Git Collaboration Tools

FIRESPLASH
ENTERTAINMENT

Thank you for purchasing our Git Collaboration Tools V2 Asset.

We are actively developing this tool as we are also actively using it for our own projects. We think it has quite large potential but we need your thoughts about it so if you require any functionality which is not implemented or if something does not work as expected, leave us a note. We would love to hear your opinion! **Contact for all asset related inquiries: assets@firesplash.de**

If contacting us, please watch out for the automatic reply and if you don't get it, check your spam folder!

Currently the GitControl class exports a few static methods for public use. This is currently a WIP – We plan to officially support this in the future so you can create your own scripted behavior. Feel free to experiment with it (in editor scripts) and share your thoughts with us.

Lastly we want to request you to rate this asset on the store once you got familiar with it! You are doing us a great favor with leaving a review.

Git itself is a piece of software with great abilities and lots of configuration. This asset is aiming to provide you easy access to the daily development tasks and to assist you in keeping on track with your team. It does not and does not aim to replace the git console entirely.

As of the lots of features and options, the first configuration of this asset might be a bit harder than you would expect but this highly depends on your needs and on how far you are already working with git.

Mainly we want to make the all-day developer life easier for people already using git, thus most likely already covering 80% of the preparation work.

Beginning with Version 1.1 we are officially supporting MacOS. There may be minor restrictions, though and unfortunately we can't test every single change against this OS. If you happen to encounter any issues on MacOS, please report them to us. Thank you!

Git Collaboration Tools

Inhalt

Git Collaboration Tools

Quick installation and setup guide

This guide does not explain everything step by step and does not regard a lot of options.
If this quick guide does not match for you, scroll tot he next page to iterate through the full instrucion set.

Step 1: Preparation

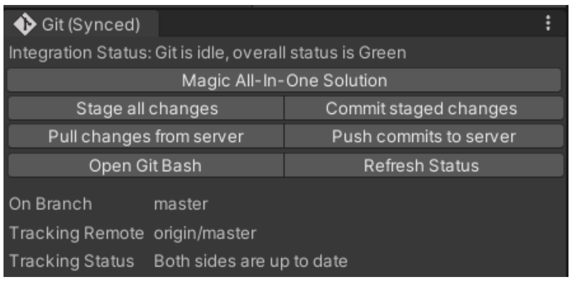
1. Install Git for windows
    a. Install Components: LFS, Explorer integration (Git Bash)
    b. Select Nano (recommended)
    c. Select Git from command line and 3rd party
    d. Use windows secure channel library
    e. Checkout and commit as-is
2. If you do not already have a SSH-Key installed at the default location (Usually at C:\Users\<YourUsername>\.ssh\id_rsa) create one by starting git bash and entert he command `ssh-keygen` and do **NOT** set a passphrase!
3. Create an empty repository at your preferred git hoster (e.g. GitLab or Github)
4. Configure your git hoster to accept your (probably newly created) SSH-Key (Get the public key using `cat ~/.ssh/id_rsa.pub` in your Git Bash)
5. Create a new Unity Project

Step 2a: Clone the repository – **Only for existing, filled repositories**

If you already got a project in an online repository, you can now use Git Bash to clone your project:

```
git clone <Repository-Address>
```

Then head over to the Unity Hub, Add the existing project and open it. You can then install our asset from the package manager, open the Git Panel and it should automatically detect that it is a git clone.



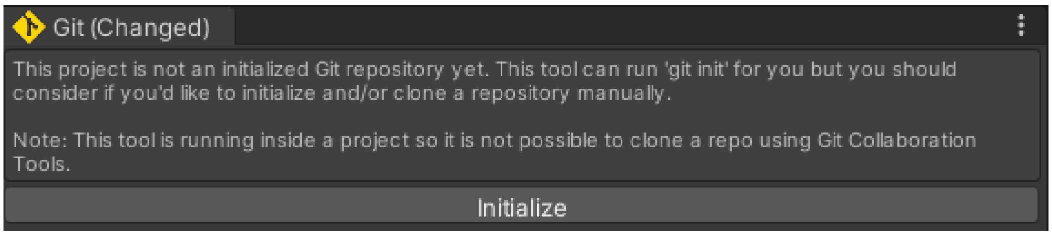
Integration Status: Git is idle, overall status is Green

On Branch	master
Tracking Remote	origin/master
Tracking Status	Both sides are up to date

Depending on your .gitignore file, you might see a different status (untracked changes)
You might probably want to recreate the .gitignore file based on the template:
Tools -> FSE Git Collaboration -> Install a standard Git-Ignores file

Step 2b: Upload the new project – **Only for empty remote repos!**

Open the „Git“ Panel (eighter under Window or under Tools), which should look like this now:



This project is not an initialized Git repository yet. This tool can run 'git init' for you but you should consider if you'd like to initialize and/or clone a repository manually.

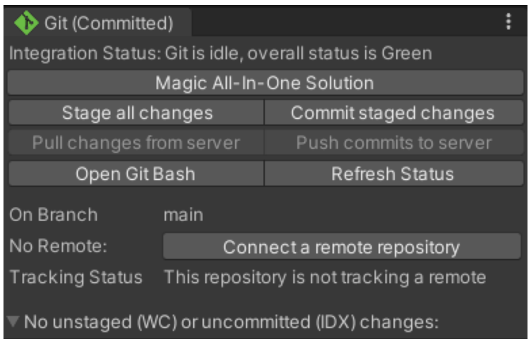
Note: This tool is running inside a project so it is not possible to clone a repo using Git Collaboration Tools.

Hint: If you imported the tool into an existing project which is already using git, it will detect that.

Click the **Initialize button** and then – before committing – add a .gitignore file by clicking **Tools -> FSE Git Collaboration -> Install a standard Git-Ignores file**

Now click the „**Magic All-In-One Solution**“ button to make a first commit**. Enter a short description** „first commit“ and **confirm the dialog**. It will throw an error that it could not push. This is because it does not know the remote repo yet. **Click OK.**

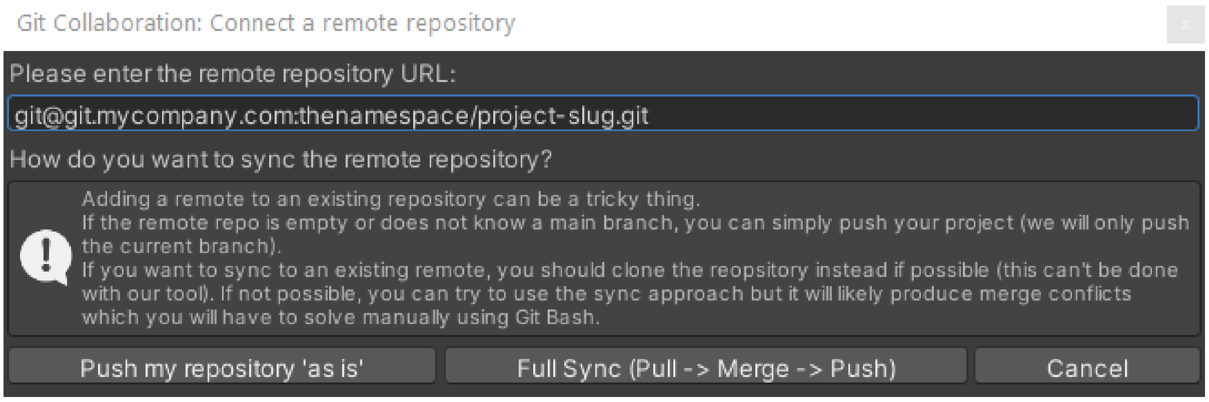
Now the Git panel should look like the righthand screenshot ->



Integration Status: Git is idle, overall status is Green

On Branch main
No Remote:
Tracking Status This repository is not tracking a remote
No unstaged (WC) or uncommitted (IDX) changes:

Next we need to click the „Connect a remote repository“ button to push our project. After clicking, a dialog will open up. Specify the repository address (SSH format!) and click the first button to push the entire branch tot he remote repository:


Git Collaboration: Connect a remote repository
Push my repository 'as is'
Full Sync (Pull -> Merge -> Push)
Cancel

Please enter the remote repository URL:

git@git.mycompany.com:thenamespace/project-slug.git

How do you want to sync the remote repository?

Adding a remote to an existing repository can be a tricky thing.
If the remote repo is empty or does not know a main branch, you can simply push your project (we will only push the current branch).
If you want to sync to an existing remote, you should clone the reopsitory instead if possible (this can't be done with our tool). If not possible, you can try to use the sync approach but it will likely produce merge conflicts which you will have to solve manually using Git Bash.

Depending on your project size (you created this one freshly, right?) after a few seconds, you will get a success dialog.

Congratulations! Your project is now linked to the remote repo.

If you need to download it on other computers (or re-downlaod it here at a later time), use Step 2a instead.

Step 3: All-Day-Work

How you utilize the features is strongly dependant on your own preferrences. We suggest to commit often and push when a piece of work is in a finished state as in „should be working“.

The workflow is always Work -> Stage -> Commit (-> probably pull) -> Push -> Work…

This can be done in a single click: The „Magic All-In-One Solution“ goes through all those steps.

Also watch out to the icon color and the name of the Git Panel itself. It informs about the overall project state. You will notice a few other useful features like Project-Tree icons and context menu options during usage. **Have fun using it!**

Git Collaboration Tools

The long version… Preparation and Setup

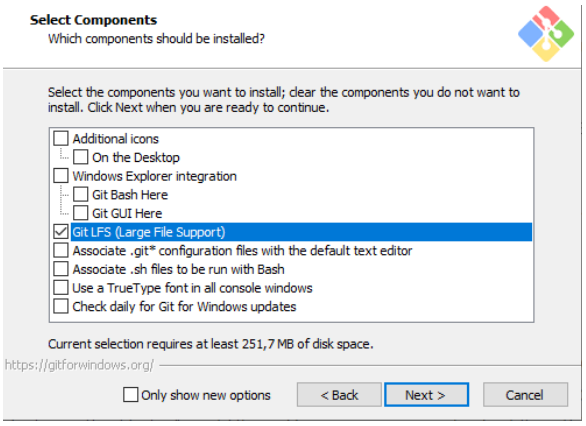
Installing Git for Windows

For the tool to work correctly you need to have Git for Windows installed. You can download it here:

https://git-scm.com/download/win

Hint: The installer is rapidly evolving so it might look a bit different but most important options should be the same. For using with Unity we suggest to install with **at least** following options:

Minimal



Select Components
Which components should be installed?

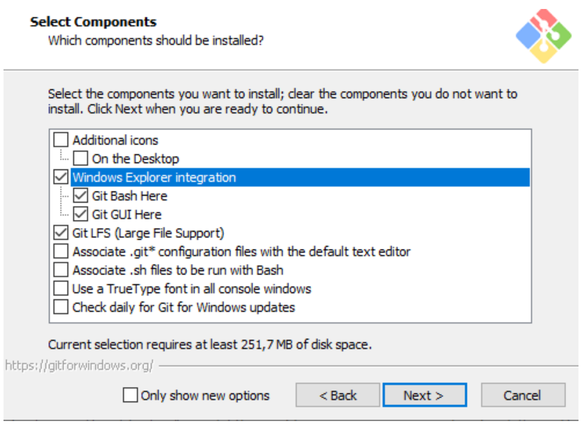
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [ ] Additional icons
  - [ ] On the Desktop
- [ ] Windows Explorer integration
  - [ ] Git Bash Here
  - [ ] Git GUI Here
- [x] Git LFS (Large File Support)
- [ ] Associate .git* configuration files with the default text editor
- [ ] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 251,7 MB of disk space.

https://gitforwindows.org/

Optimal



Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

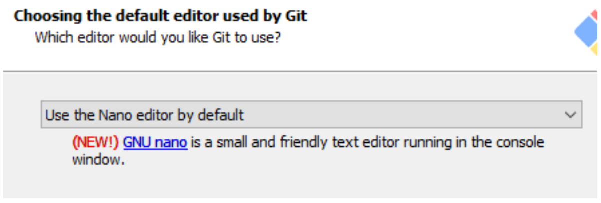
- [ ] Additional icons
  - [ ] On the Desktop
- [x] Windows Explorer integration
  - [x] Git Bash Here
  - [x] Git GUI Here
- [x] Git LFS (Large File Support)
- [ ] Associate .git* configuration files with the default text editor
- [ ] Associate .sh files to be run with Bash
- [ ] Use a TrueType font in all console windows
- [ ] Check daily for Git for Windows updates

Current selection requires at least 251,7 MB of disk space.

https://gitforwindows.org/

Having the explorer extension makes it a lot easier to clone repositories as this cannot be done from within Unity.

In the next step you should select an editor of your choice for using within Git for Windows.
You will not need this in your daily work as our Asset will provide you with an editor, however if you happen to need to investigate a merge conflict or want to manually commit, you will need it.



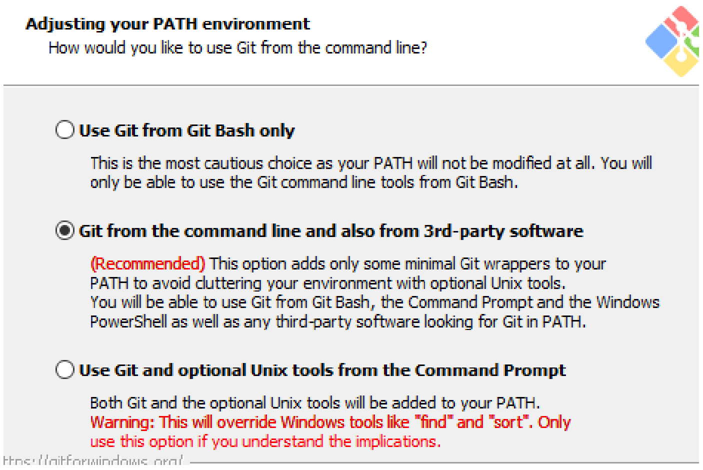
Choosing the default editor used by Git
Which editor would you like Git to use?

Use the Nano editor by default

(NEW!) GNU nano is a small and friendly text editor running in the console window.

We suggest GNU Nano as ist quite easy to use:

For the next question you should setup to use git tools **from command line and 3rd party software.**



Adjusting your PATH environment
How would you like to use Git from the command line?

○ **Use Git from Git Bash only**

This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.

● **Git from the command line and also from 3rd-party software**

(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.

○ **Use Git and optional Unix tools from the Command Prompt**

Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.

https://gitforwindows.org/

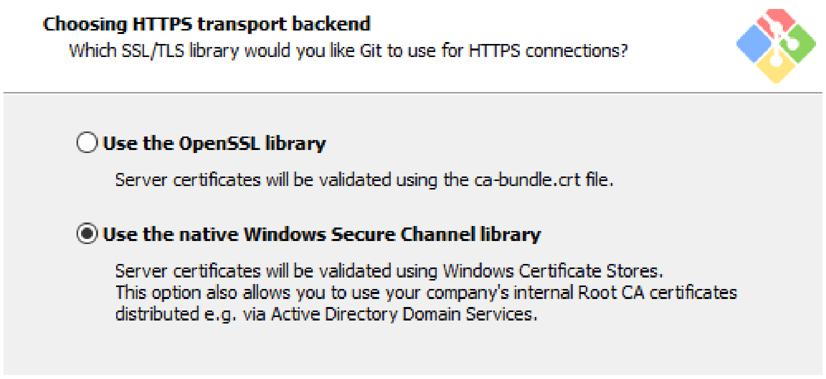
The Unity Asset is currently not using the PATH variable for discovery but you might have other tools which can utilize the git tools so it does not hurt. *However you **could** also select option 1.*

Attention!

Depending on your Computer setup, as a next step you might get asked which SSH executable you like to use. OpenSSH is usually a good choice but if you have a Key Agent like Pageant or KeeAgent you will need to use „Plink“.

You must setup SSH Key Authentication to work with our Git Collaboration asset. This will work using OpenSSH if you want to use the systems default key. *If you do not know what this is all about -> Use OpenSSH.*

In the next step you will be asked which HTTPS Transport backend you want to use. You should use the windows native library as you will always have up to date certificates aligned with your computer settings. OpenSSL keeps ist own trusted store which can be complicated to maintain, especially in an internal network using a local PKI or self-signed certificates.



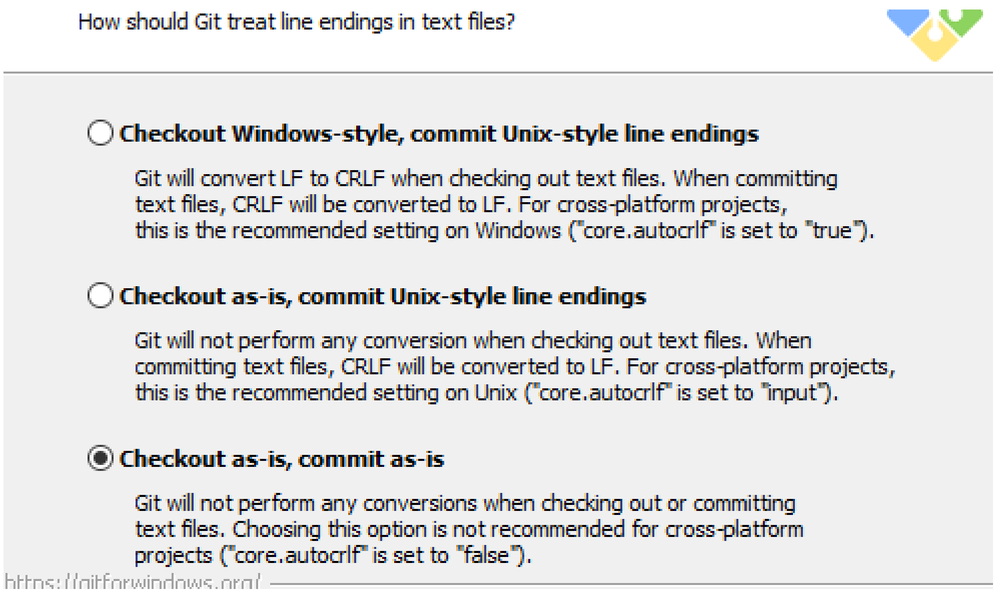
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?

○ **Use the OpenSSL library**
Server certificates will be validated using the ca-bundle.crt file.

[✓ ● **Use the native Windows Secure Channel library**]
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.

The next one is a very important step!

You will now be asked to chose the line ending conversion settings. If you select options 1 or 2 you will risk damage of some unity yaml files. **Use Checkout as-is, commit as-is** to prevent this.



How should Git treat line endings in text files?

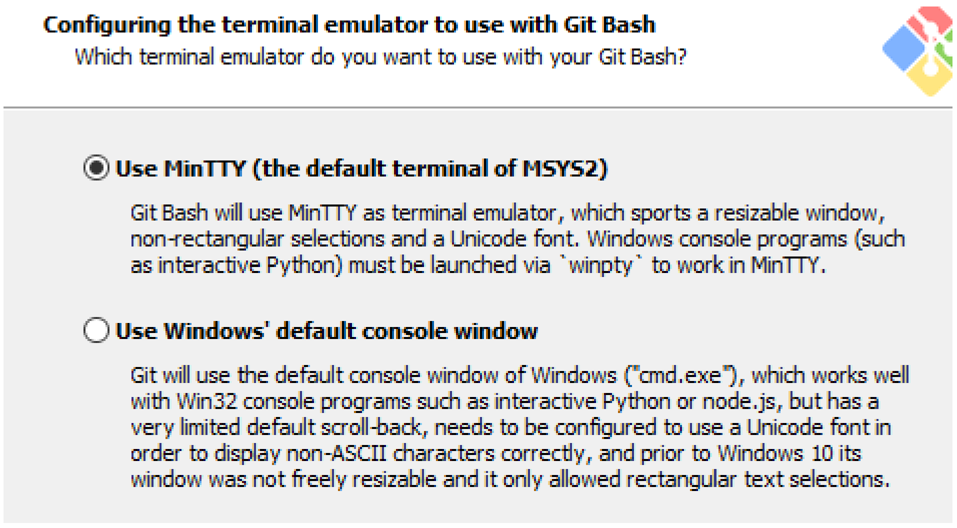
○ **Checkout Windows-style, commit Unix-style line endings**
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").

○ **Checkout as-is, commit Unix-style line endings**
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").

[✓ ● **Checkout as-is, commit as-is**]
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").

https://gitforwindows.org/

The terminal emulator does not matter for our asset but we suggest using MinTTY as it looks better with Git Bash.


Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?

[✓ Use MinTTY (the default terminal of MSYS2)]

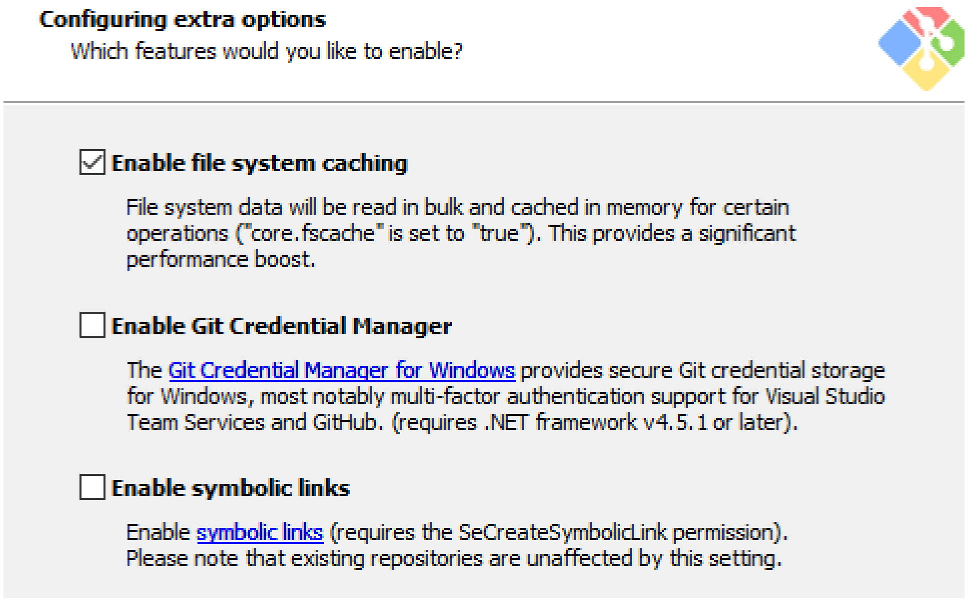
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.

Use Windows' default console window

Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.

Officially our asset does only support SSH Key authentication and User/Password authentication. However Git Credential Manager could probably solve issues if you can't use the default scenarios. This is experimental and untested but you can try and report back if it works.

Our officially supported config is the following:


Configuring extra options
Which features would you like to enable?

[✓ Enable file system caching]

File system data will be read in bulk and cached in memory for certain operations ("core.fscache" is set to "true"). This provides a significant performance boost.

Enable Git Credential Manager

The Git Credential Manager for Windows provides secure Git credential storage for Windows, most notably multi-factor authentication support for Visual Studio Team Services and GitHub. (requires .NET framework v4.5.1 or later).

Enable symbolic links

Enable symbolic links (requires the SeCreateSymbolicLink permission). Please note that existing repositories are unaffected by this setting.

After this dialog the installation will run.

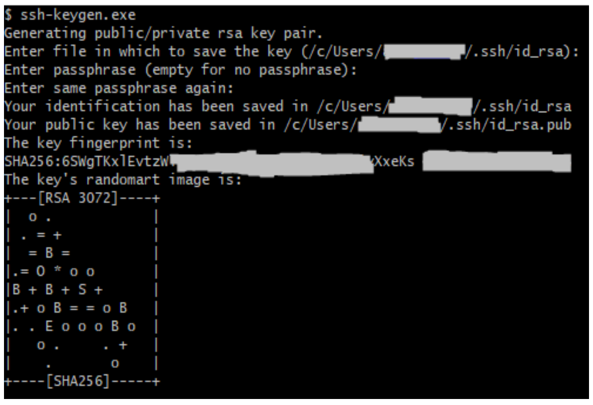
You should now have successfully installed Git for Windows.

Git Collaboration Tools

Create a SSH Key (if you don’t have one installed)

If you have just installed Git with OpenSSH and you do not know about SSH keys we will show you now, how to generate such a key:

Open Git Bash from your start menu.



```
$ ssh-keygen.exe
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/[redacted]/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/[redacted]/.ssh/id_rsa
Your public key has been saved in /c/Users/[redacted]/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:6SWgTKxlEvtzW[redacted]XxeKs [redacted]
The key's randomart image is:
+---[RSA 3072]----+
|  o .            |
| . = +           |
|  = B =          |
|.= O * o o       |
|B + B + S +      |
|.+ o B = = o B   |
|. . E o o o B o  |
|   o .     . +   |
|    .        o   |
+----[SHA256]-----+
```

In the window, which has just opened, enter **ssh-keygen.exe** and press enter.

You will be asked where to save the key. **Git will always use this default location so we need to store it there.** *There are options to change this behavior but this is complicated.*
You may not enter a passphrase! This will not work as we need to have unattended access to the authorization key as our asset is checking for changes in the repository on a regular base.

Warning: If you already have a key in place at this location cancel the operation and use that instead! You can find out the public key in the file `/c/Users/<YourUserName>/.ssh/id_rsa.pub`

Configure your Repository Server to accept the key

Now you need to setup your git repository to authenticate you using this key file. Depending on the provider this can be done in various ways.

GitHub: https://help.github.com/en/github/authenticating-to-github/adding-a-new-ssh-key-to-your-github-account

GitLab.com: https://docs.gitlab.com/ee/ssh/#adding-an-ssh-key-to-your-gitlab-account
Note: you do not need to run commands to find your key. It is written down here:
Using Git Bash: `/c/Users/<YourUserName>/.ssh/id_rsa.pub`
`Using Windows Explorer: c:\Users\<YourUserName>\.ssh\id_rsa.pub`

GitLab On Premise: Exactly like GitLab.com but login to your hosted instance instead 😊

There are various services and various ways to configure the key based authentication. Refer to the documentation of the service you are using.

OPTIONAL: Configure Git to use a non-default located SSH key

This is an advanced topic. If you do not know, what you are doing, don't.

GitLab gives an idea on how you can setup the key location if you need multiple keys:
https://docs.gitlab.com/ee/ssh/#working-with-non-default-ssh-key-pair-paths

However – We cannot support you on configuring this. **Our asset supports every constellation where git.exe can automatically log in to your repository** so when you run Bit Bash and are able to clone, push and pull without entering credentials, you are good to go.

OPTIONAL: Configure git to use a SSH-Key provided by Pageant (or a similar agent)

This is a hacky, very advanced topic! It works, but we can't provide support.

Yo achieve this, you need to replace git's default SSH-Implementation with a variant that supports agent-provided keys. Putty brings „plink" which does that for you.
Install it from here: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

You should use the full PuTTY MSI or directly download the plink.exe

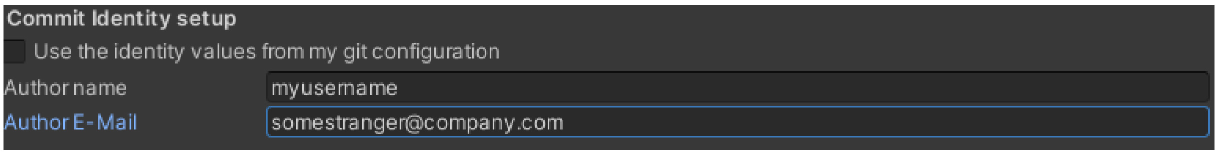
Now you can eighter set an environment variable called „GIT_SSH" to the path to plink.exe or use the settings window of our asset to set this only for our asset.

Note: You must use putty and connect tot he repository server once and accept the remote host key. If you do not do that, git will error out saying the host key is not trusted. It is not possible to accept the key using git

Configuring the asset to your needs

Our asset can be configured to your needs. You can find the settings in the editor menu under Tools -> FSE Git collaboration -> Plugin Settings

Basically our asset controls your locally installed git client. It will inherit all settings made with git config. In some cases you might want to change some portions like being able to use a different identity for unity commits. You can disable inheriting the identity by unchecking the first checkbox which will reveal two new settings:

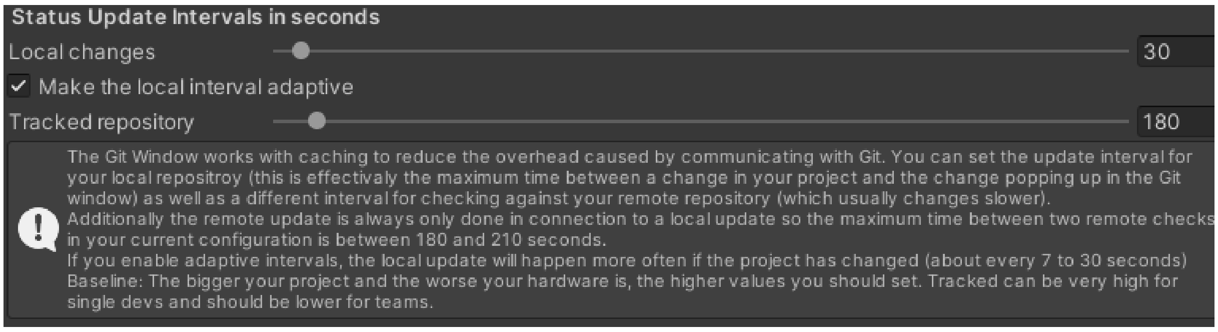


Commit Identity setup

☐ Use the identity values from my git configuration

Author name	myusername
Author E-Mail	somestranger@company.com

To provide you with a current change list as well as directing you when for example your team has changed the project on the server side, the system updates the local and remote repository states from time to time automatically. You can tune the intervals to your needs:



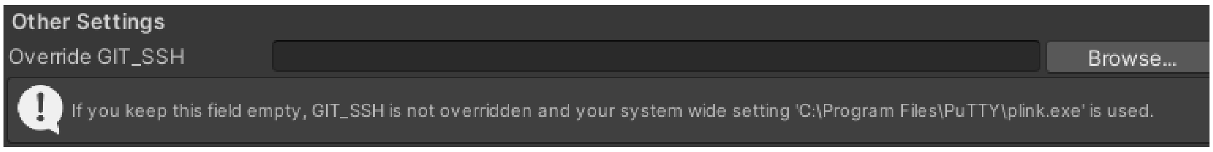
Status Update Intervals in seconds

Local changes	30
✓ Make the local interval adaptive	
Tracked repository	180

The Git Window works with caching to reduce the overhead caused by communicating with Git. You can set the update interval for your local repositroy (this is effectivaly the maximum time between a change in your project and the change popping up in the Git window) as well as a different interval for checking against your remote repository (which usually changes slower).
Additionally the remote update is always only done in connection to a local update so the maximum time between two remote checks in your current configuration is between 180 and 210 seconds.
If you enable adaptive intervals, the local update will happen more often if the project has changed (about every 7 to 30 seconds)
Baseline: The bigger your project and the worse your hardware is, the higher values you should set. Tracked can be very high for single devs and should be lower for teams.

Have an eye on the help box which explains the actual intervals based on your configuration.

You can also override the GIT_SSH environment variable. This is handy if you want to use plink or a similar tool (maybe even an intermediate script) to allow using an agent-provided SSH-Key. https://stackoverflow.com/questions/2985074/configure-git-to-use-plink



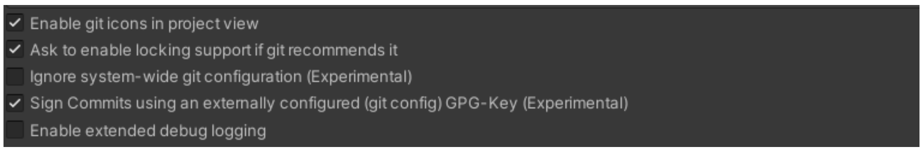
Other Settings

Override GIT_SSH

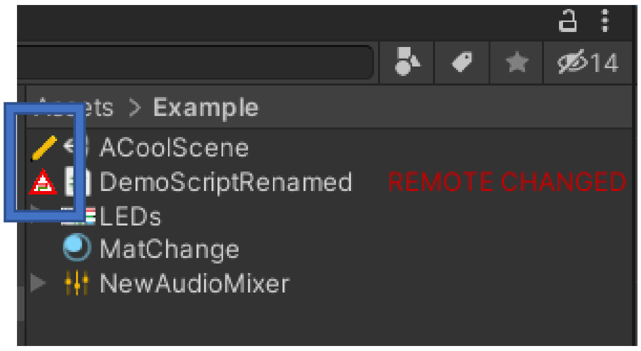
If you keep this field empty, GIT_SSH is not overridden and your system wide setting 'C:\Program Files\PuTTY\plink.exe' is used.

If you already set your git up to work as expected, there is no need to set this.

There are a few other/advanced options available:



- [✓] Enable git icons in project view
- [✓] Ask to enable locking support if git recommends it
- [ ] Ignore system-wide git configuration (Experimental)
- [✓] Sign Commits using an externally configured (git config) GPG-Key (Experimental)
- [ ] Enable extended debug logging

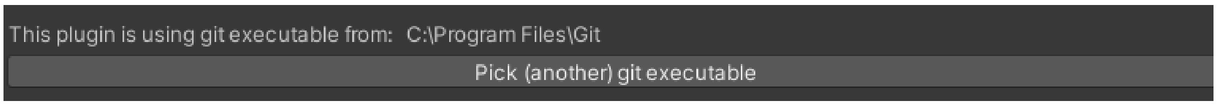


The first option can be quite handy – it displays markers next to most asset types (especially scripts and scenes) stating the status of this file in the project. A yellow pen means changed and unstaged, a green pen means staged for commit (but not committed) and a red sign means the file has been changed on the remote repository so you should not touch it. This feature is in a public beta state and enabled by default.

The experimental option to ignore system-wide config files can be enabled to instruct git to not rely on system-wide config files. This actually sets the `GIT_CONFIG_NOSYSTEM` variable.

The experimental option to sign commits can be enabled if your git is configured to use a default GPG key to sign commits and tags. This feature might especially break under linux because of the passphrase query.

In some cases people might have multiple Git versions laying around, or you can not install Git for some reason. In this case, you can pick the git.exe which the plugin should use by clicking the button:



This plugin is using git executable from: C:\Program Files\Git

Last but not least…

Git Collaboration Tools

Special Functions

Commit Templates

Some companies might need specially formatted commit messages. This asset supports templating.

Simply create a text file called „.gitmessage“ in the root of your project (the folder that contains the „Assets“ and „.git“ folder.

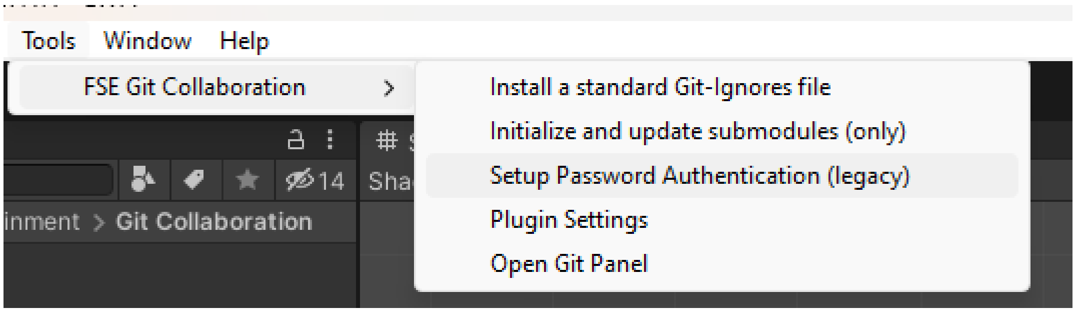
Note: If a previous commit failed, the original message is stored and restored on next commit try. The template will only be loaded on a new commit.

We do not read the git configuration files but you can configure your git to use the same message file as we do if you are committing on the console at some point: https://git-scm.com/book/en/v2/Customizing-Git-Git-Configuration

User/Password authentication

If you still require the legacy User/Password authentication, you are required to unzip the Helperscripts.zip file so that all those contained scripts lay in:
`Assets/Firesplash Entertainment/Git Collaboration/Helperscripts`

After doing so, you can configure your user and password using this menu:


Tools
Window
Help
FSE Git Collaboration
Install a standard Git-Ignores file
Initialize and update submodules (only)
Setup Password Authentication (legacy)
Plugin Settings
Open Git Panel
inment > Git Collaboration
14
Sha

Support

If you encounter any issue we are happy to help. Contact us on assets@firesplash.de