

Moving DITA content to GitLab

Table of Contents

Benefits of Git and GitLab	1
Comparing Git with Subversion	1
Required Software	3
Configuring Git and setting up SSH keys for GitLab (one-time setup)	3
Using Git for Windows	4
Getting the repo locally (one-time setup)	4
Committing changes (daily operation)	4
Using Git Bash	5
Making minor changes	5
Branching and merging	5
Using the Git plug-in for Oxygen (optional)	6
Installing the Git plug-in for Oxygen (one-time setup)	6
Using the Git plug-in for Oxygen (daily operation)	6
Appendix	7
Basic Git commands	7

This document is for people who have prior knowledge of Subversion and are planning to migrate from Subversion to Git.

Benefits of Git and GitLab

Git is an open source distributed version control system for tracking changes in files. Git can be used on a local machine, without a cloud service. Most people use a Git client locally in conjunction with a Git cloud service such as GitLab or GitHub to benefit from cloud services for collaboration, auto backup, and easy access.

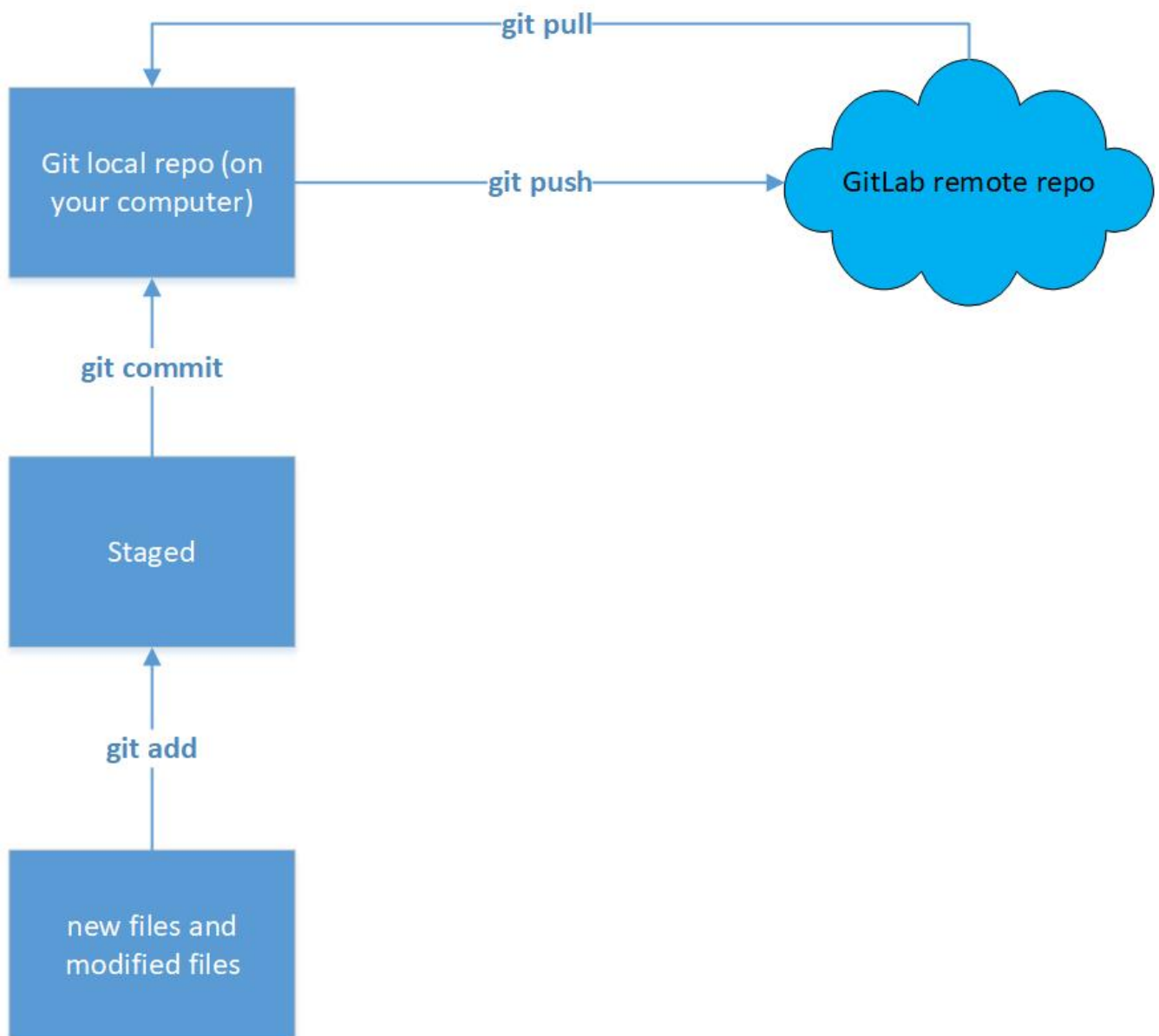
Algo is moving all software source code to SSC GitLab. The Algo TechDocs team has decided that having TechDocs in GitLab is appropriate for moving towards a docs-as-code approach. GitLab CI/CD pipelines offer excellent capacity to bring code and docs together.

Comparing Git with Subversion

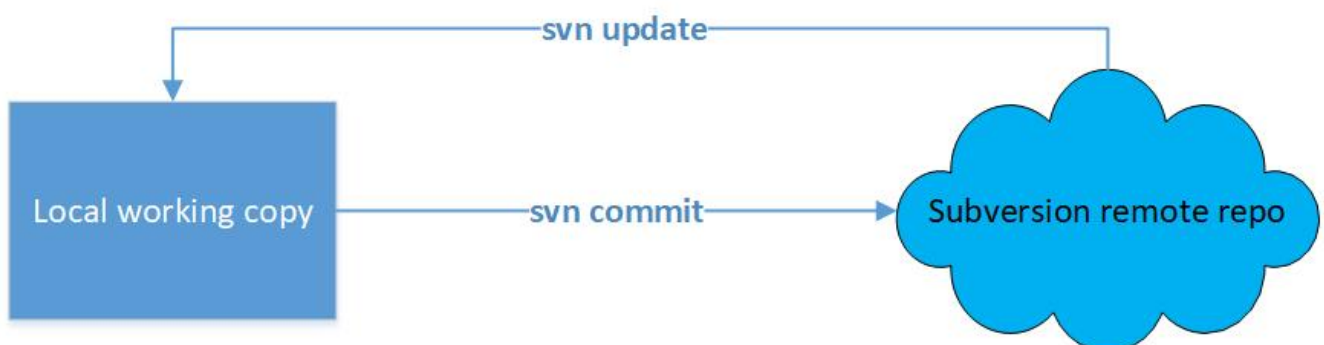
Git is similar to Subversion in many ways. They both provide typical version control features such as branching, resolving conflicts, and reverting changes.

One major difference is Git has the concept of a local repository, which is an enhanced version of Subversion's local working copy. When adding new files or making changes to existing files, you first stage the changes, then commit the changes to the local repository. This means you can add

new files and commit changes to your local repo without communicating with GitLab. After you are satisfied with the changes you have committed locally, you can then push them to GitLab. This diagram illustrates the basic workflow of using Git to submit changes to GitLab.



In comparison, Subversion has a local working copy and does not have the concept of a local repository. When you perform operations such as add, commit, or diff, your local working copy always needs to communicate with the Subversion remote repo.



Required Software

- A web browser for accessing the GitLab web GUI for SSH key configuration (one-time setup).
- A Git client for your operating system. Many free Git clients are available on the Internet. Git clients come in two flavors: as an independent application or as a plug-in to an existing application.

NOTE

This document focuses on using Windows and describes using Git for Windows and Git plug-in for Oxygen. Other Git clients work similarly.

TIP

Git plug-in for Oxygen is optional. The benefit of the Git plug-in for Oxygen is that you can stay in Oxygen and still get the latest files from GitLab or commit changes to GitLab.

CAUTION

Technically you can skip installing a Git client locally and just rely on the GitLab GUI in a browser for making edits and adding new files. However, we do not recommend this method. The GitLab GUI does not validate DITA files, which makes this method error prone.

Configuring Git and setting up SSH keys for GitLab (one-time setup)

After you have installed a Git client such as Git for Windows, you need to perform two initial setup tasks:

1. Configure Git locally.
2. Set up SSH keys for accessing GitLab.

To confirm your Git client is properly installed, start Windows PowerShell and type `git --version`.

To configure Git, start Windows PowerShell and type the following commands:

```
git config --global user.name <your name>
git config --global user.email <your email>
```

Typically GitLab repositories are accessible using either HTTPS or SSH. However SSC GitLab repositories are configured to allow SSH only. To set up SSH for Gitlab:

1. Start Windows PowerShell and type `ssh-keygen` without any parameters. This command generates the SSH public/private key pair. Follow the prompt to save the key pair, which typically are named `id_rsa` and `id_rsa.pub`. The `.pub` extension indicates a public key. Pay attention to where the key pair is saved. You need to post the content of the public SSH key to your GitLab profile.
2. Locate your public SSH key file with the `.pub` extension. Open it in a text editor and copy the full content. For example,

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQEL17Ufacg8cDh1QMS5NhV8z3GHZdhCrZbl4gz you@sscinc.com
```

3. Open a browser and log in to your GitLab account. Paste your SSH public key from step 2 to your Gitlab profile (<https://xxxxxx.com/profile/keys>).

Using Git for Windows

Git for Windows comes with a Git Bash and a Git GUI. The instructions mainly focus on using the Git GUI.

Getting the repo locally (one-time setup)

First you need to get a copy of the project from the GitLab remote repo to your local machine.

1. Open a browser and go to your GitLab project URL. Click **Clone**. Under **Clone with SSH**, click **Copy URL**.
2. Open a Windows command prompt. Navigate to the directory where you intend to put the repo. Type `git clone` and then right-click and paste. Your command should resemble the following:

```
git clone git@xxxxxxxx:xxxx/<project>.git
```

The project is copied to a local directory with the same name and is now ready for edits.

Committing changes (daily operation)

Locate the directory and make changes to the DITA files using Oxygen as usual. When you are ready for committing changes, locate the directory in Windows File Explorer, right-click and choose **Git GUI here**. Follow these steps to commit changes to the GitLab remote repo using Git GUI:

1. Get the latest version of the files from the remote repo by clicking **Remote** → **Fetch from origin** and then **Merge** → **Local Merge**.
2. Click **Rescan** to make sure all the changes you made are picked up by Git GUI.
3. Click **Stage Changed** to stage your changes to your local Git repo. Staging is a Git concept and not available in Subversion. It means getting the changes or new files ready to commit. If you have added new files, you will be asked whether to stage untracked changes. Click **Yes** to stage the new files.
4. Click **Commit**. This command commits your staged changes to your local Git repo. Write a brief commit message to describe what has been changed. For example, enter the ticket number related to this change.
5. Click **Push** to upload your changes to the GitLab remote repo. This Push command reuses the same message you put in step 4.

TIP

In step 1, instead of using two commands **Remote** → **Fetch from origin** and **Merge** → **Local Merge**, you can use just one command `git pull` to get the latest from remote. The Git GUI does not have a `git pull` command out of the box. You can add the `git pull` command to Git GUI.

a. In the Git GUI, click **Tools** → **Add**.

b. Type `git pull` in both the Name and Command fields. Click the **Add globally** check box and then click **Add**.

The `git pull` command is now added under **Tools** → `git pull`.

► [GitTutorial.mp4](#) (video)

Using Git Bash

Making minor changes

Open the Git Bash, navigate to your local Git repo master branch and perform the Git operations below:

1. `git pull` (for getting the latest from the remote GitLab repo).
2. Make changes to the files and then use `git status` to check whether there are new files or changed files that need to be committed to the local repo.
3. `git add .` (for staging untracked files or modified files in preparation to be committed to the local repo).
4. `git commit -m "<your message>"` (for committing the new files or changed files to the local repo).
5. `git push` (for pushing the changes to the remote GitLab repo).

Branching and merging

Git encourages the use of branches. The branching and merging method is applicable if your edits are substantial and you want to work on a separate branch. With this method, a branch is local on your machine. You do not push it to the remote server.

1. Run `git checkout master`
2. Run `git pull` to get latest from the master branch (HEAD).
3. Run `git checkout -b <branch_name>` to create a local branch called `<branch_name>` and switch to that branch.
4. Make your changes.
5. Run a `git status` to tell you what files have changed or are out of sync.
6. On a frequent basis, run `git add .` to add new files and `git commit -a -m <message>` to save your work (and so that your changes are staged locally).
7. Run `git checkout master` to switch to the master.
8. Run `git pull --rebase` to get the latest from the remote.

9. Run `git merge <branch_name>` to merge the branch changes into master, or `git merge --squash <branch_name>` if you have multiple commits on the <branch_name>.
10. Run `git push` to push the changes from your local `master` to the remote server.

Using the Git plug-in for Oxygen (optional)

Oxygen Author offers a Git plug-in for an integrated experience.

Installing the Git plug-in for Oxygen (one-time setup)

In Oxygen Author:

1. Go to **Help** → **Install new add-ons**.
2. Enter <https://www.oxygenxml.com/InstData/Addons/default/updateSite.xml> in the **Show add-ons from** field.
3. Select **Git Client** and click **Next**.
4. Select **I accept all terms of the end user license agreement** and click **Finish**.
5. Restart Oxygen. The Git Staging view is displayed. This view provides support for basic Git operations such as pulling content from the remote repo, committing changes to the local repo and pushing changes to GitLab.

Switch to a browser:

6. Go to your Gitlab URL.
7. Click **Clone** and next to the *Clone with SSH* URL, click **Copy URL**.

NOTE

This step assumes that you have previously generated an SSH key pair and added the public key to your Gitlab profile at <https://xxxxxxxxx/profile/keys>.

Switch back to Oxygen Author:

8. Under Git Staging, click + (Clone new repository).
9. Paste the Clone with SSH URL that you copied into the **Repository URL** field.
10. Set **Checkout branch** to `master`.
11. In **Destination path**, create a folder on your local machine. This folder is where the files are placed on your machine.

Using the Git plug-in for Oxygen (daily operation)

These are the typical steps to commit changes to the remote Git repo using the Git plug-in for Oxygen:

1. Click the Down Arrow button (**Pull Merge from origin**) to get the latest from GitLab remote repo. The number next to the Down Arrow button indicates how many commits your local repo

is behind the remote repo.

2. Open the files in Oxygen and make changes as needed.
3. Click **Stage Selected** or **Stage All** to stage your changes to the local repo.
4. Type a commit message and then click **Commit** to commit to the local repo.
5. Click the Up Arrow to push to origin(master). This step uploads your local changes to GitLab.

TIP

In step 5, if you get a message stating you cannot push because your repo is behind, then click the Down Arrow button again to pull merge from origin before pushing.

► [GitpluginOxygen.mp4](#) (video)

Appendix

Basic Git commands

This table summarizes the basic command line operations in Subversion and their Git counterparts.

Subversion command	Git command
svn checkout	git clone
svn update	git pull
svn update	<ol style="list-style-type: none">1. git fetch2. git merge
svn status	git status
svn commit	<ol style="list-style-type: none">1. git add2. git commit3. git push
svn diff	git diff