git4G
Quick Reference Guide V 1.2.0

DATA AHEAD **git4G** is an easy to use toolkit for NI LabVIEW that integrates Git- the well-known and free Source Control Management System into the development environment of LabVIEW. It provides all important source control managing functions and furthermore supports many popular Git hosting services like GitHub and GitLab

Introducing git4G

git4G is an easy to use toolkit for NI LabVIEW that integrates Git - the well-known and free Source Control Management System -- into the development environment of LabVIEW. It provides all important source control managing functions and furthermore supports many popular Git hosting services like GitHub and GitLab.

git4G uses and manipulates the fundamental functionality of LabVIEW. Because of this DATA AHEAD cannot guarantee that the behaviour of LabVIEW has changed or that it still works after the installation of the toolkit. Consider to save your important data separately. In general, DATA AHEAD cannot be held liable for any loss of data that occurs while using git4G.

Safety

> **Git4G** uses and manipulates the fundamental functionality of LabVIEW. Because of this reason, DATA AHEAD cannot guarantee that the behaviour of LabVIEW has changed or that it still works after the installation of the toolkit.

Minimum requirements

OS: Windows XP
LabVIEW: Version 2010
(For using the LabVIEW Compare and the LabVIEW Merge Tool it needs to be the LabVIEW Professional Development System)
Git: Version 2.2 (Download-Link: https://git-for-windows.github.io/)

Installation

LabVIEW 2010 or newer is required to install **git4G**.

The installation of the toolkit requires the installation of Git (at least Version 2.2).
We recommend at least a basic knowledge about the functionality of Git.
Follow the link below to learn more about Git:
https://git-scm.com/book/en/v1/Getting-Started

1. Git Installation

Download and install the latest version of Git.
Link: https://git-for-windows.github.io/

For a more detailed installation description about different operating systems, please
visit: https://git-scm.com/book/en/v2.

2. git4G Installation

The easiest way to install **git4G** is to use the National Instruments Tools Network or the
JKI VI Package Manager. Search for "git4G" and select "Install & Upgrade Packages" to
install the Toolkit to your LabVIEW installation.

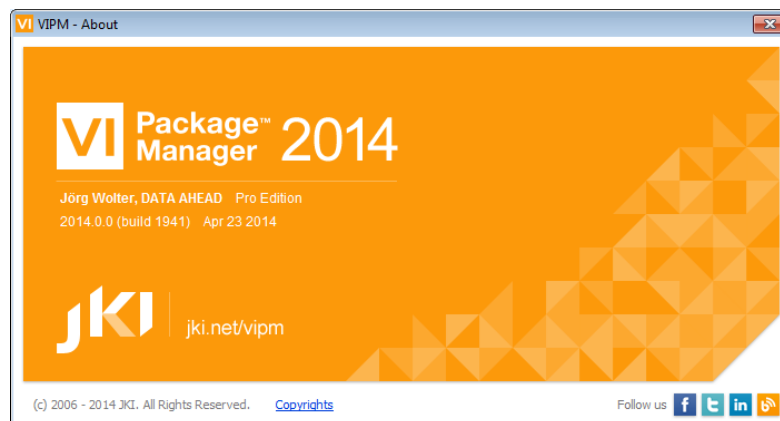**git4G** requires the JKI VI Package Manager 2014.0.0 or newer.

Figure 1: JKI Package Manager to find and install git4G

In case LabVIEW does not restart automatically after the installation, you should do it
manually to activate the tool.

Licensing

Developer license:

You need a developer license in order to use **git4G**. Upon purchasing a license from NI, you will receive an email titled "*Activation Information for your ni.com purchase:* **git4G** *-* *DATA AHEAD GMBH*" that contains your **License ID** and **Password** for the developer suite. With this information, you are able to activate **git4G.** The installation description is located in this document under "Activating".

The license is a single End-User PC License. With a single End-User PC License you may install and use one copy of the SOFTWARE PRODUCT on a single computer. In addition to the above, if you have bought a License for a number of concurrent users you may install the SOFTWARE PRODUCT on any number of computers at one single physical (geographical) location notified to Licensors provided it is used on no more computers than that number of licensed users at any one time.

You cannot deactivate the license and reactivate it on a second computer.

Activating

You can evaluate **git4G** for 30 days. After this period you have to activate the **git4G** add-on in LabVIEW. Upon purchasing a license from NI, you will receive an email titled "*Activation Information for your ni.com purchase:* **git4G** - *DATA AHEAD GMBH*" that contains your License ID and Password for the developer suite. In order to activate **git4G**, your target system needs to be connected to the Internet. If there is no Internet connectivity available on the target computer, you can also activate **git4G** through a web browser on a different computer (see step 3 – alternative option).

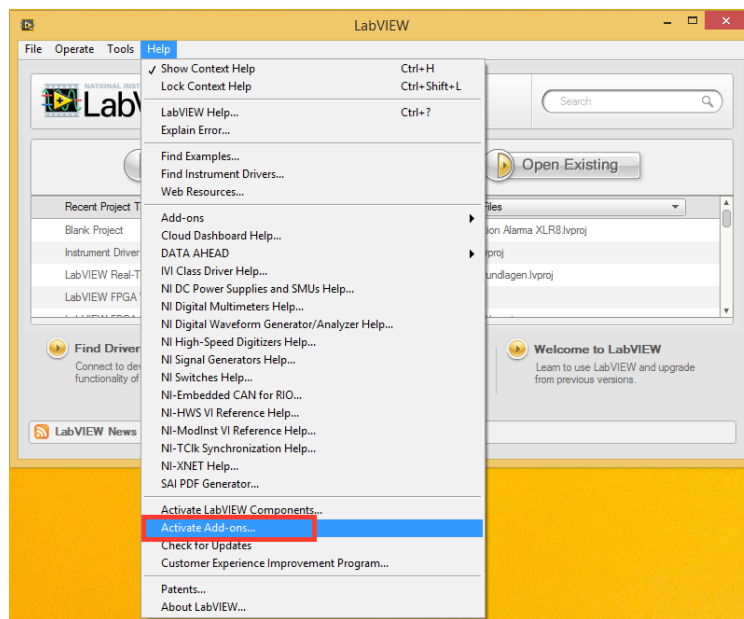**Step 1**: Open LabVIEW. Select "Help" in the menu bar then the menu point "Activate Add-ons" to carry out the function.

Figure 2: Activation Step 1

Step 2: The following pop-up window will appear and display your **git4G** installation and activation status. Select **git4G** "DATA AHEAD **git4G** x.x.x.xx" and click "next" or "Activate".
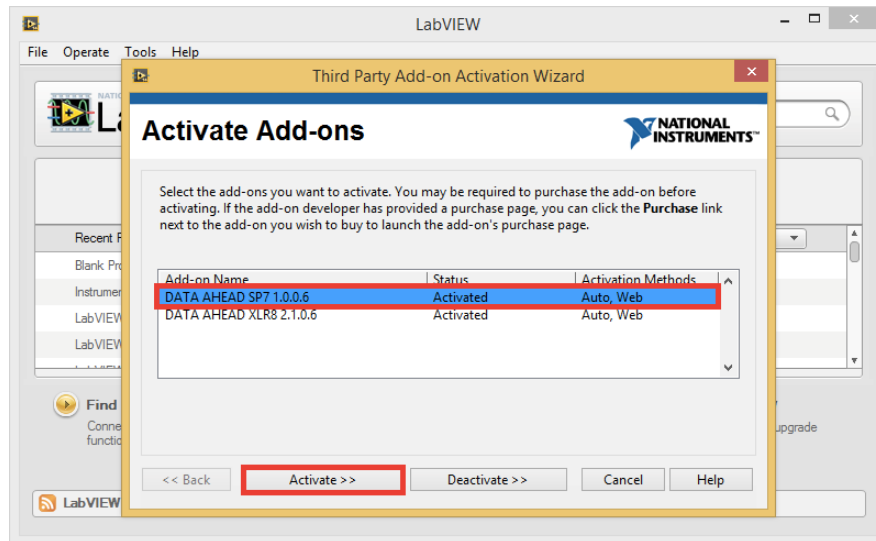

Figure 3: Activation Step 2

Step 3: Now select the method of activation. If the computer is connected to the Internet, select the first option. If you do not have Internet access, see the alternative option below. To proceed, click "Next".
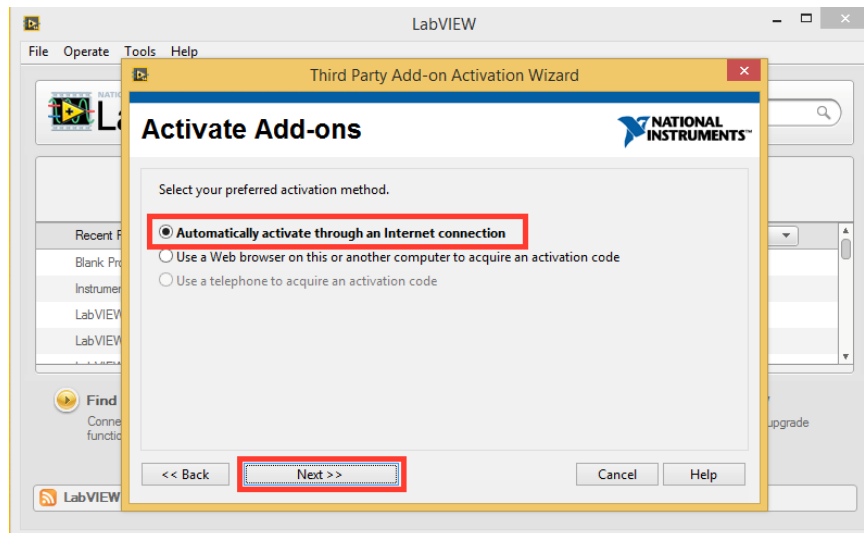

Figure 4: Activation Step 3

Step 4: To complete the activation, enter the provided License ID and Password that you received in the email from NI. Please mind capital letters.
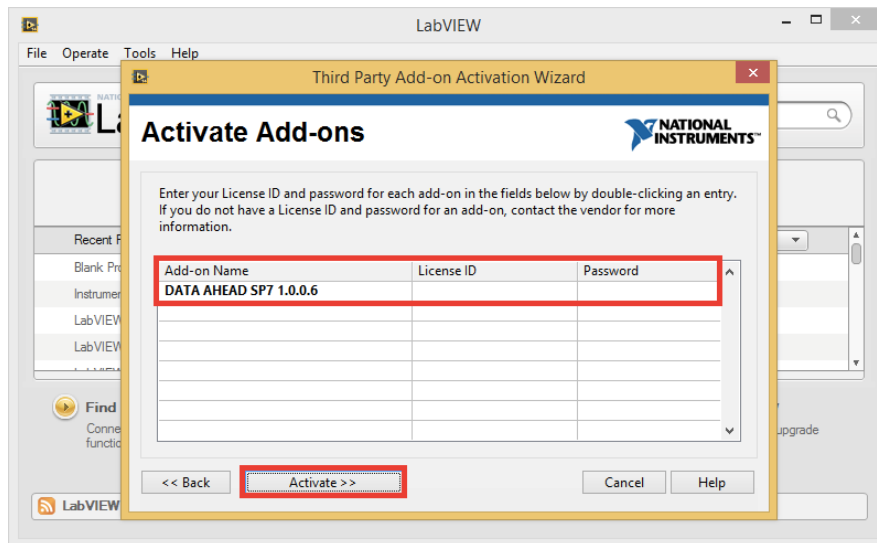After entering the License ID and Password, click "Activate".

Figure 5: Activation Step 4

Step 3 (alternative): If no Internet connection is available, selecting the second option will direct you to the National Instruments activation website where you need to enter the user codes displayed in the dialog as well as your License ID and Password. This option allows you to activate the toolkit from another computer that is connected to the Internet.
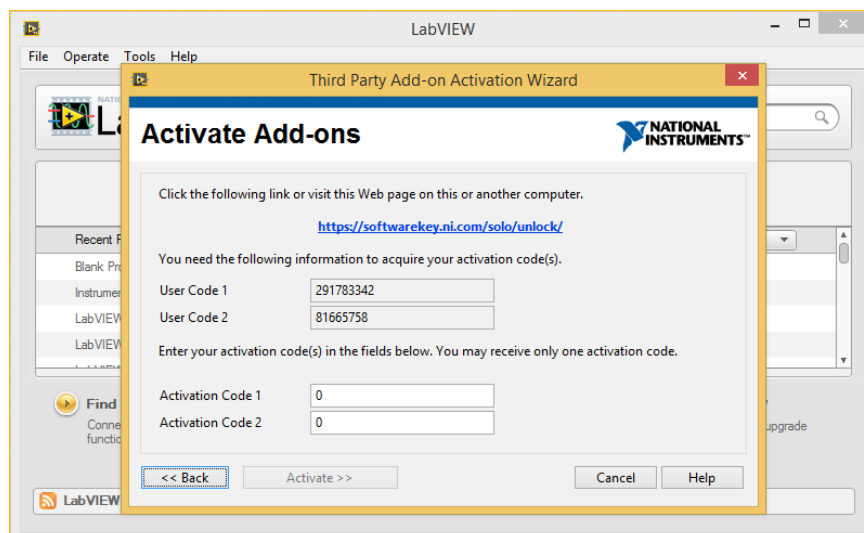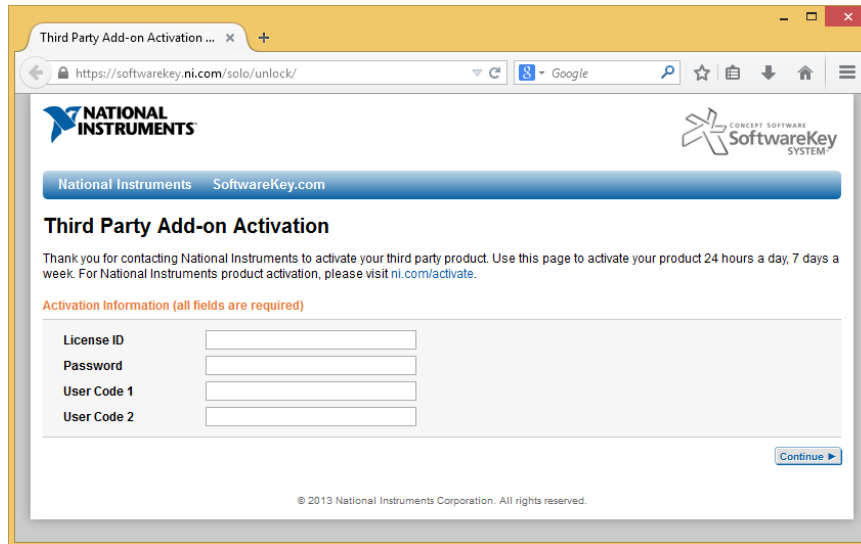

Figure 6: Activation Step 3, (alternative): activate through a web browser – LabVIEW dialog

Step 4 (alternative): Open the website https://softwarekey.ni.com/solo/unlock/ on a computer with Internet connection. Enter the User Code 1 and User Code 2 on the website, as displayed in the dialog on the target computer. Enter License ID and Password from the activation email. Click "Continue". Now the User Code 1 and 2 will be generated. Afterwards you need to type in these two Codes in the dialog box of the Activation Wizard at Activation Code 1 and 2. Finish the dialog by clicking "Activate".

Figure 7: Activation Step 4, (alternative): activate through a web browser – activation website

First start after installation

If you start LabVIEW after the installation of **git4G** the tool checks your system for Git. If **git4G** is not able to find a valid Git installation a dialog pops up where you can navigate to the location of a Git installation or deactivate the tool.

Note: This Deactivation does not affect the license.
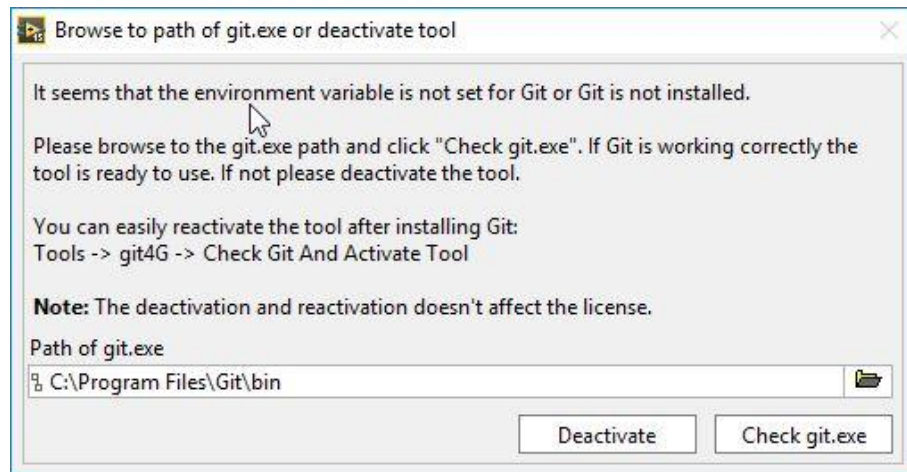
Figure 8: NI LabVIEW Project Explorer with git4G toolbar and pop-up menu

If the tool is deactivated just install Git (Download-Link: https://git-for-windows.github.io/) and start LabVIEW again.

If Git installed into the default path or the option "Run Git from Windows Command Prompt" was checked **git4G** automatically recognizes the Git installation and activates all **git4G** options. Otherwise open the dialog to navigate to the correct path ("Tools" -> "git4G" -> "Check Git And Activate Tool").

Getting Started with git4G

The **git4G** toolkit expands the toolbar by four buttons ("Git Menu", "Update", "Remote Update" and "Settings"

The "Tools" menu of the LabVIEW Project Explorer will be expanded by a **git4G** section. Here you can perform all the important Git functions to the repository corresponding to the open project, open the **git4G** settings the "Git Menu" or clone a repository

Furthermore, the toolkit adds **git4G** elements to the pop-up menus when right clicking on LabVIEW Project items. Which functions will be shown in the pop-up menu depends on the status of the repository and on which project item you click.

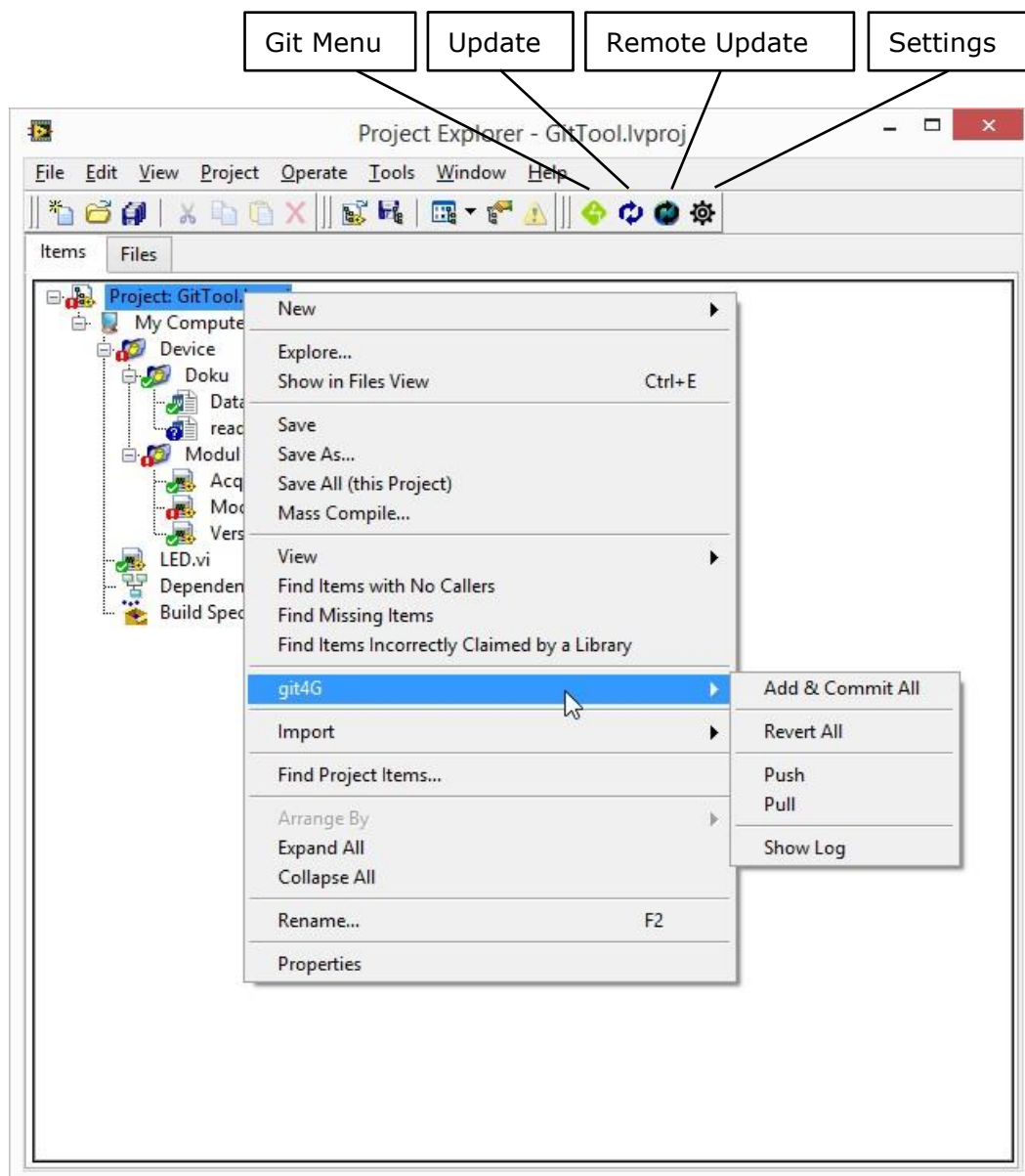**Figure 9: NI LabVIEW Project Explorer with git4G toolbar and pop-up menu**

The **git4G** toolkit uses overlay icons to display the current source control management status of the files in the Project Explorer.
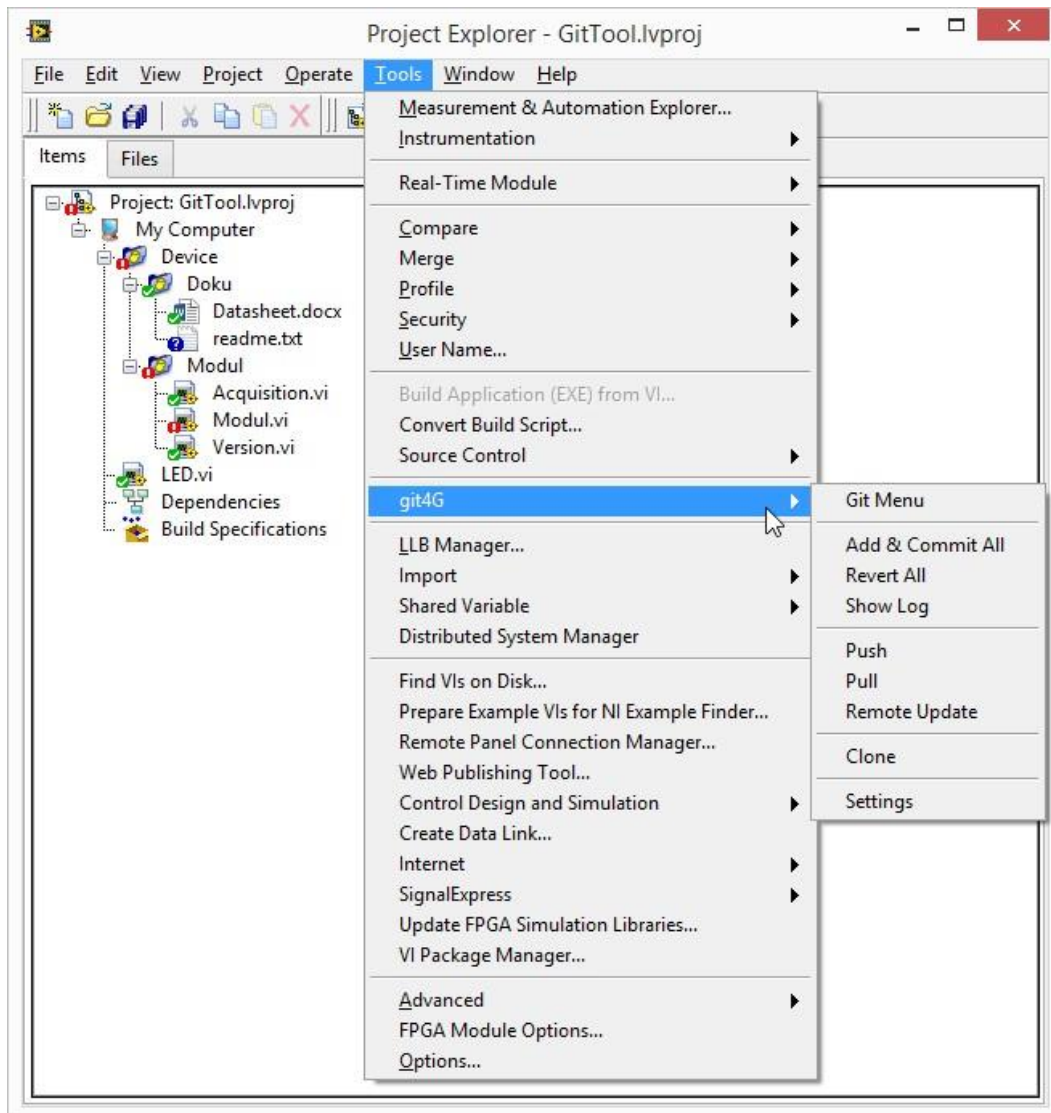

Figure 10: NI LabVIEW Project Explorer with git4G entries in "Tools" menu

To use the toolkit we recommend at least a basic knowledge about source control management using Git.

Follow the link below to learn more about Git:
https://git-scm.com/book/en/v1/Getting-Started

Use of git4G

git4G makes it easy to call all important Git functions directly from the NI LabVIEW Project Explorer. There are three different ways to call the functions:

- "git4G" submenu of the Project Explorer "Tools" menu
- "git4G" submenu of the context menu of LabVIEW Project Explorer items
- Git menu called by button ⬢ (Git Menu) in the toolbar (or "Tools" menu -> "git4G" -> "Git Menu")

Initialize source control to your project – creating a repository

To initialize Git source control to your LabVIEW project you need to create a Git-repository. Before doing this, the project file needs to be saved.

With **git4G** there are two ways to create a repository:
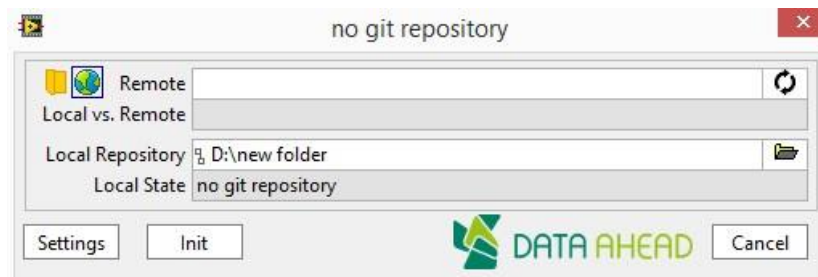
1. Open the Git menu and press "**Init**"

Figure 11: Creating a repository from Git Menu

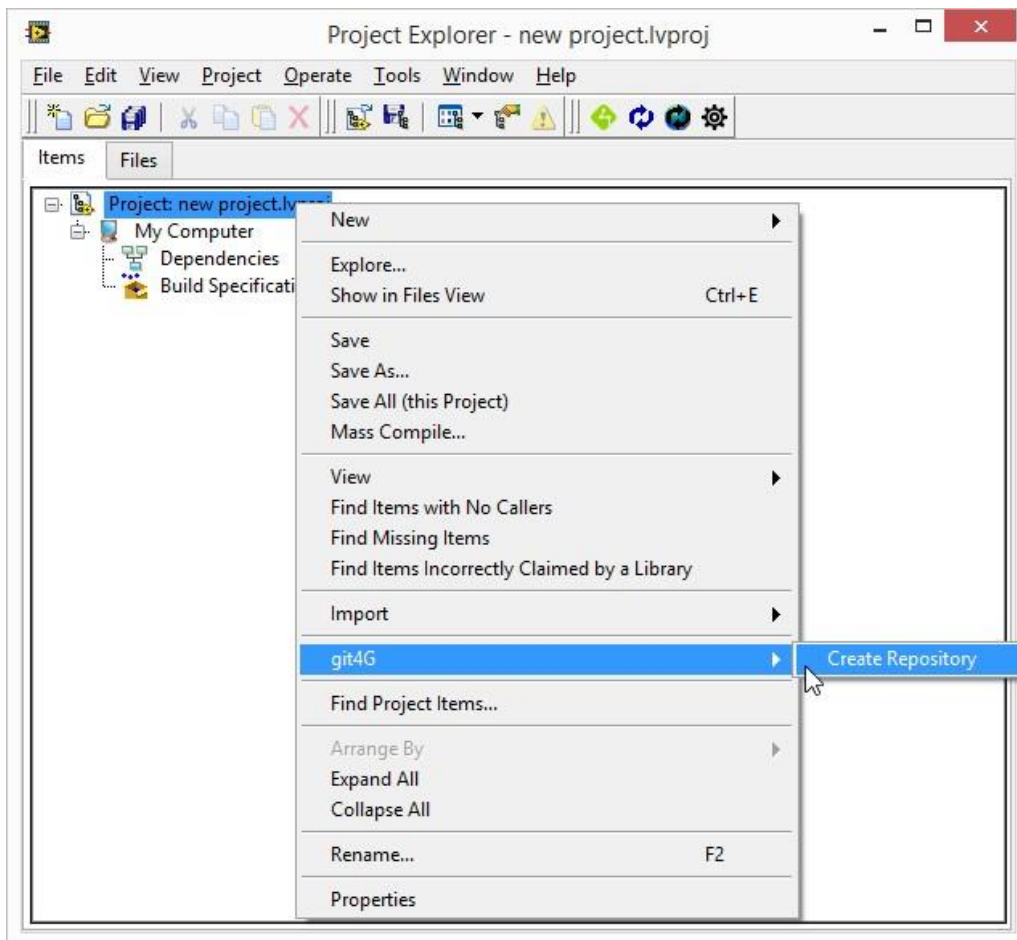2. Right-click on the project and select "git4G" -> "Create Repository"

Figure 12: Creating a repository from Right-click pop-up menu

Second step is to browse to the path where you want to create the repository. By default, the "Create Repository Dialog" starts with the project-path.
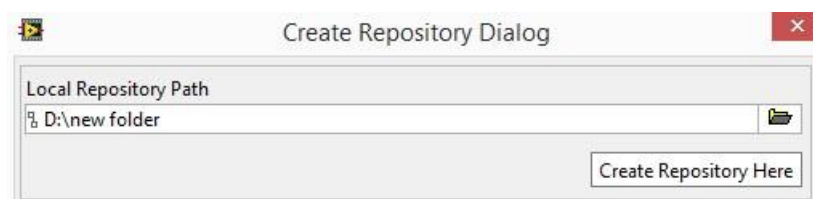

Figure 13: Browse to path where to create the repository

After pressing "**Create Repository Here**" the repository is created and at the same time a ".gitignore"-file will be copied to that location. The ".gitignore"-file contains the information, which files Git will ignore. **git4G** will automatically ignore all *.aliases-files.

Now your LabVIEW project is under Git source control. The status of the project items will be indicated in the NI LabVIEW Project Explorer. You can add and commit single/multiple/all files, take a look at the history of a single file or the whole project and restore old versions.

If you use a remote repository, add the remote address and you are ready to push and pull from or to the remote repository.

Use git4G with a remote repository

git4G can be used with all the main source control functions just with your local repository. But if you want to benefit from all the advantages, a source control like Git offers, you should consider adding a remote repository to your project - especially while working in a team on a project.

**git4G** supports two kinds of remote repositories:
- On a server using SSH to push and pull (select 🌐)
- On your local file system (select 🟧)

Server

One advantage of choosing Git as your source control system is that there are many Git hosting services on the market. These services make it easy to create a remote repository. With **git4G** you can easily add the URL of this repository to your project and you are ready to push and pull to/from this remote repository.

Local file system

Another option is to have your remote repository on your local file system. Just browse to the location of the remote repository. If you have not created one yet, browse to the desired destination and **git4G** asks you to create a bare repository to push and pull to/from it.
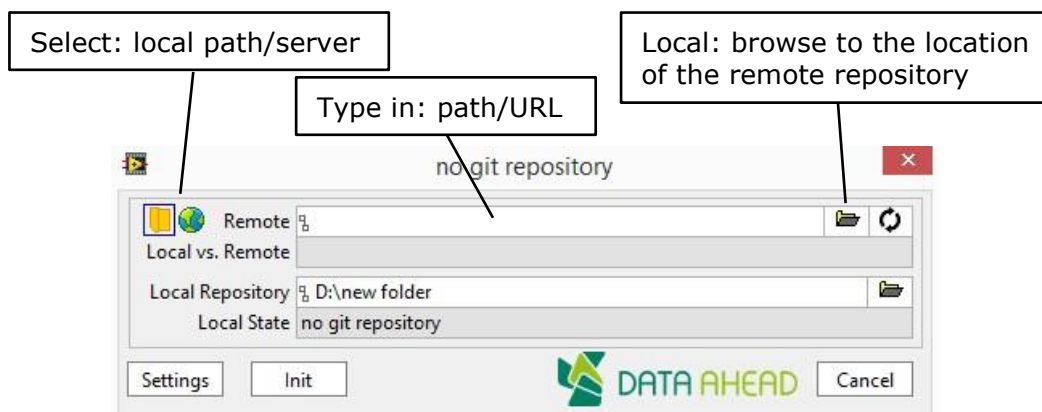
Indication of the current state of the project items in NI LabVIEW Project Explorer

When opening a LabVIEW – project, **git4G** searches for a repository, which belongs to the project, and checks its state. Further on, the state will be checked periodically if "Periodic Status Check" box is checked (see Settings). If the periodic check is not active you can check the state manually by pressings the button 🔄 (Update) in the toolbar.

If a project has a corresponding repository, the elements in the Project Explorer get overlay icons to show their current state.
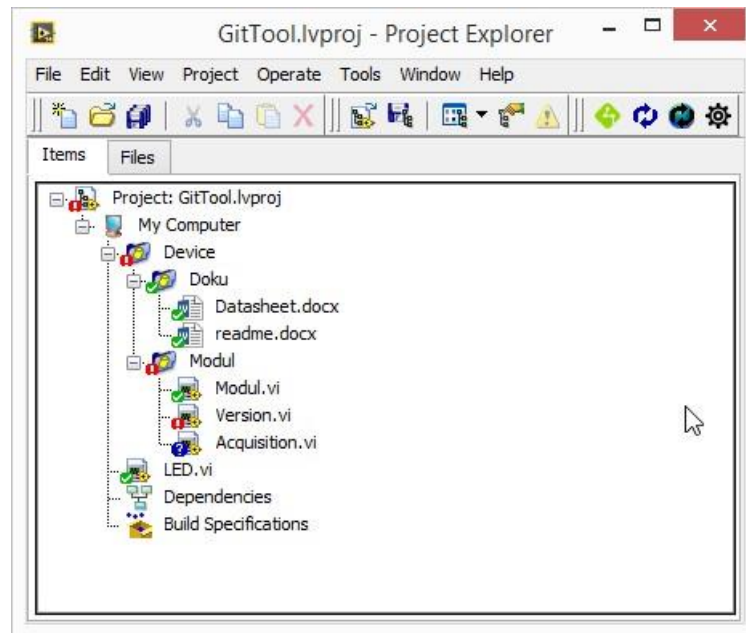

Figure 14: Current state of project items marked with overlay icons

Icon	File	Folder	Project
✅	Up to date	All files in folder are up to date	All files in the repository are up to date – the repository is "clean"
🔴	File has been changed since last commit	One or more files in folder have been changed since last commit	One or more files in repository have been changed since last commit or are unknown to Git
❓	File unknown to Git	-	-
🟡	There is a merge conflict with this file	One or more files inside folder have a merge conflict	One or more files in repository have a merge conflict
No icon	File will be ignored (see ".gitignore"-file)	-	-

Table 1: Overlay Icons

If you encounter performance issues using the periodic check (e.g. large project with a lot of files not "clean") it could be helpful to switch of the periodic check (see Settings).

Git Menu

Opening the Git menu gives you an overview about the status of the repository and its containing files.

Figure 15: Git Menu

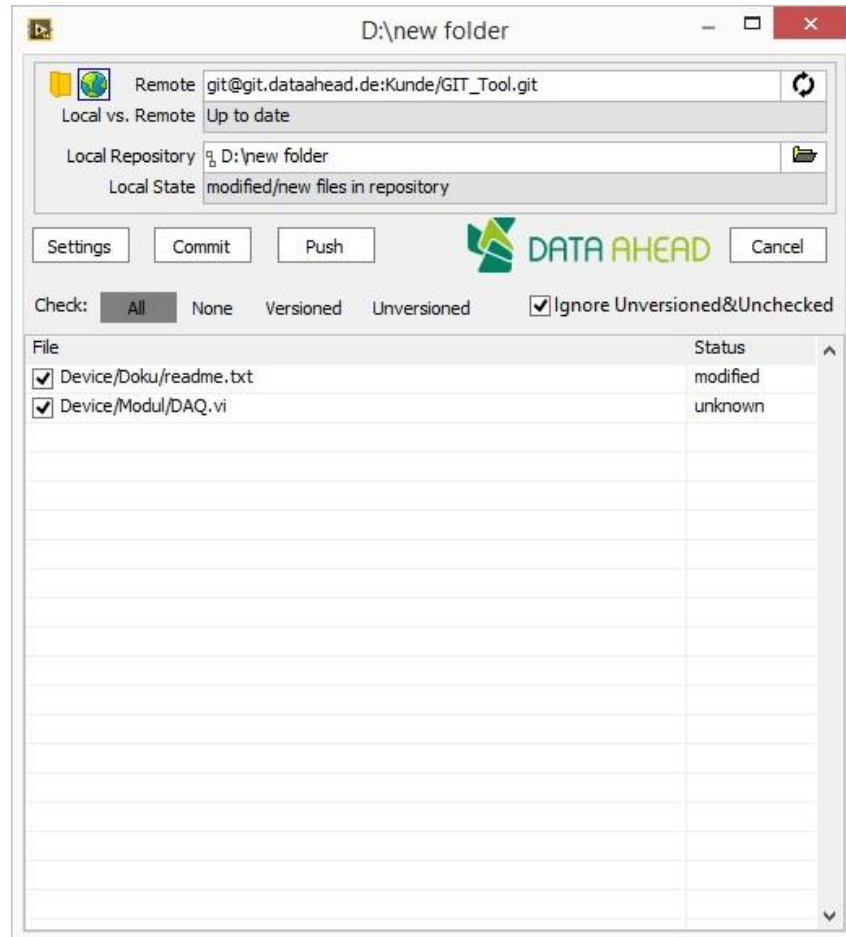If a remote repository was configured for the project, the URL will be shown and, if there is connection to the server, a comparsion between your local and the remote repository will be performed when opening the Git menu. The result will be displayed at "**Local vs. Remote**". Later on you can perform a comparsion by pressing the ⟳ (Remote Update) button.

You can change to a different local repository using the 📁 button and browsing to the path of the repository.

The **git4G** Git menu offers its functions depending on the status of the repository.
If there are files to add/commit the "**Commit**"-button will be shown.
If the path doesn't belong to a Git repository the "**Init**"-button gives you the option to create a repository.
If a remote repository was configured and your local repository is "behind" the remote the "**Pull**" button will be shown. If your local repository is "ahead" the remote the "**Push**" button will be shown.

Note: if your local repository is "ahead" and "behind" you first need to pull before you can push.

If there are files in the repository not identical to the last commit, a list with all these files will be shown on the bottom of the window. Each file gets a checkbox that allows you to choose whether the file should be commited or not. To check/uncheck multiple files at once you can use the buttons "**All**", "**None**", "**Versioned**" and "**Unversioned**".

If you check "**Ignore Unversioned&Unchecked**" all "unknown" files in the list that are unchecked will automatically be ignored by Git after commiting.

A right-click on items in the list gives you the option to perform certain functions to these elements or open the Log window for the file.
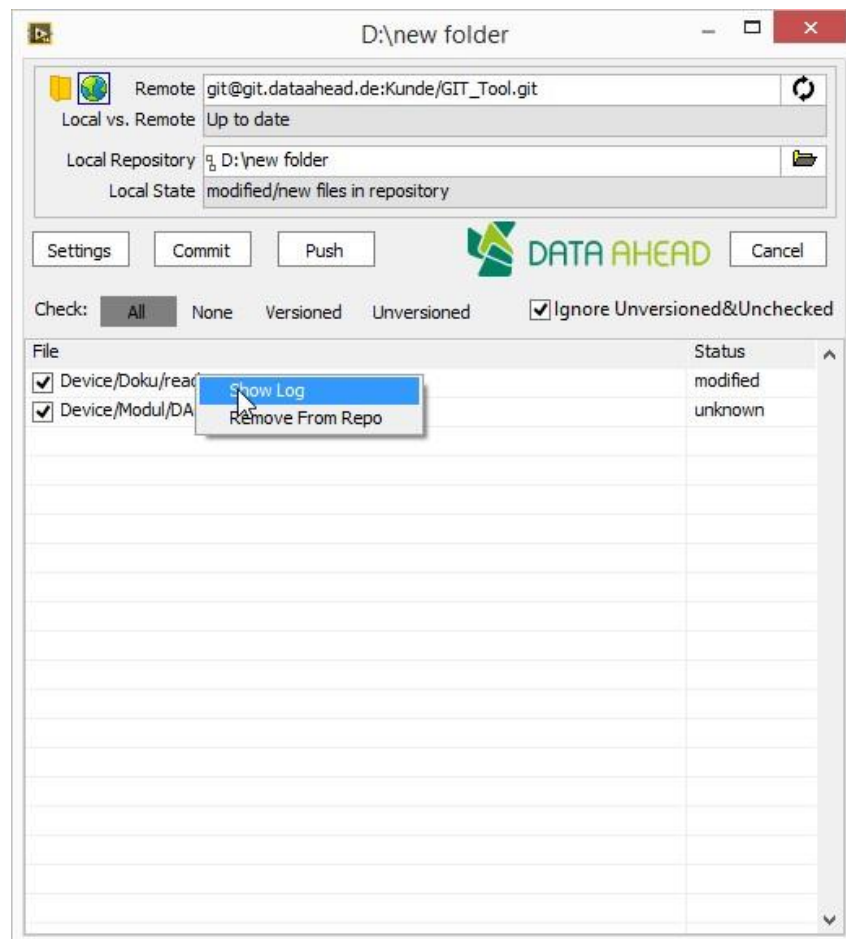

Figure 16: Git Menu with context menu for selected item

Tools Menu

git4G extends the LabVIEW Tools Menu by a **git4G** submenu with the following functions:

- **Git Menu:** Open the Git menu
- **Add & Commit All:** All files get committed. Files that are not yet part of the repository will be added before automatically.
- **Push:** Pushes from the local to the remote repository.
- **Pull:** Pulls from the remote to the local repository
- **Revert All:** Reverts all files on disk to the state of the last commit.
- **Remote Update:** Compares the lokal repository vs. the remote repository.
- **Clone:** Creates a local copy of a repository
- **Show log:** Shows the history of the project in the Log window.
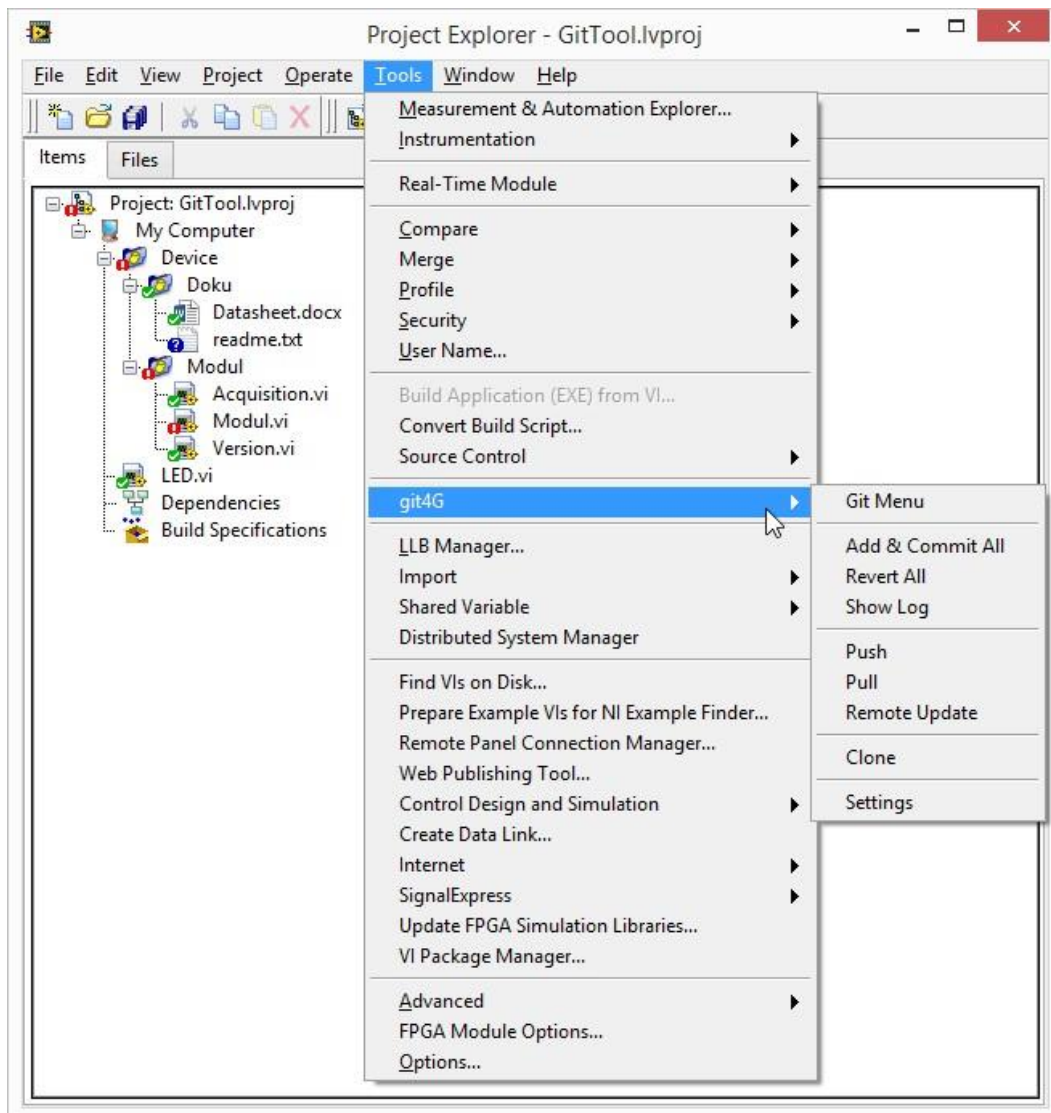- **Settings:** Open the **git4G** settings menu (page 31)

Figure 17: Tools Menu

Context Menu (Project Explorer Item)

git4G extends the context menu in the project explorer with Git functions. This context menu shows you the Git functions corresponding to the repository status/ current file status.

Project functionality:
- **Add & Commit All:** All unknown files will be added to the repository. Afterwards all changes will be committed.
- **Revert All:** Resets all modified files in the project to the state of the last commit.
- **Push:** Pushes from the local to the remote repository.
- **Pull:** Pulls from the remote to the local repository.
- **Show Log:** Shows the history of the project in the Log window.
- **Checkout All Ours/Theirs** (In case of a merge conflict)**:**Allows you to switch between your version of the project and the version that was modified from a different location.
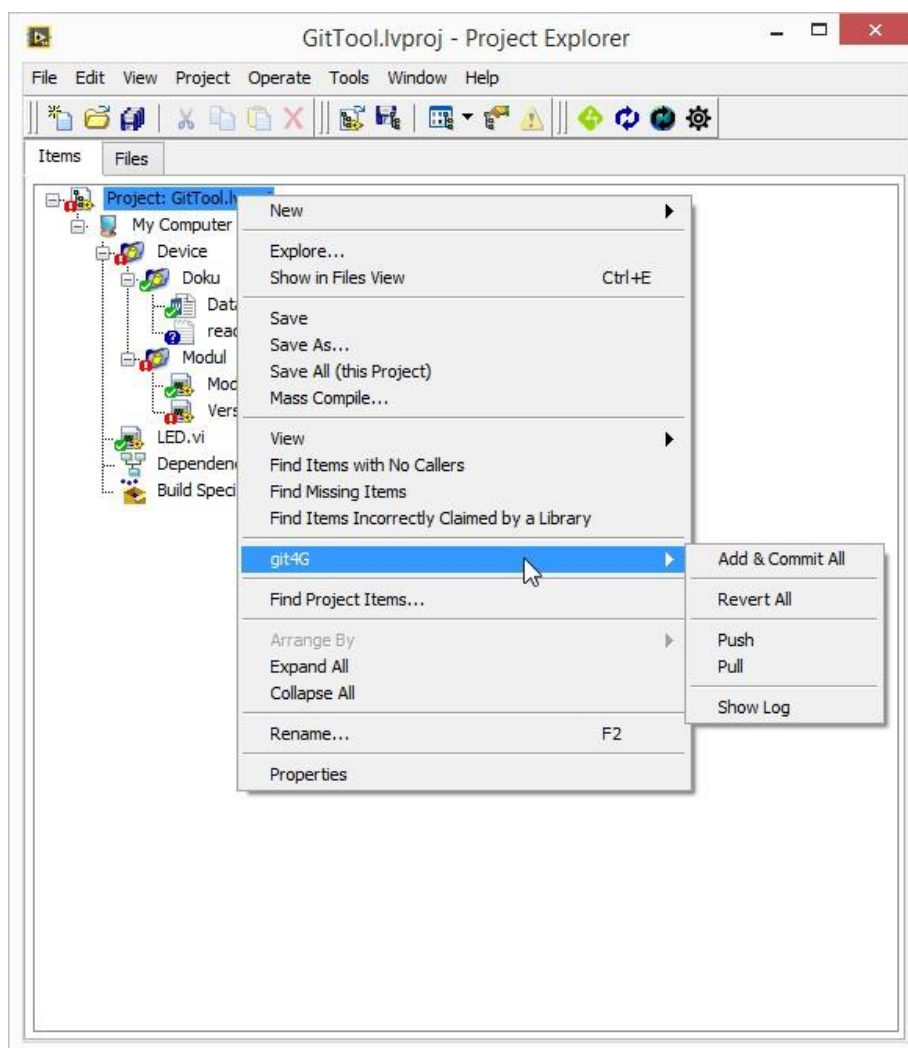

Figure 18: Project Pop-up Menu

Folder functionality:

- **Add All Items in Folder:** Adds all files in the selected folder to the local repository.
- **Commit All items in Folder:** Commits all files in the folder.
- **Revert All items in Folder:** Resets all files in the selected folder to the last commit.
- **Ignore Folder:** Ignores the whole folder with all files included.
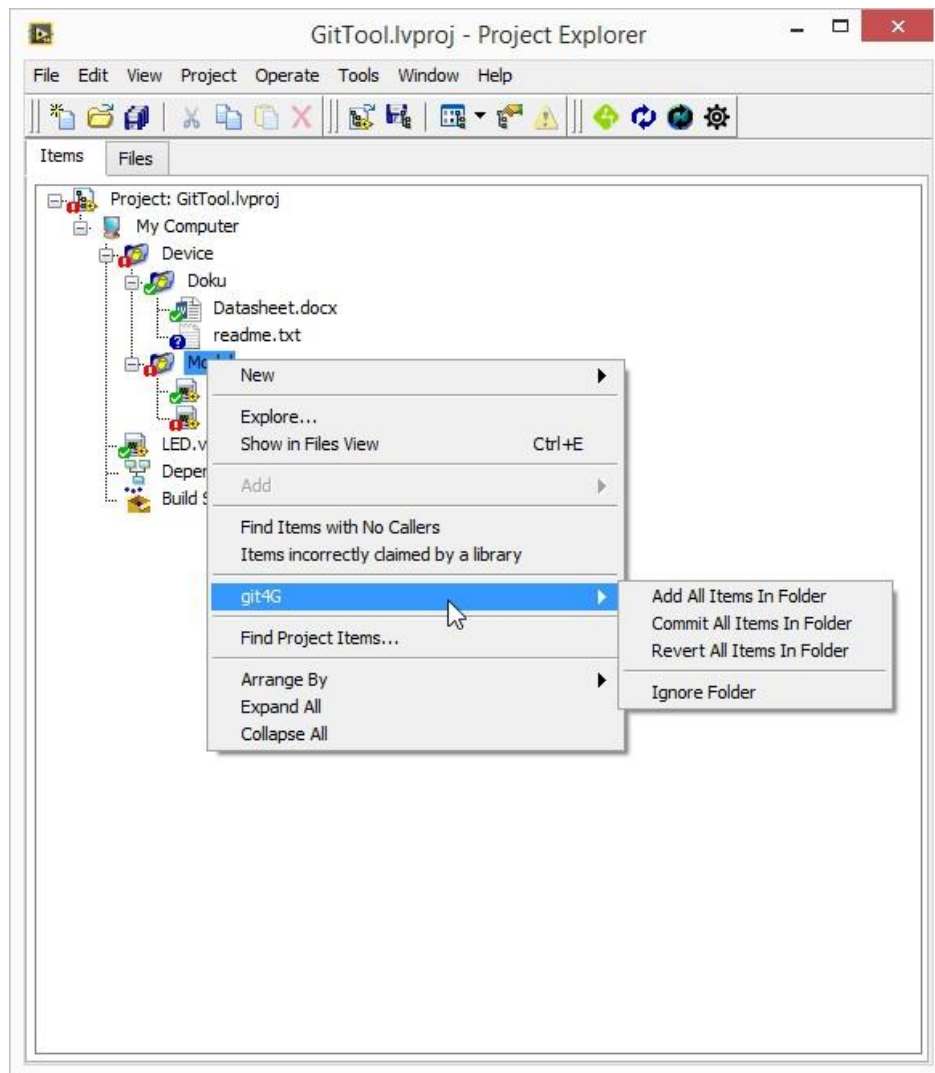

Figure 19: Folder Pop-up Menu

Single file functionality:

- **Add:** Adds the selected file to the local repository.
- **Ignore:** Adds the selected file to the .gitignore file.
- **Ignore Data Type:** Ignores all files of this data type in the entire repository.
- **Commit:** Commits the selected file.
- **Revert:** Resets the selected file to the state of the last commit.
- **Show Log:** Shows the history of the file in the Log window.
- **Compare** (LabVIEW Pro required)**:** In case of a merge conflict starts the LabVIEW Compare Tool to show differences between local and remote)
- **Merge** (LabVIEW Pro required)**:** In case of a merge conflict starts the LabVIEW Merge Tool to merge local and remote file version)
- **Checkout Ours:** In case of a merge conflict: switch to your version of the file
- **Checkout Theirs:** switch to the version of the file that was modified from a different copy of the repository
- **Resolve Conflict:** Resolves the merge conflict by using the selected file version
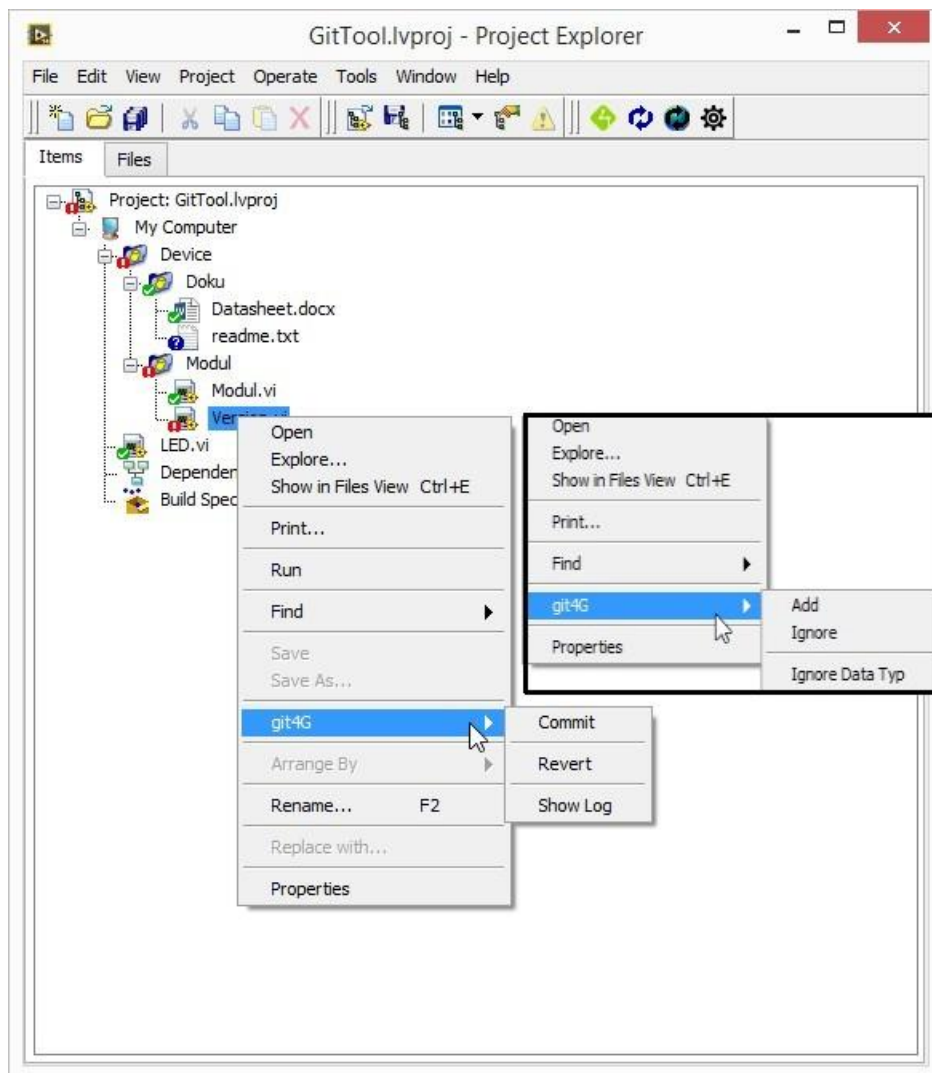

Figure 20: File Pop-up Menu

Commit Message

Every commit requires a specific commit message. Before every commit the "Commit Message" box pops up. Type in something which is important to know for this commit (see the internet for useful tips how to write a good git commit message).
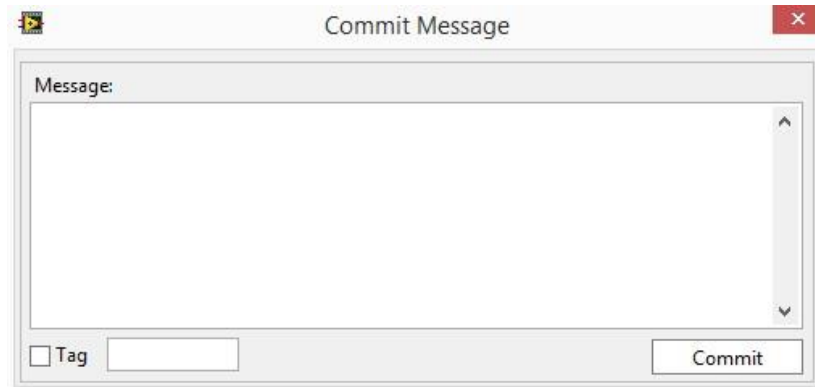

Figure 21: Commit Message

Tagging

git4G gives you the option to tag specific points in the project history as being important.

You can add a tag at the time of the commit:
Just check the box "Tag" (Figure 21: Commit Message) and type a few characters (e.g. version information) into the field right to that box.

Version.vi

git4G gives you the opportunity to generate commit messages from your documentation of your source code. For that the option "Check Version.vi" in the Settings needs to be activated.

If you have a **Version.vi** in your project, **git4G** reads the version string output and the part of the text field in the block diagram which corresponds to the version string.
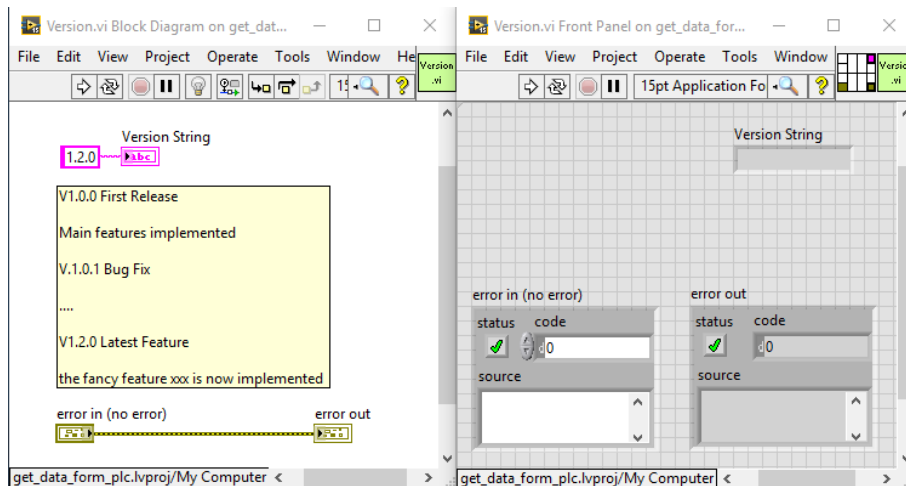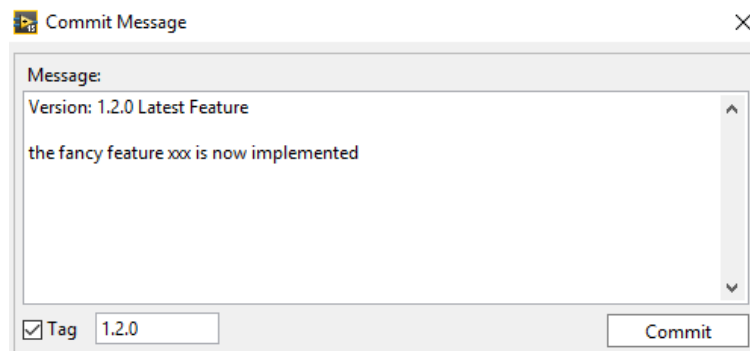

Figure 22: Version.vi example

Figure 23: Commit String and Tag generated from Version.vi

The **Version.vi** needs to meet the following requirements:
- Name: "Version.vi"
- String indicator output with version information
- Text field in block diagram with version information and details (optional)
- One of the following connector pattern:

Figure 24: Commit String and Tag generated from Version.vi

Clone

With **git4G** you can create a local copy of a repository using "Clone".
Go to "Tools" -> "git4G" -> "Clone"

Figure 25: Clone dialog

Type in a remote URL, choose a local folder ("Local Repository") and press the "Clone" button.

Log Window

With **git4G** it is possible to view the history of all commits for the whole project or for a single file.

There are two ways to open the Log Window for **project:**
- LabVIEW Tools menu -> git4G -> Show Log
- In Project Explorer right-click on the project -> git4G -> Show Log

In the top list, you can see the history of all commits. Select a commit to display the whole commit message (field below) and the details for all the files that have been changed (list on the bottom).

A right-click on the commit gives the option to save a zip-archive of the project in the state of that specific commit. It is also possible to add tags to specific commits.

A right-click on a file in the list on the bottom allows you to save a copy of that file in the state of the selected commit.
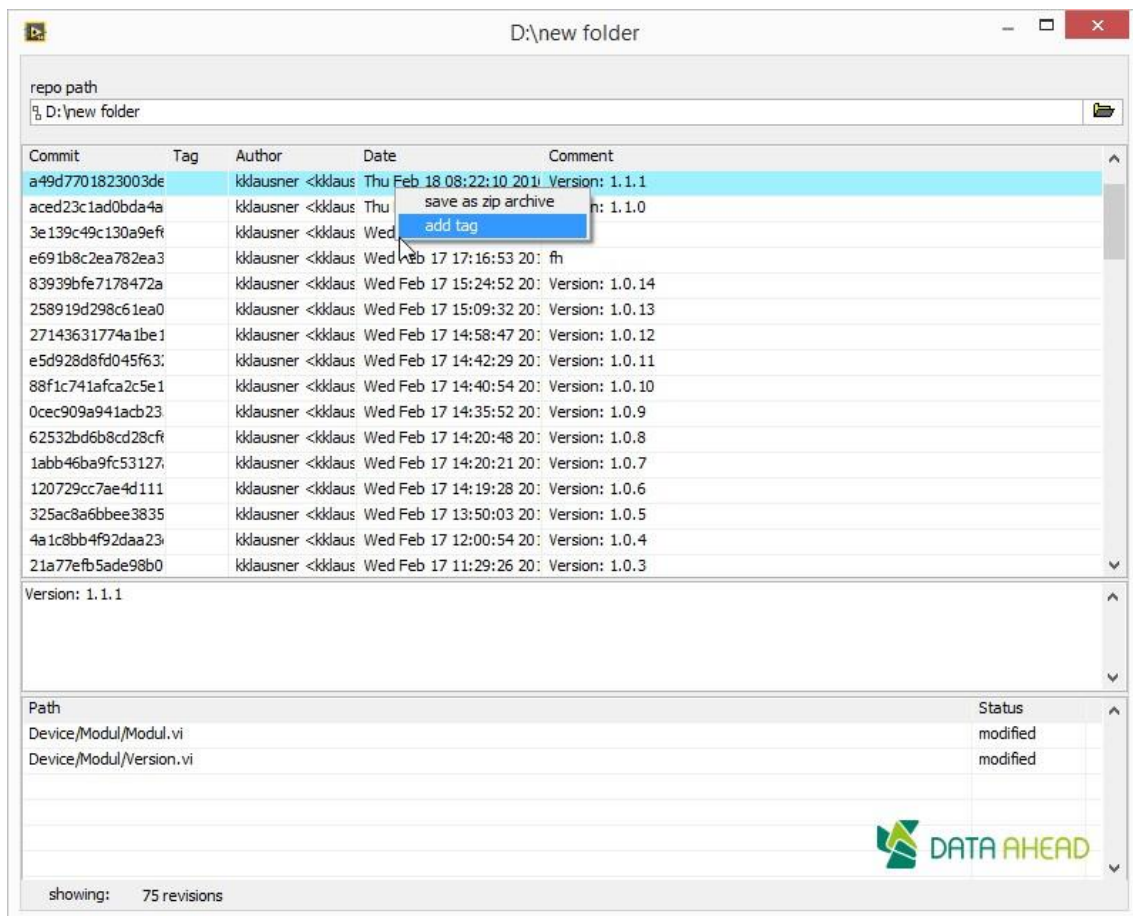

Figure 26: Log window for a project

If you want to open the Log window for a single file, you can right-click the file in the Project Explorer and select git4G -> Show Log or right-click on a file in the file list of the Git Menu.

Select a specific commit to see the details.

Right-click on a commit gives you the option to save a copy of that file in the state of that specific commit (**save file revision**).

If you are working with the **LabVIEW Professional Development System** you also have the option to compare the file in the state of that specific commit to the current version using the **LabVIEW Compare Tool** (**compare to current version**).
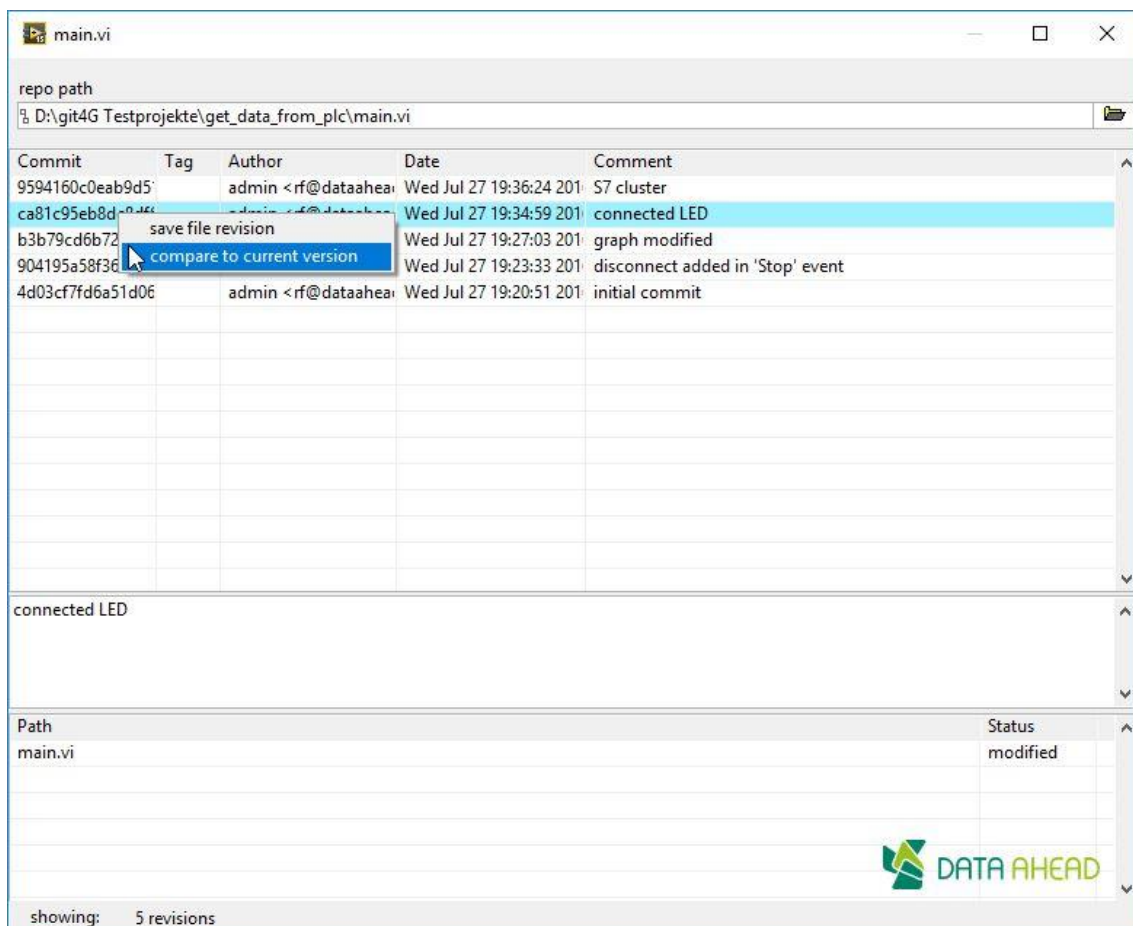

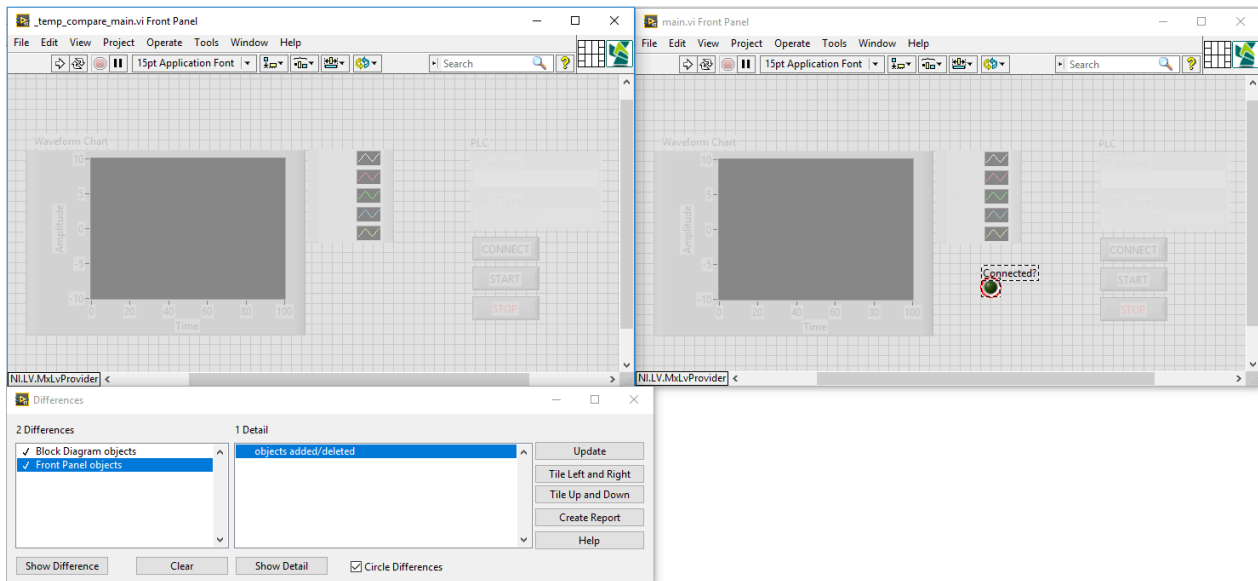**Figure 27: Log window for a single file**

**Figure 28: Start the LabVIEW Compare Tool directly from git4G
to compare the current file version with an older one**

See the LabVIEW Help for details about the LabVIEW Compare Tool.

Merge Conflict

If one (or more files) was modified at different local copies of a repository a merge conflict occurs. Git tries to merge text files but is not able to merge binary files. The user needs to decide, which version of a binary file to keep with or merge the files manually. If you are working with the **LabVIEW Professional Development System** you have also the option to merge the files using the **LabVIEW Merge Tool**.

If a merge conflict occurs **git4G** marks the file, the folder (if file is inside an auto-populating folder) and the project with the yellow overlay icons (Table 1: Overlay Icons, page 15). Additionally, the context menus for files in the file list of the Git menu and the **git4G** context menu for files in LabVIEW Project Explorer are getting entries to allow you to switch between the versions of a file, to compare or merge the files using the **LabVIEW Compare/Merge Tool** (if working with LabVIEW Professional Development System) and to resolve the conflict. The context menu for a folder allows you to switch between the versions of all files in the folder. With the context menu of the project you can switch between the versions of all files in the project (compare or merge only applies for single files).
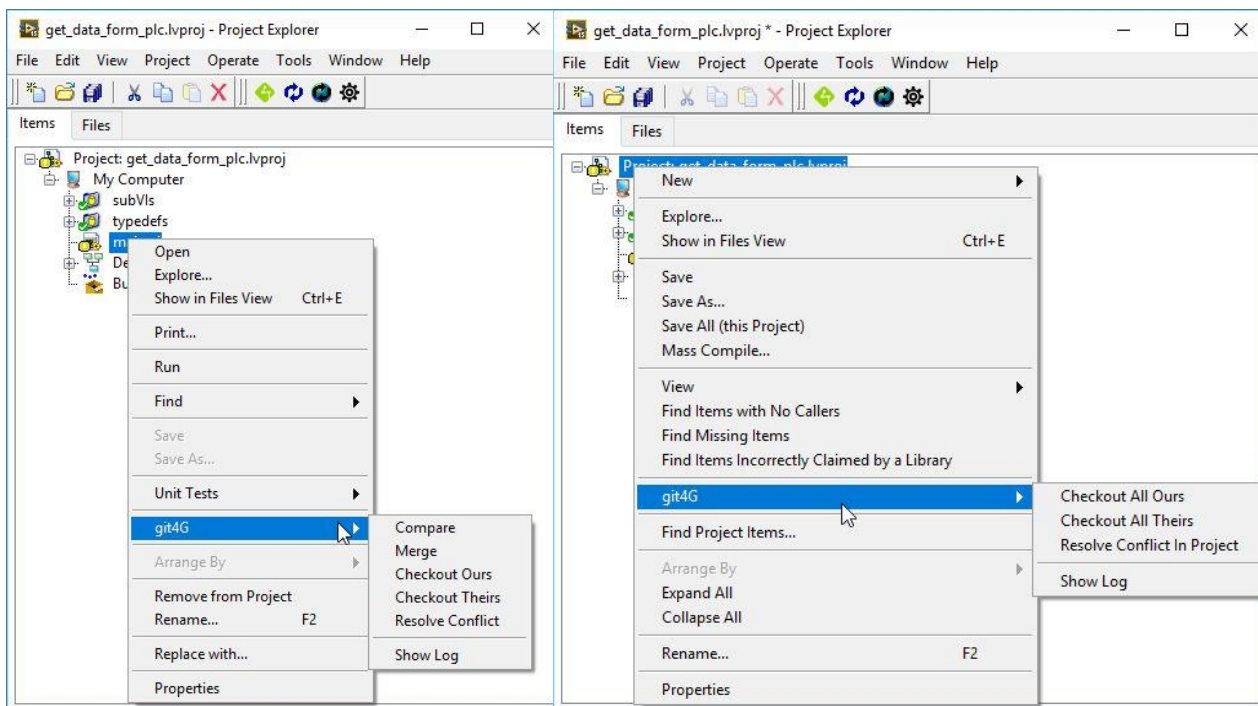

Figure 29: Context menu in case of a merge conflict, left: for a file, right: for the project

Swichting between the file versions:

- **Checkout Ours:** Copies the file version to the project, which was modified from your local repository.
- **Checkout Theirs:** Copies the file version to the project, which was modified from another copy of the repository.

Comparing the file versions using the LabVIEW Compare Tool:

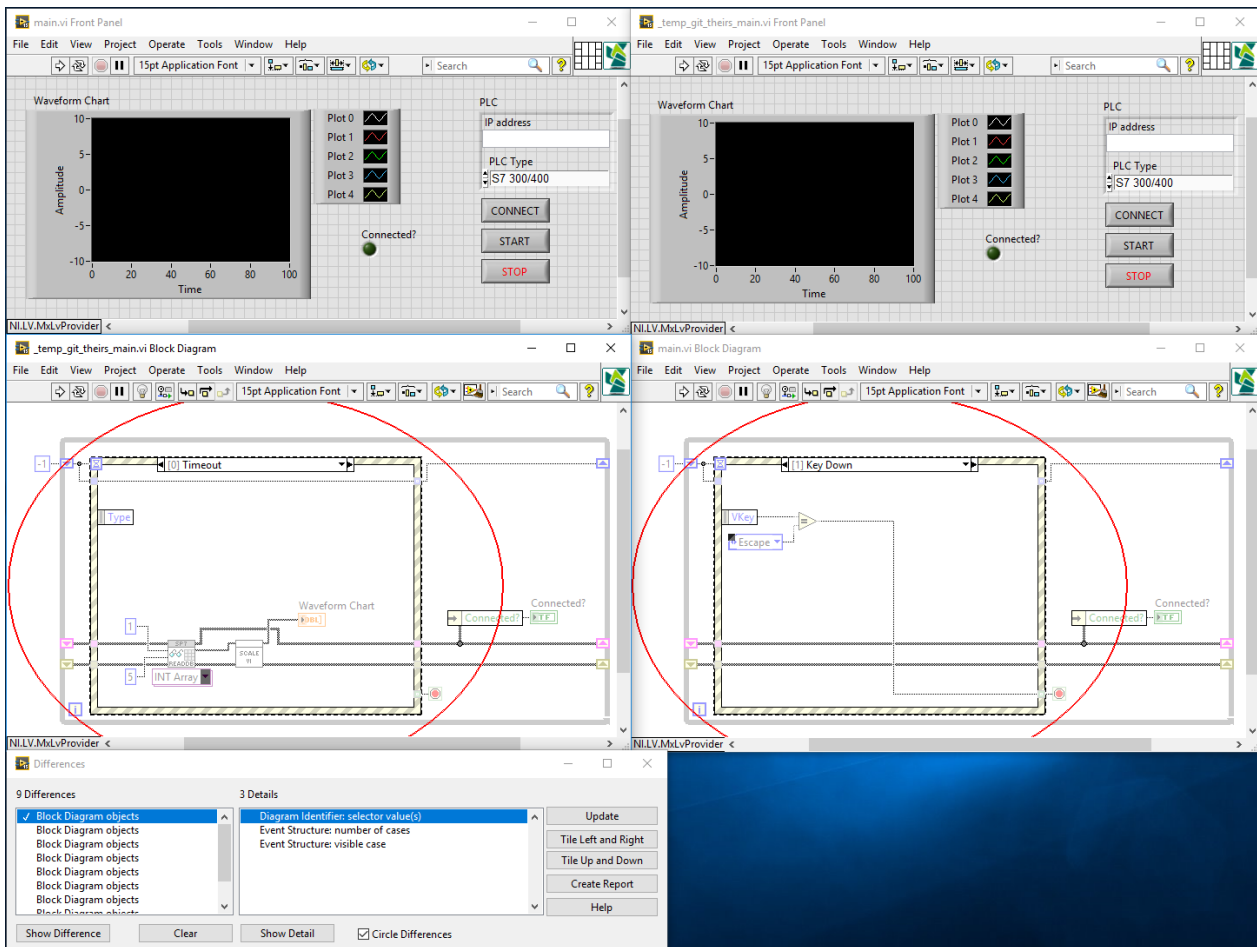- **Compare:** starts the LabVIEW Compare Tool with both versions of the file

Figure 30: LabVIEW Compare Tool directly started from git4G in case of a merge conflict

Merging the file versions using the LabVIEW Merge Tool:

- **Merge:** starts the LabVIEW Merge Tool with both versions of the file
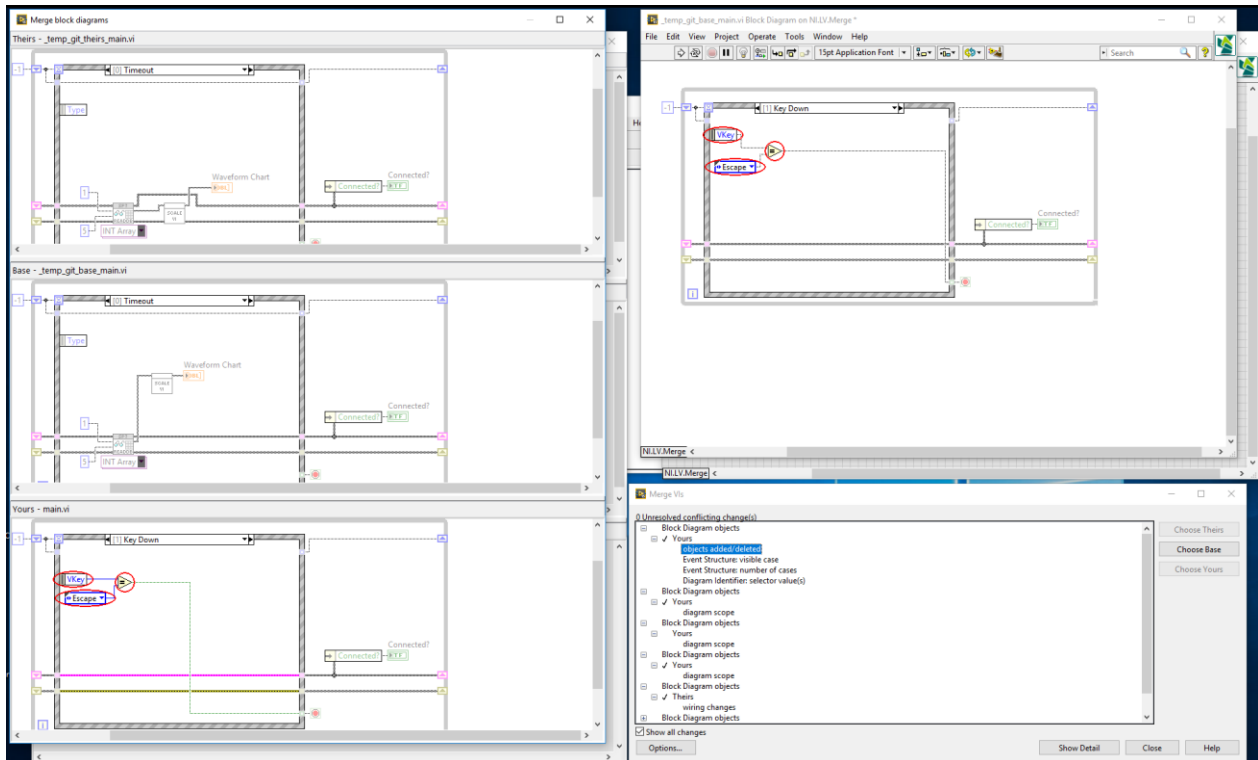

Figure 31: LabVIEW Merge Tool directly started from git4G in case of a merge conflict

See the LabVIEW Help for details about the LabVIEW Merge Tool.

Resolving the conflict:

Once you have chosen a file version or merged the file you need to resolve the conflict by choosing the context menu option:

- **Resolve Conflict**

Now a commit is required to get the repository clean.

Settings

There are three ways to open the settings menu:
- ⚙ button in the git4G toolbar
- Settings button in the Git menu
- LabVIEW Tools menu -> git4G -> Settings

The settings menu has a **git4G** and **Git** section:

git4G: here it is possible to switch on/off the "Periodic Status Check" and modify the update time. This could be helpful if you encounter performance issues. Especially for huge repositories with many new or modified files, the Git status check could cause performing issues.
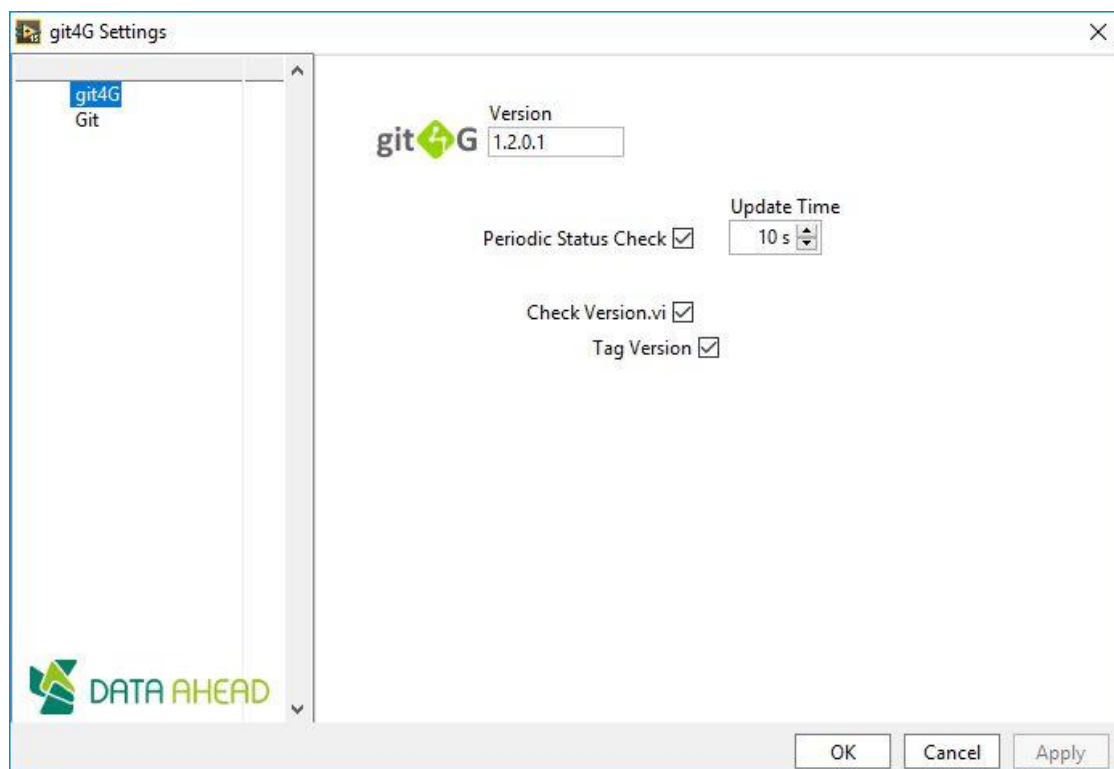
The current installed version of **git4G** is shown.

Figure 32: git4G Settings

In settings you can define if **git4G** should try to read the information from the Version.vi to be used for the commit message and for Tagging (**Check Version.vi** and **Tag Version**).

Git: Here it is possible to modify the global Git settings "Auto Convert CrLF" (we recommend unchecked), "User Name" and "User Email". The current installed version of Git will be shown.

If the settings menu was called from a project that is already under Git control, pressing the button "Open .gitignore" opens the belonging .gitignore file with the standard editor. This allows you to modify the file manually and e.g. remove entries in that file.
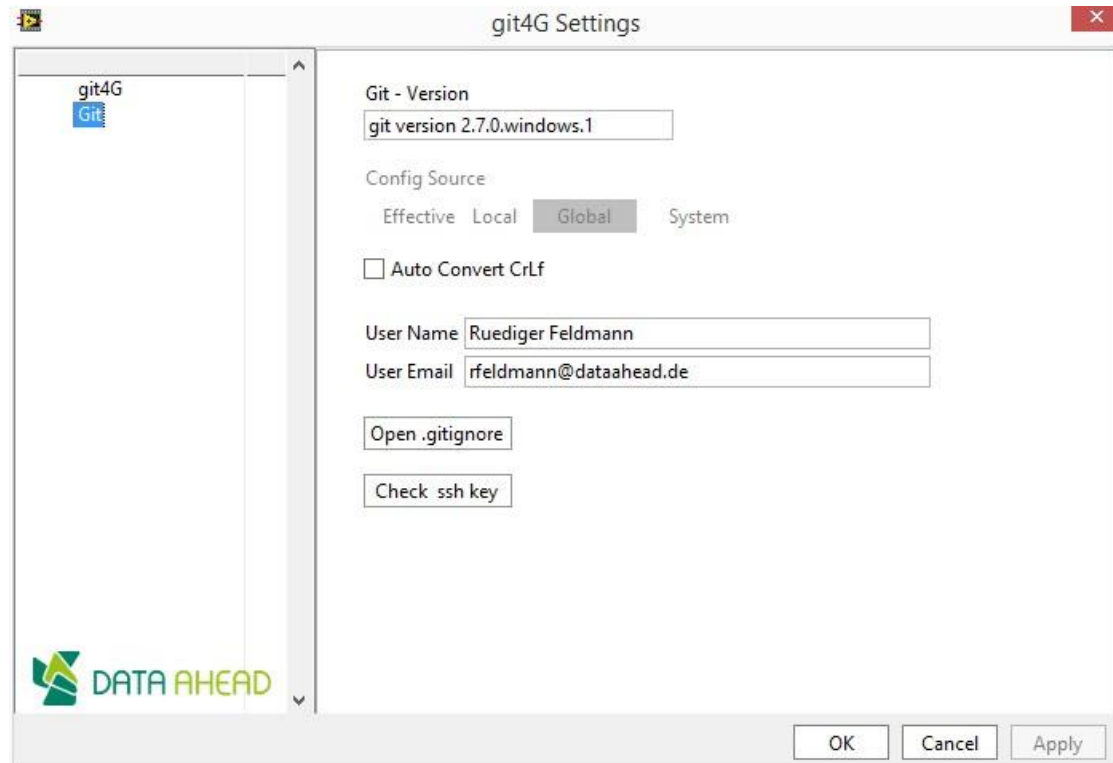

Figure 33: Git Settings

Additionally, you can check the SSH key (if using **git4G** with remote repositories on a server).

Check SSH key

To use **git4G** with a server (e.g. hosting service like GitHub or GitLab) a SSH key pair is needed to pull and push to/from the server. For authentication, the public key needs to be copied to the server.

By default **git4G** uses the a SSH key (id_rsa) inside the standard SSH key folder (\.ssh). If you want to create a new key pair, if you want to use a different key or if you want to copy the key to the clipboard to save it at the server press the "Check ssh key" button in the settings window. This button opens a window where the public key of the available SSH key pairs will be shown. If no key was previously selected **git4G** starts with the standard folder for SSH keys. If you want to search a different folder press the button "Browse to ssh key". If you want to select a different key use "Select ssh key".
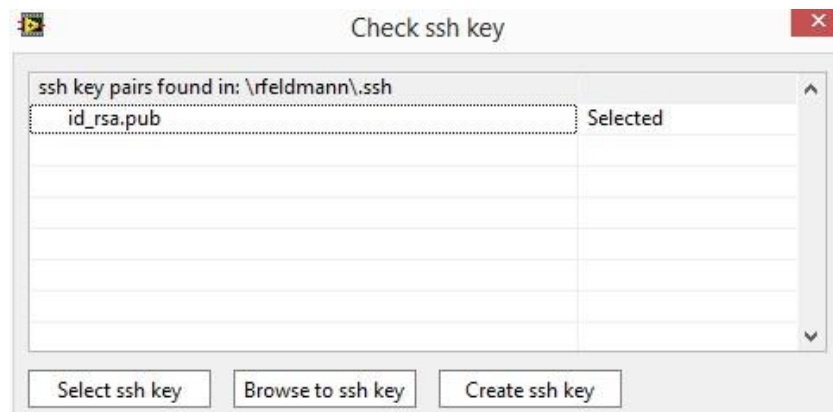

Figure 34: List shows available SSH keys

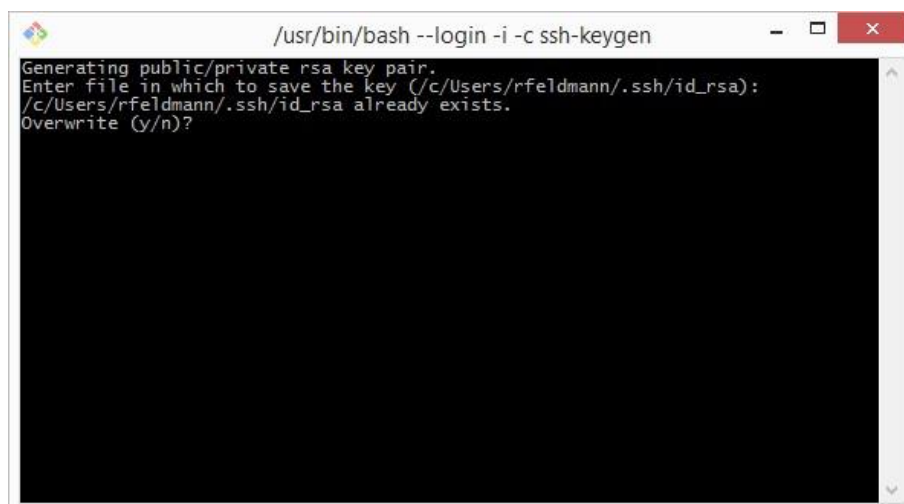If you want to create a new SSH key pair press "Create ssh key" and follow the dialog:

Figure 35: Dialog to create a new SSH key pair

Note: If you don't want to use a password with your SSH key just press the Return button when asking for a password. When using with a password every time the SSH agent starts you need to type in the password.

To show the content of the public SSH key and to get the option to copy the content to the clipboard double click on the SSH key in the list:
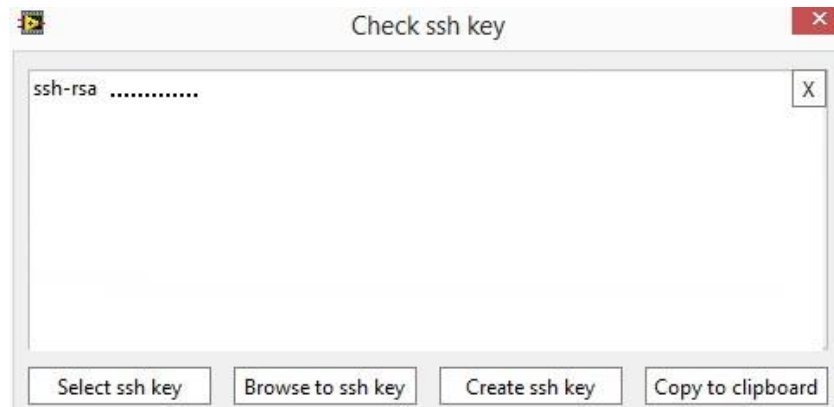

Figure 36: Content of the key (here substituted with …..) and the option to copy the string to the clipboard
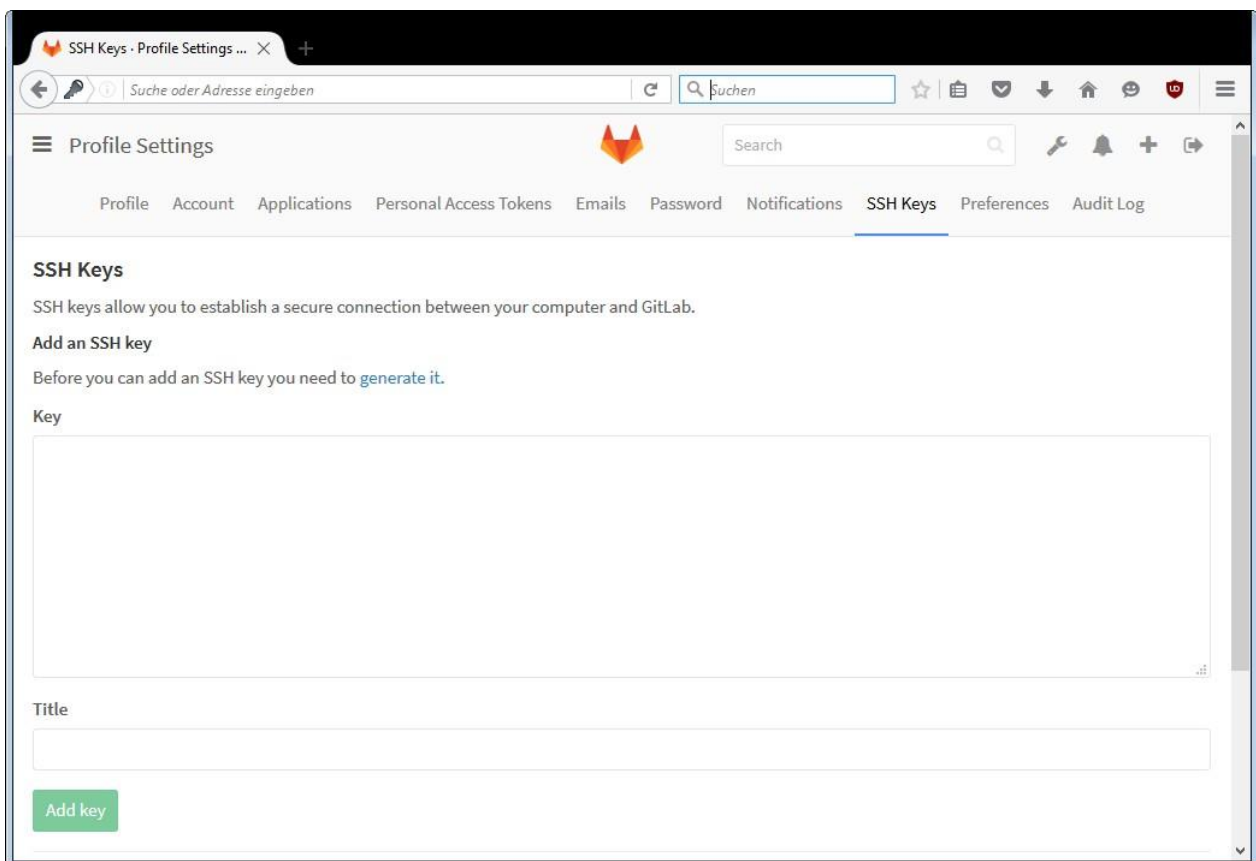

Figure 37: Example GitLab: Add the SSH key from clipboard in your Profile Settings

Support and Feedback

Please contact us at info@dataahead.de

For latest news and support on our toolkits, go to:

https://decibel.ni.com/content/groups/data-ahead-toolkit-support

DATA AHEAD GmbH
Prinzregentenufer 3
90489 Nürnberg
Germany

© Copyright 2016, DATA AHEAD