Getting started with git

Michaël Van Canneyt

August 18, 2021

Abstract

Recently, the Free Pascal and Lazarus teams switched from using Subversion to using Git as a source control system: the sources of the projects are now hosted on Gitlab. Time for a gentle introduction to git.

1 Introduction

People who want to contribute to an open source project are sooner or later confronted with a source control management system: In the early linux/unix days RCS (Revision Control system, a purely local solution), later CVS (Concurrent Version Control, which already offered client-server features), Subversion – similar to CVS, and featuring a central file repository. All contributors connect to a central repository to get the sources, and submit changes to this central repository.

To be able to manage the huge community to develop the Linux kernel, Linus Torvalds created git: Instead of a central repository (which would be prohibitively difficult to manage) it is a distributed version control system. What sets it apart from solutions such as Subversion is the lack of a central repository to which all contributors must connect.

Instead, there can be many repositories, all sharing the same source code. Changes can be migrated from one repository to another (a so-called pull request) and (usually) end up in the original repository.

The git versioning system took the programming world by storm: the appearance of github, bitbucket and gitlab source code project collaboration sites and derivatives such as gitea, have created an enormous ecosystem of tools around git. You can even use git to interact with a subversion repository, and github allows (limited) access to a git repository using subversion.

These projects build on top of git to provide easy to use merge-requests, issue tracking, CD/CI management, wikis and project management tools: all tools to facilitate cooperation on a software project.

2 Hosting a git repository

When working with a version system, the central repository must be hosted somewhere so all collaborators can reach it. With git (as with Subversion), you can host this yourself: gitlab has an installable version of their project (including an open source version), gitea is a small-scale open source server project with functionality that is a subset of what github or gitlab offer – but installing it is as easy as copying a single binary.

However, hosting it yourself means the maintainance burden also lies with you. It makes sense to use a SaaS offering and host it on gitlab, github or bitbucket: in that case the

platform maintainers will do all the maintenance. For small personal projects, it's not even necessary to purchase a license. For larger projects which require a lot of management, purchasing a license is probably the better idea.

For these reasons, the Free Pascal and Lazarus teams opted to host their sources on Gitlab.

3 Connecting to a git repository

To connect to a local or remote repository, you need a git client. There are many git clients available.

On Linux the command-line git client is available from the package manager of your Linux distribution. For MacOS, the command-line git client is installed with relatively recent versions of XCode. For both operating systems many GUI clients are available, some of them cross-platform.

On Windows, there are several available clients, many of them are cross-platform.

- Git for windows: a command-line client with a bash shell, so you can copy and paste commands you find on internet:

  `https://gitforwindows.org/`

- TortoiseGit integrates with the Windows explorer, much like TortoiseSVN does, and as such it is the only GUI client in this list that works only on windows:

  `https://tortoisegit.org/`

  If you already have TortoiseSVN installed, both can work side-by-side. If you install this client, you must also install the Git For windows client, as TortoiseGit uses that to do all the work behind the scenes.

- Github desktop is a GUI client for Github, made by the Github developers, but it can be used to connect to any Git repository:

  `https://desktop.github.com/`

  It works on all major platforms.

- Similarly, GitAhead is a git client that works on all major platforms:

  `https://gitahead.github.io/gitahead.com/`

- Sourcetree is a free client for Windows and macOS:

  `https://www.sourcetreeapp.com/`

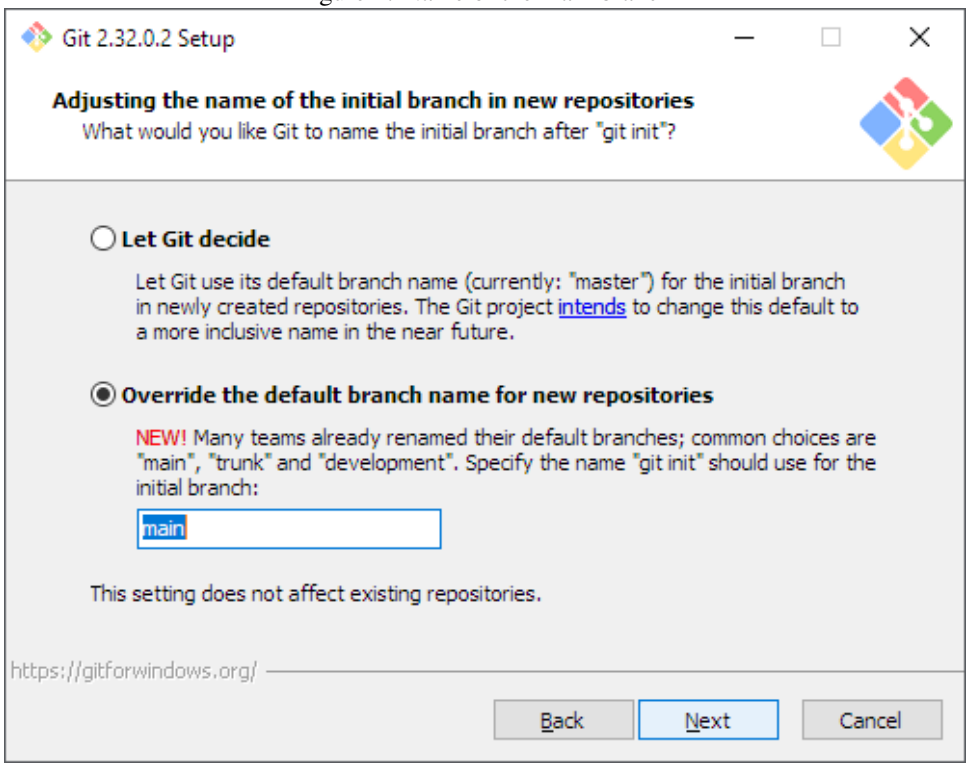
  It is made by the developers of BitBucket.

- SmartGit runs on Windows, Linux and macOS:

  `https://www.syntevo.com/smartgit/`

Last but not least, the popular VSCode and Atom editors have git support built-in: for most common operations, it is sufficient. Similarly, Delphi offers support for Git right in the IDE as well.

Figure 1: Name of the main branch



4 Git for Windows Installation

We'll discuss the installation of the Git for windows and TortoiseGit client here, because they require some configuration during setup, which can be confusing. The other mentioned tools can simply be installed without too much questions beyond the installation directory; Their configuration happens usually using some dialogs in the program itself.

Once you've downloaded the Git for Windows installer, it's a regular Windows installer like any other, but it asks some questions during the installation process. For a new git user, these questions can be confusing, so we'll go over them.

The first git-related question is what the initial branch of a new repository should be called, see figure 1 on page 3. Traditionally, the default initial branch created by git was called `master`. This choice raised some concerns about political inclusiveness of the software, so now the installer asks what the default name must be.

The second git-related question is what should be done with the Windows `PATH` environment variable, see figure 2 on page 4. Git for windows is made to run in a simulation of the unix shell (bash). But it can also be used outside this unix shell. This question asks you how you will want to use git: exclusively in the bash shell, do you want it to be usable by other tools as well, or should all provided unix-like tools also be made available in Windows terminal environment. If you want to use git for TortoiseGit, then the default choice (from the command-line and from 3rd-party software) is the correct one.

When connecting to a repository over HTTPS, the git client can use the OpenSSL library or it can use the Windows API, see figure 3 on page 4. Using the Windows API makes sense in corporate settings, where private server certificates should be accessible from e.g. an active directory. For most common setups, the use of OpenSSL is sufficient.

When working together with people that do not work on windows, it is important to use

Figure 2: Windows PATH changes

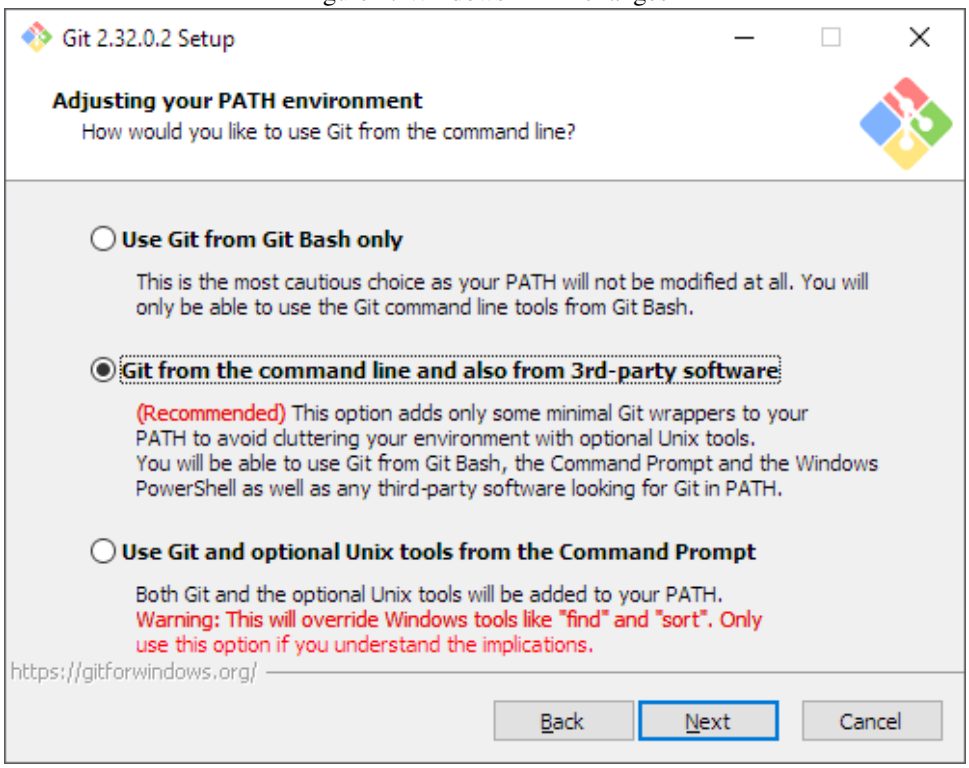


Figure 3: SSL settings

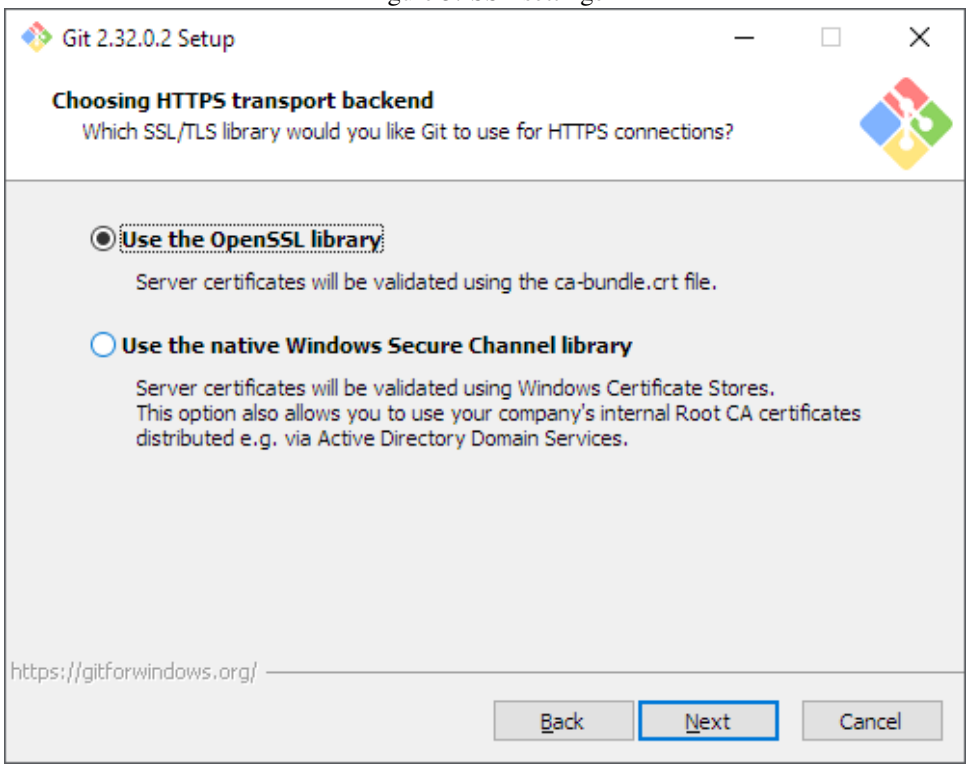
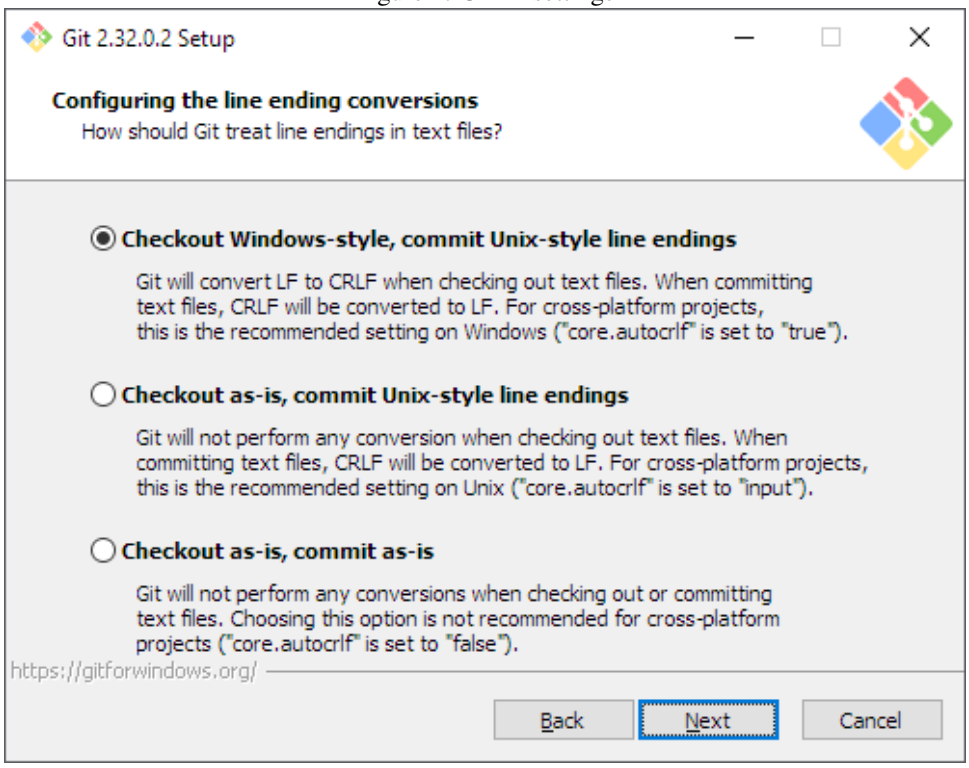


Figure 4: CRLF settings



the same settings for the line-ending character. On Windows this is traditionally CRLF, on Linux and mac, this is LF. Git can help you keeping consistent linefeeds in the source code and adjust to your platform's setting. This question asks you what you want git to do: The first choice is usually the best one. See figure 4 on page 5.

Git for windows is designed to work in a unix-like shell: bash. This runs in a terminal window. A special terminal window program (MinTTY) is provided with Git for Windows, which offers more functionality than the standard windows terminal. This question asks you which terminal window you want to use when you activate the Git for windows program. See figure 5 on page 6.

With the next question the installer wants to know what the behaviour of git should be when you get changes from the remote repository server, see figure 14 on page 16.

When you merge branches in git, or pull changes from the server, there are several ways in which git can do this. The default is fast forward: it just replays all the diffs on top of what you already have. But if that is impossible (because you have local changes that interfere with this), it will create a merge commit: git will create a new commit that marks the incorporation of the changes in the current branch.

There is an alternative, which is called 'rebase': git internally stores diffs, not files. What this option does is change the diffs which you committed locally so they look like they were applied on the last version retrieved from the server.

The git pull operation gets updates from the server and incorporates them in your local checkout. Here there is also an option to let git fail if it cannot do a fast-forward, i.e. when the remote changes cannot just be replayed on top of your work.

This question asks you which of these three scenarios you want to use.

When you want to push updates to a remote server (or pull from a protected server), you

Figure 5: Terminal settings

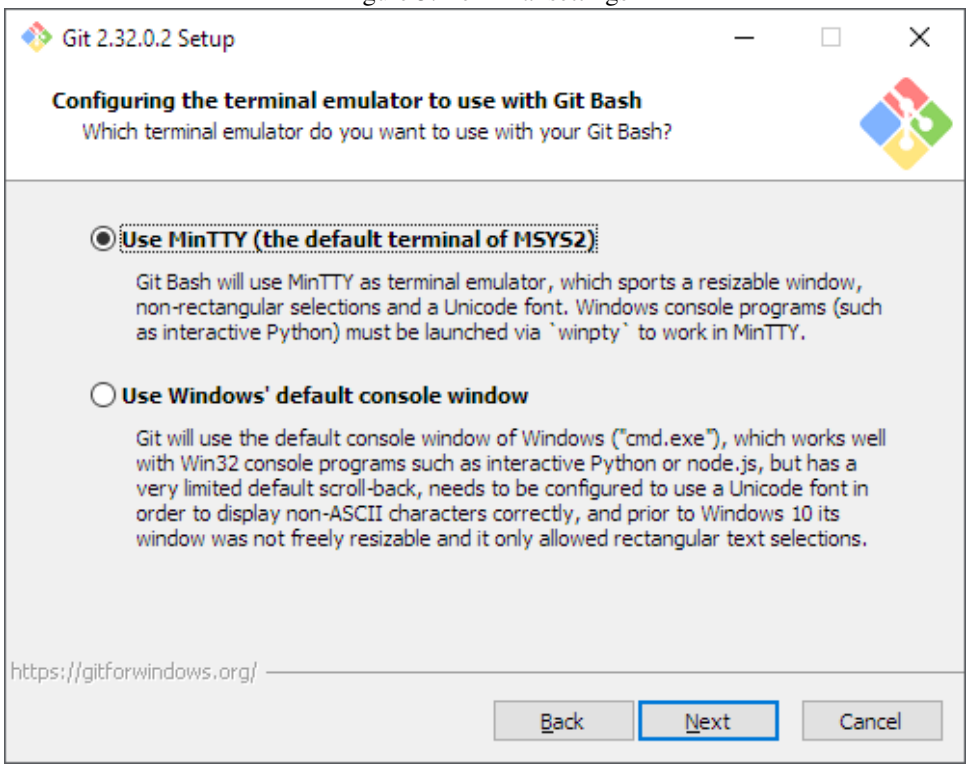


Figure 6: Pull settings

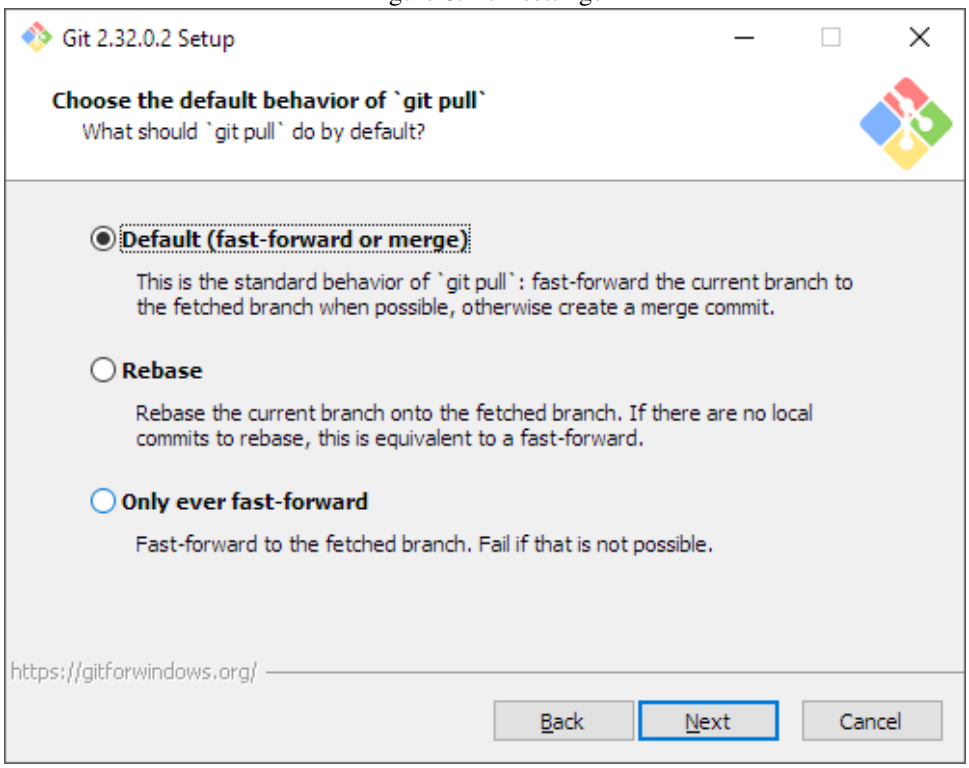
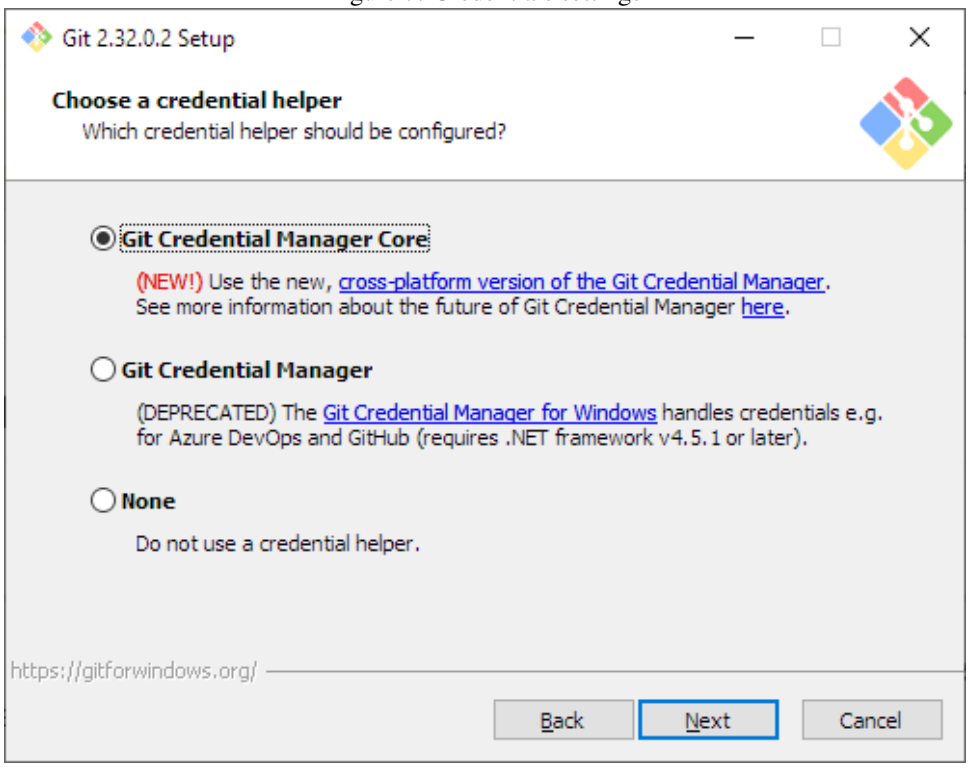


Figure 7: Credentials settings



must supply credentials: an SSH key when connecting through SSH, or a username/password when connecting using HTTPS. Unless you save the credentials, git will ask you for them every time it needs credentials. This question (figure 7 on page 7) asks you which credentials manager you want to use. You don't need to choose any one of the proposed managers, git has a default manager which stores the credentials in a text file, but you must configure that one separately.

The last question asked by the installer is whether it should cache some information from the file system, in order to speed up configuration. This has no influence on the working of git.

All of these questions result in the creation of a default system-wide configuration: your choices can be changed or reconfigured later on using the TortoiseGit client or on the git command-line.

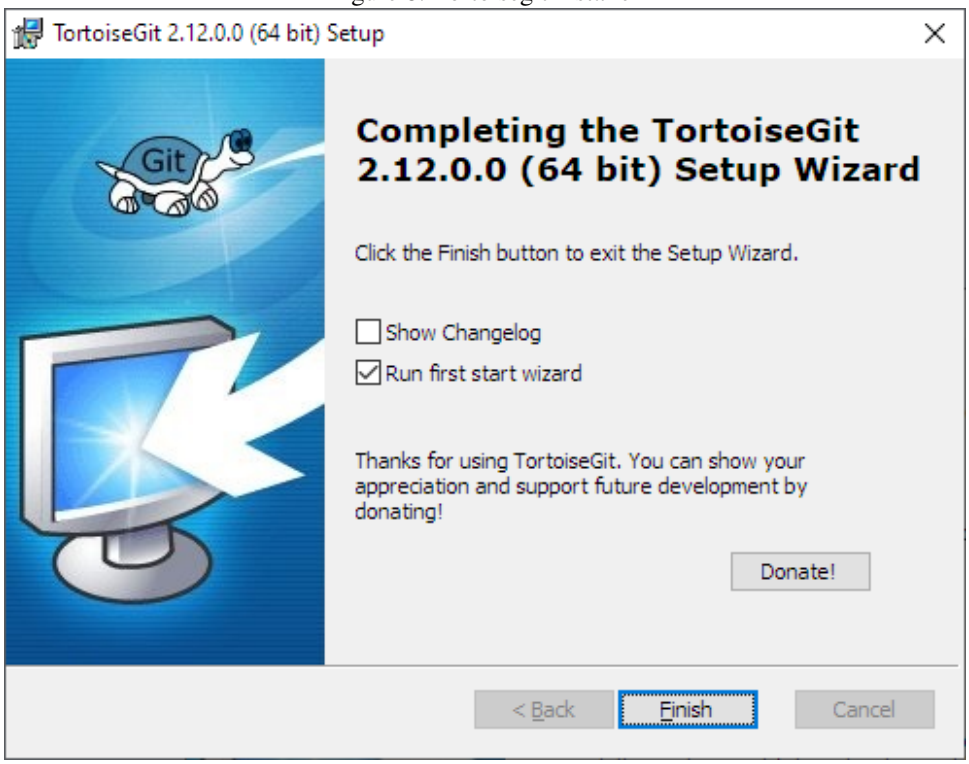
5 TortoiseGit Installation

The TortoiseGit client installer asks far less questions during installation. In fact, the only important one is whether it should start the first-start wizard. See figure 8 on page 8.

The first-start wizard will do the actual minimal configuration of TortoiseGit. After asking which language you want to use in the TortoiseGit interface, it will ask you where the git binaries are located. If you used the default settings for the PATH in the Git for Windows installer, then the TortoiseGit wizard should have detected the correct location and proposes it. See figure 9 on page 9.

When you commit source code, git will add your name and email address to the commit. It will attempt to guess it from your operating system username, which is more often than

Figure 8: Tortoisegit installer



not a bad choice.

Systems such as bitbucket, gitlab or github use the email address stored with the commit to identify the registered user that made a commit: The email address must therefore be an email address that these systems know. They have functionality so you can associate multiple email addresses with your account, but still your git client (tortoisegit) must associate one of these email addresses with each commit.

In this step of the wizard (figure 10 on page 10), the wizard asks you which username/email address you want to use. Note that this is not the identity information used to authenticate to a remote server (see also next question).

Last but not least, the TortoiseGit wizard asks how it should store or retrieve the authentication info used by git (figure 11 on page 11). In essence, this is the same question asked by the Git for Windows installer for HTTPS requests;

Additionally, TortoiseGit allows you to configure SSH access to the server: it can generate a public/private key pair, and the public key can then be uploaded to Gitea, Github, Bitbucket or Gitlab.

6 Getting code from a remote repository

To get a copy of the code in a remote repository on your local harddisk, git uses the `clone` command. If you want to check out the Free Pascal sources in a directory `D:\FPC\Source`, you would type the following commands in the bash window that the git installer has given you:

```
mkdir /d/FPC
```

Figure 9: Tortoisegit Wizard – git location

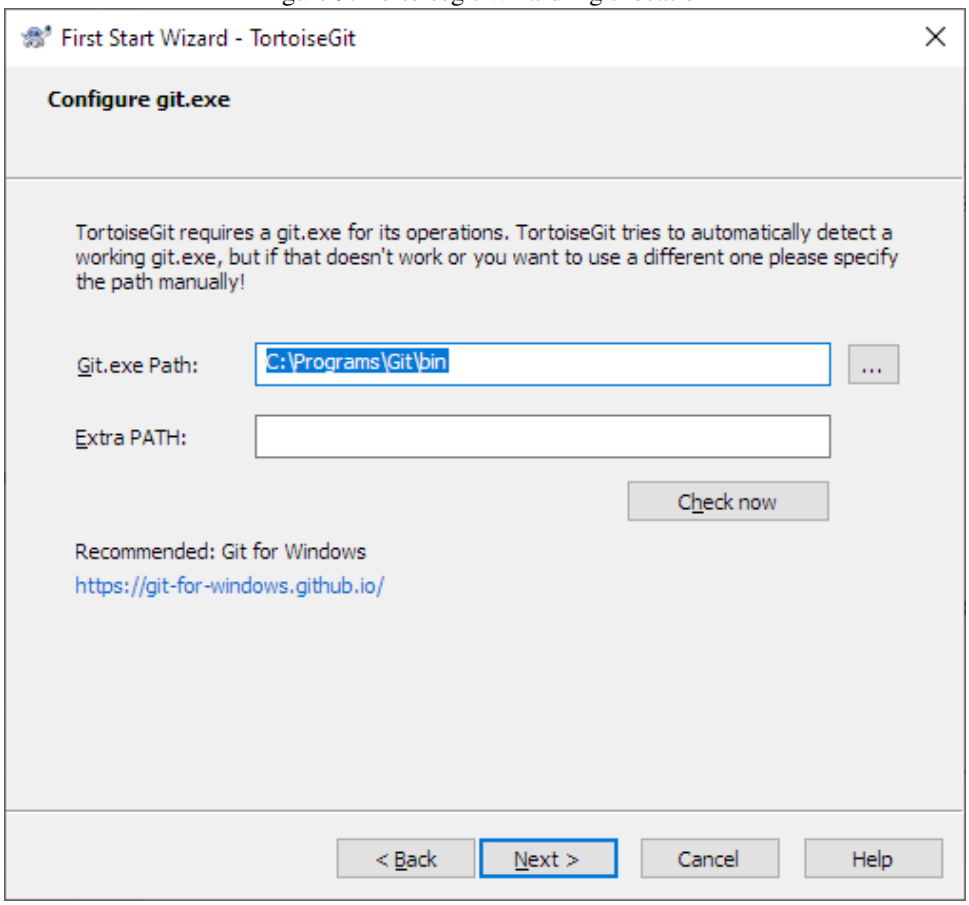


Figure 10: Tortoisegit Wizard – user identity

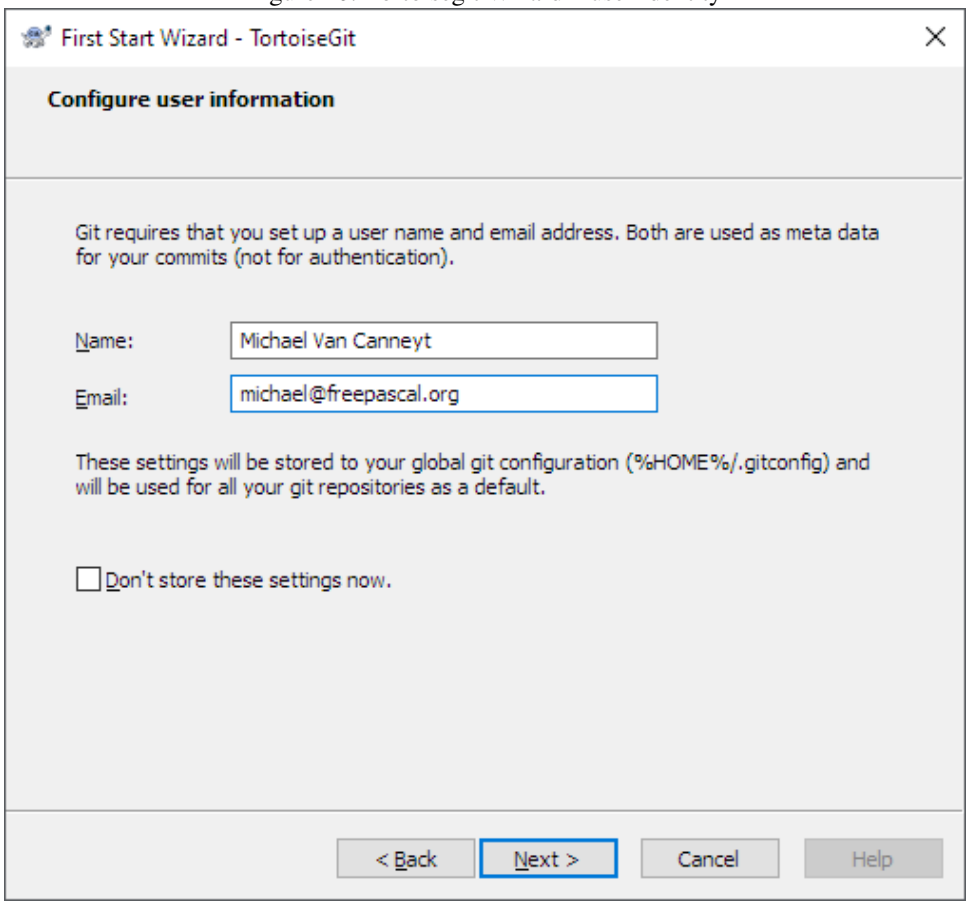
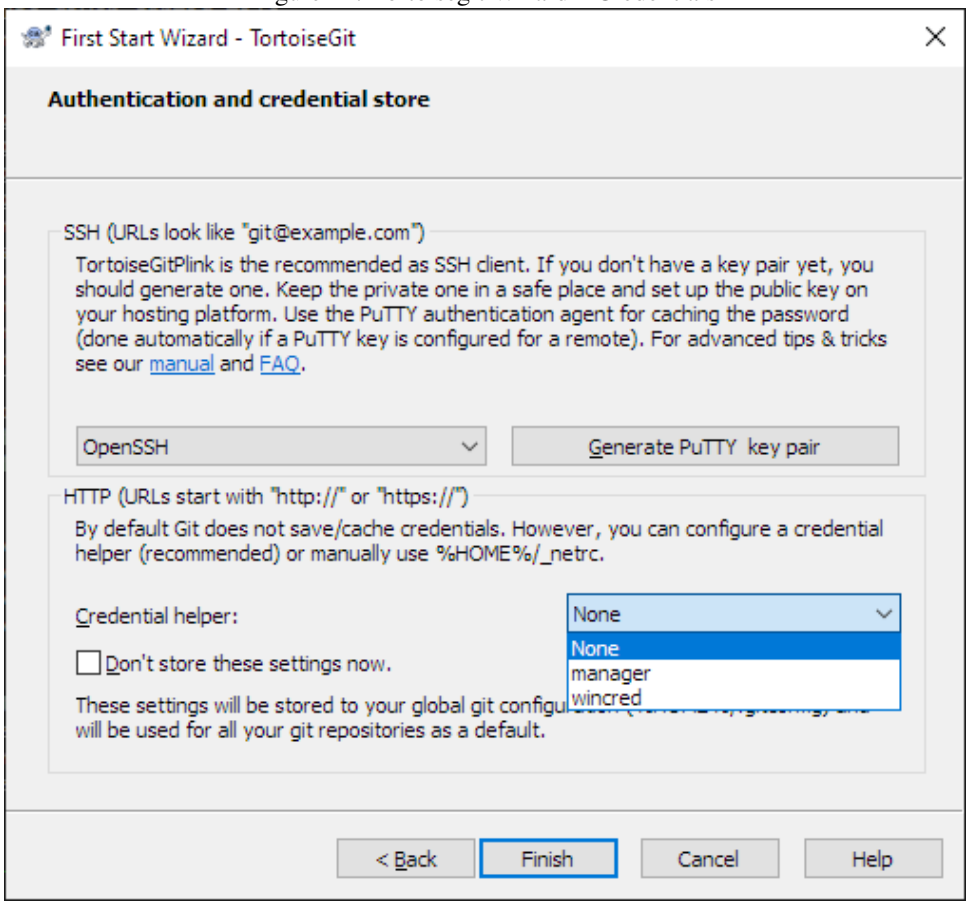


Figure 11: Tortoisegit Wizard – Credentials



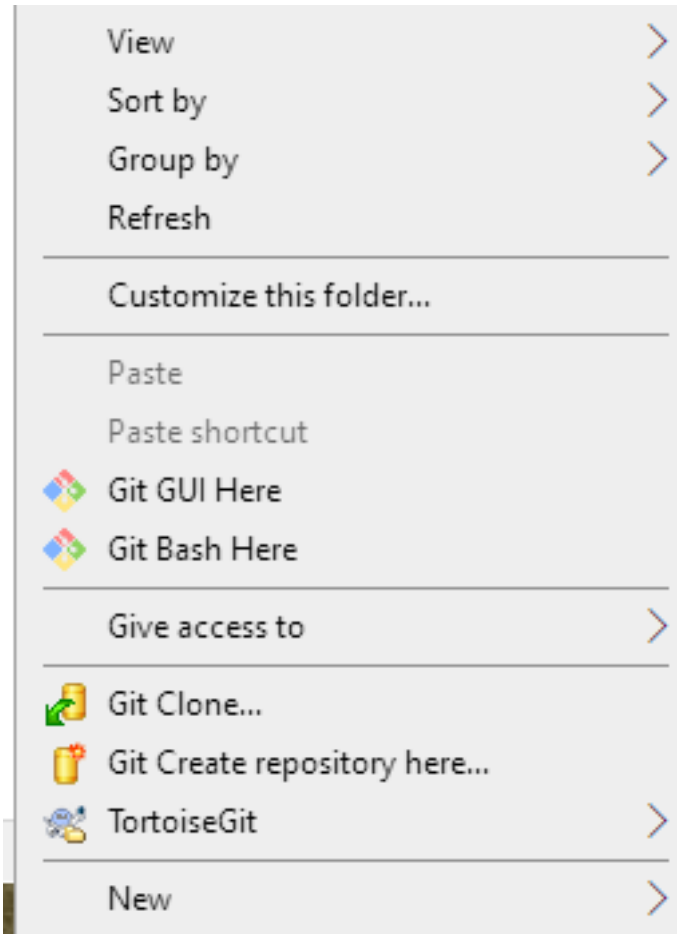
```
cd /d/FPC
git clone https://gitlab.com/freepascal.org/fpc/source.git Source
```

The first command is not necessary if you already have a FPC directory. Note that the bash unix environment uses forward slashes in path names.

This command does 2 things:

1. It fetches the contents of the remote repository, and puts them in an administrative directory (called .git).
2. From the local repository, it checks out the default branch. This step can be avoided with the `-no-checkout` or `-n` command-line option.

To do the same in the Tortoisegit shell extension, click right in the `D:\FPC` directory, and select `Git clone` from the context menu:



This will pop up the dialog in figure 12 on page 13. You can enter the URL as shown above, and the directory in which to check out the sources. There are many options you can specify, but for most purposes, the URL and directory are sufficient. Note that you can also use git as a client for remote Subversion repositories: you can also specify a Subversion server URL as remote repository.

When you press the `OK` button, git will fetch the complete repository from the server, and will check out the main branch (or any other branch you named). This can be a lengthy

Figure 12: TortoiseGit clone dialog

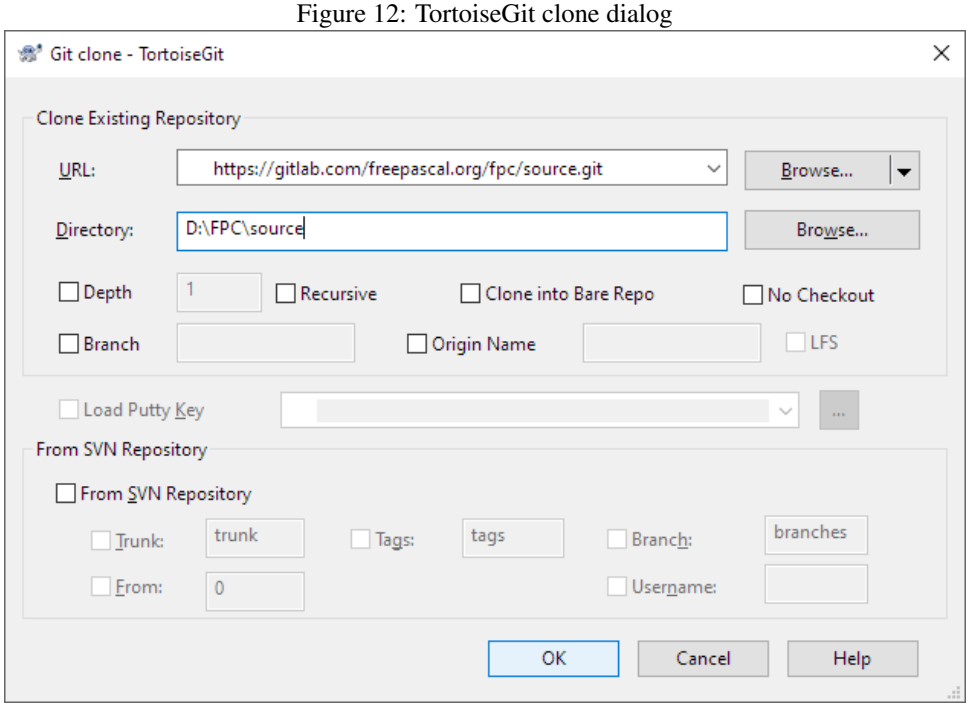
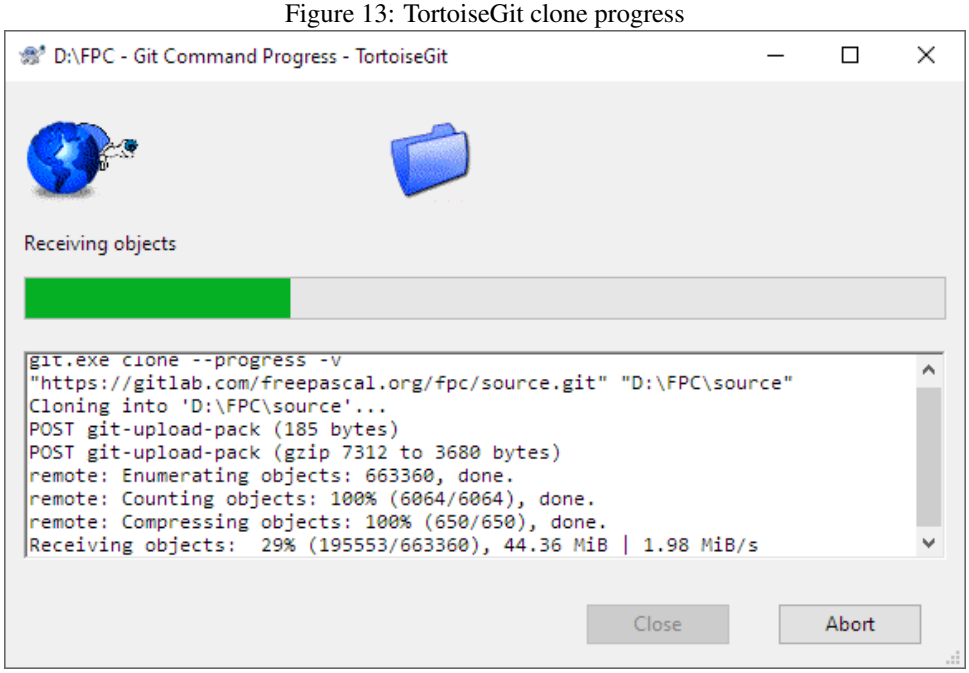
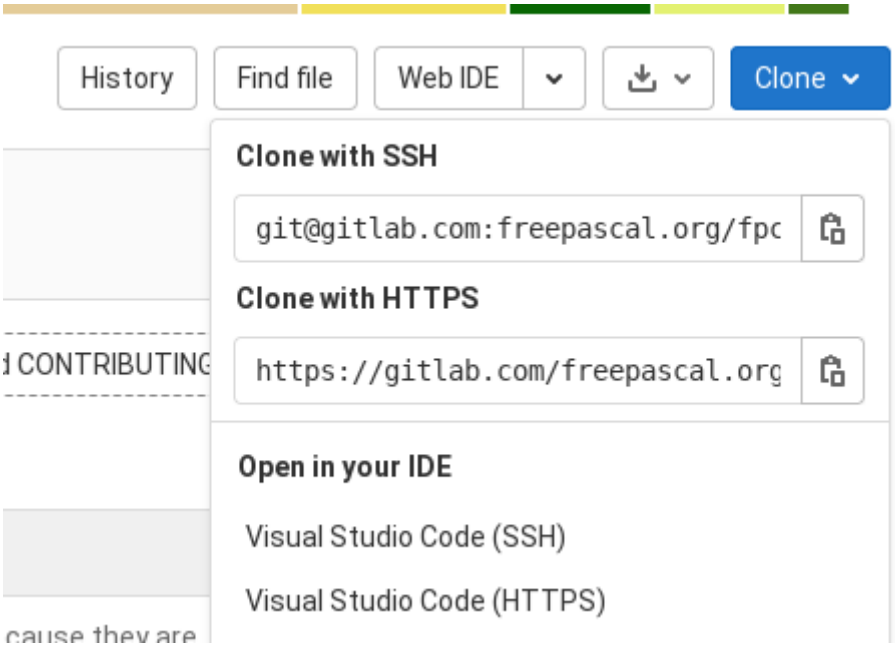


Figure 13: TortoiseGit clone progress

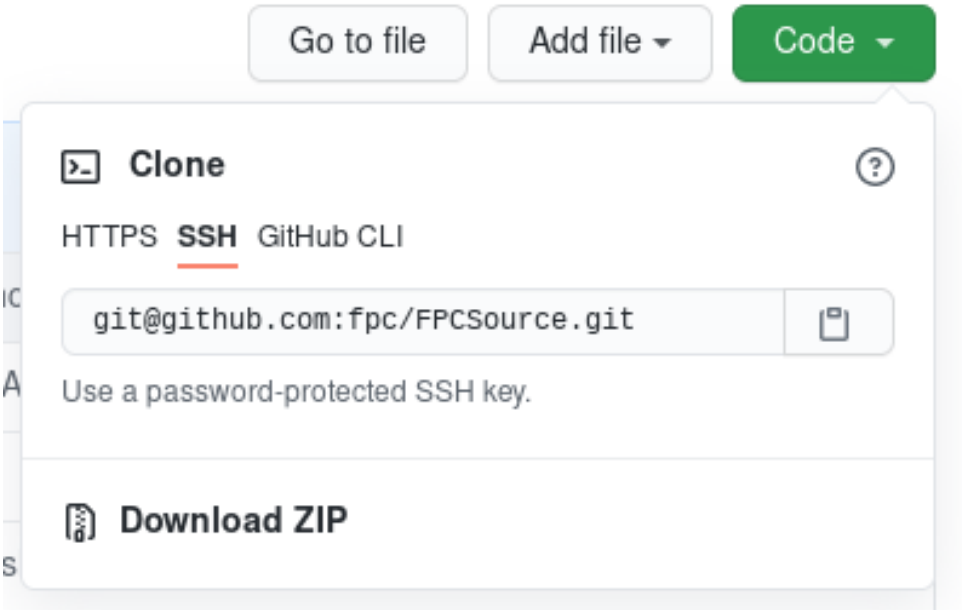


operation, and TortoiseGit will show a progress dialog as in figure 13 on page 13. It shows the same output as the command-line version of git.

How to get the remote repository URL you need from systems like github, gitea or gitlab? All these systems in their GUI show a button that, when clicked, allows you to see and copy the URL to clipboard. For example, for gitlab, the button looks as follows:



And for github, it looks like this:



7 Updating your copy of the sources

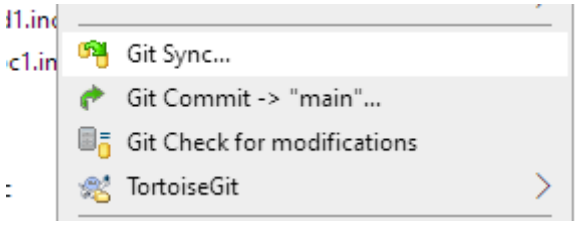
When you've cloned the repository locally, after some time, your copy will most likely be outdated: Project developers will have published new changes in the remote repository, and your copy does not yet contain these changes. To bring your local copy up to date with the remote repository, you must issue a `git pull` command. In subversion, this was the `update` command.

```
cd /d/FPC/source
git pull
```

As for `git clone`, the effect of the `pull` command is twofold:

1. It fetches all changes from the remote repository and stores them in your local copy of repository. This part is called fetching – git has actually a separate `fetch` command.
2. If in the fetching operation changes were fetched for the checked out branch, it applies these changes to your checked out branch.

In TortoiseGit, the pull operation is available under the `Git sync` menu item:



This brings up the synchronisation dialog, in which the `Pull` button can be used to do the actual pull operation. The result of the operation is shown in figure 14 on page 16.

8 Switching branches

In git, traditionally a lot of branches (parallel lines of development) are used: branching is a very cheap operation in git. Often, a new feature is developed in a branch and merged to the release branch when it is deemed ready. Using git, you can check out the branch and the sources before the feature is released.

You can get a list of available branches by simply entering the `git branch` command:

```
cd /d/FPC/source
git branch -r
```

The `-r` switch tells git it should show only remote branches. You can also specify `-a`, in which case all branches are shown.

In Tortoisegit, the same list can be obtained by choosing the `Browse references` menu item from the 'TortoiseGit' context menu. It will show the same information in a dialog as shown in figure 15 on page 16:

To actually switch to an existing branch, the git `switch` command can be used (in older git versions, this operation is called `checkout`).

Figure 14: Tortoisegit synchronisation - Pull result

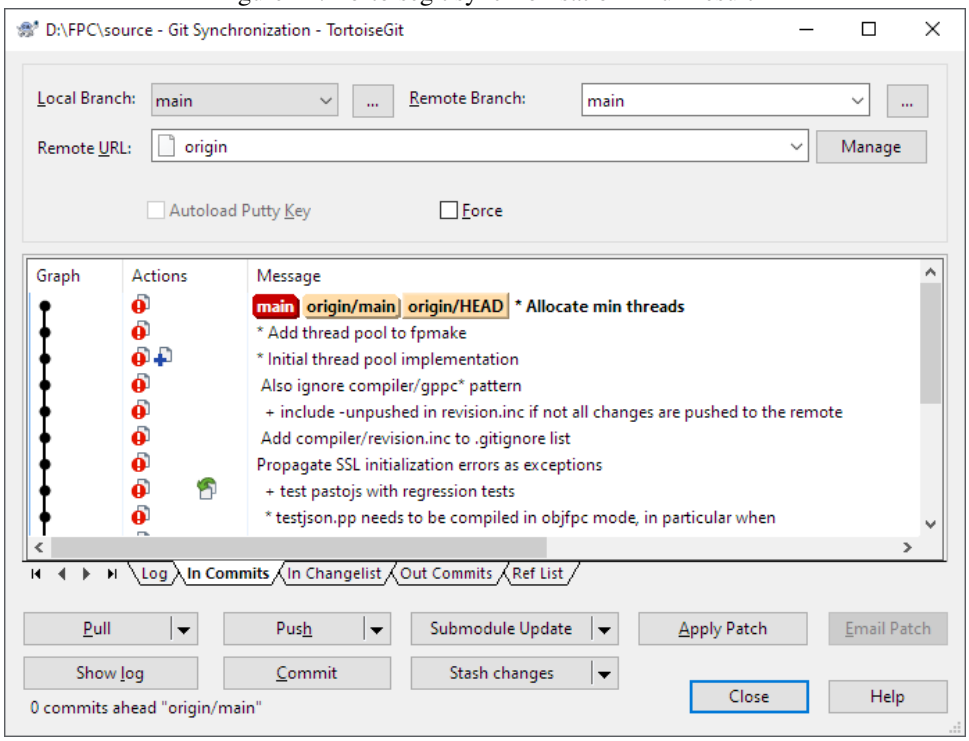
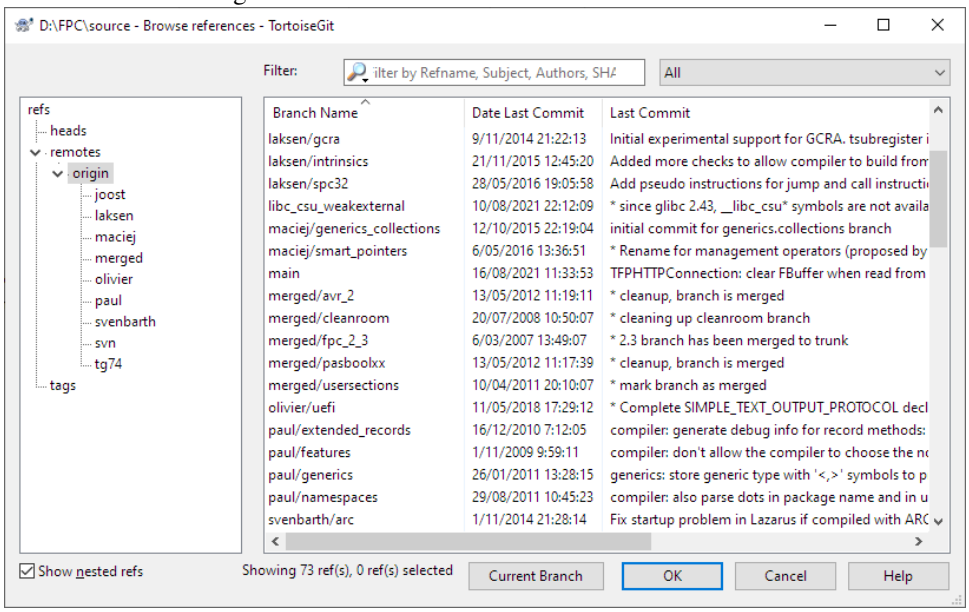


Figure 15: TortoiseGit browse references: branches



```
cd /d/FPC/source
git switch libc_csu_weakexternal
```

When you do this, git will check if the branch exists locally and switch to it if it exists. If it does not exist, but a remot branch exists with the same name, git will create a local branch with the same name as the remote one, and point the local one to the remote one: whenever the remote branch is updated, a `git pull` will update your local branch as well. Then it will switch to the new branch.

To create a new branch and switch to it, again the git `switch` command can be used:

```
cd /d/FPC/source
git switch -c issue_40100
```

You can also specify a different start point:

```
cd /d/FPC/source
git switch -c issue_40100 main
```

This will create a new branch `issue_40100` starting from the `main` branch.

When using an older git version, you use the checkout command with the `-b` option:

```
cd /d/FPC/source
git checkout -b issue_40100 main
```

You can also simply create a branch with the `branch` command, without switching to it:

```
cd /d/FPC/source
git branch libc_csu_weakexternal
```

In TortoiseGit, the `Switch/Checkout...` menu item must be used to switch to an existing branch or to create a new one. A dialog is shown (figure 16 on page 18) where the necessary parameters are given: You can select an existing branch or create a new one.

9 Temporary storing changes

Switching branches (or pulling from a remote repository) can fail if there are uncommitted changes in your working copy. If you don't want to commit these changes yet, for example because they're not yet done or not yet properly tested, this can be solved by using the `stash` command. This command sets aside the uncommitted changes in a diff, and restores the working copy to the state it was in before you created the changes.

```
cd /d/FPC/source
git stash -m '* Temp work for feature X'
```

You can leave the message empty, in that case git will create a message with a reference to the last commit.

To do this in TortoiseGit, you must select the 'Stash changes' command from the 'TortoiseGit' context menu. In the dialog that appears then (figure 17 on page 18) you can enter the stash message.

Figure 16: TortoiseGit switch dialog

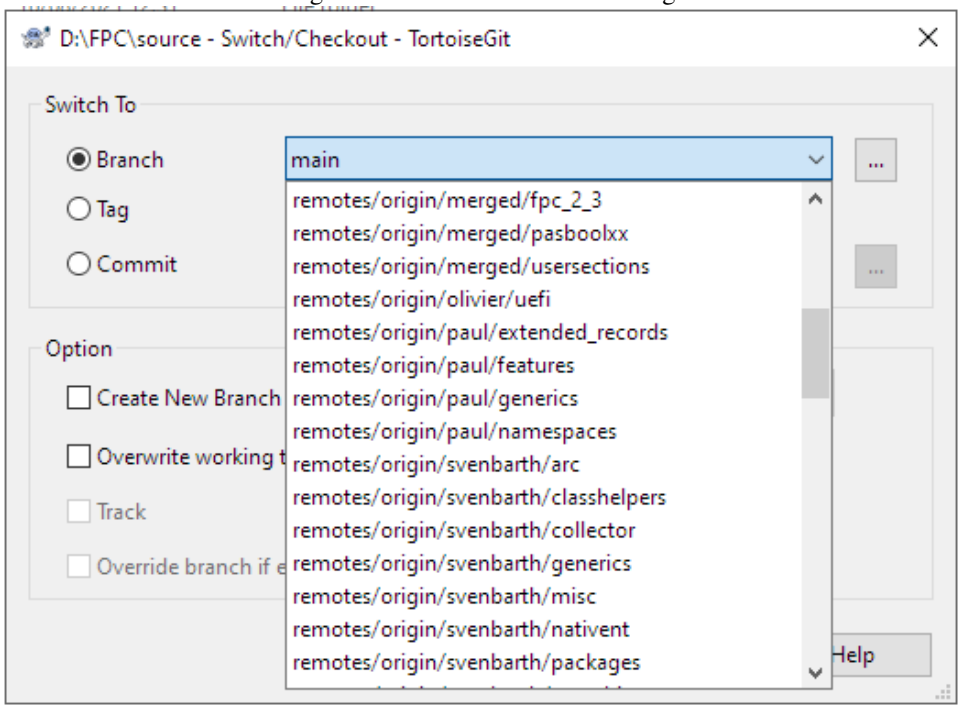
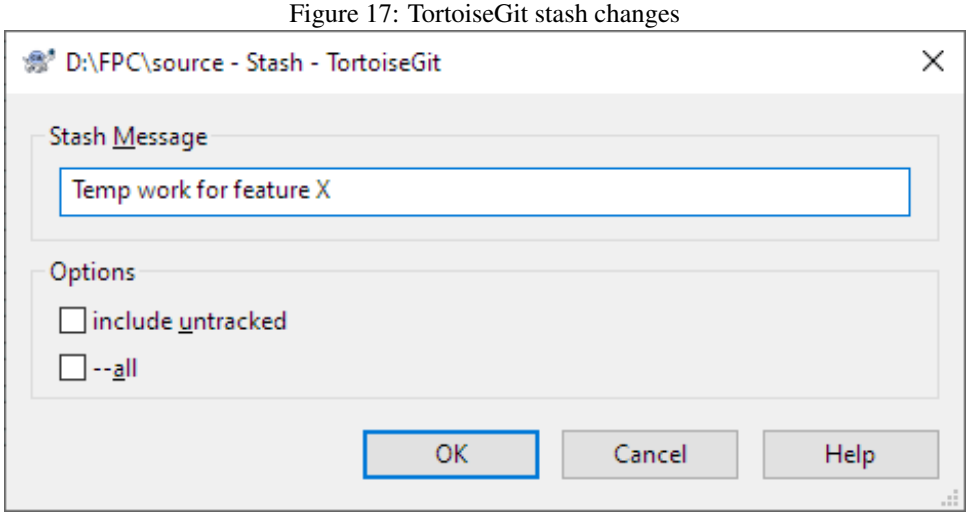


Figure 17: TortoiseGit stash changes



10 Discarding changes

Instead of setting aside changes, you may prefer to discard any changes you've made to the working copy. You can do this to resolve potential conflicts when pulling changes from a remote, or simply because you've done some changes you do not like and wish to undo them. In subversion, this operation was called `revert`. In git this operation is called `restore`.

On the command-line, the following will undo all changes to all files:

```
cd /d/FPC/source
git restore .
```

You can also specify one or more filenames instead of a directory.

In older versions of git, this had to be done with the `checkout` command:

```
cd /d/FPC/source
git checkout .
```

But this was confusing because the `checkout` command was used for a lot of other things, and the git developers made a special command for it. The checkout command also still works in newer versions of git.

In TortoiseGit, you can achieve the same with the 'Revert' menu item below the 'TortoiseGit' popup menu. You will get a dialog with a list of changed files and can select the files which must be reverted, see figure 18 on page 20.

11 Conclusion

In this article we've shown how to use git to get sources from remote repositories, and how to update your local copy. We've also shown what you can do when you changed files and wish to undo these changes, with the option to redo them later on. In a next article we'll show in more detail how to handle your own changes to sources, and how you can contribute them back to the project. This will require a deeper dive in the eco-systems built around git.

Figure 18: TortoiseGit revert