TurtoSigeIT日常使用指南（直译版）

原文地址：<https://tortoisegit.org/docs/tortoisegit/tgit-dug.html>

**Table of Contents**

[Getting Started](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general)

[Icon Overlays](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-icons)

[Context Menus](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-menu)

[Drag and Drop](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-drag)

[Common Shortcuts](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-shortcuts)

[Authentication](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-auth)

[Maximizing Windows](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-window-max)

[Create Repository](https://tortoisegit.org/docs/tortoisegit/tgit-dug-create.html)

[Clone Repository](https://tortoisegit.org/docs/tortoisegit/tgit-dug-clone.html)

[Checking Out A Working Tree (Switch to commit)](https://tortoisegit.org/docs/tortoisegit/tgit-dug-checkout.html)

[Committing Your Changes To The Repository](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html)

[The Commit Dialog](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-basics)

[Change Lists](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-changelists)

[Excluding Items from the Commit List](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-ignore)

[Commit only parts of files](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-restore)

[Commit Log Messages](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-log)

[Commit Progress](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-progress)

[Getting Status Information](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html)

[Icon Overlays](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html#tgit-dug-wcstatus-1)

[Status](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html#tgit-dug-wcstatus-2)

[Viewing Diffs](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html#tgit-dug-diffing)

[Pull and Fetch change](https://tortoisegit.org/docs/tortoisegit/tgit-dug-pull.html)

[Push](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html)

[Branch](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html#id605095)

[Destination](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html#id604404)

[Options](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html#id603445)

[Sync](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html)

[Branch](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html#id604914)

[Destination](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html#id605274)

[Options](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html#id603771)

[Daemon](https://tortoisegit.org/docs/tortoisegit/tgit-dug-daemon.html)

[Browse All Refs](https://tortoisegit.org/docs/tortoisegit/tgit-dug-browse-ref.html)

[Submodules](https://tortoisegit.org/docs/tortoisegit/tgit-dug-submodules.html)

[Change Lists](https://tortoisegit.org/docs/tortoisegit/tgit-dug-changelists.html)

[Log Dialog](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html)

[Invoking the Revision Log Dialog](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-1)

[Revision Log Actions](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-actions)

[Getting Additional Information](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-2)

[Filtering Log Messages](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-filter)

[Navigation](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-navigation)

[Statistical Information](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-5)

[Refreshing the View](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-refresh)

[Revision Graphs](https://tortoisegit.org/docs/tortoisegit/tgit-dug-revgraph.html)

[Revision Graph Nodes](https://tortoisegit.org/docs/tortoisegit/tgit-dug-revgraph.html#tgit-dug-revgraph-nodes)

[Using the Graph](https://tortoisegit.org/docs/tortoisegit/tgit-dug-revgraph.html#tgit-dug-revgraph-use)

[Refreshing the View](https://tortoisegit.org/docs/tortoisegit/tgit-dug-revgraph.html#tgit-dug-revgraph-refresh)

[Reference Log](https://tortoisegit.org/docs/tortoisegit/tgit-dug-reflog.html)

[The Repository Browser](https://tortoisegit.org/docs/tortoisegit/tgit-dug-repobrowser.html)

[Viewing Differences](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html)

[File Differences](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-diff-file)

[Line-end and Whitespace Options](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-diff-eol-whitespace)

[Comparing Version](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-compare-revs)

[Diffing submodules using Submodule Diff Dialog](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-submodulediff)

[Diffing Images Using TortoiseGitIDiff](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-idiff)

[External Diff/Merge Tools](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-diff-tools)

[Adding New Files](https://tortoisegit.org/docs/tortoisegit/tgit-dug-add.html)

[Copying/Moving/Renaming Files and Folders](https://tortoisegit.org/docs/tortoisegit/tgit-dug-copy.html)

[Ignoring Files And Directories](https://tortoisegit.org/docs/tortoisegit/tgit-dug-ignore.html)

[Pattern Matching in Ignore Lists](https://tortoisegit.org/docs/tortoisegit/tgit-dug-ignore.html#tgit-dug-ignore-glob)

[Deleting, Moving and Renaming](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html)

[Deleting files and folders](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html#tgit-dug-rename-delete)

[Moving files and folders](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html#tgit-dug-rename-move)

[Changing case in a filename](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html#tgit-dug-renameincase)

[Dealing with filename case conflicts](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html#tgit-dug-rename-case-conflict)

[Deleting Unversioned Files](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html#tgit-dug-rename-del-unversioned)

[Undo Changes](https://tortoisegit.org/docs/tortoisegit/tgit-dug-revert.html)

[Cleanup](https://tortoisegit.org/docs/tortoisegit/tgit-dug-cleanup.html)

[Reset](https://tortoisegit.org/docs/tortoisegit/tgit-dug-reset.html)

[Stash Changes](https://tortoisegit.org/docs/tortoisegit/tgit-dug-stash.html)

[Bisect](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bisect.html)

[Branching/Tagging](https://tortoisegit.org/docs/tortoisegit/tgit-dug-branchtag.html)

[Creating a Branch or Tag](https://tortoisegit.org/docs/tortoisegit/tgit-dug-branchtag.html#tgit-dug-branch-1)

[Merging](https://tortoisegit.org/docs/tortoisegit/tgit-dug-merge.html)

[Cherry picking](https://tortoisegit.org/docs/tortoisegit/tgit-dug-cherrypick.html)

[Rebase](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rebase.html)

[Resolving Conflicts](https://tortoisegit.org/docs/tortoisegit/tgit-dug-conflicts.html)

[Special conflict cases](https://tortoisegit.org/docs/tortoisegit/tgit-dug-conflicts.html#tgit-dug-conflicts-special)

[Creating and Applying Patches and Pull Requests](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html)

[Creating a Patch Serial](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-createpatchserial)

[Sending patches by mail](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-sendpatches)

[Applying a single Patch File](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-applysinglepatch)

[Applying a Patch Serial](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-applaypatchserial)

[Creating a pull request](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-pullrequest)

[Who Changed Which Line?](https://tortoisegit.org/docs/tortoisegit/tgit-dug-blame.html)

[Blame for Files](https://tortoisegit.org/docs/tortoisegit/tgit-dug-blame.html#tgit-dug-blame-simple)

[Exporting a Git Working Tree](https://tortoisegit.org/docs/tortoisegit/tgit-dug-export.html)

[Integration with Bug Tracking Systems / Issue Trackers](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html)

[Adding Issue Numbers to Log Messages](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html#tgit-dug-bugtracker-link)

[Getting Information from the Issue Tracker](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html#tgit-dug-bugtracker-ref)

[TortoiseGit's Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html)

[General Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-main)

[Icon Overlay Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-overlay)

[Network Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-network)

[External Program Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-progs)

[Saved Data Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-saved-data)

[Git](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-git)

[Client Side Hook Scripts](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-hooks)

[TortoiseGitBlame Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-tortoiseblame)

[TortoiseGitUDiff Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tsvn-dug-settings-tortoiseudiff)

[Advanced Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-registry)

[Exporting TortoiseGit Settings](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-export)

[git svn dcommit](https://tortoisegit.org/docs/tortoisegit/tgit-dug-svn-dcommit.html)

[Final Step](https://tortoisegit.org/docs/tortoisegit/tgit-dug-donate.html)

**目录**

[入门指南](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general)

[图标覆盖](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-icons)

[菜单](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-menu)

[拖放](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-drag)

[常用快捷方式](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-shortcuts)

[认证](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-auth)

[最大化窗口](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-window-max)

[创建存储库](https://tortoisegit.org/docs/tortoisegit/tgit-dug-create.html)

[克隆库](https://tortoisegit.org/docs/tortoisegit/tgit-dug-clone.html)

检出工作树(切换到提交)

[将更改提交到存储库](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html)

[提交对话框](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-basics)

[更改列表](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-changelists)

从提交列表中排除项目

[仅提交部分文件](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-restore)

[提交日志消息](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-log)

[提交进度](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-progress)

[获取状态信息](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html)

[图标覆盖](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html#tgit-dug-wcstatus-1)

[状态](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html#tgit-dug-wcstatus-2)

[查看差异](https://tortoisegit.org/docs/tortoisegit/tgit-dug-wcstatus.html#tgit-dug-diffing)

[拉取](https://tortoisegit.org/docs/tortoisegit/tgit-dug-pull.html)或获取改变

[推](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html)送

[分支](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html#id605095)

[目的地](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html#id604404)

[选项](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html#id603445)

[同步](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html)

[分支](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html#id604914)

[目的地](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html#id605274)

[选项](https://tortoisegit.org/docs/tortoisegit/tgit-dug-sync.html#id603771)

[守护进程](https://tortoisegit.org/docs/tortoisegit/tgit-dug-daemon.html)

[浏览所有参照](https://tortoisegit.org/docs/tortoisegit/tgit-dug-browse-ref.html)

[子模块](https://tortoisegit.org/docs/tortoisegit/tgit-dug-submodules.html)

[更改列表](https://tortoisegit.org/docs/tortoisegit/tgit-dug-changelists.html)

[日志对话框](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html)

调用修订日志对话框

修订日志操作

[获取其他信息](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-2)

[过滤日志消息](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-filter)

[导航](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-navigation)

[统计信息](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-5)

[刷新视图](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html#tgit-dug-showlog-refresh)

修订图

[修订图节点](https://tortoisegit.org/docs/tortoisegit/tgit-dug-revgraph.html#tgit-dug-revgraph-nodes)

使用图表

[刷新视图](https://tortoisegit.org/docs/tortoisegit/tgit-dug-revgraph.html#tgit-dug-revgraph-refresh)

[参考日志](https://tortoisegit.org/docs/tortoisegit/tgit-dug-reflog.html)

[存储库浏览器](https://tortoisegit.org/docs/tortoisegit/tgit-dug-repobrowser.html)

[查看差异](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html)

[文件差异](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-diff-file)

行尾和空白选项

[比较版本](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-compare-revs)

[使用子模块Diff对话框扩散子模块](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-submodulediff)

[利用TurtoSeigTiffDIFI图像](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-idiff)

[外部比较/合并工具](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-diff-tools)

[添加新文件](https://tortoisegit.org/docs/tortoisegit/tgit-dug-add.html)

[复制/移动/重命名文件和文件夹](https://tortoisegit.org/docs/tortoisegit/tgit-dug-copy.html)

[忽略文件和目录](https://tortoisegit.org/docs/tortoisegit/tgit-dug-ignore.html)

[忽略列表中的模式匹配](https://tortoisegit.org/docs/tortoisegit/tgit-dug-ignore.html#tgit-dug-ignore-glob)

[删除、移动和重命名](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html)

[删除文件和文件夹](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html#tgit-dug-rename-delete)

[移动文件和文件夹](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rename.html#tgit-dug-rename-move)

更改文件名中的大小写

处理文件名大小写冲突

删除未版本化的文件

撤消更改

[清理](https://tortoisegit.org/docs/tortoisegit/tgit-dug-cleanup.html)

[重置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-reset.html)

[贮藏变化](https://tortoisegit.org/docs/tortoisegit/tgit-dug-stash.html)

平分

分支/标记

[创建分支或标](https://tortoisegit.org/docs/tortoisegit/tgit-dug-branchtag.html#tgit-dug-branch-1)记

[合并](https://tortoisegit.org/docs/tortoisegit/tgit-dug-merge.html)

[Cherry picking](https://tortoisegit.org/docs/tortoisegit/tgit-dug-cherrypick.html)

[Rebase](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rebase.html)（变基）

[解决冲突](https://tortoisegit.org/docs/tortoisegit/tgit-dug-conflicts.html)

特殊冲突案件

创建和应用修补程序和拉取请求

[创建补丁序列](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-createpatchserial)

[通过邮件发送补丁](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-sendpatches)

[应用单个补丁文件](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-applysinglepatch)

[应用补丁序列](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-applaypatchserial)

[创建拉动请求](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-pullrequest)

[谁改变了哪条线？](https://tortoisegit.org/docs/tortoisegit/tgit-dug-blame.html)

文件的责任

[导出Git工作树](https://tortoisegit.org/docs/tortoisegit/tgit-dug-export.html)

[与错误跟踪系统/问题跟踪器集成](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html)

[将日志号添加到日志消息](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html#tgit-dug-bugtracker-link)

[从问题跟踪器获取信息](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html#tgit-dug-bugtracker-ref)

[TortoiseGit设置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html)

[一般设置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-main)

图标覆盖设置

[网络设置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-network)

[外部程序设置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-progs)

[保存的数据设置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-saved-data)

[Git](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-git)

[客户端外挂脚本](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-hooks)

TortoiseGitBlame 设置

TortoiseGitUDiff 设置

[高级设置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-registry)

[导出TortoiseGit设置](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-export)

[GIT SvN Debug](https://tortoisegit.org/docs/tortoisegit/tgit-dug-svn-dcommit.html)

[最后一步](https://tortoisegit.org/docs/tortoisegit/tgit-dug-donate.html)

该文档描述了TortoiseGit客户端的日常使用。它不是版本控制系统的介绍，也不是Git的介绍。它更像是一个你知道你想要做什么的地方，但不太记得如何去做的地方。

有关在哪里查找与Git进行版本控制有关的更多信息的提示[“阅读指南”一节](https://tortoisegit.org/docs/tortoisegit/tgit-preface-readingguide.html).

这个文档也是一个正在进行中的工作，就像TortoiseGit和Git一样。如果您发现任何错误，请将其报告给邮件列表，以便我们可以更新文档。日常使用指南（DUG）中的一些屏幕截图可能无法反映软件的当前状态。请原谅我们。我们在业余时间致力于TortoiseGit。

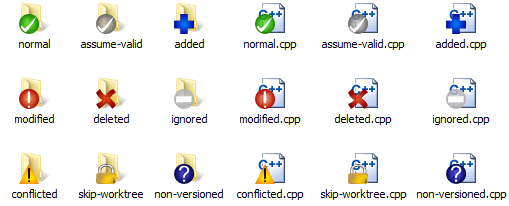
为了最大限度地利用日常使用指南：

* 你应该已经安装了TortoiseGit。
* 您应该熟悉版本控制系统。
* 你应该知道Git的基本知识。
* 您应该设置一个服务器和/或访问一个Git存储库。

## 入门指南

### 图标覆盖

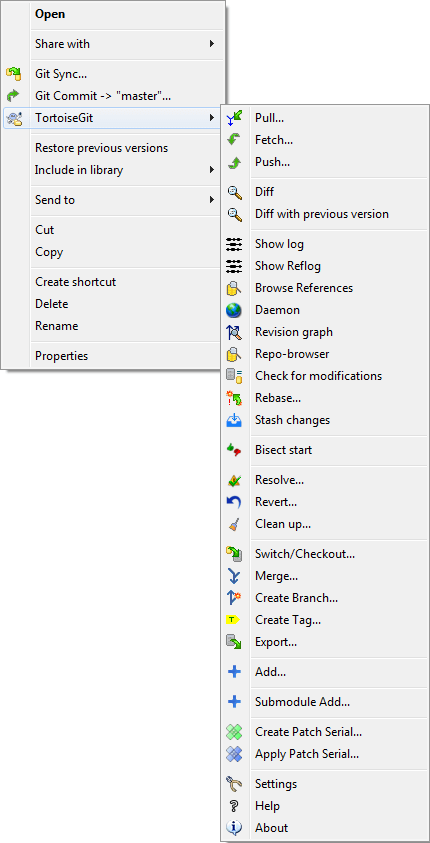
**图2.1。Explorer显示图标叠加**



TortoiseGit最明显的特性之一是出现在工作树中的文件上的图标覆盖。这些文件会让您一目了然地看到哪些文件已被修改。请参考“图标覆盖图”部分，了解不同的覆盖图代表什么。

**菜单**

**图2.2。版本控制下的目录菜单**



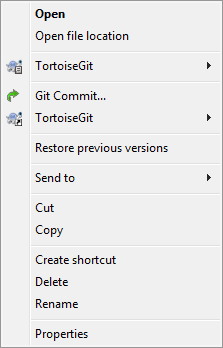
所有的TortoiseGit命令都是从Windows资源管理器的菜单中调用的。大多数是直接可见的，当你**右击**在文件或文件夹上。可用的命令取决于文件或文件夹或其父文件夹是否处于版本控制之下。您还可以将TortoiseGit菜单视为Explorer文件菜单的一部分。

### 小贴士

一些很少使用的命令只在扩展菜单中可用。要打开扩展菜单，请按住**Shift键右击**.

在某些情况下，您可能会看到几个TortoiseGit条目。这不是一个bug！

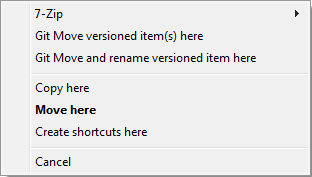
**图2.3。版本控制文件夹中快捷方式的Explorer文件菜单**



此示例用于版本化文件夹中的未版本化快捷方式，并且在资源管理器文件菜单中有两个TortoiseGit条目。一个用于快捷方式本身，第二个用于快捷方式所指向的对象。为了帮助您区分它们，图标的右下角有一个指示器，用于显示菜单项是用于文件、文件夹、快捷方式还是用于多个选定项目。

### 拖放

**图2.4。版本控制下的目录右拖动菜单**



其他命令可用作拖曳处理程序，当您**右键菜单**文件或文件夹到工作树内的新位置或当您**右键菜单**在版本控制下的目录中的非版本文件或文件夹。

### 常用快捷方式

一些常见的操作具有众所周知的Windows快捷方式，但不出现在按钮或菜单中。如果你不知道如何做一些显而易见的事情，比如刷新视图，请在这里检查。

F1

当然，帮助

F5

刷新当前视图。这也许是唯一最有用的一个关键命令。例如。。。在Explorer中，这将刷新在工作树上覆盖的图标。在提交对话框中，它将重新扫描工作树以查看可能需要提交的内容。在修订日志对话框中，它将再次与存储库联系以检查最近的更改。

Ctrl-A

选择全部。如果您收到错误消息并希望复制并粘贴到电子邮件中，则可以使用此方法。使用CTRL A来选择错误消息，然后…

Ctrl-C

…复制选定的文本

Ctrl-F

搜索

### 认证

## 认证。

## SSH(URL看起来像git@example.com)。

## 推荐TortoiseGitPlink作为SSH客户端，因为它可以更好地与Windows集成。默认情况下，TortoiseGitPlink不存储密码，您可以使用PuTTY身份验证代理来缓存密码(如果为远程配置了PuTTY密钥，则自动完成)。有关高级提示和技巧，请参阅附录F，SSH/PuTTY的提示和技巧。但是请注意，TortoiseGitPlink并不尊重特定于OpenSSH的~/.ssh/config(参见PuTTY提示和技巧或将OpenSSH配置为SSH客户端，请参阅下一段)。如果还想在Git Bash上使用TortoiseGitPlink，请创建一个名为git\_SSH的环境变量，其路径为putty plink.exe，或者最好是TortoiseGitPLink.exe。这可以通过重新执行Git for Windows安装程序(在那里可以选择要使用的SSH客户端)、在命令行上执行set git\_SSH=path\_to\_PLINK.EXE“(C：\Program Files\TortoiseGit\bin\TortoiseGitPLink.exe在默认安装上)或永久配置环境变量来完成。

## 也可以使用OpenSSH(随Git for Windows、Cygwin和Msys2一起提供)。只需打开TortoiseGit设置并打开网络页面并输入ssh.exe作为SSH客户端，请参阅“网络设置”一节以及StackOverflow上的此答案。使用OpenSSH时，还可以使用~/.ssh/config(参见。此答案位于StackOverflow上)。

## http/https(URL以https：/或http://)开头。默认情况下，Git不保存/缓存凭据。但是，您可以配置凭据帮助器(推荐，另请参阅“gitcredentials(7)”一节)或手动使用%HOME%/\_netrc。如果您已经设置了凭据存储，并且想要清除一些存储的凭据，请参阅StackOverflow上的此答案。

## 最大化窗口。

## 很多TortoiseGit的对话框都有很多要显示的信息，但通常只最大化高度或宽度，而不是最大化来填充屏幕，这通常是有用的。为了方便起见，在最大化按钮上有此操作的快捷方式。使用鼠标中键垂直最大化，使用鼠标右键水平最大化。

## 创建存储库

本节讨论如何创建Git存储库。创建一个空的Git存储库非常简单。在空目录中，只需使用Explorer菜单并选择**Git Create Repository here （在这里创建存储库）**.

**图2.5。创建仓库对话框**



您可以在这里选择一个空的或正常的Git存储库。一个普通的存储库有一个附加的工作树，文件可以被签出和提交，而一个空的存储库只能被推送和提取。在创建（非空）存储库后，将显示一个消息框：

**图2.6。成功存储库创建消息**



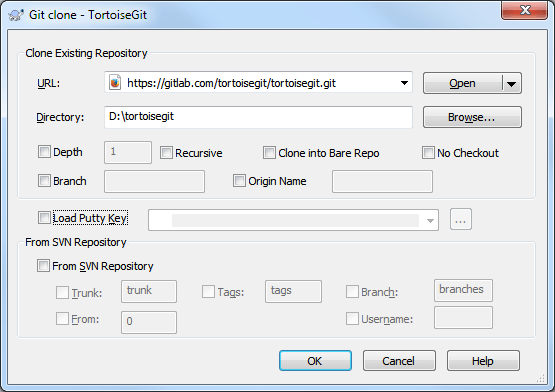
你可以找到更多的信息。[被称为“Git init（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-init(1)).

## 克隆库

本节讨论如何从现有的存储库中克隆Git存储库。此操作用于获取远程存储库的完整副本。克隆Git储存库非常简单。在空目录中，只需使用Explorer菜单并选择**Git Clone...**.

克隆对话框将显示

**图2.7。克隆对话框**



URL网址：输入库URL地址，您将克隆的来源. 你可以点击**Browse**浏览目录

Directory目录输入您的本地目录，您将克隆到. 你可以点击**Browse**浏览目录

如果你检查Load Putty Key（加载Putty密钥）复选框中，克隆将自动加载Putty密钥。你可以点击**…**浏览Putty密钥文件.

克隆将自动检测当前HEAD 到工作空间。

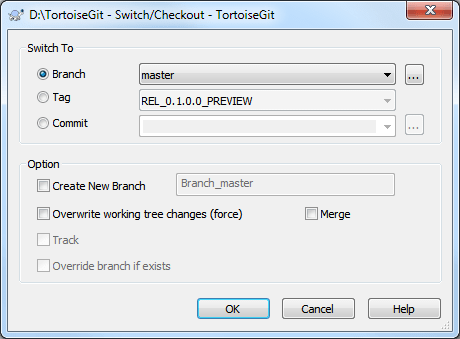
Git克隆支持HTTP、GIT和SSH协议。[称为“网络设置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-network)演示如何选择SSH客户端。OpenSSH，Plink或 TortoiseGitLink。

## 检出工作树（切换到提交）

Switch/Checkout（切换/签出）对话框可用于将特定版本检出到工作树（即，更新所有文件以匹配其所选版本的状态）。通常，特定版本将由(本地)分支表示，该分支被设置为当前分支(参见。称为“分支/标记”的部分和称为“存储库和分支”的部分)。

在windows资源管理器中选择一个git仓库目录，右键单击弹出上下文菜单，选择命令tortoiseGit→Switch/Checkout.，弹出如下对话框：

**Figure 2.8. The Switch/Checkout dialog**



如果在“创建新分支”(Create New Branch)中输入分支名称，则将创建新的分支。此外，新分支将被设置为当前分支(HEAD)。

你可以点击，浏览引用/分支/日志以选择要检出的分支。

选中覆盖工作树更改(FORCE)将用所选版本覆盖工作树中未提交的更改。

选择远程分支时，可以选中“Track(跟踪)”以跟踪远程分支。当您打开“push(推)”、“pull(拉)”或“sync(同步)”对话框时，将自动预选远程分支。

您可以在名为“git-checkout(1)”的部分找到更多信息

**重要的**

如果您结帐/切换到tag(标签)或commit(提交)您应该创建一个新的分支。否则，你将工作在“无分支”（分离的头状态），即，没有当前分支，参见。[被称为“分离头”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-checkout(1)__detached_head)）通过在这个版本中创建一个分支并切换到它，可以很容易地修复这个问题。

### 出口

有时您可能希望创建一个没有任何.git目录的本地副本，例如创建源文件的压缩tarball。阅读“导出Git工作树”一节，了解如何做到这一点.

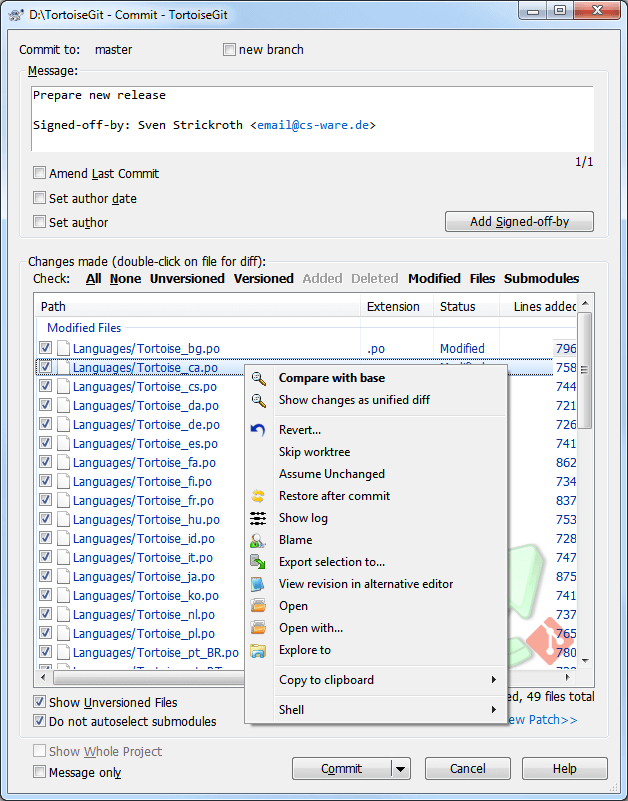
## 将更改提交到存储库

### 存储对工作树所做的更改称为提交更改。您可以先使用TortoiseGit→Check进行修改，以查看哪些文件在本地发生了更改。

### “提交”对话框。

### 如果没有冲突，您就可以提交更改了。选择要提交的任何文件和/或文件夹，然后选择TortoiseGit→Commit.提交对话框

**图2.9。提交对话框**



“提交”对话框将显示每个更改的文件，包括添加的、删除的和未版本化的文件。如果不希望提交已更改的文件，只需取消选中该文件即可。如果要包含未版本化文件，只需检查该文件以将其添加到提交中。

默认提交对话框只列出选择路径及其子目录文件。如果要列出项目的所有文件，只需单击**Whole Project**(整个项目)即可。

**提交对话框中有许多未版本化的文件。**

**如果您认为“提交”对话框显示太多未版本化的文件(例如，编译器生成的或编辑器备份的)，有几种方法可以处理此问题。你可以：**

**使用TortoiseGit→Add to Ignore List将文件添加到.gitignore列表。**

**有关详细信息，请阅读名为“忽略文件和目录”的部分。**

**双击提交对话框中的任何修改文件将启动外部差异工具以显示您的更改。上下文菜单将为您提供更多选项，如截图所示。您也可以将文件从这里拖到另一个应用程序中，例如文本编辑器或IDE。**

**您可以通过单击项目左侧的复选框来选择或取消选择项目。**

**底部窗格中显示的列是可自定义的。如果您右键单击任何列标题，您将看到一个上下文菜单，允许您选择要显示的列。也可以使用拖曳手柄更改列宽，当您将鼠标移到列边界上时会出现拖曳手柄。这些自定义内容将被保留，因此下次您将看到相同的标题。**

**不同项目的颜色代码在称为“状态”的部分中进行了描述。**

### 拖放

您可以将文件从其他地方拖到Commit对话框中，只要工作树是完全相同的。例如，您可能有一个巨大的工作树，其中打开了几个资源管理器窗口，以查看层次结构的远程文件夹。

如果要避免从顶层文件夹提交(通过冗长的文件夹爬行来检查更改)，可以打开一个文件夹的提交对话框，然后从其他窗口中拖入项目以包括在相同的原子提交中。

您可以将驻留在工作树中的未版本化文件拖到Commit对话框中，它们将被Git自动添加。

### 提交只是本地的

请注意，所有提交都是本地的，只会影响本地的工作树。为了与他人共享它们，您需要将它们推送到远程存储库。有关详细信息，请参阅名为“推送”的部分和名为“同步”的部分。

### 更改列表

提交对话框支持更改列表功能，以帮助将相关文件分组在一起。在名为“更改列表”的部分中查找有关此功能的信息。  
  
**从提交列表中排除项目**  
有时，您有经常更改的版本化文件，但您真的不想提交这些文件。有时这表明您的构建过程中存在缺陷-为什么要对这些文件进行版本化？你应该使用模板文件吗？但偶尔也是不可避免的。一个经典的原因是，每次生成时，IDE都会更改项目文件中的时间戳。项目文件必须进行版本控制，因为它包含所有的构建设置，但不需要仅仅因为时间戳更改而提交。  
  
为了在这种尴尬的情况下提供帮助，有一个名为SKIP-WORKTREE的文件的Git标志-然后文件被视为未修改，并且Git也拒绝在合并/拉取时合并这些文件(参见。称为“Skip-worktree bit”的部分)。作为处理这种情况的另一种方法，我们保留了一个名为Ignore-on-Commit的更改列表。添加到此更改列表中的任何文件都将在“提交”对话框中自动取消选中。您仍然可以提交更改，但必须在Commit对话框中手动选择它。  
  
**仅提交部分文件**  
有时，您只想提交对文件所做的部分更改。这种情况通常发生在您正在处理某件事情，但随后需要提交紧急修复时，并且该修复恰好在您正在处理的同一文件中。  
  
右键单击文件并使用上下文菜单→Restore After Commit。这将按原样创建文件的副本。然后你可以编辑文件，例如在TortoiseGitMerge中，并撤销所有你不想提交的更改。保存这些更改后，您可以提交文件。

使用TortoiseGitMerge

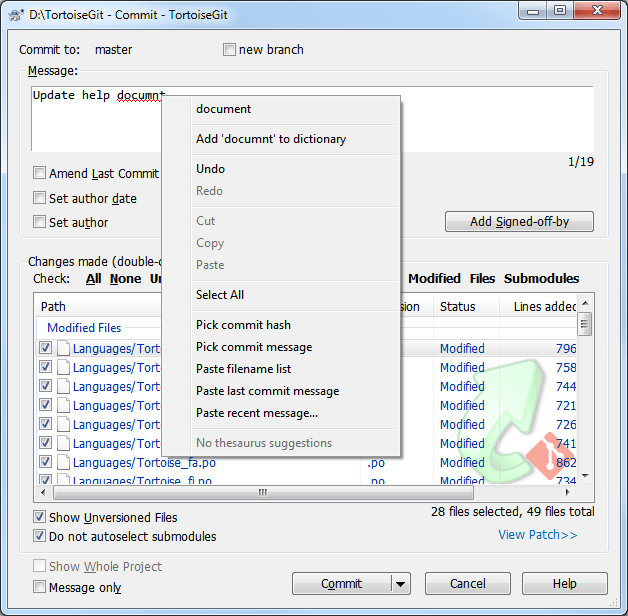
如果使用TortoiseGitMerge编辑文件，则可以按习惯编辑更改，也可以标记要包含的所有更改。在已修改的块上单击鼠标右键，然后使用上下文菜单“→标记此块”以包括该更改。最后，右键单击并使用上下文菜单→使用左侧文件，除了标记的块，这将反转您的更改(未标记的块)，您不希望它们出现在当前提交中

提交完成后，文件的副本将自动恢复，并且您拥有包含未提交回的所有修改的文件。

### 提交日志消息

请确保输入一条日志消息，该消息描述了您正在提交的更改。这将帮助您在稍后浏览项目日志消息时查看发生了什么以及何时发生了什么情况。消息可以是您喜欢的长度，也可以是简短的；许多项目都有关于应该包含的内容、要使用的语言，有时甚至是严格的格式的指导原则。  
  
您可以使用与电子邮件中使用的约定类似的约定，将简单的格式设置应用于日志消息。要对文本应用样式，请使用\*text\*表示粗体，\_text\_表示下划线，^text^表示斜体。

**图2.10。提交对话框拼写检查程序**



TortoiseGit包括一个拼写检查器，可以帮助您正确处理日志消息(参见。称为“拼写检查器”的部分)。这将突出显示任何拼写错误的单词。使用上下文菜单访问建议的更正。当然，它并不知道你做的每个技术术语，所以拼写正确的单词有时会显示为错误。不过别担心。您可以使用上下文菜单将它们添加到您的个人词典中。

日志消息窗口还包括文件名和函数自动完成功能。这使用正则表达式从您提交的(文本)文件中提取类名和函数名，以及文件名本身。如果您键入的单词与列表中的任何内容匹配(键入至少3个字符，或按Ctrl+空格键)，则会出现一个下拉列表，允许您选择全名。TortoiseGit提供的正则表达式保存在TortoiseGit安装bin文件夹中。您还可以定义自己的正则表达式并将其存储在%AppData%\TortoiseGit\autolist.txt中。当然，当您更新TortoiseGit安装时，您的私有自动列表不会被覆盖。如果您不熟悉正则表达式，请查看https：/en.wikipedia.org/wiki/Regular\_expression的简介，以及http：/www.Regular-expression/的在线文档和教程。

让正则表达式恰到好处是很棘手的，所以为了帮助你找出合适的表达式，有一个测试对话框，允许你输入一个表达式，然后输入文件名来测试它。使用命令TortoiseGitProc.exe/command：autoexttest从命令提示符启动它。

您可以重复使用以前输入的日志消息。只需使用命令上下文菜单→Paste Recent Messages即可查看您为此工作树输入的最近几条消息的列表。可以在TortoiseGit设置对话框中自定义存储的消息数量。

日志消息窗口还包括提交消息片段工具。一旦键入代码段快捷方式，这些代码段将显示在“自动完成”下拉列表中，然后在“自动完成”下拉列表中选择代码段，然后插入代码段的全文。TortoiseGit提供的代码片段保存在TortoiseGit安装bin文件夹中。您还可以定义自己的代码段并将其存储在%AppData%\TortoiseGit\snippet.txt中。#是注释字符。使用转义序列\t、\r、\n和\。

您可以通过单击Add Signed-Off-by将您的姓名和电子邮件地址添加到日志消息的末尾。

您可以从TortoiseGit设置的Saved Data页面中清除所有存储的Commit消息，也可以使用Delete键从Recent Messages对话框中清除单个消息。

如果要在日志消息中包括选中的路径，可以使用编辑控件中的命令上下文菜单→Paste FileName List。

将路径插入日志消息的另一种方法是简单地将文件从文件列表拖动到编辑控件上。

### 使用键盘

您可以访问OK按下键盘按钮**Ctrl+return**.

### 与bug跟踪工具的集成

如果已激活bug跟踪系统，则可以在Bug-ID/Issue-Nr：文本框中设置一个或多个问题。多个问题应以逗号分隔。或者，如果您正在使用基于regex的bug跟踪支持，只需添加您的问题引用作为日志消息的一部分。在名为“与Bug Tracking Systems/问题跟踪器的集成”一节中了解更多信息。

### 调整消息文本框的大小

将鼠标移动到“消息”组框和“更改”组框之间的间隙，然后拖动分隔符。

### 致力于一个新的分支

如果您想提交一个新的分支（基于当前分支），您可以检查**new branch**复选框并在显示的文本框中输入分支名称。

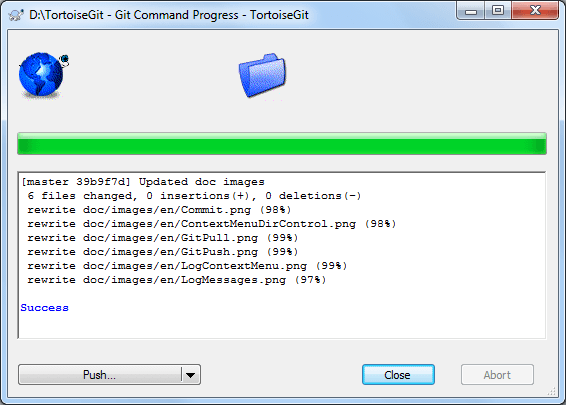
### 连续多次提交并直接推送更改

主按钮Commit(提交)有一个下拉菜单。有**ReCommit（**重新提交）和**Commit & push**（提交&推送）的选项。“**ReCommit**重新提交”选项将提交您的更改并使“提交”对话框保持打开状态，以便您可以继续提交。最后一个选项commit&push（提交&推送）将提交您的更改并立即推送您的更改。如果没有为当前活动分支配置远程跟踪分支，则推送对话框(参见。被称为“推”的部分)被打开。

### 提交进度

压后**Commit提交**出现一个对话框，显示提交的进度。

**图2.11。进程中显示提交的进度对话框**



在左下角，有一个菜单按钮，它提供了进一步步骤的快捷方式，例如重新提交(重置提交对话框并允许您继续提交)或按下以将提交推送到远程存储库。

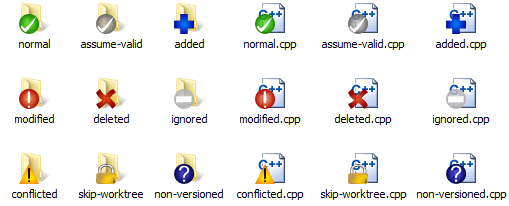
您可以在名为“git-commit(1)”的小节中找到更多信息。

## 获取状态信息

当您在工作树上工作时，通常需要知道哪些文件已经更改/添加/删除或重命名，或者甚至哪些文件被其他人更改和提交。

### 图标覆盖

**图2.12。Explorer显示图标叠加**



现在您已经检查了一个工作树，您可以在Windows资源管理器中看到具有更改图标的文件。这就是TortoiseGit如此受欢迎的原因之一。TurtoSeGIT向每个文件图标添加一个所谓的叠加图标，该图标与原始文件图标重叠。取决于文件的Git状态，覆盖图标不同。

https://tortoisegit.org/docs/tortoisegit/images/InGitIcon.png

新检出的工作树具有绿色复选标记作为覆盖。这意味着Git状态正常。

https://tortoisegit.org/docs/tortoisegit/images/ModifiedIcon.png

一旦开始编辑文件，状态将更改为“已修改”，然后图标覆盖将更改为红色感叹号。这样，您就可以很容易地看到自从上次更新工作树以来哪些文件发生了更改，并且需要提交。

https://tortoisegit.org/docs/tortoisegit/images/ConflictIcon.png

如果在更新期间发生冲突，则图标将变为黄色感叹号。

https://tortoisegit.org/docs/tortoisegit/images/ReadOnlyIcon.png

阶段性的。如果您使用“git update-index”来告诉git这个文件将被提交，那么git会将该文件暂存起来。

https://tortoisegit.org/docs/tortoisegit/images/DeletedIcon.png

此图标显示当前文件夹中的某些文件或文件夹已计划从版本控制中删除，或者文件夹中缺少受版本控制的文件。

https://tortoisegit.org/docs/tortoisegit/images/AddedIcon.png

加号表示已计划将文件或文件夹添加到版本控制中。

https://tortoisegit.org/docs/tortoisegit/images/IgnoredIcon.png

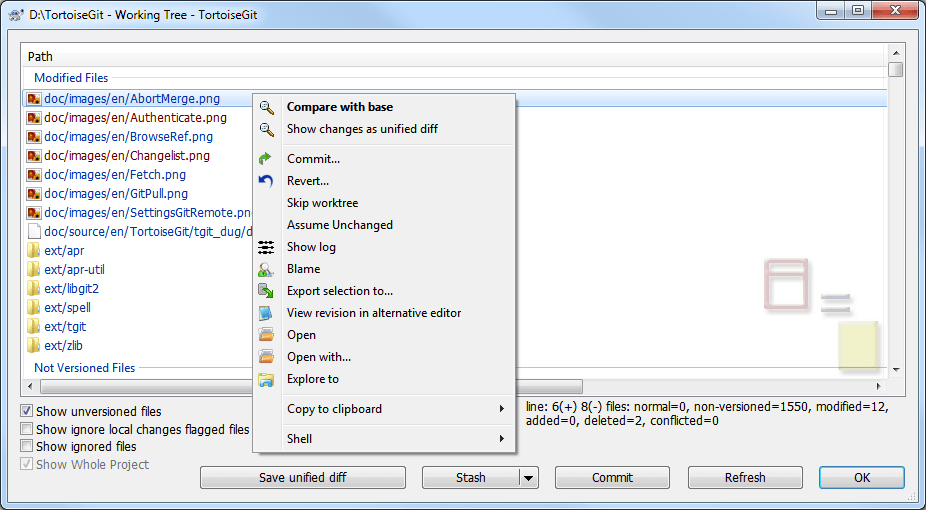
条形标志告诉您，出于版本控制目的，忽略某个文件或文件夹。此叠加是可选的。

https://tortoisegit.org/docs/tortoisegit/images/UnversionedIcon.png

此图标显示不受版本控制但未被忽略的文件和文件夹。此叠加是可选的。  
  
事实上，您可能会发现并非所有这些图标都在您的系统上使用。这是因为Windows允许的覆盖数非常有限，如果您还使用旧版本的TortoiseCVS或带有覆盖处理程序的工具(如SkyDrive、Dropbox或GoogleDrive)，则没有足够的覆盖插槽可用。TortoiseGit试图成为“好公民(Good Citizen(TM)”，并限制覆盖的使用，让其他应用程序也有机会。  
  
如果您对覆盖有问题，请查看在线常见问题解答。  
  
有关图标覆盖如何与Git状态相对应的描述以及其他技术细节，请阅读“图标覆盖”一节。

### 状态

**图2.13修改检查**



了解您更改了哪些文件以及其他人更改和提交了哪些文件通常是非常有用的。这就是命令tortoiseGit→检查修改的地方.会派上用场的。此对话框将显示工作树中以任何方式更改的每个文件，以及您可能拥有的任何未版本化文件。

该对话框使用颜色编码来突出显示状态。

**蓝色。**

**本地修改的项目。**

**紫色。**

**添加选项。已添加历史记录的项目在“文本状态”列中有一个+号，并且工具提示显示该项目是从哪里复制的。**

**暗红。**

**已删除或缺少的项目。**

**绿色。**

**在本地和存储库中修改的项目。更新时将合并更改。这些可能会在更新时产生冲突。**

**鲜红。**

**在本地修改并在存储库中删除的项，或在存储库中修改并在本地删除的项。这些将在更新时产生冲突。**

**黑色。**

**未更改和未版本化的项。**

这是默认的配色方案，但您可以使用设置对话框自定义这些颜色。有关详细信息，请阅读名为“TortoiseGit Colour Settings”的部分。

从对话框的上下文菜单中，您可以显示更改的差异。检查使用上下文菜单→与基础比较所做的本地更改。使用上下文菜单→查看其他人在存储库中所做的更改，将差异显示为统一差异。

也可以还原单个文件中的更改。如果您意外删除了一个文件，它将显示为丢失，您可以使用还原来恢复它。

未版本化和被忽略的文件可以从这里使用上下文菜单→删除发送到回收站。如果要永久删除文件(绕过回收站)，请按住Shift键，同时单击“删除”。

如果要详细检查文件，可以将其从此处拖到另一个应用程序中，如文本编辑器或IDE。

这些列是可定制的。如果您右键单击任何列标题，您将看到一个上下文菜单，允许您选择要显示的列。也可以使用拖曳手柄更改列宽，当您将鼠标移到列边界上时会出现拖曳手柄。这些自定义内容将被保留，因此下次您将看到相同的标题。

如果您同时处理几个不相关的任务，还可以将文件分组到更改列表中。有关更多信息，请阅读名为“更改列表”的部分。

在对话框的底部，有几个选项可以选择要显示的条目(例如忽略的或未跟踪的/未版本化的文件)。也可以在此处查看标记为“假设有效”或“跳过工作树”的所有文件(使用显示Ingore本地更改标记的文件)。重置这些标志(也可以使用资源管理器中Git选项卡上的文件属性编辑此标志)。

保存统一的diff：生成包含未提交的更改的统一补丁，并使用配置的diff查看器打开它。如果您想要将您的更改传递给其他人，请考虑名为“创建和应用修补程序和拉取请求”一节。

stash：是用于快速访问Git stash的下拉菜单(参见。称为“Stash Changes”的部分)。此功能允许您将未提交的更改放在一边，以便处理不相关的问题，并在以后轻松获取更改。

COMMIT：打开COMMIT对话框(参见。名为“将更改提交到存储库”的部分)，其文件夹/文件与当前对话框打开时所在的文件夹/文件完全相同。如果要仅使用FI的子集预先填充“提交”对话框，请选择文件并选择“提交”。在关联菜单中。

### 查看差异

通常你想看看你的文件里面，看看你改变了什么。您可以通过选择已更改的文件并从TortoiseGit的菜单中选择diff来完成此操作。这将启动外部比较查看器，然后将当前文件与上次检出或更新后存储的原始副本(基本版本)进行比较。

### 小贴士

即使不在工作树中，或者当您有多个版本的文件时，仍然可以显示Diffs：

在资源管理器中选择要比较的两个文件(例如使用Ctrl和鼠标)，然后从TortoiseGit的上下文菜单中选择diff。

最后单击的文件(具有焦点的文件，即虚线矩形)将被视为后一个文件。

## 拉取或获取更改

本节讨论如何从另一个存储库获取或拉取(即下载)更改。Pull和Fetch之间的区别是：

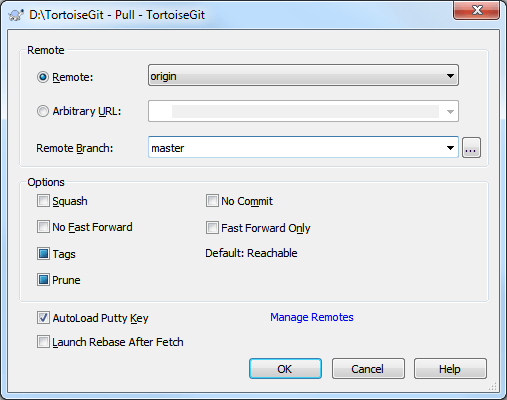
Fetch只是从远程存储库下载对象和Ref，并且通常更新远程跟踪分支。然而，Pull不仅会下载更改，而且还会合并它们-它是FETCH和Merge的组合(参见。称为“合并”的部分)。自动选择已配置的远程跟踪分支。

### 重要的

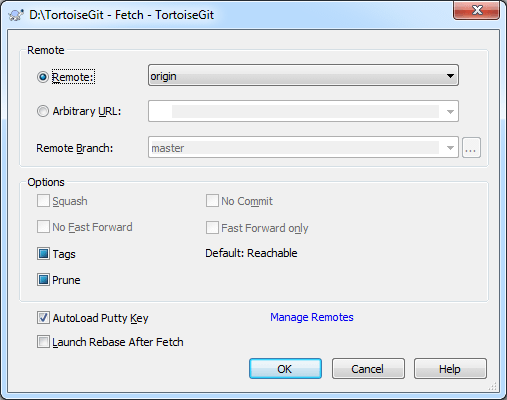
无论何时合并，都有可能在两个分支中都更改了文件，并且这些更改不能自动合并：这称为“冲突”，需要手动解决。有关详细信息，请参阅“解决冲突”一节。

可以使用TortoiseGit→Pull启动Pull/Fetch.。或者TortoiseGit→Fetch.。使用同步对话框也可以获取和拉出更改(参见。名为“Sync”的部分)，但是，您的选项较少，但同步对话框允许您启动其他操作，如推送和查看差异和更改。  
  
将会打开“获取和拉取”对话框。

**图2.14。拉取对话框**



**图2.15。获取对话框**



远程选择已配置的远程存储库(这些可以是使用Manage Remotes标签进行的更改)。您还可以将另一个存储库的URL放入任意URL文本框中，而不是配置的存储库。

如果当前活动分支具有远程跟踪分支集，则会自动选择远程分支和远程存储库。可以使用参考浏览器设置远程跟踪分支(参见。名为“Browse all refs”的部分)或使用推式对话框(参见。

称为“推送”的部分)。

Other：输入其他URL或本地目录。您可以单击。浏览目录。

如果选中Autoload Putty Key复选框，将使用Pageant自动加载已配置的Putty Key。

标签有三种状态(GIT 1.9和更高版本)：选中：下载所有标签和分支(-标签传递到GIT)；未选中：不下载标签(-no-tag被传递到GIT)；第三种状态：使用默认行为(基于remote.<name>.tagopt设置)。标签有三种状态(在GIT 1.9之前)：选中：只下载所有标签，但不下载任何分支(-标签传递给GIT)；未选中：不下载标签(-no-tag被传递给GIT)；第三种状态：使用默认行为(基于remote.<name>.tagopt设置)。

prune有三种状态：true：删除远程上不再存在的远程跟踪分支；false：不删除；第三种状态：使用默认行为(基于remote.<name>.prune或fetch.prune git设置，可以在名为“remote”的部分上进行设置)。

### 小贴士

您可以在附录F的SSH/Putty的提示和技巧中找到关于Putty和使用ssh-key的更多信息。还解释了如何为遥控器同时使用多个帐户。

### 冲突

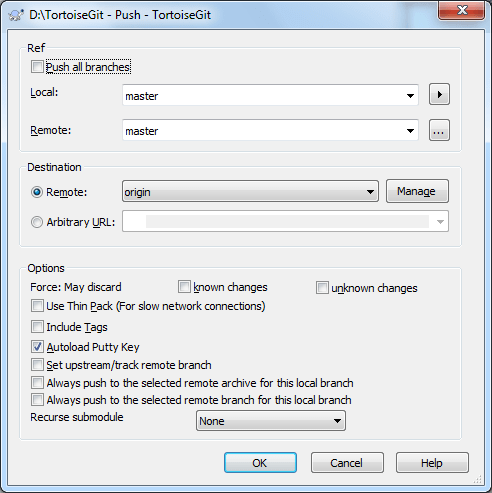
虽然主要的合并工作是由git在拉取时自动完成的，但在樱桃拾取过程中可能会发生冲突(即文件在您当前的分支中被修改，也在您正在拉取的分支中)，请参阅“解决冲突”一节，了解如何解决冲突。  
  
请注意，冲突编辑器中的“远程”/“他们的”指的是您在工作树中选择用于拉取/合并的分支上的“远程”/“他们的”和“本地”/“我的”到您的头版本的更改。

## 推送

本节讨论如何将更改推送(即发送)到另一个存储库。为了执行推送，使用TortoiseGit→推送打开推送对话框。

使用同步对话框也可以推送更改(参见。名为“Sync”的部分)，但是，您可以选择的选项较少，但同步对话框允许您启动其他操作，例如拉取和查看差异和更改。

**图2.16。推送对话框**



### 分支

local：将被推送到其他存储库的源分支。如果当前分支或所选本地分支具有远程跟踪分支集，则会自动选择远程分支和远程存储库。可以使用参考浏览器设置远程跟踪分支(参见。称为“浏览所有引用”的部分)或使用设置上游/跟踪远程分支。在此对话框中，可以使用始终推送到所选远程.中的一个来覆盖此选项.。选项，以便在合并和拉动时自动选择不同的分支。

远程：其他存储库的远程分支。

目的地

远程：选择已配置的远程存储库。

任意URL：远程存储库的URL。

您必须将更改推送到一个空的存储库。将更改推送到具有工作树的存储库可能会导致意外的结果。

选项

强制(可能丢弃已知的更改)这允许远程存储库接受更安全的非快进推送。这可能会导致远程存储库丢失提交；请小心使用它。这可以防止从遥控器上的其他人丢失未知的更改。它检查服务器分支是否指向与远程跟踪分支相同的提交(已知更改)。如果是，将执行强制推送。否则会被拒绝。由于git没有远程跟踪标记，因此不能使用此选项覆盖标记。这传递了git push命令的-force-with-lease选项。

强制(可能丢弃未知更改)这允许远程存储库接受不安全的非快进推送。这可能会导致远程存储库丢失提交；请小心使用它。这不检查任何服务器提交，因此可能会丢失远程上的未知更改。将此选项与“包含标记”一起使用可覆盖标记。这传递了git push命令的传统-force选项。

包括标签也会将标签推送到远程存储库。

自动加载Putty键

### 小贴士

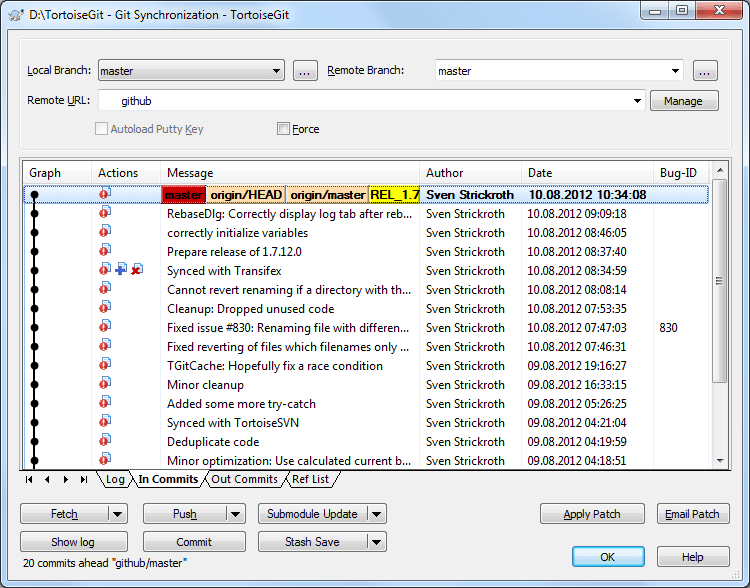
您可以在附录F的SSH/PuTTY的提示和技巧中找到关于PuTTY和使用ssh-key的更多信息。还解释了如何为遥控器同时使用多个帐户。

设置上游/跟踪远程分支：成功推送后，将设置推送的本地分支与其远程跟踪分支之间的跟踪关系。这将自动选择远程分支进行拉/推和合并。  
  
始终推送到此本地分支机构的选定远程归档。  
  
始终推送到此本地分支的选定远程分支。  
  
递归子模块无：不检查。检查：检查远程存储库上是否存在所有子模块的有界提交。如果没有推送任何子模块，则超级项目推送将失败。按需：检查远程存储库上是否存在所有子模块的有界提交。如果子模块还没有被推送，它将尝试推送它们。  
  
您可以在名为“git-push(1)”的部分找到更多信息。

## 同步

“同步对话框”在一个对话框中为与远程存储库相关的所有操作提供了一个界面。这包括推，拉，取，远程更新，子模块更新，发送补丁.。然而，同步对话框作为有关对话框提供的选项较少(参见。称为“Pull and fetch change”的部分和称为“Push”的部分)。  
  
可以使用“同步.”打开“同步”对话框。。  
  
将显示同步对话框。

**图2.17。同步对话框**



### 分支

本地分支：将向其他存储库推送/从其他存储库拉取的源分支。如果当前分支或所选本地分支具有远程跟踪分支集，则会自动选择远程分支和远程存储库。可以使用参考浏览器设置远程跟踪分支(参见。名为“Browse all refs”的部分)或使用推式对话框(参见。称为“推送”的部分)。  
  
远程分支：远程存储库的远程分支。

### 目的地

远程URL：选择远程存储库或输入远程存储库URL。  
  
管理添加新的远程名称

### 选项

强制覆盖现有分支(可能放弃更改)。  
  
推拉时自动加载 putty key

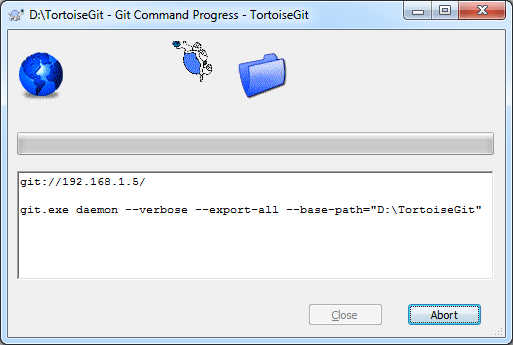
### 小贴士

当您在按住Shift键的同时选择fetch, pull, fetch, stash changes或submodule update/sync（子模块更新/同步）时，您可以打开单独的对话框Boxen，它允许您为特定操作控制更多参数。

## 守护进程

有时，您希望快速**将本地存储库共享给其他人，而无需推送到远程git存储库**。这时你需要使用**TortoiseGit** → **Daemon...**

**图2.18。运行守护进程对话框**



此命令运行在端口9418(git：//hostname/)为Git协议提供服务的Git守护程序。

### 注意安全

导出选定的存储库以进行读/写访问，而无需进一步的身份验证。

### 重要的

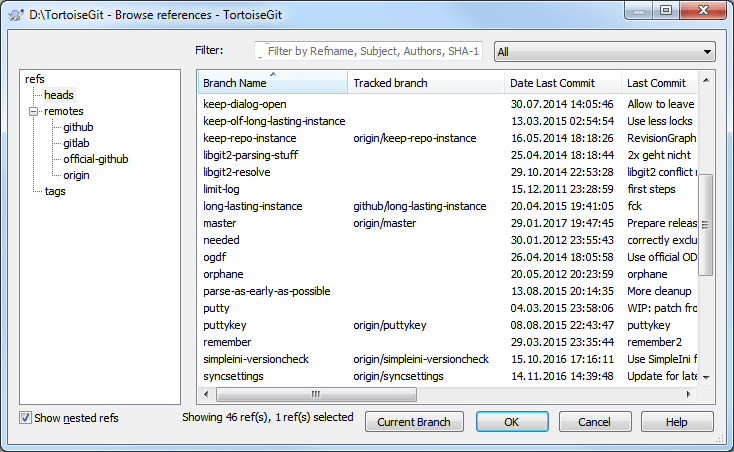
您的主机可能只能在本地网络中访问，您可能需要调整防火墙。

您可以在名为“git-daemon(1)”的小节中找到更多信息。

## 浏览所有参照

本节讨论Reference Browser，它允许您查看和使用所有参照(标记、分支、远程分支、stash等)。可以使用**TortoiseGit** → **Browse Reference...**.

**图2.19。浏览引用对话框**



左侧面板显示树中的引用“类型”，如tags、heads (本地分支)等。

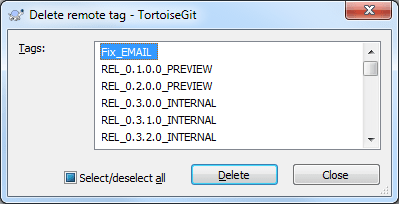
右侧面板显示选定类型的所有参照(如果未使用“显示嵌套参照”(**Show nested refs**)以递归方式禁用)，包括最新的提交、描述和(对于本地分支)其远程跟踪分支。

顶部有一个编辑栏，可用于过滤右侧面板中显示的Ref。搜索语法类似于“日志”对话框中提供的语法(参见。称为“过滤日志消息”的部分)。

在两个面板上都有一个功能强大的上下文菜单，该菜单提供了更多选项，例如删除/重命名Refs、配置远程跟踪分支(用于本地分支)和删除远程标记(选择远程时在左侧面板上)。如果恰好选择了两个Ref，则可以比较它们或打开两个分支上的所有提交的日志(显示分支1.分支2的日志)或仅在两个分支之一上(显示分支1.分支2的日志)。

要删除远程标记，请使用左侧遥控器上的上下文菜单，然后选择删除远程标记.。然后将出现下面的对话框。在那里你可以一次删除多个远程标签。

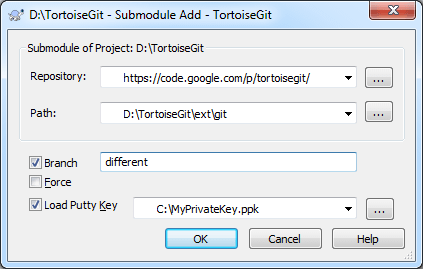
**图2.20。删除远程标签对话框**



## 子模块

当您想要将外来存储库嵌入到工作树/git存储库中时，这称为子模块。在这里，使用**TortoiseGit** → **Submodules Add**选项，可以将外部存储库嵌入到源树的专用子目录中。选择此选项时，会弹出一个对话框：

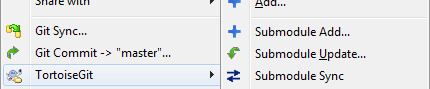
**图2.21。添加子模块对话框**



在这里，您可以输入要嵌入到目录路径中的存储库的位置/URL。路径可以作为活动源树中的相对路径输入，但也可以是绝对路径(指向活动源树)。文件夹应为空或不存在。如果不想集成存储库的Head，可以输入不同的Branch。通过按OK，输入的Repository被克隆并集成到当前源代码树中。

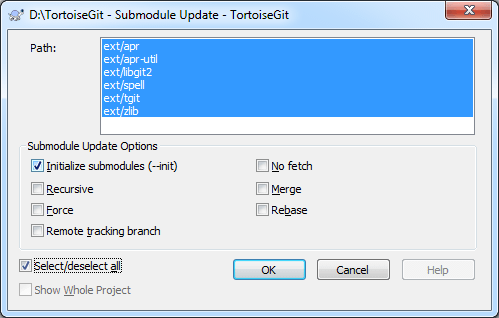
如果工作树包含子模块，则有两个新的上下文菜单项可用：

**图2.22。子模块菜单条目**



**子模块更新**：

**图2.23。更新子模块对话框**

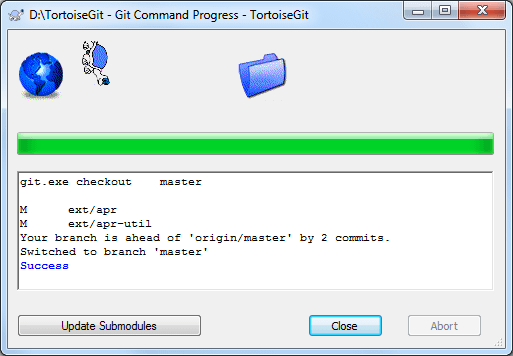


初始化子模块和/或更新已注册的子模块，即克隆缺少的子模块并签出包含存储库的索引中指定的提交。

子模块同步：将子模块的远程URL配置设置同步到.gitmodules中指定的值。当子模块URL发生上游更改，并且您需要相应地更新本地存储库时，这是非常有用的。

此外，如果工作树包含子模块，则名为“Checking Out A Working Tree(Switch To Commit)”的部分和名为“Reset”的部分包含用于更新子模块的按钮：

**图2.24。在进度对话框中更新子模块的按钮**



你可以找到更多的信息。[被称为“Git子模块（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-submodule(1)).

## 更改列表

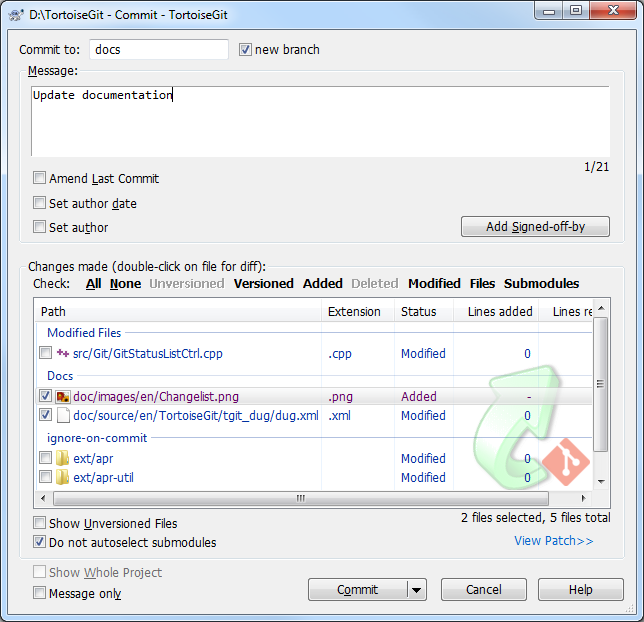
在理想的世界中，您一次只能处理一件事情，并且您的工作树只包含一组逻辑更改。好的，回到现实。经常发生的情况是，您必须同时处理几个不相关的任务，当您查看Commit对话框时，所有更改都混合在一起。changelist功能帮助您将文件分组在一起，使您更容易看到您正在做的事情。当然，这只能在更改不重叠的情况下工作。如果两个不同的任务影响同一个文件，则没有办法将更改分开。

### 小贴士

Git中有一个类似的功能叫做staging(称为“stash changes”的部分)。  
  
一般来说，您应该考虑创建一个新的分支，因为Git提交只是本地的，您不是在扰乱每个人的存储库，而只是您自己的存储库。

您可以在几个地方看到更改列表，但最重要的是Commit对话框和Check-for-Modified对话框。在您处理了几个特性和许多文件之后，让我们从Check-for-Modified对话框开始。第一次打开对话框时，所有更改的文件一起列出。假设您现在想要组织事物并根据功能对这些文件进行分组。  
  
选择一个或多个文件，然后使用上下文菜单→Move to Changelist将项目添加到更改列表。最初不会有更改列表，所以第一次执行此操作时，您将创建一个新的更改列表。为其命名，以说明您使用它的目的，然后单击“确定”。对话框现在将更改为显示项目组。

**图2.25。与变更者的“提交”对话框**



在Commit对话框中，您可以看到这些相同的文件，按Changelist分组。除了提供分组的即时可视指示外，您还可以使用组标题来选择要提交的文件。

单击组标题以选择所有条目，然后选中所选条目之一以选中所有条目。

TortoiseGit保留了一个更改列表名称供自己使用，即Ignore-on-commit。这是用来标记版本化文件的，即使它们有本地更改，您也几乎不想提交这些文件。此功能在名为“从提交列表中排除项目”一节中进行了描述。

当您提交属于更改列表的文件时，通常您会期望不再需要更改列表成员资格。因此，默认情况下，文件在提交时会自动从更改列表中删除。

### 小贴士

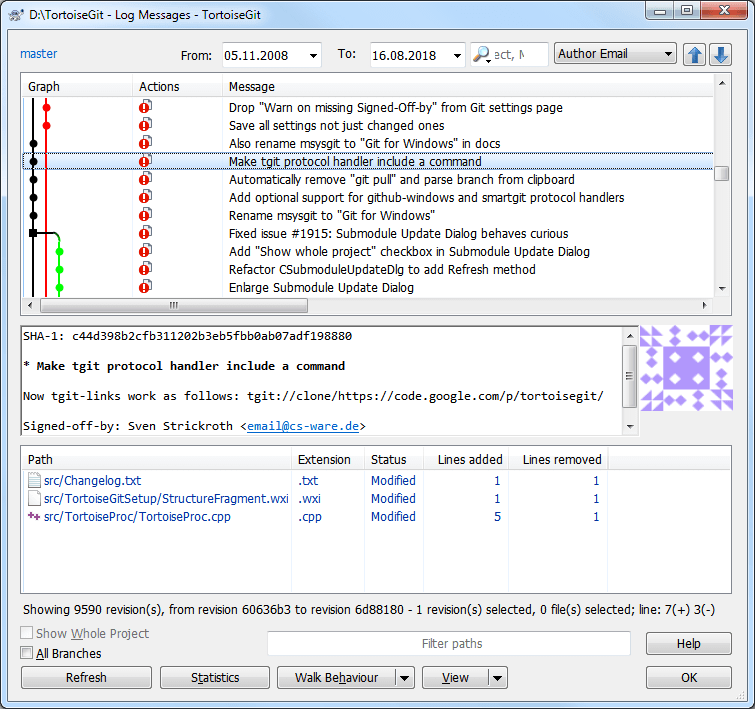
Changelists纯粹是一个本地TortoiseGit客户端功能。创建和删除更改列表不会影响存储库，也不会影响其他任何人的工作副本。它们只是一种方便的方式来组织您的文件。

## 日志对话框

对于您所做和提交的每个更改，您应该提供该更改的日志消息。通过这种方式，您可以在以后找到您所做的更改以及原因，并且您可以为您的开发过程提供详细的日志。  
  
Log Dialog(日志对话框)检索所有这些日志消息并将其显示给您。显示分为3个窗格。  
  
顶部窗格显示已提交文件/文件夹更改的修订列表。此摘要包括日期和时间、提交修订的人员以及日志消息的开始。  
  
粗体显示的行表示头部提交，条目“Working Tree Changes”是表示工作树的当前(未提交)状态的虚拟条目。  
  
中间窗格显示所选修订版本的完整日志消息。  
  
底部窗格显示作为所选修订的一部分更改的所有文件和文件夹的列表。  
  
但它所做的远不止这些-它提供了上下文菜单命令，您可以使用这些命令来获取有关项目历史记录的更多信息。

### 调用修订日志对话框

**Figure 2.26. The Revision Log Dialog**



您可以在几个位置显示“日志”对话框：

从资源管理器上下文菜单中使用TortoiseGit → Show log...。

从各种TortoiseGit对话框中，您可以选择提交(通常使用 ... 按钮)。

从各种TortoiseGit对话框中，使用上下文菜单显示提交条目或文件。

### Revision Log Actions

顶部窗格有一个Actions包含图标的列，总结在该修订中所做的内容。有四个不同的图标，每个图标都显示在它自己的列中。

https://tortoisegit.org/docs/tortoisegit/images/ActionModifiedIcon.png

如果修订修改了文件或目录，则修改后的图标将显示在第一列中。

https://tortoisegit.org/docs/tortoisegit/images/ActionAddedIcon.png

如果修订添加了文件或目录，则添加的图标将显示在第二列中。

https://tortoisegit.org/docs/tortoisegit/images/ActionDeletedIcon.png

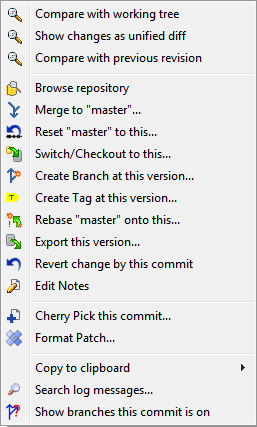
如果修订删除了文件或目录，则“已删除”图标将显示在第三列中。

https://tortoisegit.org/docs/tortoisegit/images/ActionReplacedIcon.png

如果修订替换(重命名)文件，则被替换的图标将显示在第四列中。

### 获取其他信息

**Figure 2.27. 带有上下文菜单的“修订日志”对话框顶部窗格**



“日志”对话框的顶部窗格有一个上下文菜单，允许您访问更多信息。您可以按Shift键查看包含更多选项的扩展菜单。

**与工作树比较。**

将所选修订与您的工作树进行比较。默认的diff-tool是TortoiseGitMerge，它随TortoiseGit一起提供。如果日志对话框是针对文件夹的，这将显示已更改文件的列表，并允许您单独查看对每个文件所做的更改。

**将更改显示为统一差异。**

以Unified-diff文件(GNU补丁格式)的形式查看在所选修订版本中所做的更改。这只显示了几行上下文的差异。它比可视文件比较更难阅读，但将以紧凑的格式一起显示所有文件更改。

**与以前的版本比较。**

将所选修订版本与以前的修订版本进行比较。这与工作树进行比较的方式类似。对于文件夹，此选项将首先显示“已更改的文件”对话框，允许您选择要比较的文件。

**浏览存储库。**

打开存储库浏览器以检查存储库中选定的文件或文件夹，因为它位于选定的修订版(参见。称为“存储库浏览器”的部分)。

**重置(当前分支)到此。**

将头部重置为选定的提交(参见。称为“重置”的部分)。

**切换/检出到修订。**

将工作树更新为选定的修订版本。如果您想让工作树反映过去的某个时间，或者如果已经对存储库进行了进一步的提交，并且希望一步地更新您的工作树，那么这一点很有用。

**从修订创建分支。**

基于所选修订创建分支(参见。称为“分支/标记”的部分)。

**从修订创建标记。**

在选定的修订版本上创建标记(参见。称为“分支/标记”的部分)。

**Rebase(当前分支)到此。**

在所选提交的顶部重新设置当前分支的基准(参见。称为“Rebase”的部分)。

**导出此版本.。**

将所选修订导出到存档文件(如zip)。这将弹出一个对话框，供您确认修订，并选择导出位置(参见。名为“导出Git工作树”的部分)。

**通过此提交还原更改。**

还原在所选修订中所做的更改。所有更改都将集成到您的工作树中。您可以选择立即提交或编辑并稍后提交。要放弃还原的更改，请执行硬重置。

**编辑备注。**

编辑所选提交的备注。

**Cherry Pick this commit**。

Cherry Pick这个提交在HEAD之前(参看。称为“[Cherry picking”](https://tortoisegit.org/docs/tortoisegit/tgit-dug-cherrypick.html)”的部分)。

**平分起点**。

开始一分为二。通过二进制搜索查找引入bug的更改(参见。称为“平分”的部分)。

**格式修补程序.。**

从此提交创建修补程序。

**将SHA-1复制到剪贴板。**

将所选修订版本的提交哈希复制到剪贴板。

**复制到剪贴板。**

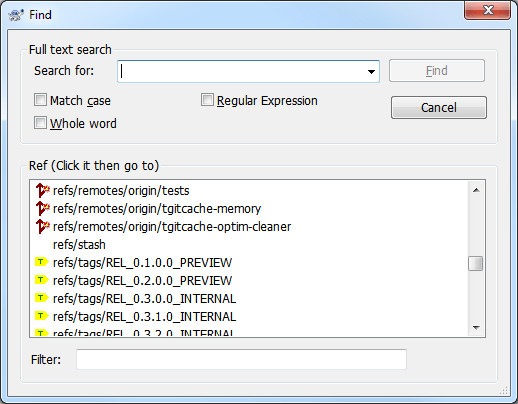
将所选修订的日志详细信息复制到剪贴板。这将复制每个修订版的修订号、作者、日期、日志消息和已更改项目的列表。

**将日志消息复制到剪贴板。**

将所选修订版本的日志消息复制到剪贴板。

搜索日志消息.

**Figure 2.28. 搜索日志信息对话框**



搜索日志消息以查找您输入的文本。这将搜索您输入的日志消息以及Git创建的操作摘要(显示在底部窗格中)。搜索不区分大小写。

### 小贴士

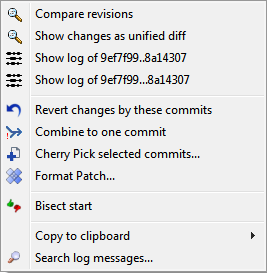
这允许您轻松搜索Refs(标记和分支)。

如果单击Find时按Shift键，则可以导航到Commit而不选择它。

**显示此提交所在的分支**

显示已选择的commit所属的分支。它同时显示本地和远程分支。

**图2.29。用于选择的2个修订的顶部窗格菜单**



如果一次选择两个修订(使用通常的Ctrl修改器)，上下文菜单会发生变化，并提供更少的选项：

比较修订。

使用视觉差异工具比较两个选定的修订。默认的diff-tool是TortoiseGitMerge，它随TortoiseGit一起提供。

将差异显示为统一差异。

以Unified-diff文件的形式查看两个选定修订之间的差异。这适用于文件和文件夹。

通过这些提交还原更改。

还原在所选修订中所做的更改。所有更改都将集成到您的工作树中。您可以选择立即提交或编辑并稍后提交。要放弃还原的更改，请执行硬重置。

合并为一次提交。

将连续提交合并为一个提交。

Cherry Pick Selected提交。

Cherry Pick Selected在当前头顶上的提交(参见。称为“樱桃采摘”的部分)。

格式修补程序.。

在两个选定的提交之间创建修补程序。

将SHA-1复制到剪贴板。

将所选修订的提交哈希复制到剪贴板，由CRLF分隔。

复制到剪贴板。

如上所述将日志消息复制到剪贴板。

将日志消息复制到剪贴板。

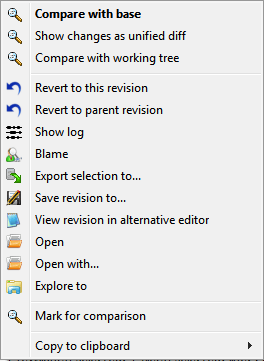
将所选修订的日志消息复制到剪贴板。这将复制每个修订版的日志消息。这有助于编写发行说明。

搜索日志消息.。

如上所述搜索日志消息。

如果选择两个或多个修订(使用通常的Ctrl或Shift修饰符)，则可以将SELECT COMMITES组合为一个COMMIT。而樱桃摘下这些提交到当前的分支。

**图2.30。带有菜单的日志对话框底部窗格**



Log(日志)对话框的底部窗格也有一个上下文菜单，允许您。  
  
与基础比较。  
将所选文件与基本版本进行比较。  
  
显示为统一差异。  
以统一的diff格式显示文件更改。此上下文菜单仅适用于显示为“已修改”的文件。  
  
与工作树比较。  
将所选文件与工作树进行比较。  
  
还原对此修订的更改。  
将所选文件还原为此修订版的SATTE。  
  
还原对父版本的更改。  
将所选文件还原到此修订之前的状态。  
  
显示日志。  
显示所选单个文件的修订日志。  
  
Blame...  
打开“Blame”对话框，允许您责备选定的修订(参见。名为“谁更改了哪一行？”的部分)。  
  
将修订保存到.。  
将选定的修订保存到文件中，以便您拥有该文件的旧版本。  
  
将所选内容导出到.。  
将所选文件保存到目标目录。与“将修订保存到.”相比。这将保留目录结构。  
  
在替代编辑器中查看修订。  
使用替代编辑器(如notepad2)显示选定的文件，并使用选定的提交。  
  
打开/打开方式.。  
使用该文件类型的默认查看器或您选择的程序打开选定的文件。  
  
浏览到。  
用EXPLOSE打开文件的目录。  
  
将路径复制到剪贴板。  
将路径复制到剪贴板。  
  
将所有信息复制到剪贴板。  
将所有信息复制到剪贴板，包括版本信息。

### 小贴士

你可能会注意到，有时我们指的是变化，而另一些时候指的是差异。有什么关系呢？

### 过滤日志消息。 如果要将日志消息限制为仅显示您感兴趣的那些消息，而不是滚动数百个列表，则可以使用“日志”对话框顶部的筛选器控件。 第一个元素是分支/修订过滤器。单击打开参照浏览器(参见“浏览所有参照”一节)。在那里可以选择单个或多个参照。如果恰好选择了两个引用，则可以选择如何组合它们(特别是同时显示A和B“A B”；显示差异“A.B”或A和B之间的所有提交“A..B”)。此过滤元素还包含一个特殊的上下文菜单。这里提供了“head”、“fetch\_head”、“all”和“all local Branch”的快捷方式。此外，那里还包括最后手动选择的过滤器。 “开始日期”和“结束日期”控件允许您将输出限制在已知日期范围内。搜索框允许您仅显示包含特定短语的邮件。可以在“对话框1”页面的设置对话框中配置From的默认限制(参见。名为“TortoiseGit Dialog Settings”的部分)。 单击搜索图标以选择要搜索的信息，并选择regex模式。通常你只需要一个简单的子串搜索，但是如果你需要更灵活的搜索词，你可以使用正则表达式。如果将鼠标悬停在框上，工具提示将提示如何使用regex函数。要反转整个搜索表达式的结果，请以感叹号(‘！’)开头字符串。您也可以在http://www.regular-expressions.info/上找到在线文档和教程。过滤器的工作方式是检查过滤器字符串是否与日志条目匹配，然后只显示与过滤器字符串匹配的条目。 简单的子串搜索以类似于搜索引擎的方式工作。要搜索的字符串用空格分隔，并且所有字符串必须匹配。您可以使用前导-来指定找不到特定子字符串(该术语的反向匹配)，并且您可以使用！在表达式的开头反转整个表达式的匹配。您可以使用前导+来指定应该包括一个子字符串，即使之前用-排除了。请注意，包含/排除的顺序在这里很重要。可以使用引号将必须包含空格的字符串括起来，如果要搜索文字引号，可以将两个引号一起用作自转义序列。请注意，反斜杠字符不用作转义字符，在简单的子字符串搜索中没有特殊意义。示例将使这一点变得更容易：

Alice Bob -Eve

搜索同时包含Alice和Bob但不包含Eve的字符串。

Alice -Bob +Eve

搜索同时包含Alice但不包含Bob的字符串，或包含Eve的字符串。

-Case +SpecialCase

搜索不包含大小写的字符串，但仍包括包含SpecialCase的字符串。

!Alice Bob

搜索不同时包含Alice和Bob的字符串

!-Alice -Bob

你还记得德摩根定理吗？Not(不是Alice和Not Bob)简化为(Alice Or Bob)。

"Alice and Bob"

搜索文字表达式“Alice and Bob”

""

在文本中的任意位置搜索双引号

"Alice says ""hi"" to Bob"

搜索文字表达式“Alice say”hi“to Bob”。

**您还可以使用查看→隐藏不相关的更改路径来过滤底部窗格中的路径名称相关路径是那些包含用于显示日志的路径。如果获取文件夹的日志，则表示该文件夹或其下的任何内容。对于一个文件，它意味着只有一个文件。如果要灰显不相关的路径，请选中查看→灰色不相关的更改路径取消选中这两个菜单项以完全隐藏不相关的路径。**

**在左下角有复选框“所有分支”和“显示整个项目”。使用这些选项覆盖分支对应。文件/文件夹过滤器，并显示整个存储库的日志。请注意，即使您在单个文件上启动了日志对话框，也会记住存储库的这些设置。**

**单击“显示整个项目”可以显示整个项目历史记录，而不是选择目录或文件。**

**视图+标签→标签视图+标签→本地分支视图+标签→远程分支您可以禁用在日志图形中显示一些引用类型。**

**查看→Gravatar您可以为特定存储库启用/禁用Gravatar。**

**Walk Behavior→First Parent只需跟进第一个父代提交。这将有助于理解整个历史。**

**Walk Behavior→No Merges跳过所有合并点。**

**Walk Behavior→Follow Renames这仅适用于单个文件，用于跟踪重命名。否则，日志列表在当前文件名引入的COMMIT处停止。**

**Walk Behavior→压缩图日志图被简化为包括合并点、带引用的提交以及可能的其他提交。**

**Walk Behavior→仅显示标记的提交日志图被简化为仅包含具有引用的提交。**

**通航。**

**您可以使用右上角的下拉控件选择导航类型(例如，作者电子邮件、父项1、选择历史记录)，然后使用向上和向下绿色按钮在与当前选定的导航类型相匹配的提交中导航。**

**也可以选择向上和向下绿色按钮，也可以使用热键ALT+UP和ALT+DOWN。**

**关于导航类型“Selection History”，TortoiseGit会记住所选提交的历史，以便您可以轻松地导航过去选择的提交。也可以通过按Alt+Left、Alt+Right、Browse Back和Browse Forward来导航它们。鼠标上的后退和前进按钮也可用。**

**如果您也按Shift键，则可以在选择历史记录中导航，而无需选择最后的提交(即，只需滚动到并突出显示它们)。这有助于您浏览提交，然后选择您真正想要选择的提交(例如，您可以将当前选择的提交与之前选择的提交进行比较)。**

**如果要跳转到具有特定散列的提交，可以通过按Ctrl+V或Shift+Insert(到除搜索框之外的任何日志对话框元素)来粘贴剪贴板中的散列。如果它具有有效的提交散列的形式，则日志对话框会尝试跳转到它。**

**统计信息。**

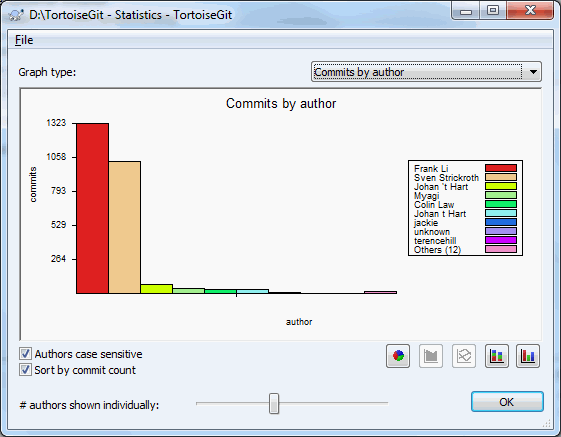
**“统计信息”按钮打开一个框，其中显示了有关“日志”对话框中显示的修订的一些有趣信息。这显示了有多少作者在工作，他们做了多少提交，每周的进展等等。现在你可以一目了然地看到谁工作最努力，谁在偷懒；-)。**

**“统计信息”页。**

**这一页给出了你能想到的所有数字，特别是期间和所涵盖的修订数量，以及一些最小/最大/平均值。**

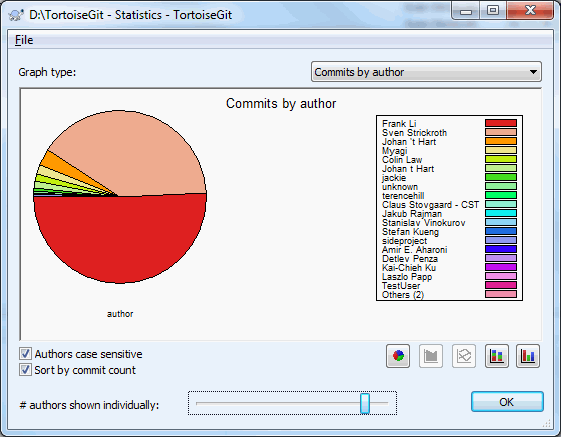
提交(按作者)

**图2.31。作者直方图提交**



此图以简单的直方图、堆叠直方图或饼图的形式向您展示了哪些作者在项目中是活跃的。

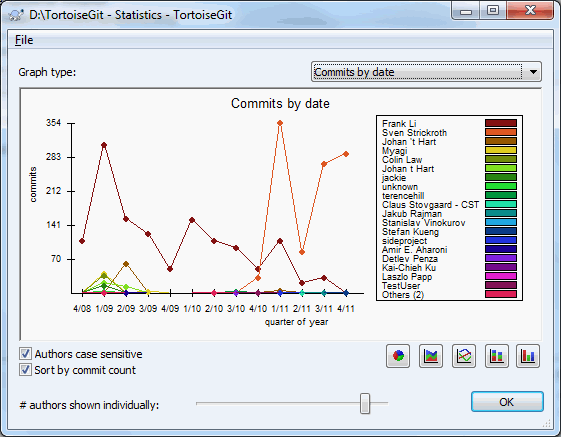
**图2.32。作者饼图提交**



在有几个主要作者和许多次要贡献者的地方，细小片段的数量会使图表更难阅读。底部的滑块允许您设置阈值(占提交总数的百分比)，低于该阈值的任何活动将被分组到其他类别中。

#### 按日期提交

**图2.33。按日期图表提交**



此页面为您提供了项目活动的图形表示形式，包括提交的数量和作者。这给出了一些关于项目什么时候在工作，以及在什么时候谁在工作的想法。

当有几个作者时，你会在图上得到许多线。这里有两种视图：普通视图，其中每个作者的活动相对于基线；堆叠视图，其中每个作者的活动相对于下面的线。后一个选项避免了交叉线，这可以使图形更容易阅读，但不太容易看到一个作者的输出。

默认情况下，分析区分大小写，因此用户PeterEgan和PeteRegan被视为不同的作者。但是，在许多情况下，用户名不区分大小写，有时输入不一致，因此您可能希望DavidMorgan和davidmorgan被视为同一个人。使用“作者不区分大小写”复选框来控制如何处理此问题。

统计对话框还支持.mailmap文件(请参阅“映射作者”一节)。

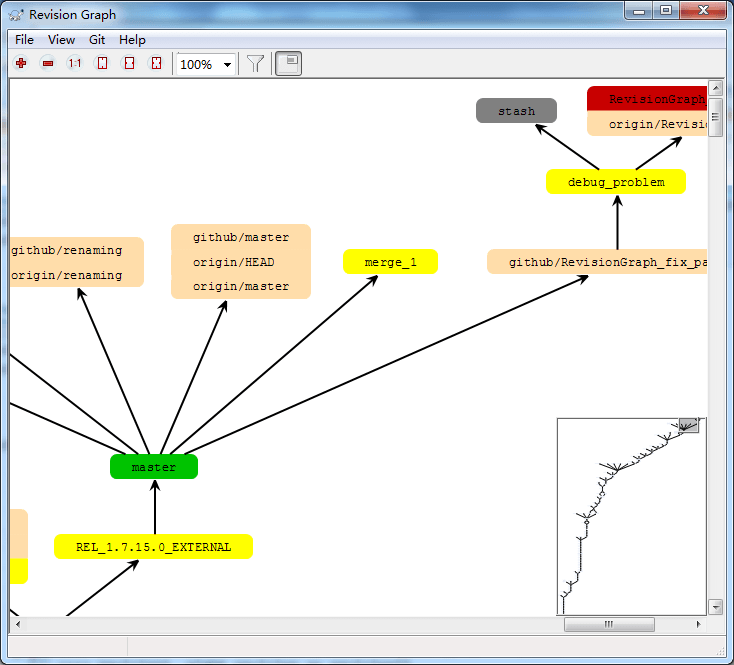
请注意，统计信息与“日志”对话框涵盖的时间段相同。如果这只显示一个修订，那么统计数据不会告诉你很多。

刷新视图。

如果要再次检查存储库中是否有较新的日志消息，只需使用F5刷新视图即可。

## 修正图

**图2.34。修订图**



有时，您需要知道分支和标记是从哪里获取的，查看此类信息的理想方式是以图表或树结构的形式。这就是你需要使用**TortoiseGit** → **Revision Graph...**的时候.

此命令分析修订历史，并尝试创建显示标记、分支和其他参照处的点的有向图。

### Important

为了生成图表，TortoiseGit必须从存储库根目录获取所有日志消息。只是显示有一些参考点的提交。

### 修正图节点

每个修订图节点表示存储库中的一个修订，其中您正在查看的树中发生了更改。可以通过颜色区分不同类型的节点，可以使用**TortoiseGit** → **Settings**进行配置。

请注意，该图仅显示了标记、分支或其他引用项目的点。显示项目的每个修订将为非平凡的案例生成一个非常大的图表。

**使用图表。**

要使导航大型图形更容易，请使用“概述”窗口。这在一个小窗口中显示了整个图表，其中当前显示的部分突出显示。可以拖动突出显示的区域来更改显示的区域。

每当鼠标悬停在修订框上时，修订日期、作者和注释都会显示在提示框中。

如果选择两个修订(使用Ctrl键并单击鼠标左键)，则可以使用关联菜单显示这些修订之间的差异。您可以选择显示分支创建点的差异，但通常需要显示分支端点的差异，也就是头修订处的差异。

您可以Unified-diff文件的形式查看差异，该文件以最小的上下文显示单个文件中的所有差异。如果选择“上下文菜单”(Context Menu)“编辑”(→)“比较修订”(Compare Revision)，将显示已更改文件的列表。双击文件名以获取文件的两个修订版本，并使用视觉差异工具对其进行比较。

如果右键单击修订版本，可以使用上下文菜单→显示日志来查看历史记录。

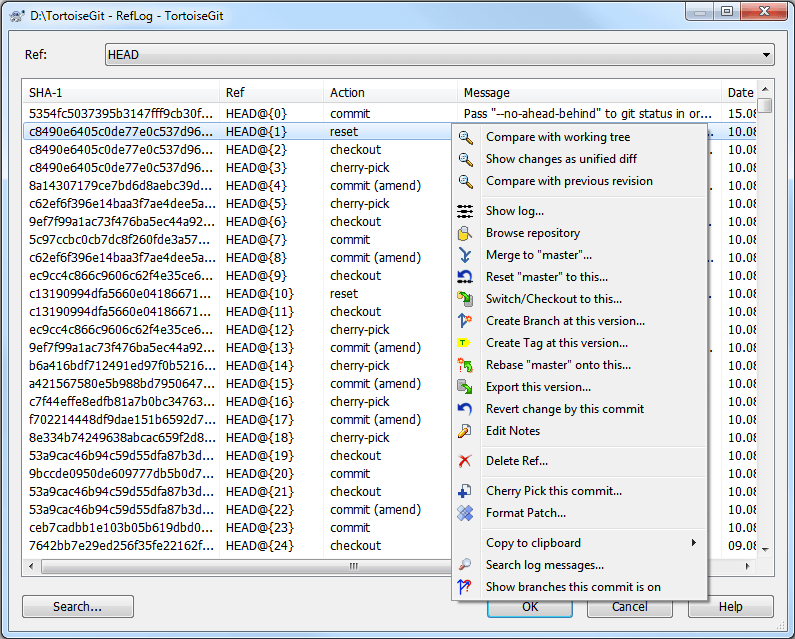
**刷新视图。**

如果要再次检查服务器以获取更新的信息，只需使用F5刷新视图即可。

**参考日志**

引用日志(Reflog)显示引用的历史(即显示它过去指向的提交)。在中可以使用**TortoiseGit** → **RefLog**打开，但是，您必须在右键单击资源管理器中的文件夹时按住Shift键才能看到这一点，因为默认情况下它在扩展的上下文菜单中。

**Figure 2.35. RefLog Dialog**



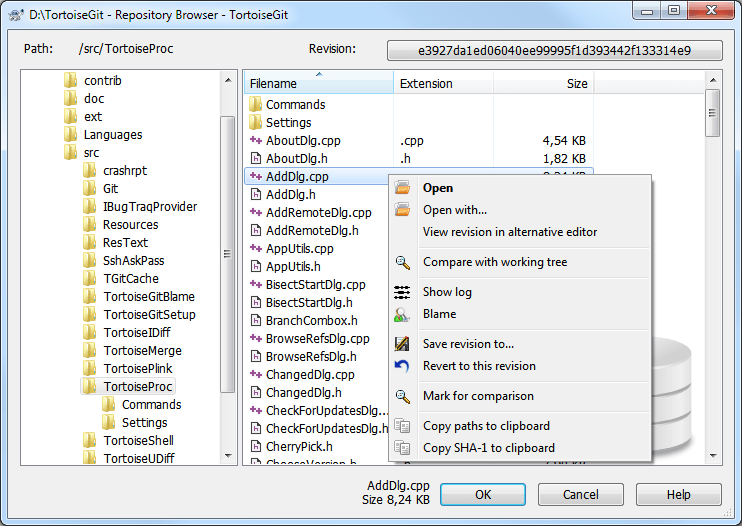
RefLog可用于恢复已删除的提交或标题位置(例如，当您删除一段时间前标题的分支时)。

您可以在名为“git-reflog(1)”的部分找到更多信息。

存储库浏览器

有时，您需要查看存储库的所有内容/文件，而没有工作树(例如，裸露的存储库)，或者您希望在不切换到修订版本的情况下查看所有文件。这就是Repository Browser的用途。您可以使用**TortoiseGit** → **Repo-browser**或从日志对话框打开它(参见。称为“Log Dialog”的部分)使用COMMIT的上下文菜单。

**图2.36。存储库浏览器**



## 存储库浏览器看起来与Windows资源管理器非常相似，不同的是它显示的是特定修订版的存储库内容，而不是您计算机上的文件。在左窗格中可以看到目录树，在右窗格中是所选目录的内容。在“存储库浏览器”(Repository Browser)窗口的顶部，您可以看到存储库中的路径以及要浏览的修订版本。

## 就像Windows资源管理器一样，如果要设置排序顺序，可以单击右窗格中的列标题。和资源管理器一样，在两个窗格中都有可用的上下文菜单。

## 为了获得文件的旧版本，您可以单击文件并选择Save Revision to，但也可以只将一个或多个文件拖到Windows资源管理器窗口中。

## 文件的上下文菜单允许您：

## 使用该文件类型的默认查看器或您选择的程序打开选定的文件。

## 显示该文件的修订日志，以便可以查看其历史记录。

## 将所选修订版本的文件与工作树中的相同文件进行比较。

## 追踪文件，看看谁改了哪行，什么时候改的。

## 将文件的未版本化副本保存到硬盘驱动器或在工作副本中还原此文件(即，将文件保存到工作树中的旧路径)。

## 将地址栏中显示的带有完整路径的文件名复制到剪贴板。

## 文件夹的上下文菜单允许您：

## 显示该文件夹的修订日志。

## 将完整路径复制到剪贴板。

## 您可以像往常一样使用F5刷新视图。这将刷新当前显示的所有内容。

## 观察差异

### 项目开发中最常见的需求之一就是查看发生了什么变化。您可能需要查看同一文件的两个版本之间的差异，或者两个单独文件之间的差异。TortoiseGit提供了一个名为TortoiseGitMerge的内置工具，用于查看文本文件的差异。为了查看图像文件的差异，TortoiseGit还提供了一个名为TortoiseGitIDiff的工具。当然，如果你喜欢，你可以使用自己最喜欢的diff程序。

### 文件差异

### 局部变化。

### 如果您想要查看您在工作树中所做的(未提交的)更改，只需使用资源管理器上下文菜单并选择TortoiseGit→diff。

### 与以前版本的不同。

### 如果要查看特定修订与工作树之间的差异，请使用“日志”对话框，选择感兴趣的修订，然后从上下文菜单中选择“与工作树比较”(参见。称为“Log Dialog”的部分)。

### 如果您想要查看上次提交的修订与您的工作树之间的差异，假设工作树没有被修改，只需右键单击该文件即可。然后选择TortoiseGit→diff with previous version。这将在最后提交日期之前的修订(如您的工作树中记录的那样)和工作基础之间执行差异。这将向您显示对该文件所做的最后一次更改，以使其处于您现在工作树中看到的状态。它不会显示比工作树更新的更改。

### 前两个版本的差异。

### 如果要查看已提交的两个修订之间的差异，请使用“日志”对话框并选择要比较的两个修订(使用通常的Ctrl修改器)。然后从上下文菜单中选择比较修订(参见。称为“Log Dialog”的部分)。然后出现“比较修订版本”对话框，显示已更改文件的列表(可能预先应用了文件夹筛选器)。请阅读“比较版本”一节中的更多内容。

### 提交中所做的所有更改。

### 如果要在一个视图中查看对特定修订版本中的所有文件所做的更改，可以使用Unified-diff输出(GNU补丁格式)。这只显示了几行上下文的差异。它比可视文件比较更难阅读，但会一起显示所有更改。从“修订日志”对话框中选择感兴趣的修订，然后从上下文菜单中选择“将差异显示为Unified-diff”。

### 文件之间的差异。

### 如果要查看两个不同文件之间的差异，可以直接在资源管理器中选择这两个文件(使用通常的Ctrl修改器)。然后从资源管理器上下文菜单中选择TortoiseGit→Diff。

### 与另一个分支/标记的区别。

### 如果您想要查看不同分支的更改(可能是当前分支到另一个分支或两个分支)，您可以使用日志对话框并选择两个修订，如上所述“两个以前的修订之间的差异”。更简单的方法是打开参考浏览器(参见。名为“浏览所有引用”的部分)。在那里，您可以单击一个分支并选择Compare to Working Tree，以查看该分支与工作树的当前状态之间的所有更改。也可以选择两个分支，并使用上下文菜单进行比较，如“浏览所有参照”一节所述。

### 文件夹之间的差异。

### TortoiseGit提供的内置工具不支持查看目录层次结构之间的差异。

### 如果您已经配置了第三方比较工具，则可以在选择比较命令时使用Shift来使用替代工具resp。内置工具。阅读名为“外部程序设置”的部分，了解有关配置其他比较工具的信息。

### 行尾和空白选项。

### 有时在项目的生命周期中，您可能会将行尾从CRLF更改为LF，或者您可能会更改某一节的缩进。不幸的是，这会将大量行标记为已更改，即使代码的含义没有更改。当涉及到比较和应用差异时，这里的选项将有助于管理这些更改。您将在比较版本对话框中看到这些设置(参见。称为“比较版本”的部分)，以及TortoiseGitMerge的设置。

### 忽略行尾排除了仅由于行尾样式不同而引起的更改。

### 比较空白包括缩进和内联空白中的所有更改作为添加/删除的行。

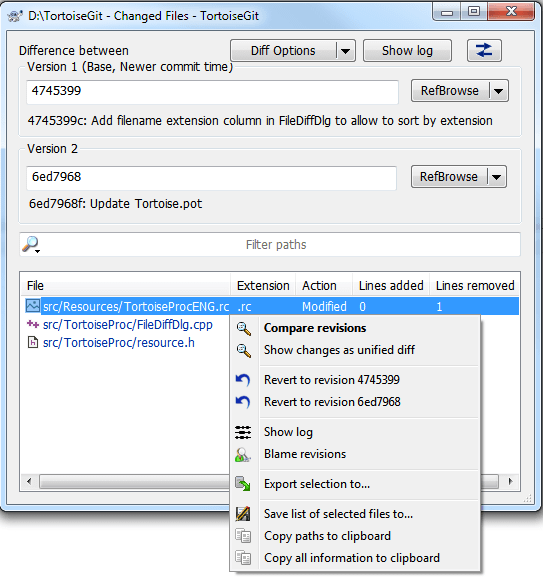
### 忽略空白更改排除仅由于空白数量或类型的更改而引起的更改，例如。更改缩进或将制表符更改为空格。在以前没有空格的地方添加空格，或者完全删除空格仍然显示为更改。

### 忽略所有空格排除所有仅包含空格的更改。

### 自然，任何具有更改内容的行总是包含在diff中。

### 比较版本

**图2.37。比较修订对话框**



在日志对话框中，当您选择两个提交**Context menu** → **Compare revisions**，或当您选择提交**Context menu** → **Compare with previous version / Compare with working tree**时；或者在Windows资源管理器中，当您选择没有文件或文件夹**TortoiseGit context menu** → **Diff with previous version**时，会出现比较修订对话框。

此对话框显示已更改的所有文件的列表，并允许您使用上下文菜单单独比较它们。

您可以将所选文件还原为版本1或版本2。有2个菜单项用于此目的。**Context menu** → **Revert to revision xxxxxxx / Revert to revision yyyyyyy**其中xxxxxxx是修订版1 short hash，yyyyyy修订版是2 short hash。

可以导出更改树，这在需要将项目树结构发送给其他人时很有用，但只包含已更改的文件。此操作仅对选定的文件起作用，因此您需要选择感兴趣的文件-通常这意味着所有文件-然后**Context menu** → **Export selection to...**。系统将提示您输入保存变更树的位置。

您还可以使用**Context menu** → **Save list of selected files to...**.将更改的文件列表导出到文本文件。

如果你想导出文件列表和操作(修改，添加，删除)，你可以**Context menu** → **Copy selection to clipboard**。

顶部的按钮允许您更改比较的方向。您可以显示从A到B所需的更改，或者如果您愿意，也可以从B到A。

修订号为ON的按钮可用于更改为不同的修订范围。当您更改范围时，两个版本之间不同的项目列表将自动更新。

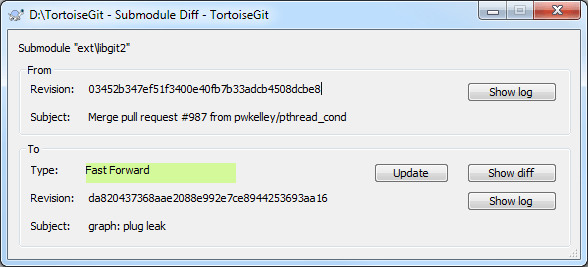
如果文件名列表很长，可以使用搜索框将列表缩减为包含特定文本的文件名。请注意，使用了简单的文本搜索，因此如果要将列表限制为C源文件，则应输入.c而不是\*.c。

搜索语法类似于“日志”对话框中提供的语法(参见。称为“过滤日志消息”的部分)。

**使用子模块Diff对话框来区分子模块**

git的内置diff命令可用于不同的子模块，但我们经常发现自己想了解更多细节，子模块是如何改变的。这就是我们创建子模块diff对话框的原因。子模块差异对话框只能使用名为“将更改提交到存储库”的部分或使用子模块上下文菜单中的**Compare with base**条目的“获取状态信息”对话框部分来访问。

**图2.38。子模块差分对话框**



顶部的“From”组框显示原始版本的信息。在它下面有一个“to”分组框，显示已更改版本的信息。对于每个组框，都会显示完整的提交散列，可以突出显示并复制到剪贴板；显示主题(即提交消息的第一行)，也可以复制到剪贴板；“显示日志”按钮会将您带到新的“日志对话框”并跳转到该修订。

为了更好地引起对安装的子模块修改的注意，我们增加了一些指标。在“至”组框中，有一个变更类型字段。在这里列出类型：

快进。

基于拓扑。这是一个快进的改变。

快退。

基于拓扑。这是快进变化的反向。

较新的提交时间。

基于时间。如果它既不是快进也不是倒带，那么我们比较提交时间。这适用于提交时间比原始修订版本更新的修订版本。

较早的提交时间。

基于时间。这是‘较新的提交时间’的反转。

相同提交时间。

基于时间。提交时间相同。这可以通过自动生成提交或由两个人同时提交来生成。

新建子模块。

这是新添加的子模块。

删除子模块。

这是针对已删除的子模块的。

未知。

这是用于子模块修订散列未更改、错误等。

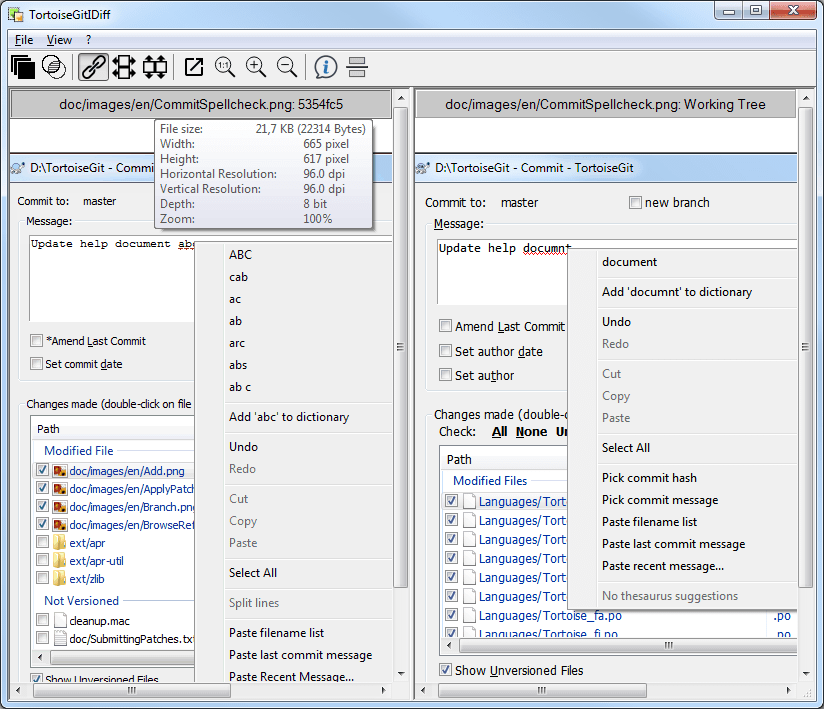
如果子模块的当前工作区是脏的，则提交散列将以黄色背景和红色文本呈现。

在两个组框中，如果未获取修订、子模块未初始化或其他错误，则提交散列将以红色背景呈现。

**使用TortoiseGitIDiff区分图像。**

有很多工具可以用来区分文本文件，包括我们自己的TortoiseGitMerge，但我们经常发现自己想要看到图像文件也发生了什么变化。这就是我们创建TortoiseGitIDiff的原因。

**图2.39。图像差分观察器**



TortoiseGit→diff对于任何常见的图像文件格式都会启动TortoiseGitIDiff来显示图像差异。默认情况下，图像并排显示，但您可以使用“视图”菜单或工具栏切换到自上而下的视图，或者如果您愿意，可以覆盖图像并假装正在使用lightbox。

当然，你也可以放大和缩小，并在图像周围平移。您也可以简单地通过左键拖动图像来平移图像。如果选择“将图像链接在一起”选项，则链接两个图像上的平移控件(滚动条、鼠标滚轮)。

图像信息框显示图像文件的详细信息，如像素大小、分辨率和颜色深度。如果此框妨碍您，请使用“查看→图像信息”将其隐藏。如果将鼠标悬停在图像标题栏上，可以在工具提示中获得相同的信息。

叠加图像时，图像的相对强度(Alpha混合)由左侧的滑块控件控制。可以单击滑块中的任意位置直接设置混合，也可以拖动滑块以交互方式更改混合。按住Ctrl+Shift-Wheels可更改混合。

滑块上方的按钮在两个预设混合之间切换，由混合滑块两侧的标记指示。默认情况下，一个在顶部，另一个在底部，因此切换按钮只是在一个图像和另一个图像之间切换。可以移动标记以选择切换按钮将使用的两个混合值。

有时你想看到不同而不是混合。您可能拥有印刷电路板的两个修订版的图像文件，并希望查看哪些轨道已更改。如果禁用Alpha混合模式，差异将显示为像素颜色值的XOR。未更改的区域将为纯白色，更改将为彩色。

外部比较/合并工具。

如果我们提供的工具不能满足您的需要，请尝试众多开放源码或商业程序中的一种。每个人都有自己最喜欢的，这个列表绝不是完整的，但这里有几个你可以考虑的：

**WinMerge**

WinMerge是一个伟大的开源diff工具，它也可以处理目录。

**Perforce Merge（强制合并）**

Perforce是一个商业RCS，但您可以免费下载diff/merge工具。从Perforce获取更多信息。

**KDiff3**

KDiff3是一个免费的diff工具，它也可以处理目录。你可以从这里下载。

**ExamDiff**

ExamDiff Standard是免费软件。它可以处理文件，但不能处理目录。ExamDiff Pro是共享软件，并添加了一些好东西，包括目录差异和编辑能力。在这两种风格中，3.2版和更高版本都可以处理Unicode。你可以从PrestoSoft下载它们。

**Beyond Compare**

类似于ExamDiff Pro，这是一个优秀的共享软件diff工具，可以处理目录差异和Unicode。从滑板车软件下载。

**Araxis Merge**

Araxis Merge是一个用于比较和合并文件和文件夹的有用的商业工具。它在Merges中进行三向比较，如果您更改了函数的顺序，还可以使用同步链接。从Araxis下载。

**SciTE**

这个文本编辑器包括统一差异的语法着色，使它们更易于阅读。从scintilla下载。

**Notepad2**

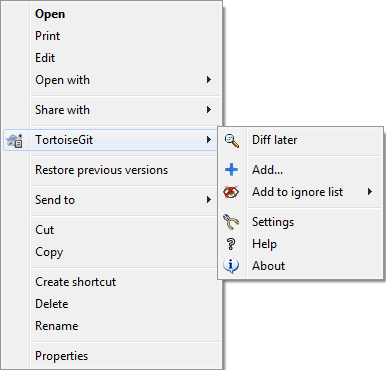
Notepad2是作为标准Windows记事本程序的替代品而设计的，并且基于scintilla开源编辑控件。除了便于查看统一的差异外，对于大多数作业，它比Windows记事本要好得多。在这里免费下载。

Notepad2作为替代编辑器包含在TortoiseGit设置中，它只支持LF行尾。将在“开始”菜单中创建一个名为Notepad2的条目。

有关如何设置TortoiseGit以使用这些工具的信息，请阅读名为“外部程序设置”的部分。

**添加新文件**

**Figure 2.40. Explorer context menu for unversioned files**



如果您在开发过程中创建了新文件，那么您也需要将它们添加到源代码管理中。选择文件和/或非空目录并使用**TortoiseGit** → **Add**。

在您将文件添加到源代码管理之后，文件会出现一个添加的图标覆盖，这意味着您必须首先提交(并推送)您的工作树，以使这些文件对其他开发人员可用。仅仅添加一个文件不会影响任何远程存储库！

**补充**

也可以对文件夹使用“添加”命令。在这种情况下，“添加”对话框将显示该版本化文件夹中所有未版本化的文件。如果您有许多新文件并且需要一次添加所有文件，这会很有帮助。

### 空目录

Git只跟踪内容，因此不能对(空)目录进行版本。如果需要在签出时自动创建目录，请确保其中至少有一个版本化文件(例如.gitKeep或.gitignore等占位符文件)。

若要从工作树外部添加文件，可以使用拖放处理程序：

A、选择要添加的文件。

B、右键将它们拖动到工作树内的新位置。

C、释放鼠标右键。

D、选择**Context Menu** → **Git Add copy and add files**。然后将文件复制到工作树并添加到版本控制中。

您也可以简单地(左键)将文件拖放到Commit对话框中来添加工作树中的文件。

如果您错误地添加了文件，您可以在使用**TortoiseGit** → **Delete (keep local)...** or **Revert**。

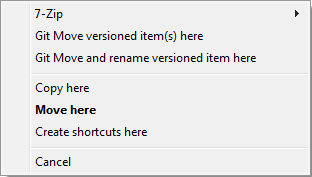
您可以在名为“git-add(1)”的部分找到更多信息

**复制/移动/重命名文件和文件夹**

经常发生的情况是，您在存储库中的另一个项目中已经有了所需的文件，而您只是想将它们复制到另一个项目中。一种方法是简单地复制文件并如上所述手动添加它们。

从工作树中复制文件和文件夹的最简单方法是使用右拖动菜单。当您右键将文件或文件夹从一个工作树拖动到另一个工作树，甚至在同一文件夹中时，释放鼠标时会出现上下文菜单。

**图2.41。版本控制下的目录右拖动菜单**



现在，您可以将现有的版本化内容复制到新位置，并可能同时对其进行重命名。

为了获得文件的旧版本，您可以使用存储库浏览器查找所需的内容，并将其直接从存储库复制到您的工作树中，或者在存储库中的两个位置之间进行复制。请参阅名为“存储库浏览器”的部分以了解更多信息。

**不法在存储库之间复制**

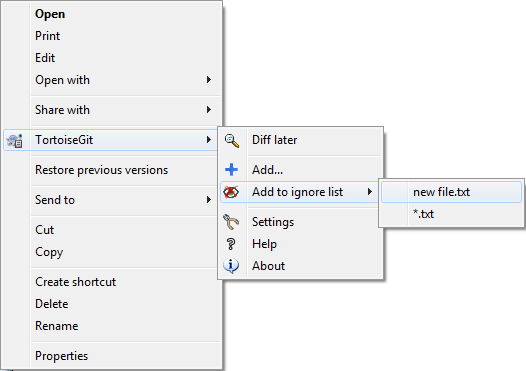
虽然您可以在存储库中复制和文件夹，但您不能在使用TortoiseGit保存历史的同时从一个存储库复制或移动到另一个存储库。即使存储库位于同一台服务器上，也不会。您所能做的就是复制当前状态的内容，并将其作为新内容添加到第二个存储库。

**Git只跟踪内容**

因为Git只跟踪内容，所以没有必要像在Subversion这样的版本控制系统中那样显式地记录副本或移动。计算日志时，Git会根据文件内容自动检测副本/重命名/移动。

**忽略文件和目录**

**Figure 2.42. Explorer context menu for unversioned files**

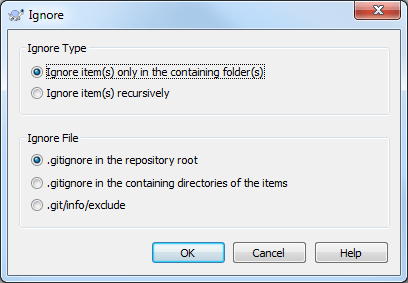


在大多数项目中，您将拥有不应受版本控制的文件和文件夹。这些文件可能包括由编译器创建的文件\*.obj，\*.lst，可能是用于存储可执行文件bin/，obj/的输出文件夹。更多示例包括用户特定的工作区设置\*.suo、\*.user(Visual Studio)、备份文件\*.bak、备份\*/、外壳元数据文件Thumbs.db、Desktop.ini、.DS\_Store/。每当您提交更改时，TortoiseGit都会显示未版本化的文件，这将填充Commit对话框中的文件列表。当然，您可以关闭此显示，但那样您可能会忘记添加新的源文件。

避免这些问题的最佳方法是将派生文件添加到项目的忽略列表中。这样它们就永远不会出现在Commit对话框中，但是真正的未版本化的源文件仍然会被标记出来。

如果右键单击一个或多个未版本化的文件，并从上下文菜单中选择命令tortoiseGit→Add to Ignore List，则会出现一个子菜单，允许您选择“按名称或按扩展名忽略”。“忽略”对话框显示，允许您选择“忽略类型”和“忽略文件”。

**图2.43。忽略对话框**



忽略类型。

仅忽略包含文件夹中的项目。

仅忽略该文件夹中选定的图案。

递归忽略项。

忽略该文件夹和子文件夹中具有选定模式的项目。

忽略文件。

.gitignore在存储库根目录中。

将忽略条目写入存储库根目录下的.gitignore中。这允许您将忽略列表与远程存储库同步。

.gitignore在项目的包含目录中。

在项目的包含目录中写入.gitignore中的忽略条目。这允许您将忽略列表与远程存储库同步。

.git/info/exclude。

将忽略条目写入存储库元数据中的.git/info/exclude中。这允许您在本地存储忽略列表，但不能与远程存储库同步。

如果要从忽略列表中删除一个或多个项目，在TortoiseGit的当前版本中，必须使用可以处理Unix EOL的文本编辑器手动编辑忽略列表文件。这允许您使用filename globbing指定更通用的模式，如下部分所述。有关更多信息，请阅读名为“gitignore(5)”的部分。请注意，每个忽略模式必须放置在单独的行上。用空格分隔它们不起作用。

**忽略列表中的模式匹配**

Git的忽略模式利用了filename globbing，这是一种最初在Unix中使用元字符作为通配符来指定文件的技术。以下字符具有特殊意义：

\*。

匹配任何字符串，包括空字符串(无字符)。

？

匹配任何单个字符。

[.]。

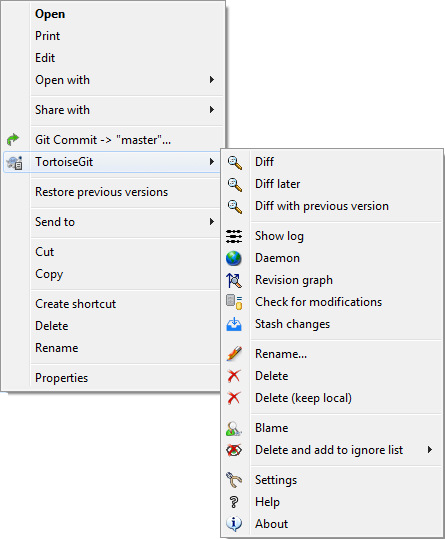
匹配方括号中的任何一个字符。在括号内，由“-”分隔的一对字符在词汇上匹配两者之间的任何字符。例如，[agm-p]匹配A、G、m、n、o或p中的任何一个。

模式匹配区分大小写，这可能导致Windows上出现问题。您可以通过配对字符来强制大小写不敏感，例如。要忽略\*.tmp而不考虑大小写，可以使用\*.[tt][mm][pp]这样的模式。

**删除、移动和重命名**

与CVS不同，Git允许重命名和移动文件和文件夹。因此TortoiseGit子菜单中有删除和重命名的菜单项。然而，与SVN不同，Git不跟踪文件名。Git只跟踪文件的内容。因此，通常不需要使用Git重命名或删除功能，甚至不需要像在SVN中那样“修复重命名”。显示日志时，会自动检测到重命名和副本。但是，使用Git删除和移动功能，文件会自动从Git索引中删除(即，不显示为缺少，但已删除)，并且在移动/重命名的情况下也会用新名称重新添加(即，新名称不显示为未版本化)。

**图2.44。版本文件的资源管理器菜单**



### 删除文件和文件夹

使用**TortoiseGit** → **Delete**从Git中删除文件或文件夹。

当您**TortoiseGit** → **Delete**一个文件时，它会立即从您的工作树中删除，并在下次提交时在存储库中标记为删除。在提交更改之前，您可以使用**TortoiseGit** → **Revert**在父文件夹上或在名为“将更改提交到存储库”部分或“获取状态信息”对话框中获取文件。

如果要从存储库中删除项目，但将其作为未版本化的文件/文件夹保留在本地，请使用**Extended Context Menu** → **Delete (keep local)**。您可能需要在资源管理器列表窗格(右窗格)中的项目上单击鼠标右键的同时按住Shift键，以便在扩展的上下文菜单中看到这一点。

### 获取删除的文件或文件夹

如果您已经删除了一个文件或文件夹，并已将删除操作提交到存储库，则正常的**TortoiseGit** → **Revert**不能再将其恢复。但是文件或文件夹根本没有丢失。如果您知道文件或文件夹被删除的修订版本(如果不知道，请使用日志对话框查找)，打开存储库浏览器并切换到该修订版本。然后选择删除的文件或文件夹，单击鼠标右键，然后选择

**Context Menu** → **Revert to this revision**。请参阅名为“存储库浏览器”的部分和名为“日志对话框”的部分以了解更多信息。

### 移动文件和文件夹

如果要对文件或文件夹进行简单的就地重命名，请使用**Context Menu** → **Rename...**。输入项目的新名称，您就完成了。

如果您想要在工作树内移动文件，可能会移动到不同的子文件夹，您可以使用鼠标右键拖放处理程序：

A、选择要移动的文件或目录。

B、右键将它们拖动到工作树内的新位置。

C、释放鼠标右键。

D、在弹出式菜单中，选择上下文菜单→Git将版本化文件移动到此处

**Do Not Git Move子模块**

您不应该在使用git子模块创建的文件夹上使用TortoiseGit move或rename命令。

**文件名中的实例更改**

对文件名进行仅大小写更改需要特别注意，因为默认情况下Windows不支持文件名大小写。因此，仅仅使用资源管理器的rename命令重命名文件可能是不起作用的。重要的是使用Git对其进行重命名，以便更新索引，使其使用新的文件名。使用重命名.。TortoiseGit子菜单中的命令。

**更改文件名中的大小写**

如果存储库已经包含两个同名但仅大小写不同的文件(例如TEST.TXT和test.txt)，您将无法提交，并且只能在Windows客户端上签出其中一个文件。虽然Git(通常)支持区分大小写的文件名，但Windows不支持。

当从具有区分大小写的文件系统(如Linux)的系统提交文件时，或者当设置core.nognrecase设置为false时，有时会发生这种情况(参见。称为“git-config(1)”的部分)。  
  
在这种情况下，您必须决定要保留其中的哪一个，并从存储库中删除其他(或重命名其他)。最简单的方法是在区分大小写的文件系统上执行此操作，然后提交并推送更改。在Windows上执行此操作需要几个步骤(和两个提交)：

解决方案

1. 在资源管理器中删除文件。

注意：

请勿使用TortoiseGit子菜单中的“删除”或“删除(保持本地)”命令！

B、打开“提交”对话框。(所有选中的项目均为已删除状态。)。

C、仅取消选中您要保留的一个项目。

D、提交更改。

E、恢复所需文件的删除，以便将其取回。如果要保留两个或多个具有“相同”名称但具有不同新名称的文件，请对所有问题的文件执行此操作，并在继续处理下一个文件之前对其进行重命名。

## 删除未版本化的文件。

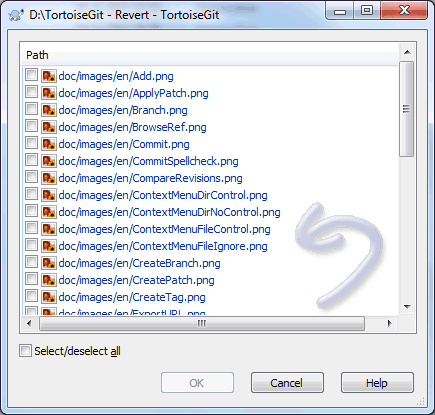
通常，您可以设置忽略列表，以便在Git中忽略所有生成的文件。但是，如果您想要清除所有那些被忽略的项目，以产生一个干净的构建，该怎么办呢？通常你会在你的makefile中设置它，但是如果你正在调试makefile，或者改变构建系统，有一种清除甲板的方法是有用的。

TortoiseGit使用扩展的上下文菜单→Clean Up.提供了这样一个选项。您可能需要按住Shift并在资源管理器列表窗格(右窗格)中的文件夹上单击鼠标右键，以便在上下文菜单中看到这一点。这将显示一个对话框，其中列出了所有可能的清理选项(参见。称为“清理”的部分)。

## 取消改变

如果您想撤消自上次提交以来在文件中所做的所有更改，您需要选择该文件，右键单击以弹出上下文菜单，然后选择命令**TortoiseGit** → **Revert**将弹出一个对话框，显示您已更改的文件并且可以恢复。选择要还原的对象，然后单击 **OK**。

**Figure 2.45. Revert dialog**



如果要撤消删除或重命名，则需要使用父文件夹(或“提交”或“存储库状态”对话框)上的Revert，因为删除的项目不存在，您可以右键单击。

如果要撤消项目的添加，这将在上下文菜单中显示为**TortoiseGit** → **Delete (keep local)**。这也是一个真正的还原，但名称已经更改，使其更加明显。

此对话框中的列可以按照与“检查修改”对话框中的列相同的方式进行自定义。有关详细信息，请阅读名为“状态”的部分。

**撤消已提交的更改 Chapter 2. TortoiseGit Daily Use Guide Undo Changes**

**回复**只会取消本地更改。它确实不撤消已提交的任何更改。如果要撤消在特定修订中提交的所有更改，请阅读[称为“日志对话”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html)和[称为“存储库浏览器”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-repobrowser.html)更多信息

### 恢复整个提交

如果要撤消整个提交，则应使用日志对话框并选择**通过提交恢复更改**on a revision/commit (cf.[称为“日志对话”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html)）然后，该提交的所有更改都被撤销，并且创建一个需要手动提交的回复提交（参见）。[被称为“Git回复（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-revert(1))）也有可能（硬）重置到以前的提交，然后所有提交之后被遗忘（参见）。[被称为“重置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-reset.html)如果更改已被按下，则不推荐使用此方法（也请参阅）[HTTPS://StAccOfFult.COM/Q/703850/39067](https://stackoverflow.com/q/27032850/3906760)）

### 回复缓慢

当您恢复更改时，您可能会发现操作花费的时间比您预期的要长得多。这是因为文件的修改版本被发送到回收站，所以如果您错误地还原，可以检索您的更改。但是，如果回收站已满，Windows需要很长时间才能找到放置文件的位置。解决方法很简单：要么清空回收站，要么停用。回收时使用回收站在TortoiseGit的设置框。

### Revert != "git revert" for files

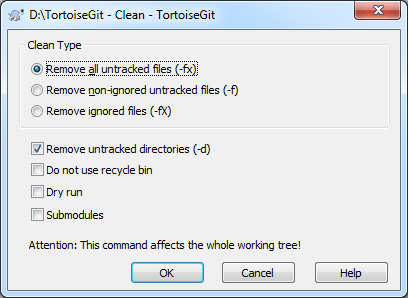
在TurtoSigeIT命名中，文件上的“回复”与Git校验头文件名（或）Git签出修订-文件名）将文件重置到其最后一个（或特定）提交状态。这与…无关。[被称为“Git回复（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-revert(1))！

[被称为“Git回复（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-revert(1))仅供参考通过提交恢复更改in log dialog (cf.[称为“日志对话”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html)）

## 清理

为了从工作树使用中移除未跟踪或忽略的文件**陆龟**渐次**清理**. 然后出现一个对话框，它可以通过递归删除不受版本控制或忽略的文件来清理工作树，从当前目录开始。或在整个工作树上（取决于安装的Git版本）。

**图2.46。干净对话框**



清除所有未跟踪的文件这将删除所有未跟踪的文件，包括那些被Git忽略的文件。这是最干净的选择。

只清除不忽略未跟踪的文件这将删除未跟踪的文件，但排除那些被Git忽略的文件。

只清除忽略的文件这只删除Git忽略的文件。

删除未跟踪目录这也删除了未跟踪的目录。

不要使用回收站如果要直接和永久地删除这些文件，请使用此选项。请务必不要后悔！

排练这只是给出要删除的文件的列表，但是它不执行任何删除。

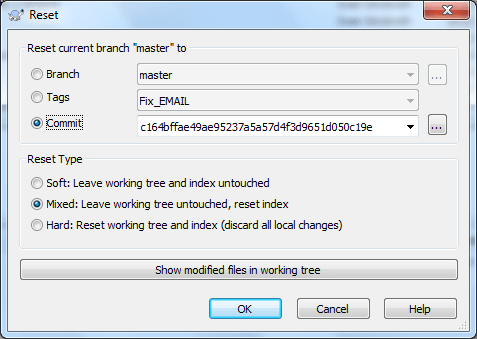
子模块这也递归地清除子模块。

你可以找到更多的信息。[“Git Curn（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-clean(1)).

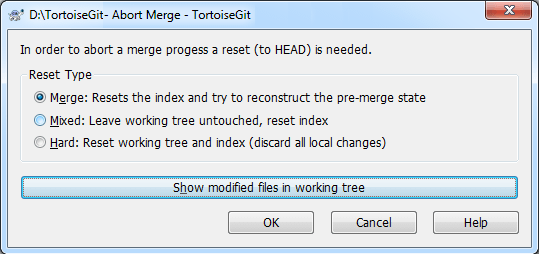
## 重置

重置对话框可用于将当前磁头重置为指定状态，并可选地重置索引和工作树。这也可以用于中止合并。

**图2.47。重置对话框**



**图2.48。中止合并对话框**



在“重置”对话框中，您可以单击**…**浏览日志并选择特定版本。在中止合并对话框中，只能重置为头。

软：离开工作树和索引未触及不触摸索引文件或工作树（但重置头到所选择的提交，就像所有的模式一样）。这使得所有更改后的文件“更改为提交”。此选项在异常合并对话框中不可用。

混合：离开工作树未被触摸，重置索引重置索引而不是工作树（即，更改的文件被保留但未标记为提交），并报告未更新的内容。这是默认的Git动作。此选项可以中止合并。

硬：重置工作树和索引（丢弃所有本地更改）重置索引和工作树。从选择的提交中删除工作树中跟踪文件的任何更改。此选项可以中止合并，它是异常合并对话框中的默认操作。

### Git硬复位不使用Windows回收站

不像TurtoSigeIT的回复或干净功能，硬复位不使用Windows回收站，即，未提交的更改可能丢失！

你可以找到更多的信息。[被称为“Git Read（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-reset(1)).

## 贮藏变化

通常，当你在项目的一部分工作时，事情处于混乱状态，你想把分支切换一点来处理其他事情。问题是，你不想做一半的工作，这样你就可以回到这一点。这个问题的答案是Git StScript命令。

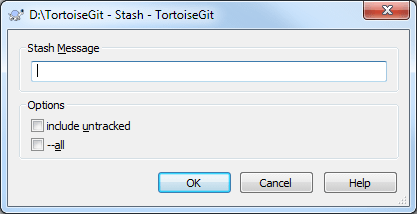
Stury占用工作目录的脏状态，即修改后的跟踪文件和阶段性的更改，并将其保存在一堆未完成的更改中，您可以在任何时候重新应用（甚至在不同的分支上）。

### 小贴士

在TotoSigeTe中有一个类似的函数叫做CeleListSistar（[称为“更改列表”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-changelists.html)可用于构造提交

当要记录工作目录和索引的当前状态时，要返回干净的工作目录，**右击**在一个文件夹中弹出菜单，然后选择命令**陆龟**渐次**贮藏变化**一个对话框将弹出，您可以在其中可选地输入该状态的消息：

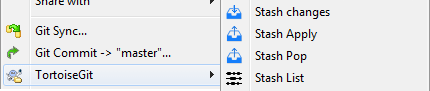
**图2.49。隐藏更改对话框**



你也可以选择包括未跟踪的把隐藏的文件也藏起来。要将所有文件隐藏起来，包括未跟踪文件之外的忽略文件，请选择全部.

当TortoiseGit检测到存在隐藏的更改时，菜单将被扩展：

**图2.50Stash Options**



堆栈实现为堆栈**堆叠应用**将将最新的堆栈的更改应用到您的工作树。**隐藏弹出**做同样的操作，但是在应用它之后会从堆栈中移除最新的堆栈。**贮藏变化**仍然是可能的，并且将当前工作副本的更改隐藏到堆栈的顶部。**隐藏列表**提供整个堆栈堆栈的概述。您也可以删除并查看其中的隐藏更改（类似于[称为“日志对话”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html)和[称为“参考日志”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-reflog.html)）

### 冲突

虽然主要的合并工作是由Git自动应用一个堆栈来完成的，但是在樱桃采摘过程中可能会发生冲突（即，在当前分支中也修改了文件中的文件），请参阅。[“解决冲突”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-conflicts.html)如何解决冲突

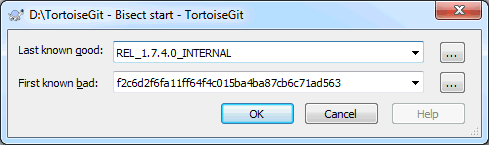
请注意，冲突编辑器中的“远程”/“它们”是指在应用堆栈之前将要合并的存储区和“本地”/“挖掘”到工作树中的版本。

你可以找到更多的信息。[被称为“Git StAsh（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-stash(1)).

## 二等分

如果你想知道哪个版本引入了一个bug，你可以使用BISECT功能。**右击**在一个文件夹中弹出菜单，然后选择命令**陆龟**渐次**平分起始**. 一个对话框将会弹出：

**图2.51。平分起始**

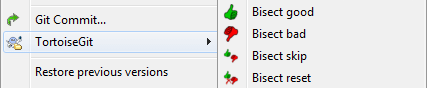


输入最后一个已知的好修订和第一个或一个已知的坏（这通常是头）。

击球后**好啊**Git将对第一次错误的修订进行二进制搜索：Git在中间切换到一个修订。现在你可以测试这个版本了。

TurtoSeigIT现在提供了菜单中的三个新选项：

**图2.52。平分期权**



如果这个修订是可以的，点击**陆龟**渐次**Bisect好**否则命中**陆龟**渐次**二分差**跳过修订版**陆龟**渐次**二等分跳跃**Git将继续进行二进制搜索并切换到“下一个”修订版，以便您可以对其进行测试。这一直持续到错误的修订被发现或者你通过点击中止这个操作。**陆龟**渐次**二次重置**（这将重置平分过程，并切换出以前的分支/头）。

### 选择修订

如果一个修订不能被测试，或者你想继续使用另一个版本，那么你可以很容易地进入日志，并且（很难）将当前的头重置为你喜欢的版本。

### 子模块

如果使用子模块，则可能需要确保在每个分步之后更新这些子模块，以便所有依赖项都是最新的。

你可以找到更多的信息。[“Git二等分（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-bisect(1))

## Branching/Tagging

版本控制系统的特点之一是将更改隔离到单独的开发线上。这条线被称为分支. 分支通常用于尝试新的特性，而不干扰编译器错误和错误的主要开发线。一旦新特性足够稳定，那么开发分支就是合并回到主要的分支

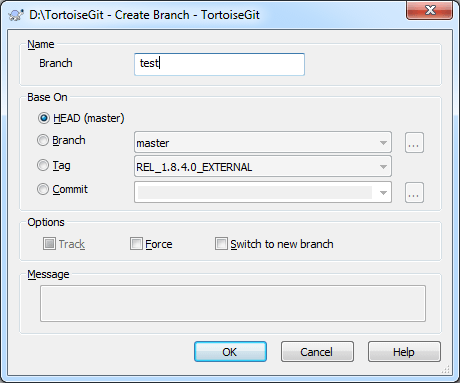
版本控制系统的另一个特点是标记特定修订（例如发布版本）的能力，因此您可以在任何时候重新创建特定的构建或环境。这个过程称为标注.

Git在分支和标记方面非常强大。创建分支和标签非常容易。

### 创建分支或标签

创建分支非常简单：**陆龟**渐次**创建分支…**

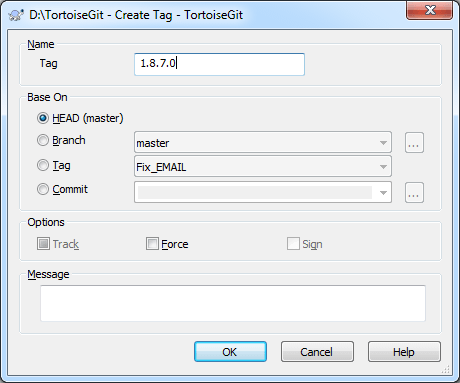
**图2.53。分支对话框**



Branch:输入您的分支名称

创建一个标签非常简单：**陆龟**渐次**创建标签…**

**Figure 2.54. The Tag Dialog**



标签input your tag name.

您可以基于此选择一个提交。

头部

当前提交已签出

分支

选择分支的最新提交

标签

选择标签的提交

提交

任何提交，您点击**…**启动日志对话框来选择提交。您还可以输入提交哈希或友好提交名称，例如头颅~ 4.

如果希望将工作树自动切换到新创建的分支，请使用切换到新的分支/标签复选框。但是如果你这样做，首先确保你的工作树不包含修改。如果确实如此，当您切换时，这些更改将合并到分支工作树中。

歌曲是一个具有三个值的复选框。如果检查歌曲传递给Git好啊如果没有检查--没有轨道传递给Git好啊. 第三种状态表明：歌曲也不--没有轨道传递给Git好啊- seeAutoStupUp合并配置变量（配置变量）[被称为“Git配置（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-config(1))）--轨道参数文件编制[“Git分支（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-branch(1)).

检查标志创建GPG签名标签。这需要GPG和配置变量。用户密钥被设置（见）[被称为“Git配置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-config)和[被称为“Git配置（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-config(1))）

### 小贴士

当使用GPG 1.4（用Git为Windows发货）时，这需要一个密钥。没有a passphrase. GPG >= 2 comes with an agent like pageant and, thus,.also works with passphrase protected keys, however, you might need to configure git to use the right gpg.exe. This can be done be setting the configuration variableGPG程序(e.g.,*C//程序文件（x86）/GNU/GNUPG/PUB/GPG.EXE*). We tested this with[GPG4WIN](http://www.gpg4win.de/)（GPG4Win香草是足够的，并且版本2.2 .x也兼容GPG 1.4密钥文件）。

新闻**好啊**创建分支或标签本地仓库.

注意，除非选择将工作树切换到新创建的分支，否则创建分支或标记。不影响你的工作树。即使从工作树创建分支，这些更改也会提交给原始分支，而不是新分支。

关于如何将工作树转换为标签/分支，请参阅[称为“检查工作树（切换到提交）”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-checkout.html).

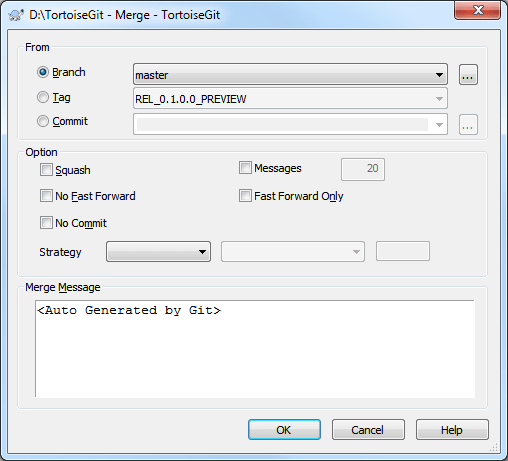
## 合并

当分支被用来保持单独的开发线时，在某个阶段，您会希望将一个分支上的更改合并到另一个分支中，反之亦然。

在开始使用Git之前，了解分支和合并是如何工作的很重要，因为它可能变得相当复杂。有关在哪里查找更多关于Git和合并的信息的提示[“阅读指南”一节](https://tortoisegit.org/docs/tortoisegit/tgit-preface-readingguide.html).

接下来要注意的是合并。总是在工作树内发生。如果要合并更改进入之内一个分支，您必须为该分支签出一个工作树，并使用该工作树调用合并向导。**陆龟**渐次**合并。。**.

**图2.55。合并对话框**



一般来说，将合并执行到未修改的工作树中是一个好主意。如果在工作树中进行了其他更改，请首先提交这些更改。如果合并未按预期执行，则可能希望使用中止合并可能废弃的命令全部更改（取决于模式），以防硬）

您可以选择要从中合并的一个提交。

头部

当前提交已签出

分支

选择分支的最新提交

标签

选择标签的提交

提交

任何提交，您点击**…**启动日志对话框来选择提交。您还可以输入提交哈希或友好提交名称，例如头颅~ 4.

壁球只需从另一个分支合并更改即可。录音机不能合并信息。新提交将不会将合并分支记录为一个父提交。日志视图将不会显示两个分支之间的合并行。

无快进即使合并解决为快进，也会生成合并提交。见[HTTPS://StAccOfFult.COM/Q/4179425/39067](https://stackoverflow.com/q/41794529/3906760)例如快进与非快进合并的例子。

不提交不要在合并后自动创建提交。

消息从正在合并的实际提交中使用一行描述填充日志消息。可以指定合并消息中包含的提交数。

### 冲突

虽然主要合并工作是由Git自动完成的，但在合并过程中可能会发生冲突（即，文件在两个分支中修改，当前的一个和您要合并的一个分支），请参阅[“解决冲突”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-conflicts.html)如何解决冲突

请注意，冲突编辑器中的“远程”/“他们”指的是你在你的工作树中选择的用于合并的分支的更改和“本地”/“挖掘”到你的头版本。

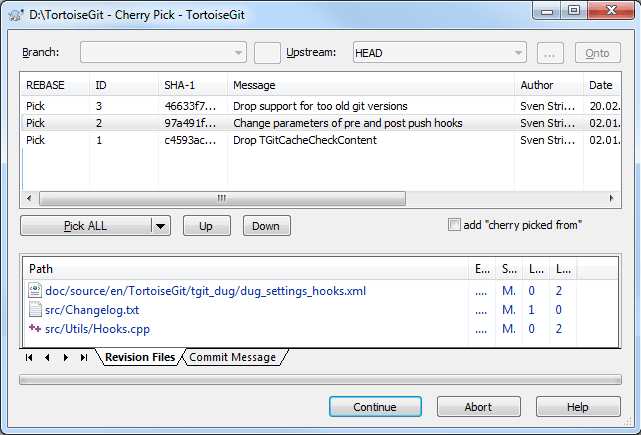
你可以看到更多的信息。[“Git合并（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-merge(1)).

## 樱桃采摘

[樱桃采摘](https://tortoisegit.org/docs/tortoisegit/git-command.html" \l "git-cherry-pick(1)" \o "git-cherry-pick(1))在TortoiseGit中调用[修订日志对话框](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html). 在这个对话框中，选择提交到樱桃挑选，然后右键单击一个选定的提交弹出菜单。选择樱桃选择这个承诺…（或）樱桃选择选择提交…如果选择一个以上的提交）。

樱桃采摘对话框将被显示。

**图2.56。樱桃采摘对话**



樱桃拾取对话框类似于[重新对话](https://tortoisegit.org/docs/tortoisegit/tgit-dug-rebase.html). 顶部表显示每一个选择提交到樱桃挑选的一行。它下面的按钮控制动作（选择、挤压、编辑、跳过）和选择多个提交的顺序。选择一行显示由提交影响的文件。

### 冲突

虽然主要的合并工作是由Git自动完成的，但在樱桃采摘过程中可能会发生冲突（即，在当前分支中修改了一个文件，并且在一个或多个提交中，您正在挑选樱桃），请参见[“解决冲突”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-conflicts.html)如何解决冲突

请注意，冲突编辑器中的“远程”/“他们”指的是你在你的工作树中选择的“局部”/“地雷”的变化。

你可以找到更多的信息。[“Git樱桃采摘（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-cherry-pick(1)).

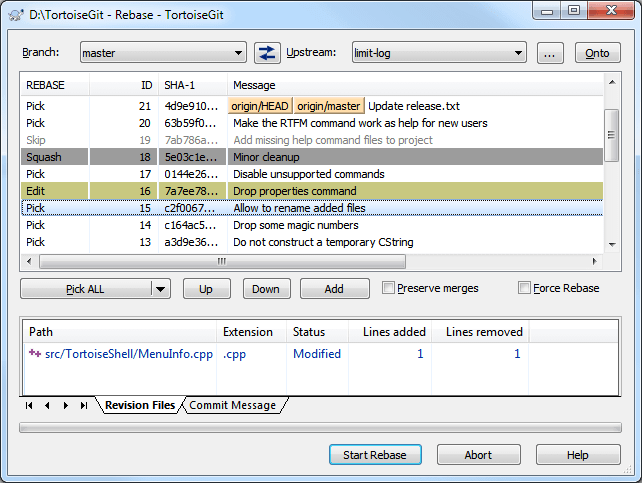
## Rebase

ReBASE是相当复杂的，它改变/重写了存储库的历史。请确保您在使用之前了解它的原理（对于一般提示，在那里可以找到更多关于Git和ReBASE见的信息）。[“阅读指南”一节](https://tortoisegit.org/docs/tortoisegit/tgit-preface-readingguide.html)特别是[被称为“Git ReBASE（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-rebase(1))）

**TortoiseGit**→**Rebase**

将显示ReBASE对话框。

**Figure 2.57. Rebase dialog**



ReBaseCasePress从列表底部到顶部（按ID列的升序）。例如，壁球意味着提交被压扁/合并到位于列表下面的前一个提交（具有较低的ID）。

### 小贴士

代替设置挑选，跳过，编辑，壁球通过使用菜单，还可以使用以下键：**空间**改变国家，**S**：跳过，**e**：编辑，**磷**：挑选，**Q**：壁球

### 小贴士

有一个按钮交换树枝和上游。假设您当前正在工作硕士分支机构，希望重新立足功能特色分支到硕士. 而不是切换到功能特色预先选择提交功能特色在日志列表中，**菜单**渐次**重碱**然后点击这个交换按钮。TortoiseGit的重新定位动作功能特色到硕士直接，然后樱桃选择提交。这种方法接触较少的文件，运行速度更快。

### 重要的

当保存合并提交时，在所有情况下都不能正确处理重新排序提交，参见香草Git ReBASE的已知错误：[被称为“Git ReBASE（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-rebase(1)).

### 冲突

虽然主要合并工作是由Git自动完成的，但在ReBASE中可能会发生冲突（即，文件在两个分支中被修改，一个是重新建立一个，而您正在重新建立一个文件），请参阅[“解决冲突”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-conflicts.html)如何解决冲突

请注意，冲突编辑器中的“远程”/“他们”指的是你重新定位的分支和“本地”/“地雷”在你正在重新建立的分支上的版本的变化。

## 解决冲突

在合并过程中，更新工作树文件以反映合并的结果。偶尔，你会得到一个冲突当你合并另一个分支时，樱桃选择提交，ReBASE或应用一个存储：在对共同祖先版本进行的更改中，非重叠的（即，改变了文件的一个区域，而另一个区域保留了完整的区域，反之亦然）则被逐字地合并在最终结果中。然而，当双方对同一区域进行更改时，Git不能随机地选择另一边，并要求您通过留下双方对该区域所做的来解决它。无论何时报告冲突，你都需要解决它！

在这样的文件中标记了冲突区域（参见CF）。[“冲突如何呈现”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-merge(1)__how_conflicts_are_presented)）：

<<<<<<< yours

your changes

=======

changes from the code merged

>>>>>>> their

可以使用任何编辑器手动解决冲突，或者可以启动外部合并工具/冲突编辑器。 **TortoiseGit**→ **Edit Conflicts**. 然后，TortoiseGit将在您的目录中放置三个附加文件，用于选定的冲突文件，并启动配置的冲突编辑器：

文件名

这是冲突文件的共同祖先版本（这个版本既不包含你的任何一个，也不包含要被合并的分支/修订的任何更改，特别是它不包含任何冲突标记）。

文件名

这是您的文件，因为它存在于您的工作树中，在您开始合并之前（即，文件符合您的本地存储库的头的最新提交状态）——也就是说，没有冲突标记。因此，这个状态/版本通常也被称为“我的”。

仅仅为了完整性，“mine”意思是在你的工作树中 "stash"/"merge"/"pull"/"cherry-pick"头的版本，并在你重新定义的分支上“重置”版本。

文件名

这是你想合并的版本文件的版本（在一个普通的合并到MeGeHead）。当您想合并其他更改时，该状态/版本通常也被称为“theirs”。

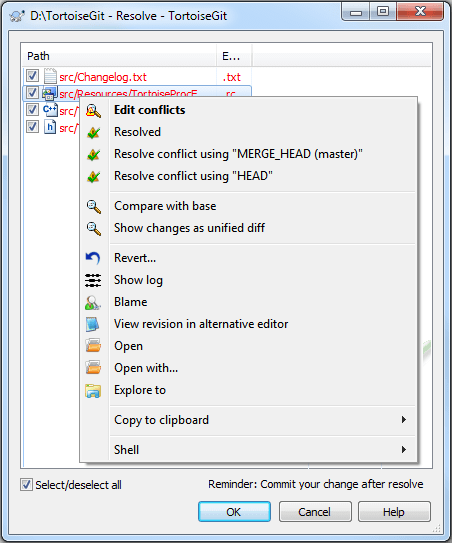
仅仅为了完整性“他们的"stash"/"merge"/"pull"/"cherry-pick"”版本的合并提交/分支，并“rebase”您重新创建的分支版本。

然后执行命令**TortoiseGit** → **Resolved**并将修改提交到存储库（如果冲突发生在重新排序或樱桃拾取时，确保使用樱桃拾取RESP。重新提交对话框，而不是正常提交对话框！请注意，解析命令并不能真正解决冲突。它使用“Git Addio”来标记文件状态，以使您能够提交更改，并移除该更改。*filename.ext.BASE.ext*, *filename.ext.LOCAL.ext* and *filename.ext.REMOTE.ext* files.

如果与二进制文件有冲突，Git不会尝试合并文件本身。本地文件保持不变（正如您上次更改它）。为了解决冲突使用**TortoiseGit** → **Resolve...**然后右键单击冲突文件并选择其中之一**Resolved**（将使用工作树中的文件的当前版本），**Resolve conflict using 'mine'**（将使用您的文件的版本），以及**Resolve conflict using 'theirs'**（合并版本/分支的文件版本将被使用）。之后提交。

你可以使用**Resolved**如果您右键单击父文件夹并选择**TortoiseGit** → **Resolve...**这将生成一个对话框，列出该文件夹中所有冲突的文件，并且可以选择要标记的文件。

**图2.58。解决冲突对话框**



### 重要的

Git（不像VSN）不会自动创建*filename.ext.BASE.ext*, *filename.ext.LOCAL.ext* and *filename.ext.REMOTE.ext*。这些命令只在TortoiseGit使用命令时按需创建。**Edit Conflicts**.

### 重要的

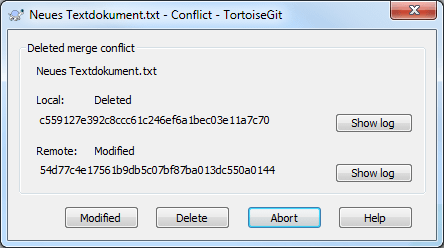
在Git（不像VSN），你必须[commit](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html)解决冲突后。然而，如果发生冲突，而重新基础或樱桃采摘确保你使用樱桃采摘。重新提交提交对话框，而不是正常提交对话框！

### Special conflict cases

#### 删除修改冲突

一个特殊的冲突案例是删除修改冲突。这里，在一个分支上删除一个文件，在另一个分支上修改同一个文件。为了解决这个冲突，用户必须决定是保留修改版本还是从工作树中删除文件。

**图2.59。解析删除修改冲突对话框**

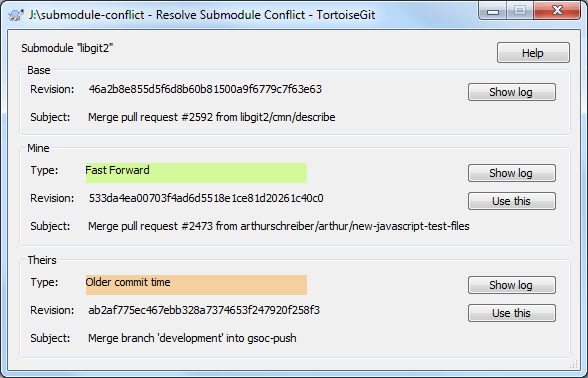


#### 子模块冲突

另一个特殊的冲突案例是涉及子模块的冲突。这里，子模块在两个分支上以不同的（冲突的）方式改变。

解析子模块冲突对话框显示了冲突子模块的基础、本地和远程提交以及提交类型（倒带、快进、……）。

**图2.60。解析子模块冲突对话框**



### 未初始化子模块

如果子模块尚未初始化，解析子模块冲突对话框只显示提交ID（SHA-1）。此外，冲突无法自动解决：首先，必须手动将子模块克隆到正确的文件夹中。然后，您可以使用TurtoSigEit或Git（通过检查子模块中的正确提交并提交父工作树）来解决冲突。

## 创建和应用补丁和拉请求

对于开源项目（像这一个），每个人都有读取（主/公共）存储库的权限，任何人都可以为项目做出贡献。那么这些捐款是如何控制的呢？如果任何人都可以改变这个中央存储库，这个项目将永远不稳定，并且可能永久中断。在这种情况下，变更是通过提交补丁文件或拉动请求对于开发团队，谁有写访问权限。他们可以首先查看这些更改，然后提交给主存储库，或者将其退回给作者。

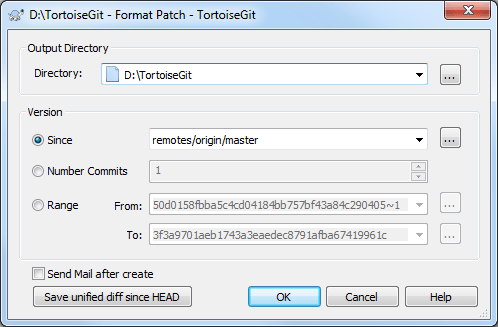
补丁文件只是统一的差异文件，显示了工作树和基本版本之间的差异。

拉请求是对另一个存储库所有者的请求。[拉](https://tortoisegit.org/docs/tortoisegit/tgit-dug-pull.html)从存储库中进行更改。也就是说，您必须访问公共存储库。[推](https://tortoisegit.org/docs/tortoisegit/tgit-dug-push.html)你的变化（通常是一个特殊的分支）。

### 创建补丁序列

首先你需要做测试你的改变。然后你提交你的更改**TortoiseGit**→**Commit...**在父文件夹中，输入一个好的提交消息。选择之后**TortoiseGit**→**Create Patch Serial...**并选择正确的选项来包括更改/提交。

**图2.61。创建补丁对话框**



Directory 目录是补丁的输出目录。补丁文件名将由提交主题创建。

Since自从从点创建补丁。你可以点击**…**启动RefScess对话框选择分支或标签。

Number Commits 数字提交限制了多少补丁将被创建。

Range范围选择范围从致力于到. 你可以点击**…**启动日志对话框来选择提交。

Send Mail after create 创建后发送邮件在创建补丁后启动发送邮件对话框（参见[被称为“通过邮件发送补丁”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-patch.html#tgit-dug-patch-sendpatches)）

你可以找到更多的信息。[“Git格式修补程序（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-format-patch(1)).

### 重要的

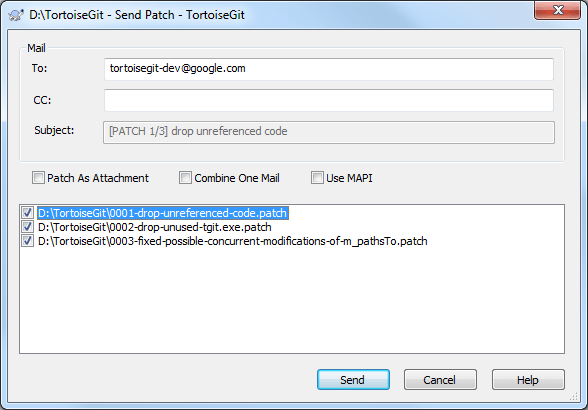
这里的Git与TurtoSeVN不同：在TurtoSeVN中，您直接创建一个补丁，而不是提交更改，然后创建一个提交的补丁（在Git中，您所克隆的项目有一个完整的本地副本/分叉-提交只是本地的）。若要生成包含未提交但已分级的更改的修补程序，请单击**Save unified diff since HEAD 从头开始保存统一差异**.

有关在哪里查找与Git进行版本控制有关的更多信息的提示[“阅读指南”一节](https://tortoisegit.org/docs/tortoisegit/tgit-preface-readingguide.html).

### 通过邮件发送补丁

为了向上游作者发送补丁，选择补丁文件，然后右键单击它们并选择**TortoiseGit**→**Send Mail...**

**图2.62。发送补丁对话框**



首先，您需要输入收件人（s）。To and/or CC）

取决于邮件类型（Patch as attachmentorCombine One Mail 贴片作为附件或合并一封邮件）你必须进入Subject (主题)为了邮件

贴片作为附件将附件（ES）作为附件（S）添加到邮件中，而不是内联。

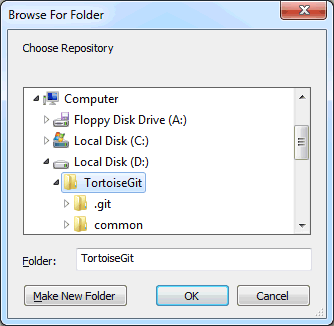
合并一封邮件将所有补丁添加到一个邮件中。你必须进入主题对于这种情况下的邮件。

### 应用单个补丁文件

补丁文件应用于您的工作树。这应该从与创建修补程序相同的文件夹级别完成。如果你不确定这是什么，只要看看补丁文件的第一行。例如，如果正在处理的第一个文件是*DOC/源程序/英文/章节1 XML*补丁文件中的第一行是*索引：英语/Chutel1.xml*然后，您需要将补丁应用到*资料来源*文件夹但是，如果您在正确的工作树中，如果选择错误的文件夹级别，TurtoSigeIT将通知并建议正确的级别。

从修补程序文件的菜单（*补丁*或*微分*扩展，点击**TortoiseGit**→**Review/apply single patch...**可能会提示您输入工作树位置：

**图2.63。选择仓库对话框**



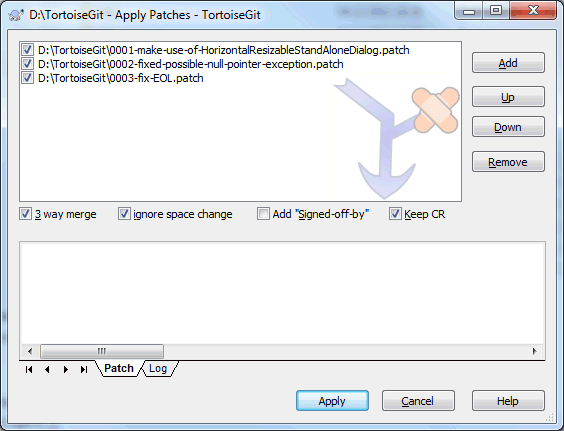
如果找到了工作树，则启动TurtoSigeIGIt来显示和应用差异。

### 应用补丁序列

补丁文件应用于您的工作树。对于这个复制补丁（或MBOX）文件到你的工作树的根目录。

从该文件夹的菜单（或所有标记的补丁文件）中，单击**TortoiseGit**→**Apply Patch Serial...**

**图2.64。应用补丁对话框**



**Add 添加**插入贴片

**Up 向上**移动选择补丁

**Down 向下**移动选择补丁

**Remove 去除**删除选定的修补程序

**Apply 申请**开始逐个应用补丁

你可以找到更多的信息。[“Git AM（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-am(1)).

### 创建拉动请求

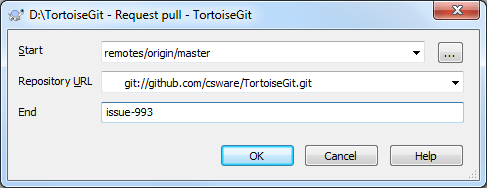
除了向其他开发人员发送补丁（或补丁系列）之外，还有两种方法让其他人将更改集成到他们的存储库中。

First:在将更改推到（公共）存储库之后，您只需向其他人提供存储库的URL和分支的名称或修订ID。GITI//Excel

第二：在将更改推到（公共）存储库之后，您可以创建一个标准化的（非常正式的）请求，以便其他人拉动您的更改并将其集成到存储库中。格式拉动请求包含所有提交的列表，并提供有关更改文件的一些统计信息，以便其他人可以快速查看。

选择**Request pull**在推送更改之后的进度对话框中。

**图2.65。请求拉动对话框**



Start 开始

这应该是您的更改基于的修订。

URL 统一资源定位地址

公共URL到您的储存库，这可以由将拉动您的更改的人访问。

End 终点

这应该是提交结束的分支名称或修订ID。

点击后**OK**创建拉请求。只需复制并传递给其他你想要改变的人。

你可以找到更多的信息。[“GIT请求拉动（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-request-pull(1)).

## 谁改变了哪条线？

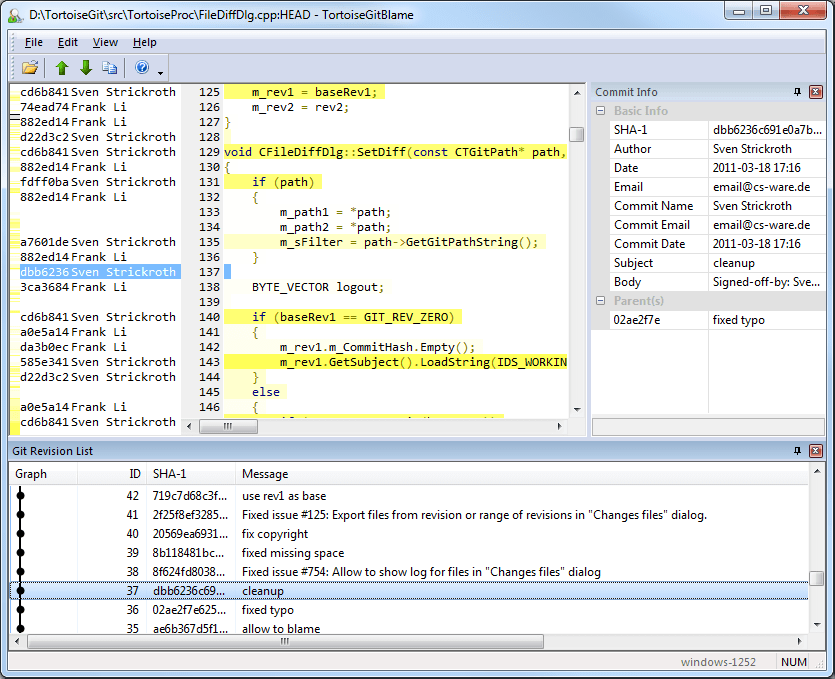
有时，您不仅需要知道哪些行已经更改，还需要确切地更改文件中的特定行。那是时候**TortoiseGit** → **Blame...**命令，有时也被称为注释命令派上用场

这个命令列出了一个文件中的每一行，作者和修改了该行。

### 归咎于档案

默认情况下，使用龟裂它突出了不同的修改，使之更容易阅读。

**图2.66。龟裂**



TurtoSigeT责备，这是与TortoiseGit。当您将鼠标悬停在Buffic信息栏中的一行上时，所有具有相同修订的线条都显示为较深的背景。由同一作者改变的其他修订的线条以浅背景示出。如果你的显示设置为256色模式，颜色可能不那么清楚。

如果你**左键单击**在一条线上（在左边的责备信息栏上），所有相同的修订的线条都被高亮显示，而来自同一作者的其他修订的线条则以较浅的颜色突出显示。这个高亮是粘性的，允许你移动鼠标而不会失去亮点。再次单击该修订以关闭高亮显示。

每当鼠标悬停在Buffic信息栏上时，修改注释（日志消息）就会显示在提示框中。如果要复制该修订的日志消息，请使用右击“责备信息”列时出现的菜单。

如果你需要一个更好的视觉指示器，指出最老的和最新的变化在哪里，选择**View** → **Colorise by age, continous**. 线的背景色强度与其年龄有关。这将使用颜色渐变来显示黄色和较老的白色线条中的新线条。默认着色很轻，但是你可以使用TurtoSigeT责备设置来改变它。

也请检查一下**View 视图**菜单在那里你可以切换**Ignore whitepace忽略白皮书**还可以切换其他文件的移动/复制行的检测。**Follow renames 重命名**.

你可以在责备报告中使用**Edit** → **Find...**. 这允许您搜索修订号、作者和文件本身的内容。日志消息不包含在搜索中-您应该使用日志对话框来搜索这些消息。

您也可以跳转到特定的行号使用**Edit** → **Go To Line...**.

当鼠标在Buffic信息栏上时，可以使用菜单来帮助比较修订和检查历史，使用鼠标下的行提交作为参考。**Context menu** → **Blame previous revision**为同一个文件生成一个责备报告，但是使用以前的修订作为上限。这给你的文件的状态报告之前，你刚才看到的线是最后一次改变。**Context menu** → **Show changes**启动DIFF查看器，向您显示文件引用修改中发生了什么变化。但是，请注意，这两个选项仅在文件初始开头时没有出现。**Context menu** → **Show log** displays the revision log dialog starting with the referenced revision.

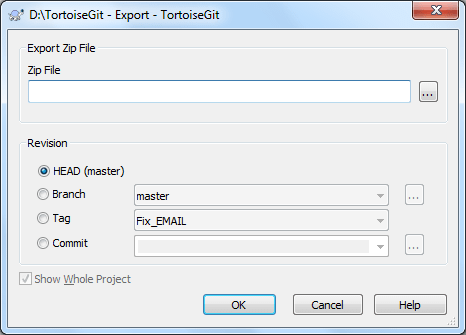
可以使用TurtoSebug设置。**TortoiseGit**→**Settings...**关于TortoiseGitBlame tab。参照[“TortoiseGitBlame Settings”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-tortoiseblame).

你可以找到更多的信息。[“Git责备（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-blame(1)).

## 导出Git工作树

有时您可能需要一个特定的修订/提交的快照，例如创建源的拉链塔式球，或者导出到Web服务器。为此，TurtoISeGIT提供了命令**TortoiseGit**→**Export...**.

**图2.67。导出对话框**



Zip File 压缩文件出口压缩文件

HEAD 头部

当前提交已签出

Branch 分支

选择分支的最新提交

Tag 标签

选择标签的提交

Commit 提交

任何提交，您点击**…**启动日志对话框来选择提交。您还可以输入提交哈希或友好提交名称，例如头颅~ 4.

你可以找到更多的信息。[“Git档案（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-archive(1)).

### 导出单个文件

导出对话框不允许导出单个文件。

要用TortoiseGit导出单个文件，必须使用存储库浏览器（参见）。[称为“存储库浏览器”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-repobrowser.html)) or log dialog (cf.[称为“日志对话”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-showlog.html)）只需将要从库浏览器导出的文件拖动到Explorer中想要的文件中，或者使用存储库浏览器中的菜单导出文件。

## 与错误跟踪系统/问题跟踪器集成

在软件开发中，与特定的bug或问题ID相关联的变化是非常常见的。bug跟踪系统的用户（问题跟踪器）希望将他们在GIT中所做的更改与他们的问题跟踪器中的特定ID相关联。因此，大多数问题跟踪器提供了一个预提交钩子脚本，该脚本分析日志消息以找到与提交相关联的bug ID。这有点容易出错，因为它依赖于用户正确地编写日志消息，以便预提交钩子脚本能够正确解析它。

TurtoSigeIT可以通过两种方式帮助用户：

1. 当用户输入日志消息时，可以自动添加包括与提交相关联的发布号的定义良好的行。这降低了用户以错误跟踪工具无法正确解析的方式输入发行号的风险。

或者TurtoSigEIT可以突出显示由问题跟踪器识别的输入日志消息的一部分。这样，用户知道日志消息可以被正确解析。

1. 当用户浏览日志消息时，TortoiseGit在日志消息中创建每个bug ID的链接，该日志消息将浏览器触发到所提到的问题。

### 将日志号添加到日志消息

您可以在TortoiseGit集成一个您选择的bug跟踪工具。要做到这一点，必须定义一些配置，这些配置从布格拉克. 可以使用TurtoSeGIT设置对话框编辑这些设置：[称为“配置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-hooks-issuetracker-config)

有两种方法将TortoiseGit与问题跟踪器集成在一起。一个是基于简单的字符串，另一个是基于正则表达式. 两种方法使用的配置是：

BugTrq.URL

将此配置设置为bug跟踪工具的URL。必须正确地进行URI编码，并且必须包含%BUGID%.%BUGID%用输入的问题号替换。这允许TotoSigeIT在日志对话框中显示链接，因此当您查看修订日志时，您可以直接跳转到bug跟踪工具。你不必提供这个配置，但是TurtoSigeIT只显示了发行号，而不是它的链接。例如TurtoSigEIT项目正在使用https://tortoisegit.org/issue/%BUGID%

警告问题

设置为真，如果您想让TortoiseGit警告您，因为一个空的发行号文本字段。有效值是真实.如果没有定义，*假*假设

#### 文本框中的发布号

在简单的方法中，TortoiseGit向用户显示一个单独的输入字段，其中可以输入一个bug ID。然后将单独的行附加/添加到用户输入的日志消息中。

BugTrq.消息

此配置激活了错误跟踪系统。输入字段模式。如果设置了此配置，则当您提交更改时，TortoiseGit将提示您输入发行号。它用于在日志消息的末尾添加一行。它必须包含%BUGID%用提交的问题号替换。这确保了您的提交日志包含对发行号的引用，该编号始终是一致的格式，可以由bug跟踪工具解析，以将发行号与特定提交相关联。作为一个例子，你可以使用问题：%BugID%但是这取决于你的工具。

BugTrq.App.

此配置定义了是否将bug ID追加到日志消息的结尾（true）或在日志消息的起始处插入（false）。有效值是真实.如果没有定义，*真*假设现有的项目不会中断。

标签标签

本文在提交对话框上由TurtoSigEt显示，以标记您输入发行号的编辑框。如果没有设置，Bug-ID / Issue-Nr:将被显示。请记住，窗口不会被调整大小以适应这个标签，所以保持标签的大小低于20-25个字符。

BugTrq.No数

如果设置为真在发行号文本字段中只允许数字。一个例外是逗号，所以你可以逗号分隔几个号码。有效值是真实.如果没有定义，*真*假设

#### 用正则表达式发出数字

在方法上与正则表达式TortoiseGit不显示单独的输入字段，但标记用户输入的日志消息的一部分，这是由问题跟踪器识别的。这是在用户编写日志消息时完成的。这也意味着bug ID可以在日志消息的任何地方！这种方法更灵活，是TurtoSigEIT项目本身使用的方法。

洛格雷克斯

此配置激活了错误跟踪系统。正则表达式模式。它包含单个正则表达式，或者包含由换行符分隔的两个正则表达式。

如果设置了两个表达式，则将第一表达式用作预过滤器，以找到包含bug ID的表达式。然后，第二表达式从第一正则表达式的结果中提取裸bug IDS。这允许您使用bug ID和自然语言表达式的列表，如果您愿意的话。例如，您可能会修复几个错误并包括这样的字符串：“这一变化解决了问题23、24、25等问题。“

如果您想在日志消息中使用上面表达式中的bug ID，可以使用以下正则表达式字符串，这是TurtoSigEIT项目使用的字符串：[Ii]ssues?:?(\s\*(,|and)?\s\*#\d+)+和(\d+)

第一个表达式被剔除“第23、第24、第25类问题“从周围的日志消息。第二个正则表达式从第一个正则表达式的输出中提取简单的十进制数，所以它将返回。“二十三“，“二十四“和“二十五“作为bug ID使用。

打破第一个正则表达式，必须从单词开始。“问题“可能是资本化的。这是可选的。“S“（不止一个问题）和可选的结肠。接着是一个或多个组，每个组具有零个或多个前导空格，可选逗号或“和“还有更多的可选空间。最后是强制性的“γ“和一个强制十进制数

如果只设置一个表达式，则必须在ReGEX字符串组中匹配裸bug IDS。例子：[Ii]ssue(?:s)? #?(\d+)这种方法是由几个问题跟踪器，如TRAC所需要的，但它很难构造正则表达式。如果您的问题跟踪器文档告诉您，我们建议只使用此方法。

如果你不熟悉正则表达式，请看下面的介绍。[HTTPS://E.WiKiTo.Org/Wik/正则表达式](https://en.wikipedia.org/wiki/Regular_expression)以及在线文档和教程<http://www.regular-expressions.info/>.

如果两者都BugTrq:消息和BugTrq:LoReGeX属性设置为洛格雷克斯优先

### 小贴士

即使你没有一个问题跟踪器和一个预先提交的钩子解析你的日志消息，你仍然可以使用它来将日志消息中提到的问题转换为链接！

即使你不需要链接，问题号也会出现在日志对话框中的一个单独的列中，这样就更容易找到与某个特定问题相关的变化。

#### 基于分层Git配置的追踪器提供者设置

这是一个分层的Git配置，用于将问题跟踪器插件与项目关联，而不是与特定的目录路径相关联。这样的设置更便于携带。若要部署设置，请将其设置为项目级别并提交TGITCONFIG.

蒲公英

这是32位问题跟踪器插件的GUID。

PogTrq.PvaveRuuID64

这是64位问题跟踪器插件的GUID。

普罗维尔帕拉姆斯

这是问题跟踪器插件的参数字符串。

这个问题跟踪器集成不限于TurtoSigeIT；它可以与其他客户端（例如TurtoSeVN）一起使用。欲了解更多信息，请阅读全文[问题跟踪器集成规范](https://gitlab.com/tortoisegit/tortoisegit/blob/master/doc/issuetrackers.txt)在TurtoSigeIT源库中。（[所谓的“TurtoSigeIT是免费的”！“](https://tortoisegit.org/docs/tortoisegit/tgit-preface-source.html)说明如何访问存储库

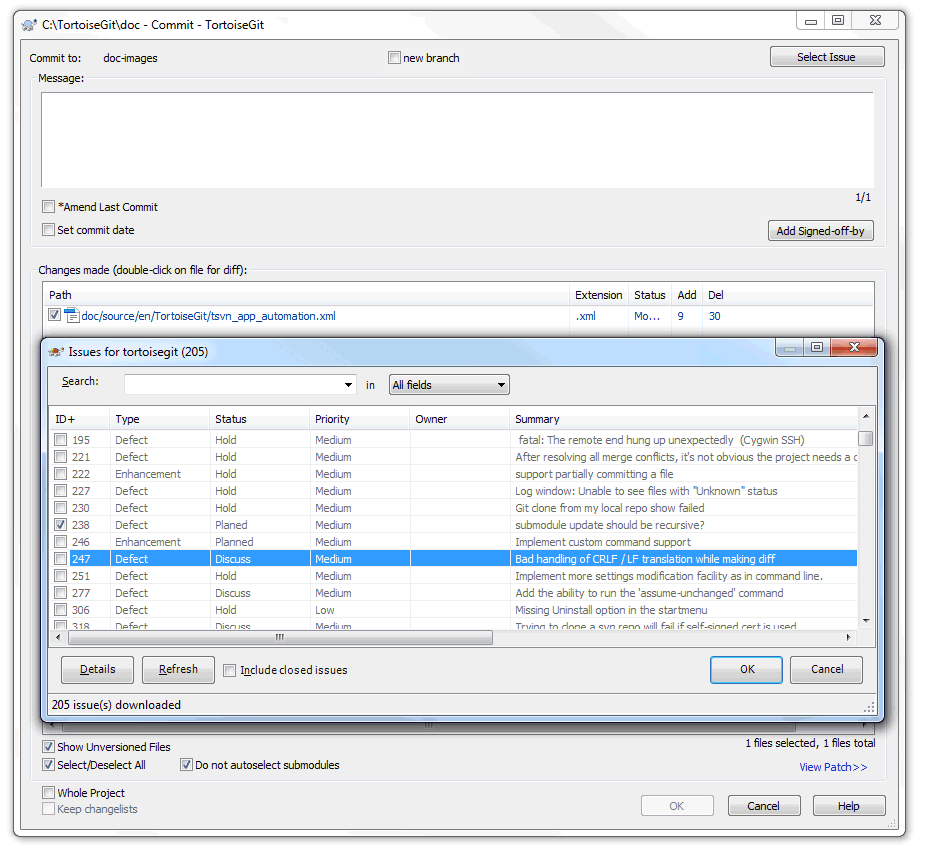
### 从问题跟踪器获取信息

前一节涉及将问题信息添加到日志消息中。但是如果你需要从问题跟踪器获取信息呢？提交对话框有一个Windows COM接口，它允许集成一个可以与您的跟踪器对话的外部程序。通常，您可能希望查询跟踪器以获取分配给您的开放问题的列表，以便您可以选择在此提交中正在处理的问题。

任何这样的接口当然对系统都是非常特定的，所以我们不能提供这个部分，并且描述如何创建这样的程序超出了本手册的范围。接口定义和示例程序可以从小事文件夹中[龟裂仓库](https://gitlab.com/tortoisegit/tortoisegit/tree/master/contrib/issue-tracker-plugins). （[所谓的“TurtoSigeIT是免费的”！“](https://tortoisegit.org/docs/tortoisegit/tgit-preface-source.html)说明如何访问存储库。也给出了API的摘要。[附录B，*总线接口*](https://tortoisegit.org/docs/tortoisegit/tgit-ibugtraqprovider.html)C语言中的另一个（工作）示例插件[古特尔](https://code.google.com/p/gurtle/)它实现所需的COM接口与[谷歌码](https://code.google.com/hosting/)问题跟踪器

为了说明的目的，让我们假设您的系统管理员已经为您提供了一个已安装的问题跟踪器插件，并且您已经设置了一些工作树来在TortoiseGit的设置对话框中使用插件。当您从已被分配了插件的工作树打开提交对话框时，您将看到对话框顶部的一个新按钮。

**图2.68。示例问题跟踪器查询对话框**



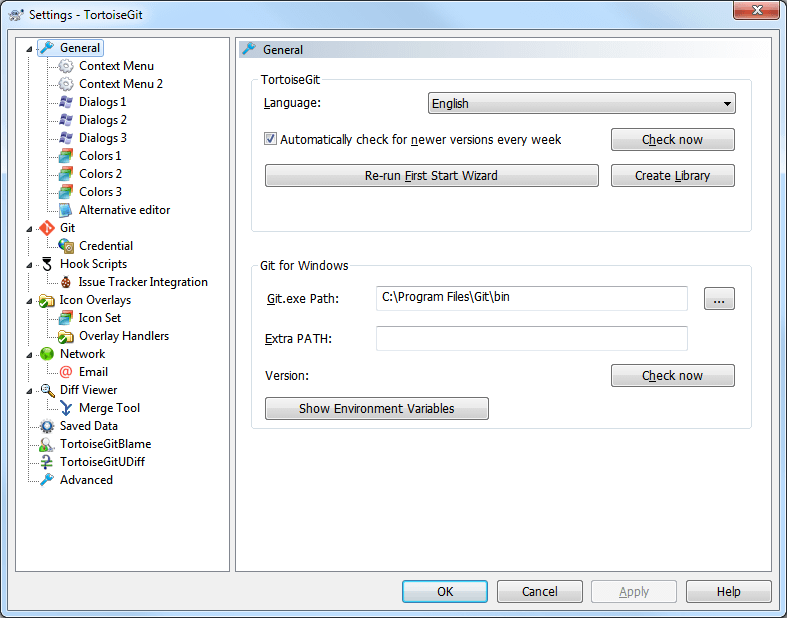
在这个例子中，您可以选择一个或多个打开的问题。插件可以生成特殊格式的文本，并将其添加到日志消息中。

## TortoiseGit设置

要找出不同的设置是什么，只需在编辑框/复选框上留下鼠标指针一秒钟……一个有用的工具提示将会弹出。

### 一般设置

**图2.69。设置对话框，Page将军**



此对话框允许您指定首选语言和特定于Git的设置。

语言

选择用户界面语言。你还期待什么？仅列出已安装的LanguagePacks语言。你可以下载语言包[TurtoSigeIT下载页面](https://tortoisegit.org/download)或[帮助翻译](https://tortoisegit.org/translate).

每周自动检查更新版本

如果被选中，TurtoSigeIT将每周联系它的下载站点，看看是否有一个新版本的程序可用。使用**现在检查**如果你马上想要答案。新版本将不会下载；您只需接收一个信息对话框，告诉您新版本可用。

创建库

在Windows 7上，您可以创建一个库，将工作副本分散在系统的各个地方。

EXE路径

托尔图瓦吉特需要知道EXE用于它的操作。输入完整路径EXE在这里

### 注意安全

不能将GIT.EXE标记为在高架模式下运行（即“以管理员身份运行”或以任何兼容模式运行）。

### 注意安全

有一个[MISSGIT/GIT中的Windows问题](https://github.com/msysgit/msysgit/issues/103)Git for Windows提供两个EXE文件（文件夹中的一个）箱子文件夹中的一个CMD）确保EXE路径指向箱子-在Windows安装文件夹的Git文件夹中。

### 注意安全

如果您不使用Git用于Windows，请参阅下面的“CygWin Git”和“Mys2 Git”的章节，因为这里需要特殊设置。

一般来说，在TortoiseGit没有CygWin或Mys2 Git的官方支持。TurtoSigit开发人员只使用Git用于Windows。然而，错误报告是受欢迎的。

### 小贴士

为了调试问题，您可以打开TurtoSigEIT高级设置，并将“Debug GuutStudio字符串”设置为“true”。[“高级设置”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-registry)）开始捕获调试输出。然后启动TurtoSigEIT设置，点击现在检查并观察调试消息

额外路径

如果您的Git安装需要在路径环境变量中添加一个额外的条目，您可以在这里输入它，当TotoSeGit启动时，它将自动添加到路径环境变量中。

这是特别需要的，如果你安装了Mysgit的开发者版本（如果你想用Git）破解Git的话，“完整安装程序（自包含）”。Msysgit-Fullinstall在这种情况下，有必要[Msysgit-Install-Path]-Mingw..-文件夹在路径上（即进入额外路径文本框）以执行GIT.EXE。

通常，当您启动GIT.EXE时，您可以看到是否需要这样做。[Msysgit-Install-Path]-Mingw..文件夹中，您得到一个消息框，表示缺少DLL。

西文吉特

如上所述：在TurtoISeGit中没有对CygWin Git的官方支持（不要为Windows软件包提供Git！）TurtoSigit开发人员只使用Git用于Windows。然而，错误报告是受欢迎的。如果你真的想使用它，下面就是你必须执行的步骤：

1）选择[ CygWin安装路径] \bin-folder as git.exe folder.

2）配置主页Windows中的环境变量，因此CygWin和TortoiseGit使用相同的主目录和全局Git配置。在这里使用正常的Windows符号（例如，C:\用户\用户名）。默认情况下，TurtoSigeIT使用通常位于下面的Windows home目录。C:用户CyGWin使用自己的家乡目录[ CygWin安装路径] \家.

3）配置AutoCrLf，这是必要的，因为TortoiseGit和CygWin Git有不同的缺省值。CygWin Git中的默认值是真.

4）去龟[“高级设置”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-registry)并设置齐格温哈克到真为了激活CygWin的解决方案。

5）重新启动

MSYS2 GIT

如上所述：在TurtoSeigit中没有官方支持Mys2 Git（不允许使用“Windows封装的Git！”）。TurtoSigit开发人员只使用Git用于Windows。然而，我们欢迎。如果你真的想使用它，下面就是你必须执行的步骤：

1）选择[MysS2安装路径] \UR\bin-folder as git.exe folder.

2）配置主页Windows中的环境变量，使MyS2和TortoiseGit使用相同的主目录和全局Git配置。在这里使用正常的Windows符号（例如，C:\用户\用户名）。默认情况下，TurtoSigeIT使用通常位于下面的Windows home目录。C:用户MyS2使用位于下面的自己的主目录。[ Mys2安装路径] \家.

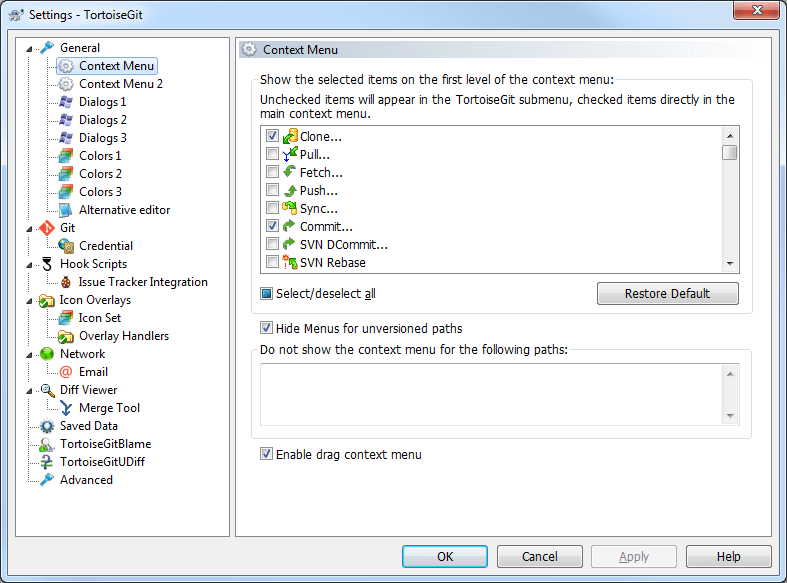
3）配置AutoCrLf，这是必要的，因为TortoiseGit和MyS2 Git可能有不同的缺省值。

4）去龟[“高级设置”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-registry)并设置MSES2HACK到真为了激活MyS2的解决方案。

5）重新启动

#### Context Menu Settings

**图2.70。设置对话框，上下文Menu Page**



此页允许您指定TurtoSigEIT菜单条目中的哪一个将显示在主菜单（第一级）上，哪些条目将出现在TurtoSigEIT子菜单中。默认情况下，大多数项目未选中并出现在子菜单中。如果要隐藏特定条目，请参见[称为“菜单2设置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-look-feel-extended).

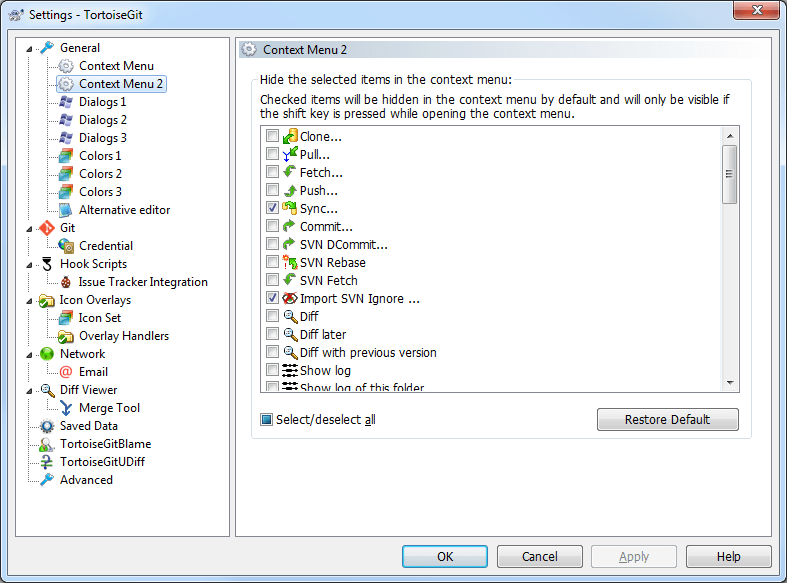
大多数情况下，除了Git版本控制下的文件夹外，您不需要TurtoSigit菜单。对于非版本文件夹，您只需要在进行签出时需要菜单。如果您检查选项为非版本路径隐藏菜单TortoiseGit不会将其条目添加到非版本文件夹的菜单中。但是在版本化文件夹中，所有条目和路径都添加了条目。您可以通过保存**换档**key down while showing the context menu.

如果你的电脑上有一些路径，你根本不想让TortoiseGit的菜单出现，你可以把它们放在底部的框中。

If you right click and drag folder/file in Windows Explorer, a context menu will be shown when you drop. It provides some TortoiseGit actions. You can uncheck**启用拖动菜单**以防止不小心点击TotoSigeIT动作。

#### Context Menu 2 Settings

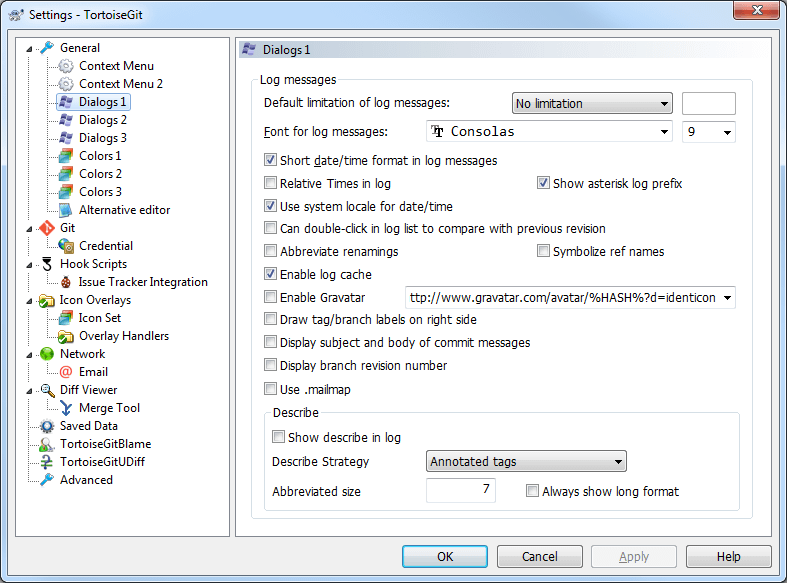
**图2.71。设置对话框，菜单2**



此页允许您指定默认情况下隐藏的TurtoSigEIT菜单条目中的哪一个。选定项目只有在保存时才可见。**换档**关键在**右击**（这是所谓的扩展菜单，请不要将它与TurtoSigeIT子菜单混合，这也是可配置的。[称为“菜单设置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-look-feel)）此配置帮助您根据需要减少菜单项的数量。

#### TurtoSigIt对话框设置

**图2.72。设置对话框，Dialogs Page**



这个对话框允许你按照TortoiseGit喜欢的方式配置一些对话框。

Font for log messages

选择用于在修订日志对话框的中间窗格中显示日志消息本身的字体面和大小，以及在提交对话框中编写日志消息时所使用的字体和大小。

Short date / time format in log messages

如果标准长消息占用了屏幕上太多的空间，请使用短格式。

显示星号日志前缀

在日志对话框中插入星号作为日志消息的前缀。

申请-托普订单

通常，日志项/提交按提交日期的降序排序。-Topo命令使提交出现在拓扑顺序中（即子代提交在其父母之前显示）。不使用此选项，可能会打破日志对话框中的图表。但是，此选项较慢，因为所有日志项都必须在显示它们之前进行处理。

可以双击日志列表与以前的版本进行比较

如果您经常发现自己比较日志对话框顶部窗格中的修订，则可以使用此选项在双击中允许该操作。默认情况下，它不能启用，因为获取DIFF通常是一个漫长的过程，许多人宁愿避免意外的双击之后的等待，这就是为什么默认情况下无法启用该选项的原因。

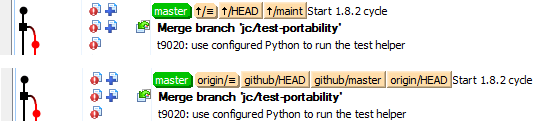
缩写重命名

Normally renamed files are listed as "long/path/for/file.txt (from long/path/to/file.txt)". If you check this option renamed files will be listed in a shorter format ("long/path/{to => for}/file.txt"), however, this abbreviated format might be harder to understand.

象征裁判名称

在ReF标签上显示符号以替换部分ReF名称以使它们更小。如果启用此选项，将应用下面的描述和示例。如果只有一个远程，则一个上箭头符号将代替每个远程分支的远程名称部分。如果远程分支是本地分支的上游，则一个等效符号将代替远程分支的分支名称部分。

**图2.73。象征裁判名称的例子**



启用日志缓存

加载/保存.git文件夹中的日志缓存（TurtoISeig.DATA，TotoSigeIt.index），以提高后续使用日志列表的性能。如果禁用此选项，则不会读取或写入缓存文件。启用默认设置。

启用重力仪

显示日志对话框中提交的作者的GravaTar图像。URL是可定制的，因此您可以指定服务器支持的更多选项，或者使用您自己的化身服务器。默认URL是http://www.gravatar.com/avatar/%HASH%?d=identicon目前，支持的参数是页：1，这是MD5邮件哈希。若要指定默认图像，请添加d=参数，例如http://www.gravatar.com/avatar/%HASH%?d=identicon见[重力：图像请求](http://en.gravatar.com/site/implement/images/)参数列表

在右侧画标签/枝标签

显示提交消息后的标签/分支标签。

显示分支修订号

在日志对话框的提交消息字段中为每个选定的提交显示一个所谓的“分支修订号”。分支修订号是通过调用来计算的。GIT-Rev列表-计数-第一亲本[Sa1]并表示开始时间与选定提交之间的提交数。这个数字不能保证是唯一的，特别是如果你改变历史（例如使用ReBASE）或者同时使用多个分支。它可以看到“独特的”每一个分支，如果你不改变它的历史（例如，通过重新设置，重置），并只提交或合并其他分支上。这个数字只显示给第一个父提交，而不是用于非快速合并的提交（这里可能出现重复的数字）。见<https://gcc.gnu.org/ml/gcc/2015-08/msg00148.html>和[https://gitlab.com/tortvoisegit/tortoisegit/merge/uu requests/1](https://gitlab.com/tortoisegit/tortoisegit/merge_requests/1)更多细节

在日志中显示描述

在日志对话框中显示上述提交消息。例如，v0.21.0-589-gdeadc43指提交deadc43这是标签前面的589。v0.21.0. 注意：如果提交远离标签，则可能需要更长的时间运行。

描述策略

确定参考查找策略：可用选项：带注释的标签、所有标签、所有参考文献。默认策略仅是注释标签。如果存储库使用轻量级标签来标记发布，则选择所有标签。

描述缩写大小

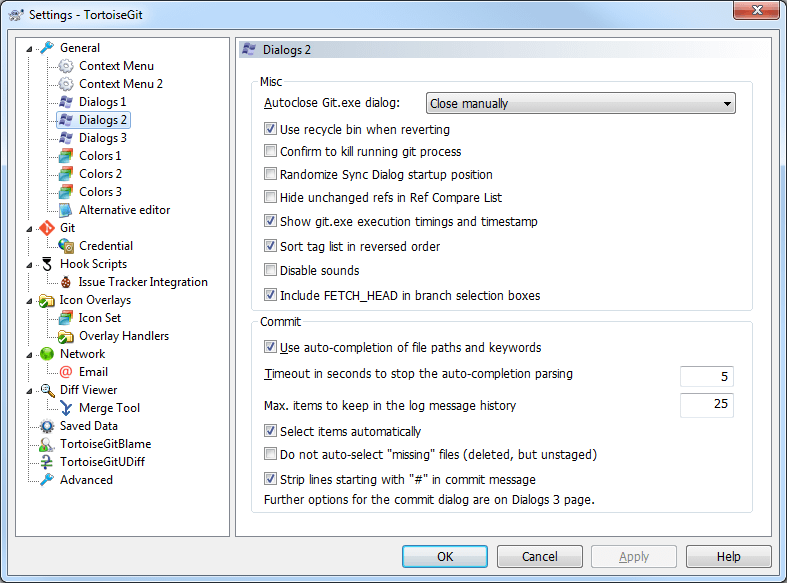
在描述中显示的缩写提交ID的字符数。默认值为7。

描述总是显示长格式

即使使用更短的名称，是否使用长格式。例如，提交时g28f087c有标签v0.21.0它仍然显示长格式v0.21.0-0-g28f087c而不仅仅是v0.21.0.

#### TurtoSigIt对话设置2

**图2.74。设置对话框，对话框2页**



这个对话框允许您按照TortoiseGit喜欢的方式配置更多的对话框。

进度对话框

当操作完成时，TurtoSigEIT可以自动关闭所有进度对话框。此设置允许您选择关闭对话框的条件。默认（推荐）设置是手动关闭它允许您查看所有消息并检查发生了什么。但是，如果不发生重大更改，您可能会决定忽略某些类型的消息，并自动关闭对话框。

如果没有其他选项，自动关闭如果GIT.EXE干净地退出（即没有发生错误），并且在进度对话框中没有其他选项，则会关闭对话框。

自动关闭，如果没有错误如果GIT.EXE退出0错误代码，则总是关闭对话框。

回收时使用回收站

当您返回局部修改时，您的更改被丢弃。TurtoSeigIT给你一个额外的安全网，通过将修改后的文件发送到回收站，然后带回原始副本。如果您希望跳过回收站，请选中此选项。

杀灭运行GIT过程的确认

启用后，如果您关闭进程对话框或同步对话框与运行Git进程，您将被要求确认之前杀死它。这避免了意外关闭运行Git进程的对话框。

随机化同步对话框启动位置

启用时，同步对话框的启动位置将随机化。如果你同时打开许多同步对话框和按拉按钮，你可以很容易地按下拉按钮在任何以前的同步对话框，如果它完成，并成为前景。

在REF比较列表中隐藏未修改的参考文件

启用后，RIFF比较列表中将不显示未修改的参考文件，因此可以将焦点放在更改的参考文件上。当前，这个列表在同步对话框REF列表选项卡中。

显示GIT.EXE执行时间和时间戳

启用后，GIT.EXE执行时间和时间戳将附加在进度消息的末尾。

按顺序排序标签列表

启用时，标签列表按相反顺序排序。这是因为新版本更有用。例如，Exchange对话框允许在启用该选项时选择最新标签。

使用自动完成文件路径和关键字

提交对话框包括一个用于解析提交的文件名列表的工具。当您键入列表中一个项目的前3个字母时，自动完成框弹出，您可以按Enter来完成文件名。选中框以启用此功能。

以秒为单位超时停止自动完成解析

如果有很多大文件要检查，自动完成解析器会相当慢。此超时停止了提交对话框被占用太长时间。如果缺少重要的自动完成信息，则可以延长超时时间。

Max. items to keep in the log message history

当您在提交对话框中键入日志消息时，TurtoSigeIT将其存储起来以便以后再使用。默认情况下，它将保留每一个存储库的最后25条日志消息，但您可以在此自定义。如果您有许多不同的存储库，您可能希望减少这一点以避免注册表的填充。

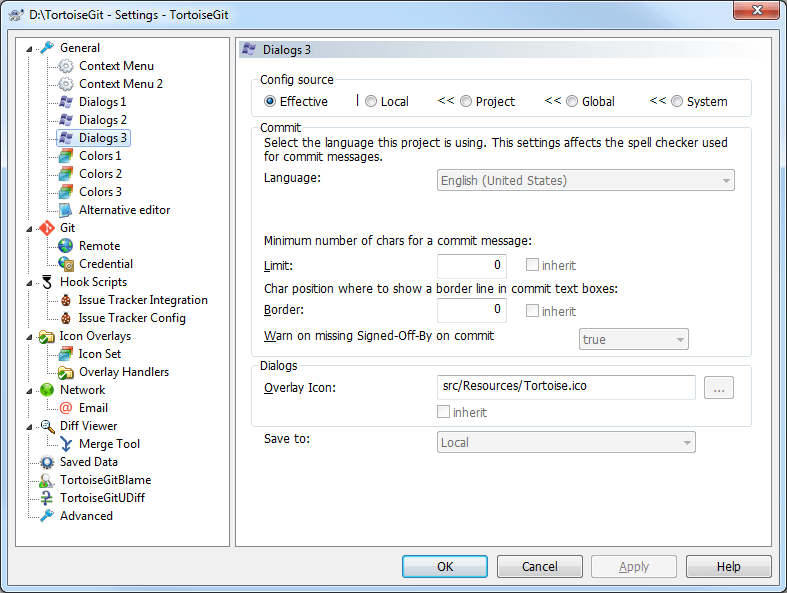
请注意，此设置仅适用于键入此计算机上的消息。它与日志缓存无关。

自动选择项目

提交对话框中的正常行为是为了自动修改提交的所有修改（版本化）项。如果您希望从没有选择任何东西开始，并手动选择要提交的项目，请选中此框。

#### TurtoSigIt对话设置3

**图2.75。设置对话框，对话框3页**



这个对话框允许你按照TortoiseGit喜欢的方式配置一些对话框。这第三页主要影响提交对话框和存储在Git配置文件中的设置。

### 重要的

如果输入/存储数据有问题请参见[称为“分层Git配置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-hierarchical).

语言

TotoSigeIT可以使用OpenOffice和Mozilla所使用的拼写检查器模块。如果您安装了这些属性，则该属性将决定使用哪个拼写检查器，即在哪种语言中，您的项目的日志消息应该被写入。这个投影语言CONFIG键设置拼写检查引擎在输入日志消息时应该使用的语言模块。您可以在本页找到您的语言的值：[语言标识符](https://docs.microsoft.com/windows/desktop/intl/language-identifier-constants-and-strings).

用十进制输入这个值。例如，英语（美国）可以输入为一千零三十三.

使用- 1禁用拼写检查器

极限

LogimStudio设置提交的日志消息的最小长度。如果输入的消息比这里指定的短，则禁用提交按钮。此功能对于提醒您为每个提交提供适当的描述性消息非常有用。如果未设置此属性，或值为零，则允许空日志消息。

边界

LogWITH标记用于在断线之前需要以最大宽度（通常为72个字符）格式化日志消息的项目。将此属性设置为非零将放置一个标记来指示最大宽度并执行线环绕。注意：如果您为日志消息选择了固定宽度字体，则此功能仅能正常工作。

签署承诺后发出警告

温哥华与需要在提交消息中逐行签名的项目一起使用。

覆盖图标

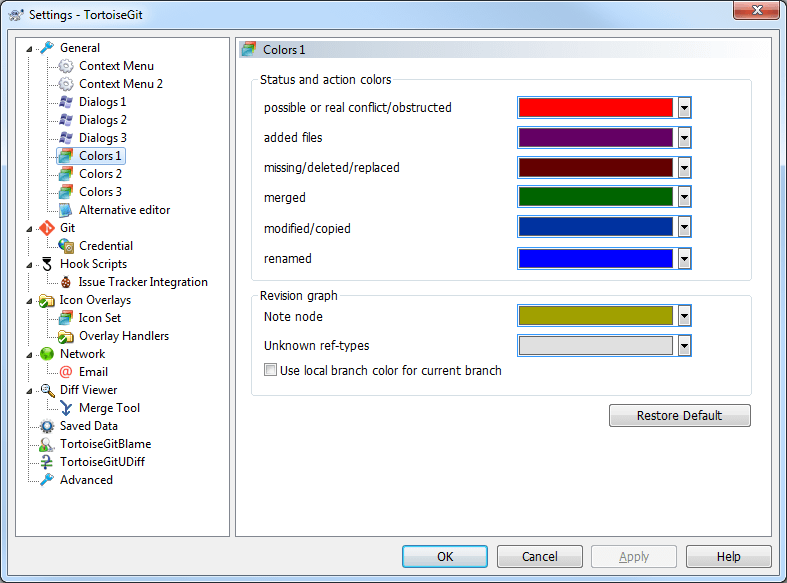
图标与希望在任务栏上显示徽标的项目一起使用，以便在不同项目的多个TurtoSigEIT应用实例同时运行时更容易识别。



如果图标不是16x16px的大小，它将被自动缩放。支持格式为ICO、PNG、JPG、GIF、BMP。如果该项目没有包含图标，您可以自己找到一个，放入.git文件夹，并在本地配置中设置相对路径。例如.git/logo.ico如果你想禁用它，你可以设置图标作为本地配置中的空字符串。当禁用或加载失败时，它将返回到颜色块。注意高级选项石斑鱼应该是3或4，以便使用这个函数。

#### TortoiseGit Colour Settings

**图2.76。设置对话框，Colours Page**



这个对话框允许你按照你喜欢的方式配置TurtoSigIt对话框中使用的文本颜色。

可能的或真实的冲突/阻碍

在更新期间发生冲突，或者在合并期间可能发生冲突。更新是由与版本化的同名的现有版本的文件/文件夹阻塞的。

此颜色也用于进度对话框中的错误消息。

添加文件

添加到存储库中的项

丢失/删除/替换

从存储库中删除的项目，从工作副本中删除，或者从工作树中删除，并用另一个同名文件替换。

合并

从存储库中成功地将更改合并到工作树中而不产生任何冲突。

Modified / copied

添加历史记录，或者复制到存储库中的路径。也用于日志项中包含复制项的条目对话框中。

注释节点

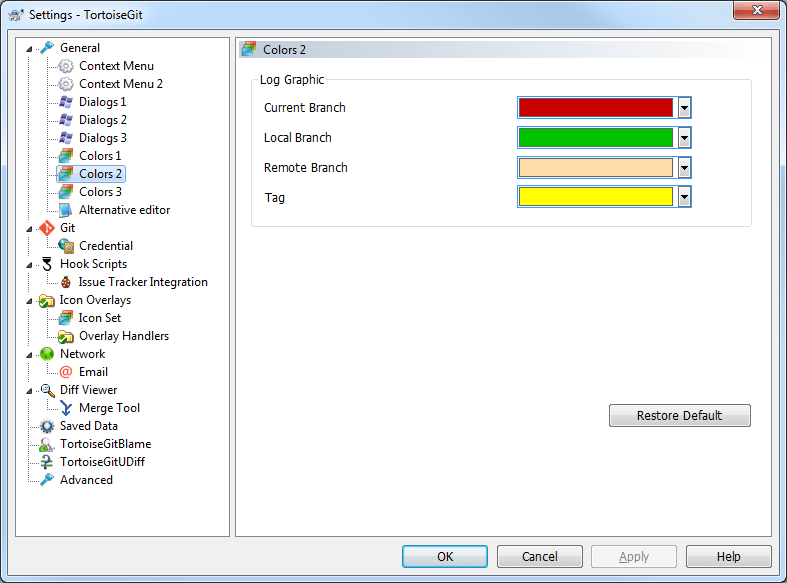
引用Git注释的引用，在引用/注释命名空间下。

为当前分支使用本地分支颜色

在修订图中，为当前分支使用局部分支颜色。您可能不想在修订图中强调本地存储库的当前分支。

#### TortoiseGit Colour Settings 2

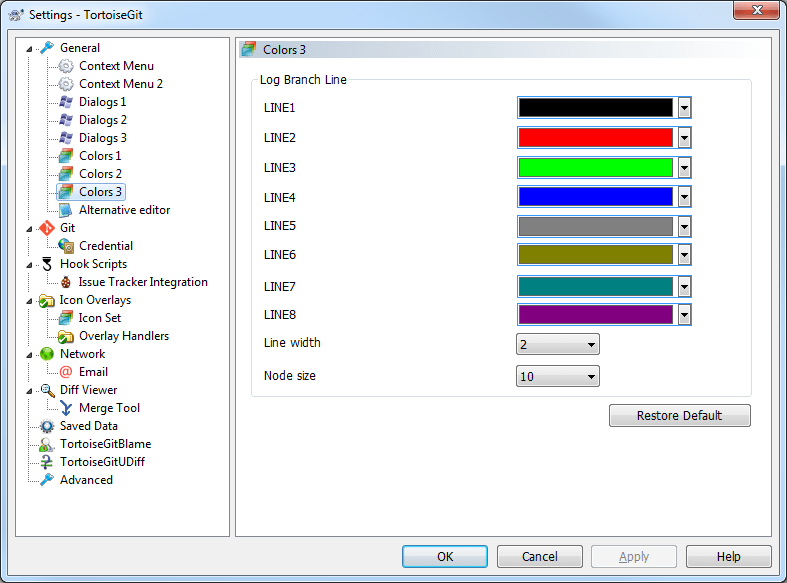
**图2.77。设置对话框，Colours Page**



这个对话框允许你按照你喜欢的方式配置TurtoSigIt对话框中使用的文本颜色。

#### TortoiseGit Colour Settings 3

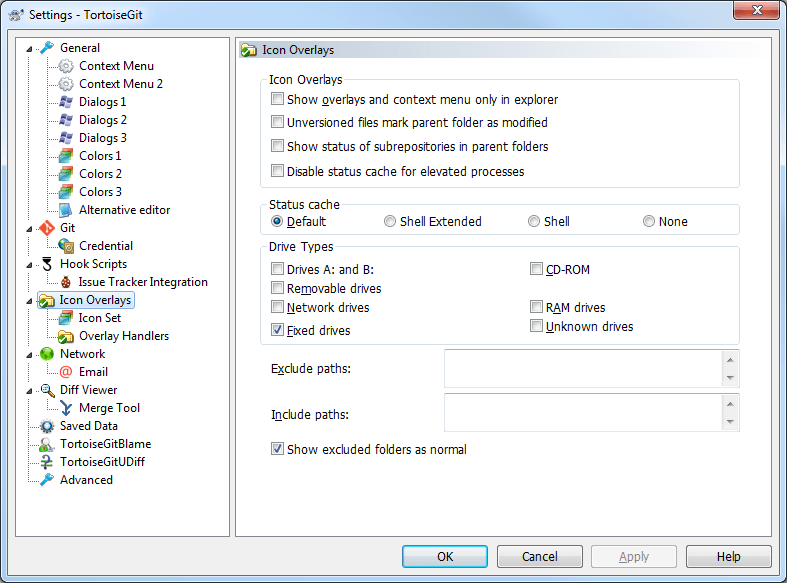
**图2.78。设置对话框，Colours Page**



这个对话框允许您配置TurtoSigeIT日志对话框中使用的图形列的线条颜色、线宽和节点大小，就像您喜欢它们一样。

### Icon Overlay Settings

**Figure 2.79. The Settings Dialog, Icon Overlays Page**



此页允许您选择TurtoSigeIT将显示图标覆盖的项目。

默认情况下，覆盖图标和菜单将出现在所有打开/保存对话框以及Windows资源管理器中。如果你想让他们出现只有在Windows Explorer，check the仅在Explorer中显示覆盖和菜单包装

未被忽略的项目和未被版本化的项目通常不被给予覆盖。如果要在这些情况下显示覆盖，只需检查框。

如果文件夹包含未版本化的项，则还可以选择标记为已修改的文件夹。这可能会提醒您，您已经创建了尚未被版本化的新文件。此选项仅在使用默认status cache option (see below).

由于获取工作树的状态需要相当长的一段时间，所以TortoiseGit使用缓存来存储状态，这样当显示覆盖时，Explorer不会被占用太多。您可以根据您的系统和工作树大小选择使用哪种类型的缓存TurtoSigeTe:

默认

在一个单独的进程中缓存所有状态信息（*TGITCACH.EXE*）该过程监视所有驱动器的更改，并且如果工作树内的文件被修改，则再次获取状态。进程运行的优先级最低，因此其他程序不会因为它而被占用。这也意味着状态信息不是实时但是覆盖层需要改变几秒钟。

优点：覆盖物递归地显示状态，即如果修改了工作树深处的文件，所有到工作树根的文件夹也将显示修改后的覆盖。由于进程可以向shell发送通知，所以在左树视图上的覆盖也经常改变。

缺点：即使你不在项目上工作，这个过程也会持续运行。它也使用大约10-50MB的RAM取决于工作树的数量和大小。从版本1.7.0到1.7.12TGITCHILL没有检查文件的内容，它只是检查了最后修改时间与存储在Git索引文件中的时间。从1.7.13TGITCHILL开始，现在也默认检查文件的内容。如果您想恢复旧的行为，您可以禁用通过设置对话框->高级检查内容，并设置TGITCaseCeCuthCractMax Stand到“0”。

外壳扩展

缓存是直接在shell扩展DLL内完成的。每次导航到另一个文件夹时，再次获取状态信息（递归地）。

优点：能显示状态实时.

缺点：只有一个文件夹被缓存，对于大的工作树，在Explorer中显示一个文件夹要比默认的缓存或shell模式花费更多的时间。shell变体只显示文件系统与Git索引的差异（不包括修订特定信息，例如，如果从索引中删除文件，该文件将显示为未版本化的，但是TGITCHIGH文件将显示为删除，直到您提交此更改）。

壳

缓存直接在shell扩展DLL内完成，但仅针对当前可见的文件夹。每次导航到另一个文件夹时，再次获取状态信息。

优点：只需要很少的内存（大约1兆字节的RAM），并可以显示状态实时.

缺点：因为只有一个文件夹被缓存，所以覆盖不会递归地显示状态。对于大型工作树来说，在Explorer中显示一个文件夹要比默认缓存占用更多的时间。shell变体只显示文件系统与Git索引的差异（不包括修订特定信息，例如，如果从索引中删除文件，该文件将显示为未版本化的，但是TGITCHIGH文件将显示为删除，直到您提交此更改）。

无

通过这个设置，TortoSeGIT在Explorer中根本不获取状态。正因为如此，文件没有得到覆盖，而文件夹只得到一个“正常”覆盖，如果他们是版本控制。没有显示其他覆盖，也没有额外的列可用。

优势：使用绝对没有额外的内存，并不会放慢浏览器在浏览时。

缺点：在Explorer中没有显示文件和文件夹的状态信息。要查看您的工作树是否被修改，您必须使用“修改检查“对话框

默认情况下，覆盖图标和菜单将出现在所有打开/保存对话框以及Windows资源管理器中。如果你想让他们出现只有在Windows Explorer，check the仅在Explorer中显示覆盖和菜单包装

可以强制状态缓存无对于高架过程，通过检查禁用进程的状态缓存包装这是有用的，如果你想防止另一个*TGITCACH.EXE*进程以高特权创建

如果文件夹包含未版本化的项，则还可以选择标记为已修改的文件夹。这可能会提醒您，您已经创建了尚未被版本化的新文件。此选项仅在使用默认status cache option (see below).

下一组允许您选择哪些类的存储应该显示覆盖。默认情况下，只选择硬盘驱动器。你甚至可以禁用所有图标叠加，但在那里的乐趣呢？

网络驱动器可能非常慢，因此默认图标不会显示位于网络共享的工作树上。

USB闪存驱动器似乎是一种特殊的情况，因为驱动器类型是由设备本身识别的。一些显示为固定驱动器，一些作为可移动驱动器。

这个排除路径用来告诉龟龟应该走哪条路不显示图标覆盖和状态列。这是有用的，如果你有一些非常大的工作树，只包含库，你根本不会改变，因此不需要覆盖，或者如果你只想TurtoSigeIT查看特定文件夹。

假定这里指定的任何路径递归应用，所以子文件夹也不会显示覆盖。如果你想排除只有命名文件夹，追加？经过小路

同样适用于包含路径. 除了对于那些路径，即使对于特定驱动类型的覆盖被禁用，也可以通过上面指定的排除路径来显示覆盖。

用户有时会问这三个设置是如何交互的。对于任何给定的路径，检查包含和排除列表，通过目录结构向上搜索直到找到匹配为止。当找到第一个匹配时，遵守包含或排除规则。如果存在冲突，单个目录规范优先于递归规范，则包含优先于排除。

一个例子将有助于：

排除：C：C：发展？开发\\tgIT\Objc:\Deave\tgIT\BiCube：C:\Debug

这些设置禁用C覆盖的图标覆盖，除了c:\develop. 该目录下的所有项目将显示覆盖，除了c:\develop文件夹本身，这是被特别忽略的。也排除了高翻转二进制文件夹。

TGitCache.exe还使用这些路径来限制其扫描。如果希望只在特定文件夹中查找，请禁用所有驱动器类型，只包括特定要扫描的文件夹。

### 排除潜艇驱动器

使用A通常很方便潜艇驱动器访问您的工作树，例如使用命令

subst T: C:\TortoiseGit\doc

然而，这可能导致覆盖不更新，因为TGITCHIGH只会在文件更改时接收到一个通知，通常是针对原始路径。这意味着你的覆盖在潜艇路径可能永远不会更新。

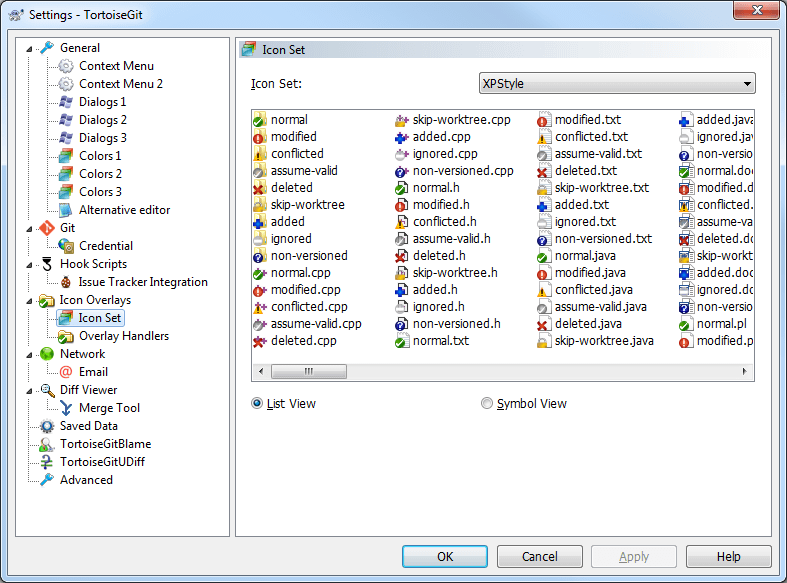
解决这一问题的一个简单方法是将原始路径排除在显示覆盖上，从而覆盖在潜艇路径代替

有时，您将排除包含工作树的区域，这些区域通过扫描和监视更改来保存TGITCHIPS，但您仍然希望看到一个文件夹包含工作树的可视指示。这个将排除的文件夹显示为“正常”复选框允许您这样做。使用此选项，任何排除区域中的工作树文件夹（未检查的驱动器类型，或具体排除）将显示为正常的和最新的，带有绿色复选标记。这提醒您，您正在查看工作树，即使文件夹覆盖可能不正确。文件根本没有覆盖。注意，菜单仍然有效，即使没有显示覆盖。

作为一个特殊的例外，驱动器A和B:从来没有考虑过将排除的文件夹显示为“正常”选项这是因为Windows被迫看在驱动器上，这可能导致在启动资源管理器时延迟几秒钟，即使您的PC有软盘驱动器。

#### 图标集选择

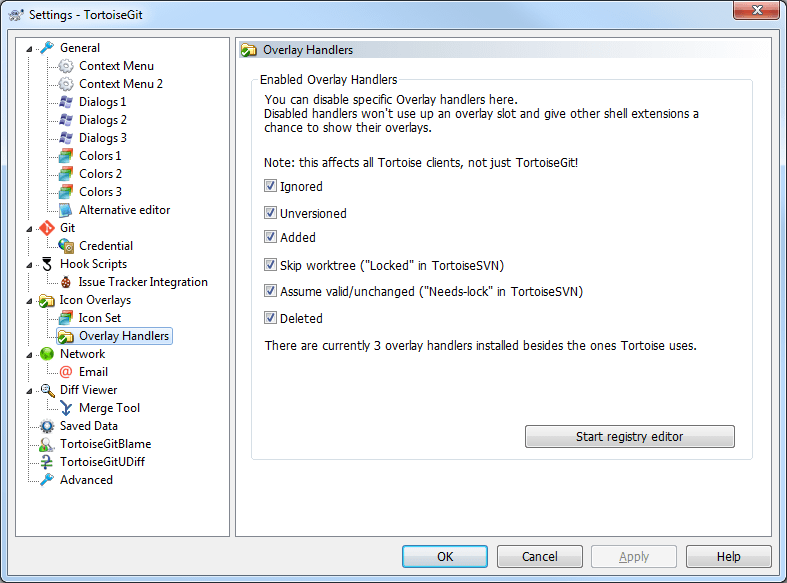
**图2.80。设置对话框，Icon Set Page**



您可以将覆盖图标设置更改为您最喜欢的图标设置。特别是，你可以禁用覆盖你不需要假设有效，跳过工作树，但是其他龟\*工具使用这两个不同的目的。请注意，如果更改覆盖集，则可能需要重新启动计算机以使更改生效。

#### 启用覆盖处理程序

**图2.81。设置对话框，Icon Handlers Page**



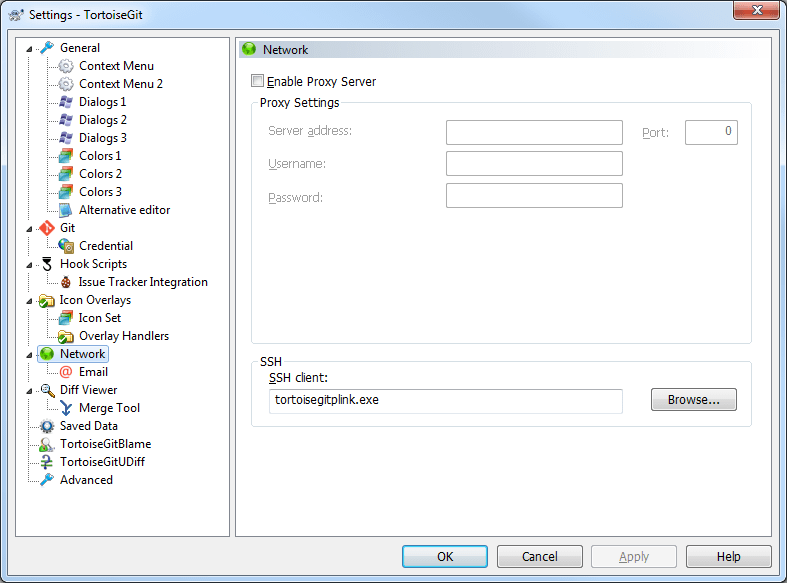
由于可用的覆盖数量受到严重限制，您可以选择禁用某些处理程序，以确保所需的加载程序将被加载。因为TurtoSeigTE使用与其他龟客户端共享的普通龟覆盖组件（例如TurtoSeVN、TurtoSeVS、TortoiseHg），这种设置也会影响这些客户端。

有关图标覆盖如何对应GIT状态和其他技术细节的描述，请阅读[被称为“图标叠加”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-app-internals.html#tgit-app-internals-overlays).

Windows资源管理器可以处理一个固定数量的不同覆盖提供者（15），而TortoiseGit则使用6个（这6个是由TurtoSeSuffer处理的，因此，与TortoSeVN和TortoiseCVS共享）。如果没有正确显示TurtoSigIt图标，这可能是由提供覆盖的其他程序（如Dropbox、OnWin、BoxSync和各种其他）引起的，并以更高的优先级注册。使用启动注册表编辑器按钮用于在注册表编辑器打开的关键处打开注册表编辑器。只需删除或重命名你不需要的或准备好的乌龟有双引号或空格字符，以便在列表中排名第一。欲了解更多信息，请参阅[TotoSigeIT常见问题解答](https://tortoisegit.org/support/faq/#ovlnotshowing).

### 网络设置

**Figure 2.82. The Settings Dialog, Network Page**



在这里你可以配置你的代理服务器，如果你需要一个通过你公司的防火墙。

这里的代理服务器设置只影响Windows的Git（即HTTP和HTTPS协议）。如果使用OpenSSH/腻子/龟（Git）Pink，则必须单独设置代理服务器设置。为了做到这一点，你需要的主要油灰工具，这不是与TortoiseGit装运。最好将代理设置存储到“默认设置”配置中，以便默认情况下应用它。

如果需要设置每个存储库代理设置，则需要使用Git。*配置*配置文件。查阅[被称为“Git配置（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-config(1))更多细节

您还可以指定TotoSigeit应该使用哪个程序来建立与GSH存储库的安全连接，这是使用SSH访问的。我们建议您使用TurtoSigItPiLink .EXE。这是流行的Pink程序的一个版本，它包含在TortoiseGit中，但是它被编译成一个无窗口的应用程序，所以每次验证时都不会弹出一个DOS框。

必须指定可执行文件的完整路径。对于TortoiseGitPlink.exe，这是标准的TortoiseGit bin目录。使用**浏览**按钮来帮助定位它，例如：

“C:\程序文件\ TotoSigeI\bin \ TotoSigeGiTLink .exe”

### 小贴士

如果您想使用GITS为Windows /Mysgit提供的OpenSSH，请输入*SX.EXE*.

没有窗口的一个副作用是没有任何错误消息要去，所以如果认证失败，你只会得到一个类似的消息。“无法写入标准输出“. 为此，我们建议您首先使用标准Pink进行设置。当一切工作时，您可以使用TortoiseGitPLink和完全相同的参数。

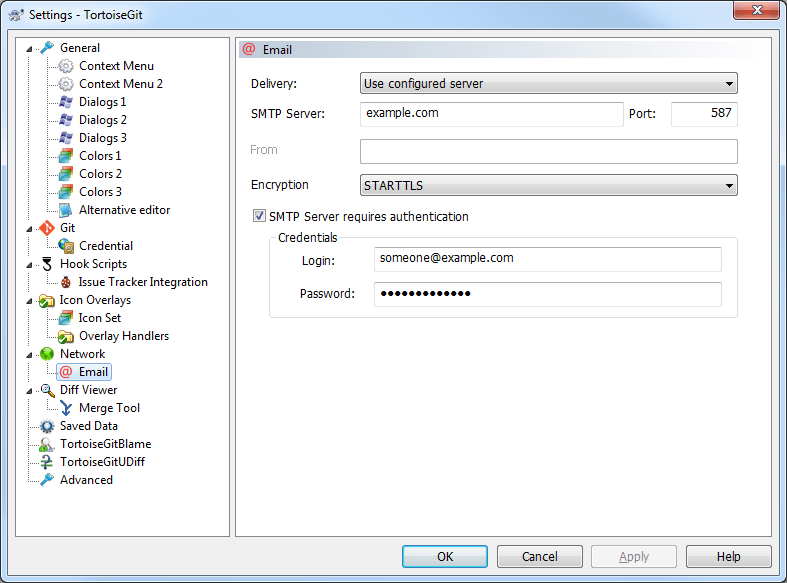
TortoiseGitPlink没有任何自己的文档，因为它只是Pink的一个小变体。从命令中查找命令行参数[PuTTY网站](https://www.chiark.greenend.org.uk/~sgtatham/putty/)

为了避免重复提示密码，您也可以考虑使用密码缓存工具，如PaulAdter。这也可从P腻y网站下载或包含在TurtoSigeIT软件包中。（也见[“认证”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug.html#tgit-dug-general-auth)）

最后，在客户端上设置SSH是一个非平凡的过程，超出了该帮助文件的范围。但是，你可以在TurtoSigeIT常见问题解答中找到一个指南[附录F*SSH/腻子的窍门和窍门*](https://tortoisegit.org/docs/tortoisegit/tgit-ssh-howto.html).

#### 电子邮件设置

**Figure 2.83. The Settings Dialog, email settings**



此页允许您指定配置邮件应该如何发送。

SMTP，直接到目标服务器

当选择此选项时，TortoiseGit直接连接到SMTP服务器（在端口25上），它负责特定的目的地电子邮件地址（ES）。这是TortoiseGit的默认值（除非配置了一些不同的方法）。

### 重要的

如果您的ISP阻塞输出的SMTP连接（端口25），或者您有拨号上网连接，这可能是个问题。在梯子的情况下，一些目的地MTAs可能不接受你的邮件或标记他们为垃圾邮件。

MAPI

当选择此选项时，TurtoSigeIT使用微软消息API（MAPI）来发送邮件。为此，您需要一个具有MAPI能力的邮件客户端（例如雷鸟或Outlook）。

### 重要的

如果你不发送补丁作为附件，你可能需要确保没有自动线缠绕发生。对于雷鸟来说，有一个附加装置。[切换字包](https://addons.mozilla.org/de/thunderbird/addon/toggle-word-wrap/)）可用

使用配置服务器

这是推荐的发送邮件的方式。只需输入与邮件工具（MUA）相同的数据即可。

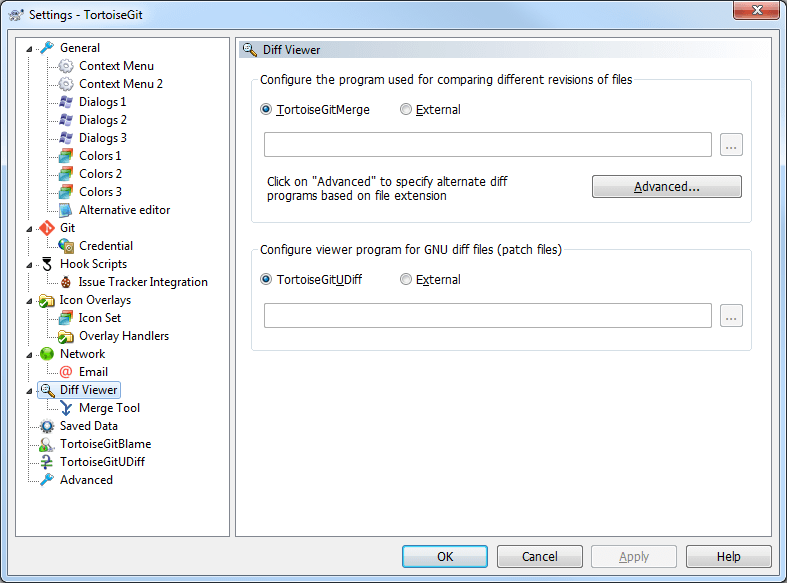
### 外部程序设置

在这里，您可以定义自己的程序，TurtoSigeIT应该使用。默认设置是使用与TortoiseGit一起安装的工具。

阅读[称为“外部差异/合并工具”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-diff.html#tgit-dug-diff-tools)对于人们使用TortoiseGit的一些外部差异/合并程序的列表。

#### 微分观察器

**图2.84。设置对话框，Diff Viewer Page**



外部差异程序可用于比较不同的文件修订。外部程序将需要从命令行获得文件名，以及任何其他命令行选项。TotoSigeIT使用前缀替换参数%. 当它遇到其中的一个时，它将替换适当的值。参数的顺序将取决于您使用的DIFF程序。

%base

没有更改的原始文件

%bname

基本文件的窗口标题

%mine

您自己的文件，随您的更改

-YNAME

文件的窗口标题

%bpath

原始文件的完整路径

%YPATH

文件完整路径

%brev

原始文件的修订，如果可用

%yrev

如果可用的话，修改第二个文件

%WTROOT

工作树路径

窗口标题不是纯文件名。TurtoSeigt将其视为名称，以相应地显示和创建名称。例如，如果你在一个版本123中的文件中使用一个文件在你的工作树中，那么名字将是*filename: revision 123*和*文件名：工作树*

例如，使用EximDIFO PRO：

C:\Path-To\ExamDiff.exe %base %mine --left\_display\_name:%bname --right\_display\_name:%yname

或与KDIF3：

C:\Path-To\kdiff3.exe %base %mine --L1 %bname --L2 %yname

或与WinMerge：

C:\Path-To\WinMerge.exe -e -ub -dl %bname -dr %yname %base %mine

或与Araxis：

C:\Path-To\compare.exe /max /wait /title1:%bname /title2:%yname %base %mine

如果您配置了一个替代的差异工具，您可以访问TortoiseGitMerge和来自菜单的第三方工具。**菜单**渐次**微分**使用主差异工具，以及**换档**+**菜单**渐次**微分**使用辅助差异工具

##### 统一的差异/ GNU差异/补丁文件查看器

代替TurtoSudiDIFF，可以使用统一的差异文件（GNU DIFF或补丁文件）的外部查看器程序。基本上，不需要参数-如果要打开的统一的差异文件将自动附加文件名。如果需要将其作为不同的参数传递，则替换为% 1可以使用。还有参数替换%标题可以通过标题栏中显示的标题（即，差异数据的元数据）。

例如，使用NotoPAD2（用TurtoSigeIT装运）：

NotoPAD2.EXE/S“差异文件”/t%标题

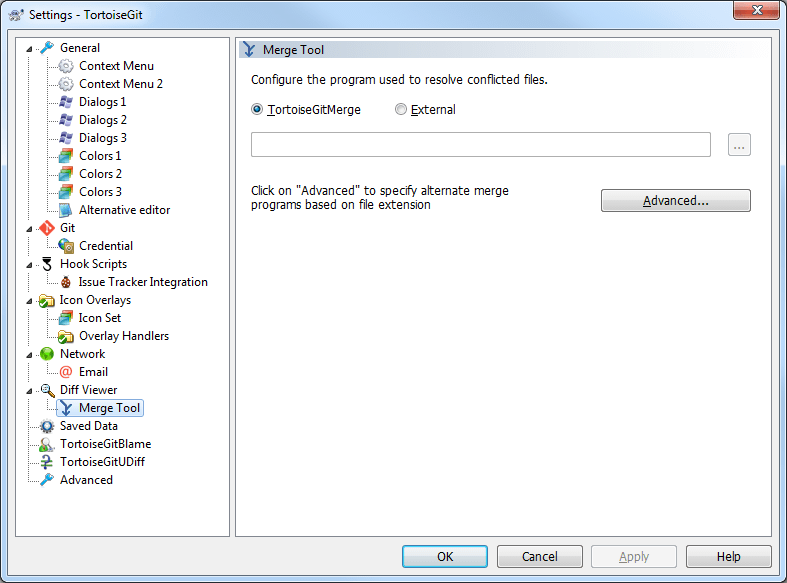
或（等）

NotoPAD2.EXE/S“差异文件”/t%标题%% 1

如果您配置了一个替代的统一的差异工具，您可以访问TurtoSigeTuuDIFF。和来自菜单的第三方工具。当你握住**换档**在打开菜单时，启动辅助统一差异工具。

#### 合并工具

**图2.85。设置对话框，Merge Tool Page**



用于解决冲突文件的外部合并程序。参数替换以与DIFF程序相同的方式使用。

%base

原始文件没有您或其他更改

%bname

基本文件的窗口标题

%mine

您自己的文件，随您的更改

-YNAME

文件的窗口标题

%theirs

在存储库中的文件

%TNAME

存储库中文件的窗口标题

合并%

冲突文件，合并操作的结果

MNAME%

合并文件的窗口标题

%WTROOT

工作树路径

例如，PrimCE合并：

C:\Path-To\P4Merge.exe %base %theirs %mine %merged

或与KDIF3：

C:\Path-To\kdiff3.exe %base %mine %theirs -o %merged --L1 %bname --L2 %yname --L3 %tname

或与Araxis：

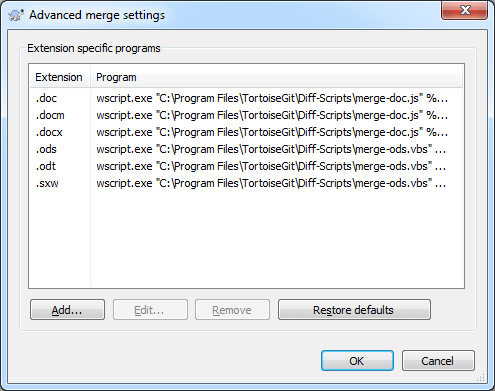
C:\Path-To\compare.exe /max /wait /3 /title1:%tname /title2:%bname /title3:%yname %theirs %base %mine %merged /a2

或使用WiMyc（2.8或更高版本）：

c：path-to-winmerge.exe%merged

#### 差异/合并高级设置

**图2.86。The Settings Dialog，DIFF/MERGE Advanced Dialog**

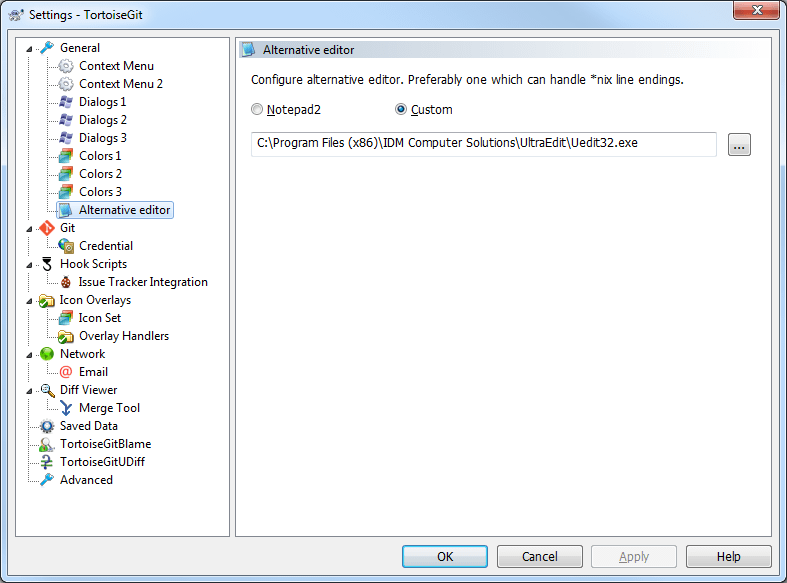


在高级设置中，可以为每个文件扩展定义一个不同的差异和合并程序。例如，你可以把PS图象处理软件联系起来。“微分“程序用于*JPG*文件：-）

若要使用文件扩展名进行关联，需要指定扩展名。使用BMP描述Windows位图文件。

#### 替代编辑器

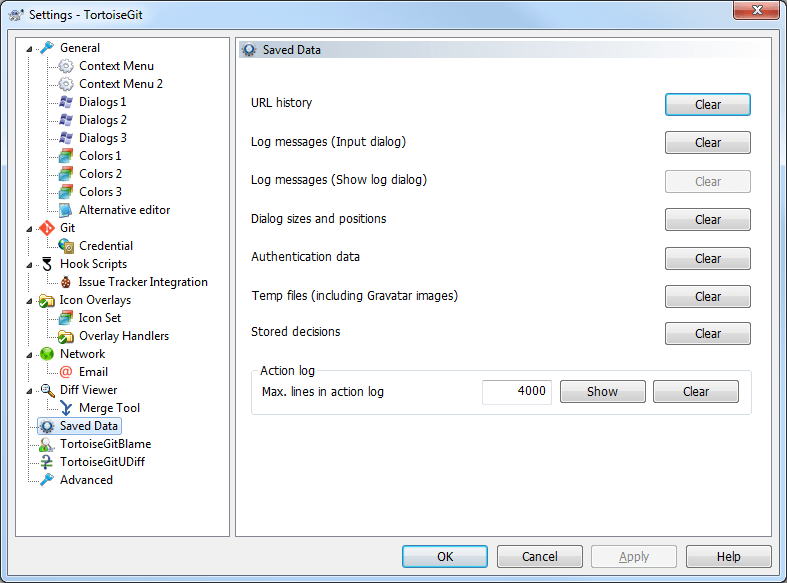
**图2.87。设置对话框，替代编辑器页面**



原始Windows记事本程序在没有标准CR-LF线路末尾的文件上表现不好。然而，许多Git配置文件没有标准的CR-LF线路结束。因为这个TotoSigeIT使用免费的（发货）记事本替换。[Notepad2](http://www.flos-freeware.ch/notepad2.html)默认情况下显示正确的行结束。

### 保存的数据设置

**Figure 2.88. The Settings Dialog, Saved Data Page**



为方便起见，TurtoSigeTE保存了许多你使用的设置，并记得最近你在哪里。如果你想清除数据缓存，你可以在这里做。

网址历史

无论何时签出工作树、合并更改或使用存储库浏览器，TurtoISeGIT都会保存最近使用的URL的记录，并在组合框中提供这些URL。有时这个列表会被过时的URL弄得乱七八糟，所以定期刷新它是很有用的。

如果你想从一个组合框中删除一个项目，你可以把它放在适当的位置。只需单击箭头，将组合框下移，将鼠标移到要移除的项并键入**移位删除**.

Log messages (Input dialog)

TurtoSeGIT存储您输入的最近提交日志消息。这些存储在每个存储库中，因此，如果您访问许多存储库，这个列表可能会变得相当大。

日志消息（显示日志对话框）

TotoSigEIT缓存日志消息，由显示日志对话框获取，以节省下次显示日志时的时间。如果其他人编辑日志消息，并且已经缓存了该消息，则在清除缓存之前，不会看到更改。启用日志消息缓存日志缓存标签

Dialog sizes and positions

许多对话框记住上次使用的大小和屏幕位置。

动作日志

TurtoSeigt将所有写入进程对话框的日志记录下来。例如，当您想检查最近更新命令中发生了什么时，这可能是有用的。

日志文件的长度有限，并且当它增长过大时，丢弃最旧的内容。默认情况下，保留4000行，但您可以自定义该数字。

从这里你可以查看日志文件内容，也可以清除它。

### 吉特

#### 分层Git配置

GIT使用分层配置的概念（参见）。[被称为“Git配置（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-config(1))）即，有多个级别；更高级别的设置覆盖较低级别的值。这个有效的选项卡显示当前作用域的有效值（只读）。

Select any level (e.g.当地-当前存储库设置在本地存储在git /CONFIG中，项目-在//tgITCONFIG存储库中存储的当前存储库的设置，全球-当前用户的设置，系统-为系统的所有用户设置）查看存储在那里的值。

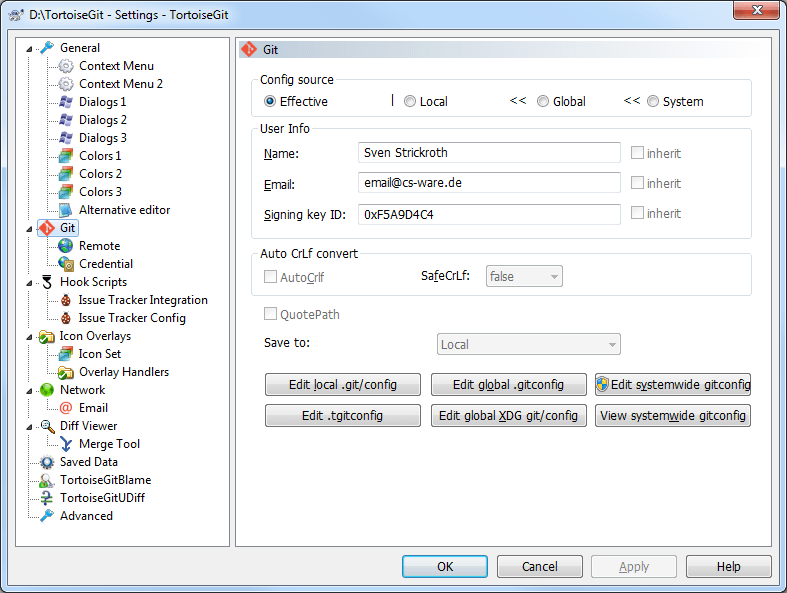
若要更改设置，请选择一个级别，输入值，选择要存储的位置，然后单击**申请**.

### 注意安全

如果要继承更高级别的值，请不要将文本框空置（这意味着将存储一个空字符串），这可能会对真选择继承相反

#### Git配置

**图2.89。设置对话框**



设置Git基本配置

姓名和电子邮件GIT需要正确操作

独立自主如果为真，则当从文件系统读取时，Git将文本文件末尾的CRLF转换为LF，并在写入文件系统时反向转换。变量可以设置为输入，在这种情况下，转换只在文件系统读取时发生，但是文件在LFF结束时用LF写出。基于文件的CRLF属性，或者如果CRLF是未指定的，则基于文件的内容，将文件视为“文本”（即，受AutoLoLF机制的影响）。

保险柜如果TRUE，使GIT检查是否转换CRLF由C.AutoCulf控制是可逆的。Git将验证命令是否直接或间接修改工作树中的文件。例如，提交一个文件，然后签出同一个文件，应该在工作树中产生原始文件。如果这不是当前设置Cy.AutoLoLF的情况，Git将拒绝该文件。变量可以设置为“警告”，在这种情况下，Git只警告不可逆转换，但继续操作。

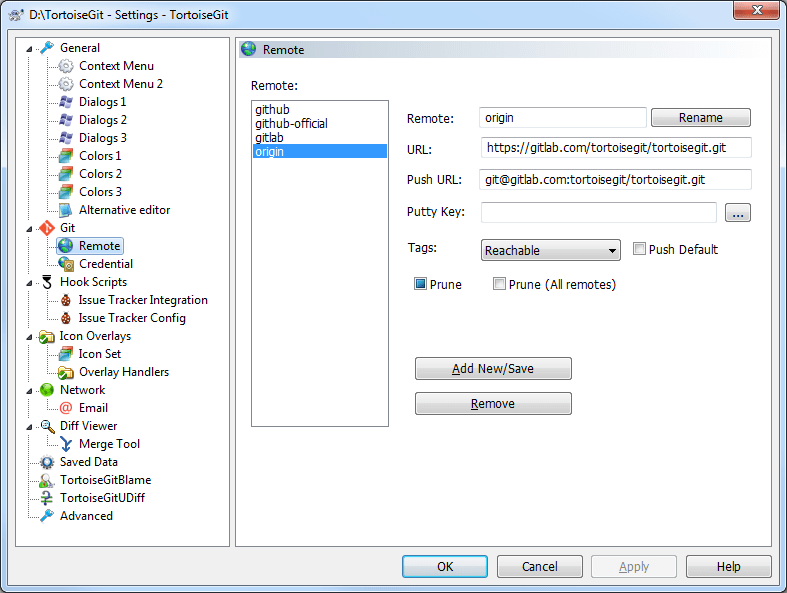
五通控制QueTePATH设置当您有非ASCII文件名时可能会很有趣：请参阅[被称为“Git配置（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-config(1)).

### 重要的

如果输入/存储数据有问题请参见[称为“分层Git配置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-hierarchical).

#### 远程的

**图2.90。设置对话框，Git，远程**



设置Git远程配置

远程的远程的名称，通常是默认的名称。起源.

统一资源定位地址远程的URL。它可以是HTTP/HTTPS/SSH/GIT协议或本地文件系统。

推送网址远程的推送URL。对于某些情况，您不能使用相同的URL来获取和推送（例如，通过无密码GIT协议获取，但通过SSH推送）。否则，将其留空。注意：这不是为分叉工作流设计的。对于分叉工作流，你应该有2个遥控器。格式与URL相同。

油灰钥匙在执行网络操作时加载的PUTY密钥文件。

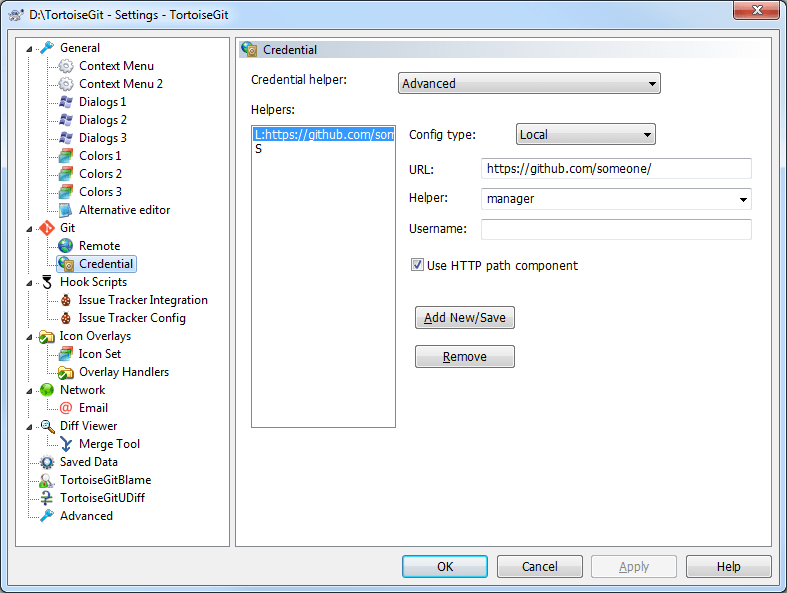
标签这套remote.<name>.tagoptCONFIG，它控制指定远程的默认标签获取行为。可达成的下载可从远程分支头到达的标签（默认行为）。无如果您尚未加入，请点击这里注册。--no-tags）（Git 1.9及以后）全部所有标签和分支都下载了（--tags目录（续）（prior to git 1.9）所有标签只只下载所有标签，但没有下载任何分支（--tags). Use case of全部：总是从Git VSN镜像中获取标签。在树干上不存在颠覆标签，所以这样的标签不能从分支头到达。

默认推送选择这意味着总是推到这个远程（参见）。[被称为“Git配置（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-config(1))) Default is false.

修剪这套remote.<name>.pruneCONFIG，它控制指定远程的远程跟踪分支的默认修剪选项。默认为false。

#### 凭据

**图2.91。设置对话框，git，凭据**



设置简单凭据助手配置

高级如果证书助手配置与任何简单设置不匹配，则使用此设置。如果你选择了其他高级，除了冲突帮助者，所有其他配置密钥credential.\*或credential.\*.\*被移除

无在所有配置级别都没有凭据配置密钥。

wincred - this repository only在本地配置中启用WIFSTD。只有在安装WSPEDD时才能看到此选项。

winstore - this repository only在本地配置中启用了WiSt店。只有当WINStULT安装到当前Windows用户时，此选项才可见。

当前Windows用户WistRD在全局配置中启用。只有在安装WSPEDD时才能看到此选项。

WINStur--当前Windows用户WiSturn在全局配置中启用。只有当WINStULT安装到当前Windows用户时，此选项才可见。

所有Windows用户在系统配置中启用WSPITD。只有在安装WSPEDD时才能看到此选项。

高级证书助手配置

配置类型无论是本地的、全局的还是系统配置的。

统一资源定位地址基于URL模式定义特定于上下文的配置。默认情况下，路径组件不被视为不同的上下文。

帮手选择凭证帮助程序。在TortoiseGit中预先定义WSPACD和WSt存储。可以使用其他证书帮助者或额外的选项。

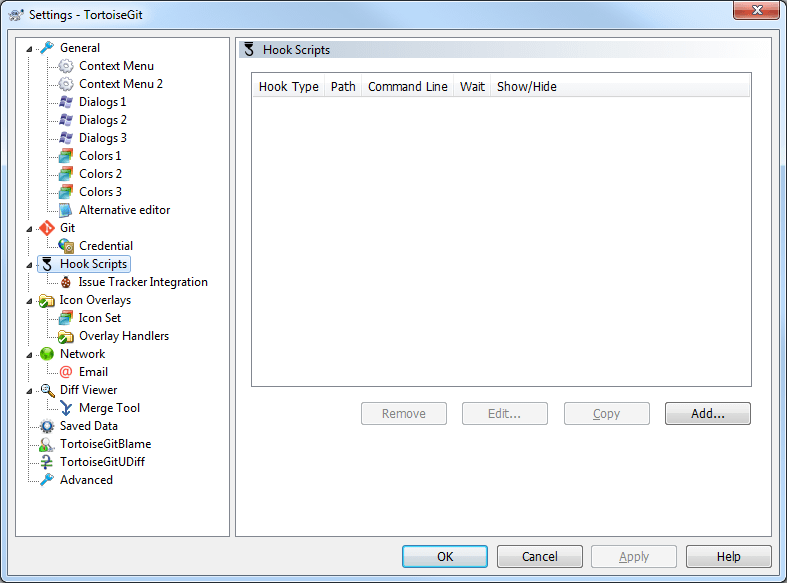
用户名缺省用户名，如果没有在URL中提供。

Use HTTP path component还考虑URL的路径组件以匹配配置上下文。

你可以找到更多的信息。[被称为“GITICONTITY（7）”的部分](https://tortoisegit.org/docs/tortoisegit/git-misc.html#gitcredentials(7)).

### 客户端侧钩脚本

**图2.92。设置对话框，Hook Scripts Page**

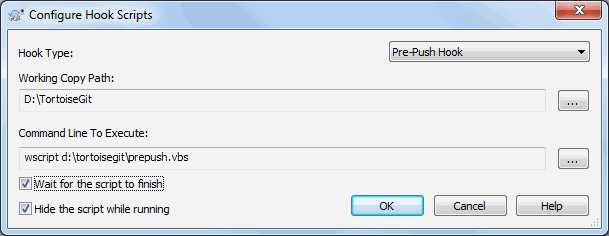


此对话框允许您设置钩子脚本，在客户端执行某些TotoSeGeT操作时，将自动执行钩子脚本。

由于各种安全性和实现原因，钩子脚本是在机器上本地定义的，而不是作为项目属性定义的。您可以定义发生了什么，不管其他人提交到存储库。当然，您总是可以选择调用脚本本身的脚本。

这种钩子的一个应用可能是调用一个程序。GETWCREV.EXE（[第3章*GITWCREV程序*](https://tortoisegit.org/docs/tortoisegit/tgit-gitwcrev.html)）更新提交后的版本号，并可能触发重新生成。

**Figure 2.93. The Settings Dialog, Configure Hook Scripts**



要添加一个新的钩子脚本，只需单击**添加**并填写细节

目前有六种类型的钩子脚本可用。

开始提交

在提交对话框显示之前调用。如果钩子修改版本化文件并影响需要提交和/或提交消息的文件列表，则可能需要使用此方法。但是，您应该注意到，因为钩子在早期被调用，所以为提交而选择的对象的完整列表是不可用的。

预提交

用户点击后调用**好啊**在提交对话框中，并在实际提交开始之前。这个钩子有一个确切的列表。

后提交

在提交成功后调用

预推

在实际Git推出之前调用。

后推

在完成完成后调用（无论成功与否）。

预重排

在重新开始之前调用（点击开始或自动启动）。

为特定的工作树路径定义了钩子。您只需要指定顶层路径；如果在子文件夹中执行操作，TortoiseGit将自动搜索匹配路径。使用\*用于匹配所有工作树

如果复选框**Run for this repository**然后检查钩子脚本被附加到当前存储库并为每个克隆和结帐自动配置（钩子信息存储在文件中）。*TGITCONFIG*in the repository root so that it will be automatically shared with all other developers using TortoiseGit >= 2.7.1; for security reasons TortoiseGit asks the user before running a hook which is configured and shared in the repository). In this case, you can specify paths for the command line with the replacement string%根%指向工作树文件夹的路径。钩子脚本必须在存储库中，当然也要检查（请注意下面的安全性暗示）。如果用户在本地为精确存储库根文件夹配置钩子，则客户端定义的钩子优先。

接下来，必须指定要执行的命令行，从钩子脚本或可执行文件的路径开始。这可以是批处理文件、可执行文件或任何其他具有有效Windows文件关联的文件，例如Perl脚本。

命令行包括由TurtoSigeIT填充的几个参数。通过的参数取决于调用哪一个钩子。每个钩子都有自己的参数，这些参数按以下顺序传递：

开始提交

路径邮件服务器随钻测井

预提交

路径邮件服务器随钻测井

后提交

随钻测井（承诺被修改（真或假））

预推

误差随钻测井

后推

误差随钻测井

预重排

（上游分支）（重新支行）误差随钻测井

这里描述了每一个参数的含义：

路径

一个临时文件的路径，它包含操作开始的所有路径。每个路径位于TEMP文件中的一条单独的行上。

邮件服务器

指向包含提交的日志消息的文件的路径。该文件包含UTF-8编码中的文本。在成功执行开始提交和预提交钩子之后，读取日志消息，给钩子修改它的机会。

误差

包含错误消息的文件的路径。如果没有错误，文件将是空的。

随钻测井

运行脚本的当前工作目录。这被设置为工作树根。

请注意，尽管我们为方便起见给出了这些参数名称，但在钩子设置中不必提及这些名称。对于特定钩子所列出的所有参数，无论您是否需要，都是通过的；

如果您希望GIT操作停止，直到钩子已完成，检查等待脚本完成.

通常，当脚本运行时，您希望隐藏丑陋的DOS框，因此运行时隐藏脚本默认情况下检查

### 注意安全

如果您正在从存储库执行版本化的文件/脚本，请注意，该文件可能被第三方更改，未被注意到（例如在拉或合并之后）。

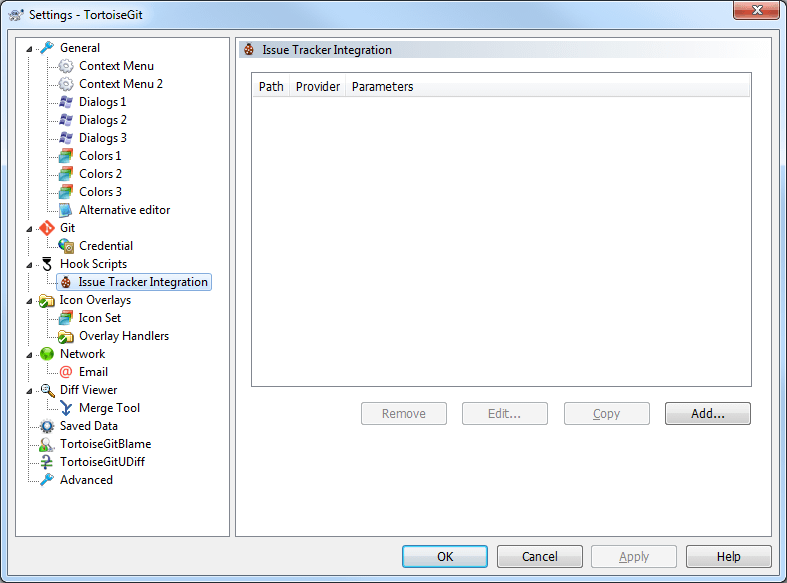
#### 问题跟踪器集成

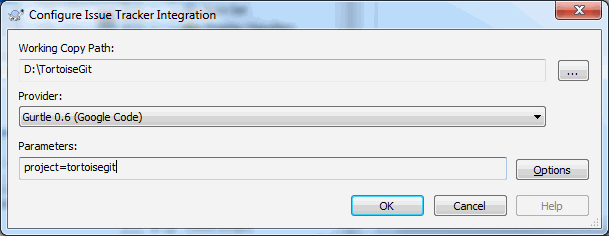
在提交对话框中，TotoSigeIT可以使用COM插件来查询问题跟踪器。描述了这样的插件的使用。[称为“从问题跟踪器获取信息”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html#tgit-dug-bugtracker-ref). 如果您的系统管理员已经为您提供了一个插件，您已经安装和注册了插件，这是指定它如何与您的工作树集成的地方。

### 小贴士

还存在一个分层的Git配置，将问题跟踪器插件与项目关联，而不是与特定的目录路径相关联。这样的设置更便于携带。见[称为“与bug跟踪系统/发布跟踪器集成”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html)配置这些设置

**图2.94。设置对话框，发布跟踪器集成页面**

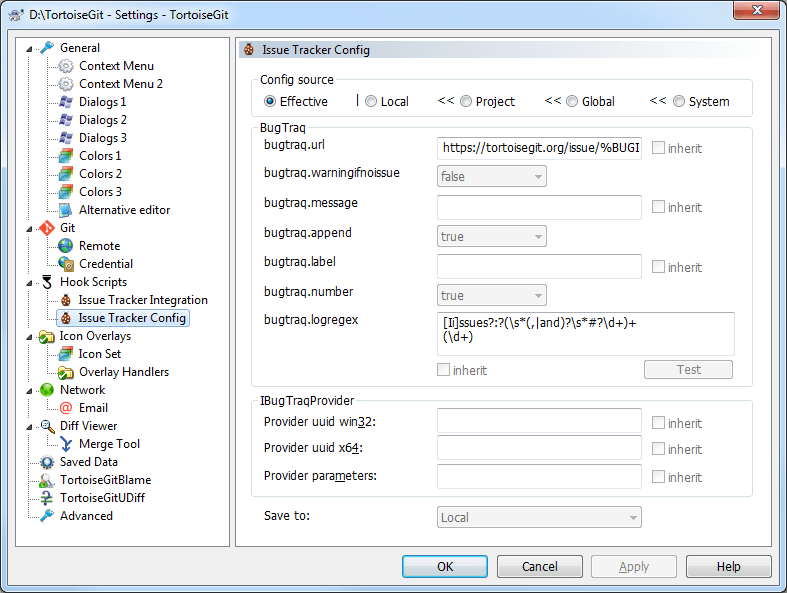




点击**添加**使用特定的工作树来使用插件。在这里，您可以指定工作树路径，选择从所有已注册的发布跟踪器插件的下拉列表中使用的插件，以及通过的任何参数。参数将是特定的插件，但可能包括您的用户名在问题跟踪器，使插件可以查询的问题，分配给你。

#### 配置

**图2.95。设置对话框，问题跟踪器配置**



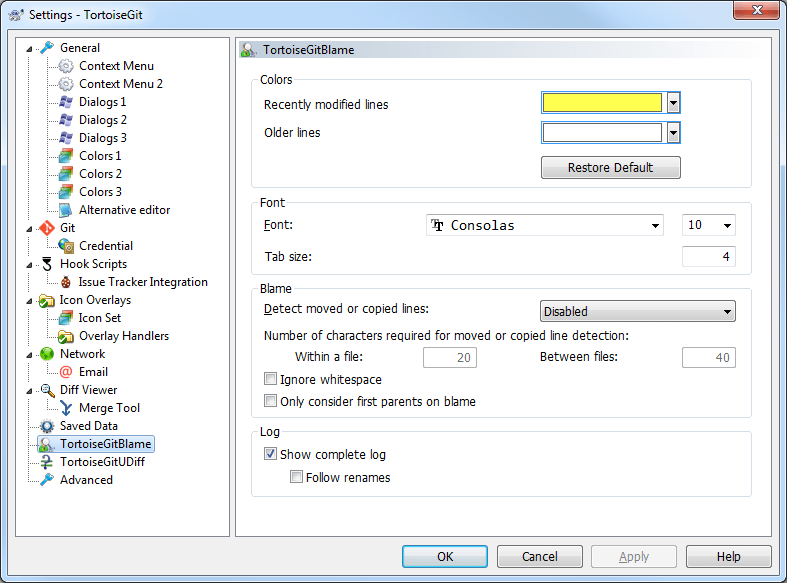
见[称为“与bug跟踪系统/发布跟踪器集成”一节](https://tortoisegit.org/docs/tortoisegit/tgit-dug-bugtracker.html)对不同选项的描述

### 重要的

如果输入/存储数据有问题请参见[称为“分层Git配置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-hierarchical).

### 托瓦耶吉特布拉姆庄园

**Figure 2.96. The Settings Dialog, TortoiseGitBlame Page**



TurtoSigEtE责备使用的设置是从主菜单控制的，而不是直接与TurtoSigeT责备本身。描述了错误算法的参数细节。[“Git责备（1）”一节](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-blame(1)).

颜色

TurtoSigeTebug可以使用背景颜色来指示文件中的行的年龄。通过指定最新和最旧版本的颜色来设置端点，TurtoSeigTebug根据每个行所指示的库修订使用这些颜色之间的线性插值。

字体

您可以选择用于显示文本的字体，以及使用的点大小。这既适用于文件内容，也适用于作者和左窗格中显示的修订信息。

标签大小

定义在文件内容中发现选项卡字符时要使用多少空间来展开。

检测移动或复制的行

残疾人传统的BREST算法，对父类的搜索仅限于文件，并将遵循重命名。

文件内应用额外的检查来检测文件中的移动和复制的行（git blame -M）

从修改文件除了注释文件，还可以检测提交中的所有修改文件的移动或复制行。git blame -C）

在文件创建时除了注释文件和在提交中修改的文件，还可以从创建文件的提交中的其他文件移动或复制行。git blame -C -C）

从现有文件此外，在任何提交中检测其他文件的移动或修改的行（git blame -C -C -C）

移动或复制线路检测所需的字符数

Git必须在文件间移动/复制时检测到的字母数字字符的下限，以便将这些行与父提交关联。

在文件内检测文件内移动行所需的字母数字字符数（git blame -M|<num>|）

文件之间检测文件之间移动或复制的行所需的字母数字字符数（git blame -C|<num>|）

忽略空白

在比较父版本和子版本时查找空白是否被忽略，以查找行从何处来。git blame -w）

显示完整日志

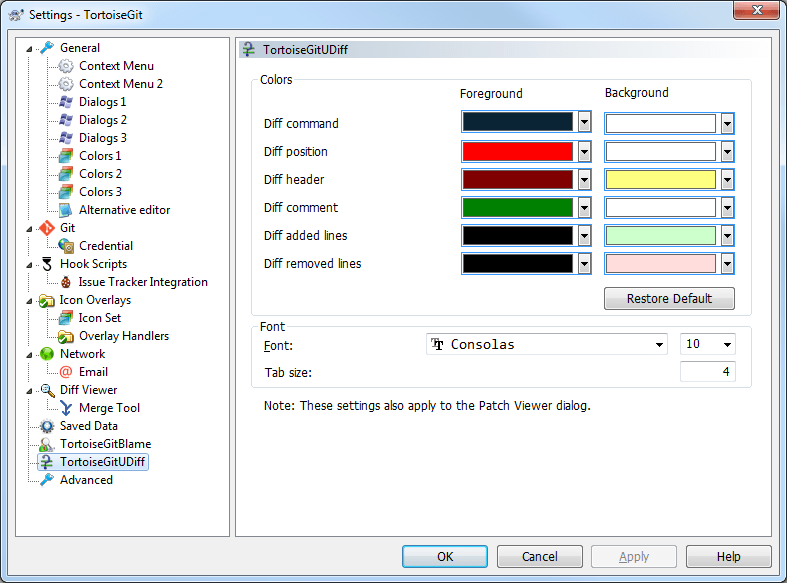
定义日志是否完整，即日志包含文件的所有更改，即使更改对注释修订的文件内容没有影响。如果停用日志，则只包含最后修改注释注释行的修订。

重命名

定义日志是否应遵循重命名，即如果文件在过去重命名文件时不停止，而是在重命名之前包含所有更改。

### 侵权行为

**图2.97。The Settings Dialog，Tortosegitutiff page**



TurtoSigItuuDIFF所使用的设置是从主菜单控制的，而不是直接与TurtoSigiTuffIF本身。

颜色

TurtoSigItuuDIFF使用的默认颜色通常是OK，但您可以在这里配置它们。

字体

您可以选择用于显示文本的字体，以及使用的点大小。

制表符

定义在文件差异中发现选项卡字符时要使用多少空间来展开。

### 高级设置

一些不常用的设置仅在设置对话框的高级页面中可用。这些设置直接修改注册表，您必须知道这些设置中的每一个是用来做什么的，以及它是做什么的。除非您确实需要更改这些设置，否则不要修改这些设置。

AutoCompleteMinChars

编辑器显示自动完成弹出窗口的最小字符量。默认值为三.

自闭

提交消息编辑器中显示的自动完成列表可以解析源代码文件并显示方法和变量名。这就限制了文件的大小以字节为单位进行解析。默认值为三十万.

AutocompleteParseUnversioned

提交消息编辑器中显示的自动完成列表可以解析源代码文件并显示方法和变量名。默认情况下，只对版本文件进行解析。将此值设置为真in order to also parse unversioned files.

自动完成扩展

提交消息编辑器中显示的自动完成列表显示提交的文件的名称。还包括删除扩展名的这些名称，将此值设置为真.

阻塞状态

如果您不希望Explorer在另一个TortoSeGIT命令运行时更新状态覆盖（例如，更新、提交、…），则将该值设置为真.

高速缓存图标

若要为TGITCache程序添加缓存托盘图标，请将该值设置为真. 这对于开发人员来说是非常有用的，因为它允许您优雅地终止程序。

高速帆船

若要禁用对TGITCache程序的加载和保存缓存，请将该值设置为假. 如果不想将缓存写入磁盘，这可以是一个大文件，这是有用的。默认值为真.

齐格温哈克

这使得一些可以使TurtoSigeit与CygWin Git一起使用的解决方案。然而，CygWin Git并没有得到TurtoSigit的正式支持。见[称为“一般设置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-main)欲了解更多信息。默认值为假.

调试

设置为真如果希望每个命令弹出对话框，显示命令行用于启动TortoiseGitProc.exe。

调试输出字符串

设置为真如果希望TortoiseGit在执行期间打印调试消息。只能用特殊的调试工具捕获消息（如[调试视图](http://technet.microsoft.com/en-us/sysinternals/bb896647.aspx)from the SysInternals Suite).

分散性指数门限

此设置控制将相似性索引阈值传递给Git DIFF（作为参数的值）。-M和-C百分之五--查找副本进入[被称为“Git DIFF（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-diff(1))）默认值为五十. 通过将此文件设置为“禁用”，可以禁用重命名和复制的文件。零仅用于检测精确重命名使用一百. 您可能需要删除缓存文件。*TurtoSigeIT.数据*和*龟裂指数*在*Git*更改此值后的文件夹

全选

在各种对话框中使用的状态列表控件（例如提交、检查修改、添加、回复、…）使用全行选择（即，如果选择一个条目，则选择了完整行，而不只是第一列）。这很好，但是选择的行也覆盖右下角的背景图像，看起来很难看。若要禁用全行选择，请将此值设置为假.

石斑鱼

此选项决定如何将各种TurtoSigIt对话框和Windows的WIN7任务栏图标分组在一起。

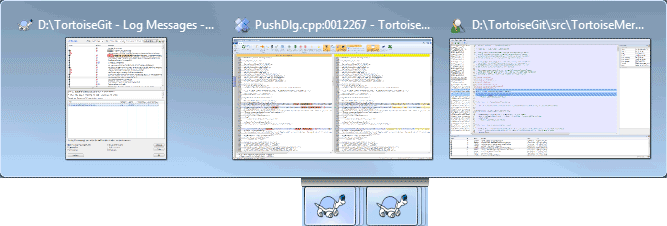
1. 默认值为三. 通过这种设置，图标按每个工作树的应用类型分组在一起。从一个工作树的TurtoSigEIT中的所有对话框被分组在一起，从一个工作树的TurtoSeigItMyGyt的所有窗口被分组在一起，…例如，如果有一个日志对话框和一个打开工作树的推送对话框C:\A检查工作树的修改对话框和日志对话框C:\B然后，在Wi7任务栏中显示了两个应用程序图标组，每个工作树都有一个组。但是TotoSigeIGielyWindows不是与TortoiseGit对话框一起分组的。

**图2.98。缺省分组任务栏**

Taskbar with default grouping

1. 如果设置为四然后，分组与设置集一起工作。三除了TurtoSeigIT，TurtoSeigiTeMy，TortoiseGitBlame，TurtoSeigTiDIFF和TurtoSigiTuffi窗口的一棵工作树都被组合在一起。例如，如果打开了日志对话框，然后双击修改后的文件，打开的TotoSigeIGiDebug窗口将被放置在任务栏上的相同图标组中，作为日志对话框图标。

**图2.99。带有存储库分组的任务栏**



1. 如果设置为一然后，分组与设置集一起工作。三（按应用分组），除了分组独立于工作树之外。这是TGIT 1.81.2之前的默认值。
2. 如果设置为二然后，分组与设置集一起工作。四，除了分组独立于工作树之外。因此，所有的TurtoSigIt图标都被分组，只显示一个图标。

Grouptaskbariconsperrepooverlay

如果选择权不起作用石斑鱼设置为零(see above).

如果此选项设置为真然后，WIN7任务栏上的每个图标显示一个小的彩色矩形叠加，表示对话框/窗口使用的工作树。

**图2.100。具有库颜色覆盖的任务栏分组**

Taskbar grouping with repository color overlays

子显示模块

这个选项定义了超级库指向哪个子模块的提交是否用分支标记来突出显示（参见）。[第2826题](https://tortoisegit.org/issue/2826)). The default is真.

最大改写项目

此选项设置最大浏览REF历史记录（右击参考超链接找到它）。默认值为五.

MSES2HACK

这使得一些可以使TurtoSeigit与MyS2 Git（不为Windows软件包启用Git）的解决方案相结合。然而，Mys2 Git并未被TurtoSigit正式支持。见[称为“一般设置”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-settings.html#tgit-dug-settings-main)欲了解更多信息。默认值为假.

远程调用

当设置为假，取来和拉不要为指定的远程调用获取默认的ReFSPEC。默认值为真.

NoSortLocalBranchesFirst

如果分支按名称完全排序，则此选项切换。真）或者如果本地分支应该出现在远程分支上（Git默认），假）默认值为假.

数字警告

如果您想同时显示更多的项目，而不是用这个设置指定的项目，首先显示一个警告对话框。默认值为十.

过度敏感的

从TurtoSigeIT 2.4.0开始，覆盖图标对文件名是区分大小写的。介绍了改变与套管有关的几个问题（如[第2654题](https://tortoisegit.org/issue/2654)）和Git工具（如GIT日志在路径上区分大小写。在[第2980题](https://tortoisegit.org/issue/2980)这是从TotoSigeGe2.5.0开始配置的，但是，不建议启用。默认值为真.

进度限制

Git进度对话框显示执行的GIT.EXE命令的输出。由于性能原因，线路的数量受到限制。默认值为五万最小值是五十.

后提交文件

此选项切换提交后未添加的添加文件的重新添加。在TurtoSigeIT 1.7.10中添加了未在提交中检查的文件，从索引中删除，提交后未版本化。将此值设置为假恢复旧行为。将此值设置为真在提交（默认）之后再次读取这些文件。

RefreshFileListAfterResolvingConflict

此选项切换提交对话框的文件列表，解决冲突和ReBASE对话框在冲突被标记为已解析时自动刷新。默认情况下，设置为真但是，在某些情况下，例如刷新时占用大量时间，或者您希望阻止滚动到顶部，这可以设置为假. 然而，然后手动刷新（例如按**F5**）是必要的

记住文件位置

此选项切换添加、提交、回复、解析和ReBASE对话框的文件列表是否记得刷新时最后选定的行。默认值为真.

消毒杀菌剂

此选项在您输入的提交消息结束时修剪空间、CR、LF字符。这包括提交、ReBASE、注释、注释标记。这个值是真默认情况下。如果这样的修剪破坏了你的脚本/插件，你可以通过设置修剪来禁止修剪。假.

Scientilladirect2d

此选项允许使用Simull控件中的DistT2D加速绘图，该控件用作编辑框，例如提交对话框（也适用于附加的补丁窗口）、统一的DIFF查看器和TurtoSigeT责备。然而，使用一些图形卡，这有时不能正常工作，使光标输入文本并不总是可见的，重绘不起作用或背景闪烁。默认情况下禁用。可以通过将此值设置为真.

ShellMenuAccelerators

TortoSeGIT为其Explorer菜单条目使用加速器。因为这会导致加倍的加速器（例如GIT提交有**ALT-C**加速器，但也是如此拷贝资源管理器的输入）。如果您不希望或需要TurtoSigEIT条目的加速器，请将该值设置为假.

表演视频

如果您使用的是Windows资源管理器之外的其他内容，或者如果菜单显示错误，那么这会很有用。将此值设置为假如果您不希望TortoiseGit为shell菜单项显示图标。将此值设置为真再次显示图标

显示连接菜单

如果您不希望TortoiseGit在自己的对话框中显示菜单的图标，请将该值设置为假.

显示列表格式

如果不希望在列表控件（例如提交对话框）中有一个小背景图像，则将该值设置为假. 将此值设置为真再次显示图像（默认）。

平方毫米

使用此设置，您可以控制哪个日期用于挤压提交。将此值设置为一如果您想使用最新提交的日期。将此值设置为二如果您想使用当前日期。将此值设置为零要使用第一个提交的日期（所有其他文件都被压缩，缺省）。

样式信息

提交和日志对话框在提交消息中使用样式（例如粗体、斜体）（参见[称为“提交日志消息”的部分](https://tortoisegit.org/docs/tortoisegit/tgit-dug-commit.html#tgit-dug-commit-log)详情请参阅。如果不想这样做，将值设置为假.

Tgitcacheckcontentmaxize

TGitCache checks the content of files by hashing them and comparing the SHA1 in order to calculate the file statuses if the timestamps (to index) mismatch. This option allows to restrict this behavior for files which do not exceed a specific size (in KiB). The default maximum file size is 10 MiB (i.e., 10 \* 1024 KiB =一万零二百四十KiB）将此设置为0，以便使TGITCHILL只检查时间戳（TurtoSigeIT 1.7.0到1.7.12Do；在TortoiseGit 1.90.0之前，这是由TGITCaseCeChar内容控制的）。禁用文件内容可以降低TGITCACHE进程的磁盘访问和CPU时间，但是，覆盖精度可能不如检查启用的文件内容那么精确。

用户名破译

标准编辑控件不会在前斜杠上停止，就像在路径和URL中找到的一样。TurtoSeGIT使用自定义的Word中断程序进行编辑控件。如果您不希望使用默认值，则将该值设置为0。如果只想在组合框中编辑控件的默认值，请将该值设置为1。

UseLibgit2

这使得TurtoSigIT尽可能多地使用LIGBGIT2（例如，将文件添加到索引中）。如果不希望TotoSigeIT使用LBGIT2进行文件操作，请将该值设置为假.

版本检查

TurtoSeigt检查是否有一个新版本可用一周一次。如果不希望TortoiseGit执行此检查，请将此值设置为假.

版本检查预览

设置为真使TurtoSigEIT也检查新的预览版本。所有稳定版本中的默认值为假.

### 导出TurtoSigeIT设置

如果希望导出所有客户端设置以在另一台计算机上使用，则可以使用Windows注册表编辑器来执行此操作。*regedt32.exe*. 转到注册表项HKCU软件公司并将其导出到Reg文件。在另一台计算机上，只需再次导入该文件（通常双击Reg文件就可以了）。

请记住保存Git的一般设置，您可以在Git配置文件中找到这些设置GITCONFIG和/或文件夹CONFIG/GIT它们都位于您的用户配置文件目录中。

## GIT SvN Debug

将指定的头部中的每个DIFF直接提交到SvN存储库，然后重新设置或重置（取决于是否存在Svn和Head之间的差异）。这将为Git中的每个提交创建一个在VPN中的修订。建议您运行GITSvnFETH和ReBASE（不拖拽或合并），以防SN存储库中的最新更改。

如果你需要/想要使用--使用日志作者或--添加作者请将这些设置为Git配置（参见）。[被称为“Git配置（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-config(1))), also see[第2824题](https://tortoisegit.org/issue/2824).

Git风格提交（-RMIDR）如果没有遗留文件，则从目录树中删除目录。Svn可以版本空目录，如果没有文件，它们不会被默认删除。Git不能版本空目录。启用此标志将使提交到SN的行为像Git。

你可以找到更多的信息。[被称为“Git Svn（1）”的部分](https://tortoisegit.org/docs/tortoisegit/git-command.html#git-svn(1)).

## 最后一步

[**捐赠！**](https://tortoisegit.org/donate)

即使TurtoSeigIT和TortoiseGitMerge是免费的，你也可以通过发送补丁来支持开发人员，并在开发中发挥积极的作用。你也可以帮助我们在我们在电脑前度过的漫长的时光中振作起来。

也请看一下通过发送补丁或翻译来为项目做出贡献的人的名单。