Jasmine Granot 308120559 Yasminso@mta.ac.il

Vered Harel 205680523 Veredha@mta.ac.il

**Available commands:**

1. Changing username:
   1. To change the username, press on the button with the pencil sign.
   2. Default user is "Administrator"
2. Load a repository from XML:
   1. To load a repository from XML, press the "load repository" button and select the XML file.
   2. After loading, the system repository will be changed to the new repository
   3. In case repository already exists – a proper message is shown.
   4. In case repository is invalid – last repository will be restored.
   5. When loading remote tracking repository, one must make sure the name of the remote repository's reference is an existing repository.
3. Create new repository:
   1. To create new repository, press the "create new repository" button.
   2. If the repository folder already exists – select it from the file chooser.
   3. Else, create through the file chooser the folder and select it.
4. Change repository:
   1. To replace a repository, press the "change repository" button and select the wanted file from the file chooser.
   2. If the chosen folder is not a valid repository – a proper message is shown.
   3. Remote tracking data will not available.
5. Show commits content:
   1. To show a specific commit data, press the "commit data" button and insert the commit sha1.
   2. In the top center of the window the commit's data will be shown.
   3. To see the content of a file as it was during the commit – press twice on the row of the file.
6. Show status:
   1. To show the working copy status, press the "show status" button.
   2. Shows in the top center of the window all changes in the WC vs. last commit, divided to New Files, Changed Files and Deleted Files.
   3. All 3 option will not show any files if the WC hasn't changed.
7. Commit:
   1. To commit, press the "commit" button and insert the commit message.
8. Show all repository's branches:
   1. In the top center of the window, below the magit header there is a combo-box which contains all of the branches in the repository (if the repository contains a folder with RB, the branches in that folder will not be shown! Only branches under the direct folder "Branches").
   2. Head is marked by "(Head)".
9. Create new branch:
   1. To create new branch, press the "new" button.

Insert the branch's name and the wanted sha1.

If the sha1 cell is left empty, the branch will point to the head sha1.

Else, the program will check if the sha1 if pointed by a "remote branch".

If so, choose between creating the branch as "remote tracking branch" (then the branch name will automatically be changed to the remote branch's name) or to abort.

* 1. Branch name mustn't contain spaces.
  2. Branch mustn't be already in the system.

1. Delete a branch:
   1. To delete a branch, select a branch from the combo-box and press the button with the trash sign.
   2. Head branch cannot be deleted.
2. Checkout branch:
   1. To checkout, select a branch from the combo-box and press the "checkout" button.
   2. Head will be pointing to the new branch.
   3. WC will be restored to the last committed state.
   4. In case of non-committed changes before checkout – a proper message will be shown. select if to ignore the changes, which will delete all changes, or abort.
3. Reset head branch:
   1. To reset head branch, press the "reset" button and insert a sha1.
   2. A valid commit must be provided.
   3. In case of non-committed changes before checkout – a proper message will be shown. select if to ignore the changes, which will delete all changes, or abort.
   4. WC will be restored to the commit's given state.
4. Commit graph:
   1. In the bottom center of the window the commit tree will be shown.
   2. Head branch name will be shown at the end of the row of the commit that the branch is pointing to.
   3. The tree will be updated through each change in the repository – load/change repository, change head branch, merge etc.
   4. Commit node that was created through merge will have 2 edges in the tree, one for each "parent" commit.
5. Merge:
   1. To merge, press the "Merge" button. Insert the name of the branch-to-merge-with.
   2. If it is a FF merge – a proper message will be shown and if needed the program will commit the changes.
   3. At the end of the merge, a list of the files and their status in each branch will be shown.
   4. If there is a conflict during the merging, a conflict resolve window will be shown.

Select from the 3 text options in the top of the window the wanted data and copy it to the bottom of the window. If you wish to delete the file, save it with no data.

* 1. At the end of the merge, to commit the data, insert a commit message and press OK.

1. Clone:
   1. To clone a repository, press the "clone" button, choose from the file chooser the repository to clone from and insert the full path of a folder to clone to and its name.
   2. The new repository will contain the same data as the remote repository.

The branches of the remote repository will be cloned to a folder inside the "Branches" folder with the name of the remote repository.

The head branch will be automatically created in the "Branches" folder and will be classified as a "remote tracking branch".

1. Fetch:
   1. To fetch, press the "fetch" button.
   2. This command will update the branches and objects from the remote repository.
2. Pull:
   1. To pull, press the "pull" button.
   2. This command will update the objects and System file's list in local repository from the head's remote tracked branch.
   3. Cannot pull when there are open changes.
3. Push:
   1. To push, press the "push" button.
   2. Cannot push when there are open changes.
   3. This command will update the objects and System file's list in remote repository from the local repository's head branch.
4. Layout buttons (Bonus):
   1. To change layouts press the "v" button in the bottom right of the window and choose between the 3 different layouts.

**Main project classes:**

The project is divided to two packages – engine and UI.

In the UI package one can find the project's main function and the UI engine itself.

The engine package is divided to 6:

1. Parser – contains JAXB magit objects for parsing XML files.
2. XML handler – contains XML utilities and validator.
3. Utils –
   1. Magit Utiles – used as a general application utiles class for common functions.
   2. Helping objects – Working Copy Changes, Magit String Result, Result list.
   3. Merge utils –
      1. File state – interface.
      2. Find current file state – used to find the correct merge case for each file.
      3. Merge result – initialize a file with all the data that was collected on it during the search of its merge case.
4. Git objects –
   1. Blob – represents a file in the system, inherits from Git Object Base.
   2. Folder - represents a folder in the system, inherits from Git Object Base.
   3. Commit - represents a commit in the system, inherits from Git Object Base.
   4. File details – represents static information of each file.
   5. Git Object Base – represents the common git objects.
   6. Branch – represents a branch in the system.
   7. Sha1 object – used for calculating sha1 recursively.
5. Exceptions – Data already exists, Invalid data, Creating file errors.
6. Main engine classes –
   1. Repository – represents the repository in the system and used as the main application engine.
   2. Magit – used as a connector between the UI and the engine.

UI package is divided by 6 also:

1. Controllers –
   1. Commit node controller – responsible for the display of each node in the commit tree.
   2. Main controller – main connector between the UI and the engine.
   3. Merge node controller – responsible for the display of the merge conflict resolver.
   4. Show commit data – responsible for the display of the commit data.
   5. Show merge case controller – responsible for the display of the list of changes in each file during the merge.
   6. Show status controller – responsible for the display of the working copy status.
2. CSS – responsible for the design of the program and its skins.
3. Layout – util class for updating the commit tree.
4. Magit main – runs the program.
5. Resources – fxml files.
6. UIUtils –
   1. Common used – util functions that are commonly used in the UI.
   2. Commit node – util function for the updating of a commit node in the commit tree.

**Known issues:**

1. Load a repository from XML – not from a different task as wanted.
2. When switching repository to a new repository (which was not in use during the current program run), remote data is lost.
3. When open changes are found one can only commit or disregard it and continue with action, there is no option to cancel currently. Sorry.
4. Sometimes when switching to local repository, access to the sub-folder of Remote Branches raises access denied error
5. Commit tree –
   1. Can't show details on the changes in each commit through the tree.
   2. From the second merge commit, only one edge is added to the tree.
6. Visual –
   1. When maximizing the magit window and afterwards minimizing it to its original size, when the computer mouse is hovering over the commit graph, the window changes its resolution and the magit content doesn't fit the window anymore.

**We did our best, hope you'll enjoy ☺**