# Table of Contents

**OVERVIEW** ......................................................................................................................................4  
**TERMINOLOGY** ..............................................................................................................................5  
**GIT** ....................................................................................................................................................6  
  Installation of Git Standalone .....................................................................................................6  
**GIT LAB** ..........................................................................................................................................10  
  Browser Interface .....................................................................................................................10  
    Login ...................................................................................................................................10  
    Merge Requests .................................................................................................................11  
    Pipelines .............................................................................................................................11  
    Project Dashboard .............................................................................................................12  
    Issues .................................................................................................................................12  
    Wiki .....................................................................................................................................13  
    Groups ................................................................................................................................13  
    Continuous Integration .............................................................................................................13  
      Configuring .gitlab-ci.yml ...................................................................................................13  
**WORKFLOWS** ...............................................................................................................................15  
  Git Bash ....................................................................................................................................15  
    Create a local repository ....................................................................................................15  
    Connect local repository to remote ....................................................................................15  
    Clone a remote repository ................................................................................................17  
    Make repository the working directory ...........................................................................17  
    Stage and commit changes to a file ..................................................................................17  
    Check out refs from the remote repository ........................................................................18  
    Check in refs to the remote repository ............................................................................19  
    Check out a branch .............................................................................................................19  
    Resolve a merge conflict after a pull .................................................................................20  
  Visual Studio Team Explorer ...................................................................................................22  
    Open Team Explorer ..........................................................................................................22  
    Create a local repository ....................................................................................................22  
    Connect local repository to remote ....................................................................................23  
    Add an existing local repository ..........................................................................................25  
    Clone a remote repository ................................................................................................26  
    Stage and commit changes to a file ..................................................................................26  
    Check out refs from the remote repository ........................................................................30  
    Check in refs to the remote repository ............................................................................32  
    Check out a branch .............................................................................................................33  
    Resolve a merge conflict after a pull .................................................................................36  
  GitLab .......................................................................................................................................39  
    Copy remote repository URL .............................................................................................39
Groups................................................................................................................................39
Projects ....................................................................................................................................40
Files......................................................................................................................................41
Update ....................................................................................................................................43
Commits ...................................................................................................................................43
Branches ...................................................................................................................................43
Tags .......................................................................................................................................45
Issues .....................................................................................................................................45
Merge Requests .......................................................................................................................46
Pipelines .................................................................................................................................49
Wiki .......................................................................................................................................49
Project Settings ......................................................................................................................49
Overview

This document was written to familiarize the reader with utilizing GitLab and the Git version control system as tools in their software projects. This document assumes no prior knowledge in Git or any other version control system and will cover the following topics:

- Git Terminology
- Git Installation
- Basic workflows for:
  - Git Bash Commands
  - Git in the Visual Studio Team Explorer
  - GitLab Browser Interface
- Basic setup of GitLab-CI (Continuous Integration)

Users unfamiliar with Git and other version control systems may wish to review the Terminology section to become familiar with the vocabulary in this document. Users well versed in Git would be better off going straight to the GitLab and GitLab Workflow sections.
Terminology

For the purposes of this introduction, only the basic and frequently encountered terminology is covered. For a more complete and comprehensive glossary of the terminology below please refer to the Official Git Documentation.

- **Index** – A staging area for all changes made to files
- **Commit** – A set of changes created from the index, identified by a unique SHA hash value.
- **Branch** – An active line of development, used to simultaneously track different sets of commits. One branch is checked out (i.e. active) at any given time in a repository. New commits always become the new tip of the currently checkout out branch.
  - Example: Repositories initially start with a single branch (**master**), from which another branch can be created, worked on separately, and optionally have all its commits merged into master.
- **Ref** – A reference to a specific set of changes, can be represented in several ways (e.g. the head of a branch, a tag, or a commit itself).
- **Working Tree** – The files, as they appear on your file system, that are referenced by the currently checked out ref (usually the head of the branch unless manually set otherwise).
- **Check out** – An action that updates files in the working tree to reflect the ref (e.g. branch or commit) that you specify.
- **Head** – Reference to the tip (i.e. the latest commit) of the current checked out branch.
- **Merge** – Adds all the new refs from a source branch to a target branch.
  - Note: If both the source and target contain a ref that makes changes the same line(s) of code on a single file, a merge conflict resolution must be performed to decide which changes to keep in the target branch.
- **Remote Repository** – The primary copy of your repository, all refs here are accessible by collaborators.
- **Local Repository** – Your local copy of the repository, an exact copy of the remote repository in addition to refs that have not been pushed yet.
- **Push** – An action to submit new refs in your active branch to the corresponding branch in the remote repository. Checks to see if the remote head is an ancestor to your local head. If so, the new refs will be copied and your local head will become to new remote head (i.e. the remote branch you pushed to will match your local branch). Otherwise, the push will fail.
- **Pull** – Gets the latest refs from the remote repository and attempts to merge with local refs that the remote doesn’t have.
- **Fetch** – Gets the latest refs from the remote repository, doesn’t attempt to merge with local refs.
- **Clone** – Makes a local copy of a remote repository on your file system.
Git

There are a few different ways Git can be utilized:

- Git standalone (GUI & Bash interface)
- Third-party Git IDE tools (Visual Studio, Eclipse, & Others)

The best choice for any given developer depends on their preferred development environment. For example: a .NET developer who primarily works in Visual Studio might prefer to use Visual Studio’s Git tools that exist in the VS Team Explorer. It varies depending on the IDE in question, but Git IDE tools usually support the basic operations such as staging, committing, checking-in, and checking-out changes.

On the other side of the spectrum, one might prefer to use lightweight text editors (such as Notepad++). For those people, the Git standalone software will be needed. The Git standalone software can be utilized via a GUI or a command-line interface and supports all standard Git features. It can also be used to perform more complex operations on the repository that some IDE tools cannot.

Even if you do work in an IDE with integrated Git tools, it's often beneficial to have both. IDE Tools offer the convenience of reducing common operations down to a few clicks on the same interface used for your code editing. Running Git from the command line allows for more fine-grained control and advanced operations.

**Installation of Git Standalone**

To install the Git standalone software on Windows:

1. Download Git here: [https://git-scm.com/](https://git-scm.com/)
2. Run the downloaded installer (e.g. Git-2.12.2-64-bit.exe)
3. Make your preferred selection(s) for Bash/GUI support, then click **Next**.
   a. Note: Users comfortable with command-line interfaces may prefer Bash, otherwise Git GUI can be chosen, but only Git Bash is covered in this document. Both can be chosen as well.

![Select Components](image)

4. Choose the second option to make **git** commands available from the default Windows command-line prompt. Select **Next**.

![Adjusting your PATH environment](image)

5. Confirm **Use the OpenSSL Library** is selected, then click **Next**
6. Using **Checkout Windows-style, commit Unix-style line endings** is highly recommended for Windows users. Only choose other options if you understand the implications of each one. **Checkout as-is, commit as-is** is discouraged. Confirm your option then **Next**.

![Checkout Windows-style, commit Unix-style line endings](https://git-for-windows.github.io/)

7. Either terminal emulator can be chosen but **Use MinTTY** is recommended. Select **Next**.

![Use MinTTY](https://git-for-windows.github.io/)
8. Keep default selections, then click **Install**.

![Git 2.12.2 Setup](image)

9. Once the installation finishes, validate the installation by opening a command prompt and typing `git --version`, the output should appear as below:

![Command Prompt Output](image)

10. **Important**: Use the following commands to configure your name and email that Git will use to identify you in the commits you create:

```bash
git config --global user.name "Your Name"
git config --global user.email "your_email@company.com"
```

**Note**: Make sure the email you use is the same one associated with your GitLab account.
**GitLab**

GitLab is an open source platform built to collaborate on every aspect of the software development cycle: planning, coding, testing, deployment, and more. It offers a variety of tools such as version control, issue tracking, automated tests, automated deployments, and more that will be covered in this document.

As evident by the name, GitLab hosts Git repositories as a version control system (VCS). Git is a widely-adopted VCS across the software industry and a powerful tool for multiple developers working on the same code base.

**Browser Interface**

The GitLab browser interface contains several powerful tools for managing software projects. The interface allows many Git operations to be performed directly on the remote repository that would otherwise need to be done locally and pushed. GitLab also contains a powerful and customizable continuous integration (C.I.) system for performing automated builds, tests, and deployments.

**Login**

- Navigate to gitlab.epri.com and use your EPRI AD credentials to login
**Merge Requests**

Merge requests must be used to promote changes to protected branches. Depending on the configuration of the branch protection, developers may be allowed to push to it, approve merges on to it, both, or neither. For example: For projects in the SWS group, users with the Developer role working have permission to push and approve merges onto the DEV and TEST branches, but cannot push or approve merges onto the PROD branch. Merges requests to the PROD branch will always require approval by a user with the Master role (i.e. a member of the SQA team for the SWS group).

A merge request can be created by following these steps.

**Pipelines**

The Pipelines tab displays the status of automated tests, builds, and deployments. A pipeline is a series of jobs created from the .gitlab-ci.yml file in a ref (tag or branch) in the repository. Configuring the .gitlab-ci.yml file is covered in the continuous integration section.
**Project Dashboard**

For reference, this is the project dashboard from which most functions and settings for a project can be accessed by users with the proper permissions. The following sections will refer to the buttons from this screen as **tabs**. The following section is an overview of the project dashboard’s various tabs and their functions, for workflow tutorials on the browser interface [click here](#).

![Project Dashboard](image)

**Issues**

GitLab’s issue tracking features can be used to effectively coordinate a group of developers working on the same code. The main purpose of the issue tracker is to create issues, categorize them (e.g. feature, to-do, bug, completed), track their status, track time spent, and discuss them with collaborators in the issue comments. Issues can also be associated with project milestones.

GitLab also allows users to add references to issues, merge requests, and snippets in issues, merge request descriptions, and comments ([read more here](#)).

![Issue Details](image)

A – Issue key  
B – Issue title  
C – Label  
D – Number of comments
Wiki

The wiki is a space for developers to create lightweight pages using Markdown, Rdoc, or AsciiDoc. [GitLab’s documentation](https://docs.gitlab.com/ee/) covers their implementation of markdown, other resources exist for [Rdoc](https://github.com/rdoc) and [AsciiDoc](https://asciidoctor.org/about/).

Groups

<table>
<thead>
<tr>
<th>A</th>
<th>Group name</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Current user’s role within group</td>
</tr>
<tr>
<td>C</td>
<td>Group description</td>
</tr>
<tr>
<td>D</td>
<td>Number of projects in group</td>
</tr>
<tr>
<td>E</td>
<td>Number of members in group (doesn’t include sum of members with access to specific projects within group).</td>
</tr>
</tbody>
</table>

Continuous Integration

Continuous Integration (C.I.) in GitLab is configured by creating a file with the name `.gitlab-ci.yml` in the root level of a project repository. C.I. Pipelines will only be created if the `.gitlab-ci.yml` file exists. The basic principle behind the `.gitlab-ci.yml` file is to create a series of jobs that build, test, and deploy applications. These jobs are executed by a Runner that runs on the Build Server. These runners can be configured to have one of seven executors: Shell, Docker, Docker Machine and Docker Machine SSH, Parallels, VirtualBox, SSH, or Kubernetes. This documentation will cover usage of the Shell executor, please refer to the official GitLab documentation for more information on the other executors.

Configuring `.gitlab-ci.yml`

For developers who wish to get started learning how to configure a `.gitlab-ci.yml` file, start with the [GitLab Quick Start Guide](https://docs.gitlab.com/ee/) (skip the runner setup) and [Configuration of `.gitlab-ci.yml` Guide](https://docs.gitlab.com/ee/). There are build servers setup with active shared runners that can be utilized to conduct automated builds, tests, & deployments. Further documentation
on configuring .gitlab-ci.yml, as it pertains to EPRI procedures, will be created in the future. **For Subscriber Websites (SWS)** SQA will configure the .gitlab-ci.yml file and consult with developers to ensure that it meets the site’s requirements.

**Limitations of Build Servers**
As of March 2018, the build servers have restricted access to the external internet. This means that the source of any external download must be whitelisted before it can be accessed.
Workflows

Two types of Git workflows are described in this document: The standalone command-line interface and the Visual Studio Team Explorer.

Git Bash

Note: With the exception of `git clone` and `git init`, all the commands listed below must be performed with the folder containing your Git repository as the working directory in your command-line interface.

Create a local repository

- Commands:
  
  ```
  git init <your-repository-name>
  ```

Connect local repository to remote

As stated previously, GitLab manages a collection of Git repositories (one for each project). These repositories all have a URL of the format:

http://gitlab.epri.com/<user or group namespace>/<project name>.git

1. Obtain the remote repository URL by following the format above or by following these steps.
2. In a command-line or terminal, navigate to the repository directory.
3. Checkout the branch that you wish to push or pull to.
   
   ```
   git checkout <name-of-branch>
   ```

4. Replace `<remote-url>` with the URL from step 1 and replace `<remote-branch>` with the branch on the remote repository that you wish to push to; then execute the following commands:
   
   ```
   git remote add origin <remote-url>
   git push -u origin <remote-branch>
   ```

5. Subsequent pushes from the branch you checked-out in step 3 can be performed using the following simple push command:
   
   ```
   git push
   ```

  **Note:** If the remote repository isn’t empty you will need to pull before executing the push.
If the repository already has an remote named origin set, it can be updated with this command:

```
git remote set-url origin <remote-url>
```

A second remote can be set and used with these commands:

```
git remote add alternate_origin <remote-url>
git push -u alternate_origin master
```
Clone a remote repository

- Prerequisite:
  - Remote repository must exist and read access must be granted.
  - Remote repository URL must be obtained (steps).
- Commands:
  
  ```
  git clone <remote-url>
  cd <project-name>
  ```

Make repository the working directory

- Prerequisite:
  - An existing repository and its path
- Commands:
  
  ```
  cd <path-to-repository>
  ```

Alternative Steps (if Windows Explorer Integration > Git Bash Here was enabled during step 3 of installation):

1. Navigate to the repository in the Windows Explorer
2. Right click on any blank space in the explorer and select Git Bash Here
3. This will open a bash terminal that can be used for remaining Git operations and other command-line operations.

Stage and commit changes to a file

- Prerequisite:
  - A change must be made to a file after the head has been checked out
- Commands:
  
  ```
  git add filename.extension
  git commit -m "describe changes in the commit message here"
  ```
Check out refs from the remote repository

- Note: Always pull the latest refs frequently to reduce the possibility of merge conflicts in projects with multiple people working on the same file(s).
- Prerequisite:
  - Un-staged changes must be added to a commit or stashed (read more about git stash)
- Commands:

  git pull
Check in refs to the remote repository

- Prerequisites:
  - Local repository must contain all the refs from the remote (pull or fetch refs).
  - Must have permission to push to the remote branch that you currently have checked out.
- Commands:

  ```
  git push origin branch-name
  ```

Check out a branch

- Commands:

  ```
  git checkout -b branch_name
  ```
Resolve a merge conflict after a pull

- Scenario:
  - Two developers (Dev A and Dev B) have made modifications to line 258 on the file default.asp in their respective local branches.
  - Dev A pushed their changes to the remote branch TEST first with the following content in line 258:
    ```
    <h3 class="page_title">Welcome to the Red EPRI Software Development Website</h3>
    ```
  - Dev B committed the following code to their local branch TEST on line 258:
    ```
    <h3 class="page_title">Welcome to the Blue EPRI Software Development Website</h3>
    ```
  - When Dev B attempts to pull the remote TEST branch the pull command will attempt to merge it with the local TEST branch and encounter a conflict, this conflict must be resolved by the Dev B.

1. Steps:
   1. Run the `git pull` command after committing your changes, if there is a conflict you will get the following message:
      ```
      $ git pull
      remote: Counting objects: 3, done.
      remote: Compressing objects: 100% (3/3), done.
      remote: Total 3 (delta 2), reused 0 (delta 0)
      Unpacking objects: 100% (3/3), done.
      From http://SQA03.com:SQA/Software-Development-Website:origin/DEV
      Auto-merging default.asp
      CONFLICT (content): Merge conflict in default.asp
      Automatic merge failed; fix conflicts and then commit the result.
      ```
   2. The simplest approach to resolve the merge conflict is to simply open the file with the conflict and search for the following tags the Git inserts into
3. The code beneath the """" tag is your local change, the code beneath the """" is the remote change. You must select which one you would like to keep by deleting the unwanted changes and tags. In this example, we are deleting the tags and local change to keep the remote change on line 258:

```
<h3 class="page_title">Welcome to the Red EPRI Software Development Website</h3>
```

4. To finish resolving the conflict we must commit the change. Be sure to indicate that a conflict was resolved in the commit message:

```
$ git add default.asp
$ git commit -m "default.asp conflict resolved"
```

5. After the change has been committed it can be pushed as normal.
Visual Studio Team Explorer
The Visual Studio Team Explorer cannot perform all Git operations but it can certainly handle the most common and frequently used Git workflows and will be the only tool needed for most developers.

Open Team Explorer
1. In the bottom-right corner (for the default Visual Studio layout) beneath the Solution Explorer, click the Team Explorer tab to open the Team Explorer.

   ![Team Explorer Tab](image)

   Note: If the Team Explorer tab doesn’t show, open it by clicking View > Team Explorer

Create a local repository
1. Open the Team Explorer
2. Click the Manage Connections button
3. Click the New button
4. Enter a path for the repository (the last path element will be its name) and click Create
Connect local repository to remote

As stated previously, GitLab manages a collection of Git repositories (one for each project). These repositories all have a URL of the format:

http://gitlab.epri.com/<user or group namespace>/<project name>.git

1. Obtain the remote repository URL by following the format above or by copying it from GitLab (steps).
2. Add the existing local repository to the Visual Studio Team Explorer (steps).
3. Open the local repository and click Settings

![Team Explorer - Home](image)

4. Click Repository Settings

![Team Explorer - Settings](image)

5. Expand the Remotes section and click Add

![Remotes](image)
6. Enter **origin** for the **Name**
7. Enter the URL from step 1 for **Fetch** and ensure that **Push matches fetch** is checked

8. Click **Save**
Add an existing local repository

1. Open the Team Explorer
2. Click the Manage Connections button

3. Click the Add button

4. Enter the path to the local repository and click Add
Clone a remote repository

1. Prerequisite:
   1. Remote repository must exist and read access must be granted.

2. Steps:
   1. Open the Team Explorer and click the Manage Connections button at the top of the Team Explorer.
   2. Click on the Clone button.
   3. Enter your remote repository URL.
   4. After the repository has been cloned, it will appear in the Team Explorer's list of connections. Double-click the repository in the list of connections to view it.
5. From the repository level view in the Team Explorer you can perform many basic Git operations and open the Solution File if your project contains one.
Stage and commit changes to a file

- Prerequisite:
  - Make a change to a file

- Steps:
  1. Click on the **Changes** button in the Team Explorer
  2. Right-click on the file you wish to stage changes for and click **Stage**
3. Type a short commit message describing the change(s) made.

4. Click the **Commit Staged** button to create the commit.
5. At this point you can optionally sync your changes with the remote repository at this point.
   - **Note:** The Team Explorer sync operation is effectively a normal Git pull operation followed by a push operation pending the success of the pull.

   ![Team Explorer - Changes](image)

   **Check out refs from the remote repository**
   - **Steps:**
     1. The pull, push, fetch, and sync operations can be accessed from the Home page for the repository in Team Explorer. Click the Sync button to open the Synchronization menu.
     
     ![Team Explorer - Home](image)

     2. Click Pull to check for remote refs that the local repository may not have.
        - **Note:** Pulling (or fetching and merging manually) before a push is necessary to ensure that your changes can be merged successfully with the remote changes.
3. If there are no merge conflicts, the repository should now have the latest refs
Check in refs to the remote repository

- Steps:
  1. The pull, push, fetch, and sync operations can be accessed from the Home page for the repository in Team Explorer. Click the Sync button to open the Synchronization menu.

  ![Team Explorer Home Page](image1)

  ![Team Explorer Synchronization Menu](image2)

  2. Click Push to push your local refs to the remote repository.
3. The push should be successful if there are no missing remote refs. Be sure to pull if the push fails.

![Screenshot of Synchronization](image1)

**Check out a branch**

1. Steps:
   1. Navigate to the **Home** page of the repository and click the **Branches** button

![Screenshot of Team Explorer - Home](image2)
2. Your local branches will be displayed by default, if you wish to select a remote branch you must expand the remotes/origin node.

3. To check out either a local or remote branch, right-click the desired branch and select Checkout.

4. If you checked out a remote branch, it will now appear as a local branch.
Resolve a merge conflict after a pull

- Scenario:
  - Two developers (Dev A and Dev B) have made modifications to line 258 on the file `default.asp`.
  - Dev A pushed a change to the remote branch `TEST` first with the following content in line 258:
    ```html
    <h3 class="page_title">Welcome to the Red EPRI Software Development Website</h3>
    ```
  - Dev B committed the following code to his local branch `TEST` on line 258:
    ```html
    <h3 class="page_title">Welcome to the Blue EPRI Software Development Website</h3>
    ```
  - When Dev B attempts to pull the remote `TEST` branch the pull command will attempt to merge it with the local `TEST` branch and encounter a conflict, this conflict must be resolved by Dev B.

- Steps:
  - Run the `git pull` command after committing your changes, if there is a conflict you will see the following:
    ```
    Team Explorer - Synchronization
    Synchronization | Software-Development-Website
    
    Pull completed with conflicts in the 'Software-Development-Website' repository. Resolve the conflicts and commit the results.
    Branch: DEV
    
    Merge In Progress
    A merge operation is in progress in the 'Software-Development-Website' repository. Resolve the conflicts and commit the results.
    Conflicts: 1 | Abort
    
    Incoming Commits (1)
    Fetch | Pull
    Update default.asp
    Administrator 3 minutes ago
    
    Outgoing Commits (1)
    Push
    Change site welcome to blue
    Sean Vail 2 minutes ago
    ```
  - Click the Resolve the conflicts link and then click on the file that wish to resolve:
From here you can choose to keep your local change (Keep Local), take the remote change (Take Remote), or Compare the files (Compare Files) if you need more information to resolve the conflict. The Diff links will show you the individual change made in local or remote.

In this example, we will keep the local change by clicking Keep Local.

Start creating a commit for the merge by clicking Commit Merge.
Enter a commit message and click **Commit All**. In your commit message, be sure to indicate that a conflict was resolved.

After the change has been committed it can be pushed as normal.
**GitLab**

**Copy remote repository URL**
1. Login to GitLab
2. Select the project with the needed repository
3. Click the **Copy URL to clipboard** button

![GitLab project screen](image)

**Groups**

**View groups**
1. Click on the hamburger button in the upper left hand corner of the page
2. Click **Groups**

**Create a group**
1. Click the **New Group** button
2. Enter a **Group path** (your group name)
3. Enter a **Description** (optional)
4. Upload a **Group avatar** image (optional)
5. Set a **Visibility Level**
6. Click **Create group**
Projects

Viewing your projects

- After logging in, you will be taken to the projects page. Projects that you have access to will appear here.

Creating a new project

**Note:** For subscriber websites, please follow the [SQA SWS Requirements](#). SQA will create the project in GitLab for you as part of the SWS process.

1. New projects can be created via the **New Project** button (upper right)

![New Project Button](image)

2. **Project path** will determine which user or group your project belongs to.
   - **Note:** When a project is owned by a group, it will inherit certain security settings by default unless overridden in the specific project’s settings.

![Project Path](image)

3. **Project name** is self-explanatory; however, it can contain only letters, digits, '_', '-', ' ' and '.'. Cannot start with '-', end in '.git' or end in '.atom'
   - **Note:** This is due to the fact that the project name determines part of the project and Git repository URL.
4. **Import project from** options allow the user to import existing Git repositories from other services/software.

5. **Visibility Level** determines who can see your project.

6. After the project is created, follow the **New Project Workflow** to set it up.

**Files**

**View files from specific ref**
1. Click on the **Repository > Files** tab to view a list of files on the default branch.
2. Find the project refs drop-down menu (upper left) and select the desired ref (branch or tag).
3. For branches, any CRUD (create, read, update, delete) operations performed after changing the ref will create a commit on the selected branch. Only read operations can be performed after selecting a tag.

**Create repository files**
1. Click the + button at the top of the **Repository > Files** tab to open the **New File** screen.
2. Enter a Filename.
3. Enter content via the text editor (optional).
4. Enter a commit message (optional).
5. Click **Commit Changes** at the bottom of the screen.
Read repository files
1. Click on the desired file from within the list of files on the Files tab to open a read-only view of the file.
Update repository files
1. Open a file to read
2. Click the Edit button
3. Make changes in the text editor
4. Enter a commit message (optional)
5. Click Commit Changes at the bottom of the screen

Delete repository files
1. Open a file to read
2. Click the Delete button
3. Enter a commit message (optional)
4. Click Delete file to confirm

Commits
View commit history
1. Click the Repository > Commits tab to view a list of commits.

Branches
View branches
1. Click the Repository > Branches tab to view a list of branches.

Create new branch
1. Click the New Branch button from within the Branches tab (upper right)
2. Give the new branch a name
3. Select an existing branch from which the new branch will be created
4. Click Create branch

Protect a branch
1. Click on the Settings > Repository tab
2. Scroll down to Protected Branches
3. In the Protect a branch menu enter the specific branch name for the Branch field (or enter a wild card for multiple branches)
4. Set the desired permissions for Allowed to merge and Allow to push (reference GitLab Roles)
**Unprotect a branch**
1. Click on the **Settings > Repository** tab
2. Scroll down to **Protected Branches**
3. Click **Unprotect** next to the desired branch

**Set default branch**
1. Click on the **Settings > General** tab
2. Use the **Default Branch** dropdown menu to select the default branch
3. Scroll down to the **Save changes** button and click it
Delete a branch
1. Click on the Repository > Branches tab
2. Select the Delete branch button next to the branch to be deleted
   a. Note: Branch must not be the default branch nor protected in order for deletion to be permitted

Tags
Create a new tag
1. Click on the Repository > Tags tab
2. Click the New tag button
3. Enter your tag name (e.g. v1.0)
4. Select a branch or commit SHA for the tag to reference
5. Enter a tag message (optional)
6. Enter release notes (optional)
7. Click Create tag

Delete a tag
1. Click on the Repository > Tags tab
2. Click the Delete tag button next to the desired tag

Issues
Create issue
1. Click on the Issues > List tab
2. Click the New Issue button (upper right)
3. Enter a short summary of the issue in the Title field
4. Enter a description in the Description field
5. Make someone or yourself the Assignee (optional)
6. Assign a Milestone (optional)
7. Assign a Label (optional, but recommended)
8. Assign a Weight (optional)
9. Set a Due date (optional)
10. Click Submit issue

Create label
1. Click on the Issues > Labels tab
2. Click the New label button (upper right)
3. Enter a Title
4. Enter a Description
5. Select a hexadecimal RGB Background color value
6. Click Create Label

**Create milestone**
1. Click on the Issues > Milestones tab
2. Click the New Milestone button (upper right)
3. Enter a Title
4. Enter a Description
5. Set a Start Date & End Date (optional)

**Close issue**
1. Click on the Issues > List tab
2. Enable the checkbox next to the issue(s) you would like to close
3. Click the Status dropdown menu (top of the list) and select Closed
4. Click Update issues

**Auto-close issue**
Issues can be auto-closed by putting specific phrases into commit messages. When a commit message with a valid phrase is pushed to the remote repository GitLab will automatically close the issue (read more here).

Example:

7. Add the phrase “Fix #20” to a commit message to close issue #20

**Merge Requests**

**Create a merge request**
1. Click on the Merge Requests tab
2. Click on the New Merge Request button (upper right)
3. Select the Source branch and Target branch
4. Click Compare branches and continue
5. Enter summary of the request into the Title field (e.g. Merging DEV into TEST)
6. Enter a Description (optional)
7. Select an Assignee (optional)
8. Select a Milestone (optional)
9. Select a Label (optional)
10. Click **Submit merge request**
11. Have a user, with permission to merge onto the target branch, accept the request.
   (See the [accept a merge request](#) section for more details).
Accept a merge request
To approve a merge request, the user’s role must have permission to merge into the target branch (reference permissions).

1. Click on the Merge Requests tab
2. Click on the title of the merge request to be approved

<table>
<thead>
<tr>
<th>Merge DEV into TEST</th>
</tr>
</thead>
<tbody>
<tr>
<td>!24 · opens less than a minute ago by Administrator</td>
</tr>
</tbody>
</table>

3. Click Accept Merge Request
4. After a request is accepted there will be options to Revert the merge or Cherry-pick changes from the source branch

Resolve merge conflicts
When merging two branches (e.g. Branch A and Branch B): if Branch A and Branch B both contain different commits with changes to the same line(s) of code; a conflict will occur and it must be resolved by deciding which changes to keep. GitLab provides an easy and intuitive interface to resolve these conflicts when they occur.

In the scenario below, a different change has been made to the same line of code in both the target and source branch. When the request is created GitLab will detect the conflict and it will need to be resolved.

1. Create the merge request
2. Click the Resolve conflicts button

<table>
<thead>
<tr>
<th>This merge request contains merge conflicts</th>
</tr>
</thead>
<tbody>
<tr>
<td>To merge this request, resolve these conflicts or ask someone with write access to this repository to merge it locally</td>
</tr>
<tr>
<td>Resolve conflicts</td>
</tr>
</tbody>
</table>

3. For each conflict, choose either to keep the code from the source branch (Use ours) or keep the code from the target branch (Use theirs)

4. When done, click Commit conflict resolution.
5. Have a user, with permission to merge onto the target branch, accept the request. (See the accept a merge request section for more details).

Pipelines

Create a pipeline
1. Click on the Pipelines > Pipelines tab
2. Click the Run pipeline button (upper right)
3. Enter the name of a branch or tag
   a. Note: The jobs will be created from the .gitlab-ci.yml file in the commit referenced by the branch or tag.
4. The pipeline will start with a pending status, then switch to running once a runner becomes available. The pipeline will end with passed, failed, or canceled. A build fails whenever a non-zero error code is returned.

See job output log
1. Click on the Pipelines > Pipelines tab
2. Find the job you wish to view in the Stages column in the table of Pipelines, click on it to expand, and click once more.
   a. Note: You can open the build log at any point after the job is created (i.e. even when it is running) and the log will update automatically as the job executes.

Wiki

Create a wiki page
1. Click on the Wiki tab
2. If you haven’t created a wiki page yet, you will be taken to the new page form at this point. Otherwise click the New Page button.

Project Settings

Members
Add a user to project
1. Click on the Settings > Members tab
2. Enter the user’s name or username
3. Choose the user’s role
4. Choose an expiration date if applicable (optional)
5. Click **Add to project**

**Remove a user from project**
1. Click on the **Settings > Members** tab
2. Scroll down to **Members with access to <project name>**
3. Find the user you wish to move and click the **Remove user from project** trash icon
4. Click **Ok** when prompted