GitX

# User Manual

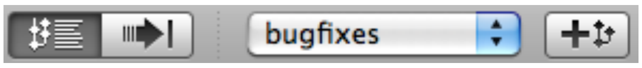
This is the GitX user's manual. This document tries to explain the most common used features and explains the concepts that might be confusing if you've never used a git GUI.

## Disclaimer

GitX aims to be a graphical wrapper around the most-frequently used git-commands, enabling you to satisfy your daily git needs in one consistent program. GitX doesn't aspire to be a complete interface for the wide variety of git command-line programs with their ever-growing number of options. It's also no easy way to learn git. A sound understanding of how git stores data, how it works on that data and how to recover from (accidental) deletion of branches and commits will not only help you with git but will also enable you to learn GitX with greater confidence and speed. Since GitX is far from complete you'll probably still have to use it in conjunction with the command-line git for more complex tasks, such as merging or performing remote operations.

## Overview

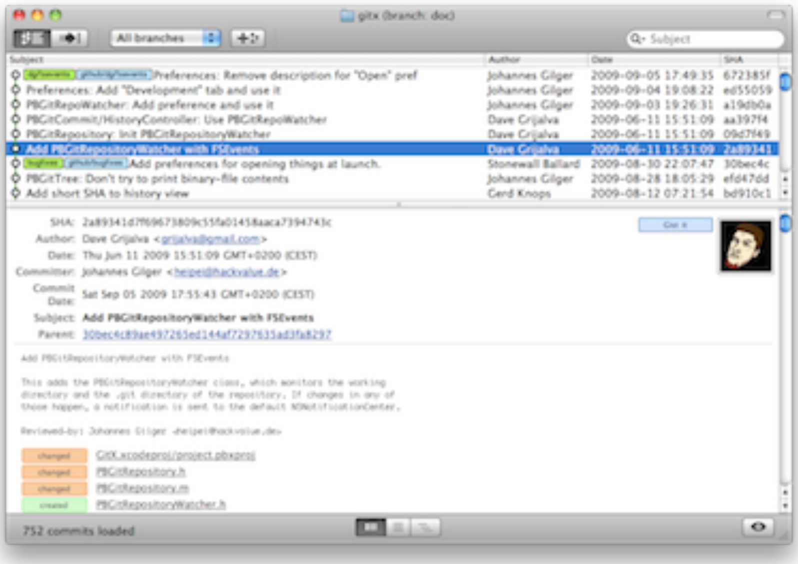
When you open GitX for the first time, you will be greeted by an open-dialog. In this dialog, you should select the Git repository you want to open. You can select either a directory containing a .git directory, or a .git directory itself.
After doing this, you will be greeted by the default GitX view, which is the history-view. The history-view consists of two main parts. The top part is used for displaying commits on branches and the bottom-view displays details about the selected commit. The window-title will always show you the currently checked-out branch (or indicate a detached HEAD).



The menu contains buttons to switch between history/commit-view, a selector for specifying which branch to show and an 'Add branch' button. Branches can be checked-out and deleted within GitX by right-clicking their colored bubbles the commit-list.

You can double-click the divider between these areas to collapse the smaller-one or you can use Command-Shift-Up and Command-Shift-Down to do so.

### The commit list



| Subject | Author | Date | SHA |
|---|---|---|---|
| Preferences: Remove description for "Open" pref | Johannes Gilger | 2009-09-05 17:49:35 | 672385f |
| Preferences: Add "Development" tab and use it | Johannes Gilger | 2009-09-04 19:08:22 | ed55059 |
| PBGitRepoWatcher: Add preference and use it | Johannes Gilger | 2009-09-03 19:26:31 | a19db0a |
| PBGitCommit/HistoryController: Use PBGitRepoWatcher | Dave Grijalva | 2009-06-11 15:51:09 | aa397f4 |
| PBGitRepository: Init PBGitRepositoryWatcher | Dave Grijalva | 2009-06-11 15:51:09 | 09d7f49 |
| Add PBGitRepositoryWatcher with FSEvents | Dave Grijalva | 2009-06-11 15:51:09 | 2a89341 |
| Add preferences for opening things at launch. | Stonewall Ballard | 2009-08-30 22:07:47 | 30bec4c |
| PBGitTree: Don't try to print binary-file contents | Johannes Gilger | 2009-08-28 18:05:29 | efd47dd |
| Add short SHA to history view | Gerd Knops | 2009-08-12 07:21:54 | bd910c1 |

SHA: 2a89341d7f69673809c55fa01458aaca7394743c
Author: Dave Grijalva <grijalva@gmail.com>
Date: Thu Jun 11 2009 15:51:09 GMT+0200 (CEST)
Committer: Johannes Gilger <heipei@hackvalue.de>
Commit Date: Sat Sep 05 2009 17:55:43 GMT+0200 (CEST)
Subject: **Add PBGitRepositoryWatcher with FSEvents**
Parent: 30bec4c89ae497265ed144af7297635ad3fa8297

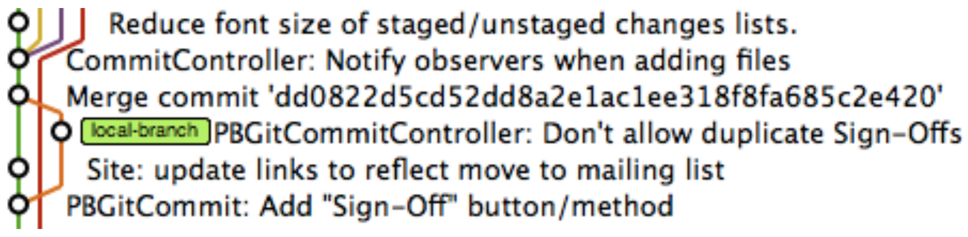
```
Add PBGitRepositoryWatcher with FSEvents

This adds the PBGitRepositoryWatcher class, which monitors the working
directory and the .git directory of the repository. If changes in any of
those happen, a notification is sent to the default NSNotificationCenter.

Reviewed-by: Johannes Gilger <heipei@hackvalue.de>
```

changed GitX.xcodeproj/project.pbxproj
changed PBGitRepository.h
changed PBGitRepository.m
created PBGitRepositoryWatcher.h

On the left side in the commit list you can see the *branch lines*. These show you how your history has diverged and merged. As newer commits are on top, two lines joining each other from the bottom is a merge. This way you see which branches were merged in without any effort.



Reduce font size of staged/unstaged changes lists.
CommitController: Notify observers when adding files
Merge commit 'dd0822d5cd52dd8a2e1ac1ee318f8fa685c2e420'
local-branch PBGitCommitController: Don't allow duplicate Sign-Offs
Site: update links to reflect move to mailing list
PBGitCommit: Add "Sign-Off" button/method

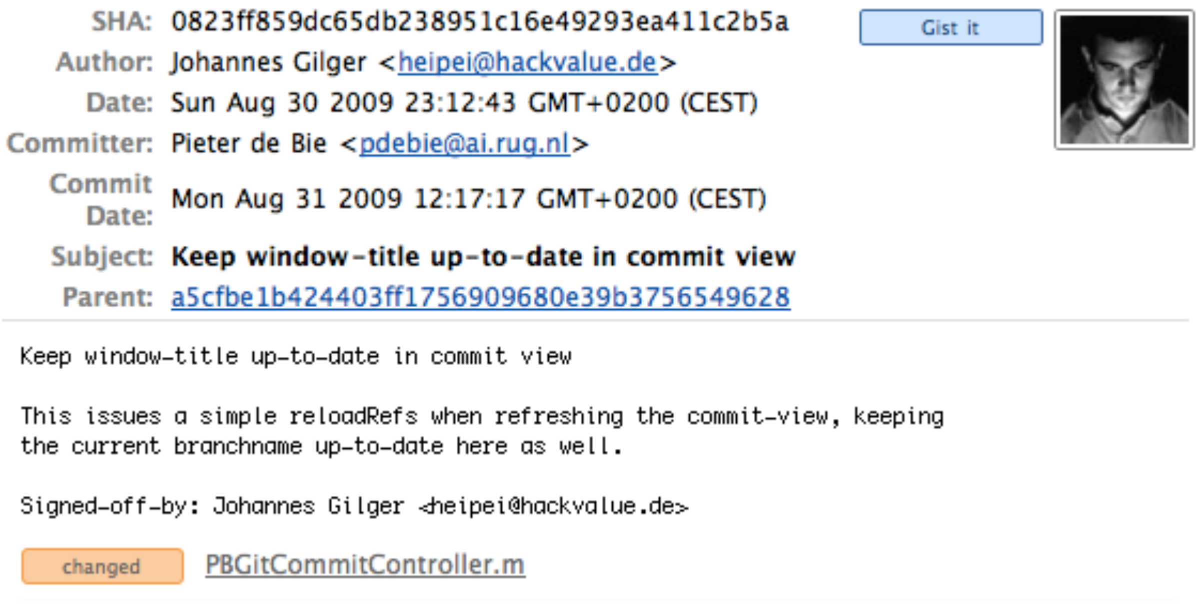
On some commits, to the right of the branch lines you will find *commit labels*. These indicate references to these commits, for example branch heads and remote heads. *Green* labels indicate local branches. *Blue* labels indicate remote branches (it will be in the form of remote/branch). *Yellow* labels indicate tags. The *orange* label indicates the currently checked-out branch.

There are four columns, the first showing the commit *subject* (which is the top line of your commit message), the second the *author name*, the third the commit date and the last showing the abbreviated SHA of the commit. You can sort on any of these columns. However, if you sort the branch lines will disappear. Repeatedly clicking the row will revert the ordering and the original order will be restored, including the branch lines.

In the top right you will also find the search bar. Here you can search on subject, author or SHA. If you do this, the branch lines will also disappear.

## The detail view

Below the list of commits rests the detail view which shows information about the currently selected commit. The detail view can switch between three different ways of displaying the commit using the three buttoms at the bottom. The first tab shows information about the current commit in a nice markup and will probably be the view you use most. In this mode, you can see a pretty diff of the commit, and information such as the parent SHA and the author's email address.



SHA: 0823ff859dc65db238951c16e49293ea411c2b5a
Author: Johannes Gilger <heipei@hackvalue.de>
Date: Sun Aug 30 2009 23:12:43 GMT+0200 (CEST)
Committer: Pieter de Bie <pdebie@ai.rug.nl>
Commit Date: Mon Aug 31 2009 12:17:17 GMT+0200 (CEST)
Subject: Keep window-title up-to-date in commit view
Parent: a5cfbe1b424403ff1756909680e39b3756549628

```
Keep window-title up-to-date in commit view

This issues a simple reloadRefs when refreshing the commit-view, keeping
the current branchname up-to-date here as well.

Signed-off-by: Johannes Gilger <heipei@hackvalue.de>
```

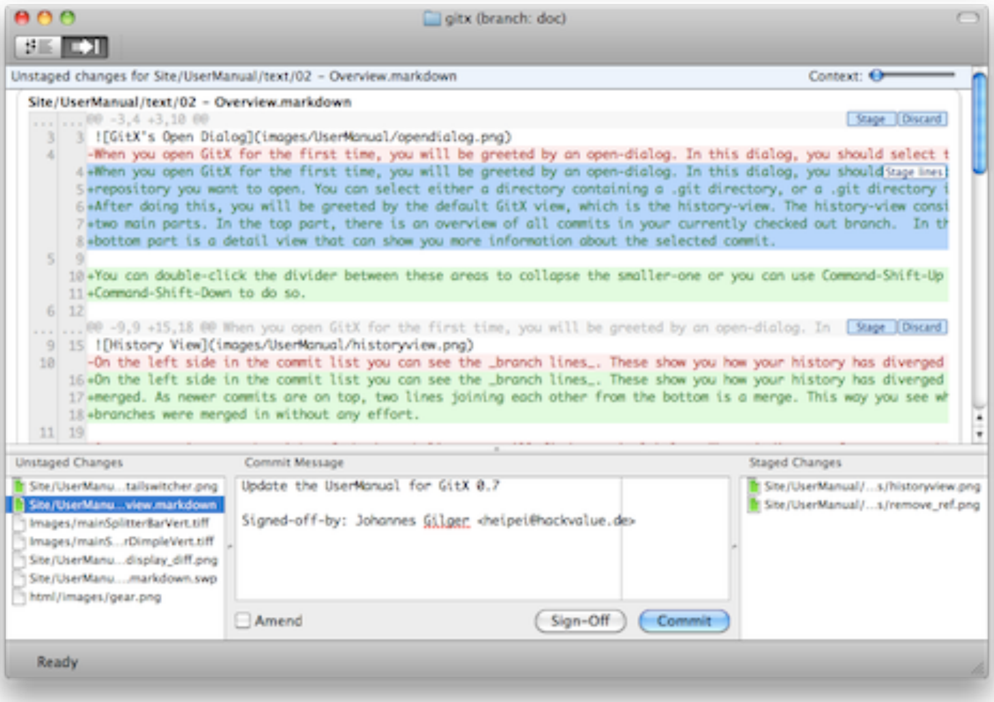
PBGitCommitController.m

The view is pretty much self-explanatory but it does contain some features which might not be obvious. You can right-click the refs and files to get a context-menu. The "Gist it" buton will upload the current patch to gist.github.com. This will use your `github.user` and `github.token` git-config options if those are set. Otherwise it will create an anonymous gist. Have a look at the *Preferences* to set options for Gisting your patches.

The second detailed view will simply show the raw content of the commit, much like `git show --pretty=raw` would.

The third detailed view is more interesting: It allows you to browse your history in tree-view and export files/trees from certain commits. To do so, select the commit and then simply drag-and-drop the wanted dir/files from the tree to a folder in the Finder. This is also where the Quicklook button in the bottom right comes in: pressing it while selecting a file that can be QuickLooked (an image, for example), will display it in the same way as the Finder's quicklook. GitX even imitates this behaviour by allowing you to press space to quicklook a file.

## Committing


gitx (branch: doc)
Unstaged changes for Site/UserManual/text/02 - Overview.markdown
Context:
Site/UserManual/text/02 - Overview.markdown
Stage
Discard
Stage lines
Unstaged Changes
Commit Message
Staged Changes
Site/UserManu...tailswitcher.png
Site/UserManu...view.markdown
Images/mainSplitterBarVert.tiff
Images/mainS...rDimpleVert.tiff
Site/UserManu...display_diff.png
Site/UserManu....markdown.swp
html/images/gear.png
Site/UserManual/...s/historyview.png
Site/UserManual/...s/remove_ref.png
Amend
Sign-Off
Commit
Ready

```
@@ -3,4 +3,10 @@
 ![GitX's Open Dialog](images/UserManual/opendialog.png)
-When you open GitX for the first time, you will be greeted by an open-dialog. In this dialog, you should select t
+When you open GitX for the first time, you will be greeted by an open-dialog. In this dialog, you should
+repository you want to open. You can select either a directory containing a .git directory, or a .git directory i
+After doing this, you will be greeted by the default GitX view, which is the history-view. The history-view consi
+two main parts. In the top part, there is an overview of all commits in your currently checked out branch.  In th
+bottom part is a detail view that can show you more information about the selected commit.

+You can double-click the divider between these areas to collapse the smaller-one or you can use Command-Shift-Up
+Command-Shift-Down to do so.

@@ -9,9 +15,18 @@ When you open GitX for the first time, you will be greeted by an open-dialog. In
 ![History View](images/UserManual/historyview.png)
-On the left side in the commit list you can see the _branch lines_. These show you how your history has diverged
+On the left side in the commit list you can see the _branch lines_. These show you how your history has diverged
+merged. As newer commits are on top, two lines joining each other from the bottom is a merge. This way you see wh
+branches were merged in without any effort.
```

```
Update the UserManual for GitX 0.7

Signed-off-by: Johannes Gilger <heipei@hackvalue.de>
```

The commit-interface of GitX also consists of two parts. The top part will show the diff of the currently selected file. The bottom part is a bit more complex.

In the left list you will see files that are *unstaged*. If you are already used to Git's terminology, you will know what this means. These are the changes that have not been added to the index, and thus also will not be committed.

The rightmost part shows the *staged changes*. These *will* be committed in the next commit.

It is possible to have a filename in both lists at the same time. In that case, there are some changes in the file that will be committed, and some that have been left out. Clicking on their name will show you what the difference is in the top pane.

To stage a file, double-click its icon. This will move the file from the left to the right. In the same way, you can unstage changes by double clicking on the name in the right pane. If you prefer, you can also drag and drop the files from one list to the other. You can also select multiple files, and then drag them to the other side all at once.

Every file has an icon associated with it. This shows the status of that file:

- A *green* icon indicates there a changes to a file that is already tracked.
- A *white* icon indicates a file that is not tracked by Git, but also not ignored.
- A *red* icon indicates a file that has been deleted.

### Partial staging

Sometimes you have made more than one change to a file, but would still prefer to create more than one git commit; the more specific the commits, the better, right? This is where hunk and line-wise staging comes in. Next to each specific change in a file, called a *hunk*, you will see a blue 'Stage' button. If you click on this button, only those specific changes will be staged. Clicking the 'Discard' button will irreversibly throw away this change, so use it with care! The confirmation can be silenced using Alt-Click.

GitX 0.7 introduced a new way of staging lines: Simply drag-select a few of the lines you want to stage/unstage and a 'Stage lines' button will appear next to it. This allows for much finer granularity than the hunks determined by diff. Double-clicking a changed line selects the entire surrounding block of changes. Selecting lines across hunks is not possible.

### Committing

Once you have staged all your changes, you can commit them. Supply a commit message in the center pane, but remember to use proper commit messages: the first line should be a short description of what you have changed followed by an empty line and a longer description. Make sure the subject of your commit is short enough to fit in GitX's history view!
Pressing the commit button will hopefully convey to you that the commit was successful, and also give you the commit hash.

The 'Amend' checkbox represents `git commit --amend`: It enables you to amend the commit on the tip of the current branch. Checking 'Amend' will give you the commit-message of mentioned commit and also will display all of the changes the commit introduced as staged in the right-hand pane. You may then stage/unstage further changes, change the commit-message and hit 'Commit'. The 'Sign-Off' button adds your signature to the commit which is common practice in a lot of Open-Source projects.

## Advance GitX usage

While GitX can be used just fine without ever touching the terminal, some of it's power can currently only be used through the command line utility.

You can install this utility by choosing "Enable Terminal Usage…" from GitX's menu. This will ask you if GitX is allowed to install the `gitx` utility in `/usr/local/bin`. After you have done this, you can use gitx from the terminal in any Git repository!

To become familiar with this utility, you can read it's short help message:

```
Tirana:UserManual pieter$ gitx --help
Usage: gitx --help
   or: gitx (--commit|-h)
   or: gitx <revlist options>

    -h, --help          print this help
    --commit, -c        start GitX in commit mode

RevList options
    See 'man git-log' and 'man git-rev-list' for options you can pass to gitx

    --all                   show all branches
    <branch>                show specific branch
     -- <path>              show commits touching paths
```

As you can see already, `gitx` has some useful options. To quickly start committing, use `gitx -c`.

However, to really make use of all the options, you should read the `git-log` and `git-rev-list` man-pages. Gitx takes most of the commands that normal git utilities also take.

For example, if you want to show commits that change lines with the word 'cool' in it, you can use:

```
gitx -Scool
```

and to see commands that touch only files in the "Documentation" directory, you can use

```
gitx -- Documentation
```

You can also specify which branches gitx should display, for example:

```
gitx commit_view log_options --grep="fix" --author=Pieter
```

will show all commits in the `commit_view` and `log_options` branches that have the word 'fix' in their commit message and which are committed by Pieter.

If you have a lot of commits, you can choose to limit the output. For example,

```
gitx -1000
```

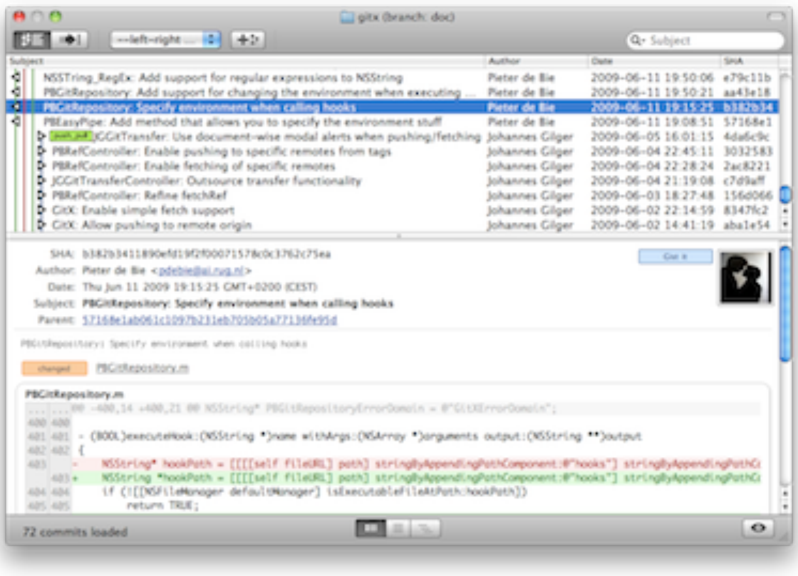
will only show the first 1000 commits. Limiting commits can be useful in other contexts too. For example, if you want to know what you have committed that isn't on the remote yet, you can use

```
gitx origin/master..
```

If there are changes both on a remote and on your local side, you can show them at once with the *symmetric set difference*:

```
gitx --left-right origin/master...HEAD
```

will show commits that are only on your side OR only on the remote side. The `--left-right` options does some other magic: instead of the usual circles GitX uses in the branch lines, it will now use an arrow pointing left for commits only on the left side (that is, in `origin/master`), and an arrow to the right for commits only on the right side. For example, when I issued the same command on this repository, I got:



| Subject | Author | Date | SHA |
|---|---|---|---|
| NSSTring_RegEx: Add support for regular expressions to NSString | Pieter de Bie | 2009-06-11 19:50:06 | e79c11b |
| PBGitRepository: Add support for changing the environment when executing ... | Pieter de Bie | 2009-06-11 19:50:21 | aa43e18 |
| PBGitRepository: Specify environment when calling hooks | Pieter de Bie | 2009-06-11 19:15:25 | b382b34 |
| PBEasyPipe: Add method that allows you to specify the environment stuff | Pieter de Bie | 2009-06-11 19:08:51 | 57168e1 |
| push_pull JGGitTransfer: Use document-wise modal alerts when pushing/fetching | Johannes Gilger | 2009-06-05 16:01:15 | 4da6c9c |
| PBRefController: Enable pushing to specific remotes from tags | Johannes Gilger | 2009-06-04 22:45:11 | 3032583 |
| PBRefController: Enable fetching of specific remotes | Johannes Gilger | 2009-06-04 22:28:24 | 2ac8221 |
| JGGitTransferController: Outsource transfer functionality | Johannes Gilger | 2009-06-04 21:19:08 | c7d9aff |
| PBRefController: Refine fetchRef | Johannes Gilger | 2009-06-03 18:27:48 | 156d066 |
| GitX: Enable simple fetch support | Johannes Gilger | 2009-06-02 22:14:59 | 8347fc2 |
| GitX: Allow pushing to remote origin | Johannes Gilger | 2009-06-02 14:41:19 | aba1e54 |

SHA: b382b3411890efd19f2f0007157 8c0c3762c75ea
Author: Pieter de Bie <pdebie@ai.rug.nl>
Date: Thu Jun 11 2009 19:15:25 GMT+0200 (CEST)
Subject: **PBGitRepository: Specify environment when calling hooks**
Parent: 57168e1ab061c1097b231eb705b05a77136fe95d

PBGitRepository: Specify environment when calling hooks

changed PBGitRepository.m

**PBGitRepository.m**

```
@@ -400,14 +400,21 @@ NSString* PBGitRepositoryErrorDomain = @"GitXErrorDomain";

- (BOOL)executeHook:(NSString *)name withArgs:(NSArray *)arguments output:(NSString **)output
{
-	NSString* hookPath = [[[[self fileURL] path] stringByAppendingPathComponent:@"hooks"] stringByAppendingPathC
+	NSString *hookPath = [[[[self fileURL] path] stringByAppendingPathComponent:@"hooks"] stringByAppendingPathC
	if (![[NSFileManager defaultManager] isExecutableFileAtPath:hookPath])
		return TRUE;
```

72 commits loaded

You can also pipe any command that produces a diff to GitX to see its changes. For example, to compare version 0.5 and version 0.6 of GitX, you can do the following:

```
git diff v0.5 v0.6 | gitx
```