

# Studio Pro 9.0.4

## BYO Git User Guide



# Table of Contents

<b>BYO Git Requirements .....</b>	<b>3</b>
Supported Authentication Mechanisms .....	3
<b>Supported Git Service Providers .....</b>	<b>3</b>
Azure Repos and Azure DevOps Server .....	3
GitHub .....	3
GitLab .....	4
BitBucket .....	4
AWS CodeCommit .....	4
<b>Steps to Set Up Environment .....</b>	<b>4</b>
Preparing Your Repository .....	5
Preparing Studio Pro .....	5
Creating an App.....	7
Downloading from a Private Repository .....	7
Opening the Existing Git App .....	8
Uploading to the Private Git Server .....	10
“Converting” a Subversion App to Git.....	12
<b>Day-to-Day Use .....</b>	<b>14</b>
Committing .....	14
Updating .....	16
Pushing to the Remote Server .....	20
<b>Working with Branches .....</b>	<b>21</b>
Creating a New Branch Line.....	21
Switching Between Existing Branch Lines.....	23
Deleting an Existing Branch Line.....	23
Merging into Main Branch Line .....	24
<b>Known Issues and Troubleshooting.....</b>	<b>25</b>
Known Issues .....	25
Troubleshooting.....	26

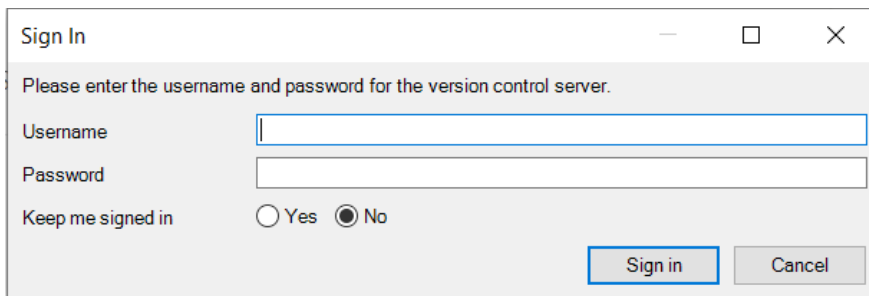
# BYO Git Requirements

Preliminary Git support in Mendix Studio Pro is limited to specific Git service providers and specific authentication mechanisms. We currently only support Git Private Server.

## Supported Authentication Mechanisms

Currently, we only support HTTP Basic authentication for Git service providers. With most providers this means the use of Personal Access Tokens (PATs).

To use PAT (or equivalent), you need to specify it in the **Password** field when Studio Pro requests to enter credentials for the version control server:



See below for instructions on how to set up Personal Access Tokens or equivalent for the supported providers.

## Supported Git Service Providers

### Azure Repos and Azure DevOps Server

We support both Microsoft's [Azure Repos](#) (Azure DevOps Services) hosted Git service, as well as Azure DevOps Server(former Team Foundation Server) which is an on-premises solution for hosting your Git repos on the private infrastructure.

To get a PAT for your user account, see the [Use personal access tokens](#) instructions in the Microsoft documentation.

You would need `Code (full)` permission for your token.

### GitHub

We support GitHub's hosting solutions, including the free GitHub.com cloud-hosted service and GitHub Enterprise, both hosted (Enterprise Cloud) and on-premises (Enterprise Server).

To get a PAT for your user account, see the [Creating a personal access token](#) instructions in the GitHub

documentation.

You would need `repo` permissions for your token.

## GitLab

We support all tiers of GitLab's service, including the free GitLab.com, GitLab Community Edition and GitLab Enterprise Edition.

To get a PAT for your user account, see the [Personal access tokens](#) instructions in the GitLab documentation.

You would need `write_repository` permission for the token.

## BitBucket

We support all tiers of Atlassian's BitBucket service, including the free BitBucket.org, BitBucket Server and BitBucket Data Center on-premises solutions.

On BitBucket.org, the Personal Access Tokens are called App Passwords.

To setup an App Password for your BitBucket.org account, see the [App passwords](#) instructions.

BitBucket Server and BitBucket Data Center, on the other hand, still use the term Personal Access Tokens. To set up a personal access token, see [Personal access tokens](#) instructions.

In both cases you would need `repository write` permission.

## AWS CodeCommit

We have a known compatibility issue with AWS CodeCommit in Git Technology Preview for Studio Pro. We are working to resolve the issue in the next release.

# Steps to Set Up Environment

This section describes all the initial steps that are necessary to use Studio Pro to manage a Git versioned app. To do so, you need a Git server and an initially unversioned Mendix app.

## Preparing Your Repository

Studio Pro is able to use the following Git server providers:

- Azure DevOps Server
- Azure Repos
- GitHub.com
- GitHub Enterprise Edition
- GitLab.com
- GitLab EE
- GitLab CE
- BitBucket.org
- BitBucket Server
- BitBucket Data Center

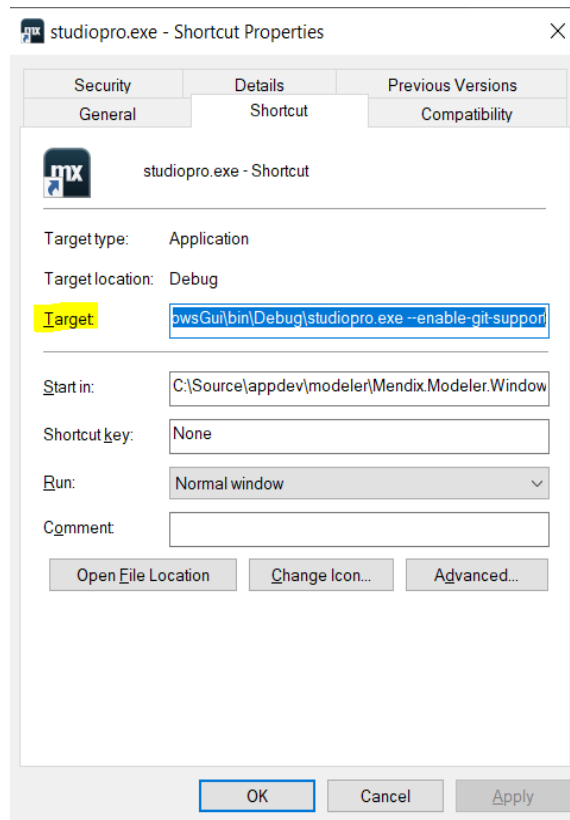
You need to create a private repository in your provider of choice and create a Personal Access Token (PAT) in order to provide access to it. The PAT is used as the password to connect to it. To interact with this private repository from Studio Pro, you need the link to the repository and a PAT.

For more information, see the [Supported Git Service Providers](#) section above.

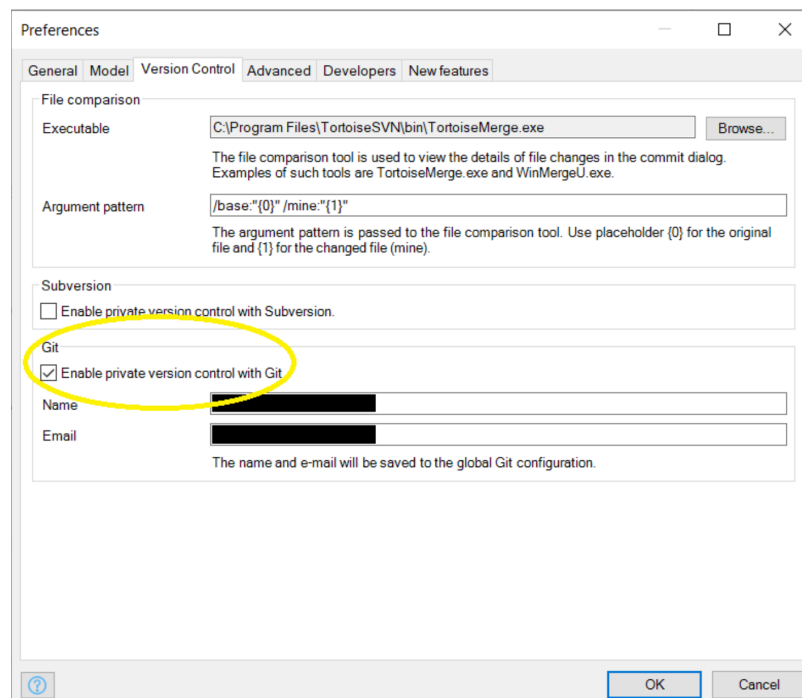
## Preparing Studio Pro

To use Git, you need to start Studio Pro with a specific feature flag. Navigate to the directory where your installation of Studio Pro is located, specifically where the **studiopro.exe** file is. There are two ways to enable the feature flag. Do one of the following:

- Open the command line at that location (or navigate to it from the command line) and type the following command: `studiopro.exe --enable-git-support`.
- Right-click the file, choose **Create shortcut**, and do the following:
  - If you are asked to save it on the Desktop, click **Yes**.
  - Once the shortcut is created, right-click it and select **Properties**.
  - Put your cursor all the way to the end of the **Target** field and enter the feature flag `--enable-git-support` after a space.
  - Click **OK**.
  - Start Studio Pro by double-clicking the shortcut. Make sure the **Target** option has Git enabled:



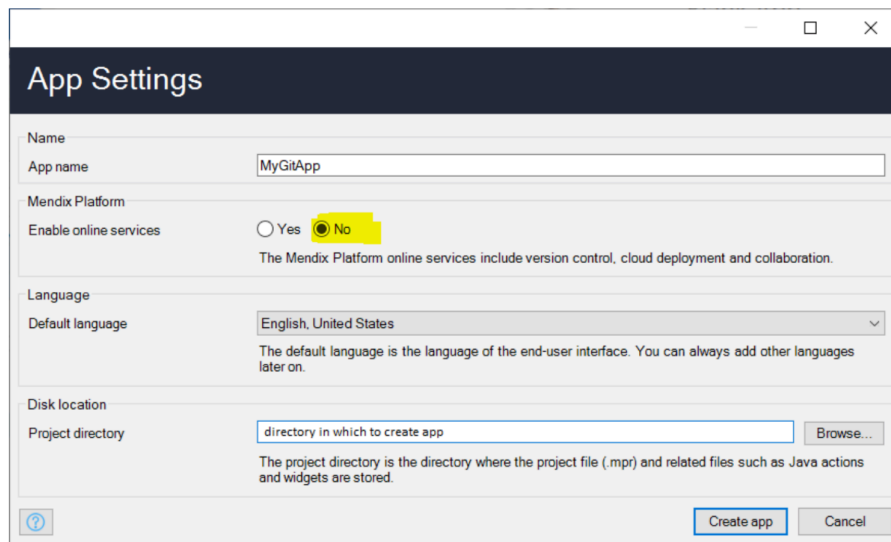
- Once Studio Pro is opened, go to **Edit > Preferences > Version Control** and make sure to activate private version control for Git. The name and email values will be used to identify your commit:



Studio Pro now has the ability to support Git.

## Creating an App

Create an unversioned app in Studio Pro.



Once the app is created, you can now upload it to your private Git repository, using **Version Control > Upload to Version Control Server**. To upload it, see the steps in the [Uploading to the Private Git Server](#) section below.

Once the app is uploaded, it will be a version-controlled app under Git.

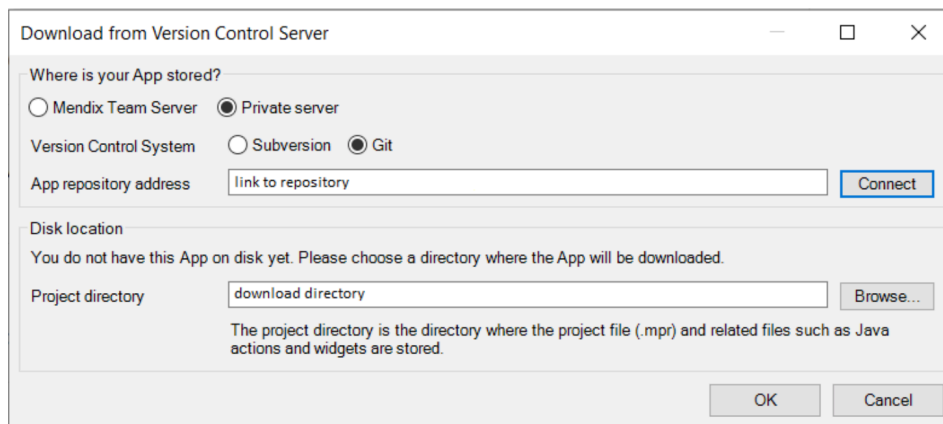
## Downloading from a Private Repository

Now that you have a Git app on your server, you can download it to another directory or one of your team members can download it on their machine. Follow the steps below:

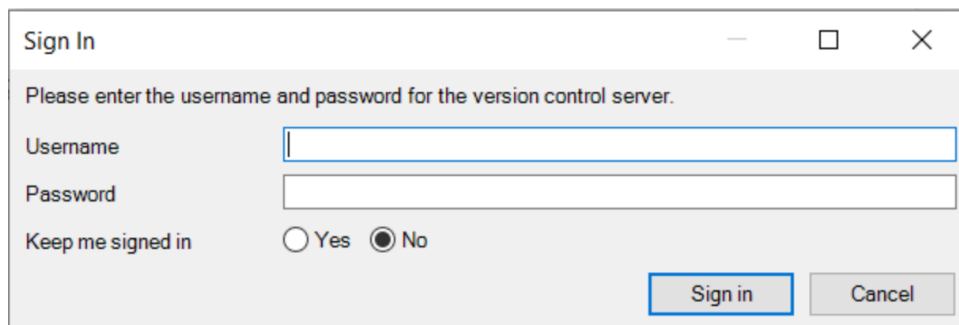
1. Under **Version Control > Download from Version Control Server**, select the **Private server** option and enter the URL of your repository which contains the app you want to download. If you are not sure which URL this is, you can find this info in your Git server.
2. If you have *both* Subversion and Git enabled for private version control in your preferences in Studio Pro, specify which version control system your repository uses. For this case, check the **Git** radio button.



3. Enter the link to the repository in the **App repository address** and click **Connect**. Now you will have the option to change the directory where the app is downloaded into.



4. Click **OK**.
5. In the **Sign In** dialog box, enter your credentials.



6. Enter username (can be anything except empty) and use the PAT you saved earlier as the password.

The app is downloaded and ready to be used with version control.

## Opening the Existing Git App

There are a few ways to open a Git Mendix app, as long as you have Studio Pro started up with the git flag mentioned in the [Preparing Studio Pro](#) section.



## Recent Apps List

In your **Recent Apps** list, you can simply click the app name and it will open it.

## Recent Projects Menu

Under **File > Recent Projects** you can select the app and open it.

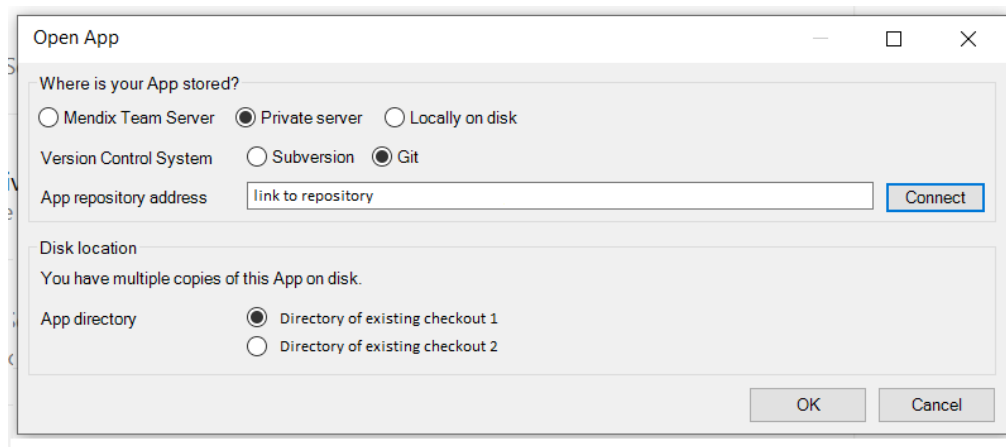
## Open App Form

The **Open App** form is accessible from two different places:

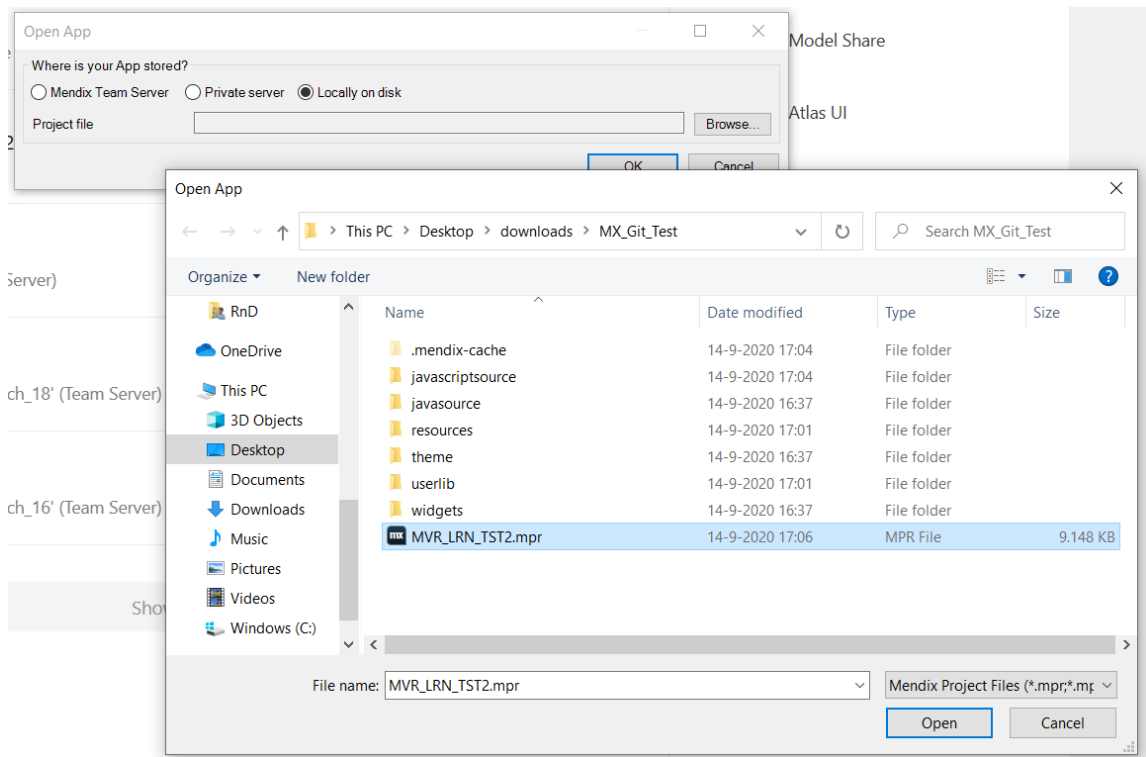
- **Open App** button on **My Apps** tab
- Under menu **File > Open Project**

In the form, there are two ways to open a Git app:

- Open from previous checkout and do the following:
  - Select the **Private server** radio button and enter the link to the repository.
  - Click **Connect**.
  - If you have already checked out the app at least once, you can then pick one of the existing locations on disk and the app on that location will open (shown here with two previous checkouts).



- You can open it locally on disk by doing the following:
  - Select the **Locally on disk** option.
  - In a file browser dialog box, browse to the directory containing your app and double-click the **.mpr** file (or select it and click **Open**).



The app is now open in Studio Pro.

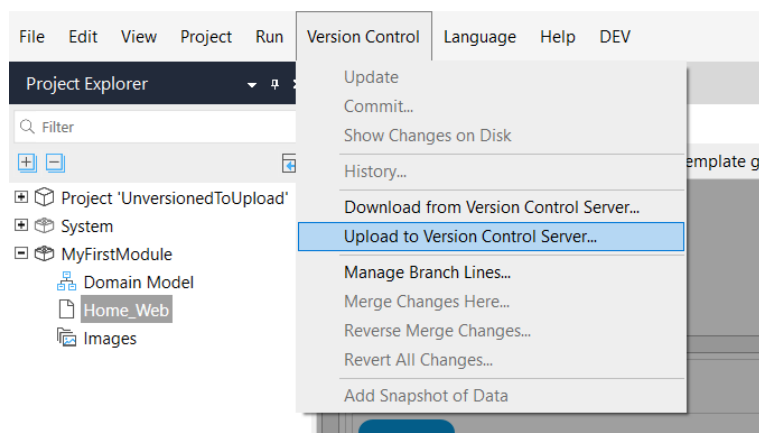
## Uploading to the Private Git Server

Once you have an unversioned app, you can upload it to your private team server.

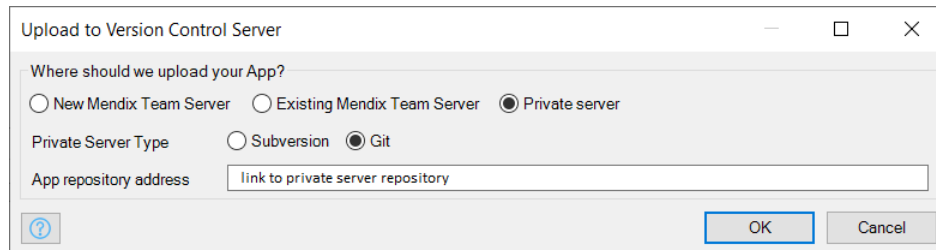
**Note:** The repository has to be completely empty (including **README.md** and **.gitignore** files), or the upload will fail.

To upload your app, do the following:

1. Open the app in Studio Pro and go to **Version Control > Upload to Version Control Server**.

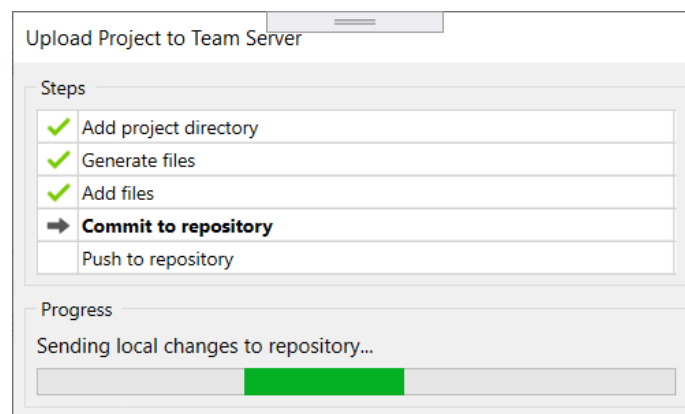


2. In the **Upload to Version Control Server** dialog box, select **Private server**.
3. Select **Git** as the private server type (if you have both **Subversion** and **Git** enabled in the **Preferences Form**).
4. Enter the link to the repository you want to upload this app to and click **OK**.

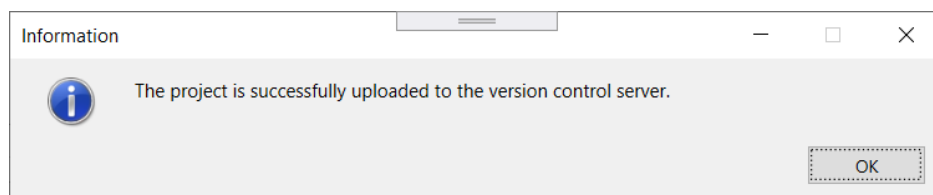


It might ask you to sign into the server, depending whether you have previously signed in and choose to stay logged in.

5. You can see the upload process in the **Upload Project to Team Server** pop-up window:



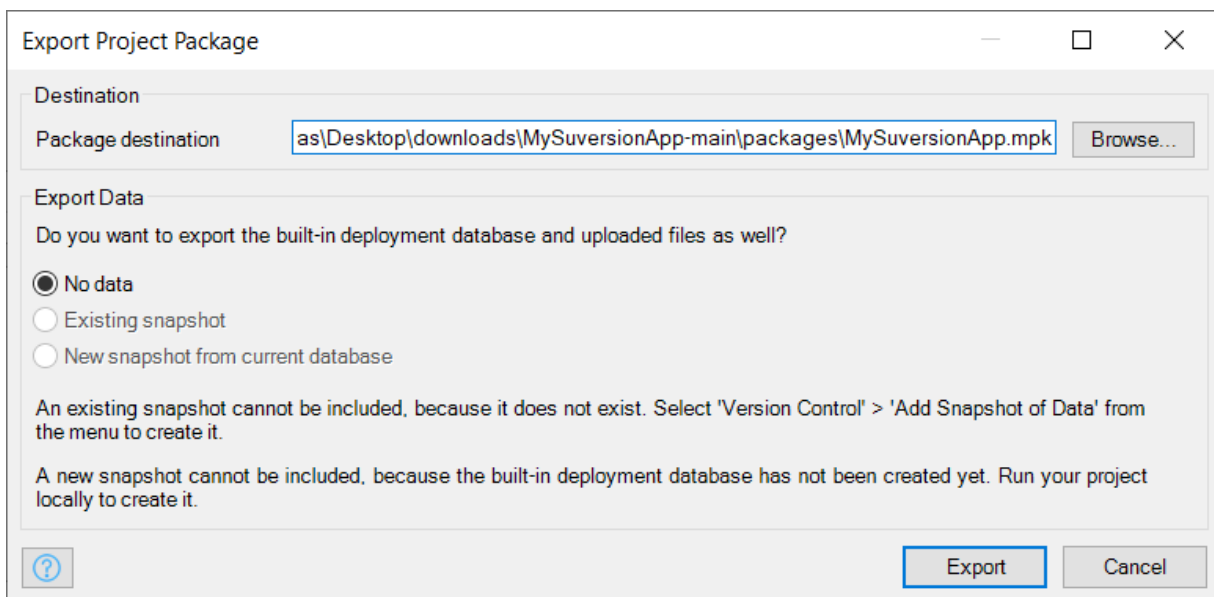
The project is uploaded successfully. You can check on your private server and see that the app is now on the desired repository.



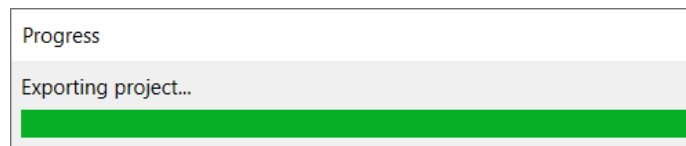
## “Converting” a Subversion App to Git

If you already have an existing versioned app (with Subversion) that you would like to upload to your Git private server instead, you can achieve so by exporting it, then re-importing it and uploading it to your server. Follow the steps below:

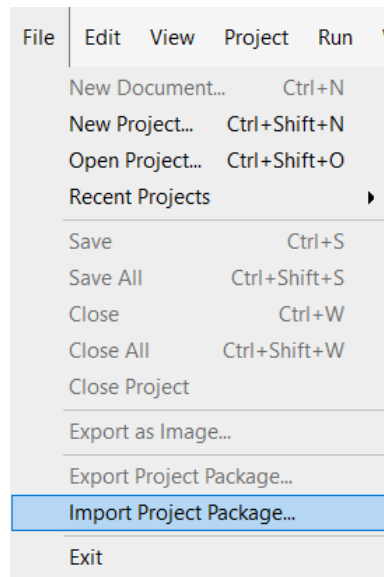
1. Once the app is opened, go to **File > Export Project Package**.
2. In the **Export Project Package** dialog box, browse to the location you would like to save the **mpk** (Mendix Package) file, or accept the default location, a new **packages** folder in the root of the application folder. Take note of this location, as you will need it later. You can also rename the **mpk** file (e.g. **MyGitApp.mpk**) and the app will be named that way once you import it and upload it to the Git server.



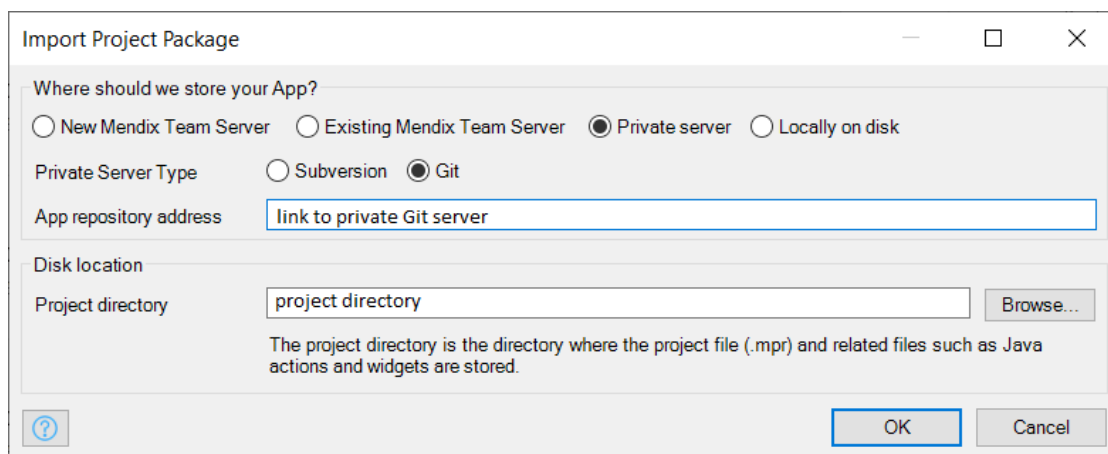
3. The **Progress** pop-up window appears, and once it is completed, you can close the project in **Studio Pro**.



4. Now you can import the project package again, and from there you can choose to upload it to your Git private server. Go to **File > Import Project Package**.



5. Once the file browser dialog open, navigate to the location you save the **mpk** file during the export process.
6. In the **Import Project Package** dialog box, select **Private server** option in the **Where should we store your App?** section.
7. In the **Private Server Type** option, select **Git** (if you have both **Subversion** and **Git** enabled in the **Preferences** form).
8. Enter the link to the private repository in the address textbox and click **OK**. Remember, the repository **must be completely empty**, or **Studio Pro** will not be able to upload a project to it.




9. Click **OK**.

After the import process is completed, your previous Subversion app will be now versioned using Git.

Note that your previous app still exists, **Studio Pro** will simply make an unversioned copy and upload it to your private Git server. So, in your **Recent Apps** list, you will still see both:


### Recent Apps

 **MyGitApp**  
Branch line 'master' (Private Server, Git)

[Project Dashboard](#)

[Open](#)

---

 **MySuversionApp**  
Main line (Team Server)

[Project Dashboard](#)

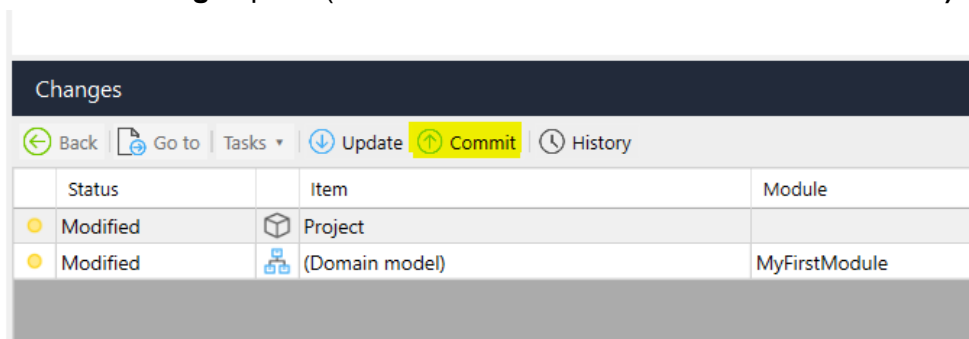
[Open](#)

## Day-to-Day Use

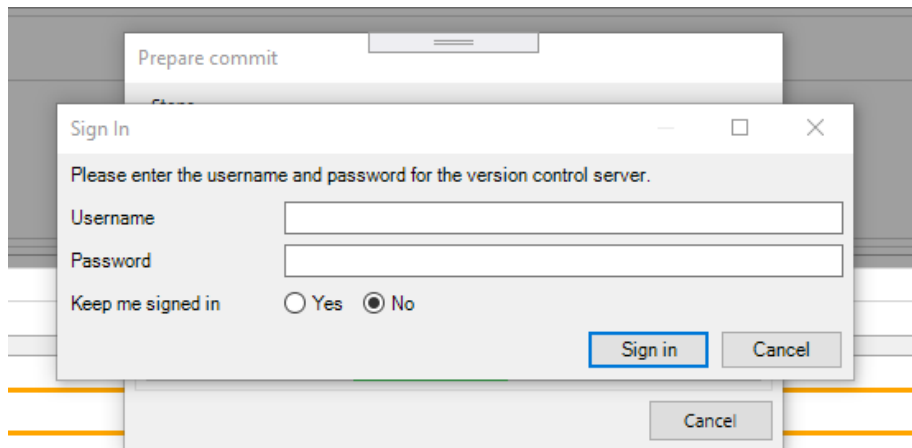
### Committing

To commit your changes, do the following:

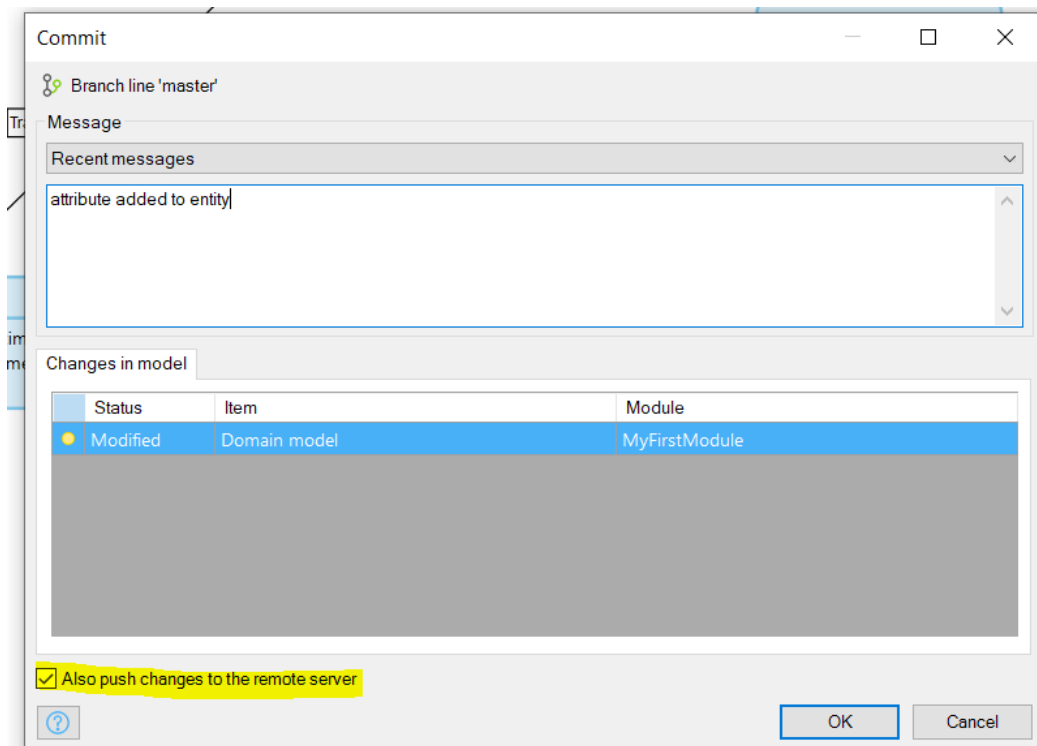
1. Open a Git app in Studio Pro.
2. Make the necessary changes and save them.
3. Click **Commit** in the **Changes** pane (or under the menu **Version Control > Commit**).



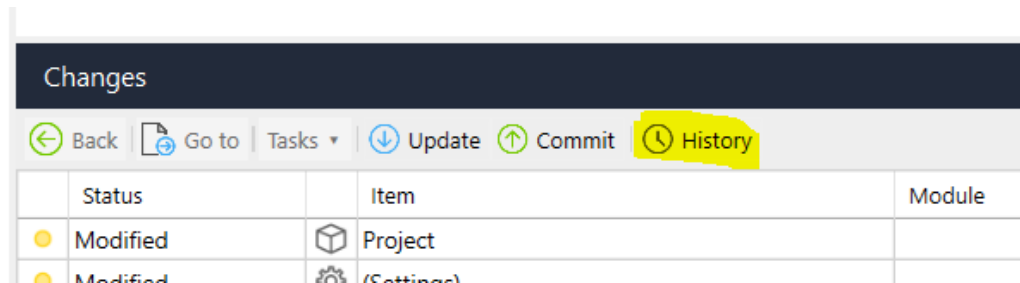
4. In the **Sign In** dialog box, enter your credentials:



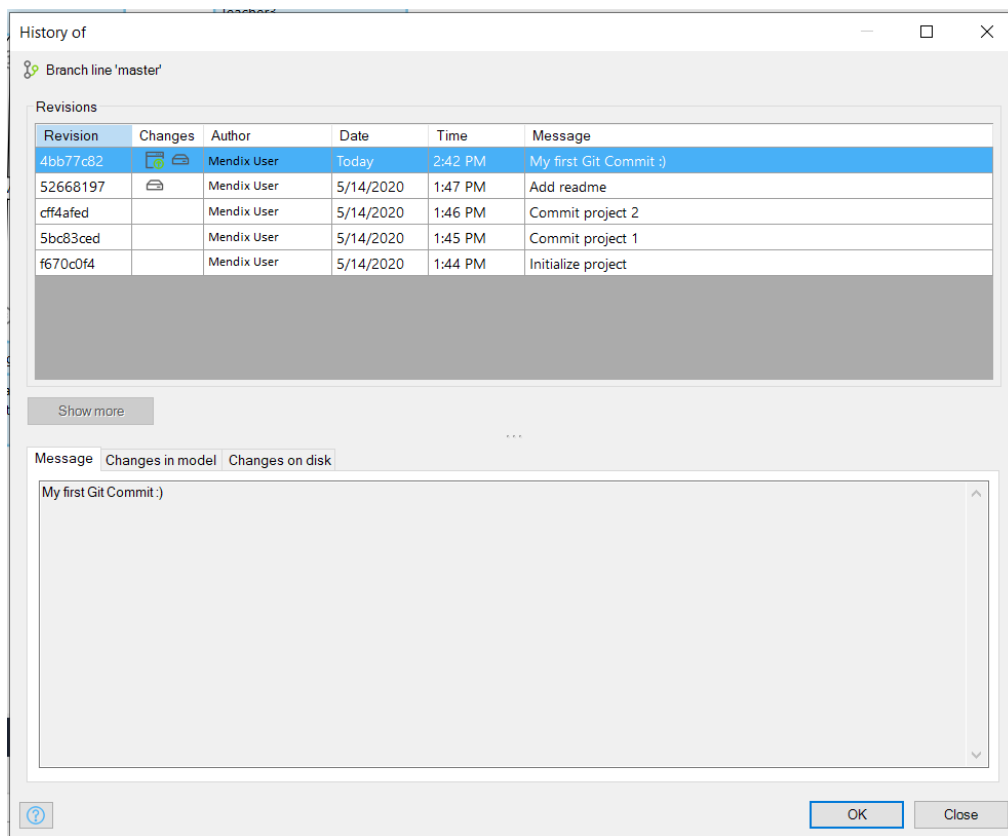
5. Enter an author name (can be anything except empty) and use the PAT that accesses the repository as the password.
6. Enter a commit message and you can de-select **Push to repository** if you want to. This will make a local commit that you can push later under **Version Control > Push**.



7. After you made a commit, click the **History** button in the **Changes** pane or go to **Version Control > History** to view the history of the commits.



8. In the **History of** dialog box, you can view the list of commits that shows a commit hash, together with the commit's author, time, and message:



Once you have pushed the commit, you will also see the commit on the server's commit history.

## Updating

If the same app has been downloaded from the same server into two different locations (whether it is on your machine in two different folders or two users have one copy each), you can keep them up to date with each other by using the **Update** feature.

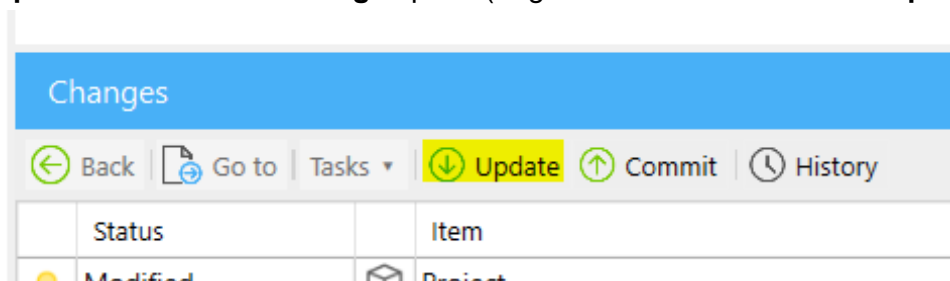


For example, Studio Pro user Andrew (user A) has a copy of **MendixGitApp** on his computer. Another user Barbara (user B) has downloaded **MendixGitApp** from the same private repository on her machine.

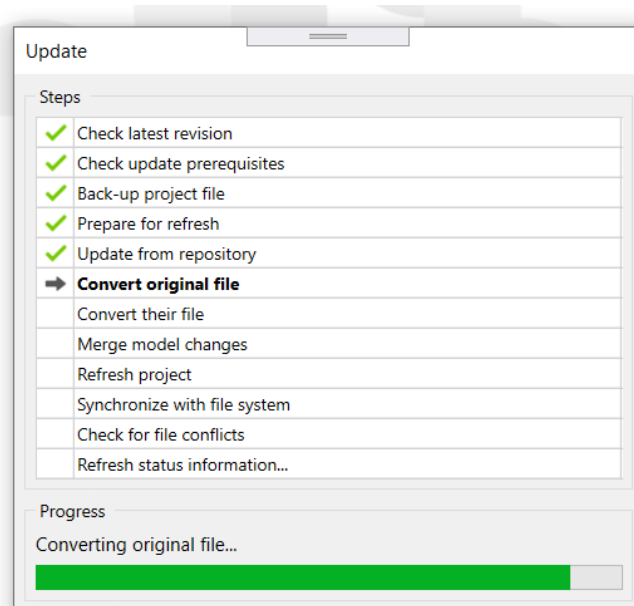
### Updating a Working Copy with Changes from Another User

Andrew adds a new entity to the domain model, commits, and pushes that change to the server. Andrew messages Barbara and tells her that there are new changes on the server, and she should make her version of the app be up to date.

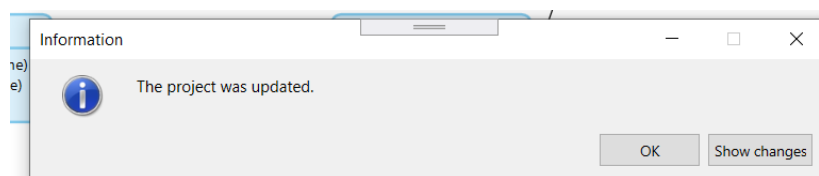
Barbara clicks the **Update** button on the **Changes** pane (or goes to **Version Control > Update**):



The update process will begin, and a dialog will show the progress:



Once completed, Barbara can review the changes by pressing the **Show changes** button on the confirmation dialog:



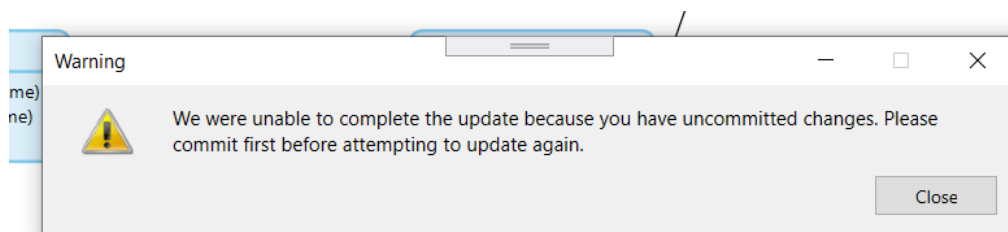
The entity that Andrew added on his version of the app will now be shown as a change in Barbara's version on the **Changes** pane. Barbara can now keep working and eventually commit (and push) this change.

There are some other scenarios where the update might require a bit more user action:

- Barbara cannot update because she has uncommitted changes
- Barbara cannot push her changes because Andrew already pushed to the server
- Barbara and Andrew made changes to the same item and the changes will conflict

### Failing to Update

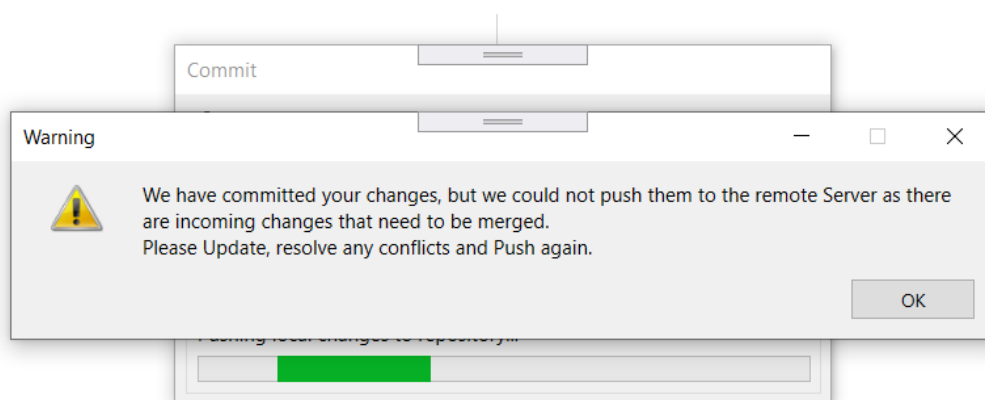
Barbara created a new page, but before she gets to commit it, she receives a message from Andrew that there are some new changes on the server from him. So, Barbara pushes the **Update** button on the **Changes** pane, but she then gets the warning below.



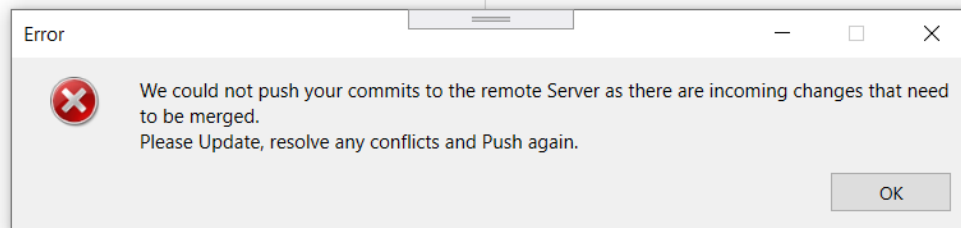
Barbara cannot receive the remote changes until she commits her own. Only then she can try to update again.

### Failing to Push

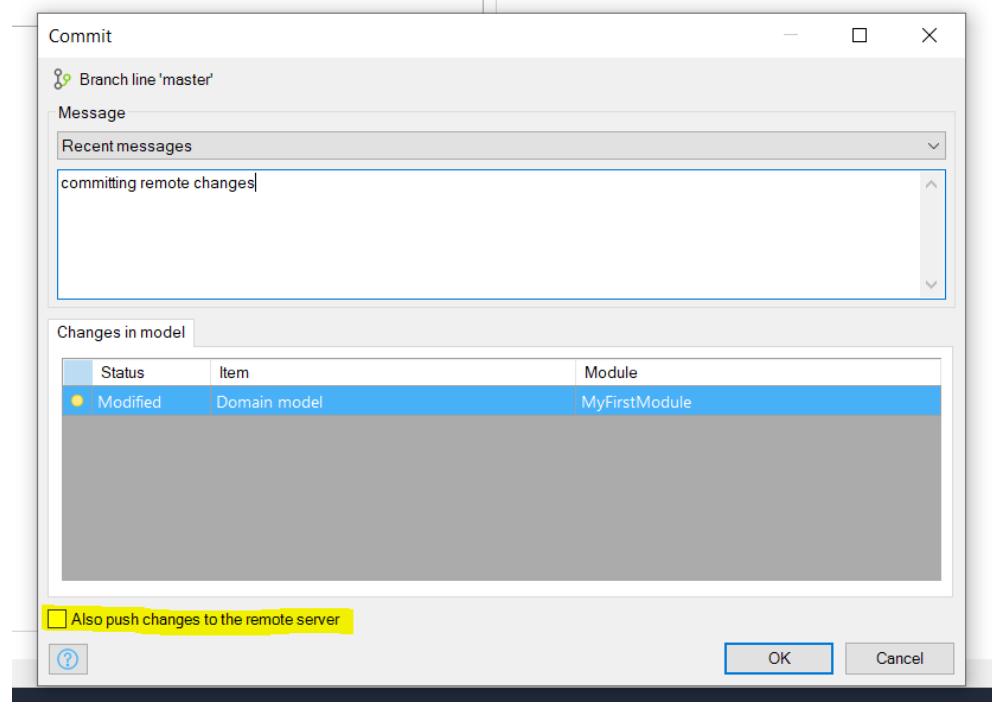
Andrew creates a new page, commits the changes and pushes it to the server. Five minutes later, Barbara is modifying the domain model and also commits and pushes her changes. A warning message will show, telling Barbara that her commit has been performed, but it was not pushed successfully to the server.



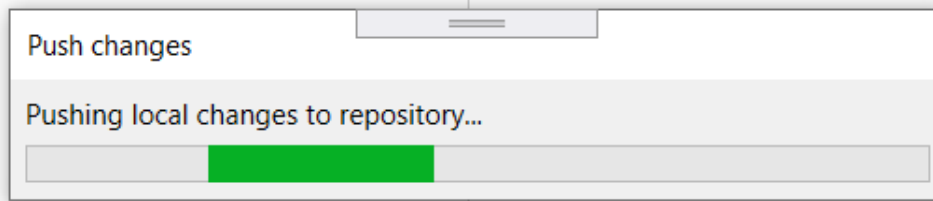
If Barbara opens her commit history, she will see that the last commit was created. If Barbara looks up the commit history on the server, she will see that the commit has not yet been pushed to it. She will need to perform an update (button on **Changes** pane or from **Version Control > Update** menu). Once she receives the remote changes from the server, she needs to commit them, and only then she is able to push her own. If Barbara tries to push before committing the remote changes, she will get the error message below.



So, Barbara will perform a commit, but she needs to deselect the **Also push changes to the remote server** option:



After the commit was performed, Barbara can now push to the remote server, from **Version Control > Push**:

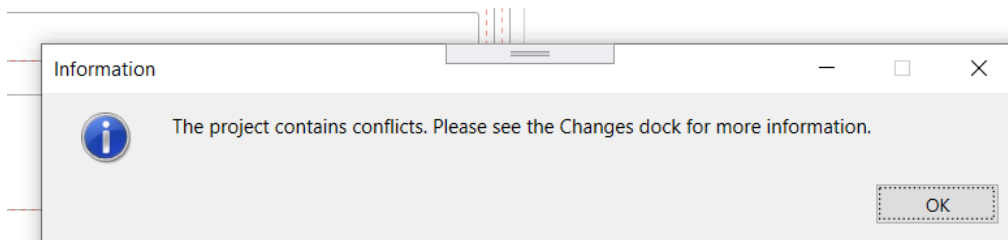


After the push is completed, Barbara and Andrew will have matching versions of the application again.

### Having Conflicting Changes

Barbara changed the title of a page, but a few minutes earlier, Andrew already changed the same title on the same page with a different text and pushed that change to the server.

When Barbara performs an update, she will get Andrew's changes on her version of the application, but they will conflict with her own. After the update, she will get the information message below:



To resolve the conflicts, see the Mendix documentation on [dealing with conflicts](#).

After Barbara has resolved the conflicts, she can now commit the resolution of that conflict.

## Pushing to the Remote Server

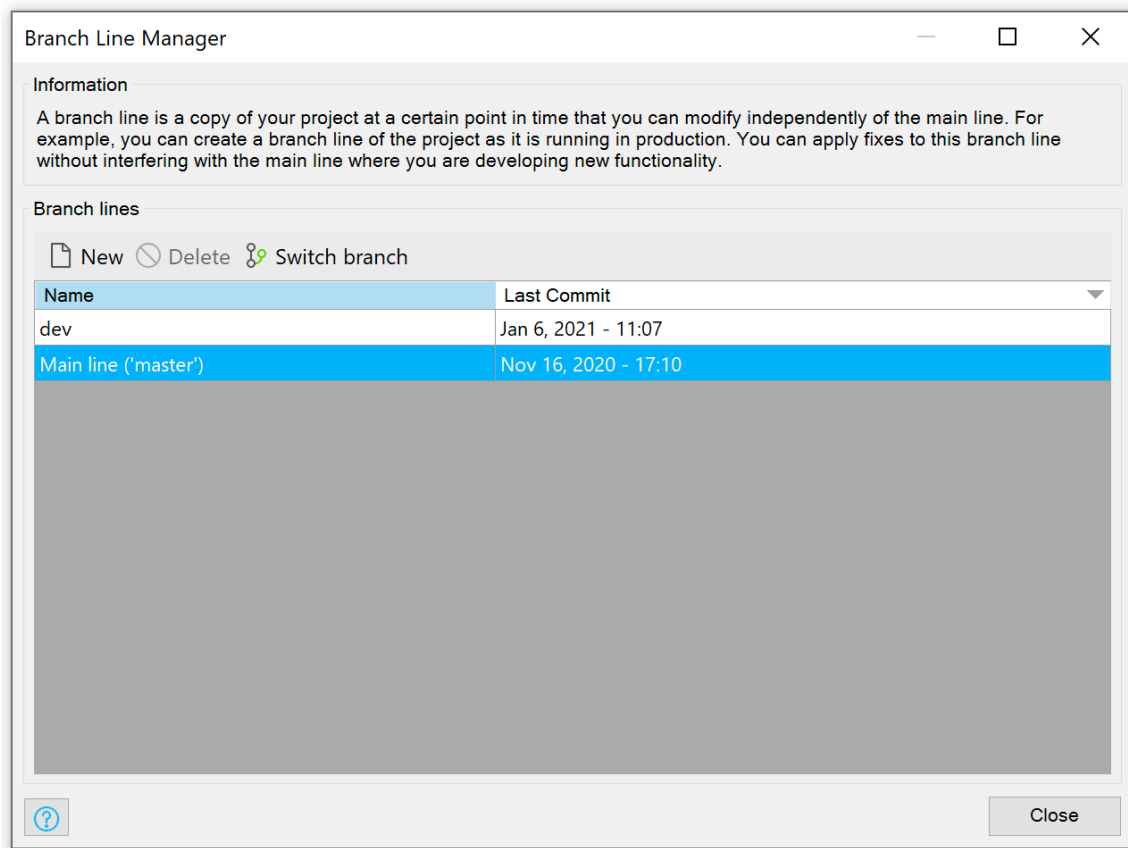
If you have performed one or more local commits, you will need to eventually push them to the remote server. During the commit process, Studio Pro has the option to push together with the commit already enabled, but you might choose to disable it, resulting in local commits that will be missing from the server. Or you could get a successful local commit that has not been pushed yet, if you suddenly lose your internet connection when Studio Pro begins the push process.

In order to push all your local commits, go to **Version Control > Push** (note that there is no such option in the **Changes** pane). After doing so, all your previously local commits will be available for your team to download.

## Working with Branches

Development lines other than the main line are called branch lines. Our advice would be to develop new features in the *main line* and to use *branch lines* for fixing bugs in versions that have been deployed. This is the scenario Studio Pro makes easy but other scenarios for more complex projects are supported as well.

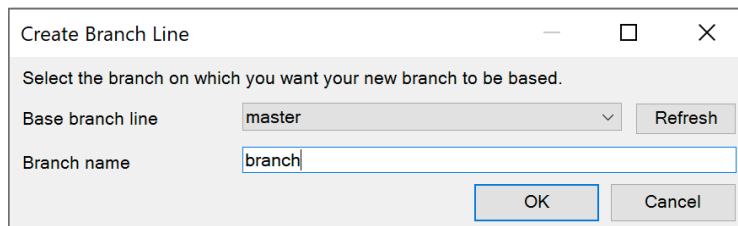
You can list or create branch lines from the **Branch Line Manager** dialog which you can find at **Version Control > Manage Branch Lines...**



## Creating a New Branch Line

To create a new branch line, do the following:

1. Click **Version Control > Manage Branch Lines**.
2. In the **Branch Line Manager** dialog box, you see the list of existing development lines. Click the **New** button to create a branch line.
3. In the **Create Branch Line** dialog box, specify a name for the new branch line and a base branch line it will be forked from.



Create Branch Line

Select the branch on which you want your new branch to be based.

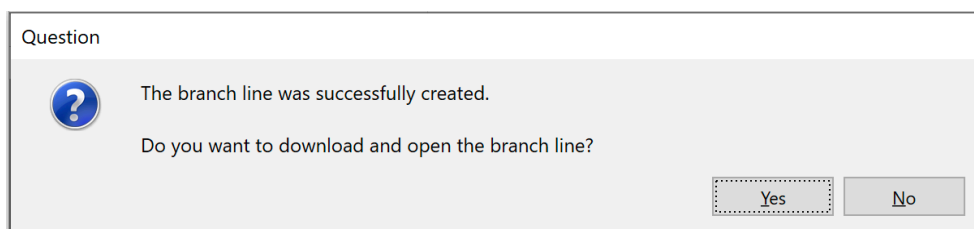
Base branch line: master (dropdown menu) Refresh

Branch name: branch (text input)


OK Cancel

4. Click **OK**.

The new branch is created, and a confirmation pop-up message appears:



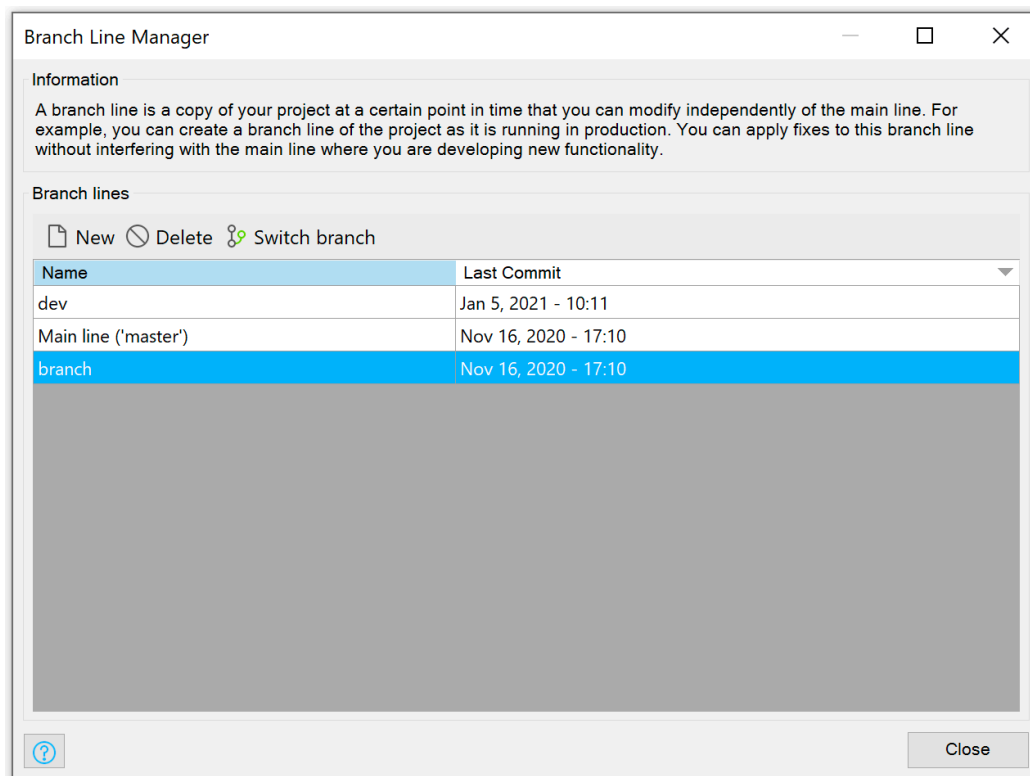
Question

 The branch line was successfully created.

Do you want to download and open the branch line?

Yes No

You can either choose to download the new branch line and switch to it or skip the transition, which will bring you back to the **Branch Line Manager** dialog box where you can find a newly created branch line in the **Branch lines** section:






Branch Line Manager

Information

A branch line is a copy of your project at a certain point in time that you can modify independently of the main line. For example, you can create a branch line of the project as it is running in production. You can apply fixes to this branch line without interfering with the main line where you are developing new functionality.

Branch lines

 New  Delete  Switch branch

Name	Last Commit
dev	Jan 5, 2021 - 10:11
Main line ('master')	Nov 16, 2020 - 17:10
branch	Nov 16, 2020 - 17:10

Close

## Switching Between Existing Branch Lines

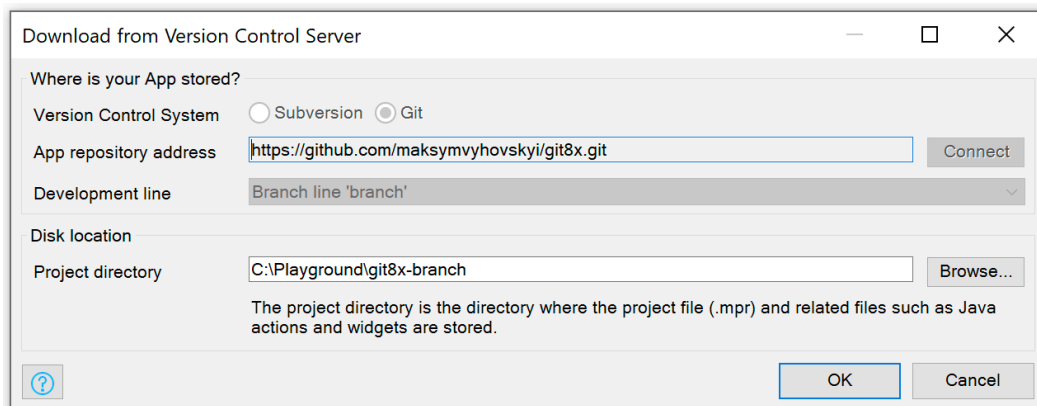
To switch between branch lines, do the following:

1. Open **Version Control > Manage Branch Lines**.
2. In the **Branch Line Manager** dialog box, you see the list of existing development lines. Select a line with the branch you would like to switch to.
3. Click the **Switch branch** button (or double-click the line).

The **Branch Line Manager** dialog box is closed automatically.

If the branch line selected has not been downloaded yet, the following process starts:

1. The **Download from Version Control Server** dialog box appears.
2. The **Version Control System**, **App repository address**, and **Development line** fields are filled in automatically.
3. Click **Ok** button to download the selected line and switch to it:

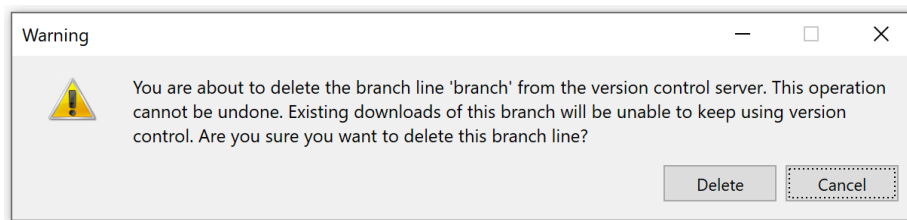


**Note:** You cannot switch to the current branch line. To prevent this the **Switch branch** functionality becomes disabled while another branch line is not selected.

## Deleting an Existing Branch Line

To delete a branch line, do the following:

1. Open **Version Control > Manage Branch Lines**.
2. In the **Branch Line Manager** dialog box, you can see the list of existing development lines. Select a line with the branch you would like to delete.
3. Click the **Delete** button.
4. in the **Warning** pop-up window, click **Delete**:

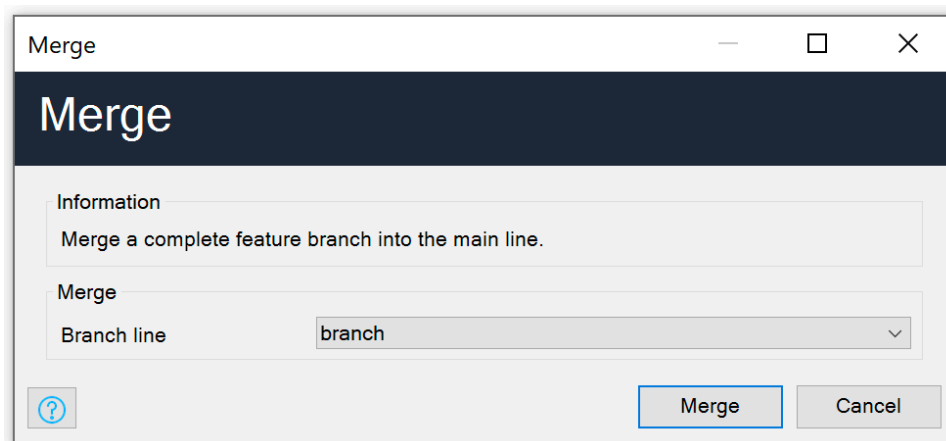


**Note:** You can delete neither the main line nor the current branch line.

## Merging into Main Branch Line

To merge your changes into the main branch line, do the following:

1. Open **Version Control > Merge Changes Here...**
2. In the **Merge** dialog box, select a branch line that you would like to merge into the main line.
3. Click the **Merge** button.



**Note:** Make sure that the changes are committed to the current branch line.



# Known Issues and Troubleshooting

## Known Issues

You may encounter the following issues:

- **Proxy servers are not supported** — we do not support connecting to Git repositories from behind a proxy server yet.
- **You might get an “Oops” dialog when setting a name and an email for Git in the Preferences dialog box or while committing** — the problem might occur if there is no global *Git config* file on the PC yet.

To fix this issue, you can use the Git command line client and do the following:

- Specify the user name:

```
git config --global user.name "<Name>"
```

- Specify the email:

```
git config --global user.email "<Email>"
```

Any of these commands will create a global *Git config*. Subsequent interactions via the Studio Pro interface will succeed from now on.

# Troubleshooting

## Connection Problems When Cloning the Git Repository

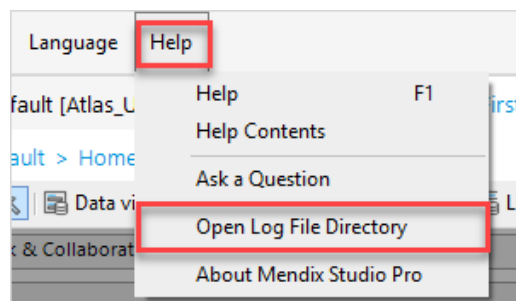
If you face connection problems when cloning the Git repository using the **Open App** or **Download App** dialog box, the first thing to check is whether the URL of the remote Git repository is correct. It should **not** be copied from the browser address bar. Most Git services have a noticeable colored **Clone** button which provides the correct URL in a pop-up window. You should use this URL with Studio Pro.

## Customer-Facing Issues

For Studio Pro developers to be able to troubleshoot issues that the customers face with experimental Git support, Studio Pro provides logging mechanism.

When filing a Git support issue with Mendix Support, attach the log files doing the following:

1. Navigate to the **Help** menu > **Open Log File Directory**:



2. Copy the file called *log.txt* into your ticket. You can also attach additional *log.X.txt* files if they exist.

## Git Properties Useful for Troubleshooting

There are properties of the Git repository that provide you the information useful for troubleshooting different issues execute the following Git command line commands in the project's folder:

```
git status -b — provides information on the current state of the repository
```

```
git remote -v — lists the remotes specified for the repository
```

```
git config --list --show-origin --show-scope — provides information on user's Git config
```

**WARNING:** PLEASE NOTE THAT THIS MIGHT CONTAIN PERSONALLY IDENTIFIABLE INFORMATION, SO MAKE SURE ALL THE PRIVATE INFORMATION IS REMOVED BEFORE SHARING!

Mendix is the fastest and easiest platform to build and continuously improve mobile and web applications at scale. Recognized as a market leader by leading analysts, we help our customers digitally transform their organizations and industries by building, managing, and improving apps at unprecedented speed and scale. More than 4,000 forward-thinking enterprises, including Kuehne + Nagel, KLM, Philips and North Carolina State University use our platform to build business applications to delight their customers and improve operational efficiency.