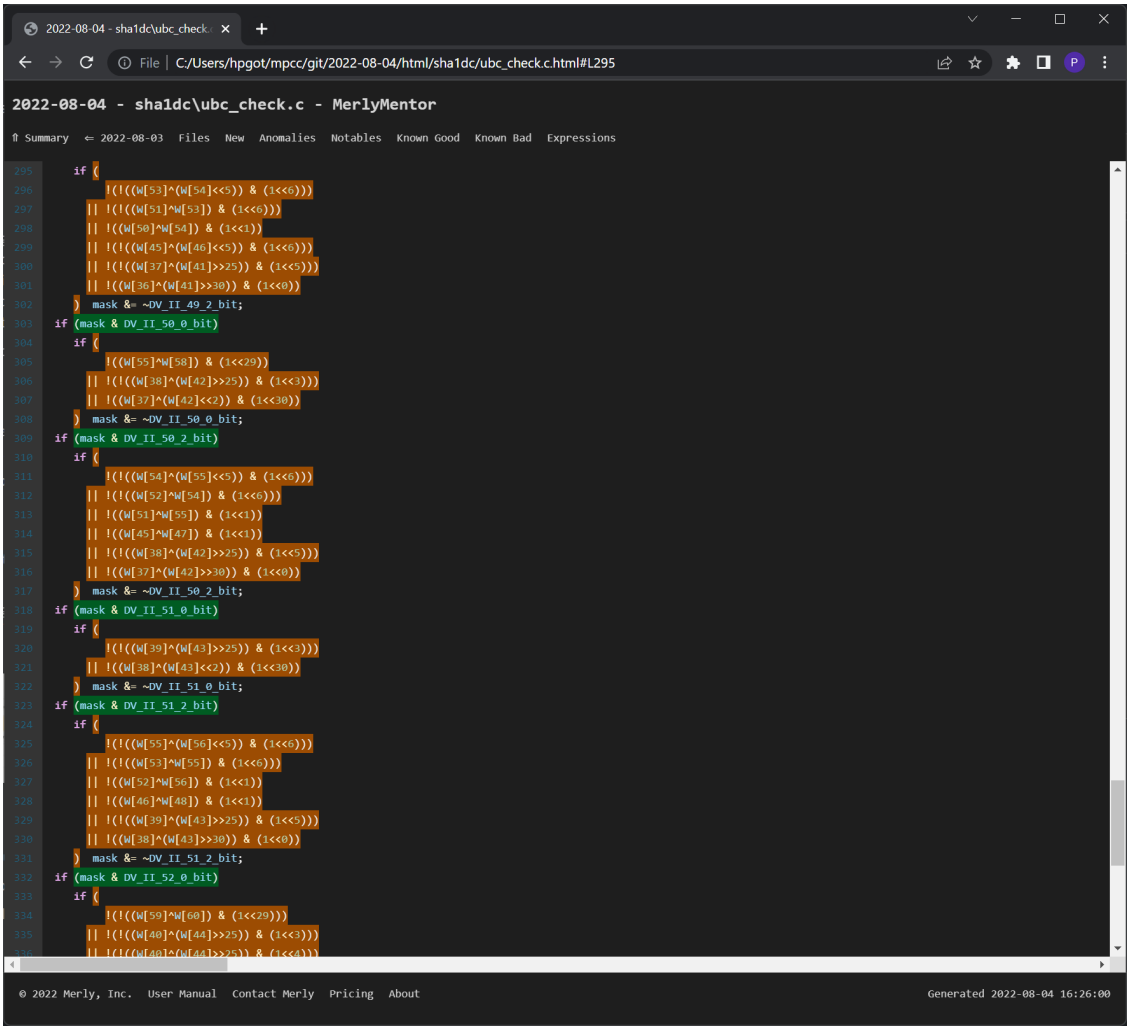


Figure 60: Clicking on a notable from the notables detail page takes the user to the code containing the notable.

15.6 Known Good page

If you have identified Known Good items in your code, then clicking on the Known Good link will take you to a list of all of the expressions marked as Known Good.

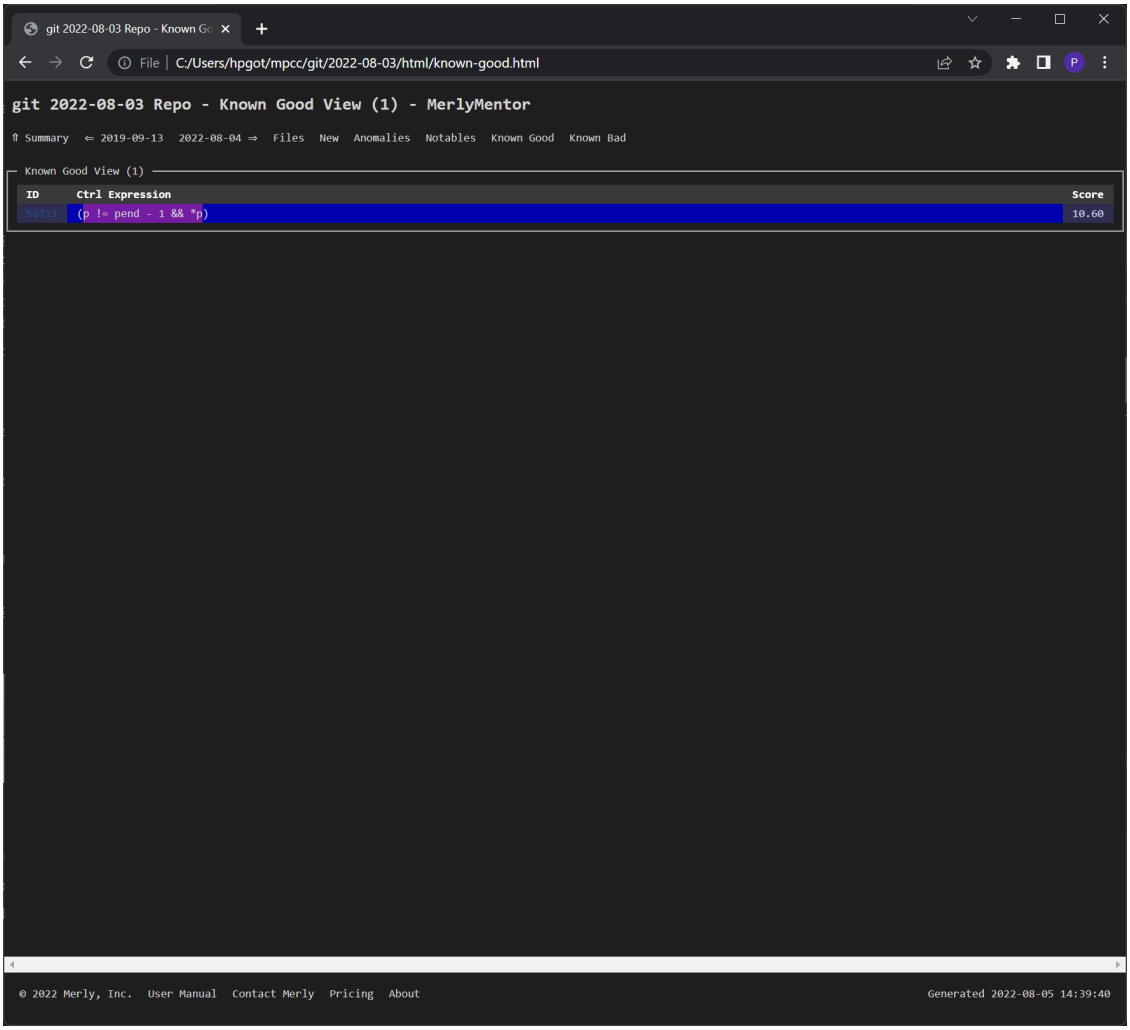


Figure 61: The Known Good page shows a list of the expressions that have been marked as Known Good, along with each item's assigned score.

The user can review the code surrounding the known good expression and try to determine context by clicking or tapping on the desired known good expression.

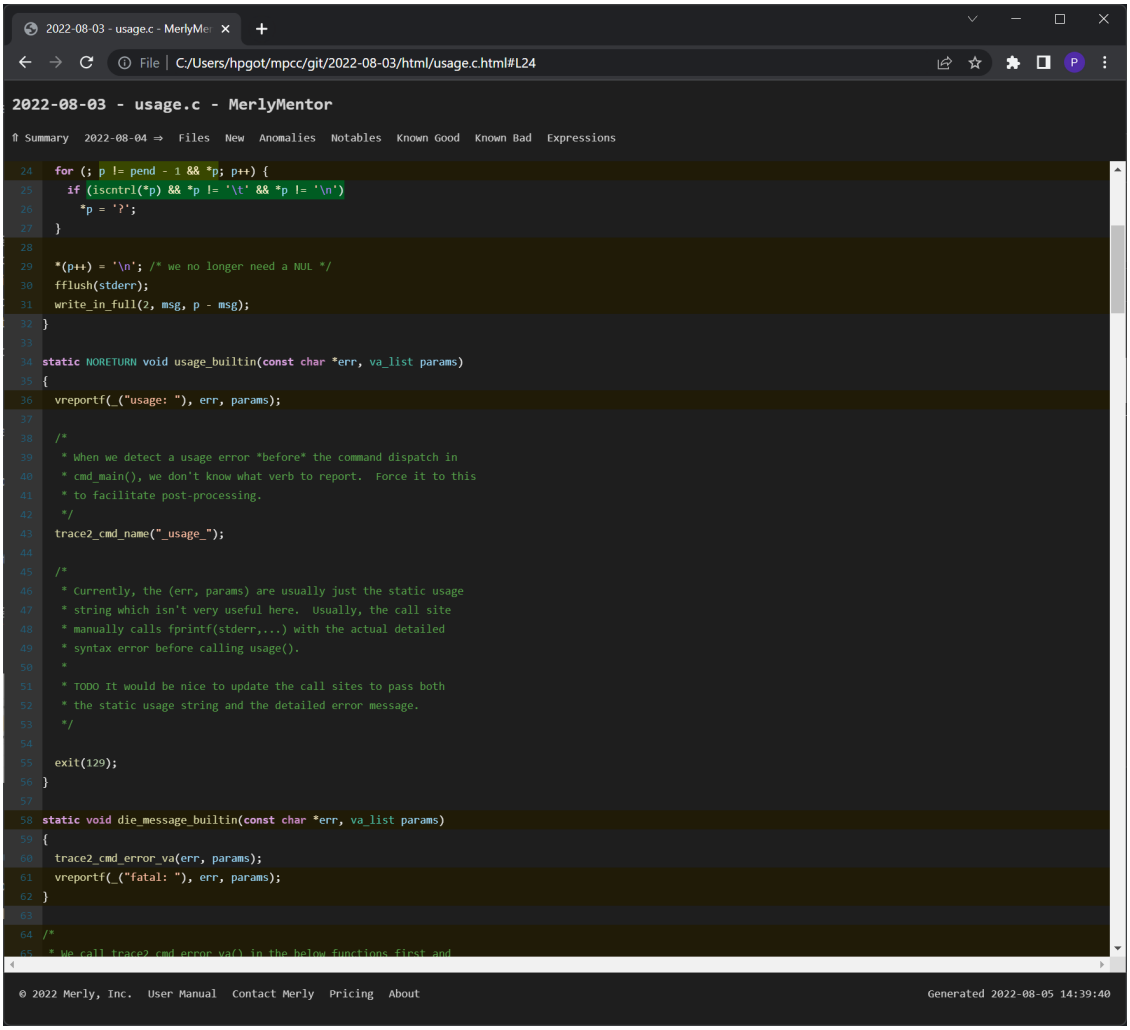


Figure 62: Clicking on a known good expression from the known good page takes the user to the code containing the known good expression.

15.7 Known Bad page

If you have identified Known Bad items in your code, then clicking on the Known Bad link will take you to a list of all of the expressions marked as Known Bad.

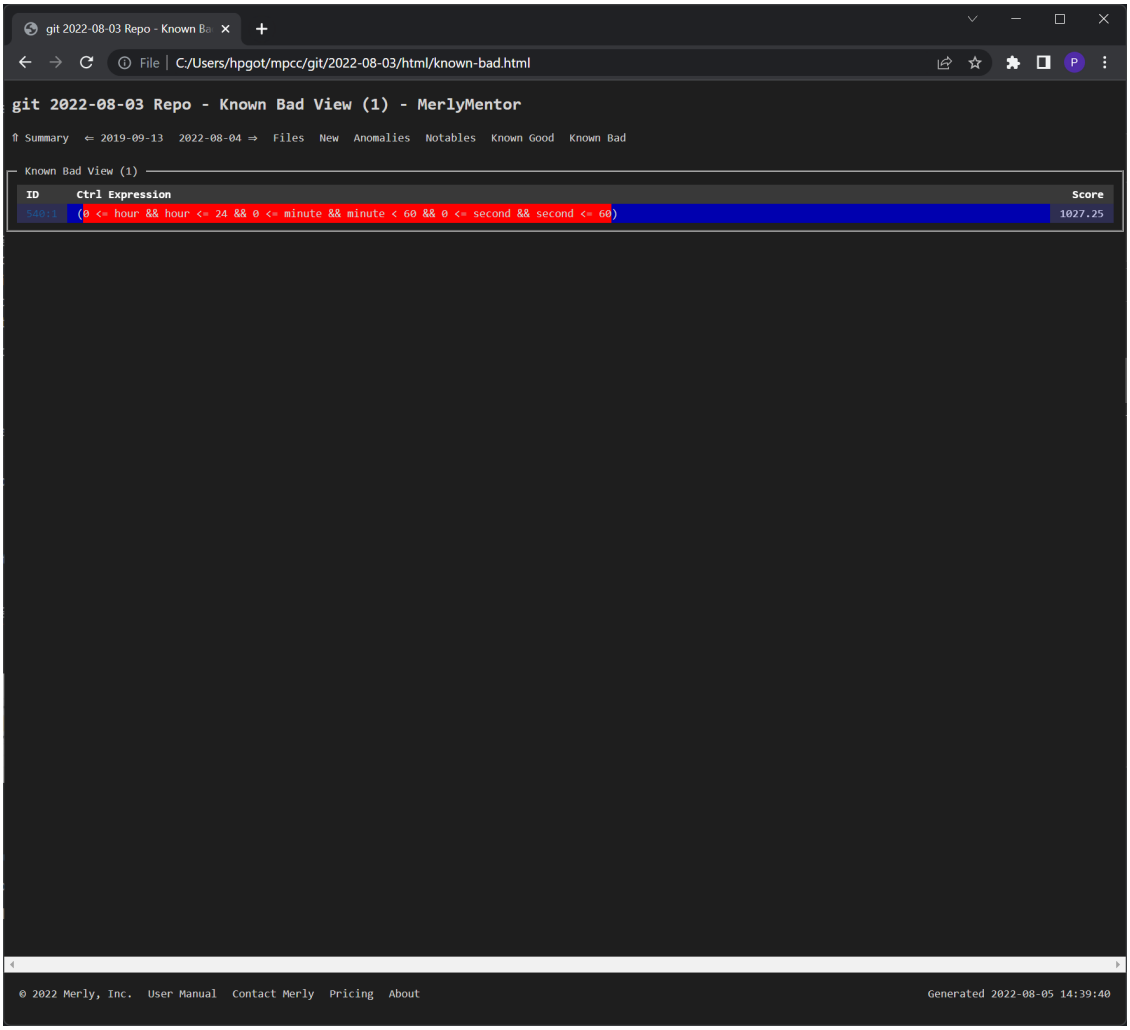


Figure 63: The Known Bad page shows a list of the expressions that have been marked as Known Bad, along with each item's assigned score.

The user can review the code surrounding the known bad expression and try to determine context by clicking on the desired known bad expression.

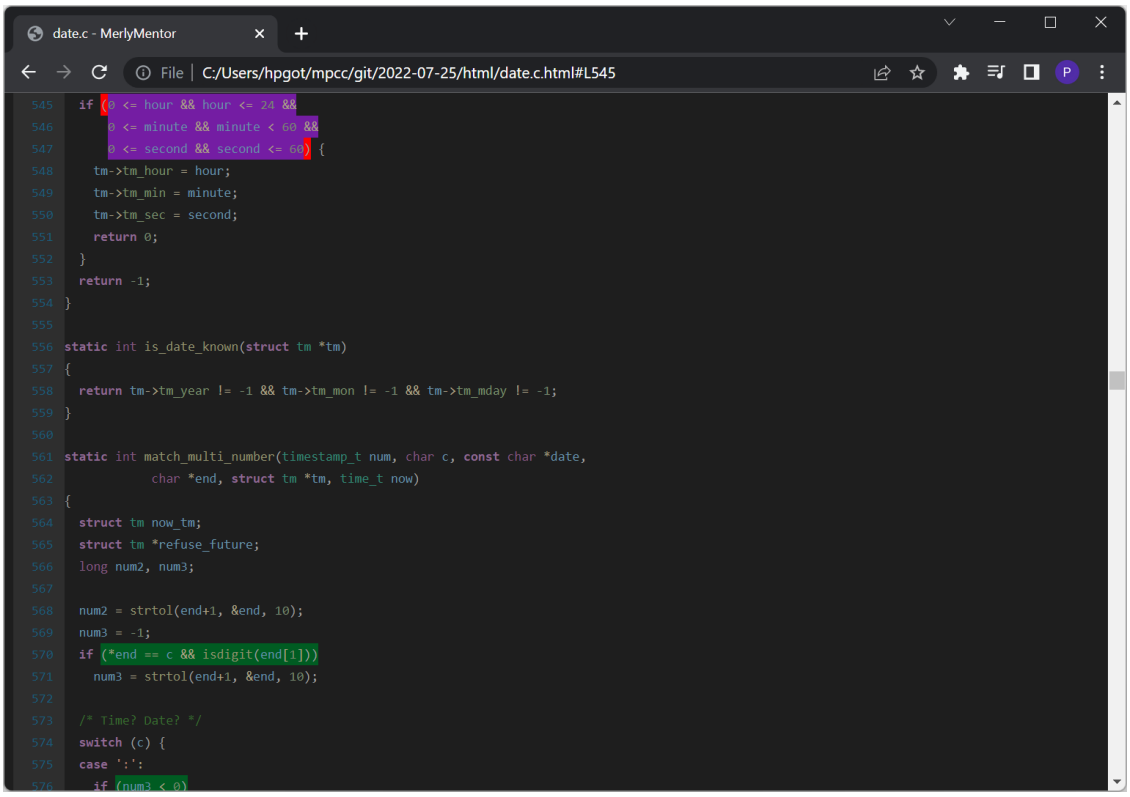


Figure 64: Clicking on a known bad expression from the known bad page takes the user to the code containing the known bad expression.

The user can review Merly Mentor's recommendations for the known bad expression by clicking on that expression within the code. For more information about Merly Mentor's recommendations, please see Section 11.3.

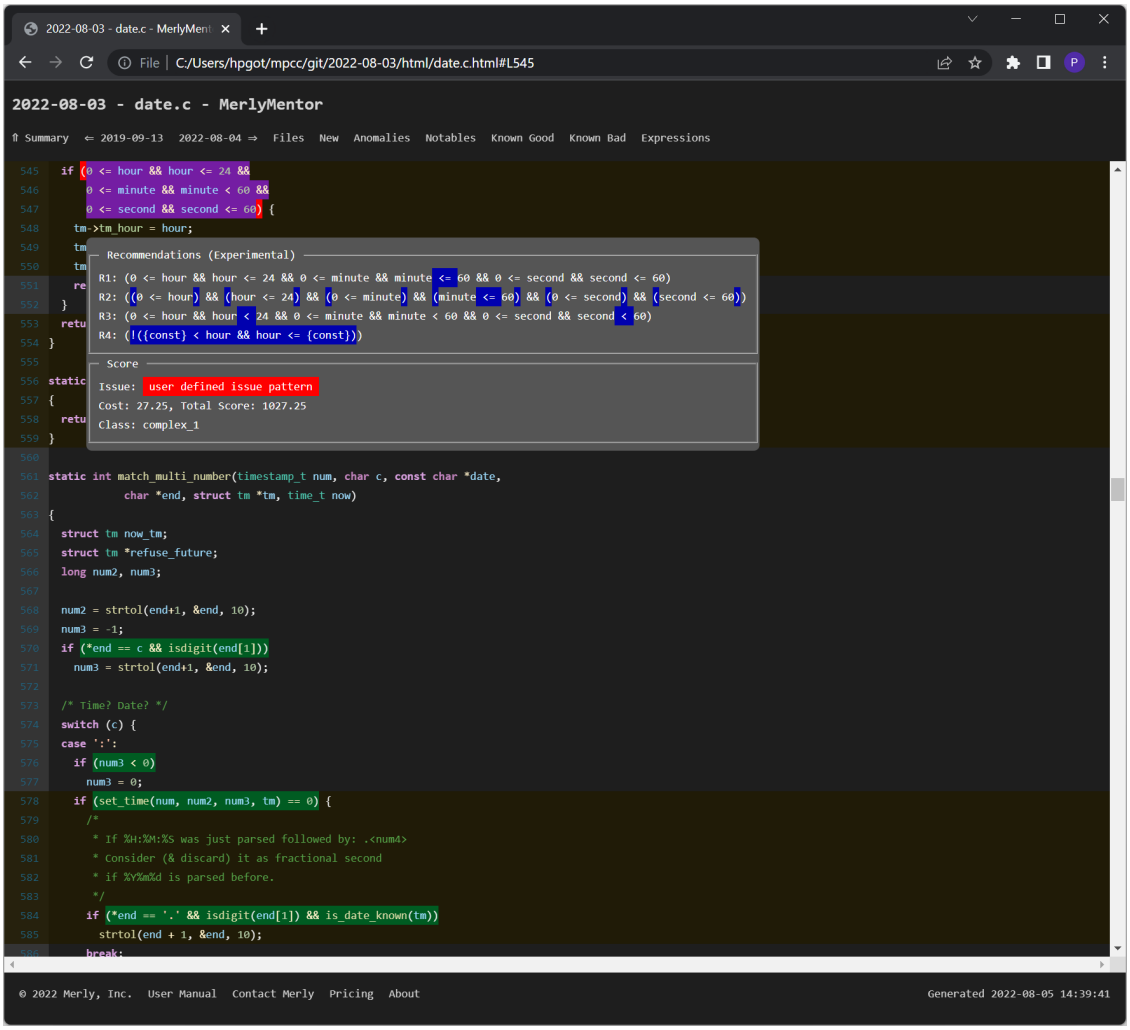


Figure 65: Clicking on a known bad expression from within the code displays Mentor's code recommendations for the expression.

15.8 Copy and Paste

At times it can be helpful to share information from Mentor's Post-Inference Web View with others who do not have access to the web view or the web pages that Mentor generates. In order to facilitate this level of information sharing, a *copy and paste* feature is available.

In order to access this feature, run inference on a repository using the most recent version of Mentor. (Note: this feature will not work on repositories which have not had inference run against them with a version of Merly Mentor before 0.0.22. If you wish to use this feature for repositories on which inference has already been run, it is highly recommended you delete the entire folder structure and regenerate the historical inference folders.)

After running inference, select the page that has the files you wish to copy. (The `Files`, `New`, `Anomalies`, `Notables`, `Known Good`, and `Known Bad` pages all have this feature.) Once on the page, press the `p` key on your keyboard. This will start copy/paste mode, and will be indicated by an orange `Copy/(P)aste Mode` indicator in the navigation bar.

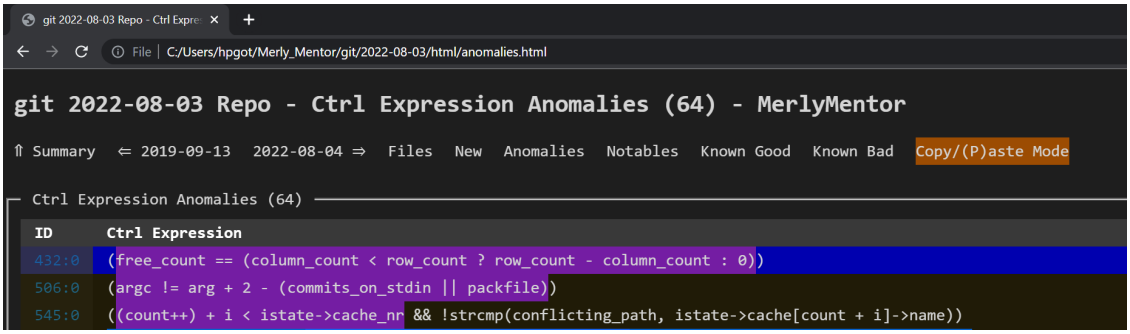


Figure 66: Pressing `p` in the Post-Inference Web View brings up Copy/(P)aste Mode.

Once in copy/paste mode, you can copy and paste text within the page via standard methods. When complete, press `p` again to re-enable drill-down navigation with the list.

16 Mentor Plans and Feature Set

Merly Mentor can be downloaded, installed, and demoed even before a subscription is purchased. All you have to do is run the Merly Installer to set up the application. See the installation section for instructions.

The feature set of Mentor is divided among a series of tiers, or plans, to best fit what your team is looking for.

For more details on the features available with each subscription plan, visit Merly.ai's pricing page.

16.1 Basic

A limited release version allows for exploration of Merly Mentor's basic feature set:

- Single file inference
- Limited code view

16.2 Mentor Professional

All Basic features, and...

- File-level inference
- Syntactic analysis only
- Code view only
- Flag known good patterns

16.3 Mentor Professional+

All Professional plan features, and...

- File + directory-level inference
- Inference on multiple languages in single execution
- Syntactic and semantics analysis
- Includes all views (code, file anomalies, expressions)
- Flag known good and bad patterns
- Repository code quality summary

16.4 Mentor Enterprise

All Professional+ plan features, and...

- Advanced version of Merly Mentor designed with Git integration
- Includes on-premises features, empowering team collaboration on code understanding and repair
- Integration with Git
- Technical support: email & phone calls

16.5 Mentor Enterprise+

All Enterprise plan features, and...

- Even more advanced version, as well as technical support via virtual meetings

16.6 Mentor Limitless

The most advanced and holistic version: All Enterprise+ plan features, and...

- Quality trending and reporting, driving code quality and metrics
- A graphical trending analysis for entire repository lifetimes
- Advanced false positive filtering for anomalies
- Leaderboards for multiple repositories being analyzed
- MerlyMentor as a service
- Dynamic rendering of leaderboards and graphs

Table 1: Feature Set Comparison

Feature	Basic	Professional	Professional+	Enterprise	Enterprise+	Limitless
File-level inference	✔	✔	✔	✔	✔	✔
syntactic analysis		✔	✔	✔	✔	✔
Code view		✔	✔	✔	✔	✔
Flag known good patterns		✔	✔	✔	✔	✔
File and directory-level inference			✔	✔	✔	✔
Semantics analysis			✔	✔	✔	✔
Inference on multiple languages in single execution			✔	✔	✔	✔
Includes all views (code, file anomalies, expressions)			✔	✔	✔	✔
Flag known good/bad patterns			✔	✔	✔	✔
Repository code quality summary			✔	✔	✔	✔
Integration with Git				✔	✔	✔
Technical support: email phone				✔	✔	✔
Technical support: virtual meetings					✔	✔
Quality trending and reporting, driving code quality and metrics						✔
A graphical trending analysis for entire repository lifetimes						✔
Advanced false positive filtering for anomalies						✔
Leaderboards for multiple repositories being analyzed						✔
Merly Mentor service						✔
Dynamic rendering of leaderboards and graphs						✔
Trial version available			✔	✔	✔	✔

17 Release Notes

Merly Mentor v. 0.0.22 Changelog:

- Repo modification actions (edit/delete/rerun/infer/clone/fetch) need `lead` or higher access level
- User modifications in the Mentor Service require `admin` or `owner` access level
- Fixed issue with adding a repo in the Mentor Service resulting in 404 error (switched from ✚ to # as the symbol)
- Added check to ensure username/email is unique on `new user` and `edit user`
- Added option to select Merly SMTP or custom SMTP server with password being stored scrambled
- Added option to switch between public and private service with `service_url`. Public service will generate emails for add/edit user, and private service will just add/edit without sending an email (as there is no public URL that can be used to access the service, so an email link cannot be used to access the Mentor Service)
- New: User list table display; now with role, ID, and status (email sent, inactive, active, etc.)
- Quick actions now has now an edit-then-save mode, which allows all fields to be changed and then saved; fix for erroneous behavior when in edit mode the dialog would disappear when the mouse is moved outside the dialog. Now, the dialog will close once "Save" is clicked and the mouse leaves the dialog
- Fixed a bug which allowed a `developer` to remove an repo issue by selecting an empty action or empty assigned user
- Fixed a crash bug found with the `repo issues` page

Known Issues:

- Merly Mentor has issues with git commands for git repos after git is uninstalled
- No status indicators on user list in Mentor Service

The Merly development team is constantly adding new features and enhancements to the Merly Mentor ecosystem. To report any issues, please email support@merly.ai.