

Git Backed Task Management System

Ryan Van Dijck: 1703567

BSc Computer Science

Supervisor: Dr Haider Raza

Second Supervisor: Dr Delaram Jarchi

University GitLab Page:

https://cseegit.essex.ac.uk/ce301_2020/ce301_van_dijck_ryan

Public GitHub Page:

<https://github.com/RyanVanDijck/Gask>

Acknowledgements

This project could not have been completed without the help and guidance of my supervisor, Dr Haider Raza.

The project uses the python programming language, and is deployed using the python package index. It uses the terminaltables, GitPython, matplotlib and pick libraries.

Abstract

The purpose of the project is to create a terminal based to do list application, that stores tasks in a git repository.

Existing Task Management systems make it difficult to work collaboratively, across devices or backup data over the cloud. Although existing task management software has its strengths, each one is lacking in a sense, whether lacking multi-platform capability, or a lack of useful features.

This project will use git as a underlying technology to provide these features with reliability and the ability to track changes. It will be designed to work with central repository such as GitHub or GitLab. The software will use Python, SQLite and JSON to implement a multi-platform solution, with features such as a burn down chart and tracking deadlines.

The ultimate goal of the project is to create a solid and quality foundation for a piece of open-source software that can be built on in the future or used in other software, while still being a user-friendly and complete experience on its own.

The project will be installable using the Pip python package manager, and will be completely open source.

Contents

1	Literature Review	4
1.1	Technical Details	4
1.1.1	Task Management	4
1.1.2	Git Tools	5
1.1.3	Central Repository	5
1.1.4	Similar Software	6
1.1.5	Conclusion	6
1.2	Other Issues	8
1.2.1	Sustainability	8
1.2.2	Ethics	9
1.2.3	Open Source	10
2	Technical Documentation	11
2.1	Overall	11
2.1.1	Overview	11
2.1.2	Implementation	11
2.1.3	Installation	12
2.2	Basic Usage	12
2.3	Commands	13
2.3.1	Command Search	13
2.3.2	Standard	13
2.3.3	Taskspace management	13
2.3.4	Central Repository	14

2.3.5	Extra	14
2.4	Fileutils	16
2.4.1	Directory	16
2.5	Gittools	16
2.5.1	Git commit	16
2.5.2	Central Repo	17
2.6	Start	17
2.6.1	start_prog	17
2.6.2	Create Repo	18
2.6.3	SQL Queries	18
2.7	Taskutils	18
2.7.1	Taskspace	18
2.7.2	Taskspace Class	18
3	Project Demonstration	19
3.1	Installation	19
3.2	Creating a taskspace	19
3.3	Basic Commands	20
3.3.1	Add	20
3.3.2	List	21
3.3.3	Complete	21
3.4	Central Repository	22
3.5	Burndown	23
4	Project Planning	25
4.1	Kanban	25

4.2	User Stories	26
4.2.1	Use Case Diagram	26
4.3	Program Logic	27
4.3.1	Activity Diagram	28
4.3.2	State Diagram	29
5	Conclusion	30
5.1	Achievements	30
5.2	Missed Features and Future Development	30
5.3	Development	31
5.4	Improvements	31

1. Literature Review

1.1 Technical Details

1.1.1 Task Management

In a report on the challenges for Business Process and Task Management, Riss states that task management is a process of managing tasks throughout their life cycle [1]. It can be used by individuals or by people working collectively and covers the processes of planning, testing, tracking and reporting. It can also cover organisation by complexity. In this definition, a task can be defined as a simple item of work to be completed, and task management will deal with creating an order for tasks to be completed and providing a platform to be able to view tasks that have been set.

While discussing the design of various types of task management lists, Bellotti describes a system that helps “users manage and execute their to-dos” [2]. This is a basic and crucial component of a typical task management system, and provides a core idea for a task management application to build on. It also mentioned that it should “Record notes, action items and ideas”. This is an indication that useful task management software should have more features than the simple ability to store tasks. It should also give the user the option to increase the complexity of the task. One of the higher level descriptions of the task is to “Prioritize, manage, and reason about tasks”. This implies that task management software should be able to perform a level of data processing on the tasks, and present these results to the user in a useful manner.

In terms of a collaborate tool, some aspects are mentioned by Bellotti in a report about a task management centred email tool [3]. It is mentioned that “Much of the email’s complexity depends on the nature of the task management activities it is used to support.” In this statement, it is stated that nature and detail of communication between users is determined by what tasks are being organised, discussed or completed. This highlights the need for a collaborative system that is clear to use and can handle large and complex tasks. It also shows the importance of task management in a collaborative space, such as in a workplace.

In the investigation, one important finding was that “some tasks only require a simple rapid response” and “in other cases, a response might be interrupted or delayed while one takes time to gather information”. This is important as it shows the need for a task management system to be able to take the complexity of tasks into account, as well as the quantity of tasks. It was found that users found it difficult to keep “track of lots of concurrent actions”, in terms of the tasks that belong to the user and the tasks needed to be completed by other users. This justifies the existence of such a task management tool, which aims to provide a clearer visualisation of tasks to the user, rather than a cluster of information of various levels of usefulness, or a task manager tacked on to another piece of software.

1.1.2 Git Tools

Git is a technology normally used for collaborative working in software. In their report into the use of git in open source collaborative projects, Lee describes Git as a “favourite source of data for developer behaviour because it has easily accessible rich data” [4]. This is an argument in favour of using git as a backend for a task management tool, as it is a reliable, tried and tested tool for cloud storage and collaboration of work and projects. Two successful projects mentioned by Lee are “Linux and Apache”. In the case of these two projects, particularly Linux, git shows a capability to handle large projects with frequent merges and commits. This point is further emphasised when Lee states that in these open source projects, “developers are inherently spread over the world” and that this process “consists of complex concurrent tasks”. The fact that Git has been able to handle these tasks successfully remaining popular with open source developers shows that is suitable as a platform for task management and collaborative work.

Lee also comments that “Using commit history of a project provides useful information of the development flow.” In terms of a collaborative system, this is useful as it provides a log of tasks set and completed by each individual, and allows easy processing of data in regards to the changes.

In their report on Git, Spinellis describes the git approach as “Revisions not versions” [5]. While Git is Version Control software, it has more of a focus on each individual change, rather than what changes come in each version release. This is useful for task management software as there is no clear use of versions. This evidences that Git is a superior choice than other version control software such as Apache Subversion. Spinellis also describes how git as “Decentralised Revision Control” where it is “easy to push to a remote repository” and “every developer has a separate repository copy”. This is another useful aspect of git in terms of building a collaborative system, as changes can be made offline and then finalised when there is a connection. It also means that changes made are not final in terms of the central repository, and it is easy to undo certain actions.

1.1.3 Central Repository

In this project, the main purpose is to use a third party provider of a centralised git repository such as GitHub or GitLab.

In their report on Git, Spinellis describes the purpose of a third party supplier of a git repository as “maintaining its servers and connectivity, keeping it secure and up to date, setting up user accounts, and supporting your users” [5]. This displays the main advantage of such repositories, as mechanisms for the storage and updating of tasks.

In their report on the use of GitHub in software engineering courses, Feliciano states that “many software engineering and computer science instructors have recently adopted GitHub for hosting their course content or handling assignment submissions”[6]. This is evidence that these platforms are sufficiently robust to handle various use cases, rather than simply websites to store code. Feliciano also mentions that GitHub is popular because of “its open workflow and transparency features”. This is evidence that the use of Git repositories and Application Programming Interfaces

(APIs) will be sufficiently able to create a quality and useful product. Feliciano finally mentions that “GitHub does excel by creating a culture of participation”. This supports the idea that Git repositories are an excellent way to provide a collaborative working environment, and that is has the technical capacity to achieve this goal.

1.1.4 Similar Software

Although there are many examples of command line based task management software, no examples can be found of a git backed task management system on GitHub [7] or on GitLab [8]. This is evidence that this is a unique and untested project title.

The most similar project found on either of the two websites is Busy [9]. This is a task management system which is written in python and installs using pip. This project has no simple way to allow cloud storage of tasks and collaborative use or use across different devices. This is evidence that there is gap for software based on a central repository that is possibly online.

Another similar and popular project is Taskwarrior [10][11]. Taskwarrior is written in C++, and has more features than Busy [9], such as Burndown Charts and tags [11]. Taskwarrior also comes with extensive documentation on its many features. It is also able to facilitate collaborative work and working on multiple devices though a Task server. However, unlike using a central repository such as GitHub or GitLab, users need to either run their own server, or use a community task server account. This displays a huge advantage to a git based task management system, as you can choose to store tasks on a local repository or on a central one, handled securely by the maintainers of the services [12][13].

1.1.5 Conclusion

In conclusion, the research that has been completed can be used to argue that this is a unique product that can provide a useful benefit over other task management software. This research has compiled and referenced many sources, including academic reports, GitHub and GitLab pages, and product websites.

In the first section, the definition and nature of Task Management is discussed. It is defined as “a process of managing tasks throughout their lifecycle” [1]. In this section, there is a discussion of why task management systems are needed and what constitutes useful task management software. This is important for understanding what benefits this project could deliver to the end user, with the added complexity of it being based on git.

In the second section, general features of git were discussed, along with why these features provide utility and stability in terms of a task management system. These features included a commit history and an ability to revert. This is a crucial part of the discussion as it describes the advantage of the unique aspect of the software.

In the third section, central repositories were discussed, such as GitHub or GitLab. This was also important to the review, as it showed an important possibility in terms of collaborative working and working from various devices.

In the final section, there was a discussion regarding existing task management software. It was specifically regarding what similarities there were between them and the concept for the project, what their strengths were, and how using git as a back-end could provide an advantage compared to them. This was useful to understand what features were important and how improvements could be made to features in those projects. As they were all open source, it was useful to observe how they were structured and some of the technical challenges that may be faced during the completion of this project.

Using the research, a clear argument may be formed that the result of the project is a unique piece of software with a strong technological foundation.

1.2 Other Issues

1.2.1 Sustainability

In terms of software, sustainability can be defined as actions which cause minimal damage to the economy, society, individuals and the environment or which create positive outcomes for these entities [14].

In terms of the definition, the project created has few issues with sustainability. It involves the deployment of free and open-source software involving no hardware or other physical elements. In terms of society, there is a seemingly minimal impact for such a niche product.

Environmental Factors

The main environmental threat posed by this type of software is an increased power consumption [15]. One factor in this is software that receives frequent updates. This software needs more data centres and network traffic. On average, one data centre uses the same amount of power as about 25,000 households [16].

This highlights the need to reduce unnecessary updates and other user downloads. This can be achieved by more thorough testing before deployment and the use of a nightly system, where some more advanced users will receive more frequent updates but are more likely to meet bugs [17]. This means that the vast majority of users receive fewer updates.

These factors will become most important during the maintenance and improvement of the product in the future.

Project design

However, sustainability of software goes beyond the code, and is also dependent on design documentation and architecture [18]. As such, the software was planned thoroughly using UML activity and state diagrams to demonstrate the logic of some parts of the software.

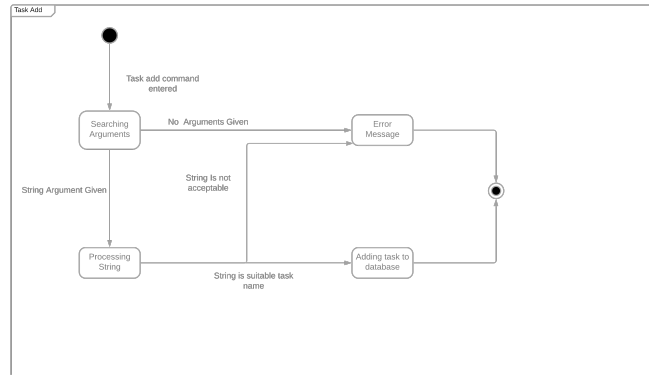


Figure 1.1: A state diagram showing the logic the "add task" portion of the program.

In terms of the architecture, a clear structure has been completed in the project code files. The functions and components of the software are stored in their labelled Python namespaces, with documentation to explain what each individual part does and in terms of the "commands" folder, what each command does and what arguments they may take.

1.2.2 Ethics

Ethics can be defined as "the study of value concepts such as 'good', 'bad', 'right', 'wrong,' and 'ought', applied to actions in relation to group norms and rules" [19].

Because of the increasing role of software in our lives, people are starting to take a larger interest in the ethical nature of software that they are using [20]. Some of the more prevalent issues in this area regard privacy, accuracy, property, and accessibility [19].

Code of Ethics

In 1993, the IEEE and ACM created a joint code of ethics for software engineers [21].

The first principle states that work should be useful and free of error. This work was completed with an agile user story model, which assures that the project is defined with the utility of features at the forefront. Principle 1.01 specifically mentions that code should be "well documented". This has been completed, with the use of commenting in code and the creation of markdown documents which explain how to use the software and how it works. This means that a potential user has a guide on how to use the software, and how to make changes. This is important for an open source project, where users are permitted and encouraged to make changes to the software to personalise it to their needs.

The second principle states that work should be "consistent with the public safety, health, and wel-

fare". In this aspect, not too many issues have been identified. There is not particularly any additional risk in this project in these regards compared to other terminal based software. Software in general is not dangerous in a typical sense, with a lack of a physical, radioactive or poisonous outcome to people or to the environment [22].

1.2.3 Open Source

The main legal issue surrounding the project is the license attached to it. While the original intention was always for it to be open sources, the choice of which license to use was more difficult.

The license that has been chosen is version 3 of the Free Software Foundation's General Public License (GPLv3) [23].

According to the preamble of the licence, it is a copyleft license, which means that you can "change the software or use pieces of it in new free programs", but you have to publish any changes.

According to Stoltz, the purpose of the GPL is to preserve the values of open source development [24]. This was the main purpose of using license. Although other licenses, such as the MIT license [25], are more permissive, the GPL license was chosen as it protects the open source nature of the software, by not permitting it to be used in proprietary software.

Version 3 of the license was released in 2007 and is the latest version of the license [23]. According to Kumar, it extends the GPL to "cover software-related patents, and prohibits digital rights management that limits access to the software" [26]. This version was chosen as it is the most up to date version meaning that it is the most suited to deal with modern issues. It was also chosen as it deals with additional areas compared to previous versions, such as the areas mentioned by Kumar, and "Tivoization", which is the "Practice of incorporating GPL software on a consumer device or appliance that the user cannot change, because the manufacturer uses hardware to disable the device if it detects modified software" [27].

2. Technical Documentation

This section of the report will discuss technical aspects of the project, discussing use of the software and highlighting technical achievements.

2.1 Overall

2.1.1 Overview

This project is a terminal application which stores a to do list over a git repository, it has support for central repositories such as a GitHub or GitLab repository and is completely multiplatform.

2.1.2 Implementation

The application is written using the Python programming language, and is split into python namespaces. It is run by running the "gask.py" file.

It contains five main namespaces:

- The commands namespace, which stores the code that runs specific to the given argument, and the commandsearch.py file, which uses to argument to run the correct function.
- The fileutils namespace, which contains functions related to important filepaths and other crucial function related to parts of the application which use file input and output.
- The gittools namespace, which contains functions that interact with the local and a remote git repository.
- The start namespace, which contains functions that run when the command is given, and if there is no existing repository will start the process of creating a new one.
- The taskutils namespace contains functions and a class pertaining to a "taskspace" or a folder which contains the task database file and the git repository, as well as an information file.

2.1.3 Installation

The software can be installed by cloning the git repository, moving to the top level directory of the repository and entering the command:

```
pip install .
```

It can also be installed using pypi using the command:

```
pip install Gask
```

2.2 Basic Usage

When the "gask" command is first given, the user will be prompted to create a taskspace, unless the import command is given.

Once a taskspace is found or created, the full breath of commands can then be used.

The basic commands are:

```
gask add <task name/description>
```

```
gask list
```

```
gask complete <task id>
```

2.3 Commands

https://cseegit.essex.ac.uk/ce301_2020/ce301_van_dijck_ryan/-/tree/master/gask/commands

2.3.1 Command Search

This file takes the command line argument given to the application and runs the corresponding function.

2.3.2 Standard

Add

This command adds a task to the database given the name of the task as an argument.

List

This commands displays the current tasks that need to be completed.

Complete

This command marks a task as complete and removes it from the list and moves it to the completed database.

2.3.3 Taskspace management

Create

This command will prompt the user to create a new taskspace.

Delete

This command will delete an existing taskspace, and give the user an option to delete the actual folder as well.

Switch

This command will switch the current taskspace.

Set

This command will set or change information about a task, such as the name or the deadline.

2.3.4 Central Repository

Set Remote

This command will set a remote repository, such as a GitHub or GitLab repository, as the remote repository for a taskspace.

Push and Pull

These commands, like their git counterparts, interact with a central repository.

Update

This command combines the push and pull commands to perform a full sync with a git repository.

2.3.5 Extra

Burndown

This command uses the matplotlib python library [28] to draw a burndown chart.

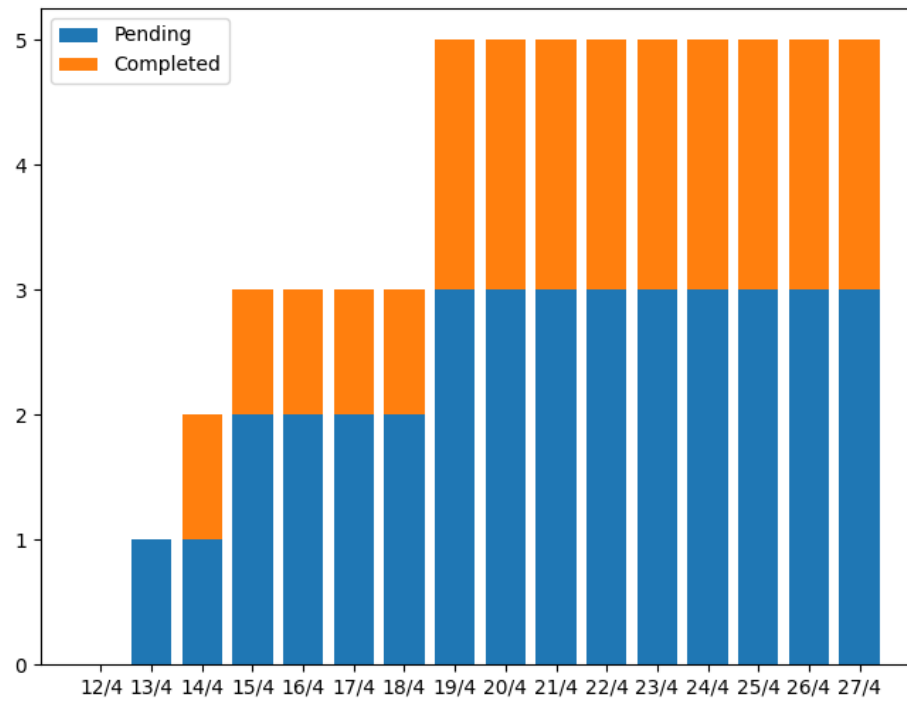


Figure 2.1: A Burndown Chart Generated by the application

2.4 Fileutils

https://cseegit.essex.ac.uk/ce301_2020/ce301_van_dijck_ryan/-/tree/master/gask/fileutils

2.4.1 Directory

This file holds functions related to important file locations.

get_top_directory

This function gets the install folder of the application.

get_repos_path

This returns the path to the repos.json file, where a list of the current taskspaces is stored.

get_user_folder

This creates a prompt for the user to select a folder. It gives a graphical file opener if it can be loaded.

2.5 Gittools

https://cseegit.essex.ac.uk/ce301_2020/ce301_van_dijck_ryan/-/tree/master/gask/gittools

2.5.1 Git commit

This file contains functions related to making and receiving changes to and from a central repository.

git_commit

This function will carry changes to the central repository in the current taskspace.

2.5.2 Central Repo

. This file contains other functions pertaining to interacting with a central repository, such as a GitHub or GitLab repository.

create_central_repo

This function creates an object which can be used to interact with a central repository.

update_to

Commits changes to the central repository.

update_from

Receives changes from the central repository.

2.6 Start

https://cseegit.essex.ac.uk/ce301_2020/ce301_van_dijck_ryan/-/tree/master/gask

2.6.1 start_prog

This file holds functions used when the program is first run.

first_check

This function checks if there is an existing taskspace and then prompts the user to create one if there isn't and if the import command hasn't been given. If there is an existing taskspace, the program will continue using the arguments.

create_taskspace

This function will prompt the user to create a taskspace.

2.6.2 Create Repo

This file contains a function to create a new taskspace.

create_repo

This function will create a new taskspace. It is the point for creating a taskspace when the program is first run and when the create command is given.

2.6.3 SQL Queries

The two functions *get_create_tasks_query* and *get_completed_tasks_query* return the queries used to create the databases.

2.7 Taskutils

https://cseegit.essex.ac.uk/ce301_2020/ce301_van_dijck_ryan/-/tree/master/gask/taskutils This namespace contains functions, methods and a class related to the creation and management of taskspaces.

2.7.1 Taskspace

A taskspace is a label for a directory in which the repository, the database file and an information file are stored.

2.7.2 Taskspace Class

This class stores information about taskspaces. It has been created to simplify the logic in other parts of the program.

3. Project Demonstration

This chapter will show a demonstration of the basic features of the program. It will be completed in a virtual environment using Python version 3.9.4 and pip version 20.2.3.

3.1 Installation

The application is installed using a pip install command.

```
(demonstration) [ryan@RyanLenovo demonstration]$ pip install gask
Processing /home/ryan/.cache/pip/wheels/12/73/39/77282ae1dfea093d36d2f783da2d409532144376eddb6a852d/Gask-0.4-py3-none-any.whl
Collecting gitpython
  Using cached GitPython-3.1.14-py3-none-any.whl (159 kB)
Collecting pick
  Using cached pick-1.0.0-py2.py3-none-any.whl (5.5 kB)
Collecting terminaltables
  Using cached terminaltables-3.1.0.tar.gz (12 kB)
Collecting gitdb<5,>=4.0.1
  Using cached gitdb-4.0.7-py3-none-any.whl (63 kB)
Collecting smmap<5,>=3.0.1
  Using cached smmap-4.0.0-py2.py3-none-any.whl (24 kB)
Using legacy 'setup.py install' for terminaltables, since package 'wheel' is not installed.
Installing collected packages: smmap, gitdb, gitpython, pick, terminaltables, gask
  Running setup.py install for terminaltables ... done
Successfully installed gask-0.4 gitdb-4.0.7 gitpython-3.1.14 pick-1.0.0 smmap-4.0.0 terminaltables-3.1.0
WARNING: You are using pip version 20.2.3; however, version 21.1 is available.
You should consider upgrading via the '/home/ryan/demonstration/bin/python -m pip install --upgrade pip' command.
(demonstration) [ryan@RyanLenovo demonstration]$
```

3.2 Creating a taskspace

The user is prompted to create a taskspace when the *gask* command is first given.

```
(demonstration) [ryan@RyanLenovo demonstration]$ gask
No repo found
Would you like to to create a new Taskspace? (y/n)
█
```

The user is then prompted to enter information about the taskspace.

```
(demonstration) [ryan@RyanLenovo demonstration]$ gask
No repo found
Would you like to to create a new Taskspace? (y/n)
y
Please enter a name
demo
Would you like to use the current folder? (y/n)
n
```

If the user elects not to use the current directory, then a graphical prompt is used to allow the user to select a folder.

```
/home/ryan/demonstration
--> Use This Folder
    Make New Folder
    Hide Hidden Folders
    ..
    bin
    include
    lib
    lib64
```

3.3 Basic Commands

This section will cover some simple commands that are standard across terminal based to do lists.

3.3.1 Add

Tasks can be added to the list using the *add* command. They can be added with or without quotation marks.

```
(demonstration) [ryan@RyanLenovo demonstration]$ gask add "complete report"
(demonstration) [ryan@RyanLenovo demonstration]$ gask add "complete presentation"
(demonstration) [ryan@RyanLenovo demonstration]$
```

3.3.2 List

Currently pending tasks can be displayed using the *list* command.

```
(demonstration) [ryan@RyanLenovo demonstration]$ gask list
+-----+-----+-----+-----+
| Id | Description | Deadline | Date Set |
+-----+-----+-----+-----+
| 1 | complete report | None | 2021-04-28 |
| 2 | complete presentation | None | 2021-04-28 |
+-----+-----+-----+-----+
(demonstration) [ryan@RyanLenovo demonstration]$
```

3.3.3 Complete

A task can be marked as finished using the *complete* command giving the task id. This will no longer show up in the list command.


```
(demonstration) [ryan@RyanLenovo demonstration]$ gask complete 2
(demonstration) [ryan@RyanLenovo demonstration]$ gask list
+-----+-----+-----+-----+
| Id | Description | Deadline | Date Set |
+-----+-----+-----+-----+
| 1 | complete report | None | 2021-04-28 |
+-----+-----+-----+-----+
(demonstration) [ryan@RyanLenovo demonstration]$
```

3.4 Central Repository

This section will cover interaction with a central repository, using a private GitLab repository.


The remote is set using the *setremote* command. The user will need to verify their account information.

```
(demonstration) [ryan@RyanLenovo demonstration]$ gask setremote https://cseegit.essex.ac.uk/rv17643/reportdemo.git
Username for 'https://cseegit.essex.ac.uk': rv17643
Password for 'https://rv17643@cseegit.essex.ac.uk':
(demonstration) [ryan@RyanLenovo demonstration]$
```

The changes can then be deployed to the central repository using the push command.

```
(demonstration) [ryan@RyanLenovo demonstration]$ gask setremote https://cseegit.essex.ac.uk/rv17643/reportdemo.git
Username for 'https://cseegit.essex.ac.uk': rv17643
Password for 'https://rv17643@cseegit.essex.ac.uk':
(demonstration) [ryan@RyanLenovo demonstration]$ gask push
Username for 'https://cseegit.essex.ac.uk': rv17643
Password for 'https://rv17643@cseegit.essex.ac.uk':
(demonstration) [ryan@RyanLenovo demonstration]$
```

The changes are then reflected on the central repository.



Completing task
 Van Dijk, Ryan authored 23 minutes ago

14c01c84

Add README



Add LICENSE

Add CHANGELOG

Add CONTRIBUTING

Add Kubernetes cluster

Set up CI/CD

Name	Last commit	Last update
 demo.db	Completing task	23 minutes ago
 info.json	Adding task	30 minutes ago

3.5 Burndown

A burndown chart can be drawn in a separate window using the *burndown* command if the matplotlib [28] library is installed.

```
(demonstration) [ryan@RyanLenovo demonstration]$ gask burndown
```



4. Project Planning

This chapter will cover the planning made before and during development of the project.

4.1 Kanban

The bulk of the planning was completed using a Kanban board in Jira. In Kanban a board "consists of several columns, each column representing a stage in the development process" [29]. This same Kanban board was also used to house supervisor feedback.

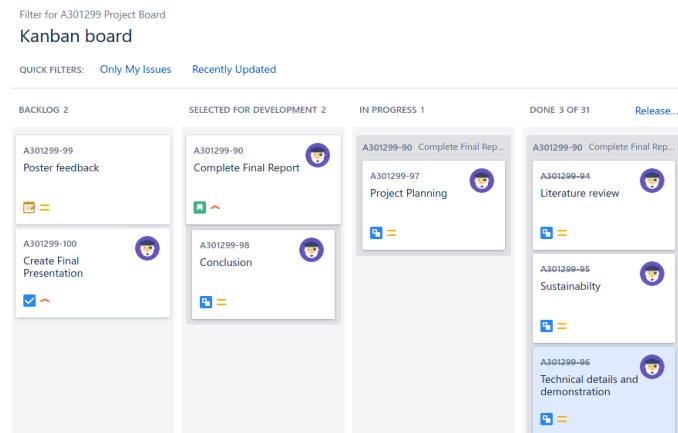


Figure 4.1: A screenshot of the Kanban board taken during the completion of this report.

On the Kanban board, there were four columns:

1. The backlog
2. Selected for development
3. In progress
4. Done

The backlog column was where the majority of tasks were stored throughout most of development. It is used to store all possible tasks and stories. This meant that some of the tasks in the backlog never moved on in development as they could have been made obsolete by later planning.

The selected for development column was used to store tasks that were not yet being implemented but that were planned to be completed.

The in progress column was used to store tasks and stories that were currently under development. It was usually kept to a maximum of two tasks, to ensure that there was proper focus, and a higher efficiency under the Kanban system.

The done column was used to store tasks that were marked as complete, or tasks that would not be completed. At some intervals, the done column was cleared to keep the board easy to look at.

The use of the Kanban system was useful as it made it easier to stay in task and easily observe how many tasks need to be completed. Kanban reduces the current work in progress [29] at one time, which makes it easier to focus on particular aspects of the software at a time. This method also could be integrated into the agile user story model used.

4.2 User Stories

The features of the software were primarily planned using user stories. This was done so that the software would be created with utility for a potential user was the first priority when deciding which features were to be implemented. The user stories were created using various perspectives, considering the needs of technical users and less technical users, as well as thinking about features that would be useful for users who are not using the system directly, such as an employer.

When completing the work for a user story, the first task was to divide it into separate tasks. This involves deciding what the story specifically entails and what technical challenges need to be overcome. After all of the tasks were completed, the stories were also marked as finished in Jira.

This process was mainly successful, with few unnecessary features and existing features programmed with different types of users in mind.

4.2.1 Use Case Diagram

At the beginning of the project, the initial user stories were used to create a basic use case diagram. This was done to illustrate a visual overview of the basic functions of the program.

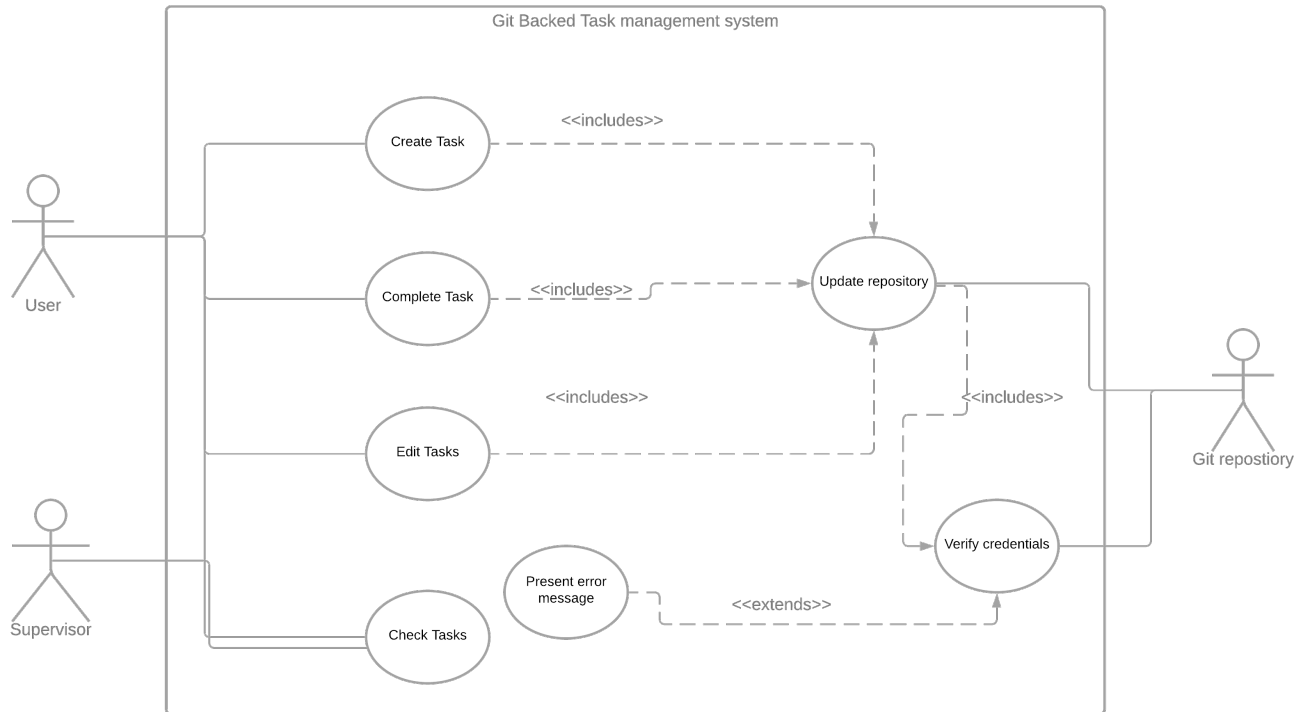


Figure 4.2: A basic use-case diagram.

4.3 Program Logic

For the most part, the logic of the program is simple, and can be explained using the markdown documentation and in code comments.

However, at the beginning of the project, some technical diagrams were used to express some of the logic of the program.

4.3.1 Activity Diagram

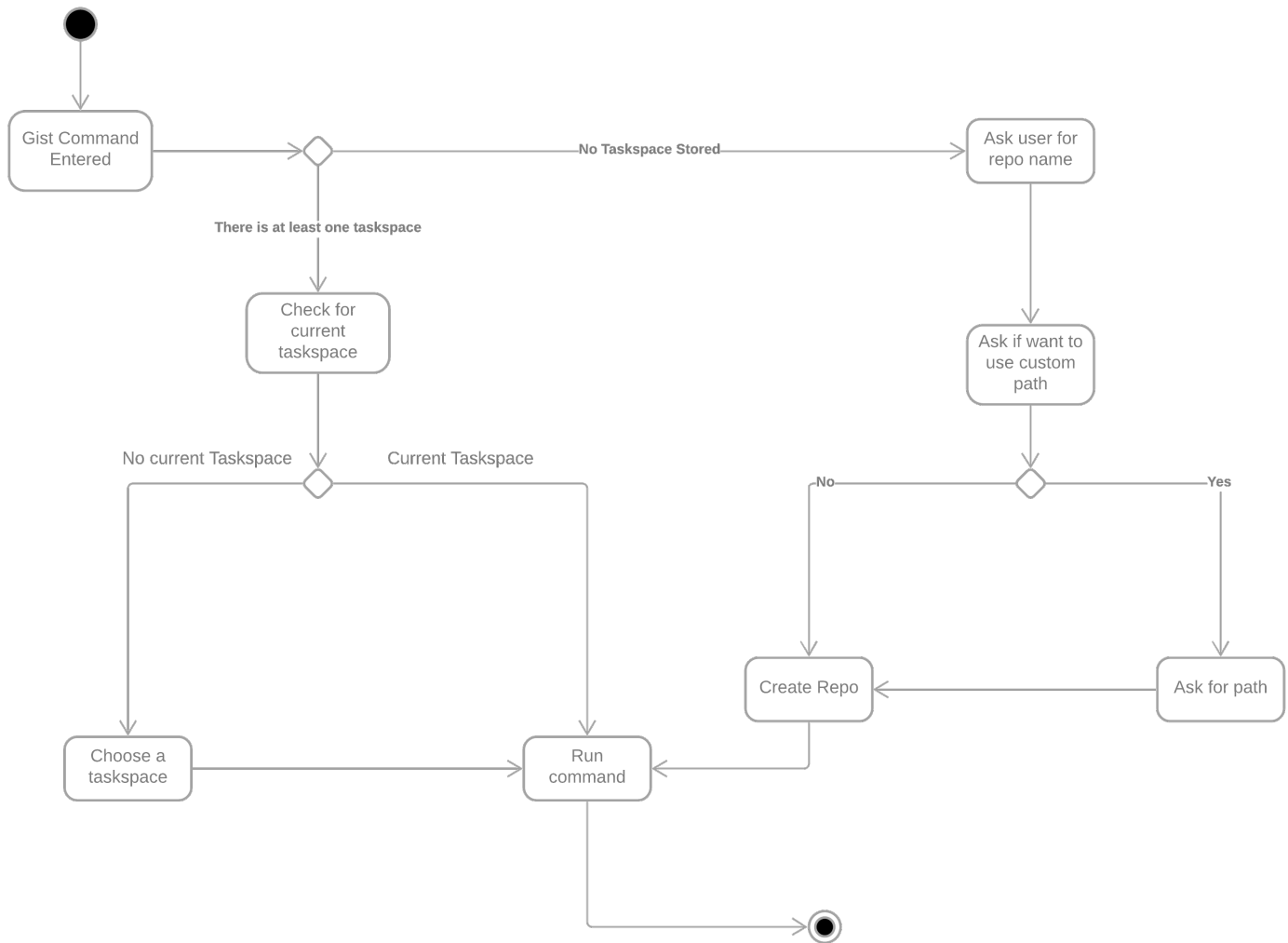


Figure 4.3: A UML activity diagram of the logic of the program at start up.

4.3.2 State Diagram

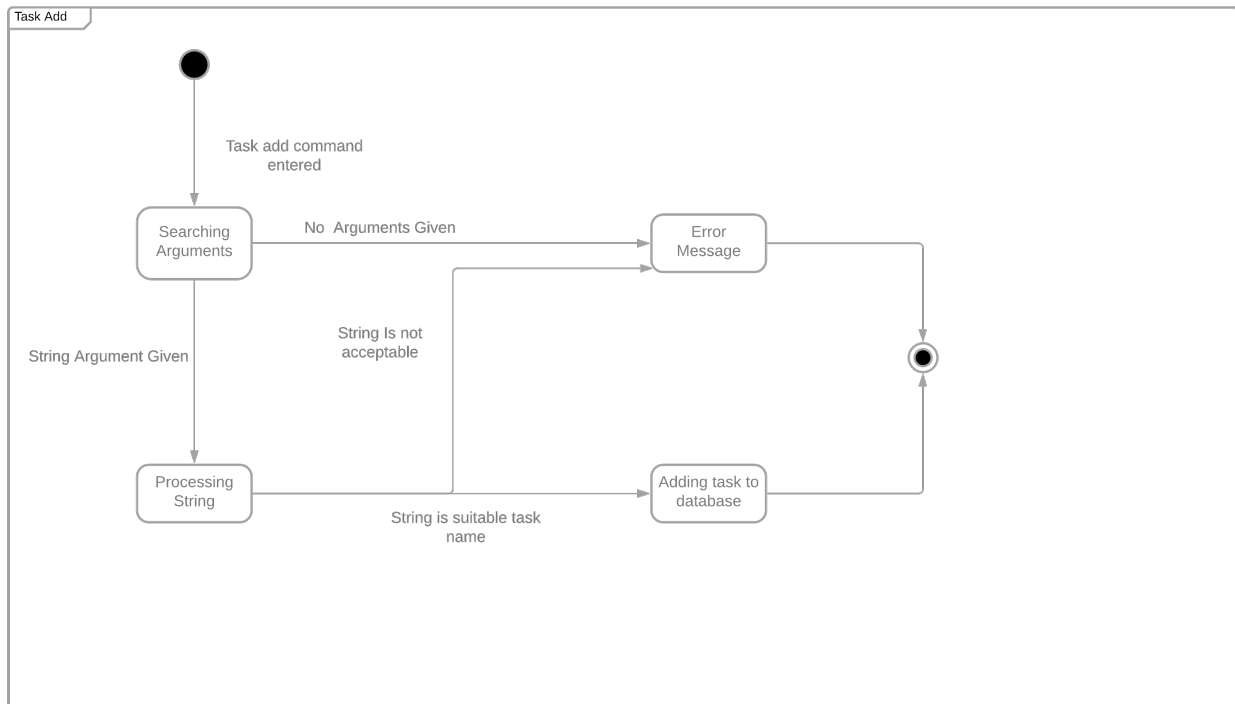


Figure 4.4: A UML state diagram to express the logic when the *add* command is given

5. Conclusion

In conclusion, the project was implemented successfully according to the given description and the planning, including the user stories. A git-backed multiplatform task management system has been created. It is able to interact with a GitHub or a GitLab repository.

In general, the software has been completed to a decent standard, and is easy enough to use with basic instruction.

5.1 Achievements

One of the main achievements is that the software is easy to install using a basic terminal command on a machine on which Python is installed. This has allowed easy testing of the software across various platforms, as well as it being beneficial for a potential user.

One of the most difficult challenges during the completion of the project was the burndown chart. This involved complex database queries, extensive use of the Python datetime library and some complicated logic. This was a large achievement and the matplotlib library has given it a quality look and feel.

The use of various technologies has been a huge benefit during the completion of the project. These technologies have included SQLite and JSON. These technologies have required a large amount of learning and evolution in programming knowledge and skill.

The GitPython library was sometimes difficult to use, especially relating to central repository use, due to limited documentation.

5.2 Missed Features and Future Development

Unfortunately, due to illness during the first term, there was not sufficient time to implement all desired features. These could be implemented to the public version of the software in the future.

One of these features was more direct git control. It would have been better for the software to be able to deal with eventualities such as merge conflicts.

It would have been better if the software had more polish, such as a help page to display the possible arguments.

There is no highlighting of tasks that are on or passed their deadline.

In the future, some of the commands, such as the *set* and *delete* command need to be developed more to allow commands to be given directly as well as the graphical options.

5.3 Development

Overall, development was an effective process. Following the user stories gave a clear pathway to completing the software with useful features. However, this also meant that a significant amount of time was spent on organisation, which could be frustrating at times.

The use of Python was helpful due to the inherent simplicity of the language syntax. The decision was difficult at first but this was the simplest way to create a terminal application that was easy to deploy and multiplatform with minimal platform specific code. There was only platform specific code in the folder choose dialogue, to allow the user to change drives on windows, which is not an existing concepts in Unix based systems, with all drives being accessible in the main filesystem [30].

GitLab was an essential tool in the development process, allowing working across devices and providing a neat template to display and organise code using markdown documents. It has also been used to store documents not related to development.

5.4 Improvements

During completion of the project, it would have been better to spread the workload more evenly. This would have meant that there was less burnout at certain periods.

It also would have been better to have written significant documentation sooner after code was written so that the logic was more familiar.

The use of various Integrated Developer Environments at the start of the project lead to confusion regarding Python namespaces and project organisation. It would have been better to sort these things before development had started.

The *import* command was placed into the codebase haphazardly, due to a lack of thought of how to implement this command when devising the command structure.

Overall, there should have been a clearer plan on how the separate functions and namespaces were combined to create a clearer project structure.

Bibliography

- [1] R. V. Uwe, A. Rickayzen, H. Maus, and W. M. P. van der Aalst, “Challenges for business process and task management,” *Journal of Universal Knowledge Management*, vol. 0, no. 2, pp. 77 – 80, 2005. [Online]. Available: http://www.jucs.org/jukm_0_2/riss/jukm_0_2_77_100_riss.pdf
- [2] V. Bellotti, B. Dalal, N. Good, P. Flynn, D. G. Bobrow, and N. Ducheneaut, “What a to-do: Studies of task management towards the design of a personal task list manager,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’04. New York, NY, USA: Association for Computing Machinery, 2004, p. 735–742. [Online]. Available: <https://doi.org/10.1145/985692.985785>
- [3] V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith, “Taking email to task: The design and evaluation of a task management centered email tool,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’03. New York, NY, USA: Association for Computing Machinery, 2003, p. 345–352. [Online]. Available: <https://doi.org/10.1145/642611.642672>
- [4] H. Lee, B. Seo, and E. Seo, “A git source repository analysis tool based on a novel branch-oriented approach,” in *2013 International Conference on Information Science and Applications (ICISA)*, 2013, pp. 1–4.
- [5] D. Spinellis, “Git,” *IEEE Software*, vol. 29, no. 3, pp. 100–101, 2012.
- [6] J. Feliciano, M.-A. Storey, and A. Zagalsky, “Student experiences using github in software engineering courses: A case study,” in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 422–431. [Online]. Available: <https://doi.org/10.1145/2889160.2889195>
- [7] GitHub. Build software better, together. A search on the official GitHub website. [Online]. Available: <https://jonas-moennig.de/how-to-cite-a-website-with-bibtex/>
- [8] GitLab. Gitlab. A search on the official GitLab website. [Online]. Available: https://gitlab.com/search?utf8=%E2%9C%93&snippets=false&scope=&repository_ref=&search=Git+Backed+Task+Management
- [9] F. Potter. Busy. GitLab page for the busy task manager. [Online]. Available: <https://gitlab.com/fpotter/tools/busy>
- [10] TaskWarrior. Taskwarrior. GitHub page for TaskWarrior. [Online]. Available: <https://github.com/GothenburgBitFactory/taskwarrior>
- [11] ——. Taskwarrior. TaskWarrior website. [Online]. Available: <https://taskwarrior.org/>

- [12] GitHub. Security. GitHub webpage about the security of the service. [Online]. Available: <https://github.com/security>
- [13] GitLab. Security. GitLab webpage about the security of the service. [Online]. Available: <https://about.gitlab.com/handbook/engineering/security/>
- [14] C. Calero, M. F. Bertoa, and M. A. Moraga, "A systematic literature review for software sustainability measures," in *2013 2nd International Workshop on Green and Sustainable Software (GREENS)*, 2013, pp. 46–53.
- [15] N. Amsel, Z. Ibrahim, A. Malik, and B. Tomlinson, "Toward sustainable software engineering: Nier track," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 976–979.
- [16] P. Lago and T. Jansen, E. M. Maximilien, G. Rossi, S.-T. Yuan, H. Ludwig, and t. E. A. i. S. O. S. E. b.-O. C. Fantinato, Marcelo", Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 181–186.
- [17] F. Khomh, T. Dhaliwal, Y. Zou, and B. Adams, "Do faster releases improve software quality? an empirical case study of mozilla firefox," in *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*, 2012, pp. 179–188.
- [18] R. Seacord, J. Elm, W. Goethert, G. Lewis, D. Plakosh, J. Robert, L. Wrage, and M. Lindvall, "Measuring software sustainability," in *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, 2003, pp. 450–459.
- [19] A. J. Thomson and D. L. Schmoldt, "Ethics in computer software design and development," *Computers and Electronics in Agriculture*, vol. 30, no. 1, pp. 85–102, 2001. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0168169900001587>
- [20] F. B. Aydemir and F. Dalpiaz, "A roadmap for ethics-aware software engineering," in *2018 IEEE/ACM International Workshop on Software Fairness (FairWare)*, 2018, pp. 15–21.
- [21] D. Gotterbarn, K. Miller, and S. Rogerson, "Software engineering code of ethics," *Commun. ACM*, vol. 40, no. 11, p. 110–118, Nov. 1997. [Online]. Available: <https://doi.org/10.1145/265684.265699>
- [22] M. P. Heimdahl, "Safety and software intensive systems: Challenges old and new," in *Future of Software Engineering (FOSE '07)*, 2007, pp. 137–152.
- [23] Free Software Foundation. The gnu public licence v3.0. The GNU public licence version 3.0. [Online]. Available: <https://www.gnu.org/licenses/gpl-3.0.en.html>
- [24] S. L. Mitchell, "The penguin paradox: How the scope of derivative works in copyright affects the effectiveness of the gnu gpl," *Boston University Law Review*, vol. 85, no. 5, pp. 1439–1478, December 2005.
- [25] Massachusetts Institute of Technology. Mit license. The Mit License. [Online]. Available: <https://mit-license.org/>

- [26] S. Kumar, “Enforcing the gnu gpl,” *University of Illinois Law, Technology & Policy*, vol. 2006, no. 1, pp. 1–36, Spring 2006.
- [27] J. Tsai, “For better or worse: Introducing the gnu general public license version 3,” *Berkeley Technology Law Journal*, vol. 23, no. 1, pp. 547–581, 2008. [Online]. Available: <http://www.jstor.org/stable/24118327>
- [28] Matplotlib. matplotlib. The matplotlib python library. [Online]. Available: <https://matplotlib.org/>
- [29] V. Mahnic, “Improving software development through combination of scrum and kanban,” *Recent Advances in Computer Engineering, Communications and Information Technology*, pp. 281–288, January 2004.
- [30] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, “A fast file system for unix,” *ACM Trans. Comput. Syst.*, vol. 2, no. 3, p. 181–197, Aug. 1984. [Online]. Available: <https://doi.org/10.1145/989.990>