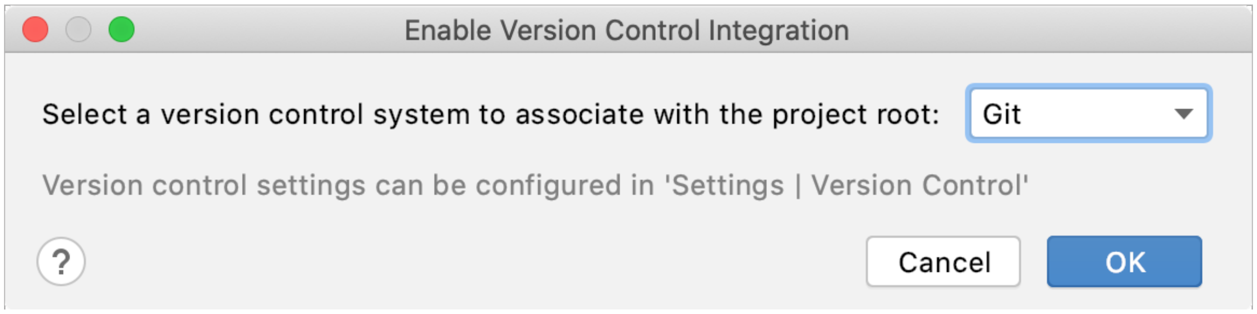
(https://www.jetbrains.com/help/clion/working-with-git-tutorial.html#github)

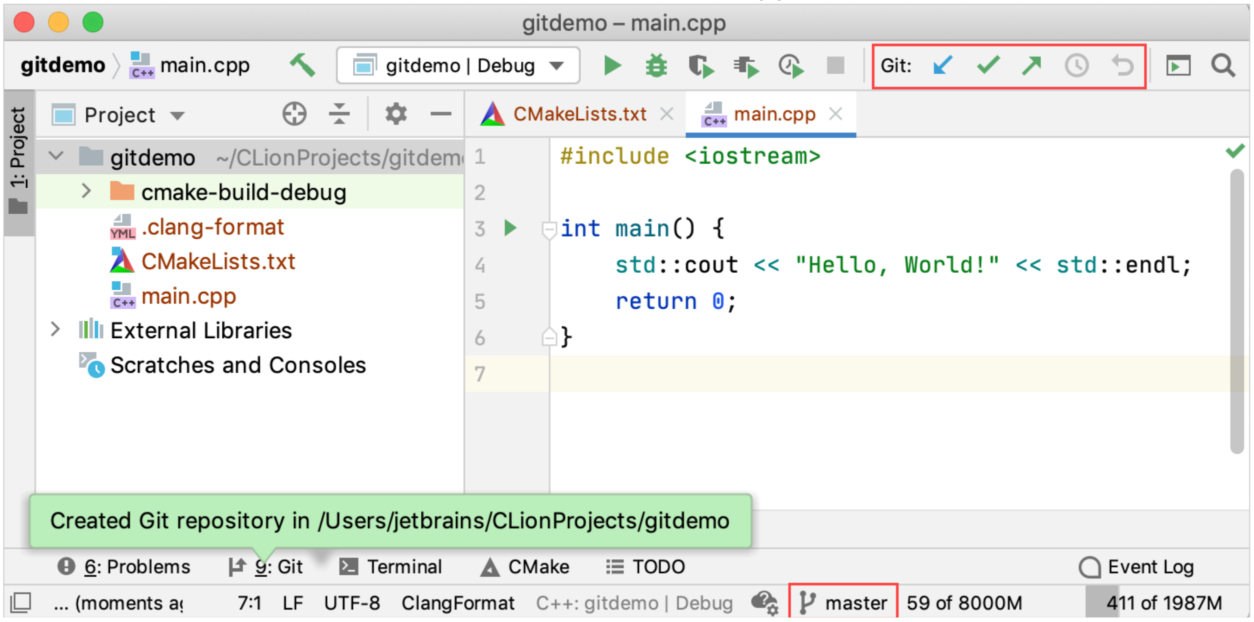
Step 1. Enable Git integration

1. From the main menu, select **| Enable Version Control Integration**.
2. In the dialog that opens, select **Git** from the list of available version control systems and click **OK**.


Enable Version Control Integration
Select a version control system to associate with the project root:
Git
Version control settings can be configured in 'Settings | Version Control'
?
Cancel
OK

Alternatively, press ^V and select **Create Git Repository** (or press 1). In the Finder window that opens, specify a root folder for the local Git repository.

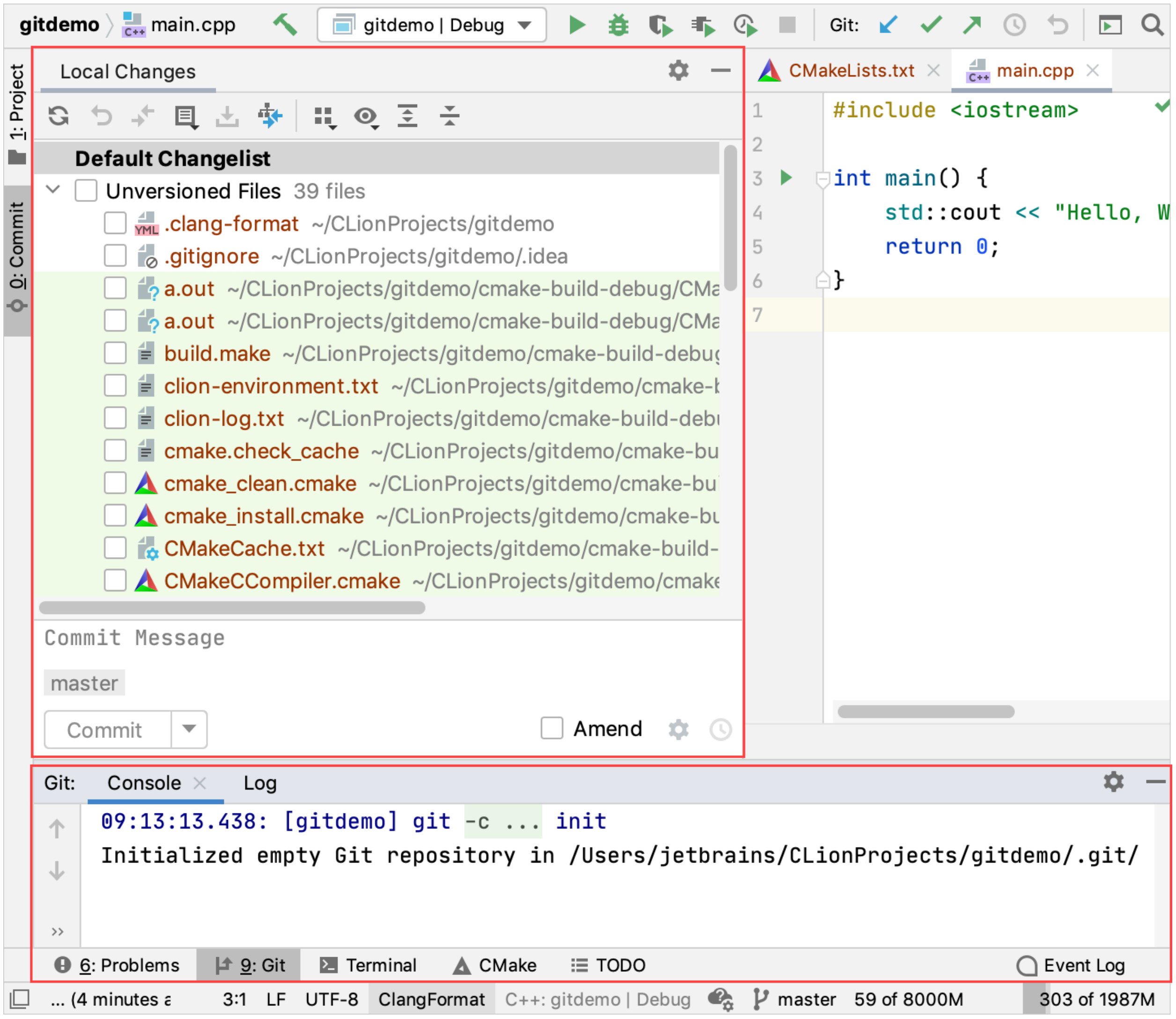
You will get a notification that the local Git repository has been created for your project. On the toolbar and status bar, the Git-related controls will appear:



```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

Created Git repository in /Users/jetbrains/CLionProjects/gitdemo

The dedicated tool windows for working with Git are now available: **Commit** (⌘K or **View | Tool Windows | Commit**) and **Git** (⌘9 or **View | Tool Windows | Git**).



Default Changelist
Unversioned Files 39 files
.clang-format ~/CLionProjects/gitdemo
.gitignore ~/CLionProjects/gitdemo/.idea
a.out ~/CLionProjects/gitdemo/cmake-build-debug/CMa
a.out ~/CLionProjects/gitdemo/cmake-build-debug/CMa
build.make ~/CLionProjects/gitdemo/cmake-build-debu
clion-environment.txt ~/CLionProjects/gitdemo/cmake-b
clion-log.txt ~/CLionProjects/gitdemo/cmake-build-deb
cmake.check_cache ~/CLionProjects/gitdemo/cmake-bu
cmake_clean.cmake ~/CLionProjects/gitdemo/cmake-bu
cmake_install.cmake ~/CLionProjects/gitdemo/cmake-bu
CMakeCache.txt ~/CLionProjects/gitdemo/cmake-build-
CMakeCCompiler.cmake ~/CLionProjects/gitdemo/cmake

```
#include <iostream>

int main() {
    std::cout << "Hello, W
    return 0;
}

```

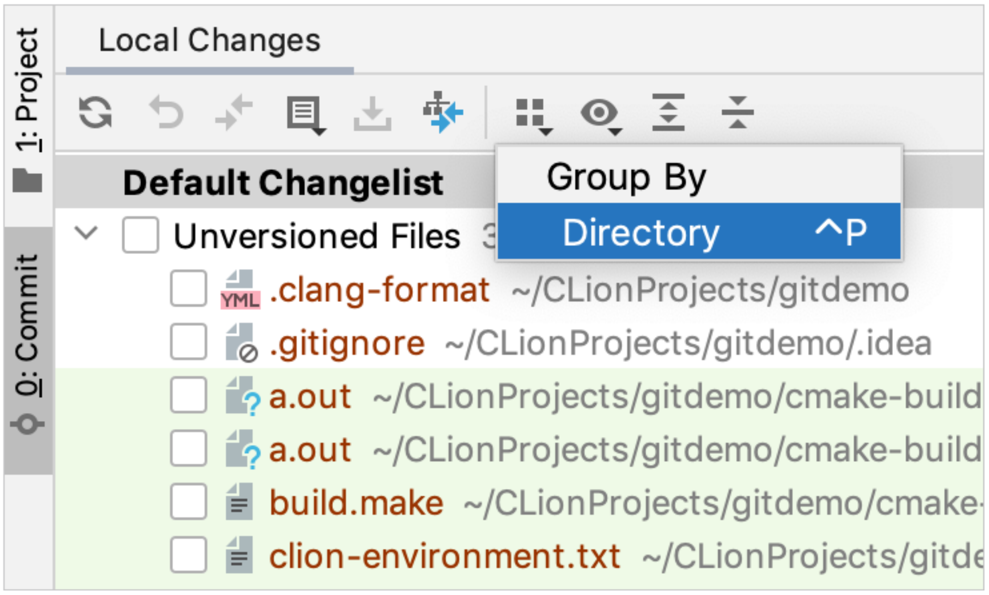
```
09:13:13.438: [gitdemo] git -c ... init
Initialized empty Git repository in /Users/jetbrains/CLionProjects/gitdemo/.git/
```

From the **Commit** tool window ⌘0, you can review the local changes and commit them to the local Git repository. From the **Git** tool window, you can work with the Git log, manage pull requests from GitHub, and more.

Step 2. Add files to .gitignore

On the **Local Changes** tab of the **Commit** tool window ⌘0, you see the list of files that belong to your project. These files are not added to the Git repository yet — you need to select which of them you want to share and which ones should be ignored by the VCS.

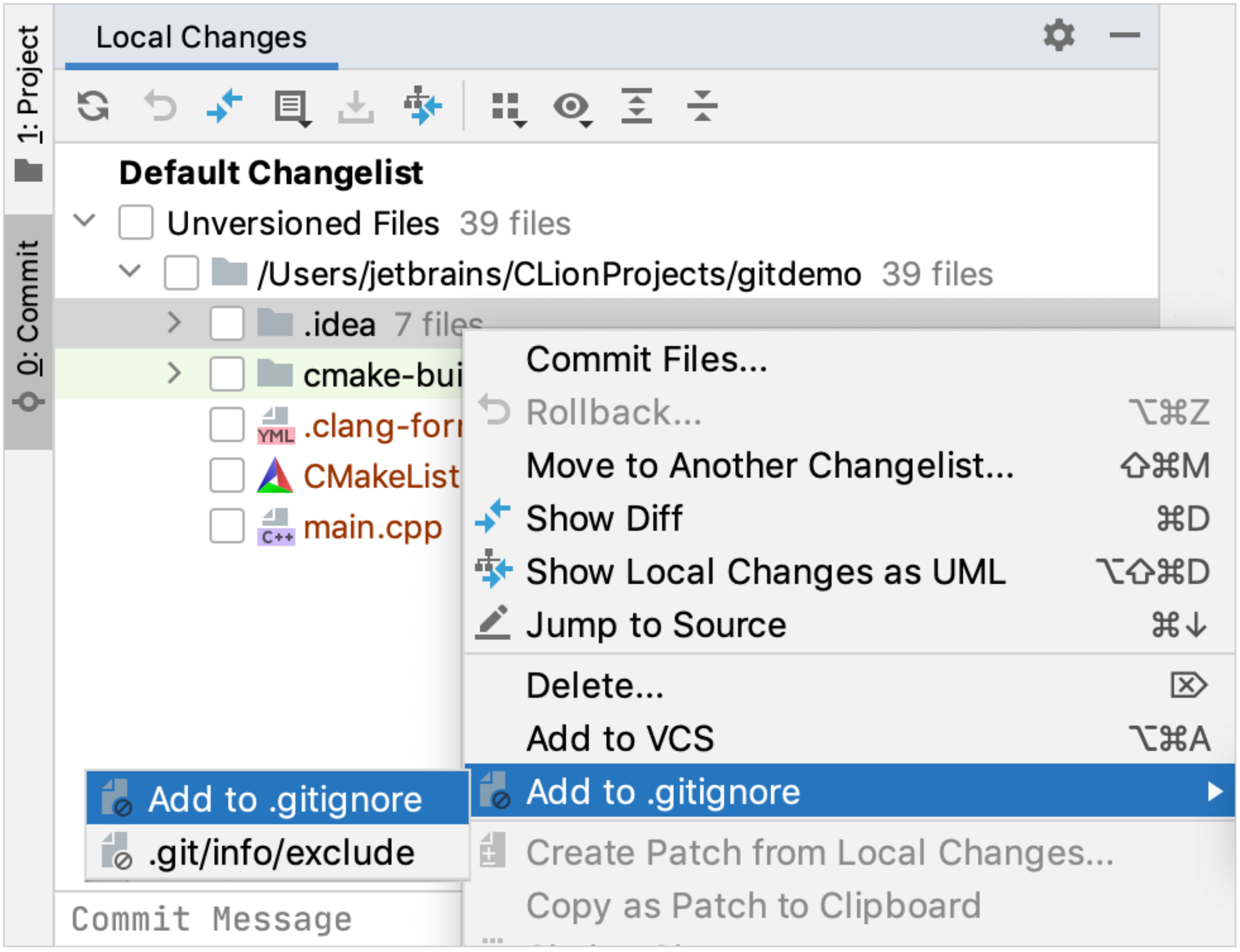
1. Group the files by directories: press ^P or click on the toolbar and select **Directory**.



```
.clang-format  ~/CLionProjects/gitdemo
.gitignore  ~/CLionProjects/gitdemo/.idea
a.out  ~/CLionProjects/gitdemo/cmake-build
a.out  ~/CLionProjects/gitdemo/cmake-build
build.make  ~/CLionProjects/gitdemo/cmake-
clion-environment.txt  ~/CLionProjects/gitde
```

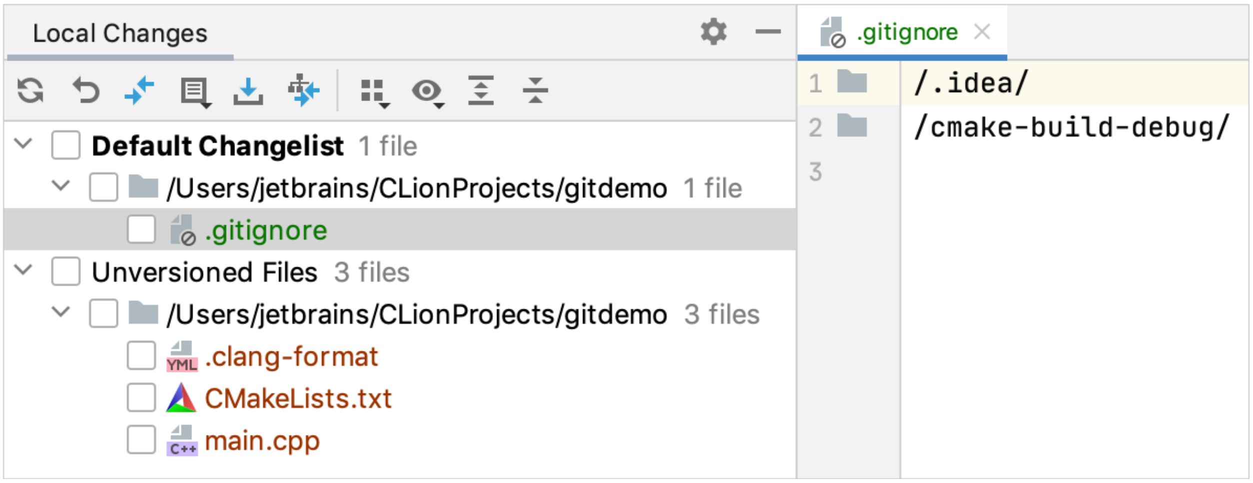
2. Select the directories that you don't want to share. For example, the following directories may be ignored without breaking the integrity of the project:
   - **.idea**: settings of your local CLion installation. Ignore this directory unless you want to share your settings among the team members.

     Generally, it's not recommended that you put the entire contents of **.idea** under *gitignore*. See this instruction for more details.
   - **cmake-build-debug**: the automatically created directory for CMake build artifacts.
3. Right-click the selection and choose **Add to .gitignore | Add to .gitignore**.



4. You will be prompted to confirm the creation of a new **.gitignore** file in the root directory of your project. Click **Create**.
5. From the dialog that opens, you can either add the newly created file to Git right away by pressing **Add** or postpone this by pressing **Cancel**.

   Click **Add**. You will see that the **.gitignore** file has been added to the root directory of your project and placed to **Default Changelist**. The directories you selected to ignore are no longer displayed in the **Unversioned Files** list:


Local Changes
Default Changelist 1 file
/Users/jetbrains/CLionProjects/gitdemo 1 file
.gitignore
Unversioned Files 3 files
/Users/jetbrains/CLionProjects/gitdemo 3 files
.clang-format
CMakeLists.txt
main.cpp
.gitignore

```
/.idea/
/cmake-build-debug/
```

Any time, you can edit the **.gitignore** file to add new directories to the list or remove existing ones.

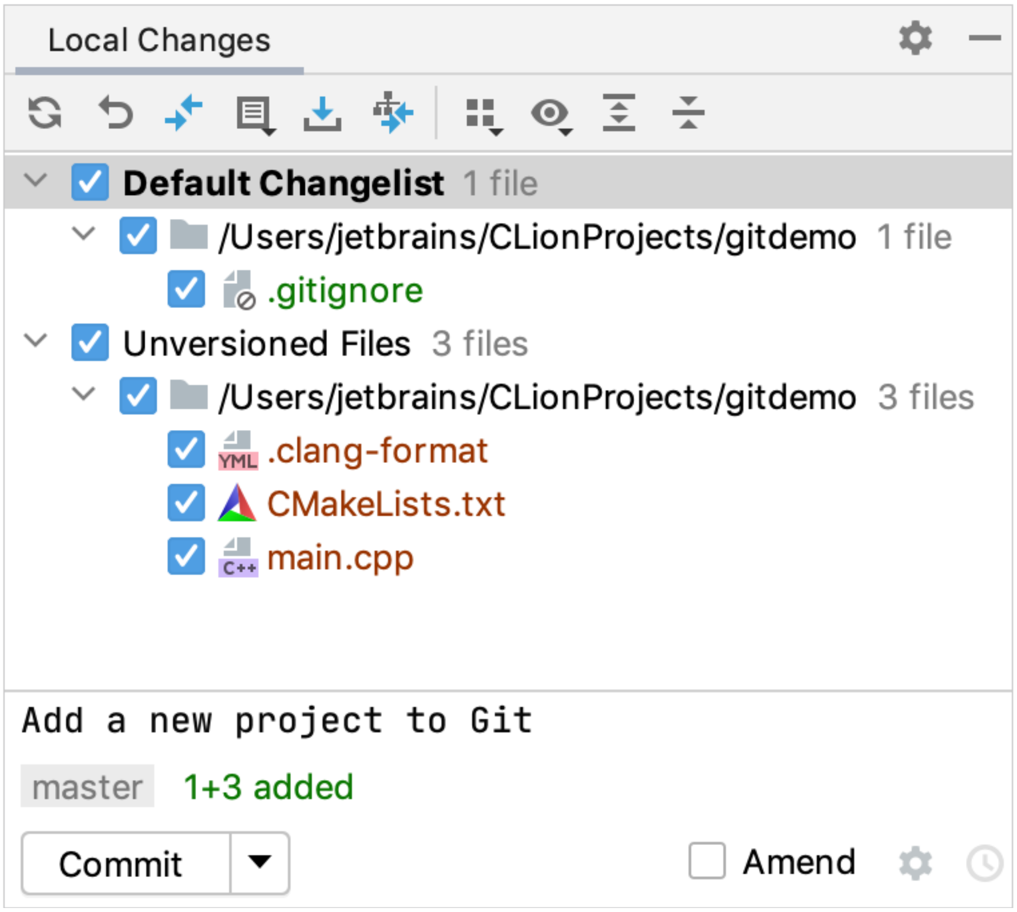
To show the ignored files again, click on the toolbar.

Step 3. Commit your project to the local Git repository

Now when all unnecessary directories are excluded from the list of unversioned files, you just need to add all the files to the repository and commit them to save their current state.

1. In the **Commit** tool window ⌘0, move all the files from the **Unversioned Files** list to **Default Changelist** using drag-and-drop. Select all the files by clicking the root folder checkbox.

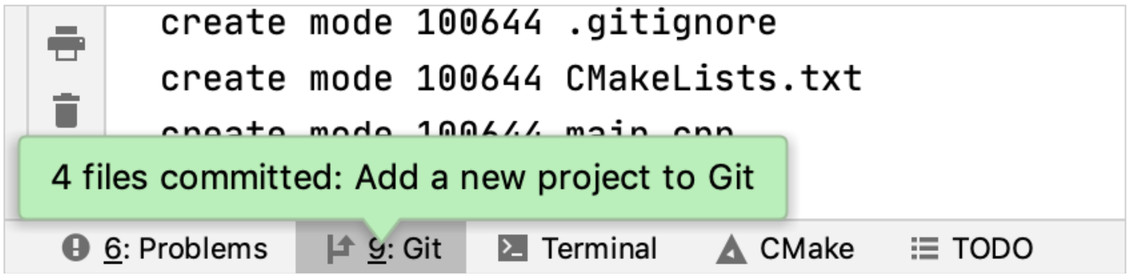
   From the **Commit** tool window ⌘0, you can also preview added and modified files, use the advanced commit options, add and exclude files from the commit, and more.
2. Type a message for your first commit:


Local Changes
Default Changelist 1 file
/Users/jetbrains/CLionProjects/gitdemo 1 file
.gitignore
Unversioned Files 3 files
/Users/jetbrains/CLionProjects/gitdemo 3 files
.clang-format
CMakeLists.txt
main.cpp

```
Add a new project to Git
```

master 1+3 added
Commit
Amend

3. Click **Commit**. The corresponding notification appears once the commit is performed:



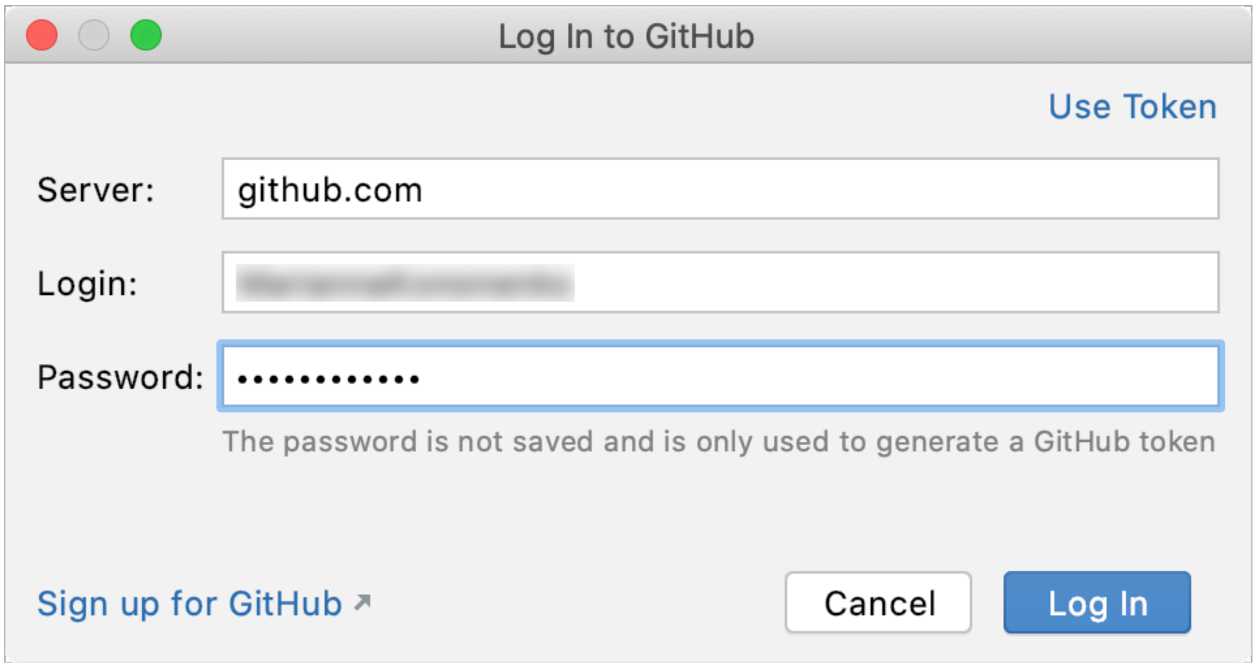
```
create mode 100644 .gitignore
create mode 100644 CMakeLists.txt
create mode 100644 main.cpp
```

4 files committed: Add a new project to Git

Step 4. Share your project on GitHub

To make your project available for other contributors, you need to publish it to a remote repository, for example, on github.com. CLion provides integration with GitHub which allows you to manage projects hosted on GitHub, fork external repositories, manage pull requests, and perform other GitHub operations from the IDE. See more details in the GitHub section.

1. From the main menu, select **VCS | Import into Version Control | Share Project on GitHub**.
2. In the dialog that opens, enter your GitHub login and password and click **Log In**:



Server: github.com

Login: [redacted]

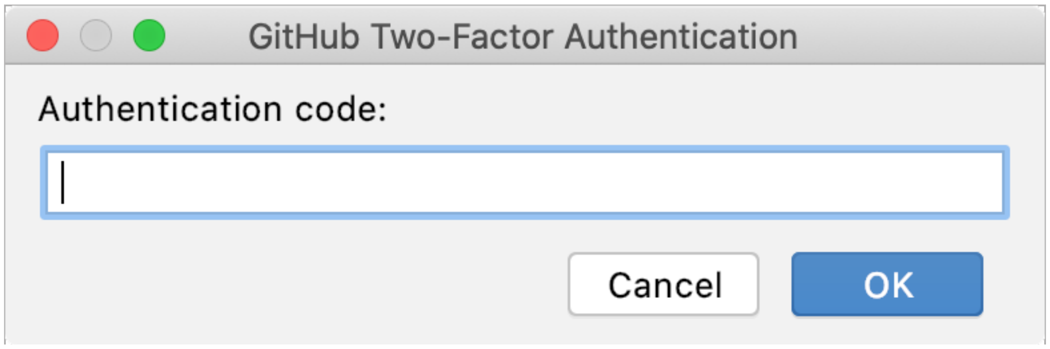
Password: ••••••••••••

The password is not saved and is only used to generate a GitHub token

If you are not registered on GitHub, click **Sign up for GitHub** to go to github.com and create a new account there.

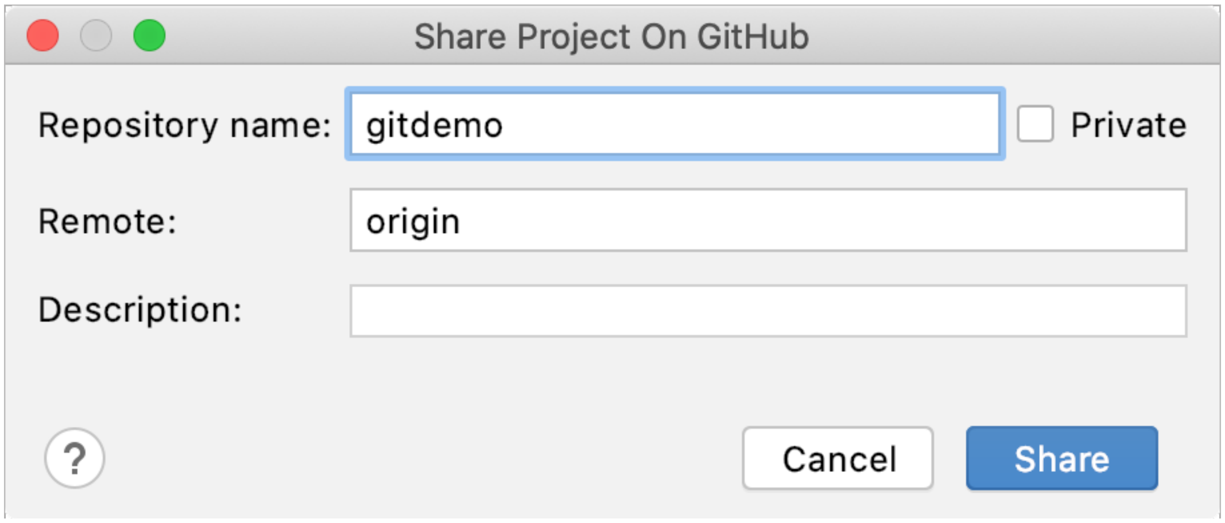
You can change the GitHub account or add a new one in **Preferences | Version Control | GitHub**. See more in Manage multiple accounts.

3. If you have two-factor authentication enabled for GitHub, the following dialog will appear:



Enter the code and click **OK**.

4. In the dialog that opens, you can change the repository name (by default, it is the same as the project name), the name of a remote (by default, origin), choose the repository type (public or private), and add some description if needed:

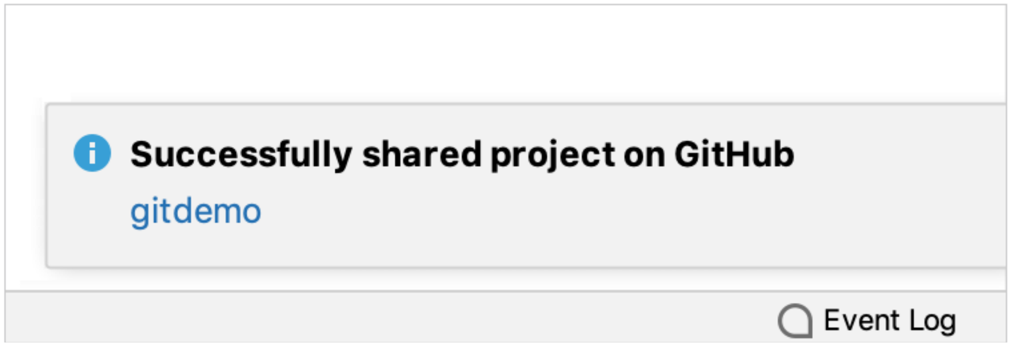


Repository name: gitdemo

Remote: origin

Description:

Click **Share**. After the project is successfully published on GitHub, the following notification will appear:



Successfully shared project on GitHub

gitdemo

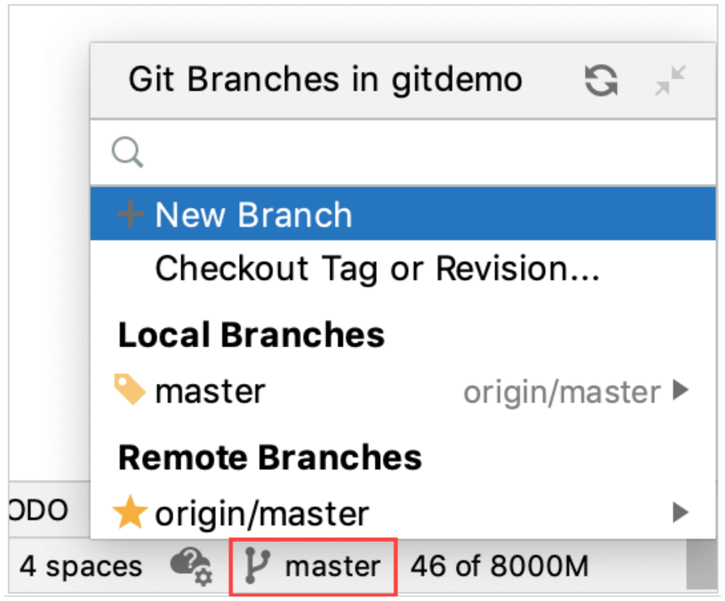
Click the link in the notification to open the repository on GitHub.

You can edit the list of remotes in the **Git Remotes** dialog. To open the dialog, select **Git | Manage Remotes**) from the main menu.

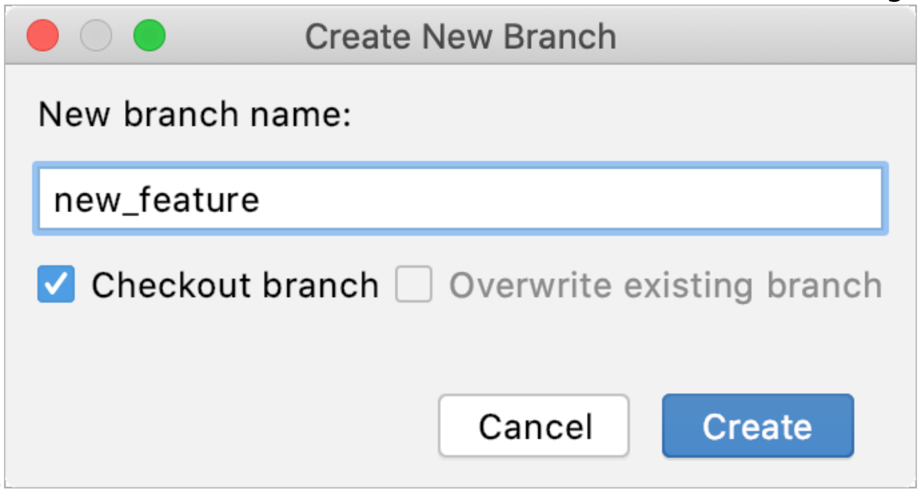
Step 5. Create a new branch

You may need to create a separate branch, for example, when you are working on a new feature and you don't want your changes to get into the main (master) branch before they are tested.

1. Press ⌘T to pull the latest version of the current branch.
2. On the status bar, you can see your current branch — `master`. Click it to open the **Git Branches** menu:

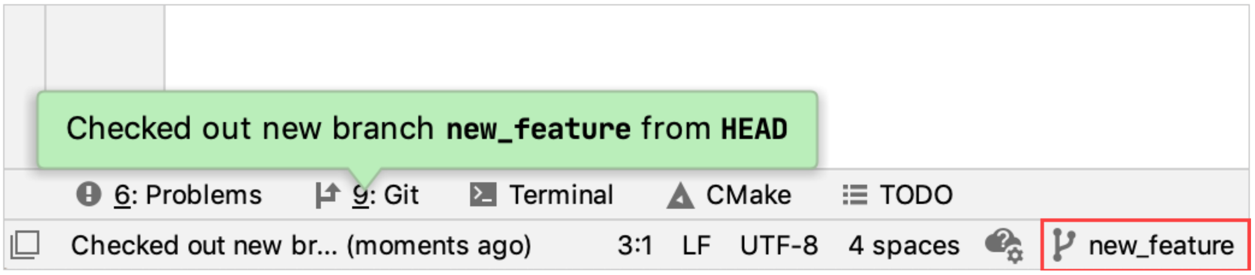


3. From the **Git Branches** menu, select **New Branch**.
4. In the dialog that opens, specify the branch name, for example, `new_feature`, and select the **Checkout branch** checkbox to switch to the new branch right away.

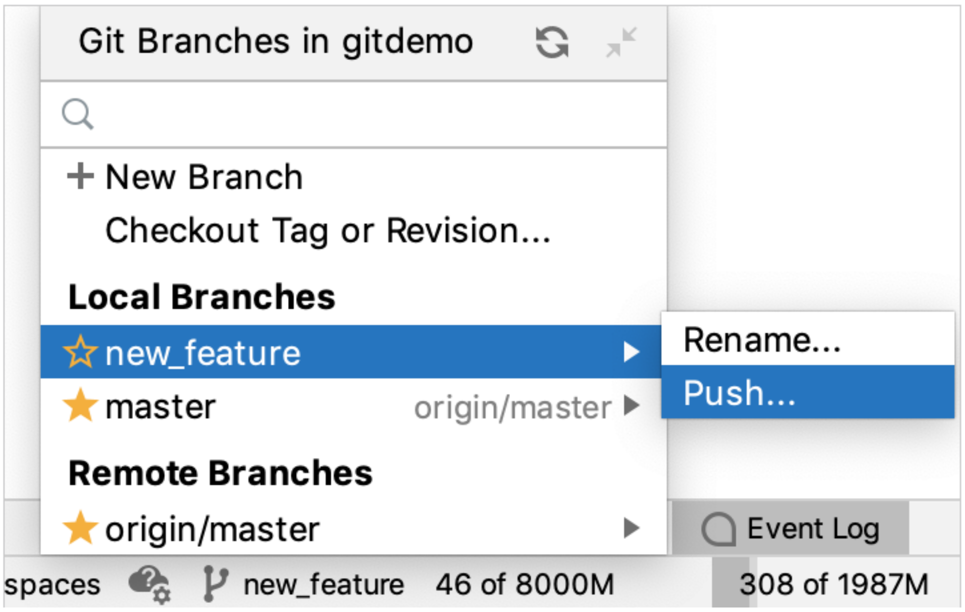


If there are some unversioned changes in the current branch, you need to commit, revert, or shelve them before switching to another branch. See more in Switch between branches.

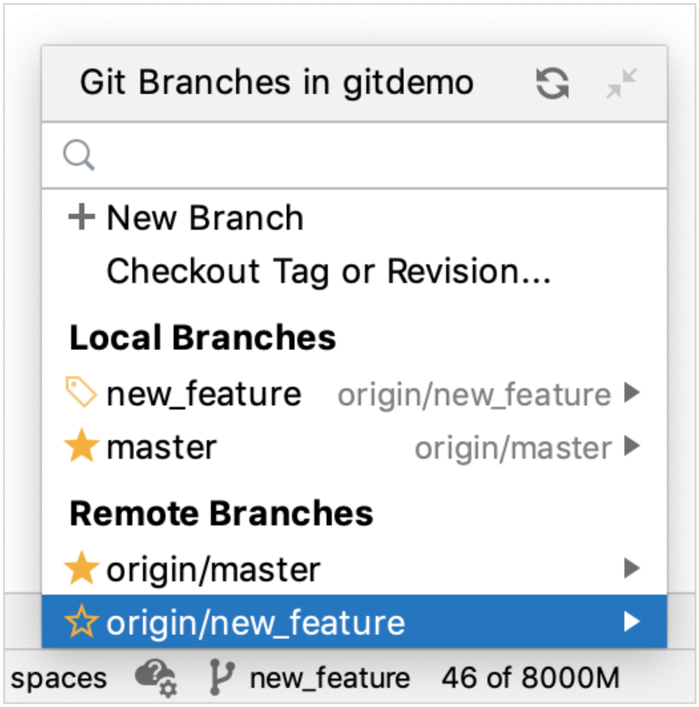
Now you are switched to the newly created branch:


Checked out new branch **new_feature** from **HEAD**

5. Add the corresponding branch to the remote repository. To do this, select the local branch in the **Git Branches** menu and click **Push**:



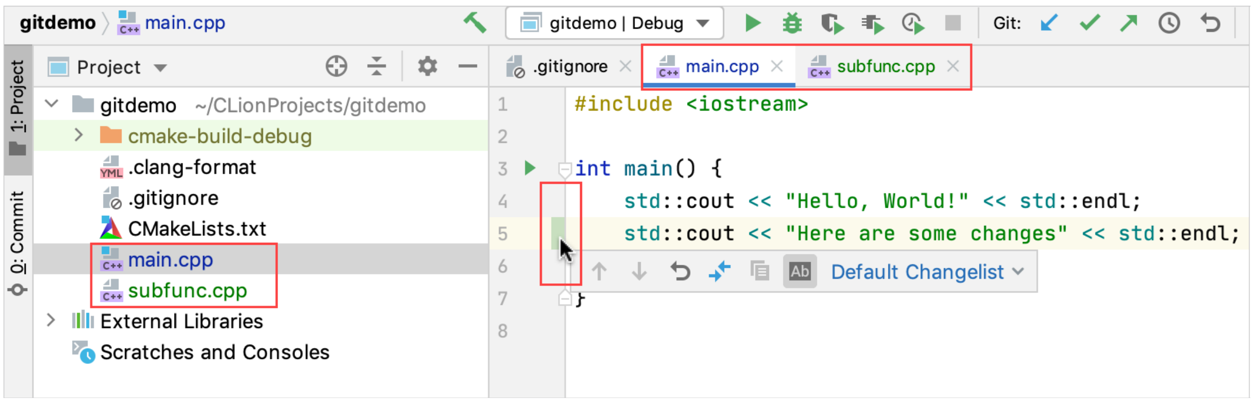
In the dialog that opens, click **Push**. The new branch will be added to the remote repository and will appear in the **Remote Branches** list in the **Git Branches** menu:



Step 6. Make and view the changes

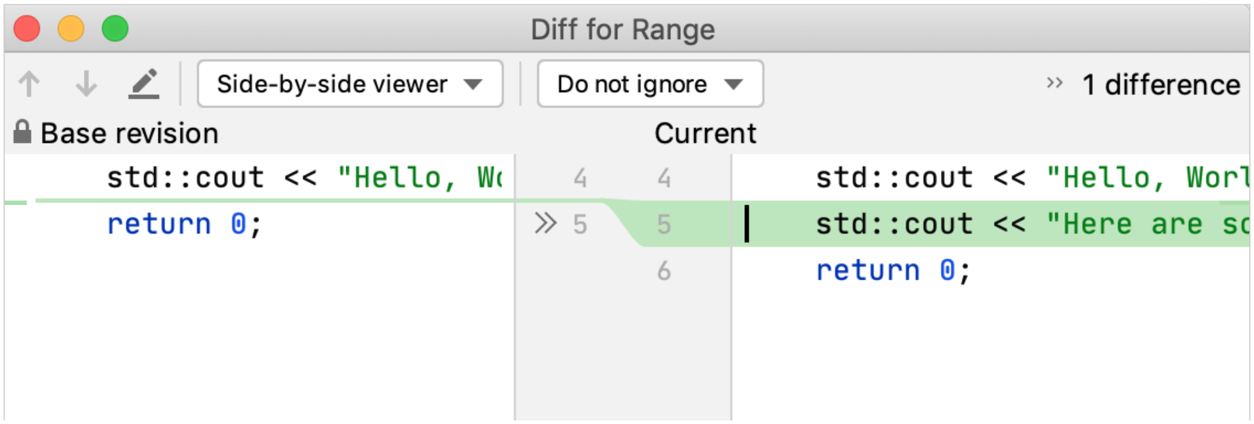
1. Press ⌘T to pull the latest version of the current branch.
2. Add a new file to the project, for example, **subfunc.cpp**, and modify the **main.cpp** file.

   In the **Project** tool window and in the editor tabs, CLion applies different colors to files: blue to modified, green — to newly added. Moreover, in the gutter area of a modified file, the colored change markers appear next to the modified lines. You can click the gutter marker to view details:



```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    std::cout << "Here are some changes" << std::endl;

}
```

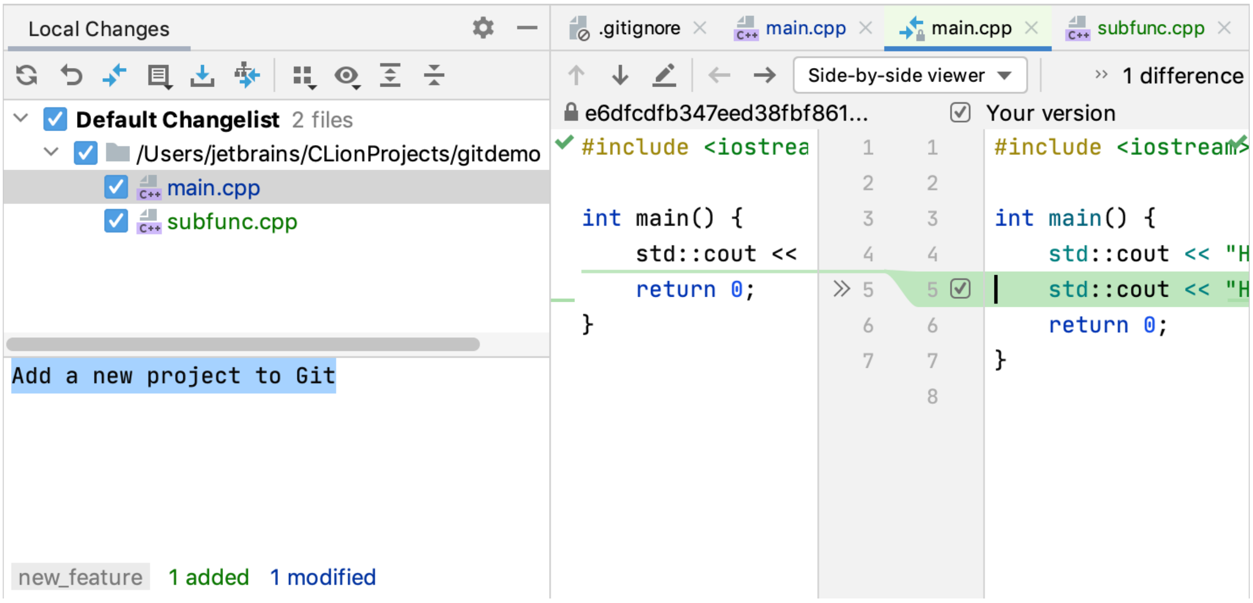
3. Click to view the difference in a separate window:



```
std::cout << "Hello, W
return 0;
```

```
std::cout << "Hello, Worl
std::cout << "Here are sc
return 0;
```

4. Go to the **Commit** tool window ⌘0, to preview all the changes at once. Double-click a file to open the difference view in the editor:



```
Add a new project to Git
```

```
#include <iostrea

int main() {
    std::cout <<
    return 0;
}
```

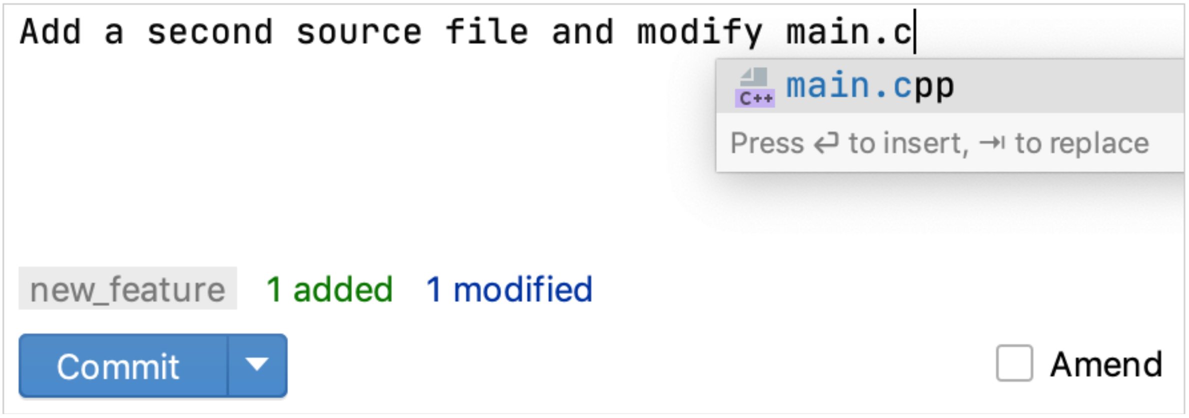
```
#include <iostream>

int main() {
    std::cout << "H
    std::cout << "H
    return 0;
}
```

See more in Review changes.

Step 7. Commit and push the changes

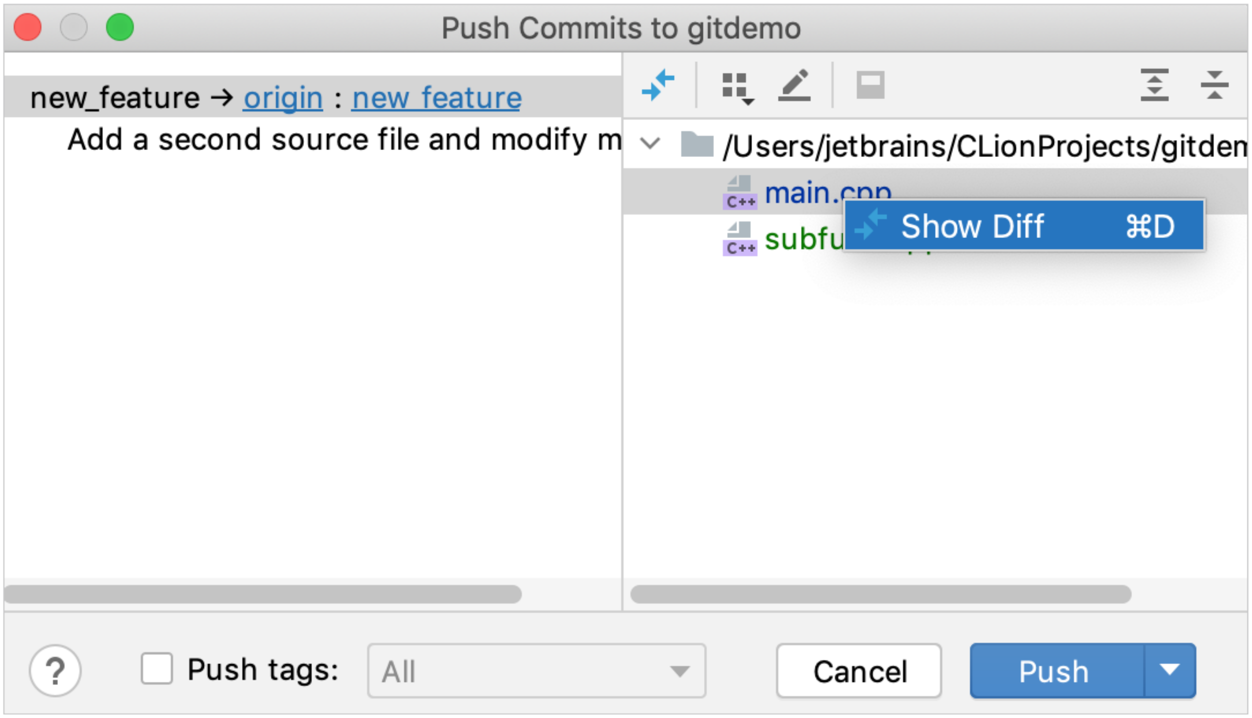
1. Press ⌘T to pull the latest version of the current branch.
2. In the **Commit** tool window ⌘0, review the changes, make sure that the new file that you're adding to Git is selected, type the commit message, and click **Commit**.

   When typing the commit message, you can use auto-completion for the project file names ^Space:



```
Add a second source file and modify main.c
```

3. Press ⇧⌘K or select **Git | Push** from the main menu to push the changes to the remote repository. The **Push Commits** dialog opens. Here you can see all the commits to be pushed as well as all the affected files. Before pushing the changes, you can see the differences for each file. To do this, right-click a file and select **Show Diff** or press ⌘D:



Push Commits to gitdemo
new_feature → origin : new_feature
Add a second source file and modify m
/Users/jetbrains/CLionProjects/gitden
main.cpp
subfu
Show Diff ⌘D
Push tags:
All
Cancel
Push

4. Click **Push**.

To revert a pushed commit, right-click it on the **Log** tab of the **Git** tool window ⌘9 and select **Revert commit**. For more details, go to Revert a pushed commit.

Step 8. Merge branches and resolve conflicts

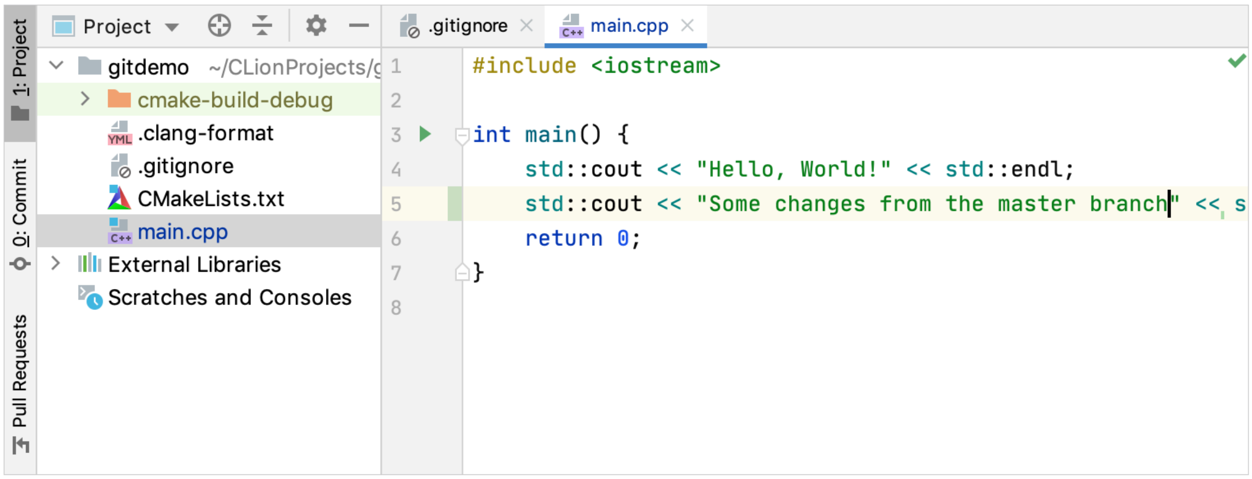
There are several ways how you can apply the changes from one branch to another, such as merging and rebasing the branches, cherry-picking commits, applying separate changes or files. All these methods are described in details in Apply changes from one Git branch to another.

In this tutorial, you will learn how to merge two branches and cherry-pick commits.

Merge branches

1. Select the `master` branch in the Git Branches menu on the status bar and click **Checkout**.

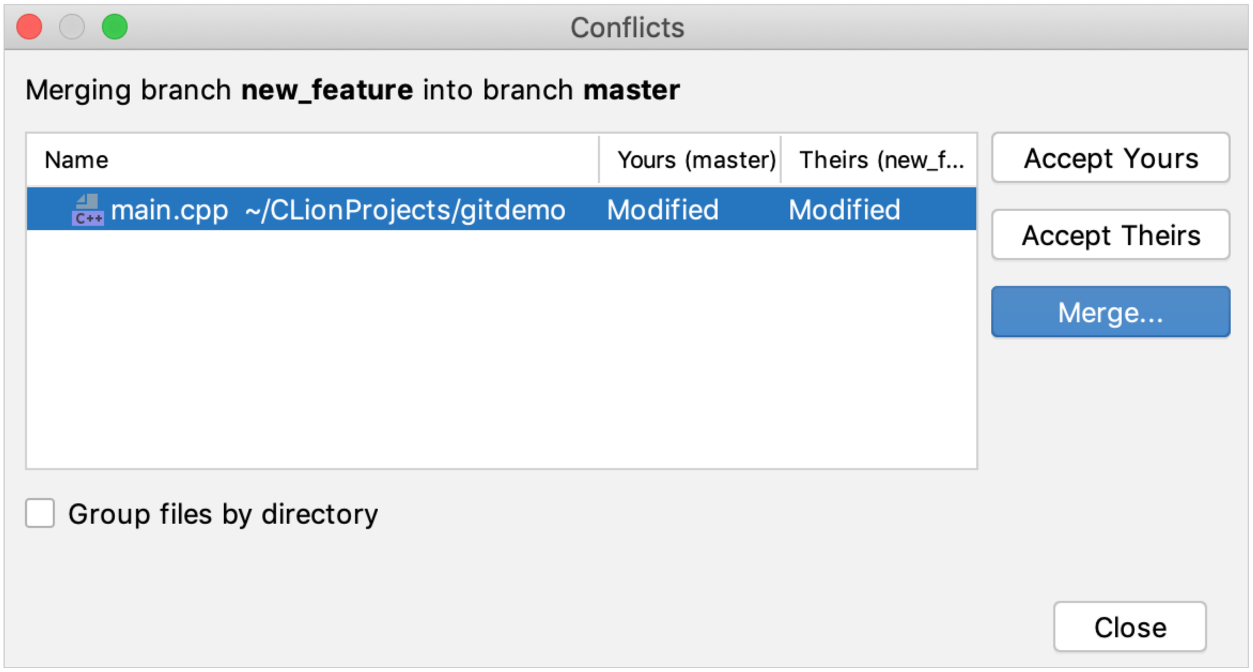
   If there are some unversioned changes in the current branch, you need to commit, revert, or shelve them before switching to another branch. See more in Switch between branches.
2. Make some changes in **main.cpp** that has been modified before in the `new_feature` branch, for example:



```
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    std::cout << "Some changes from the master branch" << s
    return 0;
}
```

   This way, we simulate a conflict situation. Push and commit the change as described above.
3. From the **Git Branches** menu, select **new_feature** and click **Merge into Current**.

Since we have made changes for the same file in different branches, the **Conflicts** dialog appears.


Conflicts

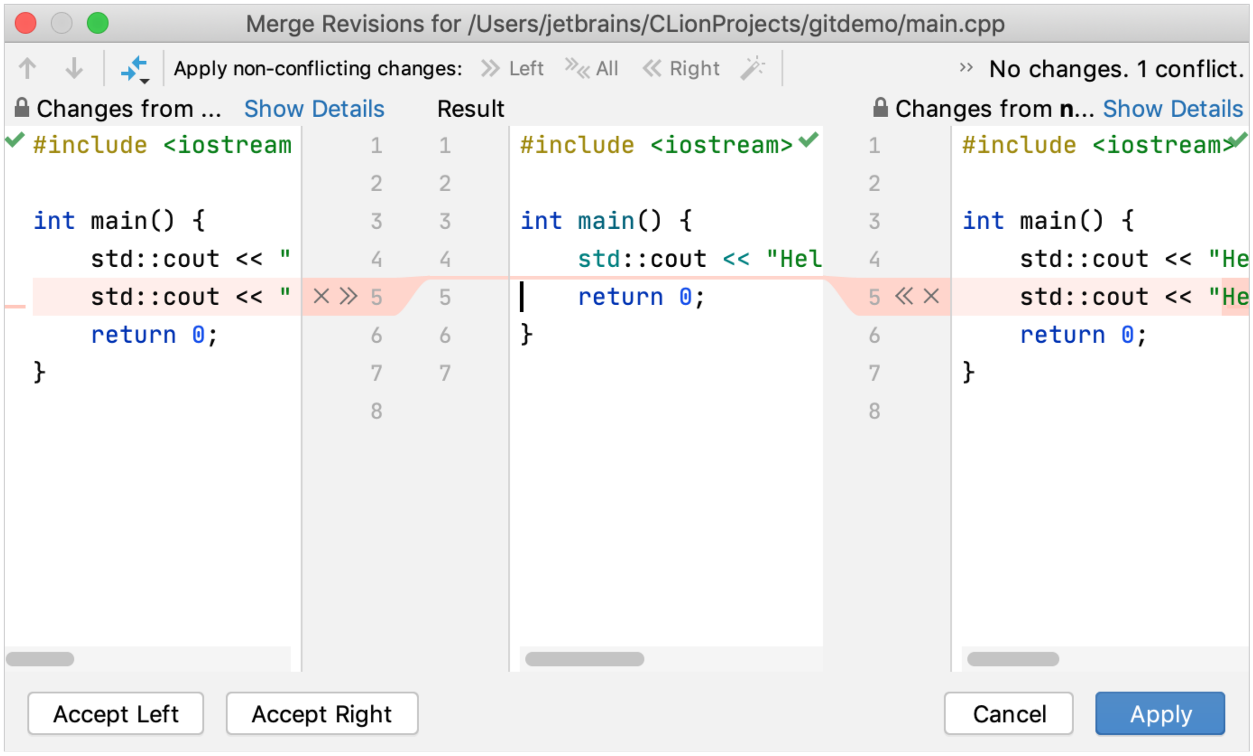
Merging branch **new_feature** into branch **master**

Name	Yours (master)	Theirs (new_f...
main.cpp ~/CLionProjects/gitdemo	Modified	Modified

Resolve conflicts

1. In the **Conflicts** dialog, you have several options to resolve the conflict:
    - **Accept Yours**: keep the changes made in the current branch.
    - **Accept Theirs**: apply the changes from the branch that you want to merge into the current one.
    - **Merge**: resolve conflicts manually in a dedicated dialog.

   Click **Merge**. The **Merge Revisions** dialog opens:



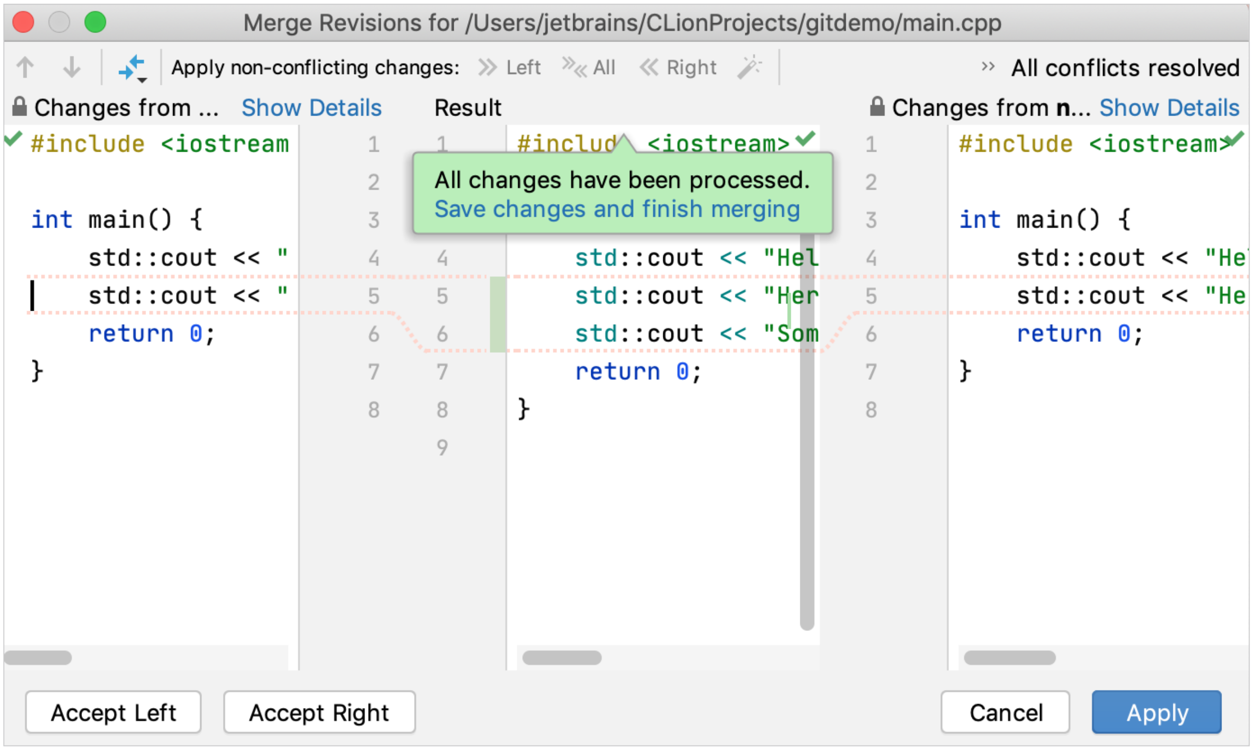
```
#include <iostream

int main() {
    std::cout << "
    std::cout << "
    return 0;
}
```

```
#include <iostream>

int main() {
    std::cout << "Hel
    return 0;
}
```

```
#include <iostream>

int main() {
    std::cout << "He
    std::cout << "He
    return 0;
}
```

2. In this dialog, you can accept changes by clicking / , decline them by clicking , and type code in the **Result** pane. See more in Resolve Git conflicts.

The syntax error are highlighted in the **Merge Revisions** window:


Merge Revisions for /Users/jetbrains/CLionProjects/gitdemo/main.cpp

Apply non-conflicting changes: Left All Right All conflicts resolved

Changes from ... Show Details | Result | Changes from n... Show Details

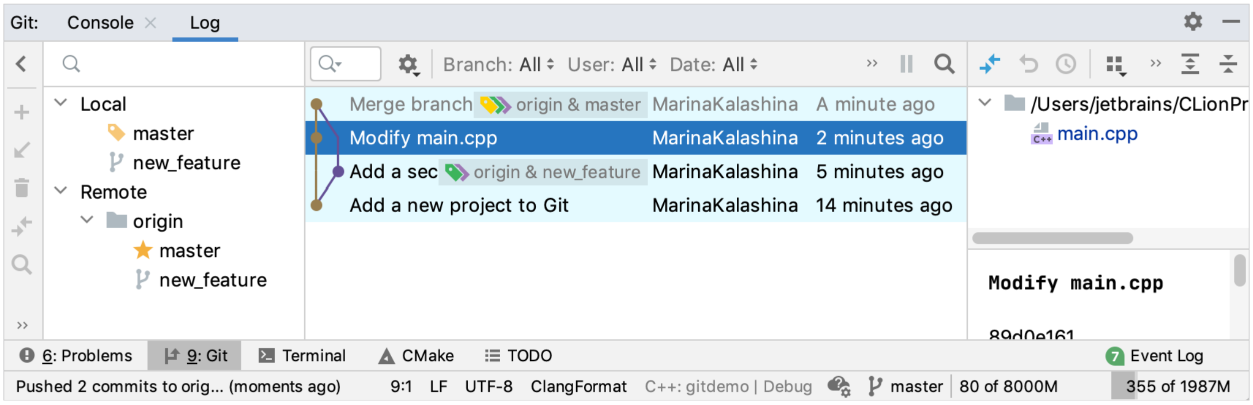
```
#include <iostream>                 #includ  <iostream>                  #include <iostream>

int main() {                                                              int main() {
    std::cout << "                      std::cout << "Hel                     std::cout << "He
    std::cout << "                      std::cout << "Her                     std::cout << "He
    return 0;                           std::cout << "Som                     return 0;
}                                       return 0;                         }
                                    }
```

All changes have been processed.
Save changes and finish merging

Resolve the conflict as on the screenshot above and click **Apply**.

3. Push the changes to the remote repository by pressing ⇧⌘K or selecting **Git | Push** from the main menu.

You can see the commits in all the branches on the **Log** tab of the **Git** tool window ⌘9:


Git: Console Log
Branch: All User: All Date: All
Local
master
new_feature
Remote
origin
master
new_feature
6: Problems 9: Git Terminal CMake TODO Event Log

Merge branch	origin & master	MarinaKalashina	A minute ago
Modify main.cpp		MarinaKalashina	2 minutes ago
Add a sec	origin & new_feature	MarinaKalashina	5 minutes ago
Add a new project to Git		MarinaKalashina	14 minutes ago

/Users/jetbrains/CLionPr
main.cpp

Modify main.cpp

89d0e161

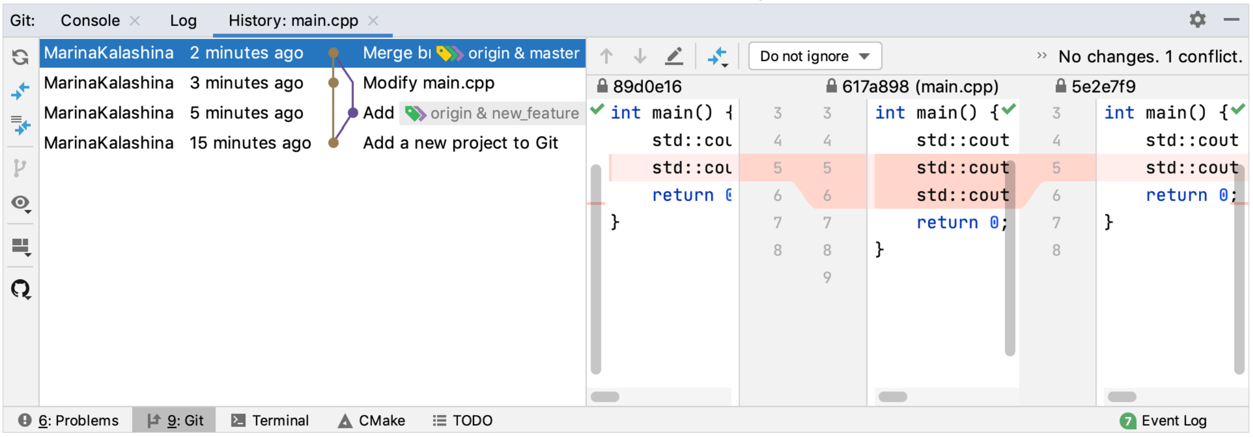
Pushed 2 commits to orig... (moments ago) 9:1 LF UTF-8 ClangFormat C++: gitdemo | Debug master 80 of 8000M 355 of 1987M

From here, you can also revert commits, cherry-pick changes from one branch to another, and more. See Log tab for more details.

Step 9. View history

In the `master` branch, open the **main.cpp** file. You see that the file's syntax is broken — there are two containers within one view. Imagine these changes were made long ago or by someone else: you might be interested which commit this code came from. To find it out, do one of the following:

- Right-click the file in the editor or in the **Project** tool window ⌘1 and select **Git | Show History**. The **History** tab of the **Git** tool window opens:



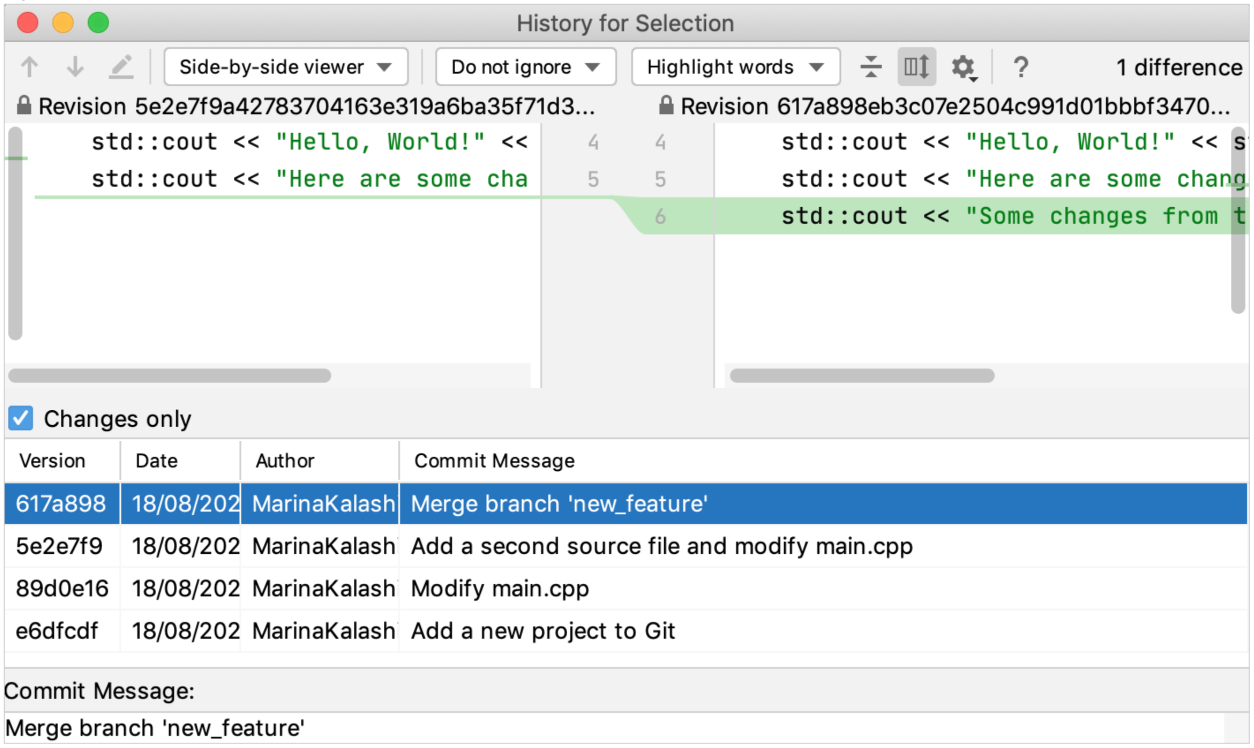
```
MarinaKalashina   2 minutes ago    Merge bı  origin & master
MarinaKalashina   3 minutes ago    Modify main.cpp
MarinaKalashina   5 minutes ago    Add  origin & new_feature
MarinaKalashina   15 minutes ago   Add a new project to Git
```

No changes. 1 conflict.

```
89d0e16              617a898 (main.cpp)      5e2e7f9
int main() {         int main() {            int main() {
    std::cou             std::cout               std::cout
    std::cou             std::cout               std::cout
    return 0             std::cout               return 0;
}                        return 0;           }
                     }
```

  On this tab you can view all commits that affected the file and find out in which commit the change of your interest was added.

- In the editor, select a code fragment you want to view history for, right-click the selection, and choose **Git | Show History for Selection**. The **History for Selection** window will open:



History for Selection

Side-by-side viewer | Do not ignore | Highlight words | 1 difference

Revision 5e2e7f9a42783704163e319a6ba35f71d3... | Revision 617a898eb3c07e2504c991d01bbbf3470...

```
std::cout << "Hello, World!" <<          4  4  std::cout << "Hello, World!" << s
std::cout << "Here are some cha          5  5  std::cout << "Here are some chang
                                            6  std::cout << "Some changes from t
```

☑ Changes only

Version	Date	Author	Commit Message
617a898	18/08/202	MarinaKalash	Merge branch 'new_feature'
5e2e7f9	18/08/202	MarinaKalash	Add a second source file and modify main.cpp
89d0e16	18/08/202	MarinaKalash	Modify main.cpp
e6dfcdf	18/08/202	MarinaKalash	Add a new project to Git

Commit Message:

Merge branch 'new_feature'

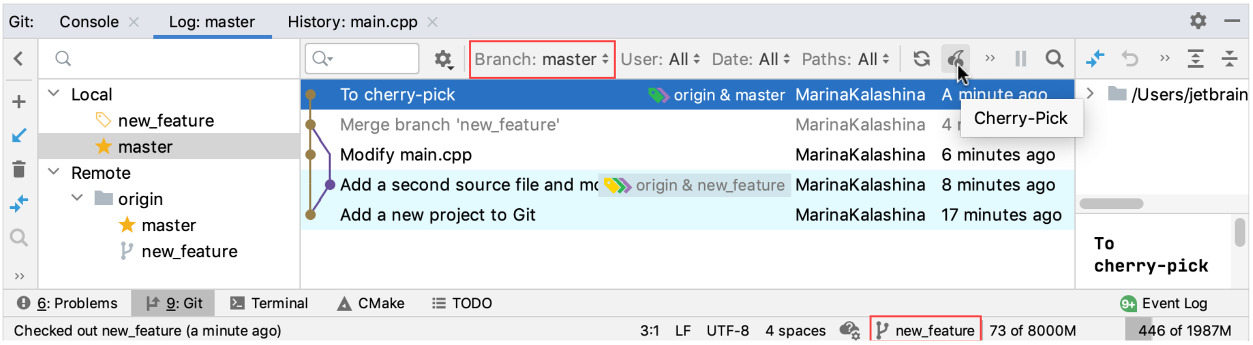
Here you can review all the commits that affected the code selection of your interest.

Find more ways of exploring the Git history in Investigate changes in Git repository.

Step 10. Cherry-pick commits

This operation allows you to apply changes from selected commits from one branch to another.

1. In the `master` branch, make a change that you are going to cherry-pick. Commit ⌘K and push ⇧⌘K this change.
2. Switch to the `new_feature` branch.
3. In the **Git** tool window ⌘9, open the **Log** tab.
4. In the **Branch** list, select `master`.
5. Select your last commit and click :

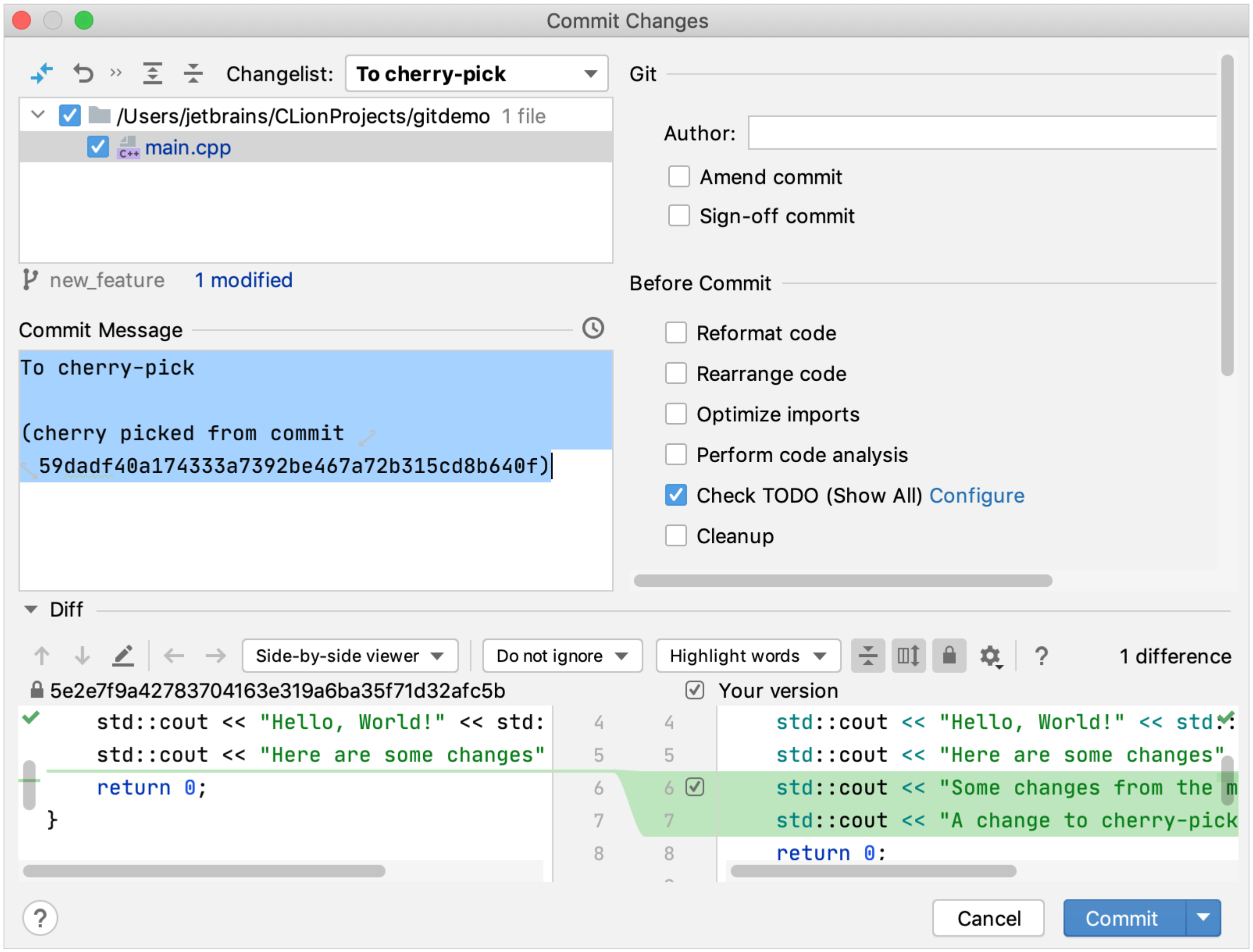

Git: Console Log: master History: main.cpp
Branch: master User: All Date: All Paths: All
Local
new_feature
master
Remote
origin
master
new_feature
Cherry-Pick
/Users/jetbrain
6: Problems 9: Git Terminal CMake TODO Event Log

To cherry-pick origin & master	MarinaKalashina	A minute ago
Merge branch 'new_feature'	MarinaKalashina	4
Modify main.cpp	MarinaKalashina	6 minutes ago
Add a second source file and mo origin & new_feature	MarinaKalashina	8 minutes ago
Add a new project to Git	MarinaKalashina	17 minutes ago

```
To
cherry-pick
```

Checked out new_feature (a minute ago) 3:1 LF UTF-8 4 spaces new_feature 73 of 8000M 446 of 1987M

6. Resolve the conflict by pressing **Accept Theirs** form the dialog that opens. This means we override the code in the **main.cpp** file by the changes from the cherry-picked commit.
7. Commit the changes from the **Commit Changes** dialog that opens and push them ⇧⌘K.



```
To cherry-pick

(cherry picked from commit
59dadf40a174333a7392be467a72b315cd8b640f)
```

5e2e7f9a42783704163e319a6ba35f71d32afc5b

```
    std::cout << "Hello, World!" << std:
    std::cout << "Here are some changes"
    return 0;
}
```

Your version

```
    std::cout << "Hello, World!" << std:
    std::cout << "Here are some changes"
    std::cout << "Some changes from the m
    std::cout << "A change to cherry-pick
    return 0;
```

If no conflicts emerge, the cherry-pick will be committed automatically, and you will just need to push the changes.