Git Extensions Documentation

Release 2.46

Contributors

October 08, 2013

CONTENTS

CHAPTER

ONE

GIT EXTENSIONS

Git Extensions is a toolkit aimed at making working with Git under Windows more intuitive (note that Git Extensions is also available on Linux and Macintosh OS X using Mono). The shell extension will integrate in Windows Explorer and presents a context menu on files and directories. There is also a Visual Studio plug-in to use Git from the Visual Studio IDE.

1.1 Features

- Windows Explorer integration for Git
- Visual Studio (2005/2008/2010/2012) plug-in for Git
- Feature rich user interface for Git
- Single installer installs Git, Git Extensions and the merge tool KDiff3
- 32bit and 64bit support
- Runs under Linux or Mac OS X using Mono

1.2 Video tutorials

There are video tutorials for some basic functions on YouTube.

1. Clone
2. Commit changes
3. Push changes
4. Pull changes
5. Handle merge conflicts
6. Install Git Extensions on Ubuntu 11.04

1.3 Links

See the following links for the Git Extensions download page, source code and documentation.

- Download page: https://sourceforge.net/projects/gitextensions/

- Source Code: https://github.com/gitextensions/gitextensions
- Source Code Issue tracker: https://github.com/gitextensions/gitextensions/issues
- Documentation: https://github.com/gitextensions/GitExtensionsDoc
- Documentation Issue tracker: https://github.com/gitextensions/GitExtensionsDoc/issues
- Wiki: https://github.com/gitextensions/gitextensions/wiki

Please feel free to raise any issues with Git Extensions or its documentation at the appropriate Issue tracker link as shown above.

CHAPTER

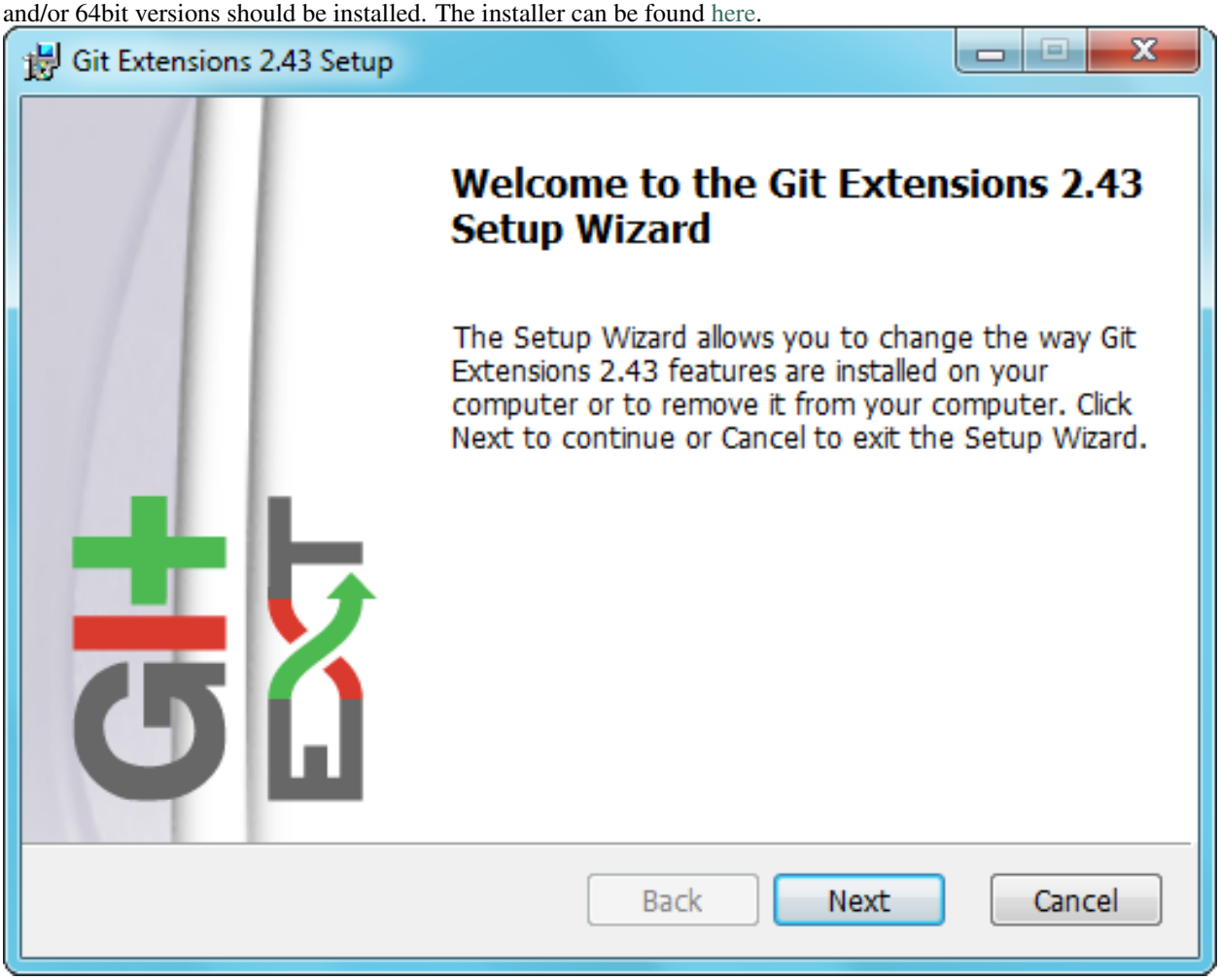
TWO

GETTING STARTED

This section is primarily written for Windows users. There are extra sections about installing Git Extensions on Linux and Mac OS X.

2.1 Installation

There is a single click installer that installs MsysGit, Kdiff3 and Git Extensions. The installer will detect if 32bit and/or 64bit versions should be installed. The installer can be found here.



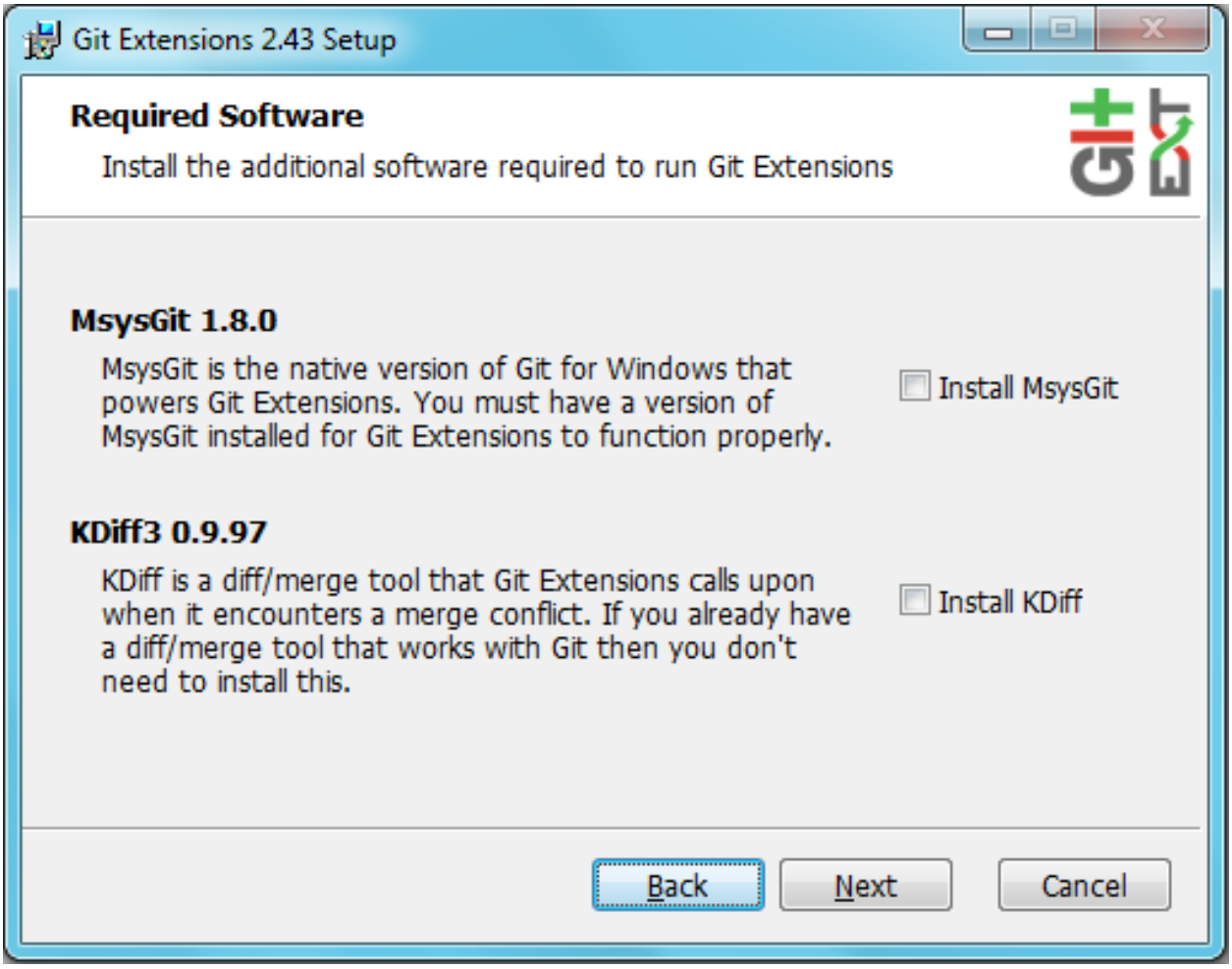
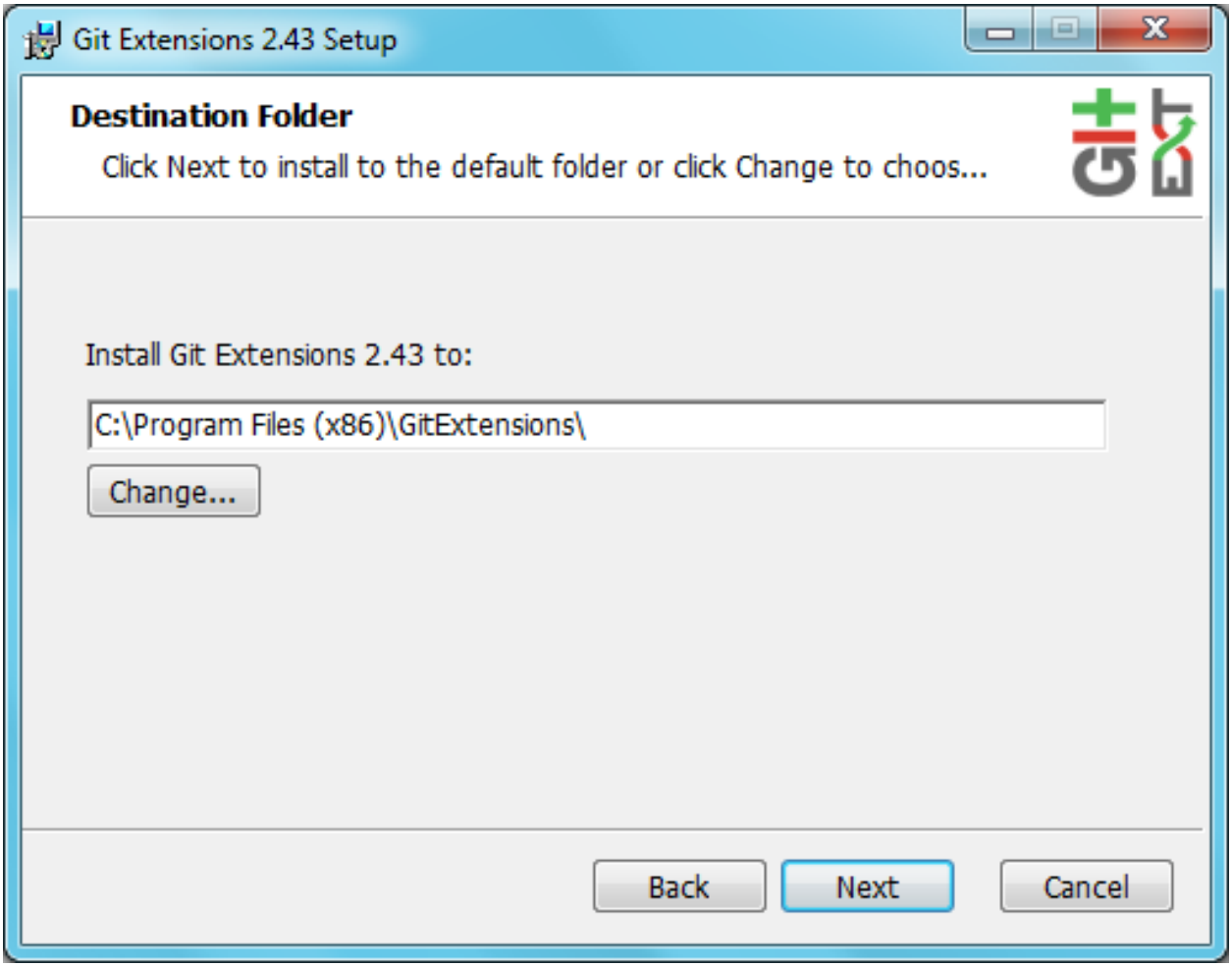


Figure 2.1: Git Extensions depends heavily on MsysGit. When MsysGit is not installed, ensure the "Install MsysGit" checkbox is checked. Kdiff3 is optional, but is advised as a merge tool.


Git Extensions 2.43 Setup
Destination Folder
Click Next to install to the default folder or click Change to choos...
Install Git Extensions 2.43 to:
C:\Program Files (x86)\GitExtensions\
Change...
Back
Next
Cancel

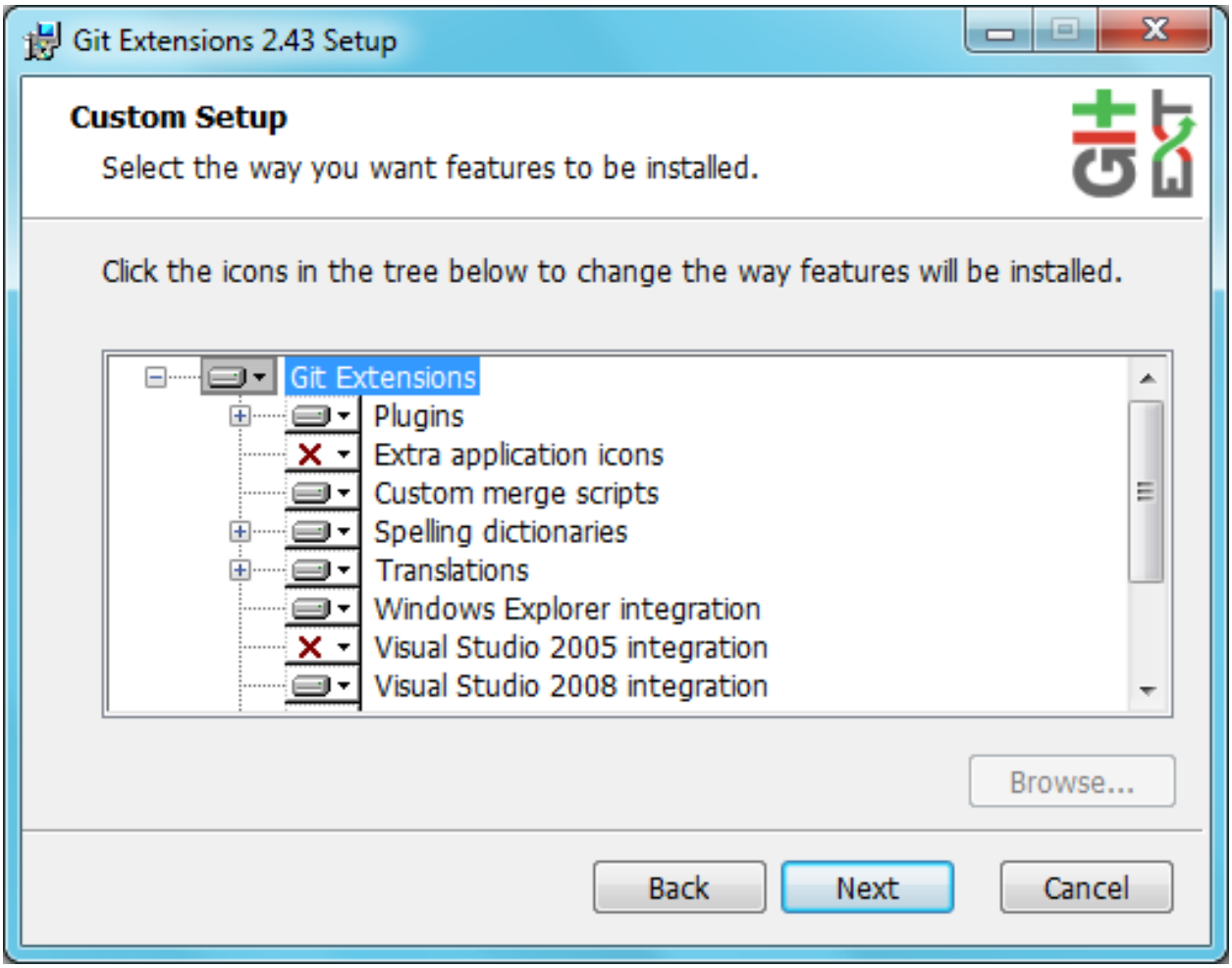


Figure 2.2: Choose the options to install.

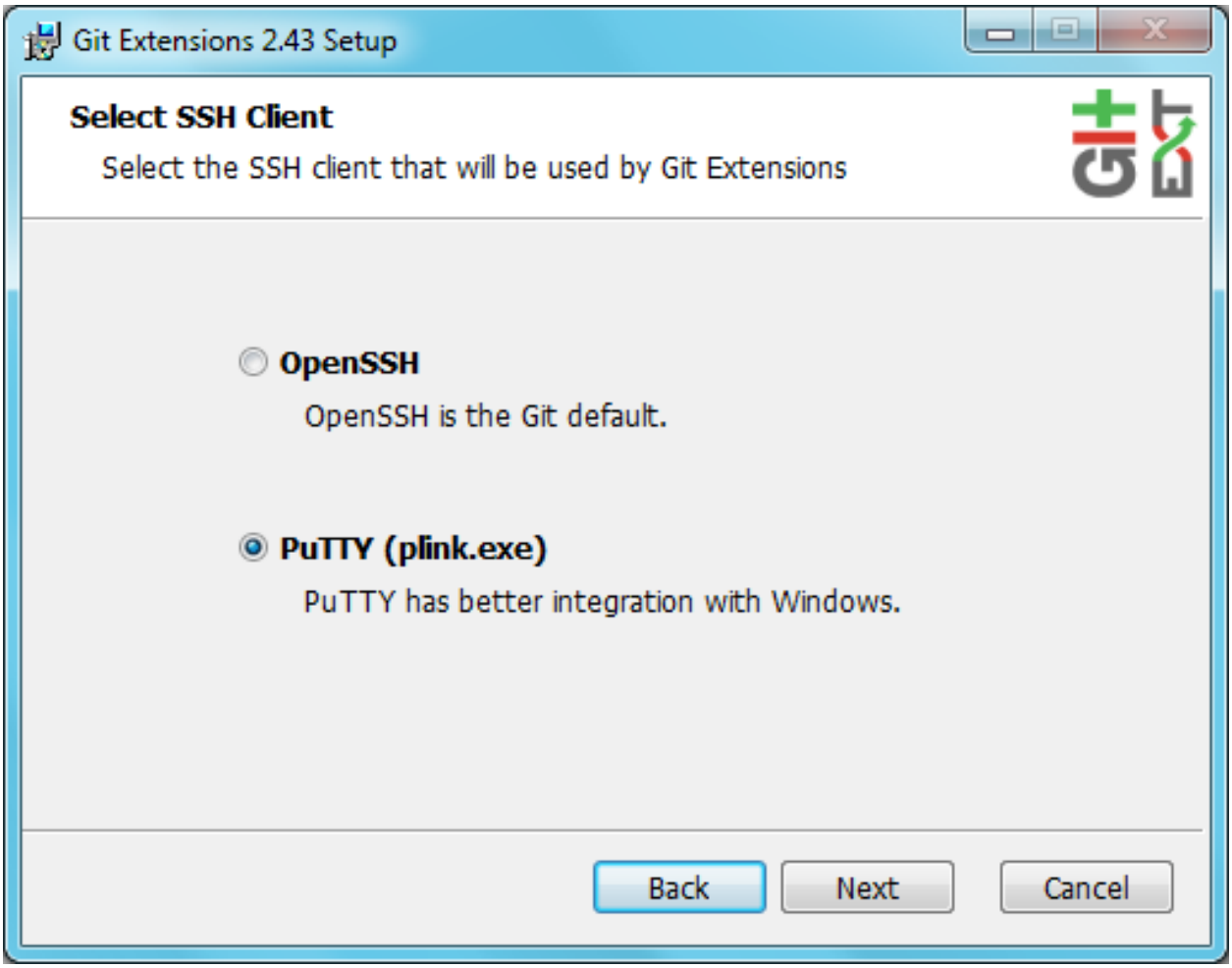
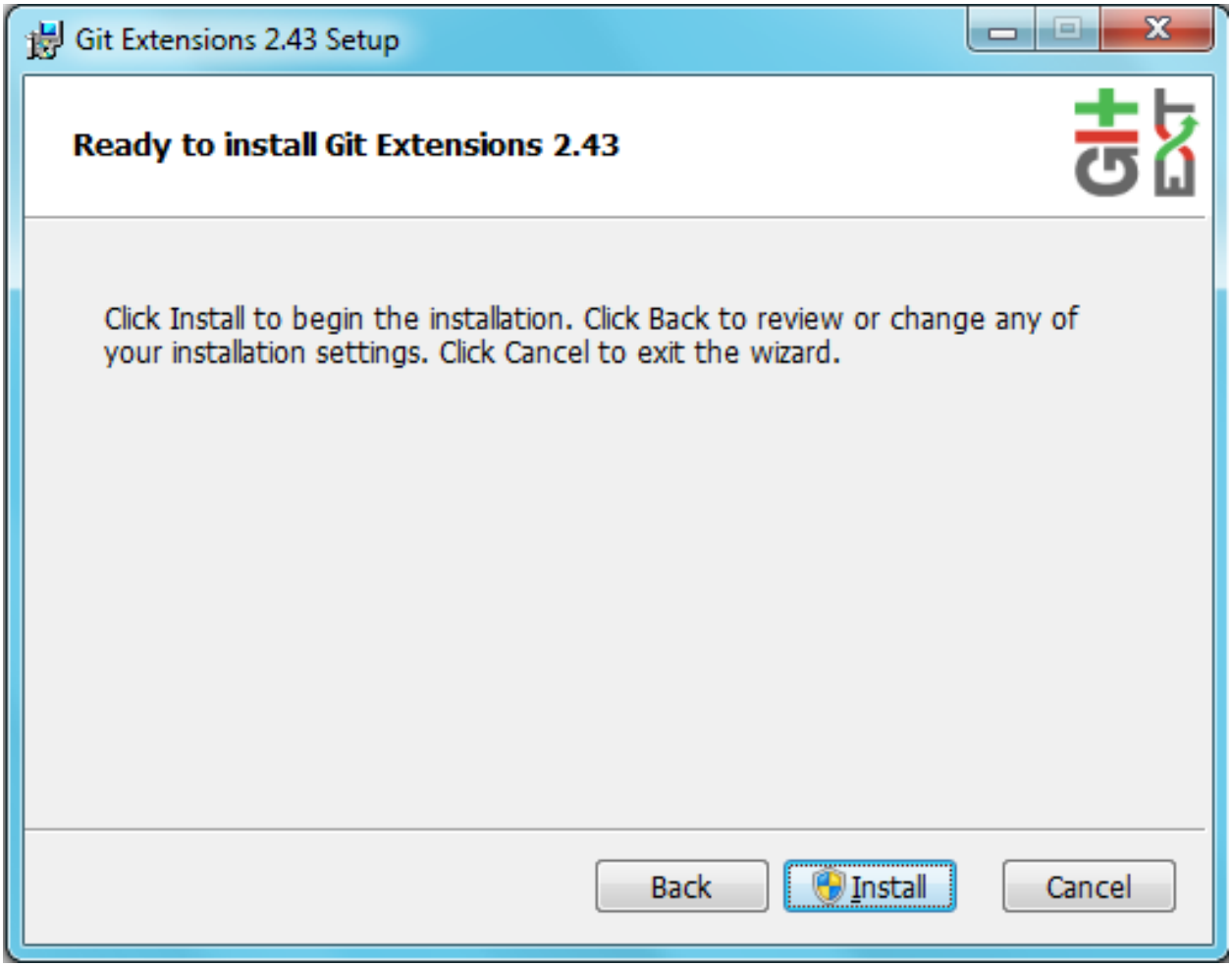


Figure 2.3: Choose the SSH client to use. PuTTY is the default because it has better Windows integration.



2.2 Installation (Linux)

You can watch this video as a starting point: Install Git Extensions on Ubuntu 11.04

For further help go to https://groups.google.com/forum/?fromgroups=#!forum/gitextensions

2.3 Installation (Mac)

First, make sure you have the latest mono version on your Mac. This section will cover installation of mono 2.10.11 on a Mac.

1. Download mono latest version. You can always check for this here: http://www.go-mono.com/mono-downloads/download.html
2. After you have completed the download, you will see a .dmg file. Double click it to open the package.
3. Inside the .dmg file you will have MonoFramework-{version}.pkg. Double click to start the installation process.
4. Follow the wizard until it's completion.
5. If everything went okay, you should open your terminal and check mono version:

```
$ mono --version
Mono JIT compiler version 2.10.11 (mono-2-10/2baeee2 Wed Jan 16 16:40:16 EST 2013)
Copyright (C) 2002-2012 Novell, Inc, Xamarin, Inc and Contributors. www.mono-project.com
    TLS:           normal
    SIGSEGV:       normal
    Notification:  kqueue
    Architecture:  x86
    Disabled:      none
    Misc:          softdebug
    LLVM:          yes(2.9svn-mono)
    GC:            Included Boehm (with typed GC)
```

6. Now download GitExtensions latest version from https://code.google.com/p/gitextensions/downloads/list. Remember to select the appropriate package otherwise you could have problems.
7. Browse into the folder where you extracted the package and just run mono command, like the example below:

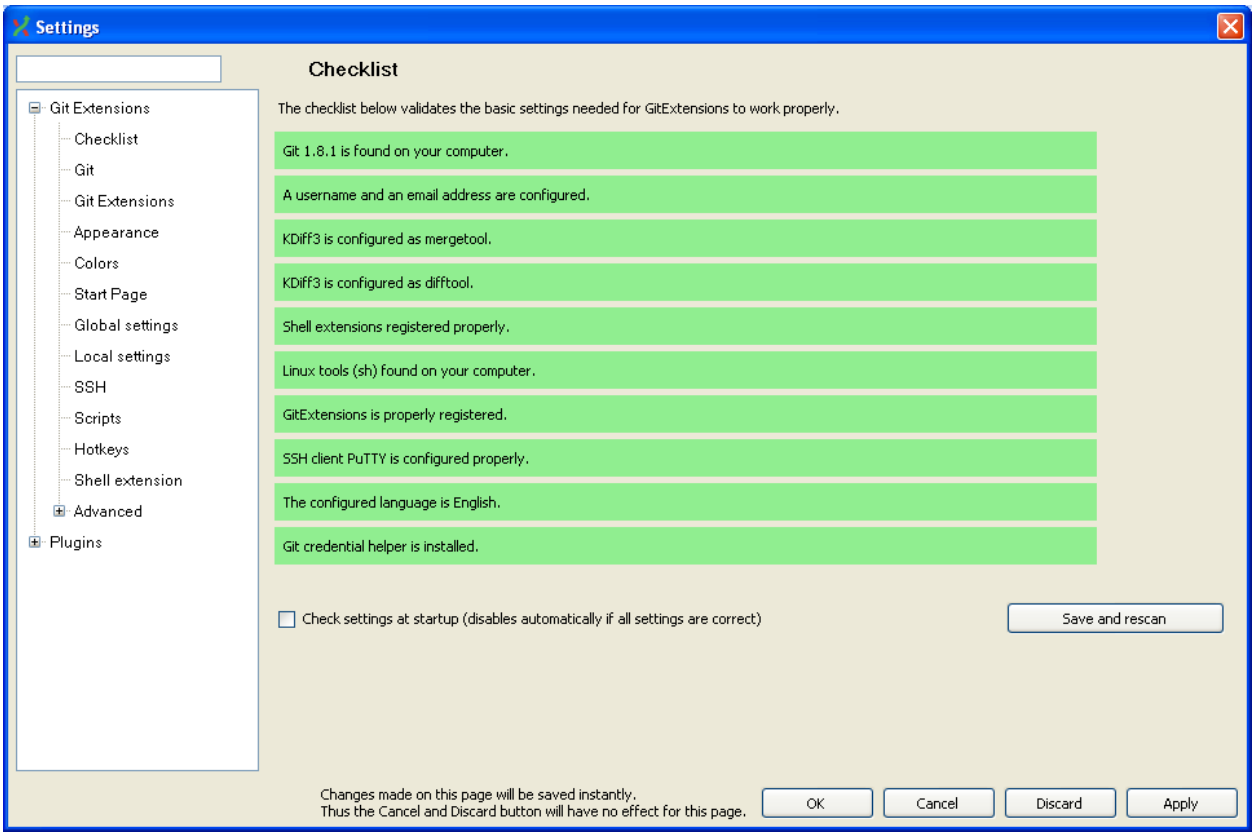
```
$ mono GitExtensions.exe
```

This is the minimal setup you need in order to run Git Extensions.

2.4 Settings

All settings will be verified when Git Extensions is started for the first time. If Git Extensions requires any settings to be changed, the Settings dialog will be shown. All incorrect settings will be marked in red. You can ask Git Extensions to try to fix the setting for you by clicking on it. When installing Git Extensions for the first time (and you do not have Git already installed on your system), you will normally be required to configure your username and email address.

The settings dialog can be invoked at any time by selecting `Settings` from the `Settings` menu option.



The following buttons are always available on any page of the Settings dialog. Sometimes the `Cancel` and `Discard` buttons have no effect for the page - this will be noted on the page in the area next to the buttons.

Button	Description
`OK`	Save any entered changes made in *any* settings page and close the Settings dialog.
`Cancel`	Any entered changes in *any* settings page are *not* saved. The Settings dialog is closed.
`Discard`	Any entered changes in *any* settings page are discarded i.e. they are reset back to their original values.
`Apply`	Any entered changes in *any* settings page are saved.

All settings that are specific to Git Extensions will be stored in the Windows registry. The settings that are used by Git are stored in the configuration files of Git. The global settings are stored in a file called `.gitconfig` in the user directory. The local settings are stored in the `.git\config` file of the repository.

2.4.1 Checklist

This page is a visual overview of the minimal settings that Git Extensions requires to work properly. Any items highlighted in red should be configured by clicking on the highlighted item.

This page contains the following settings and buttons.

Setting	Description
Check settings at startup (disables automatically if all settings are correct)	Forces Git Extensions to re-check the minimal set of required settings the next time Git Extensions is started. If all settings are 'green' this will be automatically unchecked.
Save and rescan Button	Saves any setting changes made and re-checks the settings to see if the minimal requirements are now met.

2.4.2 Git

This page contains the settings needed to access git repositories. The repositories will be accessed using external tools. For Windows usually MsysGit or cygwin are used. Git Extensions will try to configure these settings automatically.

Group	Setting	Description
Git	Command used to run git (git.cmd or git.exe)	Needed for Git Extensions to run Git commands. Set the full command used to run git (MsysGit or cygwin). Use the Browse button to find the executable on your file system.
Path to Linux tools (sh). Leave empty when it is in the path.	A few linux tools are used by Git Extensions. When MsysGit is installed, these tools are located in the bin directory of MsysGit. Use the Browse button to find the directory on your file system.	
Environment	Change HOME Button	This button opens a dialog where the HOME directory can be changed.

The global configuration file used by git will be put in the HOME directory. On some systems the home directory is not set or is pointed to a network drive. Git Extensions will try to detect the optimal setting for your environment. When there is already a global git configuration file, this location will be used. If you need to relocate the home directory for git, click the Change HOME button to change this setting. Otherwise leave this setting as the default.

2.4.3 Git Extensions

This page contains general settings for Git Extensions.

Group	Setting	Description
Performance	Show repository status in browse dialog (number of changes in toolbar, restart required)	When enabled, the number of pending commits are shown on the toolbar as a figure in parentheses next to the Commit button. Git Extensions must be stopped and restarted to activate changes to this option.
Show current working dir changes in revision graph	When enabled, two extra revisions are added to the revision graph. The first shows the current working directory status. The second shows the staged files. This option can cause slowdowns when browsing large repositories.	
Use FileSystemWatcher to check if index is changed	Using the FileSystemWatcher to check index state improves the performance in some cases. Turn this off if you experience refresh problems in commit log.	
Show stash count on status bar in browse window	When you use the stash a lot, it can be useful to show the number of stashed items on the toolbar. This option causes serious slowdowns in large repositories and is turned off by default.	
Check for uncommitted changes in checkout branch dialog	Git Extensions will not allow you to checkout a branch if you have uncommitted changes on the current branch. If you select this option, Git Extensions will display a dialog where you can decide what to do with uncommitted changes before swapping branches.	
Limit number of commits that will be loaded in list at start-up	This number specifies the maximum number of commits that Git Extensions will load when it is started. These commits are shown in the Commit Log window. To see more commits than are loaded, then this setting will need to be adjusted and Git Extensions restarted.	
Behaviour	Close Process dialog when process is succeeded	When a process is finished, close the process dialog automatically. Leave this option off if you want to see the result of processes. When a process has failed, the dialog will automatically remain open.
Show console window when executing git process	Git Extensions uses command line tools to access the git repository. In some environments it might be useful to see the command line dialog when a process is executed. An option on the command line dialog window displayed allows this setting to to be turned off.	
Use patience diff algorithm	Use the Git 'patience diff' algorithm instead of the default. This algorithm is useful in situations where two files have diverged significantly and the default algorithm may become 'misaligned', resulting in a totally unusable conflict file.	
Show errors when staging files	If an error occurs when files are staged(in the Commit dialog), then the process dialog showing the results of the	

2.4.4 Appearance

This page contains settings that affect the appearance of the application.

Group	Setting	Description
General	Show relative date instead of full date	Show relative date, e.g. 2 weeks ago, instead of full date. Displayed on the `commit` tab on the main Commit Log window.
Show current branch in Visual Studio	Determines whether or not the currently checked out branch is displayed on the Git Extensions toolbar within Visual Studio.	
Auto scale user interface when high dpi is used	Automatically resize controls and their contents according to the current system resolution of the display, measured in dots per inch (DPI).	
Truncate long filenames	This setting affects the display of filenames in a component of a window e.g. in the Diff tab of the Commit Log window. The three options that can be selected are: None: no truncation occurs; a horizontal scroll bar is used to see the whole filename. Compact: no horizontal scroll bar. Filenames are truncated at both start and end to fit into the width of the display component. Trimstart: no horizontal scroll bar. Filenames are truncated at the start only.	
Author images	Get author image from gravatar.com	If checked, gravatar will be accessed to retrieve an image for the author of commits. This image is displayed on the `commit` tab on the main Commit Log window.
Image size	The display size of the user image.	
Cache images	The number of days to elapse before gravatar is checked for any changes to an authors image.	
No image service	If the author has not set up their own image, then gravatar can return an image based on one of these services.	
Clear image cache button	Clear the cached avatars.	
Fonts	Code font	Change the font used for the display of file contents.
Application font	Change the font used on Git Extensions windows and dialogs.	
Language	Language (restart required)	Choose the language for the Git Extensions interface.
Dictionary for spelling checker	Choose the dictionary to use for the spelling checker in the Commit dialog.	

2.4.5 Colors

This page contains settings to define the colors used in the application.

Group	Setting	Description
Revision graph	Multicolor branches	Displays branch commits in different colors if checked. If unchecked, all branches are shown in the same color. This color can be selected.
Striped branch change	When a new branch is created from an existing branch, the common part of the history is shown in a 'hatch' pattern.	
Draw branch borders	Outlines branch commits in a black border if checked.	
Draw non relatives graph gray	Show commit history in gray for branches not related to the current branch.	
Draw non relatives text gray	Show commit text in gray for branches not related to the current branch.	
Color tag	Color to show tags in.	
Color branch	Color to show branch names in.	
Color remote branch	Color to show remote branch names in.	
Color other label	Color to show other labels in.	
Application Icon	Icon style	Change icons. Useful for recognising various open instances.
Icon color	Changes color of the selected icons.	
Difference Viewer	Color removed line	Highlight color for lines that have been removed.
Color added line	Highlight color for lines that have been added.	
Color removed line highlighting	Highlight color for characters that have been removed in lines.	
Color added line highlighting	Highlight color for characters that have been added in lines.	
Color section	Highlight color for a section.	

2.4.6 Start Page

This page allows you to add/remove or modify the Categories and repositories that will appear on the Start Page when Git Extensions is launched. Per Category you can either configure an RSS feed or add repositories. The order of both Categories, and repositories within Categories, can be changed using the context menus in the Start Page. See *Start Page* for further details.

Setting	Description
Categories	Lists all the currently defined Categories. Click the Add button to add a new empty Category. The default name is 'new'. To remove a Category select it and click Remove. This will delete the Category *and* any repositories belonging to that Category.
Caption	This is the Category name displayed on the Start Page.
Type	Specify the type: an RSS feed or a repository.
RSS Feed	Enter the URL of the RSS feed.
Path/Title/Description	For each repository defined for a Category, shows the path, title and description. To add a new repository, click on a blank line and type the appropriate information. The contents of the Path field are shown on the Start Page as a link to your repository *if* the Title field is blank. If the Title field is non-blank, then this text is shown as the link to your repository. Any text in the Description field is shown underneath the repository link on the Start Page.

An RSS Feed can be useful to follow repositories on GitHub for example. See this page on GitHub: https://help.github.com/articles/viewing-your-feeds. You can also follow commits on public GitHub repositories by

1. In your browser, navigate to the public repository on GitHub.
2. Select the branch you are interested in.
3. Click on the Commits tab.
4. You will find a RSS icon next to the words "Commit History".
5. Copy the link
6. Paste the link into the RSS Feed field in the Settings - Start Page as shown above.

Your Start Page will then show each commit - clicking on a link will open your browser and take you to the commit on GitHub.

2.4.7 Global Settings

This page contains the following global Git settings. These settings will affect all repositories.

Group	Setting	Description
User email	User name User email shown in commits and patches.	User name shown in commits and patches.
Editor	Editor that git.exe opens (e.g. for editing commit message). This is not used by Git Extensions, only when you call git.exe from the command line. By default Git will use the built in editor.	
Merge-tool	Merge tool used to solve merge conflicts. Git Extensions will search for common merge tools on your system.	
Path to merge-tool	Path to merge tool. Git Extensions will search for common merge tools on your system.	
Merge-tool com-mand	Command that Git uses to start the merge tool. Git Extensions will try to set this automatically when a merge tool is chosen. This setting can be left empty when Git supports the mergetool (e.g. kdiff3).	
Keep backup (.orig) after merge	Check to save the state of the original file before modifying to solve merge conflicts. Refer to Git configuration setting `mergetool.keepBackup`.	
Difftool	Diff tool that is used to show differences between source files. Git Extensions will search for common diff tools on your system.	
Path to difftool	The path to the diff tool. Git Extensions will search for common diff tools on your system.	
DiffTool com-mand	Command that Git uses to start the diff tool. This setting should only be filled in when Git doesn't support the diff tool.	
Path to commit template	A path to a file whose contents are used to pre-populate the commit message in the commit dialog.	
Line endings	Checkout/commit radio buttons	Choose how git should handle line endings when checking out and checking in files. Refer to https://help.github.com/articles/dealing-with-line-endings#platform-all
	Files content encoding	The default encoding for file contents.

2.4.8 Local Settings

This page contains the Git settings *for a repository*. These settings are only required if you wish to override the global Git settings for this specific repository.

Group	Setting	Description
User email	User name User email shown in commits and patches.	User name shown in commits and patches.
Editor	Editor that git.exe opens (e.g. for editing commit message). This is not used by Git Extensions, only when you call git.exe from the command line. By default Git will use the command line text editor vi.	
Merge-tool	Merge tool used to solve merge conflicts. Git Extensions will search for common merge tools on your system.	
Keep backup (.orig) after merge	Check to save the state of the original file before modifying to solve merge conflicts. Refer to Git configuration setting `'mergetool.keepBackup'`.	
Line endings	Checkout/commit radio buttons	Choose how git should handle line endings when checking out and checking in files. Refer to https://help.github.com/articles/dealing-with-line-endings#platform-all
	Files content encoding	Choose the encoding you want GitExtensions to use.

2.4.9 SSH

This page allows you to configure the SSH client you want Git to use. Git Extensions is optimized for PuTTY. Git Extensions will show command line dialogs if you do not use PuTTY and user input is required (unless you have configured SSH to use authentication with key instead of password). Git Extensions can load SSH keys for PuTTY when needed.

Group	Setting	Description
Specify which ssh client to use	PuTTY radio button Use OpenSSH as SSH client.	Use PuTTY as SSH client.
OpenSSH radio button		
Other ssh client	Use another SSH client. Enter the path to the SSH client you wish to use.	
Configure PuTTY	Path to plink.exe	Enter the path to the plink.exe executable.
Path to puttygen	Enter the path to the puttygen.exe executable.	
Path to pageant	Enter the path to the pageant.exe executable.	
Automatically start authentication	If an SSH key has been configured, then when accessing a remote repository the key will automatically be used by the SSH client if this is checked.	

2.4.10 Scripts

This page allows you to configure specific commands to run before/after Git actions or to add a new command to the User Menu. The top half of the page summarises all of the scripts currently defined. If a script is selected from the summary, the bottom half of the page will allow modifications to the script definition.

A hotkey can also be assigned to execute a specific script. See *Hotkeys*.

Setting	Description
Add Button	Adds a new script. Complete the details in the bottom half of the screen.
Remove Button	Removes a script.
Up/Down Arrows	Changes order of scripts.
Name	The name of the script.
Enabled checkbox	If checked, the script is active and will be performed at the appropriate time (as determined by the On Event setting).
Ask for confirmation checkbox	If checked, then a popup window is displayed just before the script is run to confirm whether or not the script is to be run. Note that this popup is *not* displayed when the script is added as a command to the User Menu (On Event setting is ShowInUserMenuBar).
Add to revision grid context menu checkbox	If checked, the script is added to the context menu that is displayed when right-clicking on a line in the Commit Log page.
Command	Enter the command to be run. This can be any command that your system can run e.g. an executable program, a .bat script, a Python command, etc. Use the `'Browse` button to find the command to run.
Arguments	Enter any arguments to be passed to the command that is run. The `'Help'` button displays items that will be resolved by Git Extensions before executing the command e.g. {cBranch} will resolve to the currently checked out branch, {UserInput} will display a popup where you can enter data to be passed to the command when it is run.
On Event	Select when this command will be executed, either before/after certain Git commands, or displayed on the User Menu bar.

2.4.11 Hotkeys

This page allows you to define keyboard shortcuts to actions when specific pages of Git Extensions are displayed. The HotKeyable Items identifies a page within Git Extensions. Selecting a Hotkeyable Item displays the list of commands on that page that can have a hotkey associated with them.

The Hotkeyable Items consist of the following pages

1. Commit: the page displayed when a Commit is requested via the `'Commit'` User Menu button or the `'Commands/Commit'` menu option.
2. Browse: the Commit Log page (the page displayed after a repository is selected from the Start Page).
3. RevisionGrid: the list of commits on the Commit Log page.
4. FileViewer: the page displayed when viewing the contents of a file.
5. FormMergeConflicts: the page displayed when merge conflicts are detected that need correcting.
6. Scripts: shows scripts defined in Git Extensions and allows shortcuts to be assigned. Refer *Scripts*.

Setting	Description
Hotkey	After selecting a Hotkeyable Item and the Command, the current keyboard shortcut associated with the command is displayed here. To alter this shortcut, just press the keyboard combination required. This field will be updated to reflect the keys pressed.
Apply button	Click to apply the entered keyboard combination to the Command.
Clear button	Sets the keyboard shortcut for the Command to 'None'.
Reset all Hotkeys to defaults button	Resets all keyboard shortcuts to the defaults (i.e. the values when Git Extensions was first installed).

2.4.12 Shell Extension

When installed, Git Extensions adds items to the context menu when a file/folder is right-clicked within Windows Explorer. One of these items is `'Git Extensions'` from which a further(cascaded) menu can be opened. This settings page identifies what items will appear on that cascaded menu.

Note: what is displayed also depends on what item is being right-clicked in Windows Explorer; a file or a folder(and whether the folder is a Git repository or not).

2.4.13 Advanced

This page allows advanced settings to be modified. Clicking on the '+' symbol on the tree of settings will display further settings. Refer *Confirmations*.

Group	Setting	Description
Checkout	Always show checkout dialog	Always show the Checkout Branch dialog when swapping branches. This dialog is normally only shown when uncommitted changes exist on the current branch
Use last chosen "local changes" action as default action.	This setting works in conjunction with the 'Git Extensions/Check for uncommitted changes in checkout branch dialog' setting. If the 'Check for uncommitted changes' setting is checked, then the Checkout Branch dialog is shown *only* if this setting is unchecked. If this setting is checked, then no dialog is shown and the last chosen action is used.	
General	Don't show help images	In the Pull dialog, images can be displayed to explain different scenarios. If checked, these Help images will not be displayed.

2.4.14 Confirmations

This page allows you to turn off certain confirmation popup windows.

Group	Setting	Description
Don't ask to confirm to	Amend last commit	If checked, do not display the popup warning about the rewriting of history when you have elected to amend the last committed change.
Apply stashed changes	In the Pull dialog, if `'Auto stash'` is checked, then any changes will be stashed before the pull is performed. Any stashed changes are then re-applied after the pull is complete. If this setting is checked, the stashed changes are applied with no confirmation popup.	
Push a new branch for the remote	When pushing a new branch that does not exist on the remote repository, a confirmation popup will normally be displayed. If this setting is checked, then the new branch will be pushed with no confirmation popup.	
Add a tracking reference for newly pushed branch	When you push a local branch to a remote and it doesn't have a tracking reference, you are asked to confirm whether you want to add such a reference. If this setting is checked, a tracking reference will always be added if it does not exist.	

2.4.15 Plugins

Plugins provide extra functionality for Git Extensions.

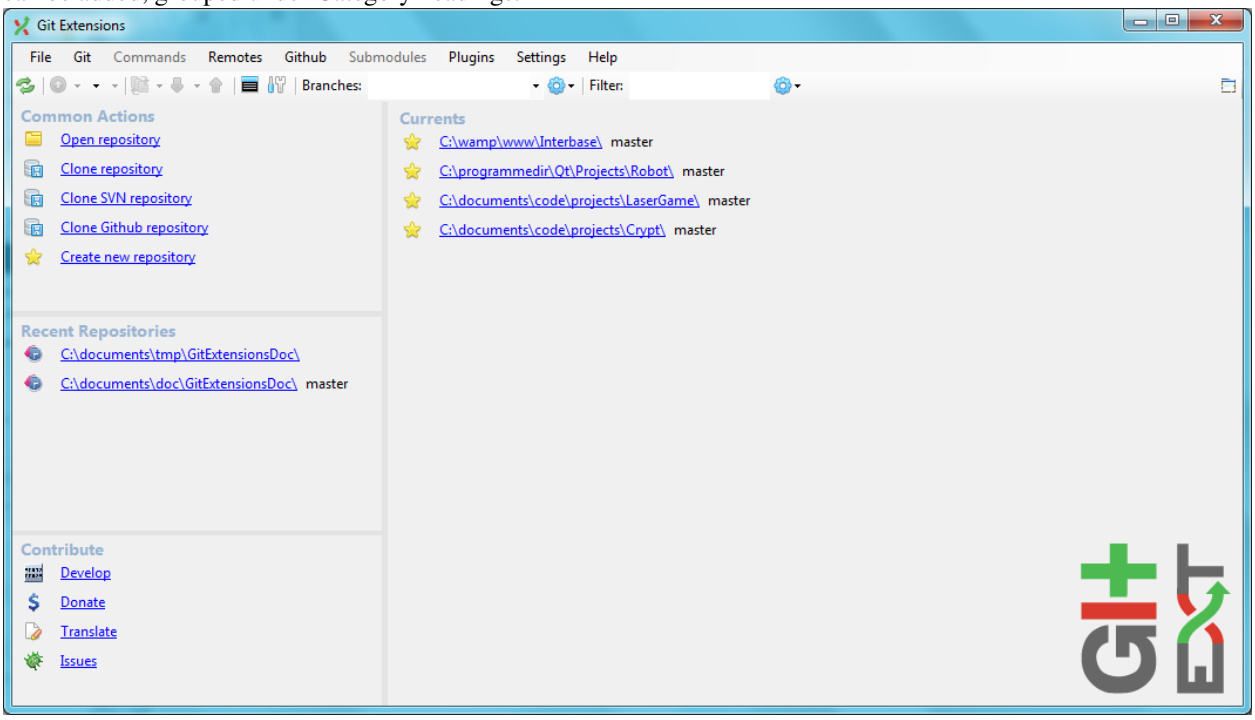
Plugin	Setting
Check for Updates	**This plugin is used by Git Extensions to check for updates to the Git Extensions s**
Enabled (true/false)	Enable or disable the check.
Check every # days	Check for updates after this number of days have elapsed since the last check.
Last check (yyyy/M/dd)	Shows date of the last check.
Auto compile SubModules	**This plugin proposes (confirmation required) that you automatically build submo**
Enabled (true/false)	Enter true to enable the plugin, or false to disable.
Path to msbuild.exe	Enter the path to the msbuild.exe executable.
msbuild.exe arguments	Enter any arguments to msbuild.
Create local tracking branches	**This plugin will create local tracking branches for all branches on a remote repos**
Delete obsolete branches	**This plugin allows you to delete obsolete branches i.e. those branches that are ful**
Delete obsolete branches older than (days)	Select branches created greater than the specified number of days ago.
Branch where all branches should be merged	The name of the branch where a branch *must* have been merged into to be considered
Find large files	**Finds large files in the repository and allows you to delete them.**
Find large files bigger than (Mb)	Specify what size is considered a 'large' file.
Gerrit Code Review	**The Gerrit plugin provides integration with Gerrit for GitExtensions. This plugi**
Github	**This plugin will create an OAuth token so that some common GitHub actions can**
OAuth Token	The token generated and retrieved from GitHub.
Impact Graph	**This plugin shows in a graphical format the number of commits and counts of ch**
Statistics	**This plugin provides various statistics (and a pie chart) about the current Git rep**
Code files	Specifies extensions of files that are considered code files.
Directories to ignore (EndsWith)	Ignore these directories when calculating statistics.
Ignore submodules (true/false)	Ignore submodules when calculating statistics.
gource	**Gource is a software version control visualization tool.**
Path to "gource"	Enter the path to the gource software.

Table 2.1 – c

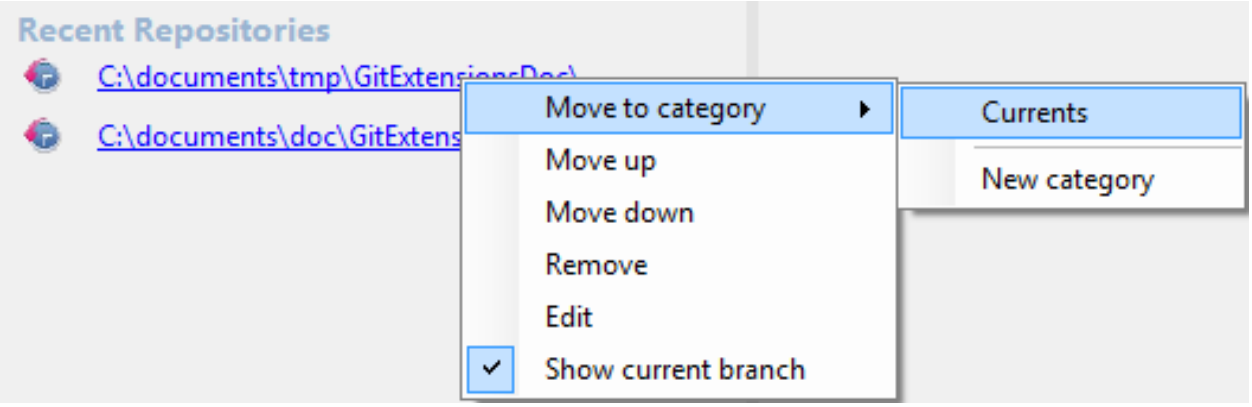
Plugin	Setting
Arguments	Enter any arguments to gource.
Username	**This plugin can set/unset the value for the http.proxy git config file key as per the** The user name needed to access the proxy.
Password	The password attached to the username.
HttpProxy	Proxy Server URL.
HttpProxyPort	Proxy Server port number.
Release Notes Generator	**This plugin will generate 'release notes'. This involves summarising all commits**

2.5 Start Page

The start page contains the most common tasks, recently opened repositories and favourites. The left side of the start page (Common Actions and Recent Repositories) is static. The right side of the page is where favourite repositories can be added, grouped under Category headings.



Recent Repositories can be moved to favourites using the repository context menu. Choose `Move to category / New category` to create a new category and add the repository to it, or you can add the repository to an existing category (e.g. 'Currents' as shown below).



A context menu is available for both the category and the repositories listed underneath it.

Entries on Category context menu

Move Up	Move the category (and any repositories under it) higher on the page.
Move Down	Move the category (and any repositories under it) lower on the page.
Remove	Remove the category (and any repositories under it) from the page. Note: Git repositories are *not* physically removed either locally or remotely.
Edit	Shows the Start Page settings window where both category and repository details can be modified. See *Start Page*.

Entries on repository context menu

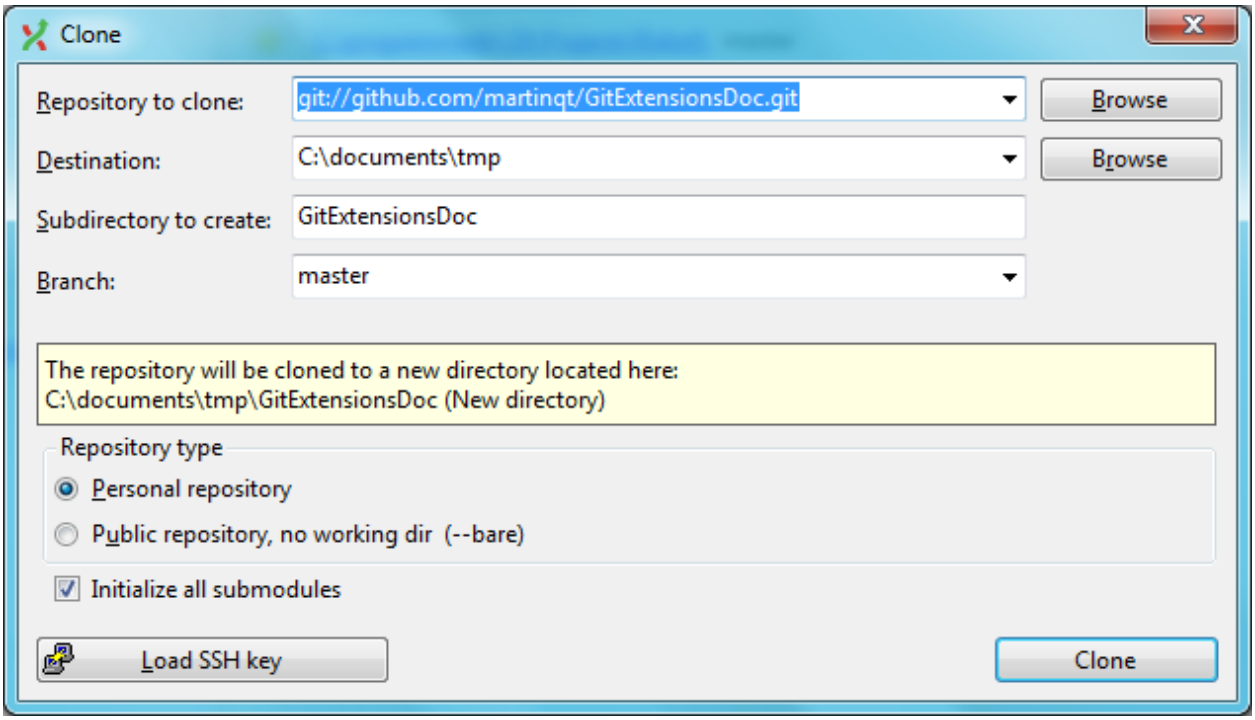
Move to category	Move the repository to a new or existing category.
Move up	Move the repository higher (within the category).
Move down	Move the repository lower (within the category).
Remove	Remove the repository from the category. Note: the repository is *not* physically removed either locally or remotely.
Edit	Shows the Start Page settings window where both category and repository details can be modified. See *Start Page*.
Show current branch	Toggles the display of the branch name next to the repository name. This identifies the currently checked out branch for the repository.

To open an existing repository, simply click the link to the repository under Recent Repositories or within the Categories that you have set up, or select Open repository (from where you can select a repository to open from your local file system).

To create a new repository, one of the following options under Common Actions can be selected.

2.6 Clone repository

You can clone an existing repository using this option. It displays the following dialog.



The repository you want to clone could be on a network share or could be a repository that is accessed through an internet or intranet connection. Depending on the protocol (http or ssh) you might need to load a SSH key into PuTTY. You also need to specify where the cloned repository will be created and the initial branch that is checked out. If the cloned repository contains submodules, then these can be initialised using their default settings if required.

There are two different types of repositories you can create when making a clone. A personal repository contains the complete history and also contains a working copy of the source tree. A central repository is used as a public repository where developers push the changes they want to share with others to. A central repository contains the complete history but does not have a working directory like personal repositories.

2.7 Clone SVN repository

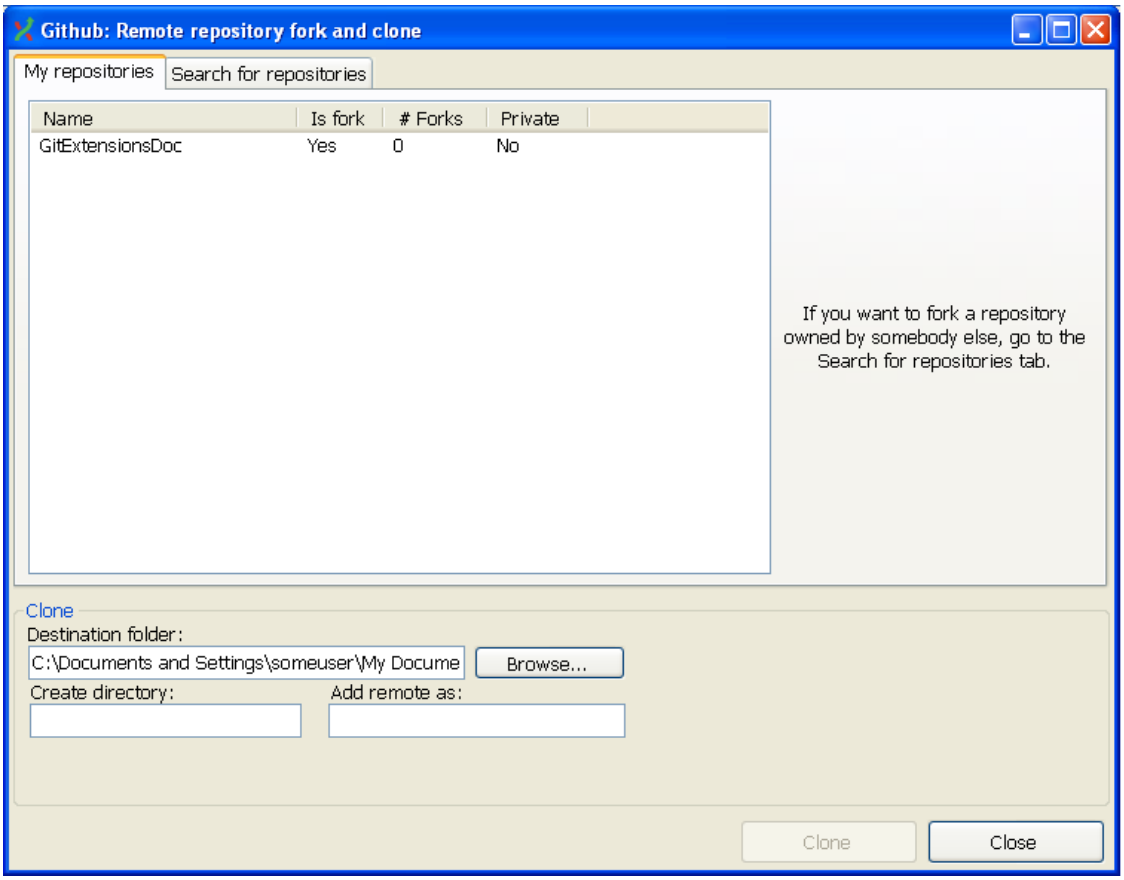
You can clone an existing SVN repository using this option, which creates a Git repository from the SVN repository you specify. For further information refer to the Pro Git book.

2.8 Clone Github repository

This option allows you to

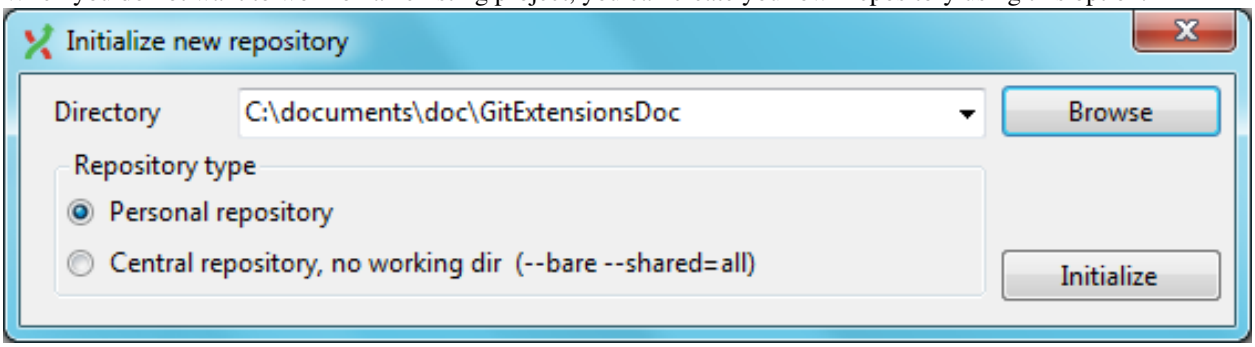
1. Fork a repository on GitHub so it is created in your personal space on GitHub.
2. Clone any repositories on your personal space on GitHub so that it becomes a local repository on your machine.

You can see your own personal repositories on GitHub, and also search for repositories using the `Search for repositories` tab.



2.9 Create new repository

When you do not want to work on an existing project, you can create your own repository using this option.



Select a directory where the repository is to be created. You can choose to create a Personal repository or a Central repository.

A personal repository looks the same as a normal working directory but has a directory named `.git` at the root level containing the version history. This is the most common repository.

Central repositories only contain the version history. Because a central repository has no working directory you cannot checkout a revision in a central repository. It is also impossible to merge or pull changes in a central repository. This repository type can be used as a public repository where developers can push changes to or pull changes from.

CHAPTER

THREE

BROWSE REPOSITORY

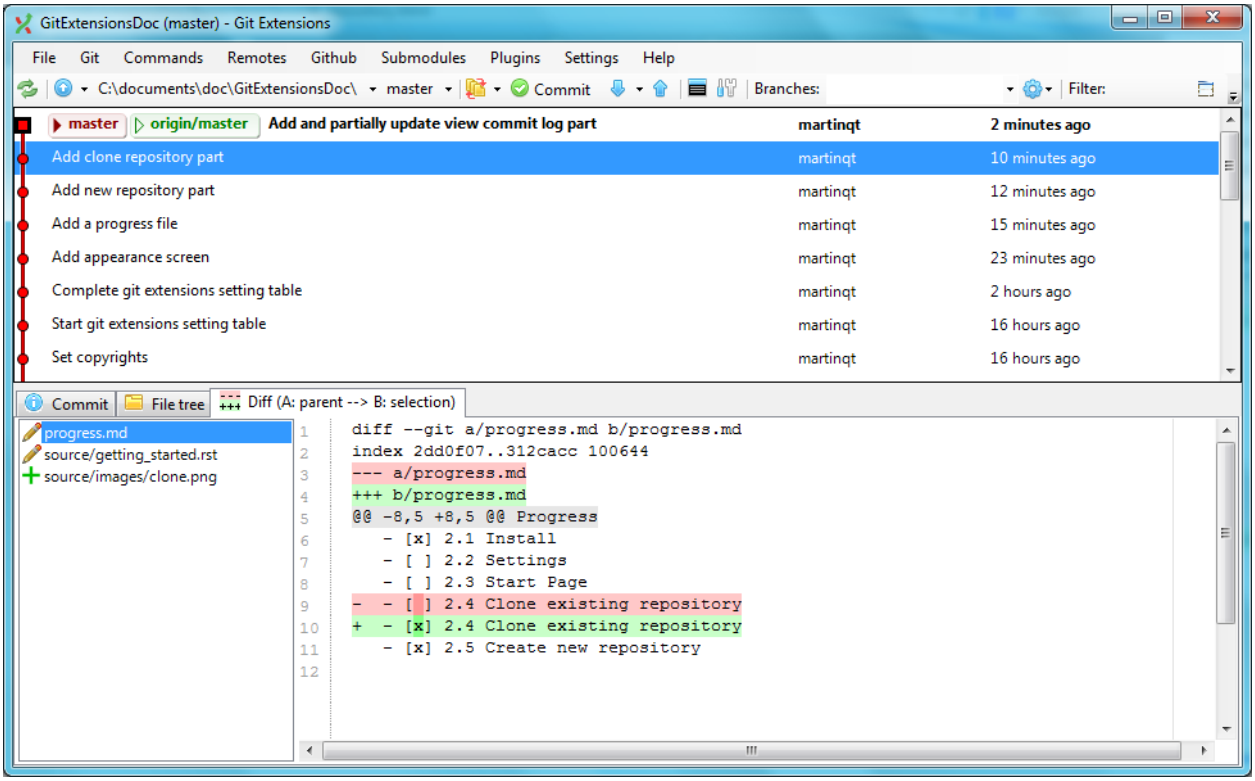
You can browse a repository by starting Git Extensions and selecting the repository to open from the *Start Page*. The Commit Log window is then displayed, which is the main window in Git Extensions. You can also open this window from the shell extensions and from the Visual Studio IDE.

3.1 Commit Log Window

The Commit Log window consists of a standard Windows Menu Bar, a Toolbar and the main window, which is split into two parts

- the commit history and graph that shows branches and merges
- three Tabs: Commit, File tree and Diff that display information about the currently highlighted commit(s) in the commit history

The commit history shows every commit to the repository (or the number of commits specified by the *Git Extensions* Setting that limits the number of commits, whichever is the lower).



3.1.1 Toolbar

The Toolbar consists of a number of buttons and text fields as described below. The items on the Toolbar and their positions are fixed and are not user-configurable.

Refresh, Refresh (Repository is 'dirty')

This is the first button on the toolbar and you will see one of the above icons. Its function is to force Git Extensions to look at the Git repository and refresh itself based on any commits, index changes etc. that have been done outside of the Git Extensions GUI (e.g. via the command line).

Note: the 'dirty' icon will only be shown for index changes if you have enabled the *Git Extensions* 'Use FileSystemWatcher' setting.

Alternatives to this button:

- pressing the `F5` key
- selecting *File* → *Refresh* from the Menu Bar.

Go to superproject TODO

Refer to the *Submodules* chapter for further information.

Change working directory

This button displays the repository that Git Extensions is currently working with. Clicking on this button will display a dropdown menu where you can

- swap to recent repositories you have accessed
- open the *Open local repository* dialog to search for a local repository
- configure this dropdown menu

Alternatives to this button:

- pressing the `Ctrl+O` key combination to open the *Open local repository* dialog
- selecting *File* → *Open* from the Menu Bar to open the *Open local repository* dialog
- selecting *File* → *Close* from the Menu Bar to close this repository and return you to the *Start Page* where a new repository can be selected
- selecting *File* → *Recent Repositories* from the Menu Bar where a list of recent repositories will be presented

Configuring this dropdown menu will present you with the following configuration options:

Group	Setting	Description
Sort most recent repositories alphabetically	Maximum number of most recent repositories	Sets the maximum number of recent repositories.
	Sorts entries in Most recent repositories combobox in alphabetic order.	
Sort less recent repositories alphabetically	Sorts entries in Less recent repositories combobox in alphabetic order.	
Shortening strategy	Do not shorten	Do not shorten the repository path as shown on the toolbar button.
The most significant directory	Displays the last entry in the path on the toolbar button. This will be the repository name.	
Replace middle part with dots	Shows the first and last parts of the repository path, with the middle bit replaced with dots.	
	Combobox minimum width	Allows you to specify the width of the part of this dialog that shows the Most/Less recent repositories comboboxes. Specifying 0 means this dialog box will expand horizontally to the largest of the repository paths.

If you select a repository in either the Most or Less recent repositories combobox, you can right-click to display a context menu with the following options:

Option	Description
Anchor to most recent repositories	Moves the repository to the Most recent repositories combobox.
Anchor to less recent repositories	Moves the repository to the Less recent repositories combobox.
Remove anchor	If this repository is selected (i.e. highlighted), it un-selects it.
Remove from recent repositories	Removes this repository from the combobox.

Change current branch

This button displays the currently checked out branch. Clicking on this button will display a dropdown menu where you can

- select a new branch to switch to from the displayed list of branches that exist on your local repository.
- open the *Checkout branch* dialog

Alternatives to this button:

- pressing the `Ctrl+.` key combination to open the *Checkout branch* dialog
- selecting *Commands* → *Checkout branch* from the Menu Bar to open the *Checkout branch* dialog
- access the context menu by right-clicking a commit that is in a branch, then selecting *Checkout branch* →. The list of branches that commit is in will be displayed and you can select one to checkout.

Refer to the *Branches* chapter for further information.

Stash changes

This button allows you to work with the Stash. Note that this button also has a dropdown menu that operates independently from the button. If you click on the button it will open a dialog where the Stash can be manipulated. If you open the dropdown menu you can

- stash current working directory changes
- pop a saved stash ie restore working directory to contents of the stash
- open the *Stash* dialog

Note: If you have enabled the *Git Extensions* 'Show stash count on status bar' setting then the number of saved stashes will be displayed next to this button.

Alternatives to this button:

- selecting *Commands* → *Stash changes* from the Menu Bar to open the *Stash* dialog

Refer to the *Commit* chapter for further information.

Commit, Commit (pending commit)

This button will open the *Commit* dialog where any uncommitted changes can be committed to the repository. The first button is displayed when there are no uncommitted changes in the working directory. The second button style indicates there are uncommitted changes and the number of those changes.

Note: the number of uncommitted changes is only displayed if you have enabled the *Git Extensions* 'Show repository status in browse dialog' setting.

Alternatives to this button:

- pressing the `Ctrl+Space` key combination to open the *Commit* dialog
- selecting *Commands* → *Commit* from the Menu Bar to open the *Commit* dialog

Refer to the *Commit* chapter for further information.

Open pull dialog, Pull - merge, Pull - rebase, Pull - fetch, Pull - fetch all

This button allows you to retrieve changes from a remote repository and apply them to your local repository. When Git Extensions is first installed, the default button displayed on the toolbar is **Open pull dialog** which will display the *Pull* dialog when clicked. This default can be changed. This button also has a dropdown menu that operates independently from the button, and when opening it you are able to

- **Merge** Fetch changes from a remote repository and merge into your local branch. Before this button is clicked, you must have checked out the local branch you are pulling to, and that local branch must be the local tracking branch for the remote repository.
- **Rebase** Fetch changes from a remote repository and rebase any local branch changes on top of the remote branch changes. As above, the local branch must be checked out and be a local tracking branch.
- **Fetch** Fetch all changes from a remote repository to your local repository, updating the remote references. If the currently checked out branch is a remote tracking branch, then the fetch is done from that remote. If the checked out branch is not a remote tracking branch, then the fetch is done from the remote called origin (if it exists).
- **Pull** Opens the *Pull* dialog
- **Fetch all** Fetch all changes from *all* remote repositories defined in your local repository. All remote references are updated.

Note: Selecting one of the above options (except the Pull option), either from the dropdown menu or when it is the default button on the toolbar, causes immediate execution of the command. There is no confirmation dialog.

Warning: Selecting **Rebase** may rewrite history on your local repository. This is not recommended if you have already published that history elsewhere.

When selecting an option from the dropdown menu, selecting an item above the divider (i.e. Merge, Rebase or Fetch) will result in the selection becoming the new default button on the toolbar.

The **don't set as default** menu item is a checkbox that only applies to the items below the divider (i.e. Pull and Fetch all). It behaves as follows:

- if checked, clicking Pull or Fetch all will *not* result in them becoming the default on the toolbar.
- if unchecked, clicking Pull or Fetch all *will* result in the selection becoming the default on the toolbar.

Alternatives to this button:

- pressing the `Ctrl+Down` key combination
- selecting *Commands* → *Pull* from the Menu Bar to open the *Pull* dialog

Refer to the *Remote feature* chapter for further information.

Push changes

This button will open the *Push* dialog where changes on your local repository can be sent to a remote repository.

Alternatives to this button:

- pressing the `Ctrl+Up` key combination
- selecting *Commands* → *Push* from the Menu Bar to open the *Push* dialog

Refer to the *Remote feature* chapter for further information.

Git bash

This button will open a bash window. In Linux, a bash shell is roughly equivalent to the Windows DOS Command shell. The bash shell allows you to enter git commands directly. For example:

```
Welcome to Git (version 1.8.3-preview20130601)


Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

user@VBOX-XP ~/My Documents/GitExtensionsDoc (fix/Chapter3)
$ git branch -v
  TestDocBuild 0839935 Updated version to 2.46
* fix/Chapter3 3d606e7 working on main window
  latest       0839935 Updated version to 2.46
  master       6c40f7a Merge branch 'latest'

user@VBOX-XP ~/My Documents/GitExtensionsDoc (fix/Chapter3)
$
```

Alternatives to this button:

- pressing the `Ctrl+G` key combination
- selecting *Git* → *Git bash* from the Menu Bar

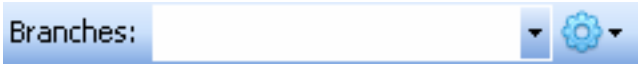
Settings

This button will open the settings dialog window.

Alternatives to this button:

- selecting *Settings* → *Settings* from the Menu Bar

Refer to *Settings* for further information.



Branches filter

This filter consists of a text box and associated dropdown list and a 'gear' icon. It is used to filter the commit history display so that commits that are part of the selected branch are the only ones displayed. You can either

- type a branch name in the check box or,
- select a branch name from the dropdown list

The branch name entries displayed in the dropdown list are also affected by the item(s) selected from the dropdown menu associated with the 'gear' icon. You can select local and/or remote branches.

Note: The commit history display is *not* updated until you press the `Enter` key, regardless of whether you type in a branch name or select one from the dropdown menu.

Refer to *Searching and Filtering* for further information.

Filter

This filter consists of a text box and associated 'gear' icon. It is used to filter the commit history display for the matching text entered in the checkbox. The dropdown associated with 'gear' icon determines what component of the commit is searched.

Note: The commit history display is *not* updated until you press the `Enter` key.

Refer to *Searching and Filtering* for further information.

Toggle split view layout

This button toggles between displaying a split screen view or only displaying the commit history and graph full screen.

Alternatives to this button:

- clicking on and dragging the divider between the two views - this will adjust the size of each view

3.1.2 Commit History and Graph

** TODO **

3.1.3 Commit Tab

The commit tab contains information about the commit that is currently selected in the commit history.

cover - image and context menu - author info - commit message - Signed off by - Contained in: branches and tag - context menu

3.1.4 File Tree Tab

** TODO **

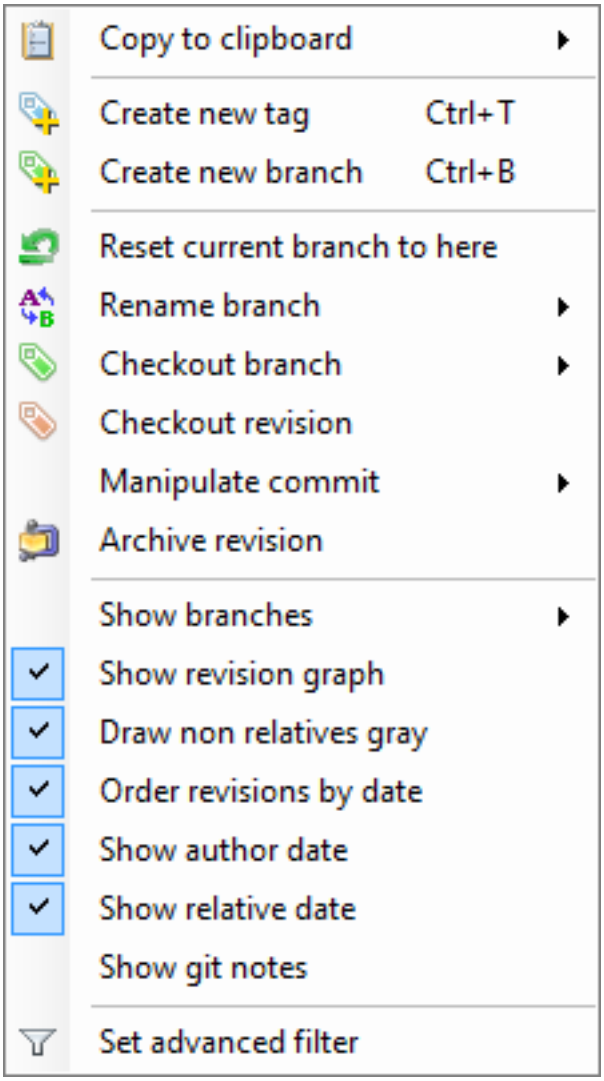
3.1.5 Diff Tab

** TODO **

===== existing doco =======

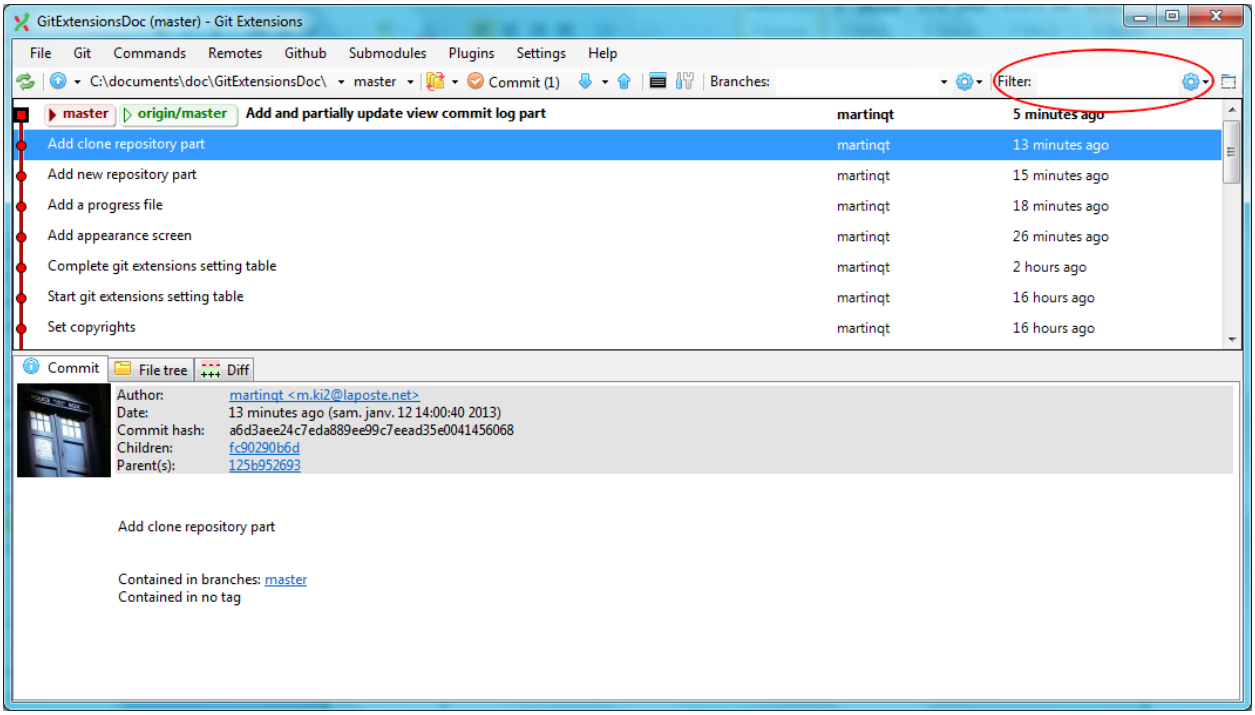
The full commit history can be browsed. There is a graph that shows branches and merges. You can show the difference between two revision by selection them using ctrl-click.

In the context menu of the commit log you can enable or disable the revision graph. You can also choose to only show the current branch instead of showing all branches. The other options will be discussed later.

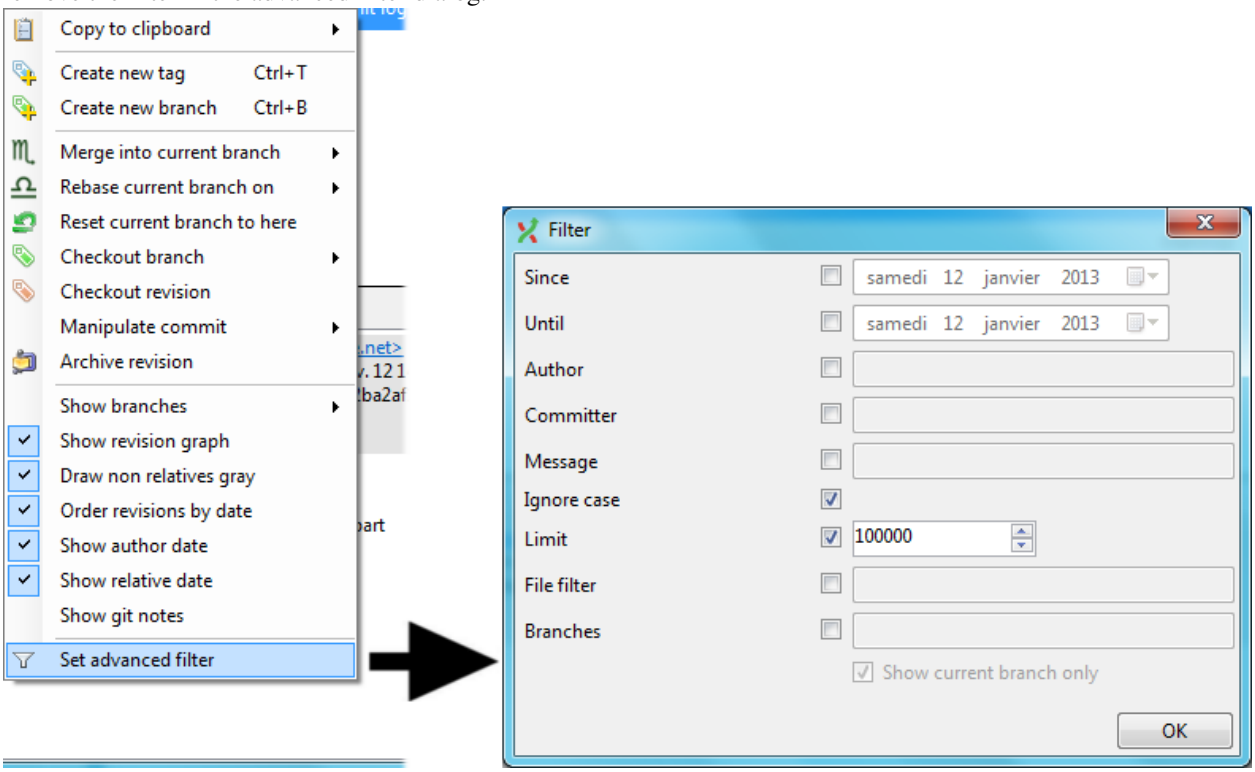


3.2 Searching and Filtering

The history can be searched using regular expressions are basic search terms. The quick filter in the toolbar searches in the commit message, the author and the committer.

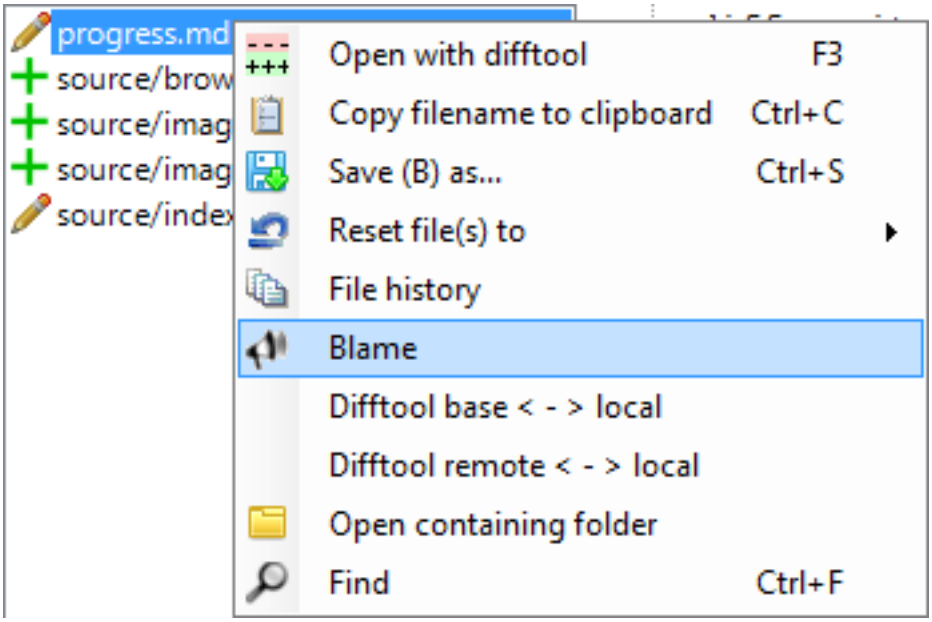


In the context menu of the commit log you can open the advanced filter dialog. The advanced filter dialog allows you to search for more specific commits. To remove the filter either remove the filter in the toolbar and press enter or remove the filter in the advanced filter dialog.

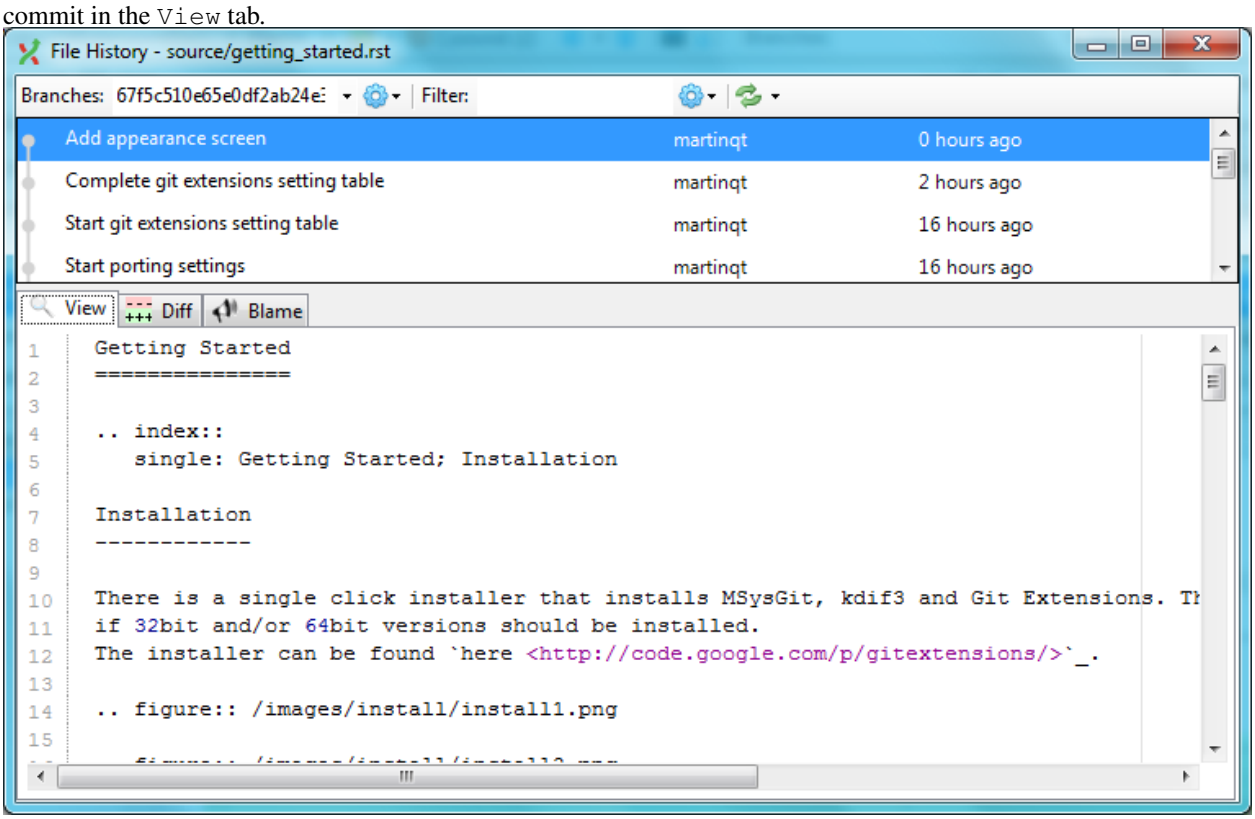


3.3 Singe file history

To display the single file history, right click on a file name in the `File tree` or in the `Diff` tab and select `blame`.

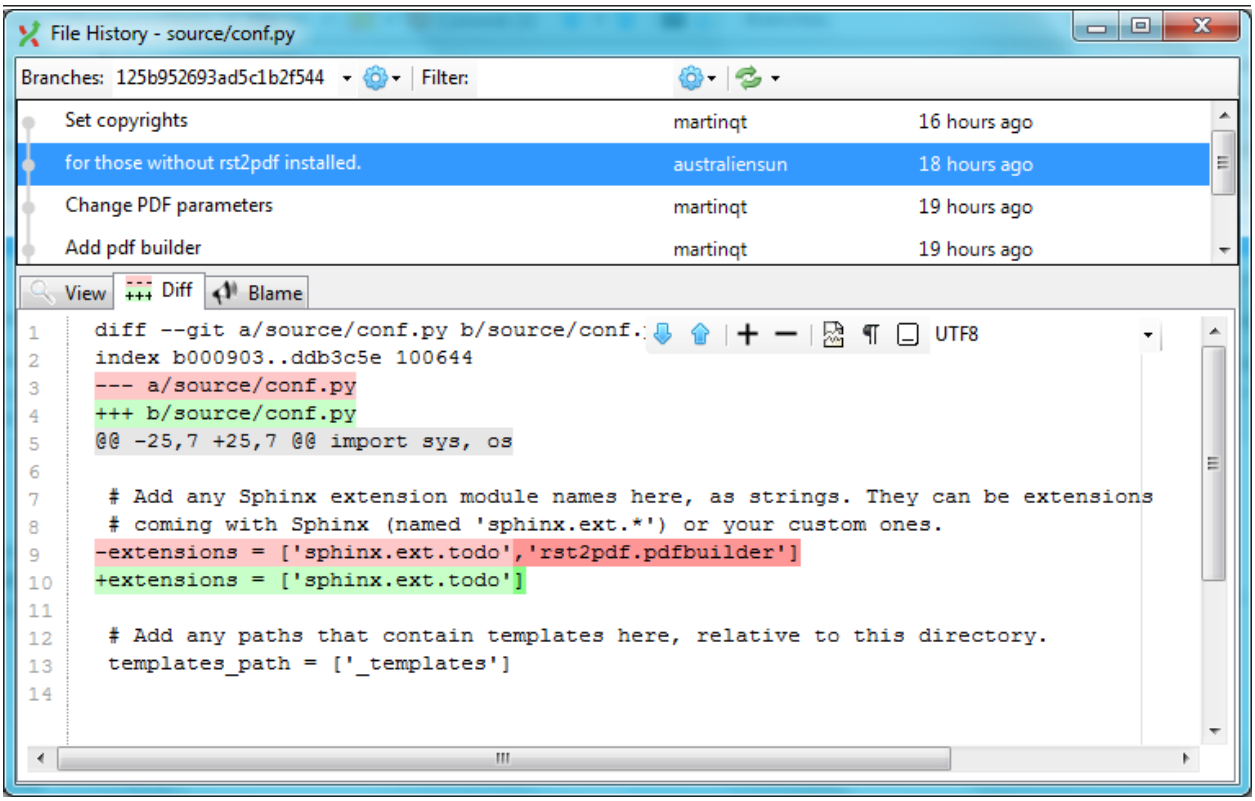


The single file history viewer shows all revisions of a single file. You can view the content of the file in after each commit in the `View` tab.



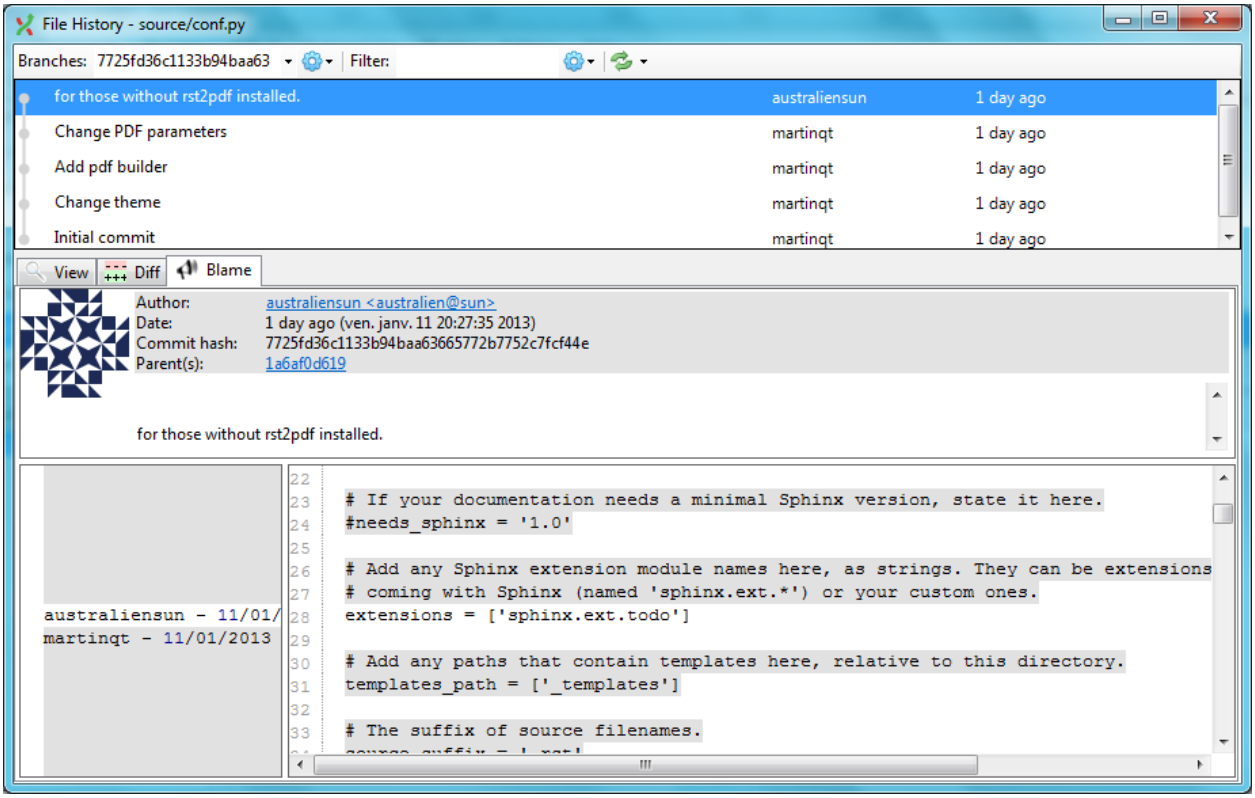
You can view the difference report from the commit in the `Diff` tab.

Note: Added lines are marked with a +, removed lines are marked with a −.



3.4 Blame

There is a blame function in the file history browser. It shows the last person editing a single line.



Double clicking on a code line shows the full commit introducing the change.

CHAPTER

FOUR

COMMIT

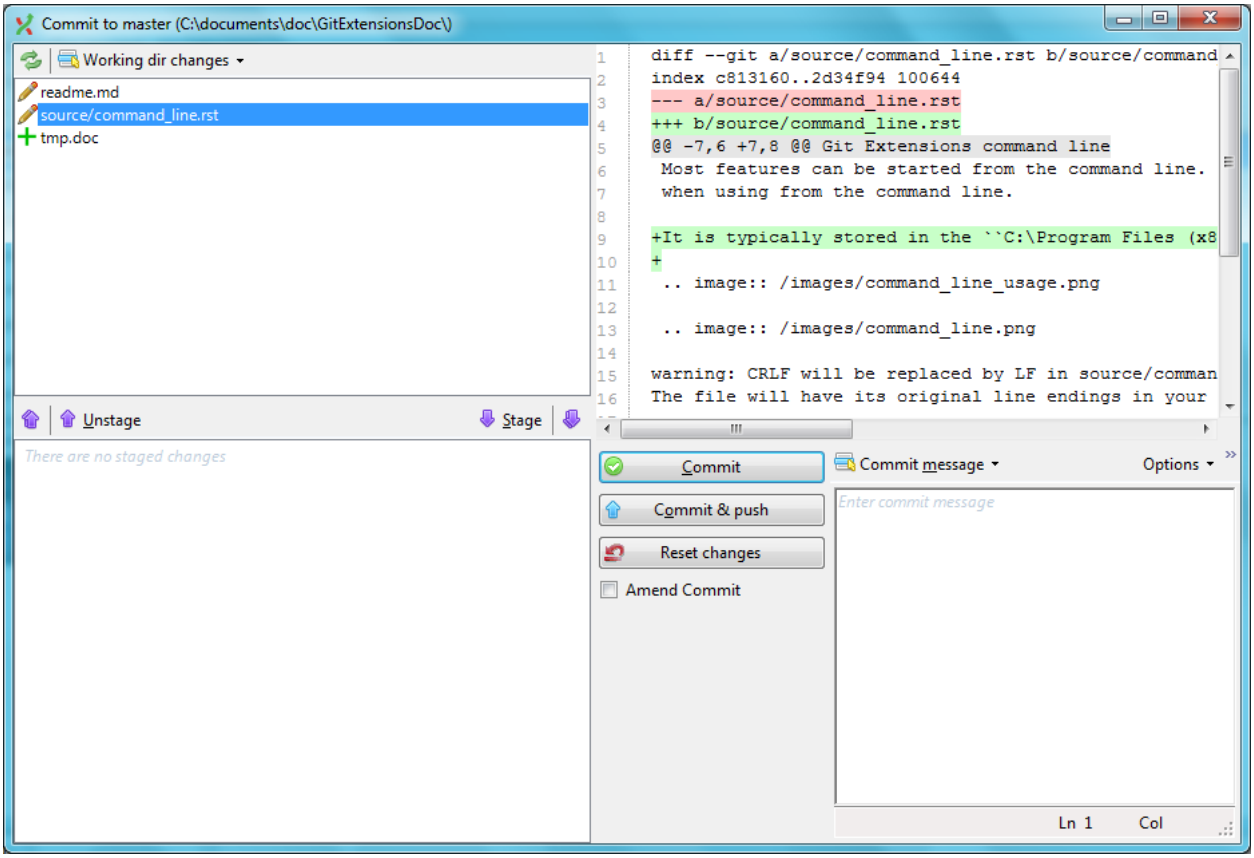
A commit is a set of changes with some extra information. Every commit contains the follow information:

- Changes
- Committer name and email
- Commit date
- Commit message
- Cryptographically strong SHA1 hash

Each commit creates a new revision of the source. Revisions are not tracked per file; each change creates a new revision of the complete source. Unlike most traditional source control management systems, revisions are not named using a revision number. Each revision is named using a SHA1, a 41 long characters cryptographically strong hash.

4.1 Commit changes

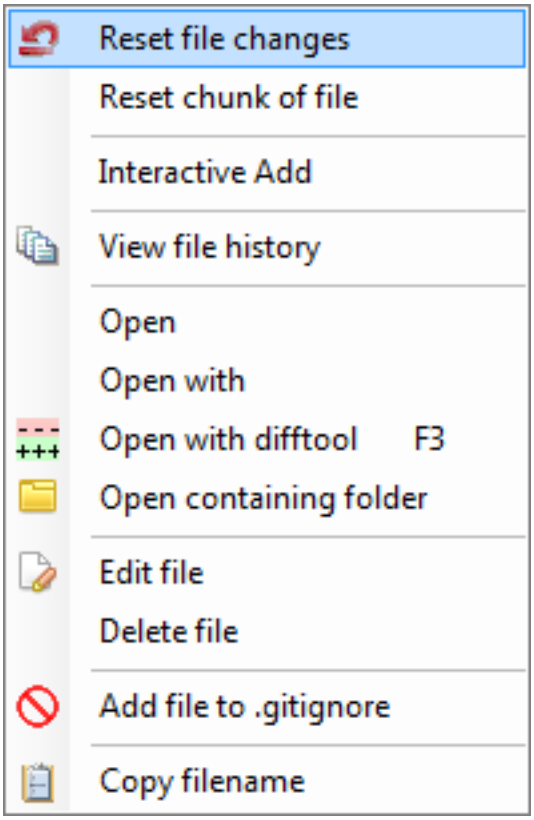
Changes can be committed to the local repository. Unlike most other source control management systems you do not need to checkout files before you start editing. You can just start editing files, and review all the changes you made in the commit dialog later. When you open de commit dialog, all changes are listed in the top-left.



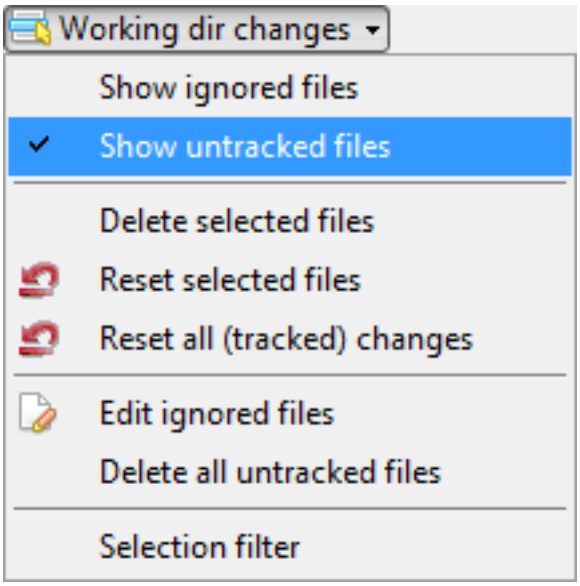
There are three kinds of changes:

Un-tracked	This file is not yet tracked by Git. This is probably a new file, or a file that has not been committed to Git before.
Modified	This file is modified since the last commit.
Deleted	This file has been deleted.

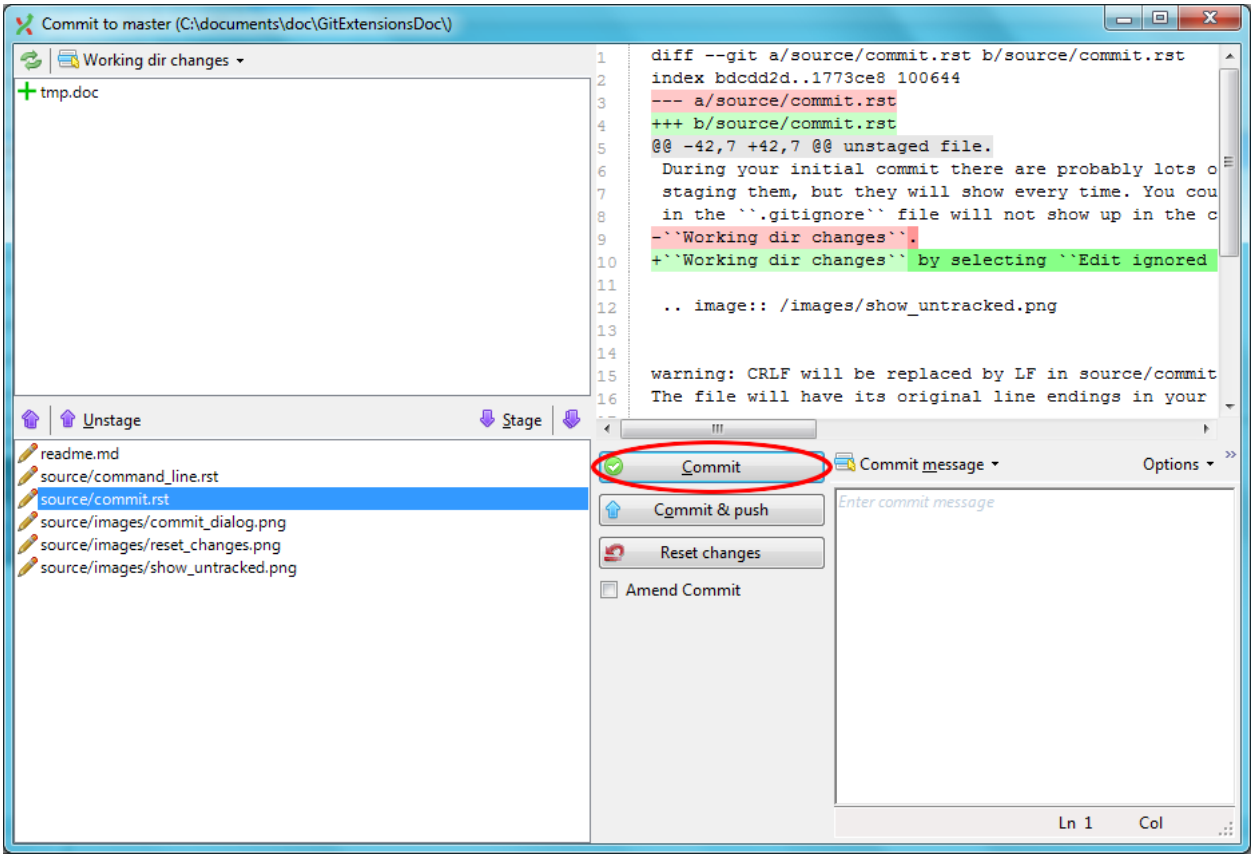
When you rename or move a file Git will notice that this file has been moved, but currently Git Extensions does not show this in the commit dialog. Occasionally you will need to undo the file change. This can be done in the context menu of any unstaged file.



During your initial commit there are probably lots of files you do not want to be tracked. You can ignore these files by not staging them, but they will show every time. You could also add them to the .gitignore file of you repository. Files that are in the `.gitignore` file will not show up in the commit dialog again. You can open the `.gitignore` editor from the menu `Working dir changes` by selecting `Edit ignored files`.

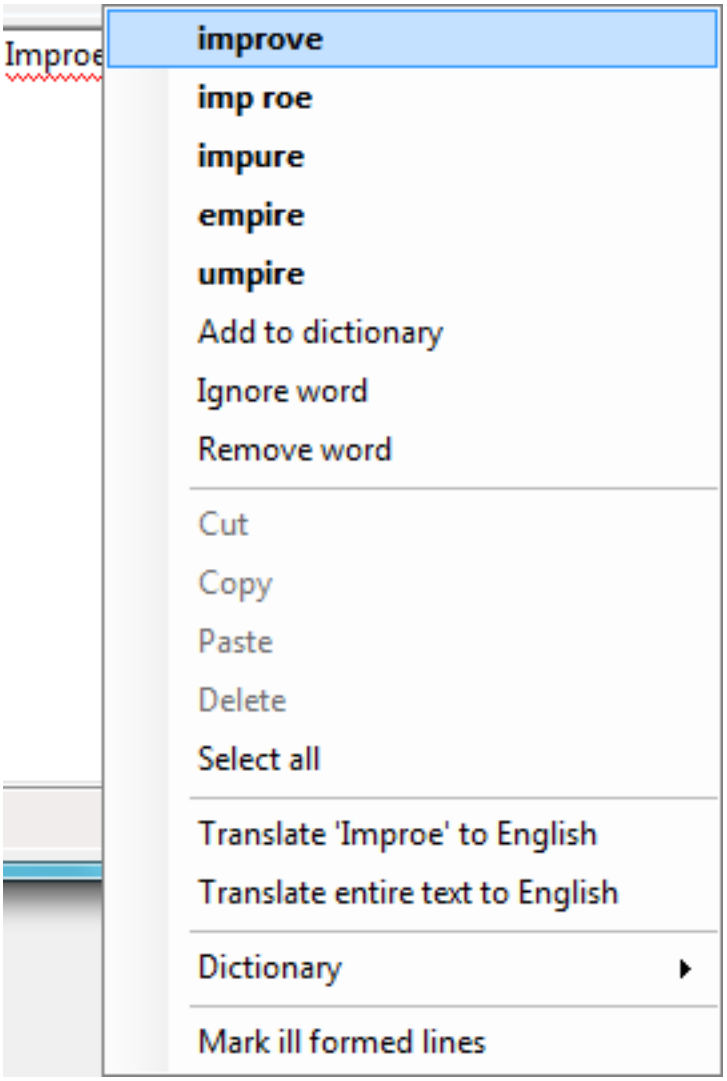


You need to stage the changes you want to commit by pressing the 'Stage selected files' button. You also need to stage deleted files because you stage the change and not the file. When all the changes you want to commit are staged, enter a commit message and press the commit button.

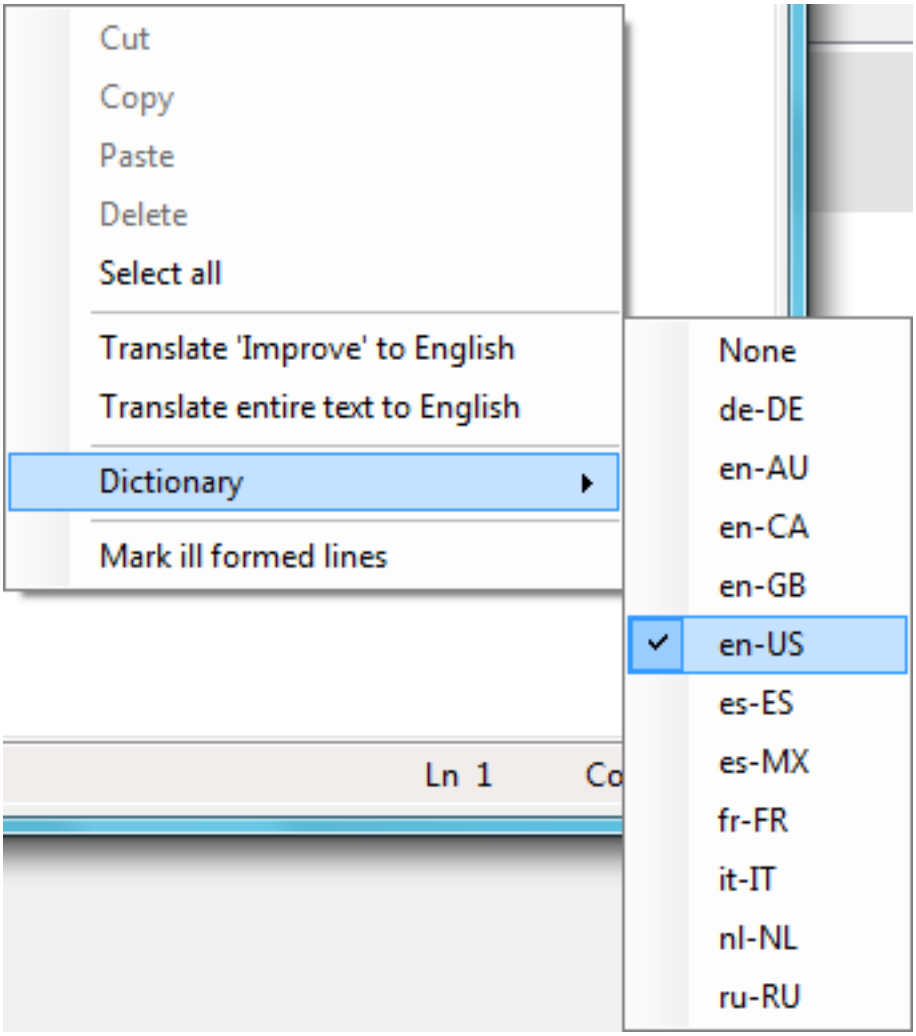


It is also possible to add files to you last commit using the `Amend to last commit` button. This can be very useful when you forgot some changes. This function rewrites history; it deletes the last commit and commits it again including the added changes. Make sure you only use `Amend to last commit` when the commit is not yet published to other developers.

There is a build in spelling checker that checks the commit message. Incorrect spelled words are underlined with a red wave line. By right-clicking on the misspelled word you can choose the correct spelling or one of the other options.

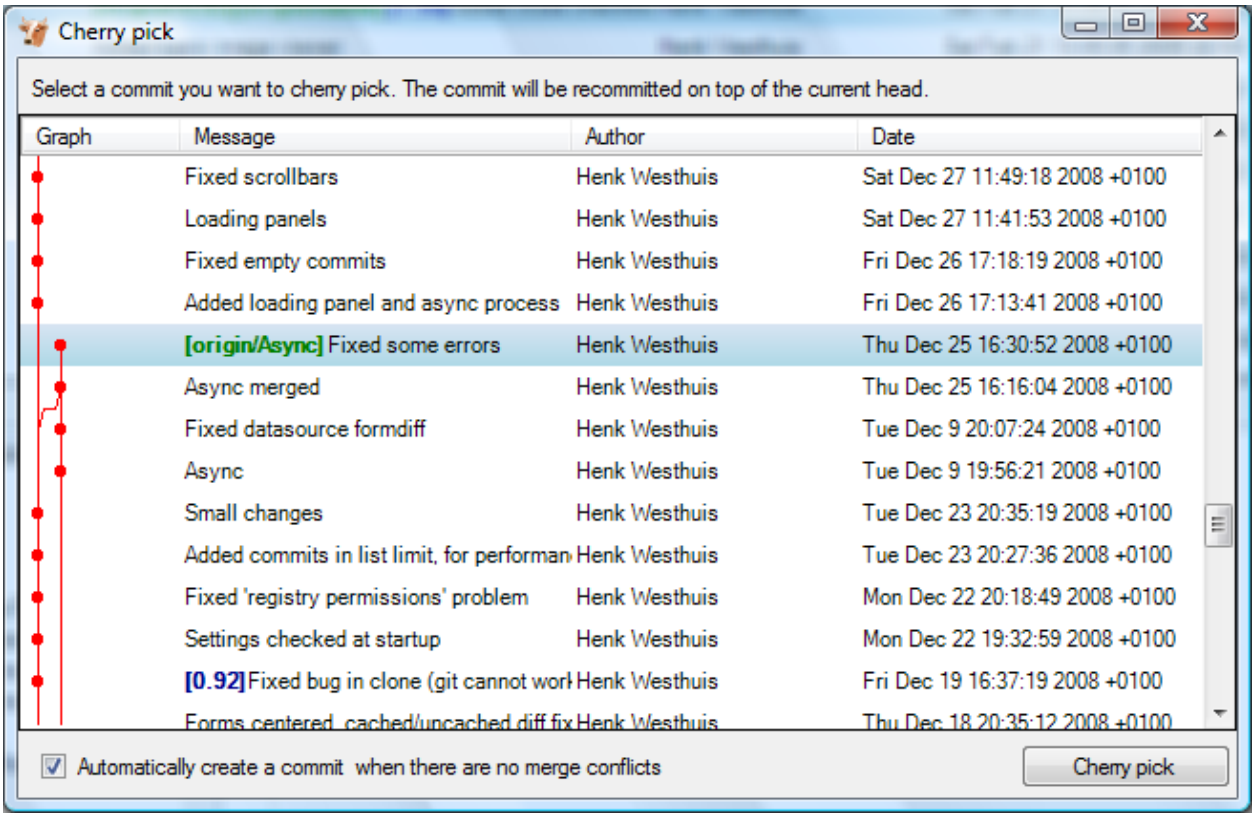


Git Extensions installs a number of dictionaries by default. You can choose another language in the context menu of the spelling checker or in the settings dialog. To add a new spelling dictionary add the dictionary file to the `Dictionaries` folder inside the Git Extensions installation folder.



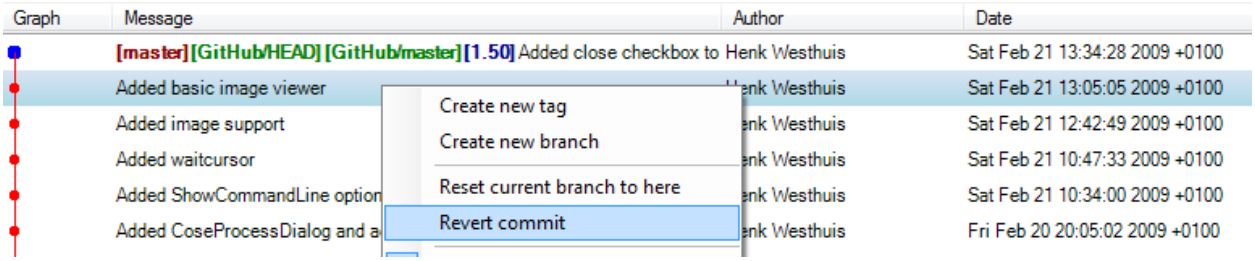
4.2 Cherry pick commit

A commit can be recommitted by using the cherry pick function. This can be very useful when you want to make the same change on multiple branches.



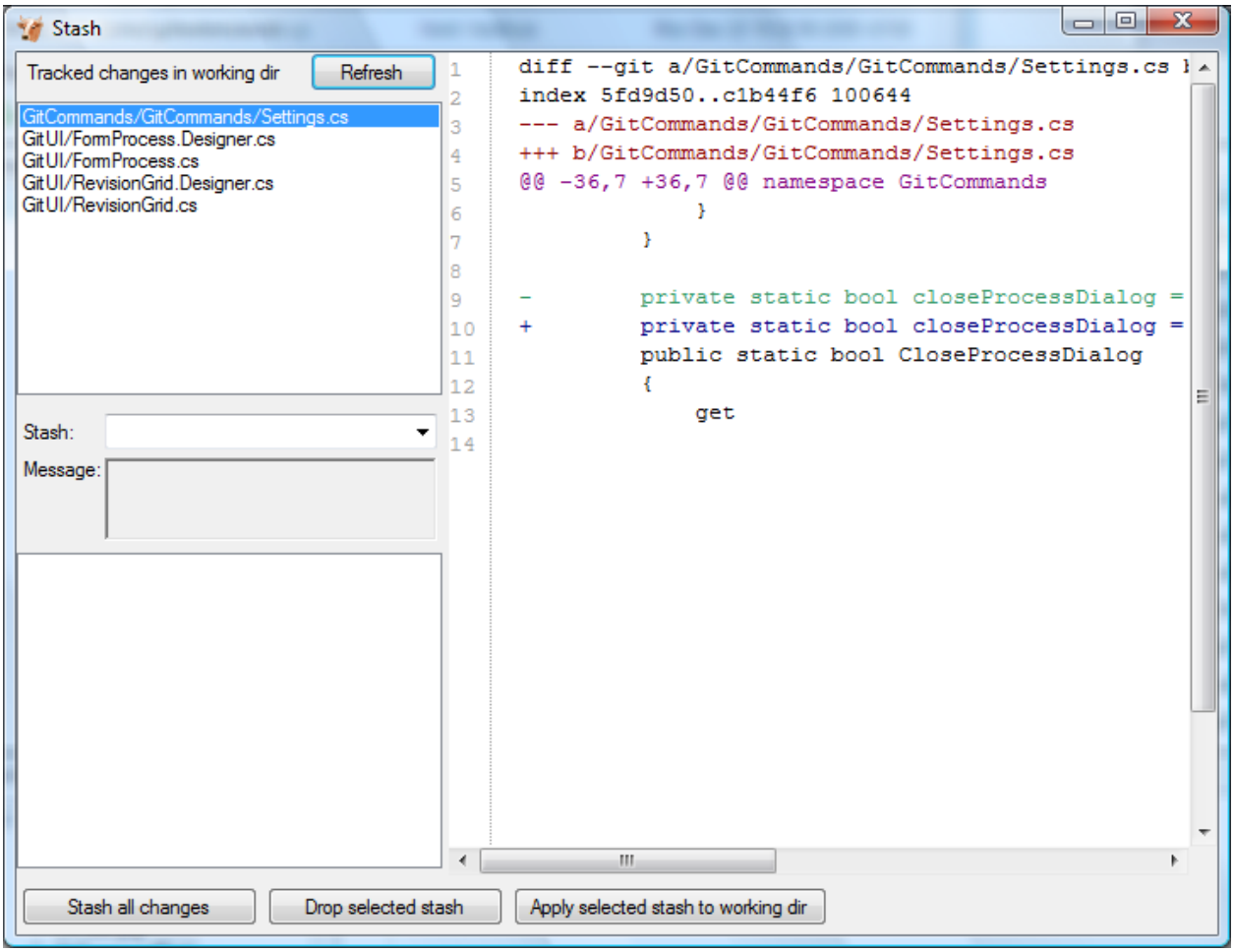
4.3 Revert commit

A commit cannot be deleted once it is published. If you need to undo the changes made in a commit, you need to create a new commit that undoes the changes. This is called a revert commit.

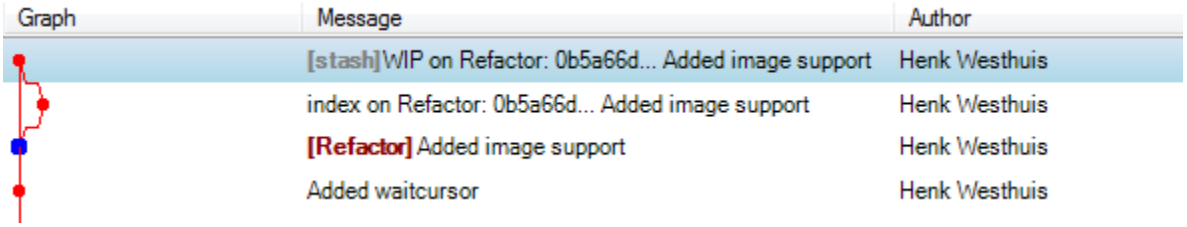


4.4 Stash changes

If there are local changes that you do not want to commit yet and not want to throw away either, you can temporarily stash them. This is useful when working on a feature and you need to start working on something else for a few hours. You can stash changes away and then reapply them to your working dir again later. Stashes are typically used for very short periods.



You can create multiple stashes if needed. Stashes are shown in the commit log with the text `[stash]`.



The stash is especially useful when pulling remote changes into a dirty working directory. If you want a more permanent stash, you should create a branch.

CHAPTER

FIVE

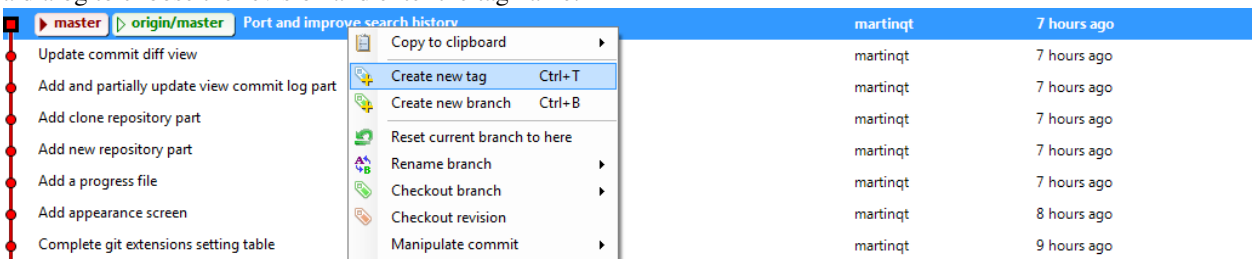
TAG

Tags are used to mark a specific version. Usually a tag will not be moved anymore. The image below shows the commit log of Git Extensions with two tags indicating version [1.08] and [1.06].

Graph	Message	Author	Date
	Fixed open working dir with spaces from VS and shell extensions and addec	Henk Westhuis	Thu Jan 8 19:04:51 2009 +0100
	Added plugin to setup	Henk Westhuis	Wed Jan 7 20:23:30 2009 +0100
	[1.08] Minor changes for version 1.08	Henk Westhuis	Tue Jan 6 19:27:35 2009 +0100
	Added archive function	Henk Westhuis	Tue Jan 6 19:22:50 2009 +0100
	Fixed using " (quote) in commit message	Henk Westhuis	Tue Jan 6 18:51:50 2009 +0100
	Fixed commits per user and added "show files to add"	Henk Westhuis	Tue Jan 6 18:48:57 2009 +0100
	Fixed directory select clone form	Henk Westhuis	Tue Jan 6 18:27:10 2009 +0100
	Added progress dialog to stash	Henk Westhuis	Mon Jan 5 19:58:12 2009 +0100
	Fixed formatpatch dialog	Henk Westhuis	Mon Jan 5 19:46:37 2009 +0100
	Added setting to locate git.cmd	Henk Westhuis	Mon Jan 5 19:25:43 2009 +0100
	Added dll's to make it easier for others to compile	Henk Westhuis	Mon Jan 5 19:25:15 2009 +0100
	[PATCH] Quote path when calling regedit.	Henk Westhuis	Mon Jan 5 17:52:52 2009 +0100
	[1.06] Fixed reset hard and fixed checkout dialog	Henk Westhuis	Sun Jan 4 16:16:16 2009 +0100
	Deleted mailmap... it was just there to test	Henk Westhuis	Sun Jan 4 15:36:24 2009 +0100

5.1 Create tag

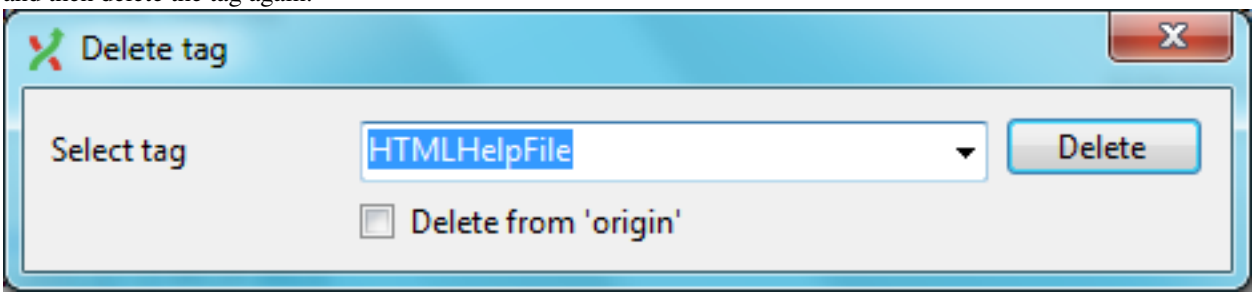
In Git Extensions you can tag a revision by choosing `Create new tag` in the commit log context menu. A dialog will prompt for the name of the tag. You can also choose `Create tag` from the `Commands` menu, which will show a dialog to choose the revision and enter the tag name.



Once a tag is created, it cannot be moved again. You need to delete the tag and create it again to move it.

5.2 Delete tag

For some operation it is very useful to create tags for temporary usage. Git uses SHA1 hashes to name each commit. When you want to merge with an unnamed branch it is good practise to tag the unnamed branch, merge with the tag and then delete the tag again.



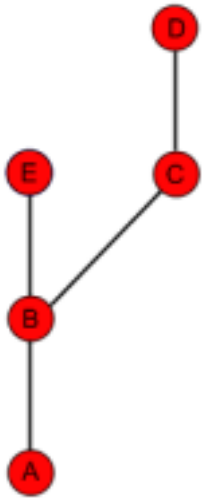
5.2.1 Re-Tag?

Read about "What should you do when you tag a wrong commit and you would want to re-tag?" here: https://www.kernel.org/pub/software/scm/git/docs/git-tag.html#_on_re_tagging

CHAPTER

SIX

BRANCHES



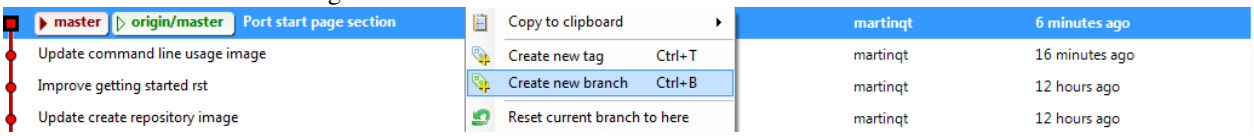
Branches are used to commit changes separate from other commits. It is very common to create a branch when you start working on a feature and you are not sure if this feature will be finished in time for the next release. The image on the right illustrates a branch created on top of commit B.

In Git branches are created very often. Creating a branch is very easy to do and it is recommended to create a branch very often. In fact, when you make a commit to a cloned repository you start a new branch. I will explain this in the pull chapter.

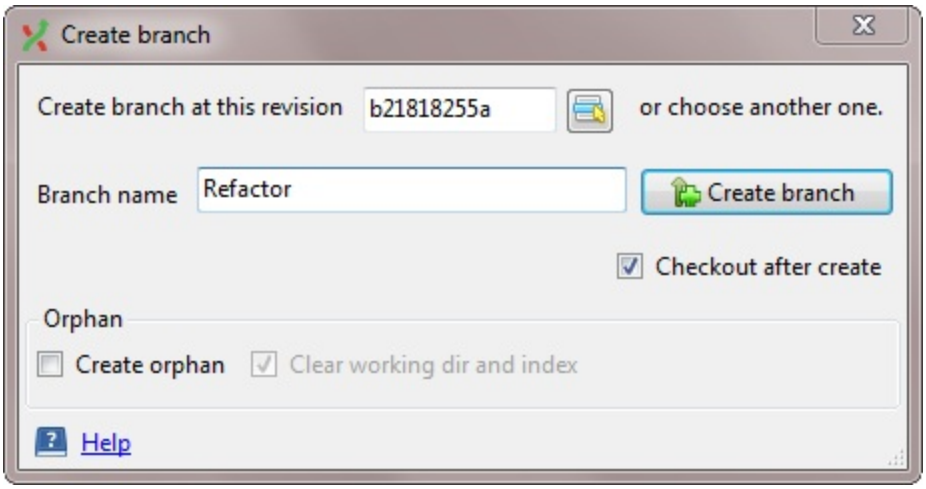
You can check on what branch you are working in the toolbar.

6.1 Create branch

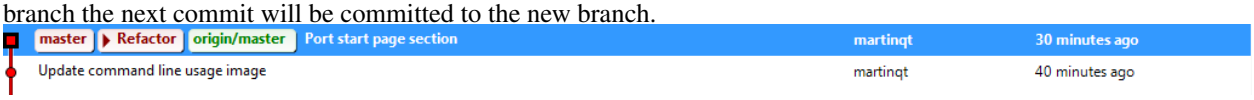
In Git Extensions there are multiple ways to create a new branch. In the image below I create a new branch from the context menu in the commit log. This will create a new branch on the revision that is selected.



I will create a new branch called `Refactor`. In this branch I can do whatever I want without considering others. In the `Create branch` dialog there is a checkbox you can check if you want to checkout this branch immediate after the branch is created.



When the branch is created you will see the new branch `Refactor` in the commit log. If you chose to checkout this branch the next commit will be committed to the new branch.



Creating branches in Git requires only 41 bytes of space in the repository. Creating a new branch is very easy and is very fast. The complete work flow of Git is optimized for branching and merging.

6.1.1 Orphan branches

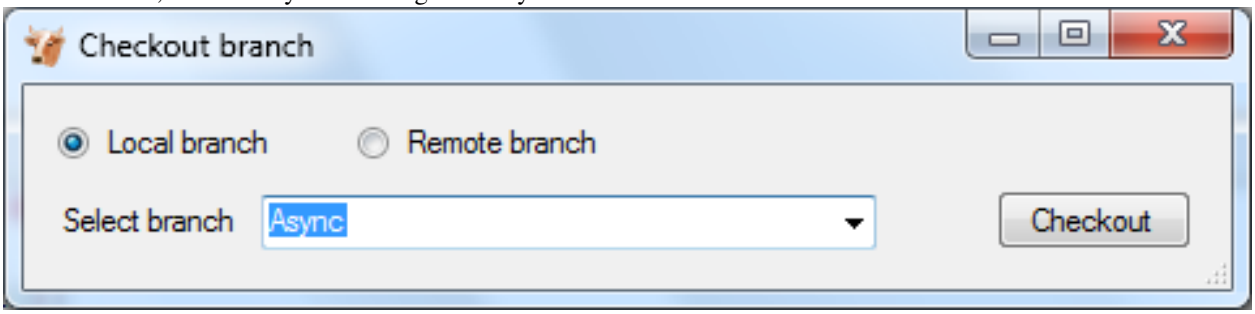
In special cases it is helpful to have orphan branches (see for example https://www.google.com/search?q=why+use+orphan+branches+in+git). Check the "Create orphan" checkbox to create an orphan branch (`--orphan` option in git).

The newly created branch will have no parent commits.

The option "Clear working dir and index" (`git rm -rf`) is active by default. So the working dir and index will be cleared. If you uncheck the last option then the working dir and index will not be touched.

6.2 Checkout branch

You can switch from the current branch to another branch using the checkout command. Checkout a branch sets the current branch and updates all sources in the working directory. Uncommitted changes in the working directory can be overwritten, make sure your working directory is clean.



6.3 Merge branches

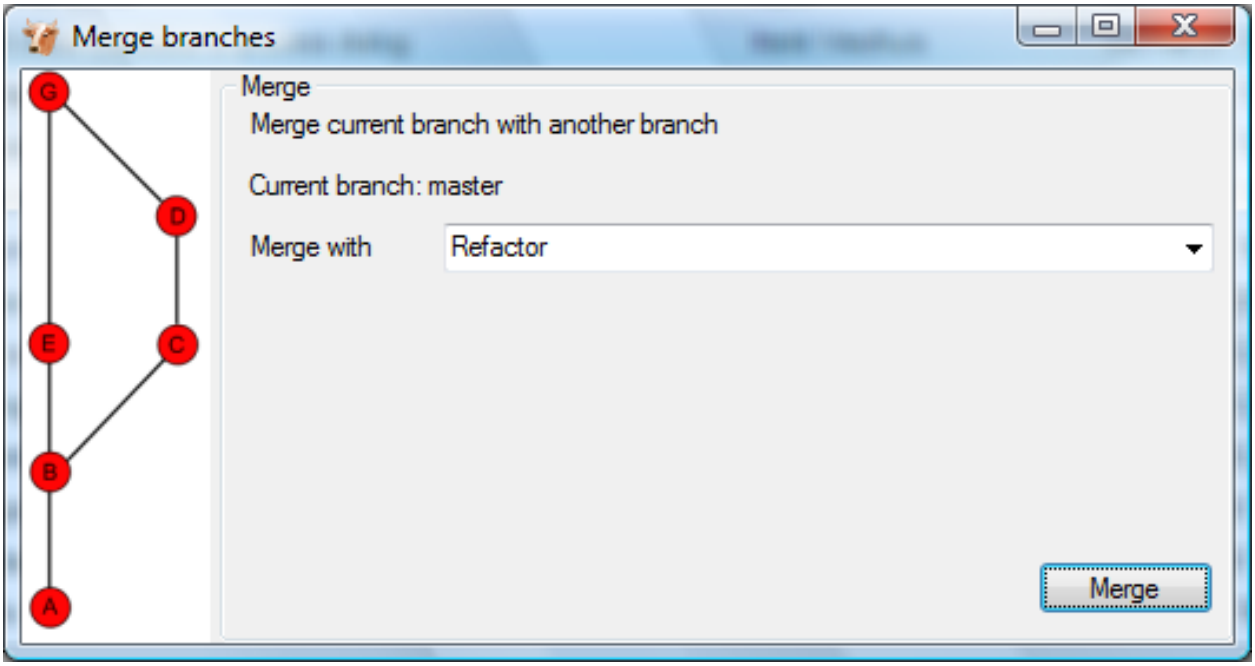
In the image below there are two branches, `[Refactor]` and `[master]`. We can merge the commits from the master branch into the Refactor. If we do this, the Refactor branch will be up to date with the master branch, but not the other way around. As long as we are working on the Refactor branch we cannot tough the master branch itself. We can merge the sources of master into our branch, but cannot make any change to the master branch.

Graph	Message	Author	Date
	[Refactor] Namespace renamed to GitExtensions.*	Henk Westhuis	Sun Feb 22 12:28:12 2009 +0100
	Sources moved to subdir	Henk Westhuis	Sun Feb 22 12:27:54 2009 +0100
	Removed unused projects	Henk Westhuis	Sun Feb 22 12:27:40 2009 +0100
	[master] Added close checkbox to process dialog	Henk Westhuis	Sat Feb 21 13:34:28 2009 +0100
	Added basic image viewer	Henk Westhuis	Sat Feb 21 13:05:05 2009 +0100
	Added image support	Henk Westhuis	Sat Feb 21 12:42:49 2009 +0100
	Added waitcursor	Henk Westhuis	Sat Feb 21 10:47:33 2009 +0100
	Added ShowCommandLine option and added doubleclick to commit dialog	Henk Westhuis	Sat Feb 21 10:34:00 2009 +0100
	Added CoseProcessDialog and added ShowRevisionGraph options	Henk Westhuis	Fri Feb 20 20:05:02 2009 +0100
	Fixed crash on some repos	Henk Westhuis	Thu Feb 19 21:38:07 2009 +0100
	Added changelog	Henk Westhuis	Thu Feb 19 20:01:54 2009 +0100

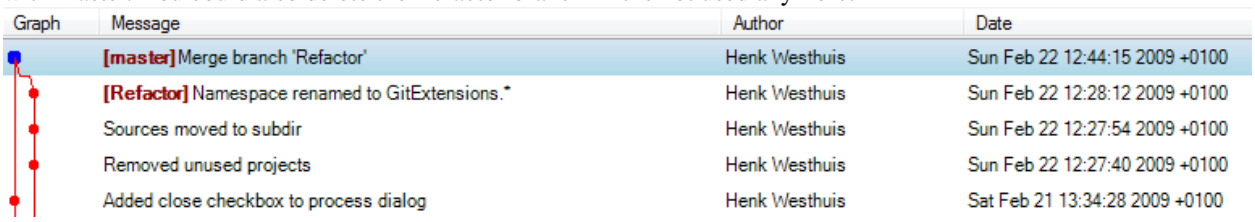
To merge the Refactor branch into the master branch, we need to switch to the master branch first.

Graph	Message	Author	Date
	[Refactor] Namespace renamed to GitExtensions.*	Henk	Sun Feb 22 12:28:12 2009 +0100
	Sources moved to subdir	Henk	Sun Feb 22 12:27:54 2009 +0100
	Removed unused projects	Henk	Sun Feb 22 12:27:40 2009 +0100
	[master] Added close checkbox to process dialog	Henk Westhuis	Sat Feb 21 13:34:28 2009 +0100
	Added basic image viewer	Henk Westhuis	Sat Feb 21 13:05:05 2009 +0100
	Added image support	Henk Westhuis	Sat Feb 21 12:42:49 2009 +0100
	Added waitcursor	Henk Westhuis	Sat Feb 21 10:47:33 2009 +0100
	Added ShowCommandLine option and added doubleclick to commit dialog	Henk Westhuis	Sat Feb 21 10:34:00 2009 +0100
	Added CoseProcessDialog and added ShowRevisionGraph options	Henk Westhuis	Fri Feb 20 20:05:02 2009 +0100
	Fixed crash on some repos	Henk Westhuis	Thu Feb 19 21:38:07 2009 +0100
	Added changelog	Henk Westhuis	Thu Feb 19 20:01:54 2009 +0100

Once we are on the master branch we can choose merge by choosing `Merge branches` from the `Commands` menu. In the merge dialog you can check the branch you are working on. After selected the branch to merge with, click the `Merge` button.



After the merge the commit log will show the new commit containing the merge. Notice that the Refactor branch is not changed by this merge. If you want to continue working on the Refactor branch you can merge the Refactor branch with master. You could also delete the Refactor branch if it is not used anymore.



Graph	Message	Author	Date
	[master] Merge branch 'Refactor'	Henk Westhuis	Sun Feb 22 12:44:15 2009 +0100
	[Refactor] Namespace renamed to GitExtensions.*	Henk Westhuis	Sun Feb 22 12:28:12 2009 +0100
	Sources moved to subdir	Henk Westhuis	Sun Feb 22 12:27:54 2009 +0100
	Removed unused projects	Henk Westhuis	Sun Feb 22 12:27:40 2009 +0100
	Added close checkbox to process dialog	Henk Westhuis	Sat Feb 21 13:34:28 2009 +0100

Note: When you need to merge with on unnamed branch you can use a tag to give it a temporary name.

6.4 Rebase branch

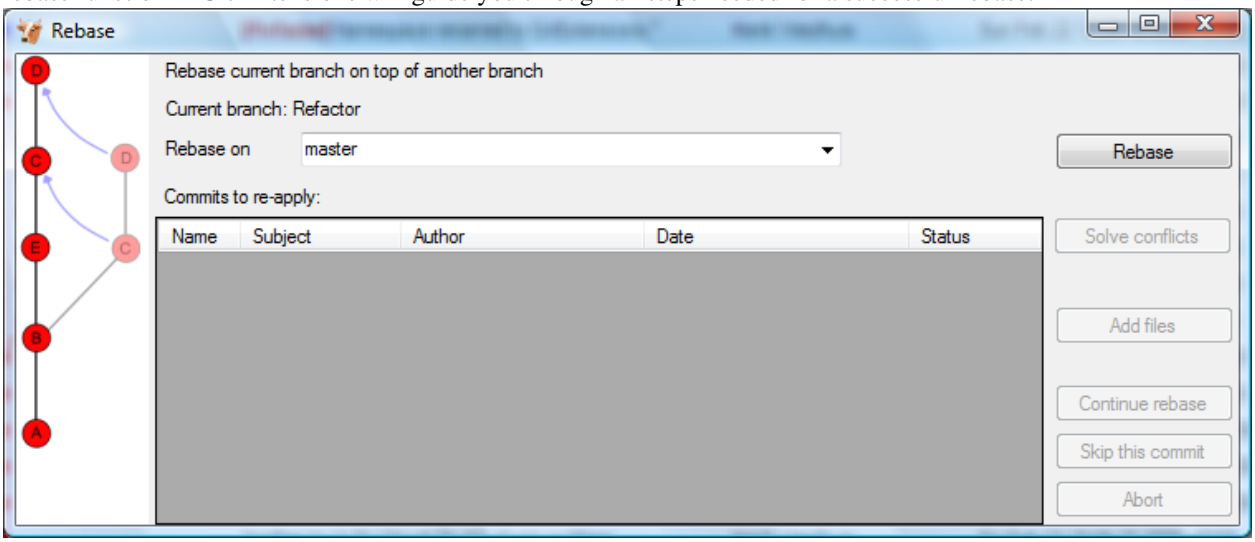
The rebase command is the most complex command in Git. The rebase command is very similar to the merge command. Both rebase and merge are used to get a branch up-to-date. The main difference is that rebase can be used to keep the history linear contrary to merges.

Graph	Message	Author	Date
	[Refactor] Namespace renamed to GitExtensions.*	Henk Westhuis	Sun Feb 22 12:28:12 2009 +0100
	Sources moved to subdir	Henk Westhuis	Sun Feb 22 12:27:54 2009 +0100
	Removed unused projects	Henk Westhuis	Sun Feb 22 12:27:40 2009 +0100
	[master] Added close checkbox to process dialog	Henk Westhuis	Sat Feb 21 13:34:28 2009 +0100
	Added basic image viewer	Henk Westhuis	Sat Feb 21 13:05:05 2009 +0100
	Added image support	Henk Westhuis	Sat Feb 21 12:42:49 2009 +0100
	Added waitcursor	Henk Westhuis	Sat Feb 21 10:47:33 2009 +0100
	Added ShowCommandLine option and added doubleclick to commit dialog	Henk Westhuis	Sat Feb 21 10:34:00 2009 +0100
	Added CoseProcessDialog and added ShowRevisionGraph options	Henk Westhuis	Fri Feb 20 20:05:02 2009 +0100
	Fixed crash on some repos	Henk Westhuis	Thu Feb 19 21:38:07 2009 +0100
	Added changelog	Henk Westhuis	Thu Feb 19 20:01:54 2009 +0100

A rebase of Refactor on top of master will perform the following actions:

- All commits specific to the Refactor branch will be stashed in a temporary location
- The branch Refactor will be removed
- The branch Refactor will be recreated on the master branch
- All commits will be recommitted in the new Refactor branch

During a rebase merge conflicts can occur. You need to solve the merge conflicts for each commit that is rebased. The rebase function in Git Extensions will guide you through all steps needed for a successful rebase.



The image below shows the commit log after the rebase. Notice that the history is changed and is seems like the commits on the Refactor branch are created after the commits on the master branch.

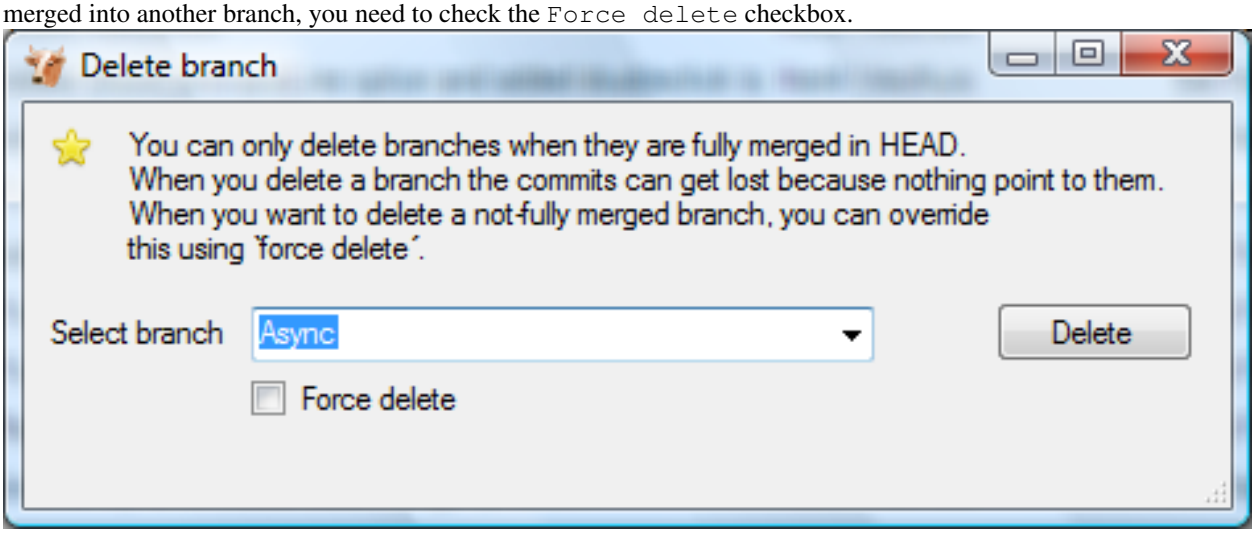
Graph	Message	Author	Date
	[Refactor] Namespace renamed to GitExtensions.*	Henk Westhuis	Sun Feb 22 13:21:26 2009 +0100
	Sources moved to subdir	Henk Westhuis	Sun Feb 22 12:27:54 2009 +0100
	Removed unused projects	Henk Westhuis	Sun Feb 22 12:27:40 2009 +0100
	[master] Added close checkbox to process dialog	Henk Westhuis	Sat Feb 21 13:34:28 2009 +0100
	Added basic image viewer	Henk Westhuis	Sat Feb 21 13:05:05 2009 +0100
	Added image support	Henk Westhuis	Sat Feb 21 12:42:49 2009 +0100
	Added waitcursor	Henk Westhuis	Sat Feb 21 10:47:33 2009 +0100
	Added ShowCommandLine option and added doubleclick to commit dialog	Henk Westhuis	Sat Feb 21 10:34:00 2009 +0100
	Added CoseProcessDialog and added ShowRevisionGraph options	Henk Westhuis	Fri Feb 20 20:05:02 2009 +0100
	Fixed crash on some repos	Henk Westhuis	Thu Feb 19 21:38:07 2009 +0100
	Added changelog	Henk Westhuis	Thu Feb 19 20:01:54 2009 +0100

Warning: Because this function rewrites history you should only use this on branches that are not published to other repositories yet. When you rebase a branch that is already pushed it will be harder to pull or push to that remote. If you want to get a branch up-to-date that is already published you should merge.

6.5 Delete branch

It is very common to create a lot of branches. You can delete branches when they are not needed anymore and you do not want to keep the work done in that branch. When you delete a branch that is not yet merged, all commits will be lost. When you delete a branch that is already merged with another branch, the merged commits will not be lost because they are also part of another branch.

You can delete a branch using `Delete branch` in `Commands` menu. If you want to delete a branch that is not merged into another branch, you need to check the `Force delete` checkbox.



CHAPTER

SEVEN

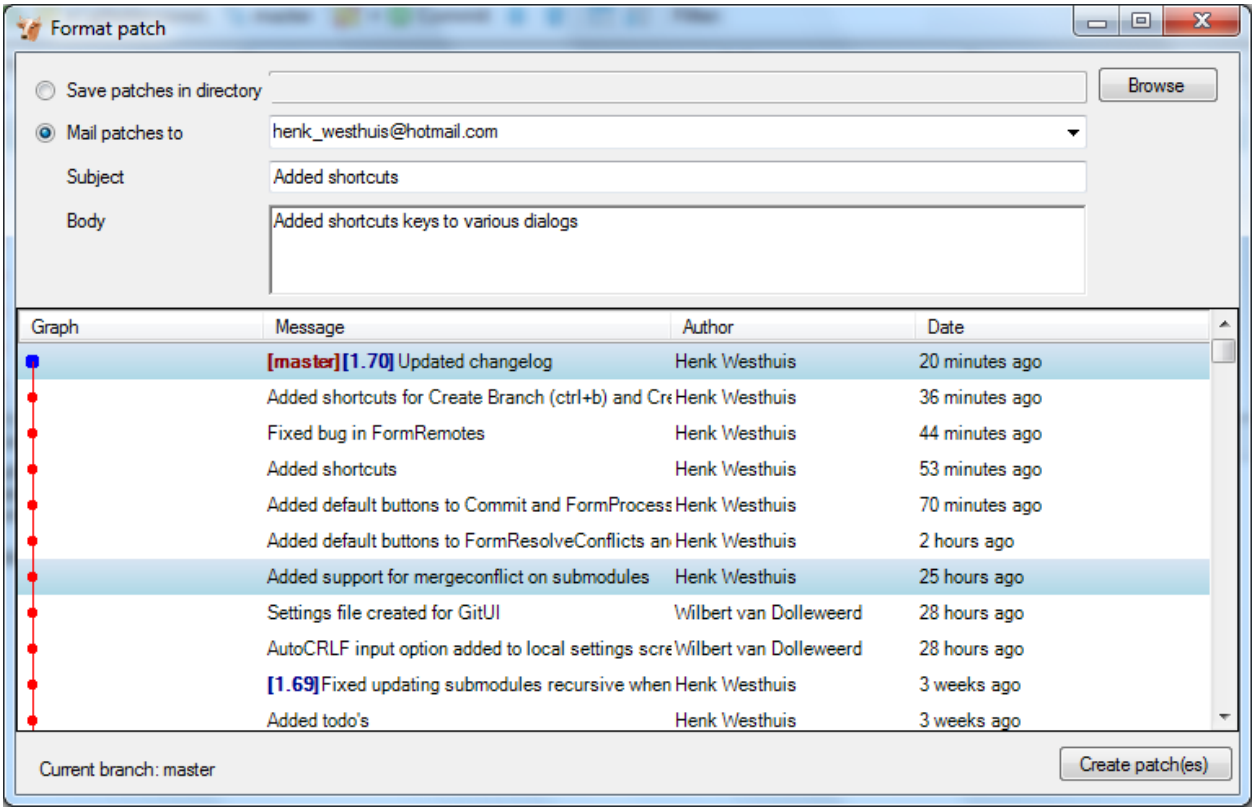
PATCHES

Every commit contains a change-set, a commit date, the committer name, the commit message and a cryptograph SHA1 hash. Local commits can be published by pushing it to a remote repository. To be able to push you need to have sufficient rights and you need to have access to the remote repository. When you cannot push directly you can create patches. Patches can be e-mailed to someone with access to the repository. Each patch contains an entire commit including the commit message and the SHA1.

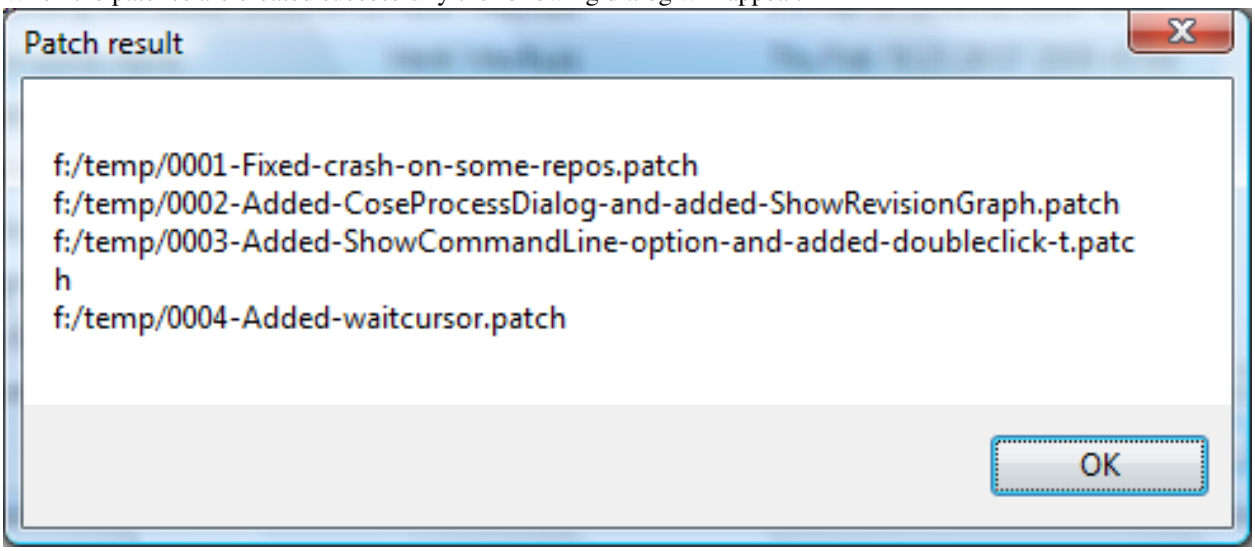
```
From 58c02ec4701c94c671a41e1e5d50c582e859851f Mon Sep 17 00:00:00 2001
From: Russell King <rmk@dyn-67.arm.linux.org.uk>
Date: Sun, 17 Apr 2005 15:40:46 +0100
Subject: [PATCH 000213/123824] [PATCH] ARM: h3600_irda_set_speed arguments

h3600_irda_set_speed() had the wrong type for the "speed" argument.
Fix this.

Signed-off-by: Russell King <rmk@arm.linux.org.uk>
---
 arch/arm/mach-sa1100/h3600.c |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)

diff --git a/arch/arm/mach-sa1100/h3600.c b/arch/arm/mach-sa1100/h3600.c
index 9788d3a..84c8654 100644
--- a/arch/arm/mach-sa1100/h3600.c
+++ b/arch/arm/mach-sa1100/h3600.c
@@ -130,7 +130,7 @@ static int h3600_irda_set_power(struct device *dev, unsigned int state)
    return 0;
 }

-static void h3600_irda_set_speed(struct device *dev, int speed)
+static void h3600_irda_set_speed(struct device *dev, unsigned int speed)
 {
    if (speed < 4000000) {
        clr_h3600_egpio(IPAQ_EGPIO_IR_FSEL);
--
1.6.1.9.g97c34
```

7.1 Create patch

Format a single patch or patch series using the format patch dialog. You need to select the newest commit first and then select the oldest commit using ctrl-click. You can also select an interrupted patch series, but this is not recommended because the files will not be numbered.

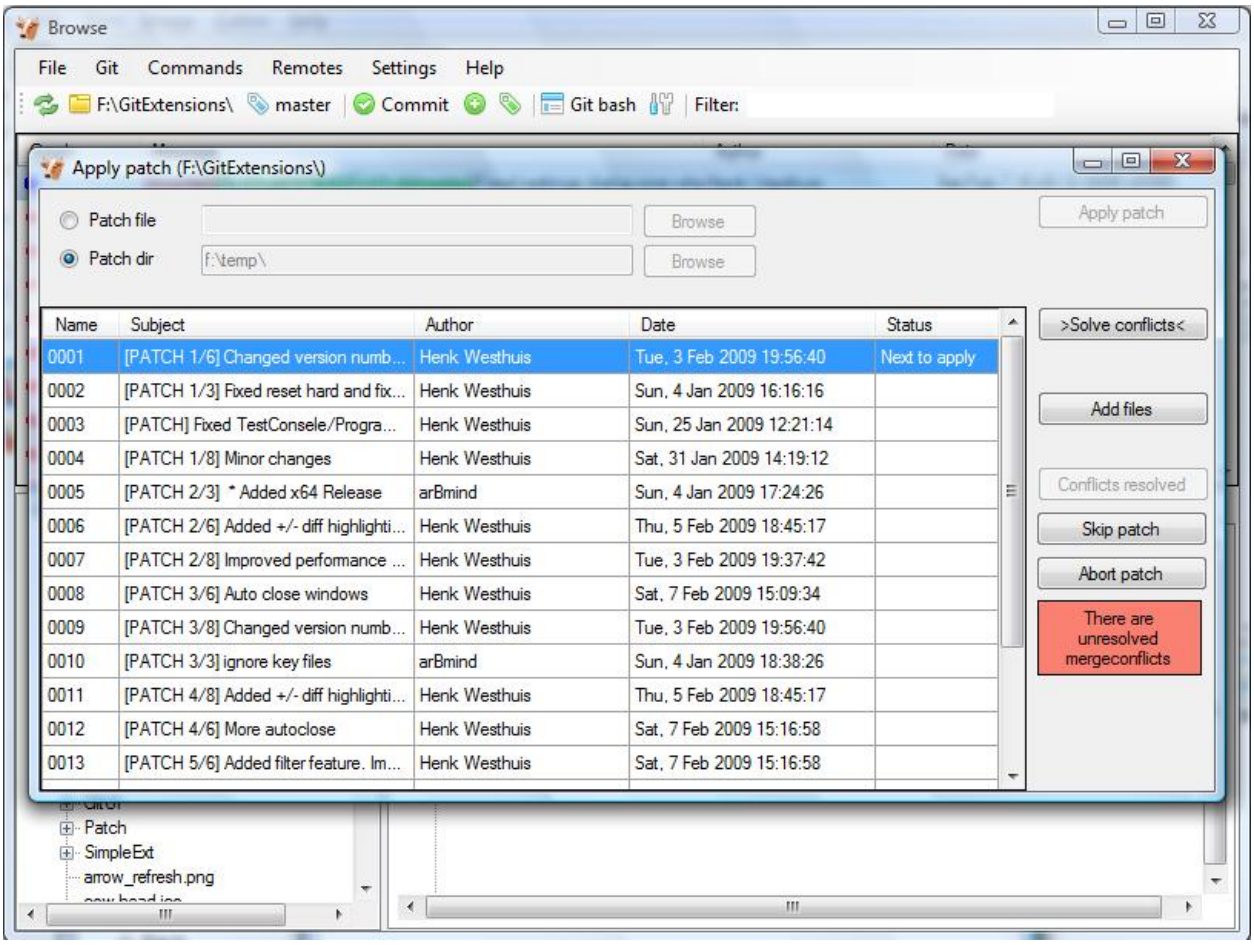


When the patches are created successfully the following dialog will appear.



7.2 Apply patches

It is possible to apply a single patch file or all patches in a directory. When there are merge conflicts applying the patch you need to resolve them before you can continue. Git Extensions will help you applying all patches by marking the next recommended step.


Browse
File Git Commands Remotes Settings Help
F:\GitExtensions\ master Commit Git bash Filter:
Apply patch (F:\GitExtensions\)
Patch file
Patch dir
f:\temp\
Browse
Browse
Apply patch
Name Subject Author Date Status
0001 [PATCH 1/6] Changed version numb... Henk Westhuis Tue, 3 Feb 2009 19:56:40 Next to apply
0002 [PATCH 1/3] Fixed reset hard and fix... Henk Westhuis Sun, 4 Jan 2009 16:16:16
0003 [PATCH] Fixed TestConsele/Progra... Henk Westhuis Sun, 25 Jan 2009 12:21:14
0004 [PATCH 1/8] Minor changes Henk Westhuis Sat, 31 Jan 2009 14:19:12
0005 [PATCH 2/3] * Added x64 Release arBmind Sun, 4 Jan 2009 17:24:26
0006 [PATCH 2/6] Added +/- diff highlighti... Henk Westhuis Thu, 5 Feb 2009 18:45:17
0007 [PATCH 2/8] Improved performance ... Henk Westhuis Tue, 3 Feb 2009 19:37:42
0008 [PATCH 3/6] Auto close windows Henk Westhuis Sat, 7 Feb 2009 15:09:34
0009 [PATCH 3/8] Changed version numb... Henk Westhuis Tue, 3 Feb 2009 19:56:40
0010 [PATCH 3/3] ignore key files arBmind Sun, 4 Jan 2009 18:38:26
0011 [PATCH 4/8] Added +/- diff highlighti... Henk Westhuis Thu, 5 Feb 2009 18:45:17
0012 [PATCH 4/6] More autoclose Henk Westhuis Sat, 7 Feb 2009 15:16:58
0013 [PATCH 5/6] Added filter feature. Im... Henk Westhuis Sat, 7 Feb 2009 15:16:58
>Solve conflicts<
Add files
Conflicts resolved
Skip patch
Abort patch
There are unresolved mergeconflicts
Patch
SimpleExt
arrow_refresh.png

CHAPTER

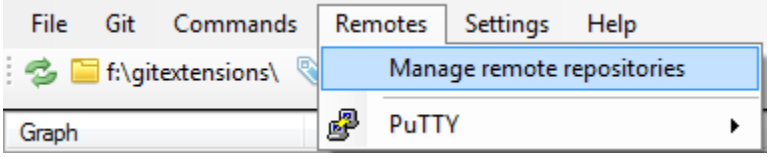
EIGHT

REMOTE FEATURE

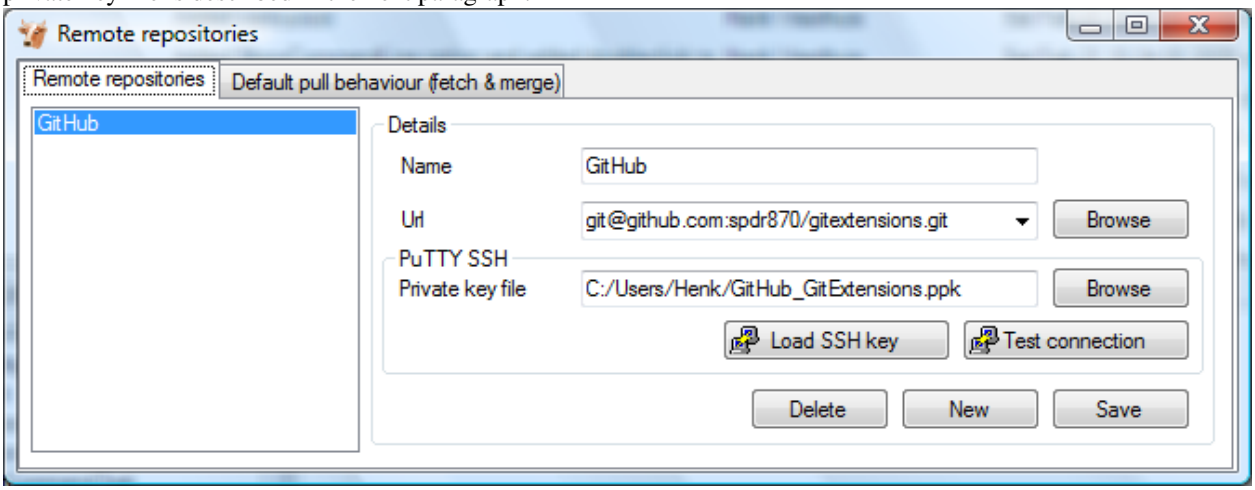
Git is a distributed source control management system. This means that all changes you make are local. When you commit changes, you only commit them to your local repository. To publish your local changes you need to push. In order to get changes committed by others, you need to pull.

8.1 Manage remote repositories

You can manage the remote repositories in the `Remotes` menu.

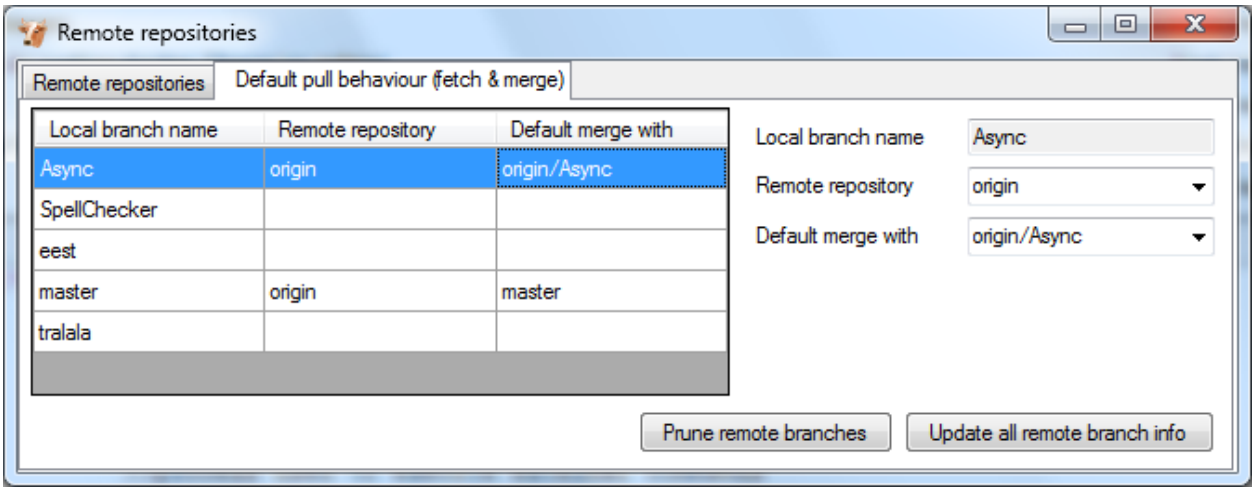


When you cloned your repository from a public repository, this remote is already configured. You can rename each remote for easy recognition. The default name after cloning a remote is `origin`. If you use PuTTY as SSH client you can also enter the private key file for each remote. Git Extensions will load the key when needed. How to create a private key file is described in the next paragraph.



In the `Default pull behaviour` tab you can configure the branches that need to be pulled and merged by default. If you configure this correctly you will not need to choose a branch when you pull or push. There are two buttons on this dialog:

Prune remote branches	Throw away remote branches that do not exist on the remote anymore.
Update all remote branch info	Fetch all remote branch information.



After cloning a repository you do not need to configure all remote branches manually. Instead you can checkout the remote branch and choose to create a local tracking branch.

8.2 Create SSH key

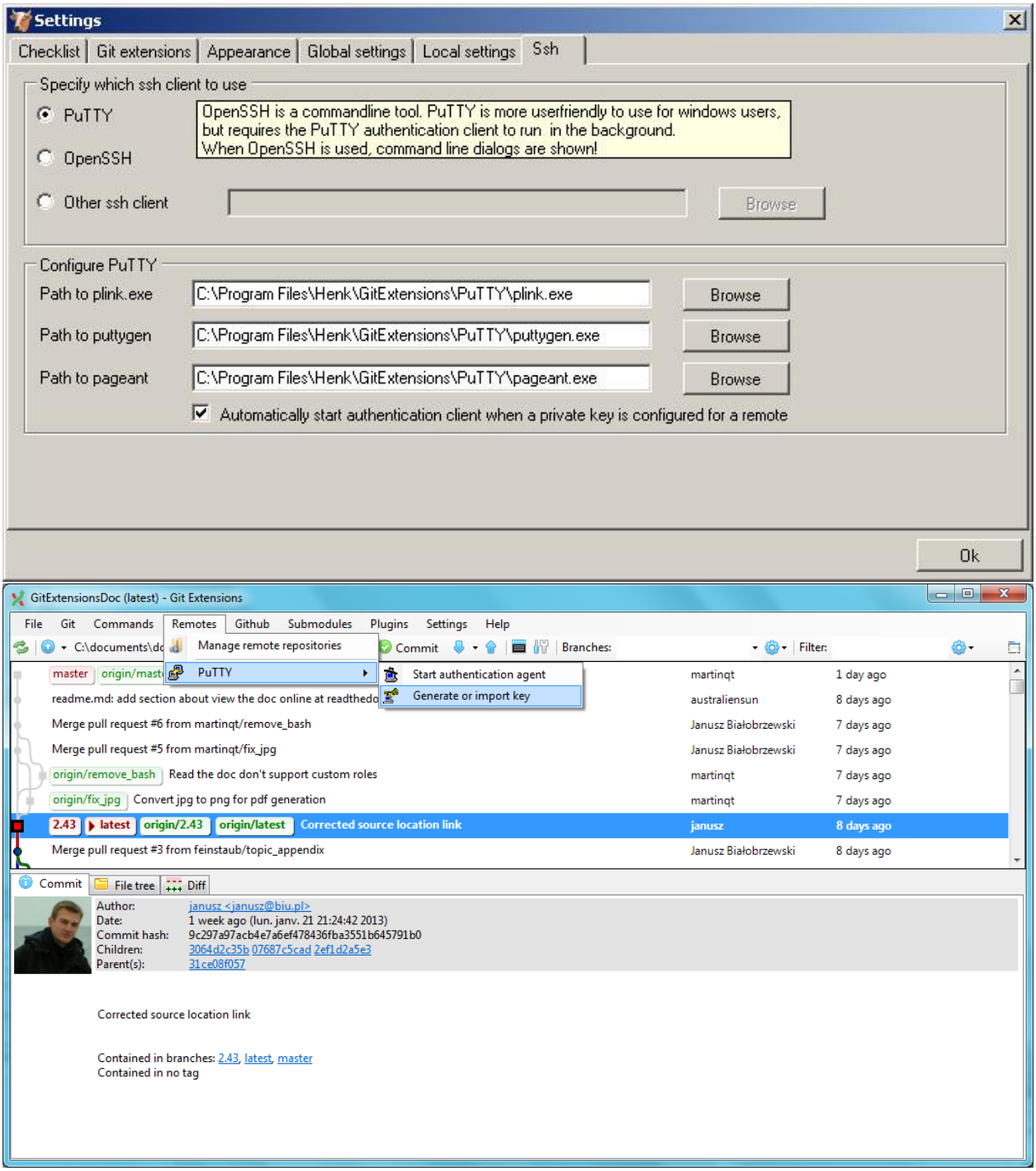
Git uses SSH for accessing private repositories. SSH uses a public/private key pair for authentication. This means you need to generate a private key and a public key. The private key is stored on your computer locally and the public key can be given to anyone. SSH will encrypt whatever you send using your secret private key. The receiver will then use the public key you send to decrypt the data.

This encryption will not protect the data itself but it protects the authenticity. Because the private key is only available to the sender, the receiver can be sure about the origin of the data. In practise the key pair is only used for the authentication process. The data itself will be encrypted using a key that is exchanged during this initial phase.

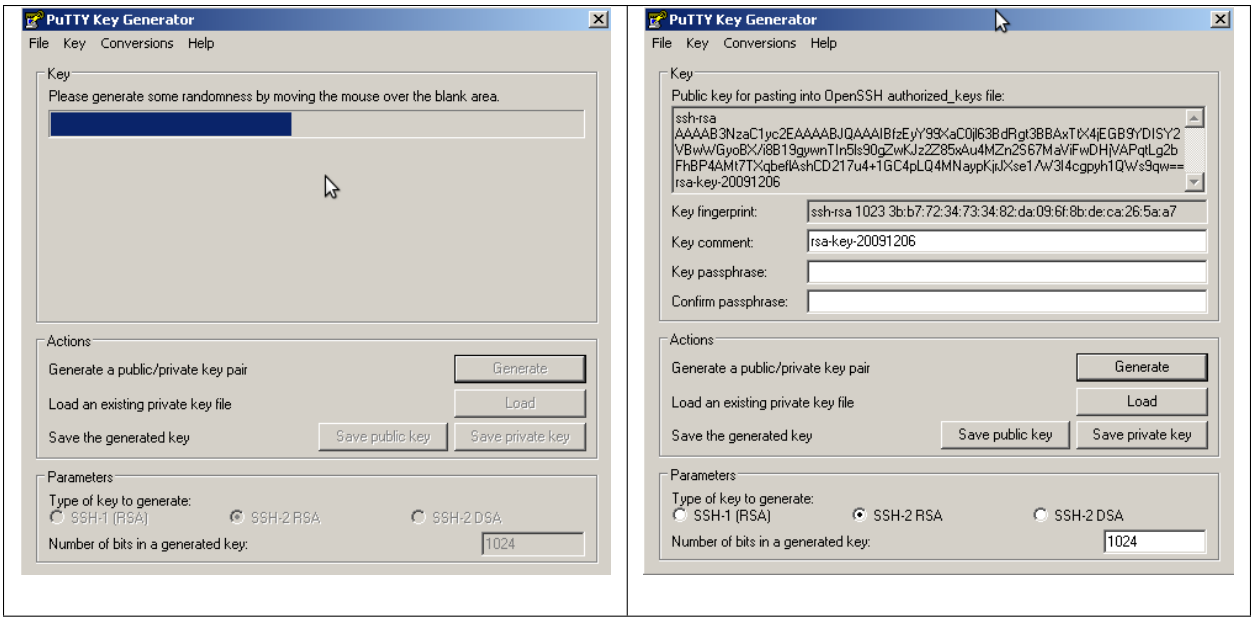
8.2.1 PuTTY and github

PuTTY is SSH client that for Windows that is a bit more user friendly then OpenSSH. Unfortunately PuTTY does not work with all servers. In this paragraph I will show how to generate a key for github using putty.

First make sure GitExtensions is configured to use PuTTY and all paths are correct.

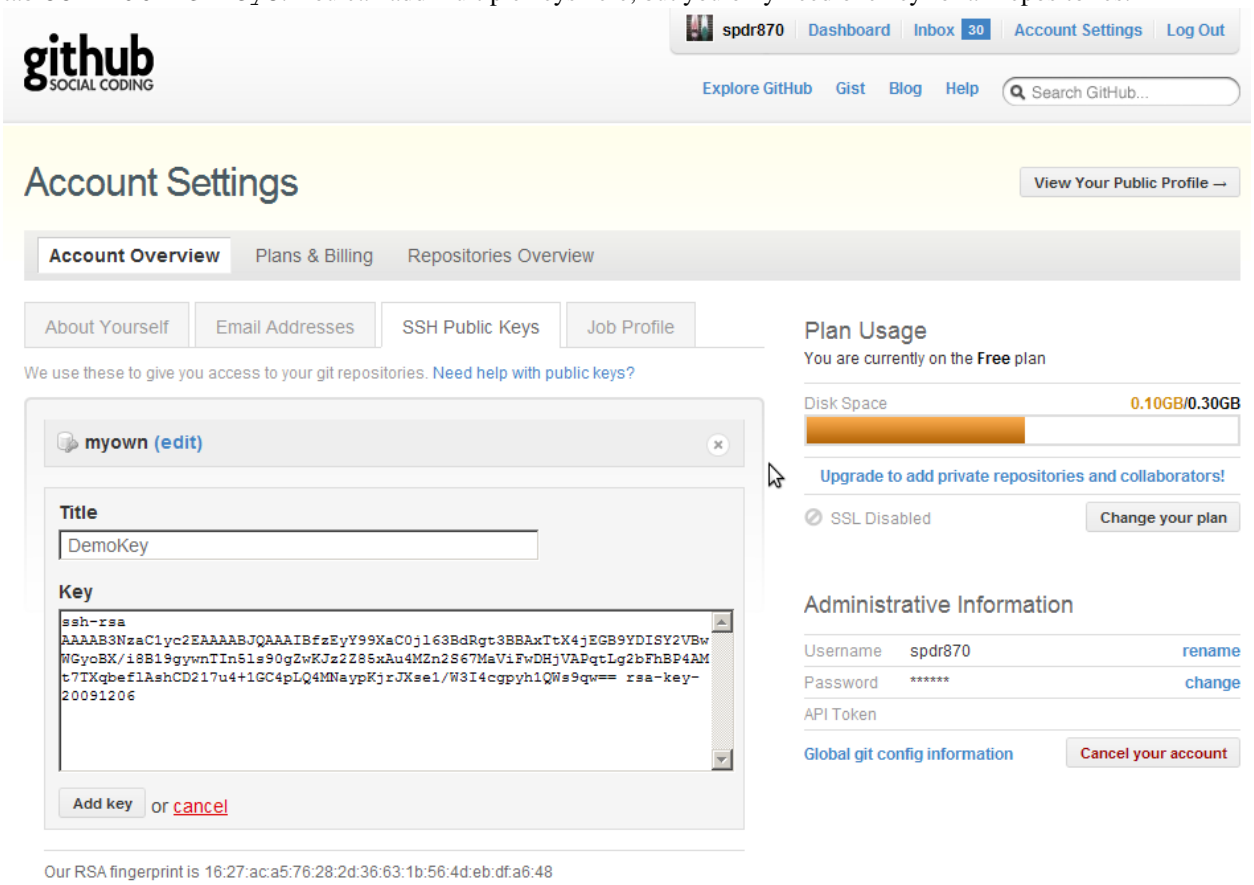


can choose `Generate or import key` to start the key generator.

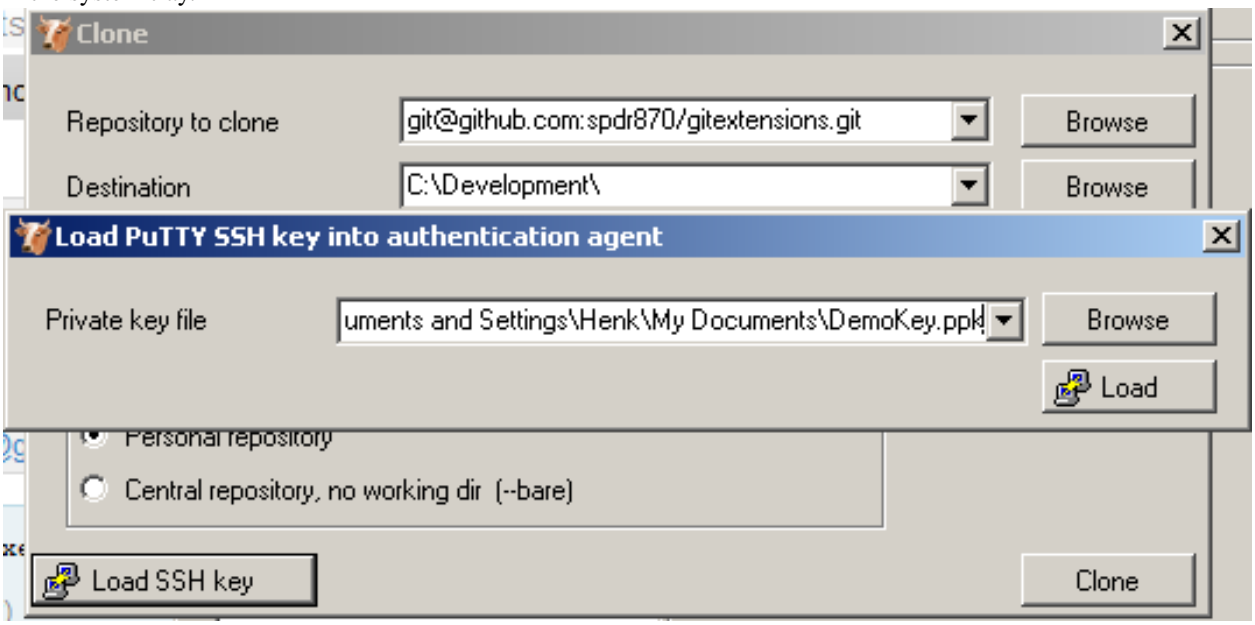


PuTTY will ask you to move the mouse around to generate a more random key. When the key is generated you can save the public and the private key in a file. You can choose to protect the private key with a password but this is not necessary.

Now you have a key pair you need to give github the public key. This can be done in `Account Settings` in the tab `SSH Public Keys`. You can add multiple keys here, but you only need one key for all repositories.



After telling github what public key to use to decrypt, you need to tell GitExtensions what private key to use to encrypt. In the clone dialog there is a `Load SSH key` button to load the private key into the PuTTY authentication agent. This can also be done manually by starting the PuTTY authentication agent and choose `add key` in the context menu in the system tray.

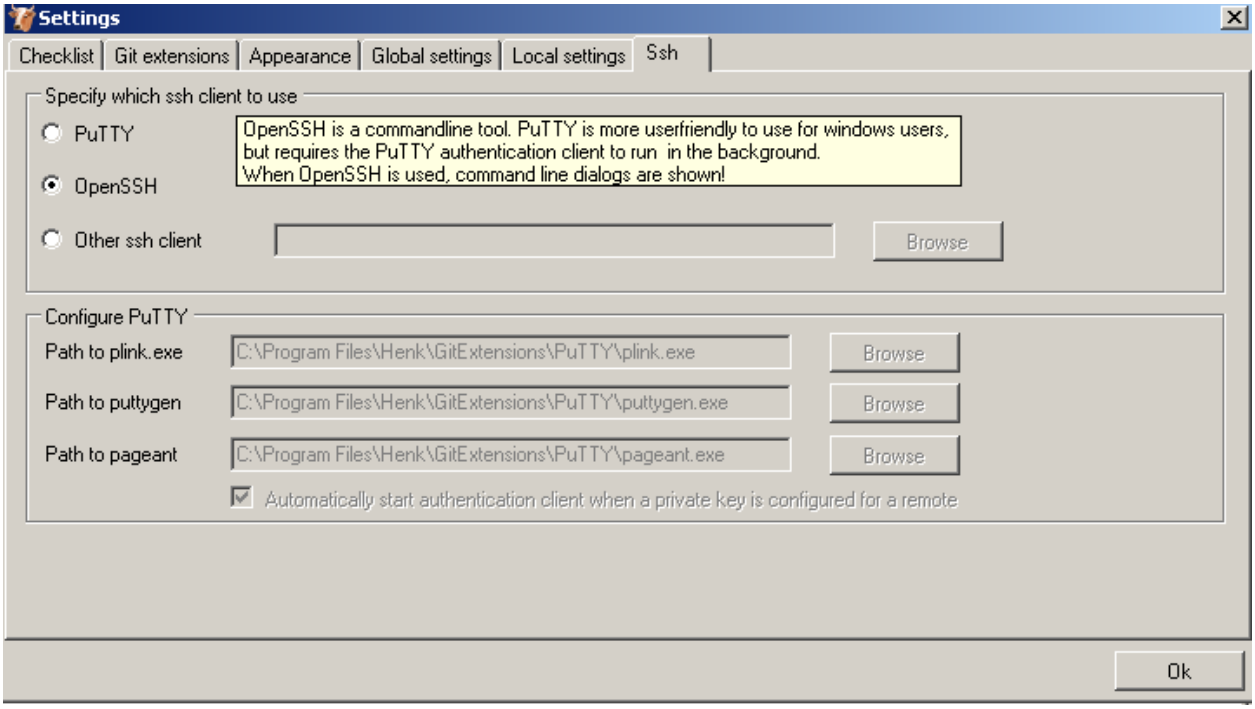


GitExtensions can load the private keys automatically for you when communicating with a remote. You need to configure the private key for the remote.

This is done in the `Manage remote repositories` dialog.

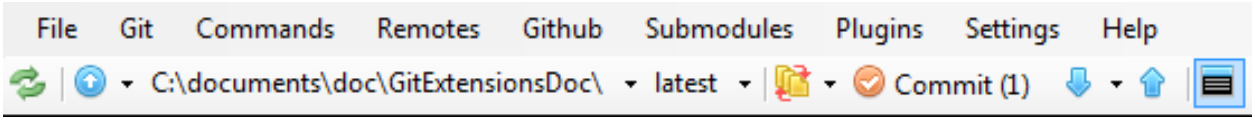
8.2.2 OpenSSH and github

When you choose to use OpenSSH you need to configure GitExtensions as shown in the screenshot below.

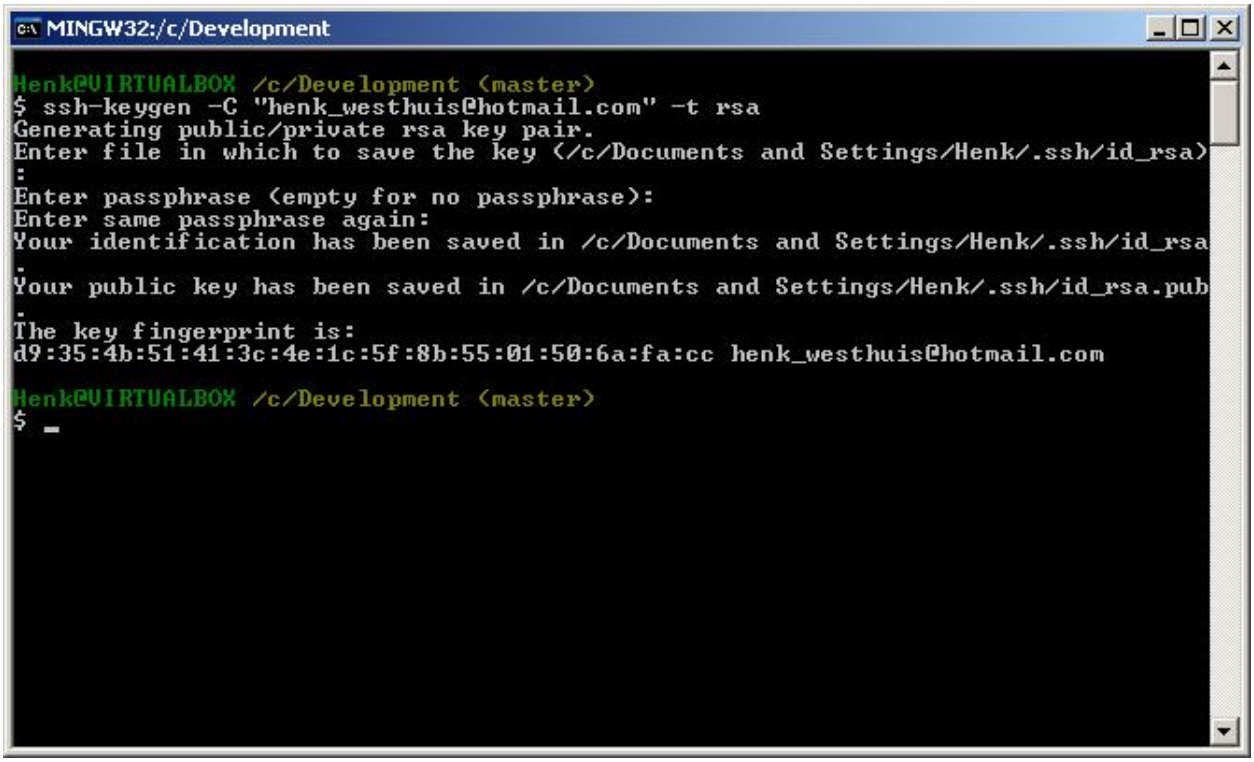


OpenSSH is the best SSH client there is but it lacks Windows support. Therefore it is slightly more complex to use. Another drawback is that GitExtensions cannot control OpenSSH and needs to show the command line dialogs when OpenSSH might be used. GitExtensions will show the command line window for every command that might require a SSH connection. For this reason PuTTY is the prefered SSH client in GitExtensions.

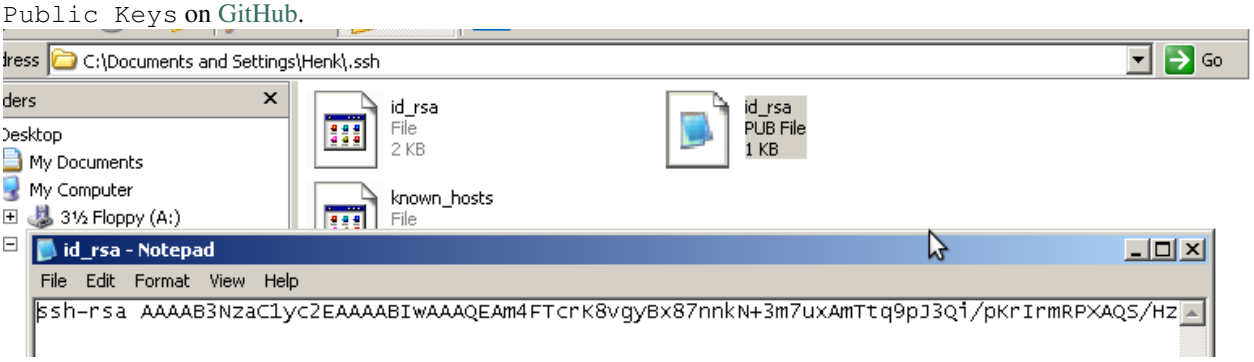
To generate a key pair in OpenSSH you need to go to the command line. I recommend to use the git bash because the path to OpenSSH is already set.



Type the following command: `ssh-keygen -C "your@email.com" -t rsa` Use the same email address as the email address used in git. You will be asked where if you want to protect the private key with a password. This is not necessary. By default the public and private keys are stored in `c:\Documents and Settings\[User]\.ssh\` or `c:\Users\[user]\.ssh\`.

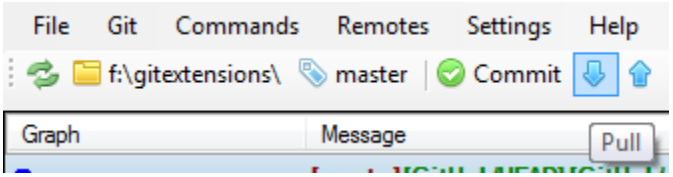


You do not need to tell GitExtensions about the private key because OpenSSH will load it for you. Now open the public key using notepad and copy the key to github. This can be done in `Account Settings` in the tab `SSH Public Keys` on GitHub.



8.3 Pull changes

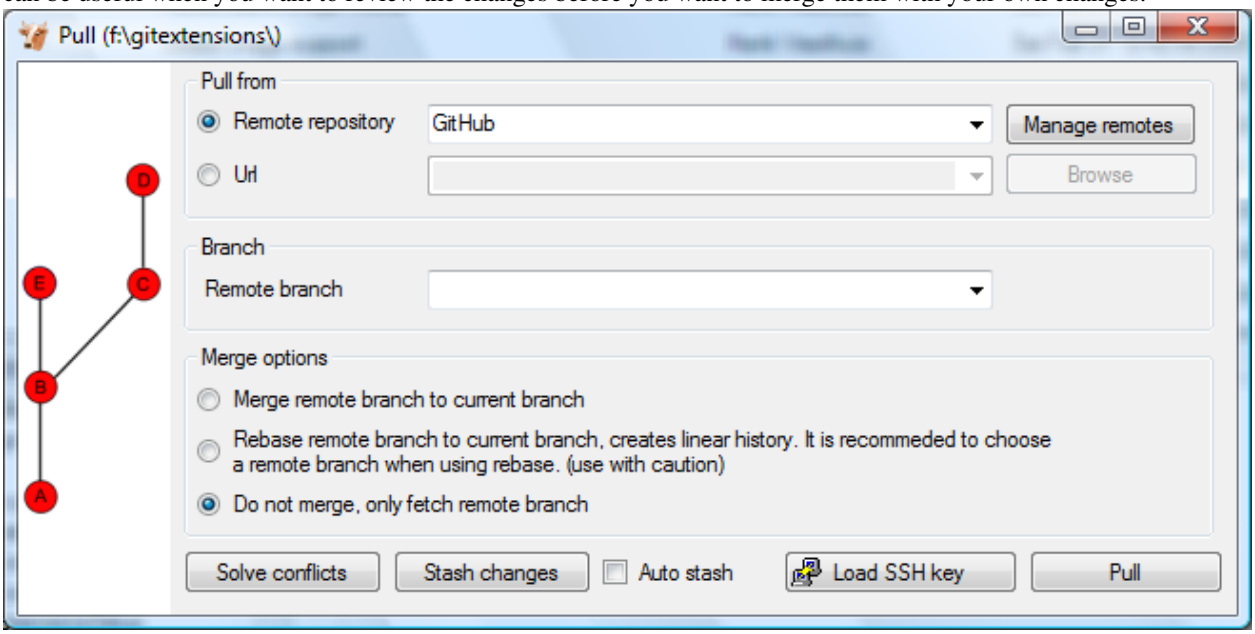
You can get remote changes using the pull function. Before you can pull remote changes you need to make sure there are no uncommitted changes in your local repository. If you have uncommitted changes you should commit them or stash them during the pull. You can read about how to use the stash in the Stash chapter.



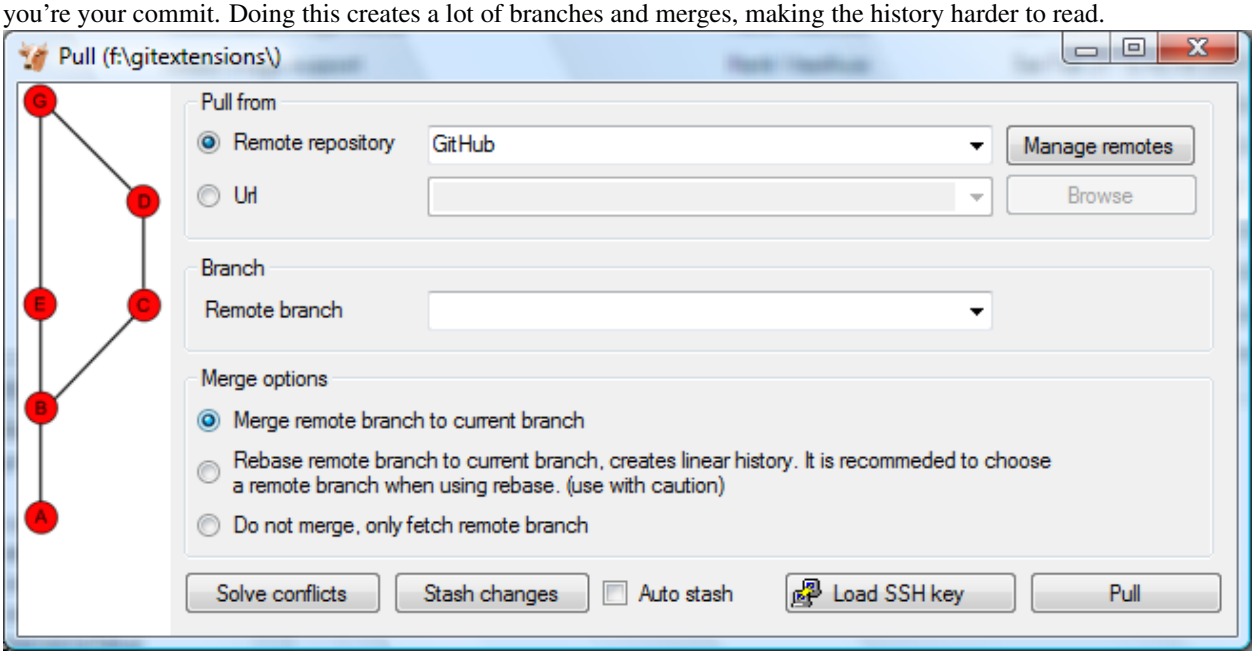
In order to get your personal repository up-to-date, you need to fetch changes from a remote repository. You can do this using the `Pull` dialog. When the dialog starts the default remote for the current branch is set. You can choose

another remote or enter a custom url if you like. When the remote branches configured correctly, you do not need to choose a remote branch.

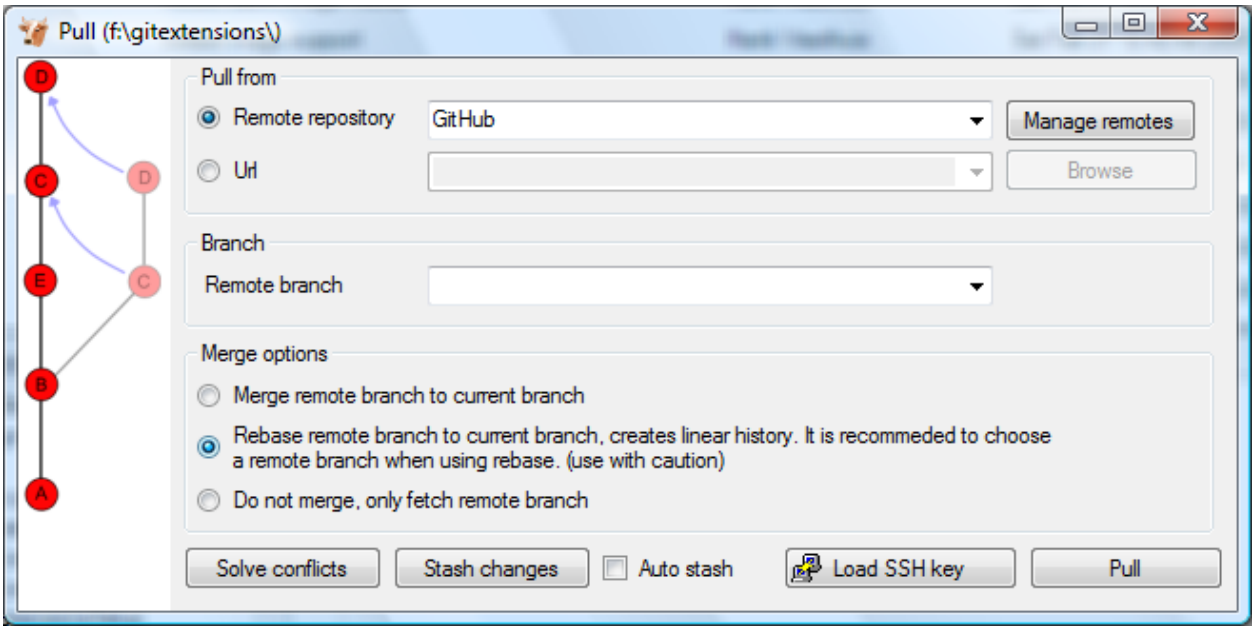
If you just fetch the commits from the remote repository and you already committed some changes to your local repository, the commits will be in a different branch. In the pull dialog this is illustrated in the image on the left. This can be useful when you want to review the changes before you want to merge them with your own changes.



When you choose to merge the remote branch after fetching the changes a branch will be created, and will be merged you're your commit. Doing this creates a lot of branches and merges, making the history harder to read.



Instead of merging the fetched commits with your local commits, you can also choose to rebase your commits on top of the fetched commits. This is illustrated on the left in the image below. A rebase will first undo your local commits (c and d), then fetch the remote commits (e) and finally recommit your local commits. When there is a merge conflict during the rebase, the rebase dialog will show.

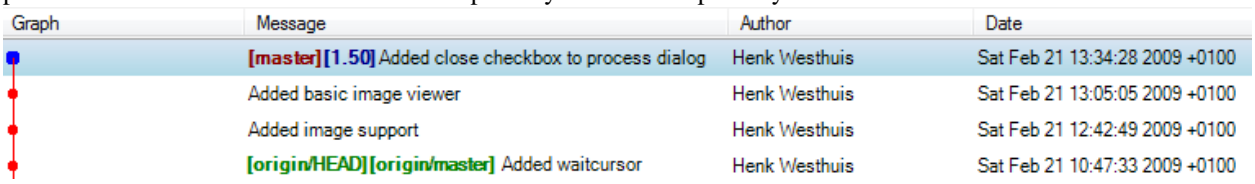


Next to the pull button there are some buttons that can be useful:

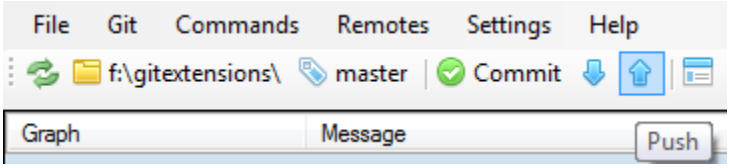
Solve conflicts	When there are merge conflicts, you can solve them by pressing this button.
Stash changes	When the working dir contains uncommitted changes, you need to stash them before pulling.
Auto stash	Check this checkbox if you want to stash before pulling. The stash will be reapplied after pulling.
Load SSH key	This button is only available when you use PuTTY as SSH client. You can press this button to load the key configured for the remote. If no key is set, a dialog will prompt for the key.

8.4 Push changes

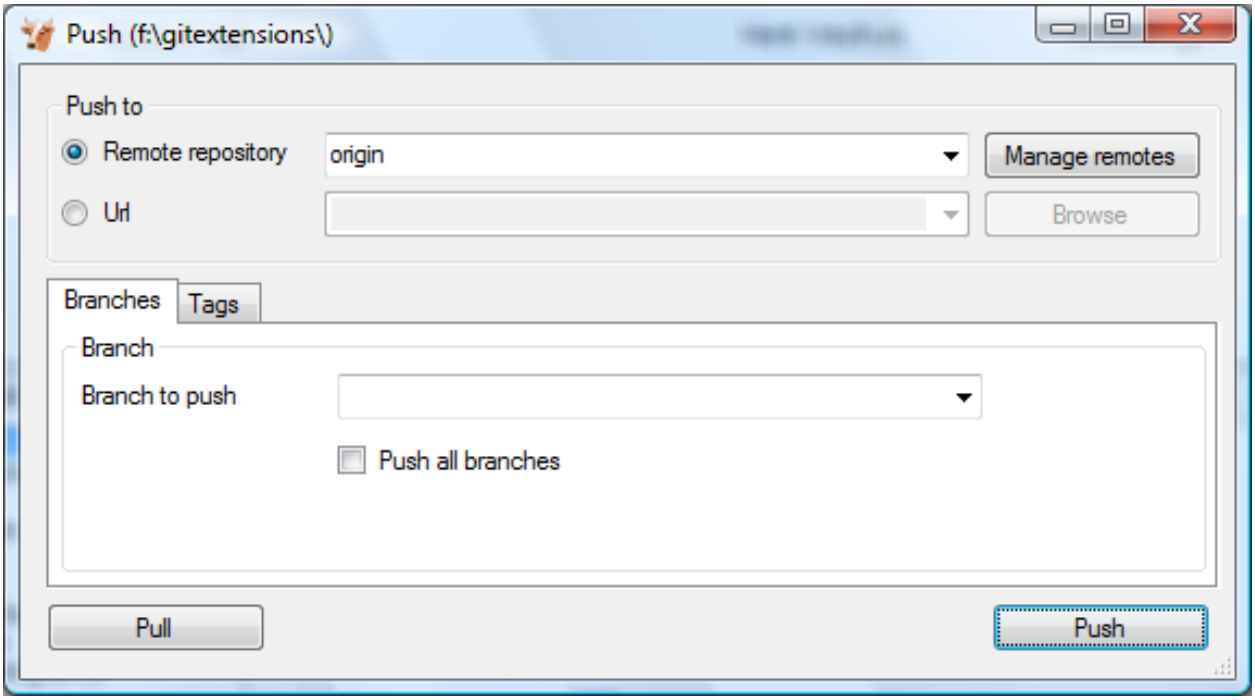
In the browse window you can check if there are local commits that are not pushed to a remote repository yet. In the image below the green labels mark the position of the master branch on the remote repository. The red label marks the position of the master branch on the local repository. The local repository is ahead three commits.



To push the changes press `Push` in the toolbar.



The push dialog allows you to choose the remote repository to push to. The remote repository is set to the remote of the current branch. You can choose another remote or choose a url to push to. You can also specify a branch to push.



Tags are not pushed to the remote repository. If you want to push a tag you need to open the `Tags` tab in the dialog. You can choose to push a singe tag or all tags. No commits will be pushed when the `Tags` tab is selected, only tags.

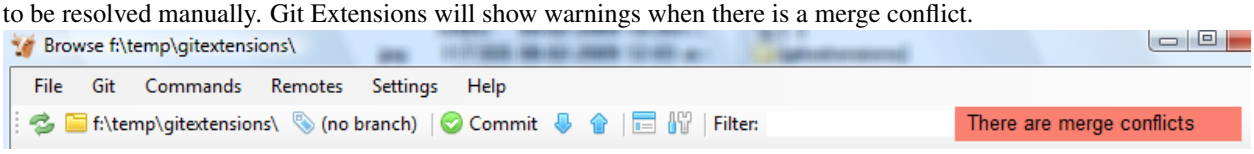
You can not merge your changes in the remote repository. Merging must be done locally. This means that you cannot push your changes before the commits are merged locally. In practice you need to pull before you can push most of the times.

CHAPTER

NINE

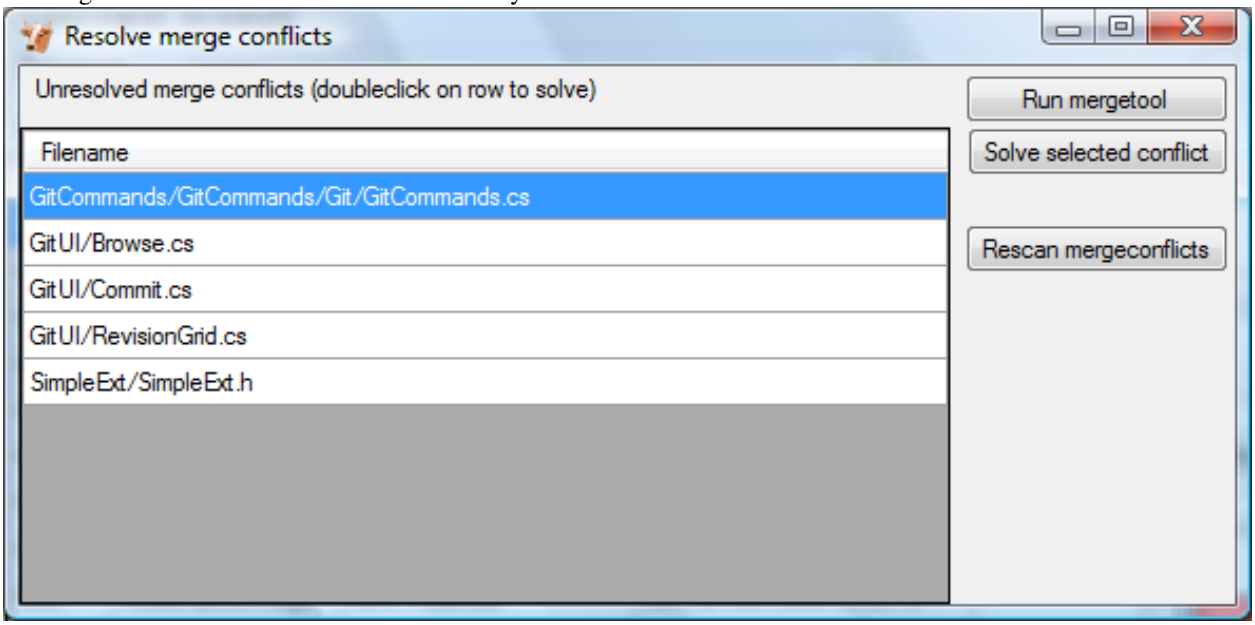
MERGE CONFLICTS

When merging branches or commits you can get merge conflicts. Git will try to resolve these, but some conflicts need to be resolved manually. Git Extensions will show warnings when there is a merge conflict.



9.1 Handle merge conflicts

To solve merge conflicts just click on a warning or open the merge conflict dialog from the menu. A dialog will prompt showing all conflicts. You can solve a conflict by double-click on a filename.



There are three kinds of conflicts:

File deleted and changed	Use modified or deleted file?
File deleted and created	Use created or deleted file?
File changed both locally and remotely	Start merge tool.

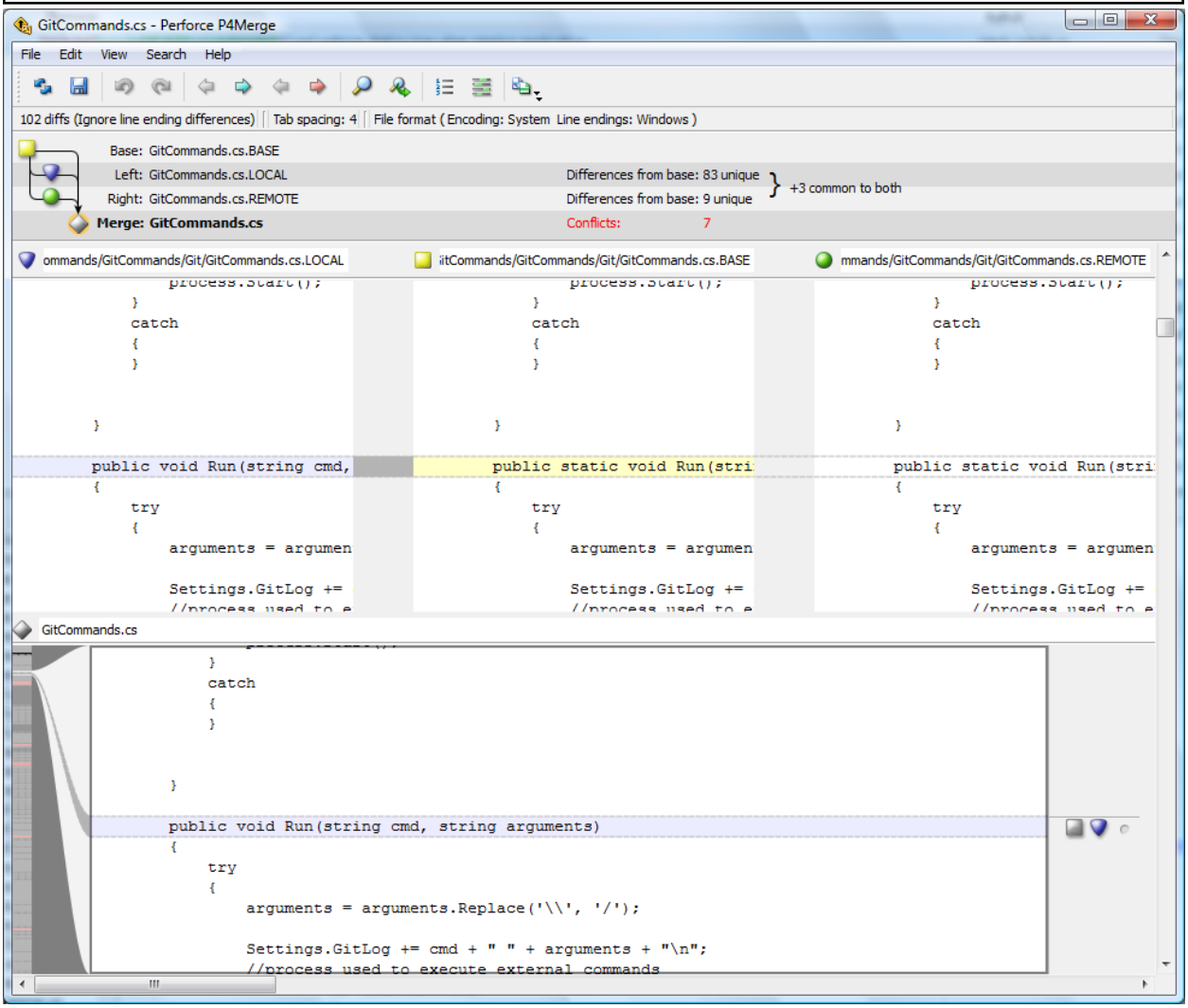
If the file is deleted in one commit and changed in another commit, a dialog will ask to keep the modified file or delete the file. When there is a conflicting change the merge tool will be started. You can configure the tool you want to use

for merge conflicts. The image below shows Perforce P4Merge a free to use merge tool. Git Extensions is packaged with KDiff3, an open source merge tool.

In the merge tool you will see four versions of the same file:

Base	The latest version of the file that exist in both repositories
Local	The latest local version of the file
Remote	The latest remote version of the file
Merged	The result of the merge

Caution: When you are in the middle of a merge the file named local represents your file. When you are in the middle of a rebase the file named remote represents your file. This can be confusing, so double check if you are in doubt.

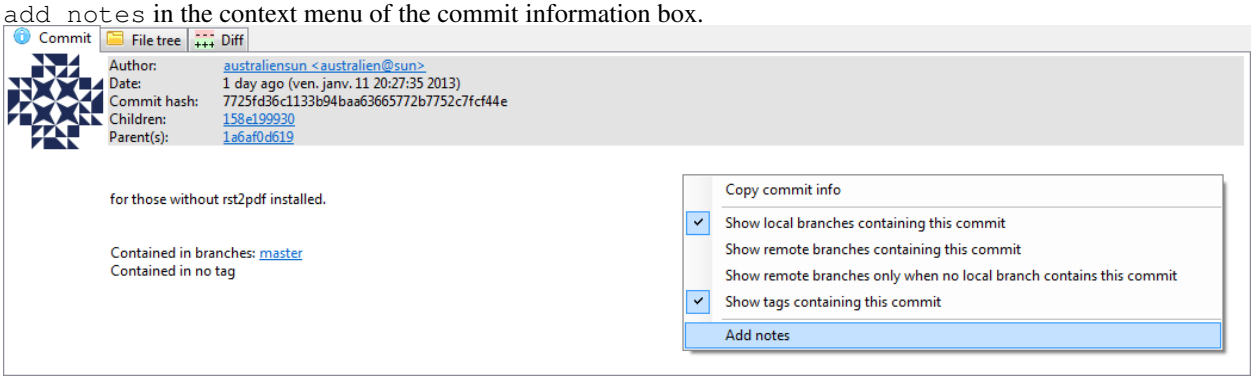


CHAPTER

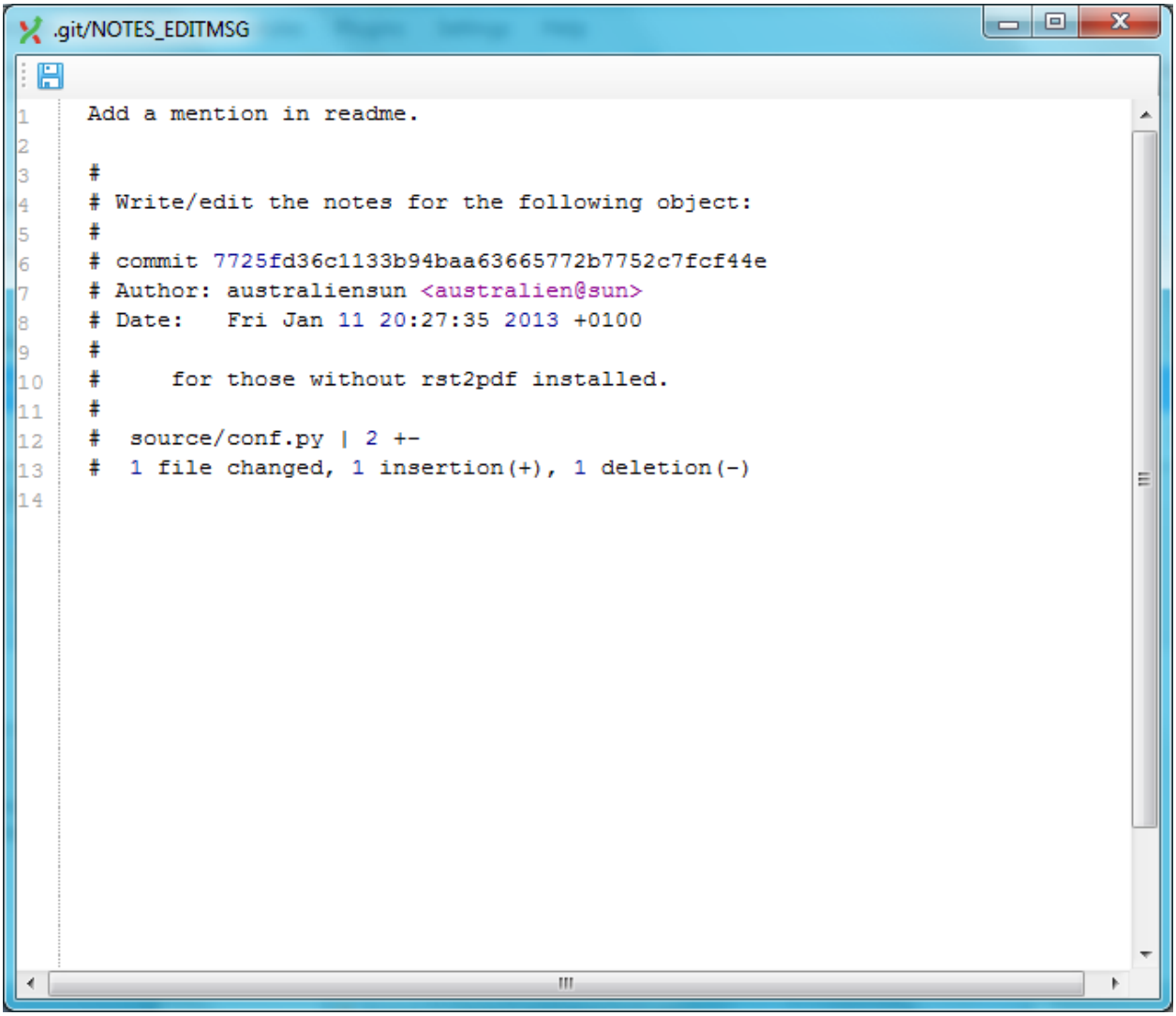
TEN

NOTES

Notes can be added to a commit. Notes will be stored separately and will not be pushed. To add a new note choose `add notes` in the context menu of the commit information box.



The editor that has been configured in the settings dialog will be used to enter or edit the notes. The Git Extensions editor is advised.

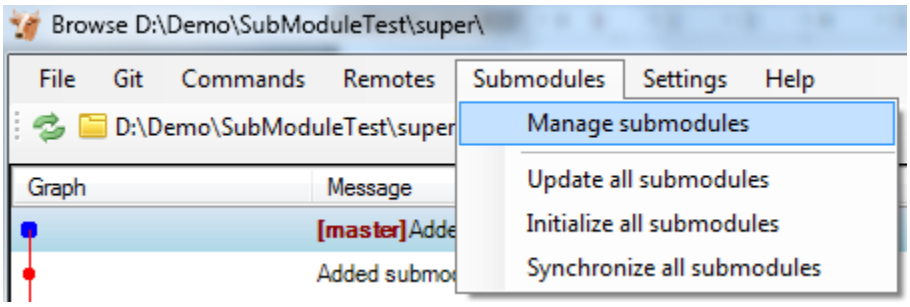

.git/NOTES_EDITMSG
Add a mention in readme.
#
Write/edit the notes for the following object:
#
commit 7725fd36c1133b94baa63665772b7752c7fcf44e
Author: australiensun <australien@sun>
Date: Fri Jan 11 20:27:35 2013 +0100
#
for those without rst2pdf installed.
#
source/conf.py | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)

CHAPTER

ELEVEN

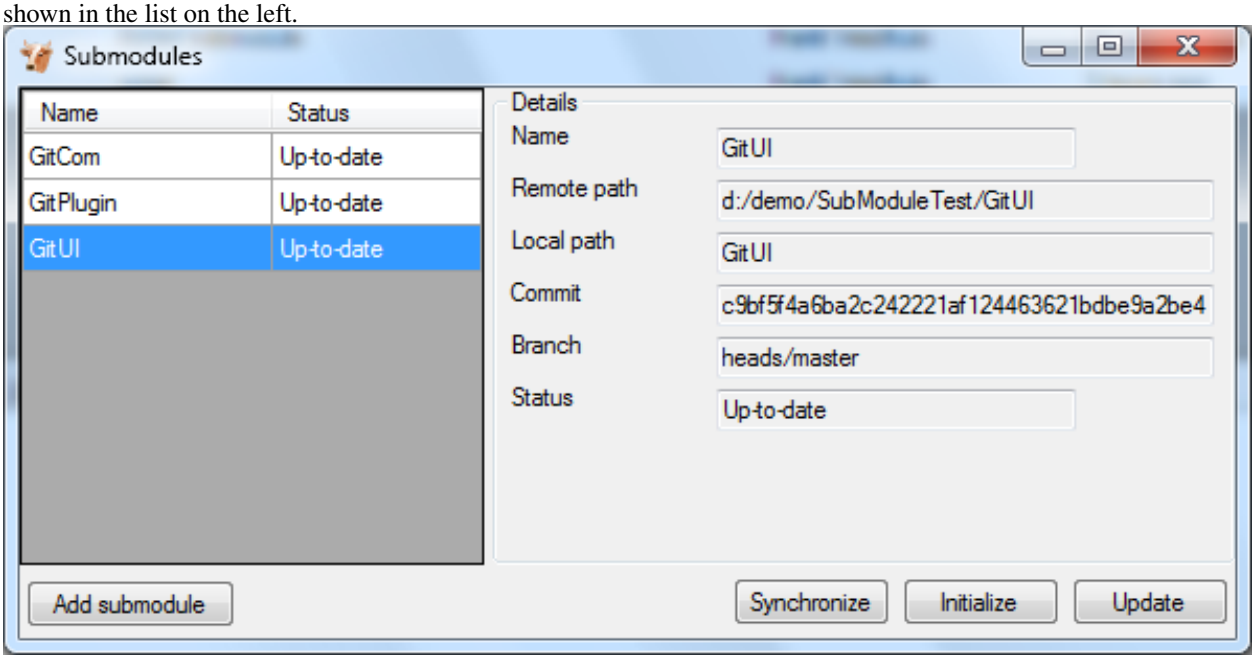
SUBMODULES

Large projects can be split into smaller parts using submodules. A submodule contains the name, url and revision of another repository. To create a submodule in an existing git repository you need to add a link to another repository containing the files of the submodule.



11.1 Manage submodules

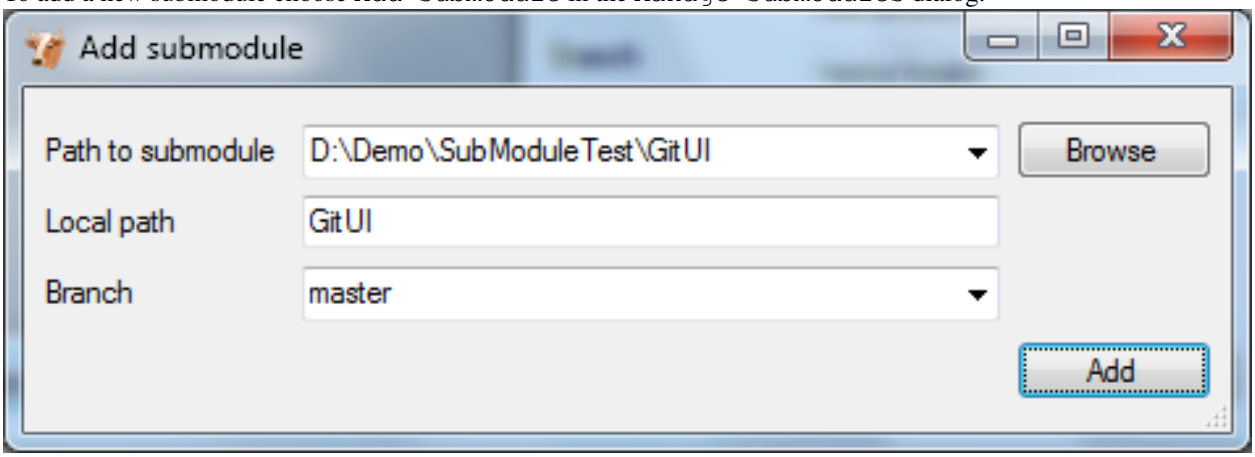
The current state of the submodules can be viewed with the `Manage submodules` function. All submodules are shown in the list on the left.



Add submodule	Add a new submodule to the repository
Synchronize	Synchronizes the remote URL configuration setting to the value specified in `.gitmodules` for the selected submodule.
Initialize	Initialize the selected submodules, i.e. register each submodule name and url found in `.gitmodules` into `.git/config`. The submodule will also be updated.
Update	Update the registered submodules, i.e. clone missing submodules and checkout the commit specified in the index of the containing repository.

11.2 Add submodule

To add a new submodule choose `Add submodule` in the `Manage submodules` dialog.



Path to submodule	Path to the remote repository to use as submodule.
Local path	Local path to this submodule, relative to the root of the current repository.
Branch	Branch to track.

11.3 Remove submodule

It is currently not possible to remove a submodule using the Git Extensions user interface. To remove a submodule you need to manually:

- Delete the relevant line from the `.gitmodules` file.
- Delete the relevant section from `.git/config`.
- Run `git rm --cached path_to_submodule` (no trailing slash).
- Commit and delete the now untracked submodule files.

CHAPTER

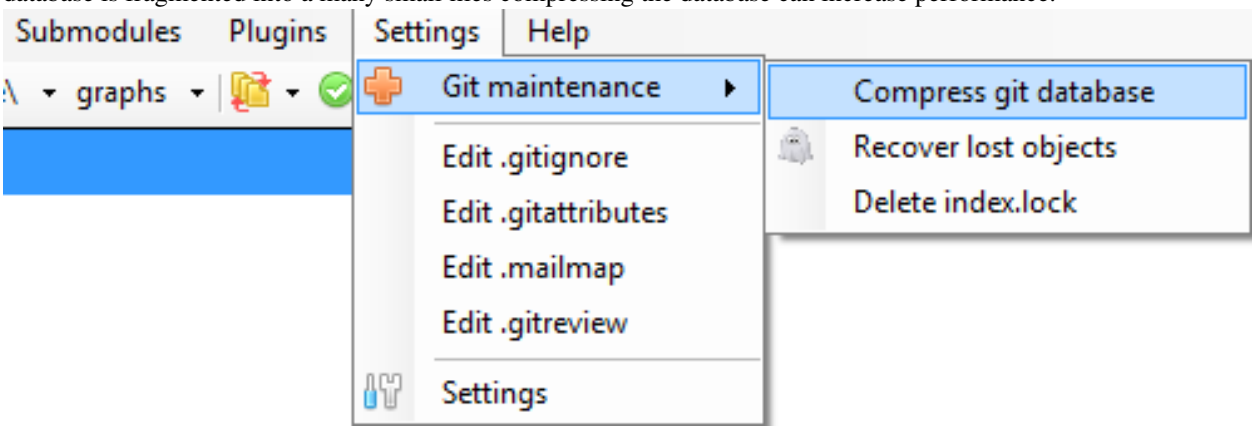
TWELVE

MAINTENANCE

In this chapter some of the functions to maintain a repository are discussed.

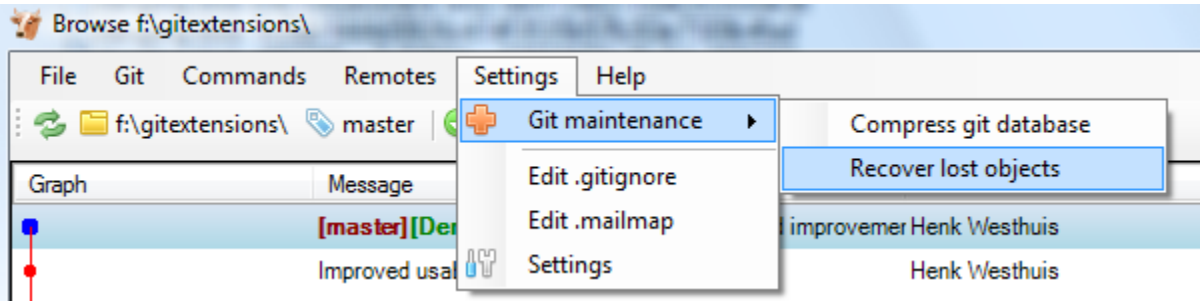
12.1 Compress Git database

Git will create a lot of files. You can run the `Compress git database` to pack all small files building up a repository into one big file. Git will also garbage collect all unused objects that are older then 15 days. When a database is fragmented into a many small files compressing the database can increase performance.

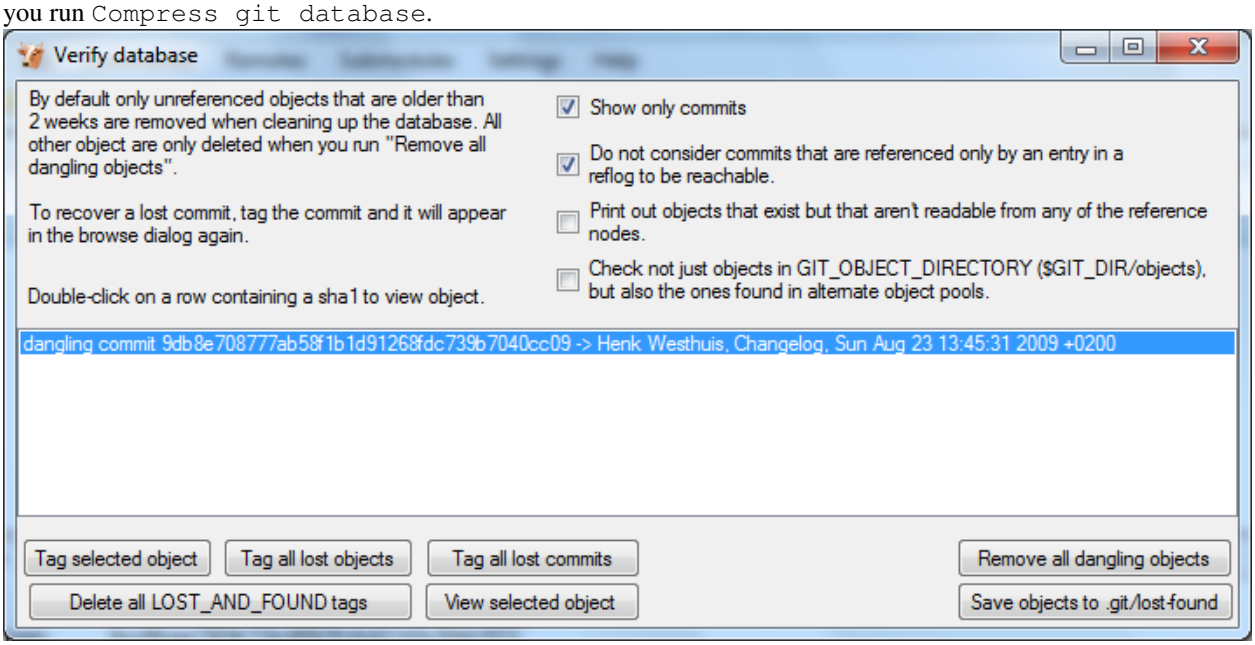


12.2 Recover lost objects

If you accidently deleted a commit you can try to recover it using the `Recover lost objects` function. A dialog will show you all dangling objects and will allow you to review and recover them.

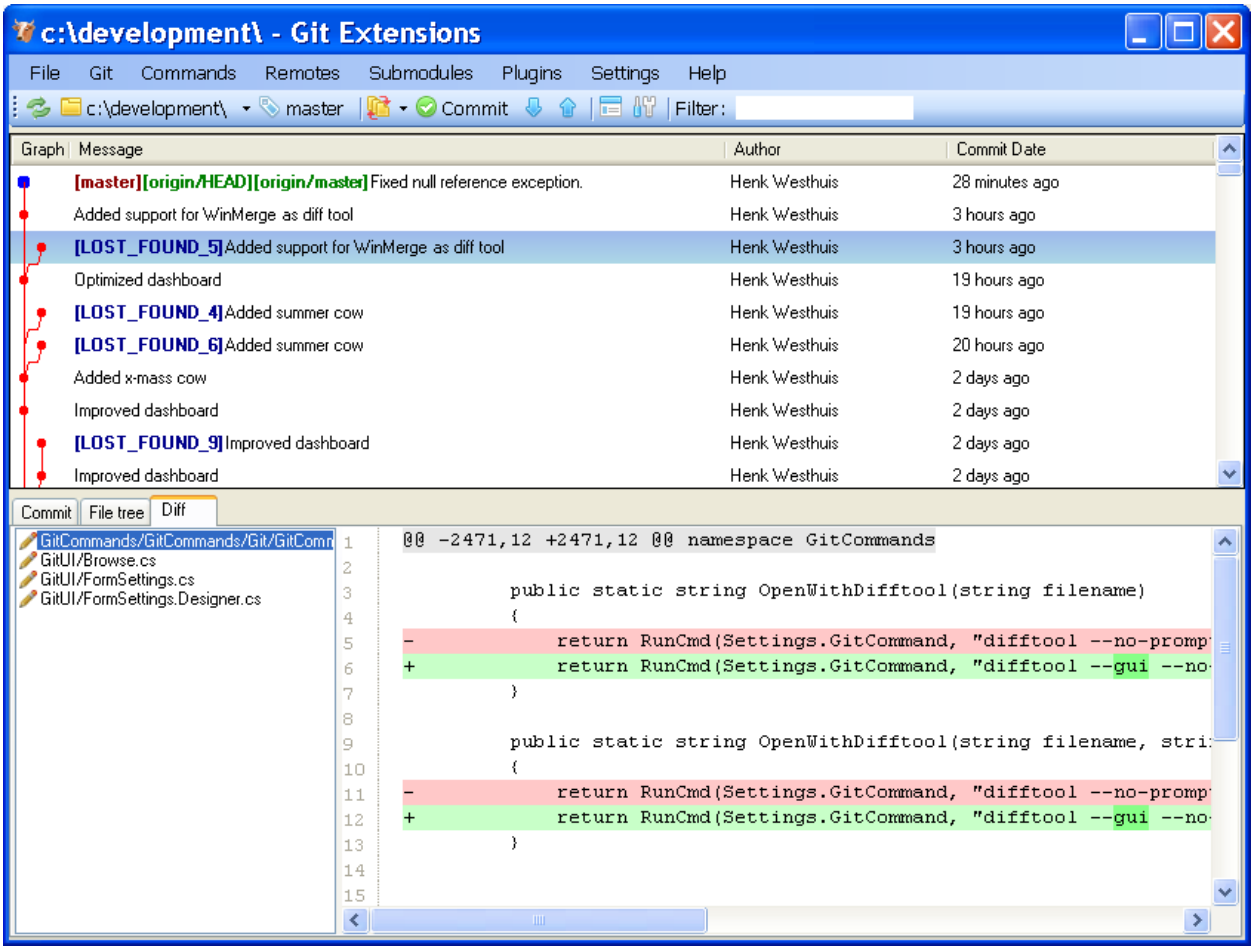


Normally Git will not delete files right away when you remove something from your repository. The reason for this is that you can restore deleted items if you need to. Git will delete removed items when they are older then 15 days and you run `Compress git database`.



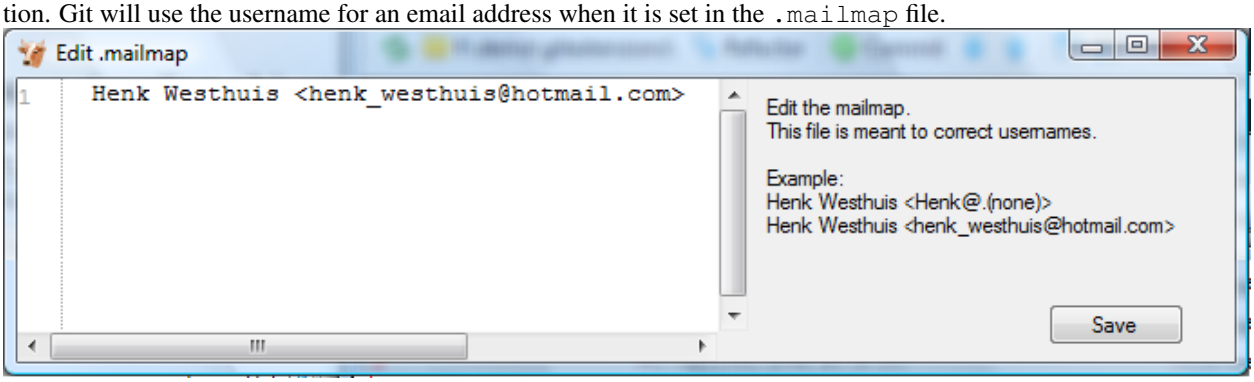
There are several functions to help you find the lost items. By default Git Extensions will only show commits. To show all items, just uncheck the `Show only commits` option. The other options can be checked/unchecked to get more/less results. Double-click on on item to view the content. When you located the item you want to recover you can tag it using the `Tag selected object` button.

Git Extensions also is able to tag all lost objects. Doing this will make all lost objects visible again making it very easy to locate the commit(s) you would like to recover. After recovering a commit using the `Tag all lost commits` button, you can remove all tags using the `Delete all LOST_AND_FOUND tags` button.



12.3 Fix user names

When someone accidentally committed using a wrong username this can be fixed using the `Edit .mailmap` function. Git will use the username for an email address when it is set in the `.mailmap` file.



Fix user name using commit email:

```
Proper Name <commit@email.xx>
```

Fix email address using commit email:

```
<proper@email.xx> <commit@email.xx>
```

Fix email address and name using commit email:

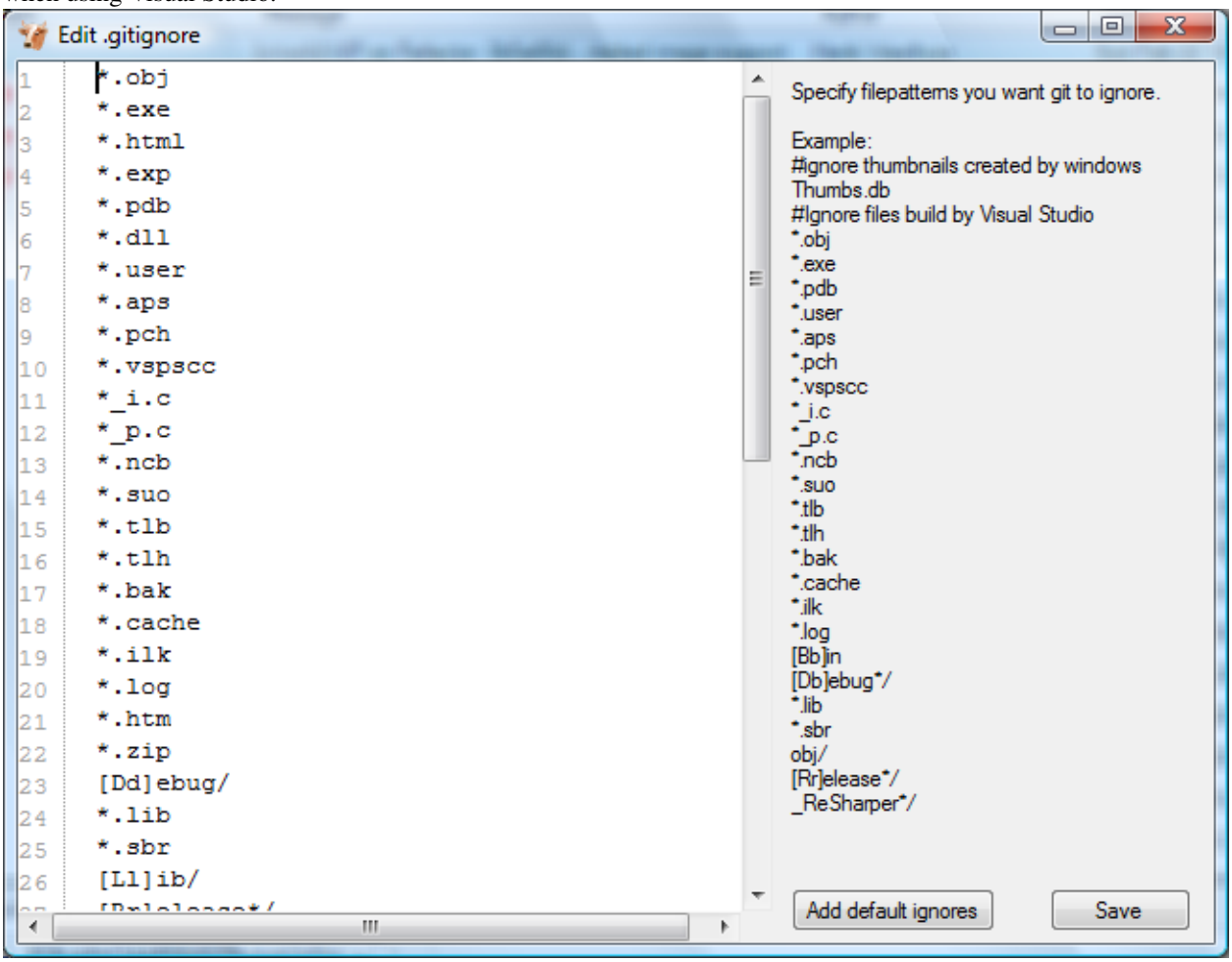
```
Proper Name <proper@email.xx> <commit@email.xx>
```

Fix email address and name using commit name and email:

```
Proper Name <proper@email.xx> Commit Name <commit@email.xx>
```

12.4 Ignore files

Git will track all files that are in the working directory. Normally you do not want to exclude all files that are created by the compiler. You can add files that should be ignored to the .gitignore file. You can use wildcards and regular expressions. All entries are case sensitive. The button `Add default ignores` will add files that should be ignored when using Visual Studio.



A short overview of the syntax:

#	Lines started with `#` are handled as comments
!	Lines started with `!` are exclude patterns
[Dd]	Characters inside `[..]` means that 1 of the characters must match
*	Wildcard
/	A leading slash matches the beginning of the pathname; for example, `/*.c` matches `cat-file.c` but not `mozilla-sha1/sha1.c`
/	If the pattern ends with a slash, it is removed for the purpose of the following description, but it would only find a match with a directory. In other words, foo/ will match a directory foo and paths underneath it, but will not match a regular file or a symbolic link foo (this is consistent with the way how pathspec works in general in git).

For more detailed information.

CHAPTER

THIRTEEN

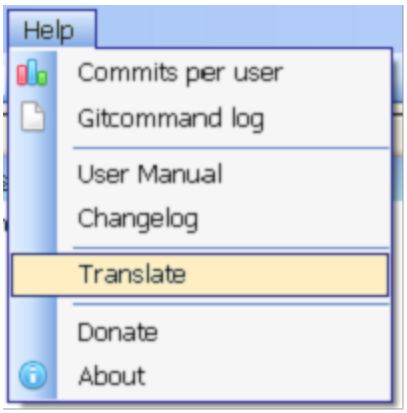
TRANSLATIONS

13.1 Change language

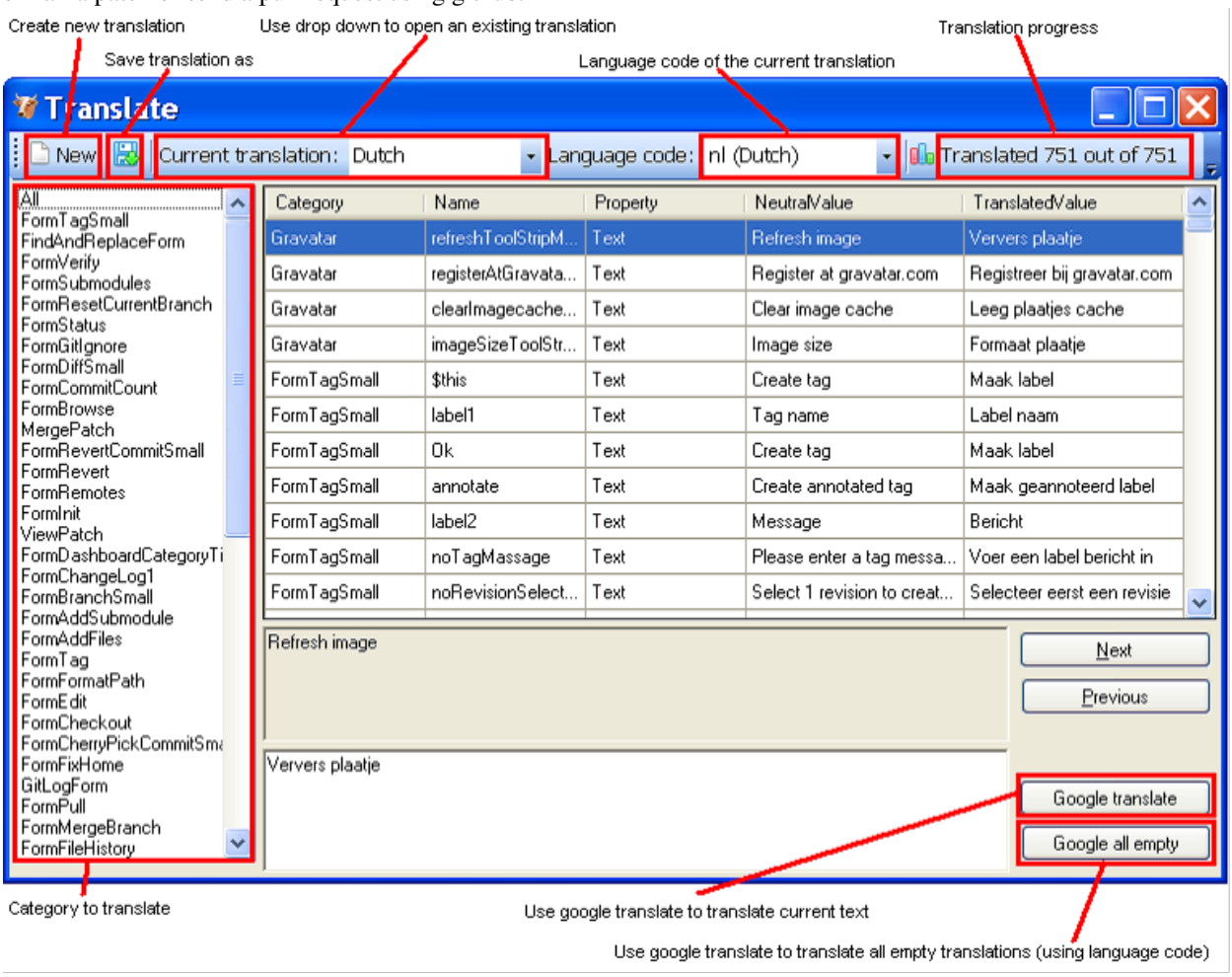
In the settings dialog a translation can be chosen. The translation files are located in a directory located in the Git Extensions installation directory. The files are readable xml files.

13.2 Translate Git Extensions

The application has a built-in translation tool to help create and edit translations. To open the translation tool choose `Translate` in the `Help` menu.



The functions of the translation tool are described in the image below. To contribute any translations you can either e-mail a patch or send a pull request using github.



CHAPTER

FOURTEEN

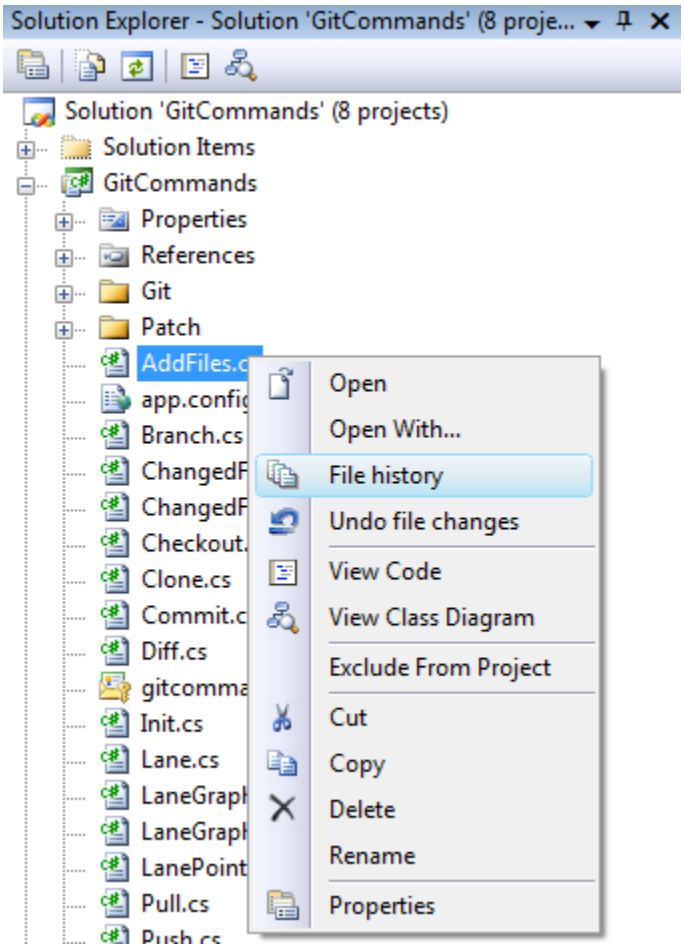
INTEGRATION

During installation you can choose to install the Visual Studio plug-in and shell extensions.

14.1 Visual Studio

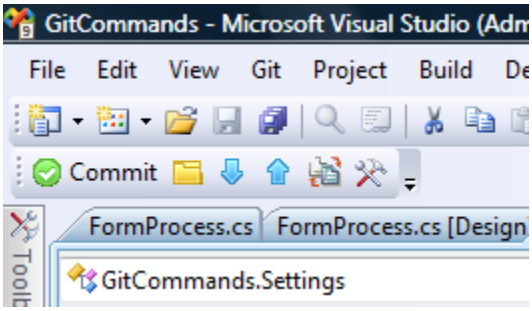
There are two options in the context menu on files:

- View the file history by choosing the 'File history' option.
- Reset the file changes to the last committed revision.

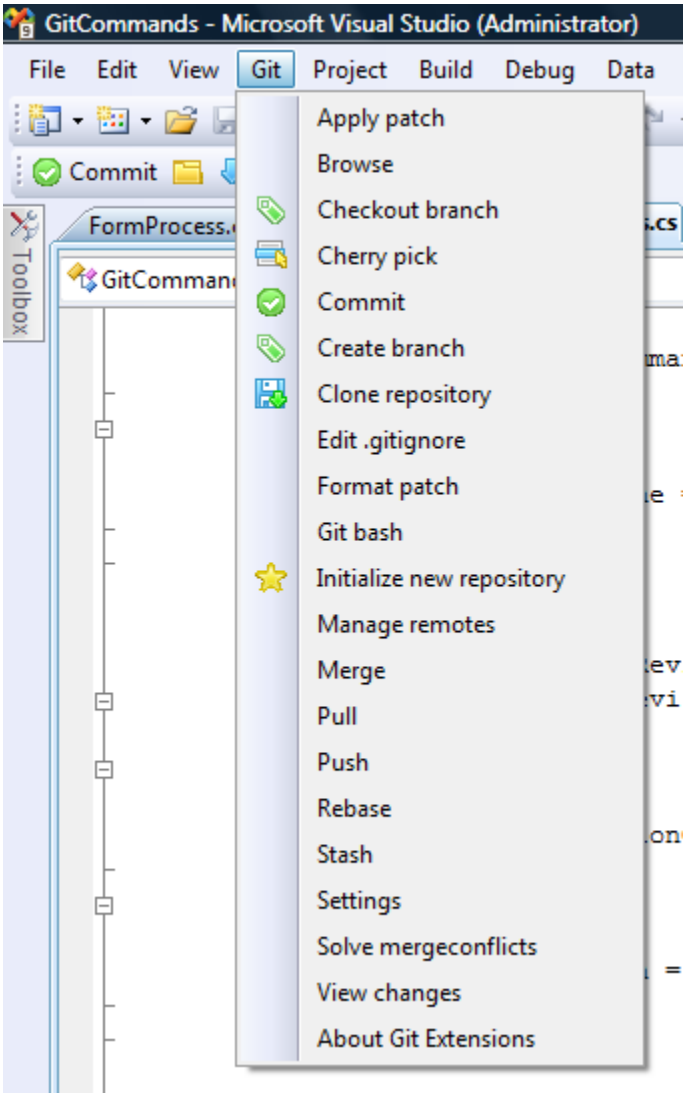


A Git Extensions toolbar allows you to perform the most common actions.

	Commit (branch)
	Browse
	Pull
	Push
	Stash changes
	Settings

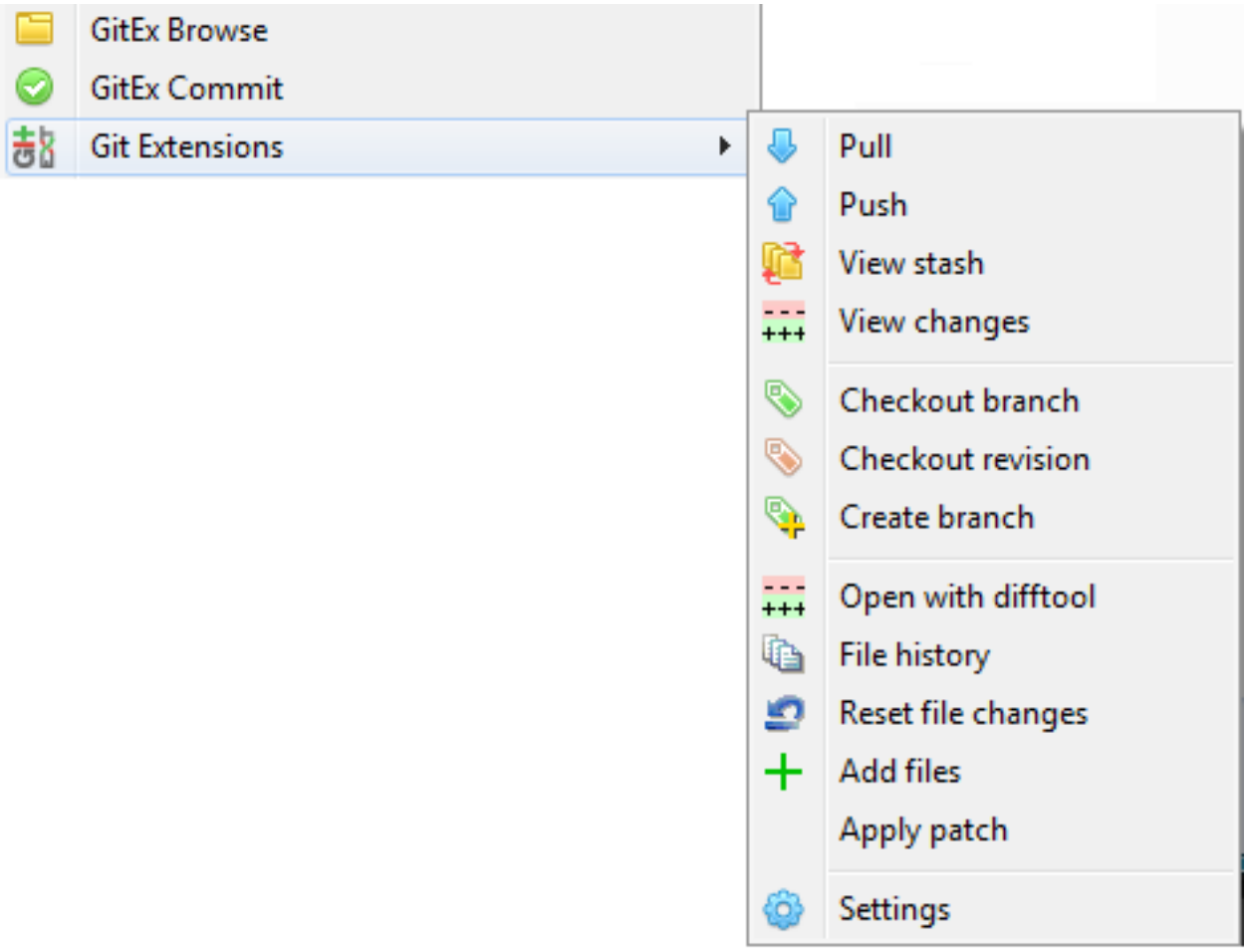


Almost all function can be started from the `Git` menu in Visual Studio.

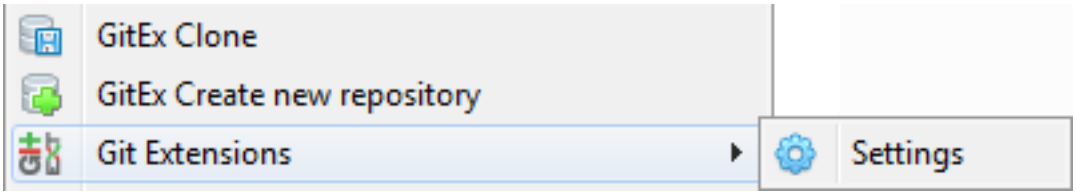


14.2 Windows Explorer

The common commands can be started from Windows Explorer using the shell extensions. This option is only available when Shell Extensions are installed.



You can even create or clone a repository in any non git folder.



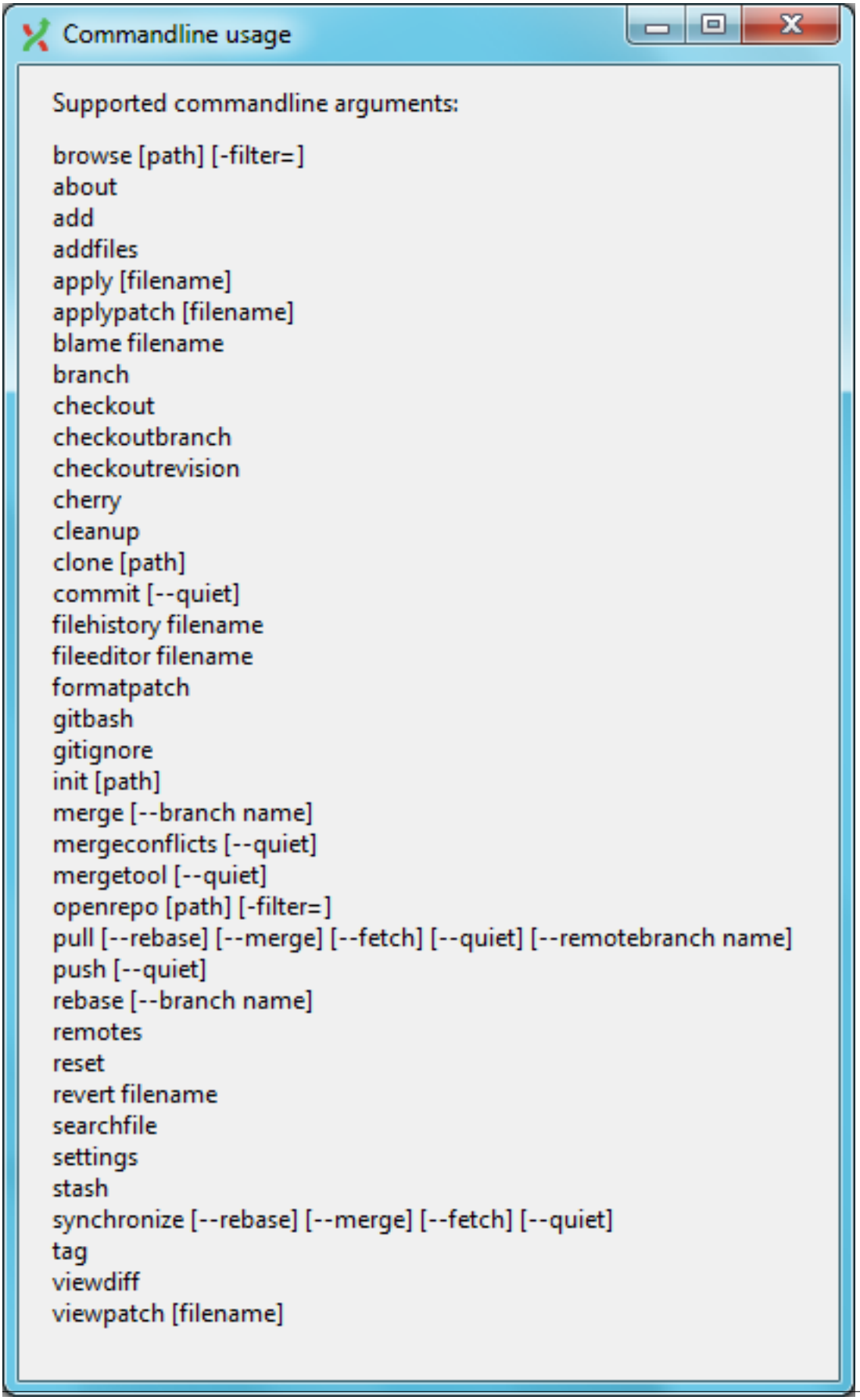
CHAPTER

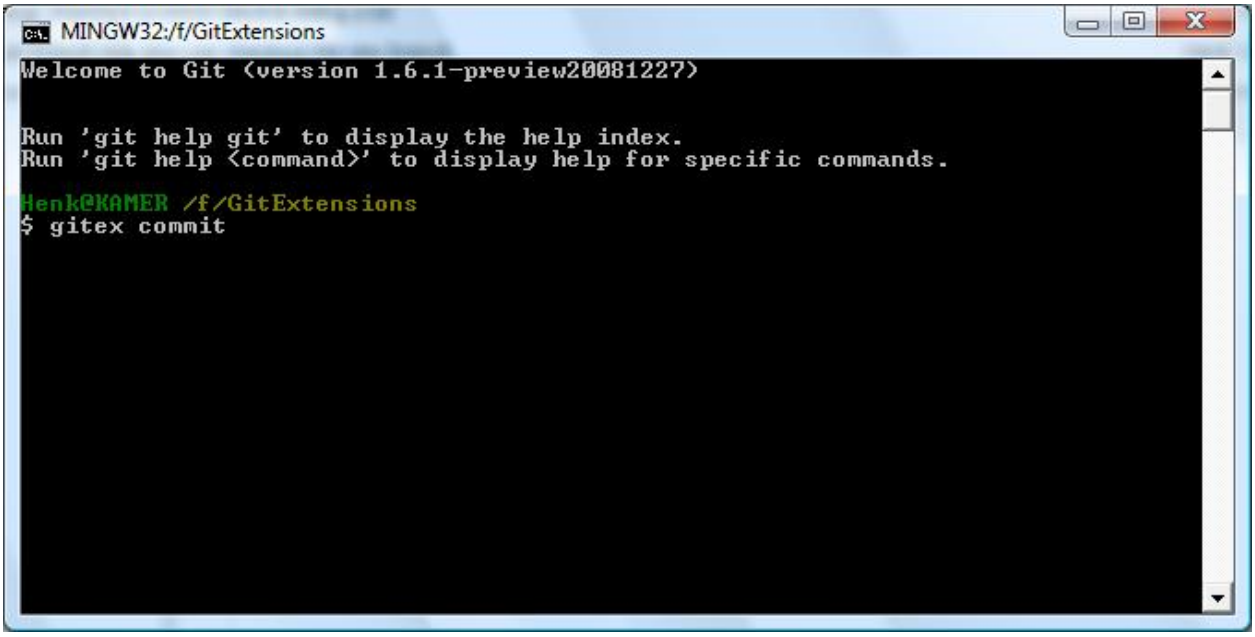
FIFTEEN

COMMAND LINE

15.1 Git Extensions command line

Most features can be started from the command line. It is recommended to add `gitex.cmd` to the path when using from the command line. It is typically stored in the `C:\Program Files (x86)\GitExtensions` folder.


Commandline usage
Supported commandline arguments:
browse [path] [-filter=]
about
add
addfiles
apply [filename]
applypatch [filename]
blame filename
branch
checkout
checkoutbranch
checkoutrevision
cherry
cleanup
clone [path]
commit [--quiet]
filehistory filename
fileeditor filename
formatpatch
gitbash
gitignore
init [path]
merge [--branch name]
mergeconflicts [--quiet]
mergetool [--quiet]
openrepo [path] [-filter=]
pull [--rebase] [--merge] [--fetch] [--quiet] [--remotebranch name]
push [--quiet]
rebase [--branch name]
remotes
reset
revert filename
searchfile
settings
stash
synchronize [--rebase] [--merge] [--fetch] [--quiet]
tag
viewdiff
viewpatch [filename]


MINGW32:/f/GitExtensions
Welcome to Git (version 1.6.1-preview20081227)
Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.
Henk@KAMER /f/GitExtensions
$ gitex commit

CHAPTER

SIXTEEN

APPENDIX

16.1 Git Cheat Sheet

Action	Command
Create new repository	*$ git init*
Create shared repository	*$ git init —bare —shared=all*
Clone repository	*$ git clone c:/demo1 c:/demo2*
Checkout branch	*$ git checkout <name>*
Create branch	*$ git branch <name>*
Delete branch	*$ git branch -d <name>*
Merge branch (from the branch to merge into):	*$ git merge PDC*
Solve conflicts (add –tool=kdiff3 if no mergetool is specified)	*$ git mergetool $ git commit*
Create tag	*$ git tag <name>*
Add files/changes (. for all files)	*$ git add .*
Commit added files/changes (–amend to amend to last commit)	*$ git commit –m "Enter commit message"*
Discard changes	*$ git reset –hard*
Create patch (-M = detect renames –C = detect copies)	*$ git format-patch –M –C origin*
Apply patch without merging	*$ git apply c:/patch/01-emp.patch*
Merge patch	*$ git am -–3way —signoff c:/patch/01-emp.patch*
Solve conflicts (add –tool=kdiff3 if no mergetool is specified)	*$ git mergetool* *$ git am —3way -–resolved*
Stash changes	*$ git stash*
Apply stashed changes	*$ git stash apply*
Pull changes (add –rebase to rebase instead of merge)	*$ git pull c:/demo1 master*
Solve conflicts (add –tool=kdiff3 if no mergetool is specified)	*$ git mergetool* *$ git commit*
Push changes (in branch $ git push c:/demo1 master master:<new>)	*$ git push c:/demo1*
Blame	*$ git blame –M –w <filename>*
Help	*$ git <command> –help*

Here are some default names used by Git.

Default names	
master	default branch
origin	default upstream repository
HEAD	current branch
HEAD^	parent of HEAD
HEAD~4	the great-great grandparent of HEAD

16.2 Menu map

The following image shows GitExtensions' menu structure at one glance (v2.43):