1. **Introduction** 
   1. **Purpose**

We want to build a Code Management and Version Control System, based on the concept of Git. This product, is inspired by the platform of Github, and will be having the necessary features to meet user requirements.

* 1. **Intended Audience**

This product will be targeting any person, who has a document, and wishes to use version control, to efficiently make & save changes while working. The code management aspect is targeting users, who are using this platform, to make changes to a code based file. The users can be anyone from a school or university students, to professional working software developers.

* 1. **Product Scope**

This product will be primarily focusing on 5 features for Version Control and Code Management, which are:

**Push, Pull, Commit, Staging and Checkout**

These features encompass all necessary requirements for making a good version control system, which can provide code management for our users. The product will be inspired by the current popular version control system of Github.

* 1. **References**

Our Product is using concepts of version control tool: Git and Github

1. **Overall Description** 
   1. **Product Perspective**

This Product is inspired by the popular version control tool Git. We hope to emulate the functionality aspects of Git & Github, to make a good Code Management & Version Control System. We will also aim to understand how Github works, in this process

* 1. **Product Functions**
* **Staging:** This is done to confirm which changes in document have to be committed
* **Commit:** This operation stores the staged changes in the git file
* **Push:** This operation uploads the git directory from local system to hosted environment
* **Pull:** This operation retrieves the git directory from hosted environment to the local system
* **Checkout:** This operation is used to revert to a specified commit
  1. **User Classes and Characteristics**
* **Observer as a User:** All Students, Professionals or Developers who are just observing the hosted repositories and are not interacting with the Git tool
* **Developer as User:** All students, Professionals or Developers who are using the Git tool functionalities
  1. **Operating Environment**

This product is compatible with both Windows and Linux Based Operating System Types. The user can access this from his/her own computing environment, and only needs to install the Git Tool, which is open source software. No specific Hardware requirements needed to execute.

* 1. **Design and Implementation Constraints**
* The push and pull operations require an internet connection
* In the hosted environment, the amount of storage for the documents is dependent on the server
* User Privacy in terms of data must be maintained
* If hosted environment is not used, then the life of the data is solely dependent on the local system of the user
  1. **Assumptions and Dependencies**
* We assume that the user knows how to operate the Git Tool Functionality
* We are depending on the Git Tool and its functions
* In the Use Case Diagrams, for the functions, it is assumed that the user is using all functions in required git directory

1. **External Interface Requirements**

**3.3 Communications Interfaces**

1. Our user should have an email for registering into our code management and version control service.
2. When users stores data in his repositories in our app the data should be communicated to the backend servers as a backup to provide reliability.
3. Communications standards required should include persistent HTTP/1.1 for sending and retrieving multiple files to and from the repositories and also FTP should be required as we are dealing with all types of files.
4. The files that the user wants to send from his local system to his repositories should get encrypted to prevent any data sniffing by malicious external sources.
5. The commits made by multiple users working on the same project should be synchronized properly and should be notified in the common directory shared by the users.

**5. System Features**

**5.1 Logging in the System**

5.1.1 Description and Priority

This feature allows a user to login to the command line with Git, so they can use functionality provided by the Product. The priority of this is medium, as the functions don’t depend on this logically, but login is required to ensure access to the system.

5.1.2 Stimulus/Response Sequences

1. User enters name and system registers it

2. User enters email id, and system registers it

3. System authenticates the user

5.1.3 Functional Requirements

* Back End Server to authenticate the login
* Software Protocol in case of Invalid Login attempt

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

**5.2 Staging the Changes**

5.2.1 Description and Priority

This feature allows a user to define which changes they want to commit, and which ones they don’t. They can use various flags to define if they want to stage all the changes, or just one of the particular documents. This is the pre-requirement of the commit function. This is given High Priority, as without staging, the user cannot commit changes.

5.2.2 Stimulus/Response Sequences

1. User enters command to stage changes he wants

2. System registers those changes and updates the tracked files

3. User can see which changes are tracked with the status command

5.2.3 Functional Requirements

* The git system must be initialized in the directory in order to track the changes
* Functionality must be made available in order to modify or undo the tracked changes

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

**5.3 Committing Changes**

5.3.1 Description and Priority

This feature allows a user to store the tracked version of a file through the Git Functionality. This is the essence of Version Control. The user can always revert back to a previous commit. This is given High Priority, as this is the most important feature responsible for version control.

5.3.2 Stimulus/Response Sequences

1. User enters command to commit the given staged changes

2. System registers the commit, and adds that to the local git file

3. User can revert to a specific commit with the checkout functionality

5.3.3 Functional Requirements

* The required changes to be committed must staged beforehand
* The git tool must be working in the required directory, which would be storing the changes

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

**5.4 Pushing the Changes**

5.4.1 Description and Priority

This feature allows a user to send the latest committed file to a remote hosted server (Github) repository, where the local files can be stores, to ensure access from anywhere in the world for the user. This feature has a medium priority, as git can work on the local system, and push is only required to use the hosted aspects of version control. However this is extremely important for work collaboration and sharing.

5.4.2 Stimulus/Response Sequences

1. User enters command to push the changes to the required repository

2. System verifies that the repository exists

3. System uses internet to ensure that the changes are sent to the hosted environment

5.4.3 Functional Requirements

* The origin repository must be set beforehand, where the files would be sent to
* There must exist some data/files in the master repository, which are going to be pushed

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

**5.5 Pulling Files from a hosted Repository**

5.5.1 Description and Priority

This feature allows a user to retrieve files from a hosted repository, and clone it to their local system. This feature has a medium priority, as git can work on the local system, and pull is only required to use the hosted aspects of version control. However this is extremely important for work collaboration and sharing.

5.5.2 Stimulus/Response Sequences

1. User enters command to pull the files from the required repository

2. System verifies that the repository exists

3. System uses internet to ensure that the files are sent to the user local system

5.5.3 Functional Requirements

* The origin repository must be set beforehand, where the files would be pulled from
* There must exist some data/files in the master repository, which are going to be sent to local system

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

**5.6 Checkout to a version**

5.6.1 Description and Priority

This feature allows a user to roll back or go to a specific version of a file (commit).This feature has a high priority, as this is an essential part of version control & code management. Without this feature, we cannot make use of the commits we have.

5.6.2 Stimulus/Response Sequences

1. User enters command to checkout to a specific version / commit

2. System verifies that the version / commit exists

3. System uses the local git file, to reset the local files to the required version / commit

5.6.3 Functional Requirements

* The git file system must exist, in order to retrieve a version

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

REQ-1:

REQ-2:

**6. Other Nonfunctional Requirements**

**6.1 Performance Requirements**

1. Our code management system should be efficient enough to handle the traffic of concurrent usage of our product by millions of developers
2. Our code management should have enough servers to ensure adequate storage for resources for our users
3. Our product should have servers dedicated for backing up our users data to ensure fault tolerance and increased reliability
4. Our system user interface should be responsive enough for better user experience

**6.2 Safety Requirements**

1. User should be aware of any damage caused due to downloading code from repositories of other users that may contain viruses and hence should always ensure downloading source codes of trusted users
2. User should specify the system requirements for execution of his projects uploaded in repositories in readme.md file to ensure safe usage of the same by other users who wish to use it as a reference
3. User should preferably upload the changes made to his files to his repositories on regular intervals to avoid loss of data from git directory because of problems in the users local system

**6.3 Security Requirements**

1. Users should sign in and authenticate themselves before they can upload resources in our system.
2. Users can make their repository private for data security and integrity.
3. User should use antivirus software to scan files they wish to download from the repositories of other users to prevent malware attacks.

**6.4 Software Quality Attributes**

1. Our product should be adaptable enough to cater to the demands of our users from different operating systems.
2. Our servers should have enough available storage space for providing adequate storage to our users.
3. Our system should be correct enough to store the users resources in their desired directory.
4. Our product should be flexible enough to cater to the demands of users from different domains.
5. Our system should be usable enough for users who are very new to code management and version control system for better user experience.
6. Our system should be robust enough to recover from any kind of erroneous state through proper software protocols.
7. Our system should be reliable enough to handle any data loss with the help of our data backup servers for better user trust and satisfaction.
8. Our product should be tested thoroughly before launching to ensure proper quality of service to the customer and increased reliability and robustness.

**6.5 Business Rules**

1. The system must always have a git directory initialized, in order to use all other functions. The user can enforce this by using the ‘init’ command.
2. In order to use the product, there must be an existing account linked. If the user does not have an account, they can sign up for the same. The system will ensure that this gets authenticated

**7. Other Requirements**

**Appendix A: Glossary**

* **Git:** This is an existing popular version control software, which provides functionality, used to help in implementing all required operations mentioned in the SRS
* **Github:** This is an existing popular hosted environment, which allows to use Git functionality, and make your own repositories, which can be accessed from the web
* **Init:** This is a command provided by the Git tool, which helps to initialize a git repository, in the specified user directory. This is used in the local system of the user
* **Developer:** This is a term given to a user who is usually well acquainted with Software Skills and/or is from a Computer Science background. For our SRS, a developer user is the one who is using the Git tool functionalities, rather than just observing the repositories for personal reasons
* **Back End Server:** This defines the Hardware Resources, which are responsible for hosting the product, authenticating users and have databases integrated which store all files of the users. The back end server is the single most expensive component of the entire product
* **Antivirus:** This is a software used to defend a hardware from dangerous software(s)
* **HTTP:** Hyper Text Transmission Protocol, is a popular network protocol responsible for sending & retrieving web objects between multiple systems
* **FTP:** File Transfer Protocol, is a popular network protocol, which is responsible for transferring all types of files over the network