

# Setting up a GIT repo for Django Application using PyCharm



Authorized & published by Summitworks Technologies Inc



# Agenda

- **Set up a Git repository for Django Application**

# Git

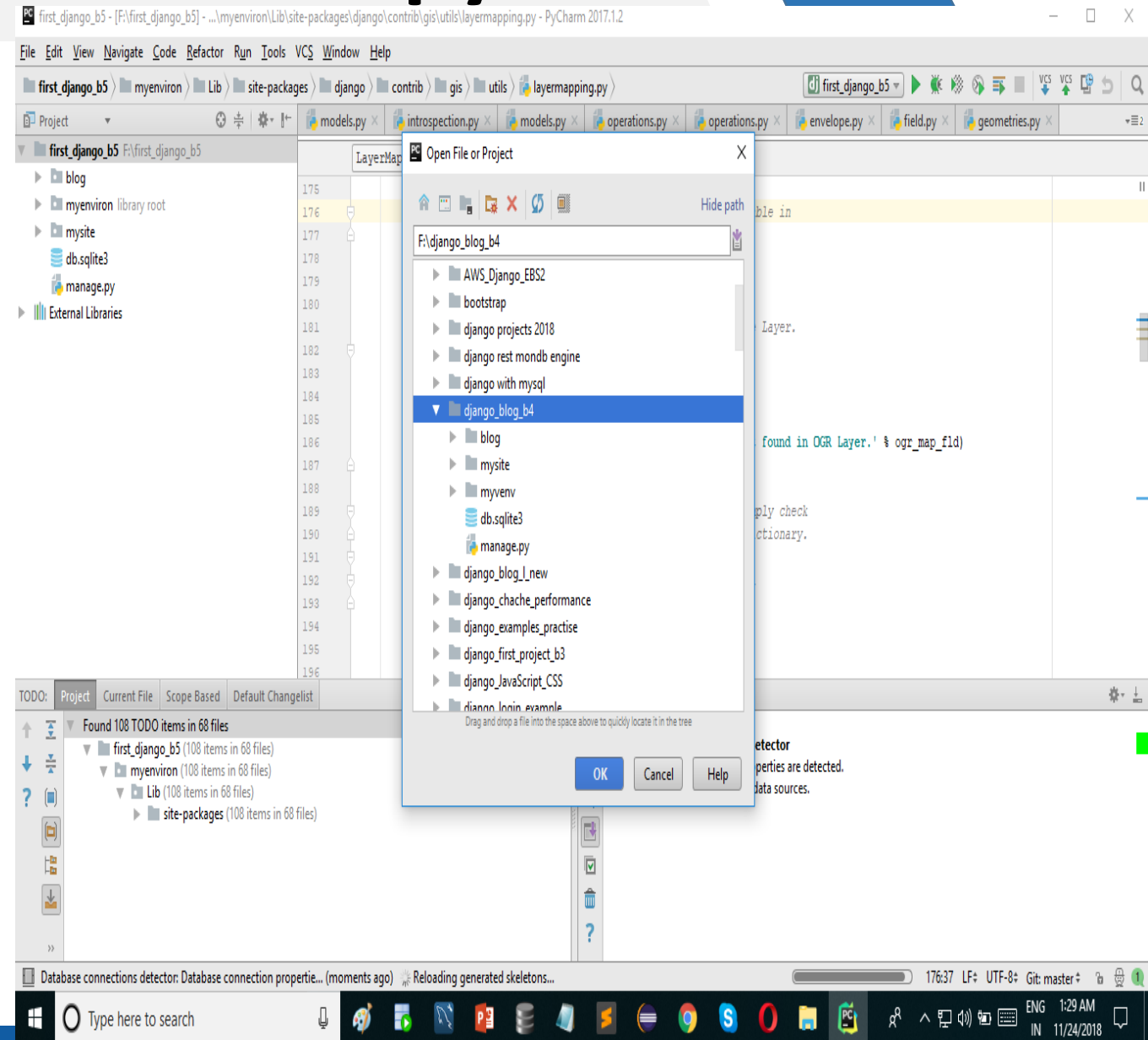
- **Git** is a revision control system, a tool to manage your source code history.
- Its for tracking changes in computer files and coordinating work on those files among multiple people. It is primarily used for source-code management in software development, but it can be used to keep track of changes in any set of files.
- **You can use IDE tools or Commands to work with GIT**

# Open virtual environment into pycharm

Use pycharm professional edition.[ its trail version ]  
Open virtual environment or create a new Django project in pycharm.

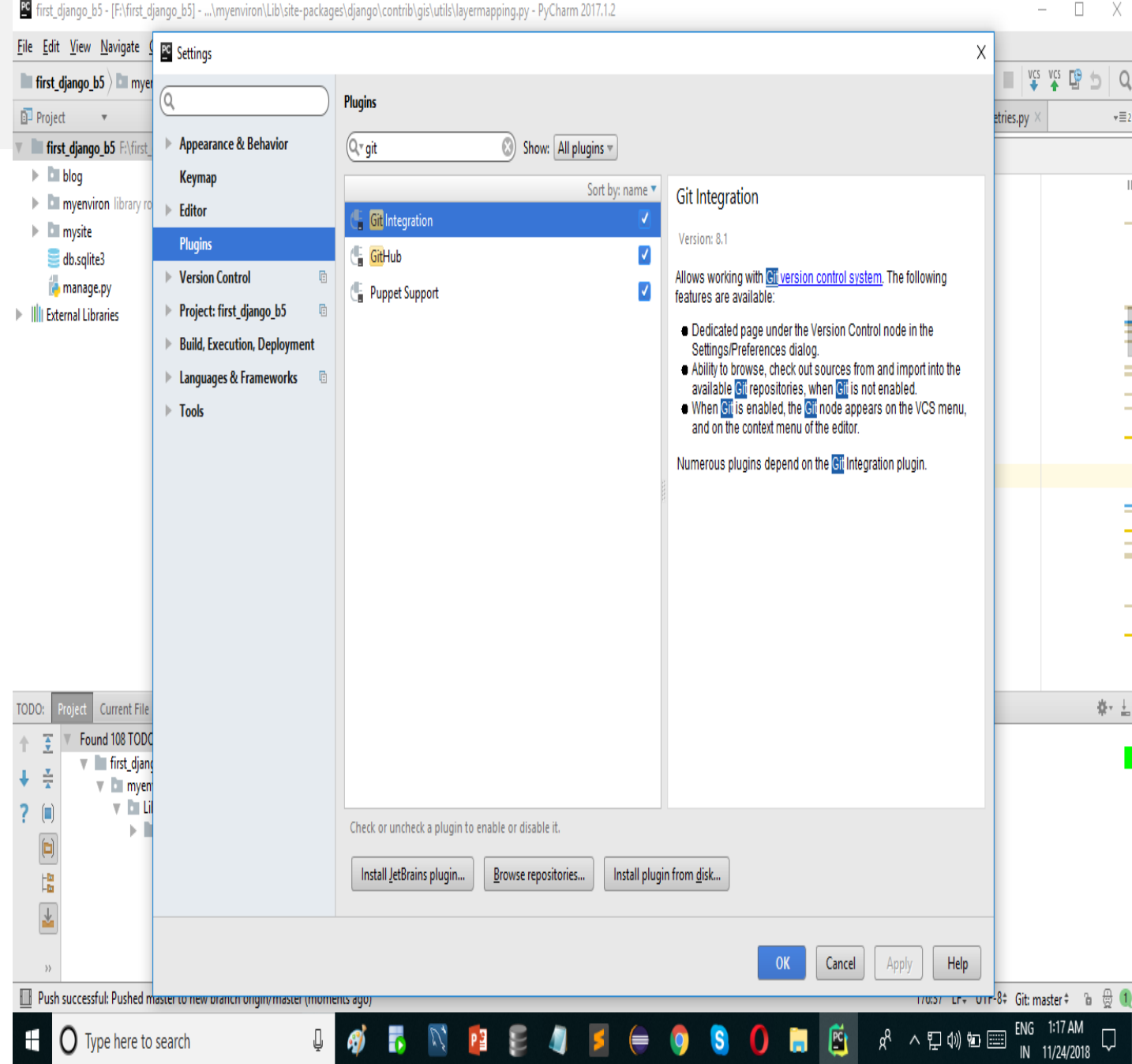
You can open virtual environment like below.  
Click file → open project → select the location of virtual environment.

[ make sure you configure python interpreter before you create / import Django project.]



# Git in Pycharm

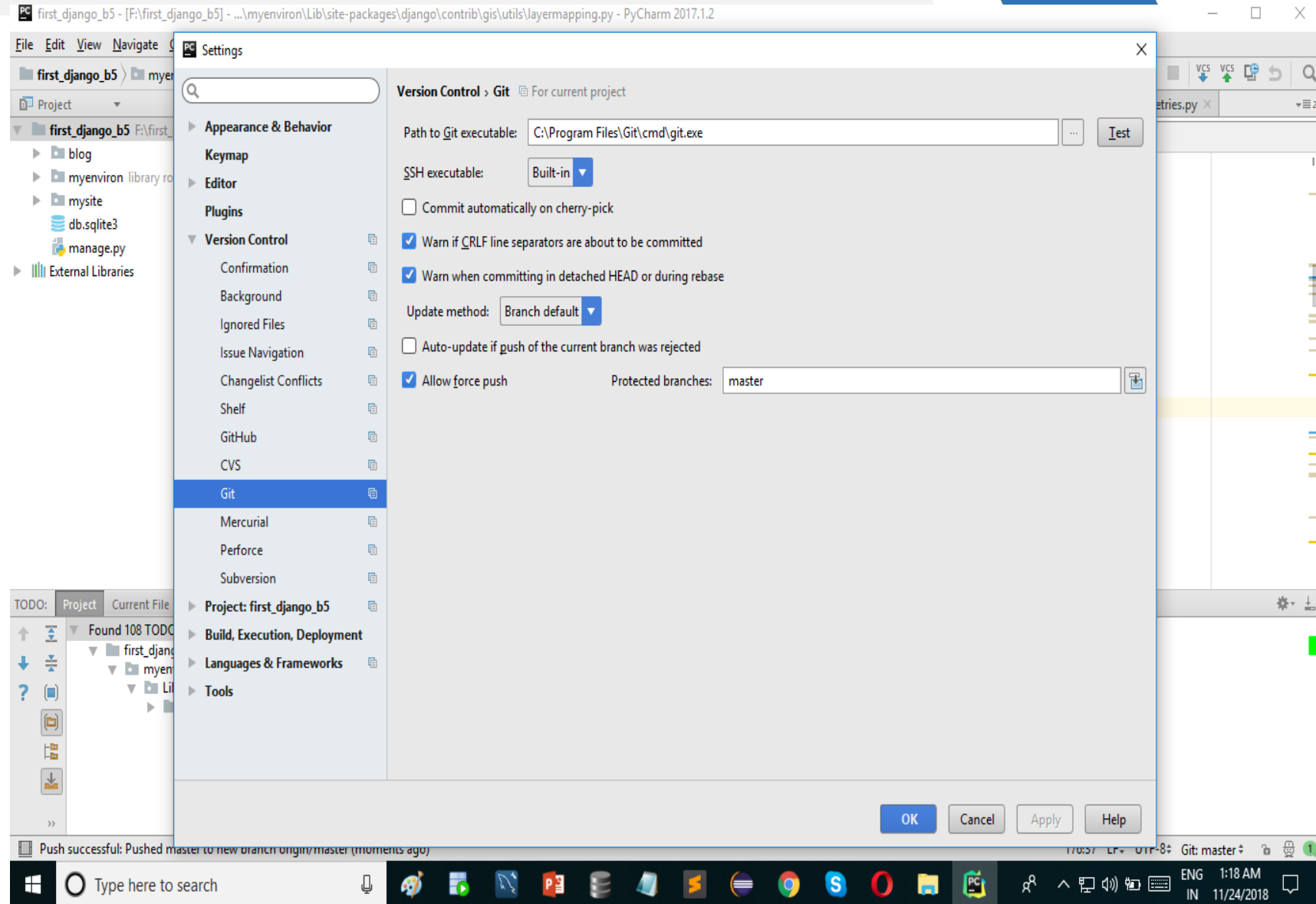
To be able to use Git for version control, make sure that the Git Integration plugin is enabled in the Settings/Preferences dialog (Ctrl+Alt+S) under Plugins.



# Git in Pycharm

Before you can enable Git version control for an existing local project, or clone a Git project from a remote repository, do the following:

- Download and install Git.
- In the Settings/Preferences dialog (Ctrl+Alt+S), select Version Control | Git in the left pane and specify the path to the Git executable.



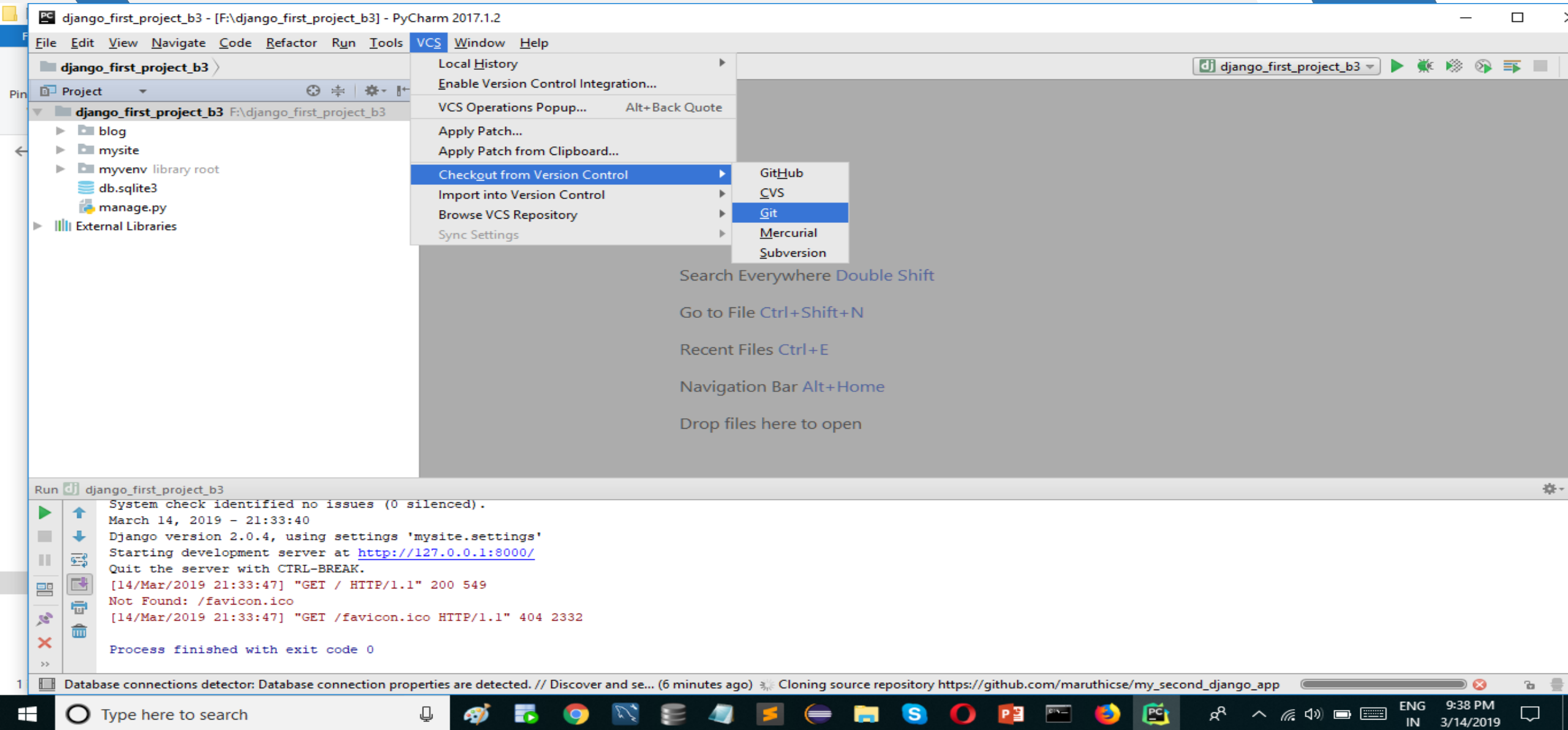
# Set up a Git repository - pycharm

- **Check out a project from a remote host (clone):** PyCharm allows you to check out (in Git terms **clone**) an existing repository and create a new project based on the data you've downloaded.
  - From the main menu, choose VCS | Checkout from Version Control | Git, or, if no project is currently opened, choose Checkout from Version Control | Git on the Welcome screen.
  - In the Clone Repository dialog, specify the URL of the remote repository you want to clone (you can click Test to make sure that connection to the remote can be established).
  - In the Directory field, specify the path where the folder for your local Git repository will be created into which the remote repository will be cloned.
  - Click Clone. If you want to create a PyCharm project based on the sources you have cloned, click Yes in the confirmation dialog. Git root mapping will be automatically set to the project root directory.

**If your project contains submodules, they will also be cloned and automatically registered as project roots.**

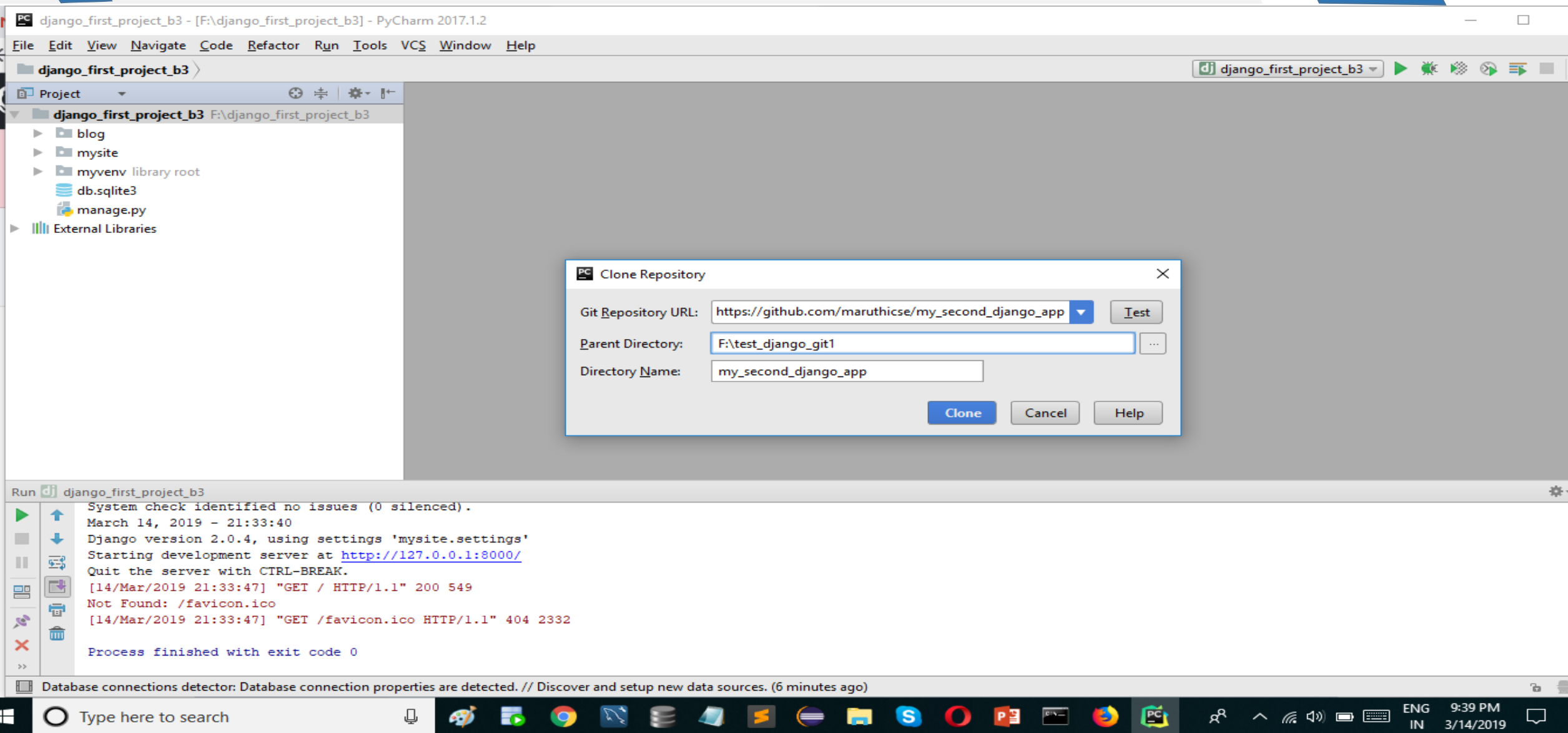


# Set up a Git repository - pycharm

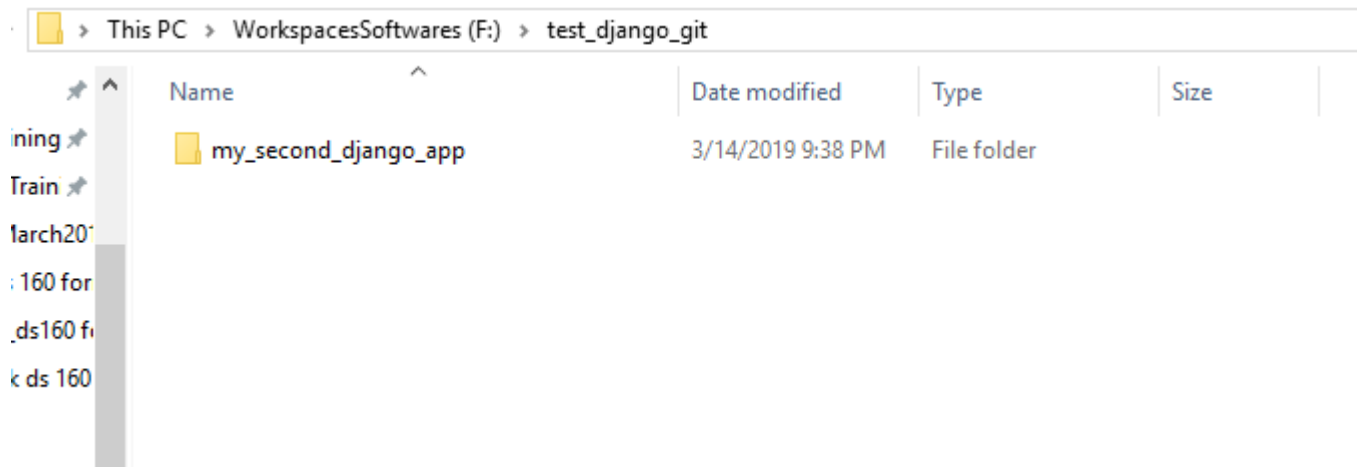




# Set up a Git repository - pycharm



# Set up a Git repository - pycharm



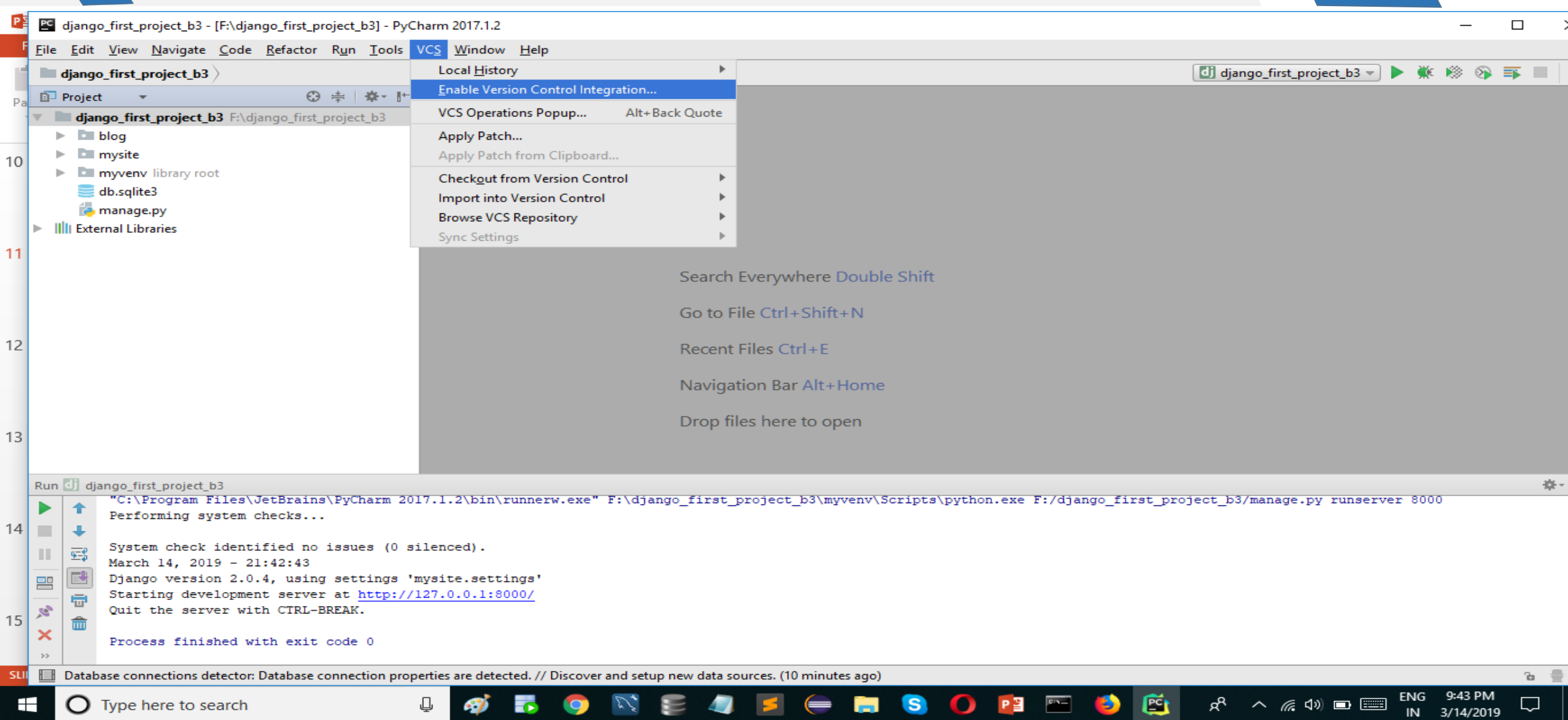
# Put an existing project under Git version control

**Apart from cloning a remote repository, you can create a local repository based on an existing project's sources.**

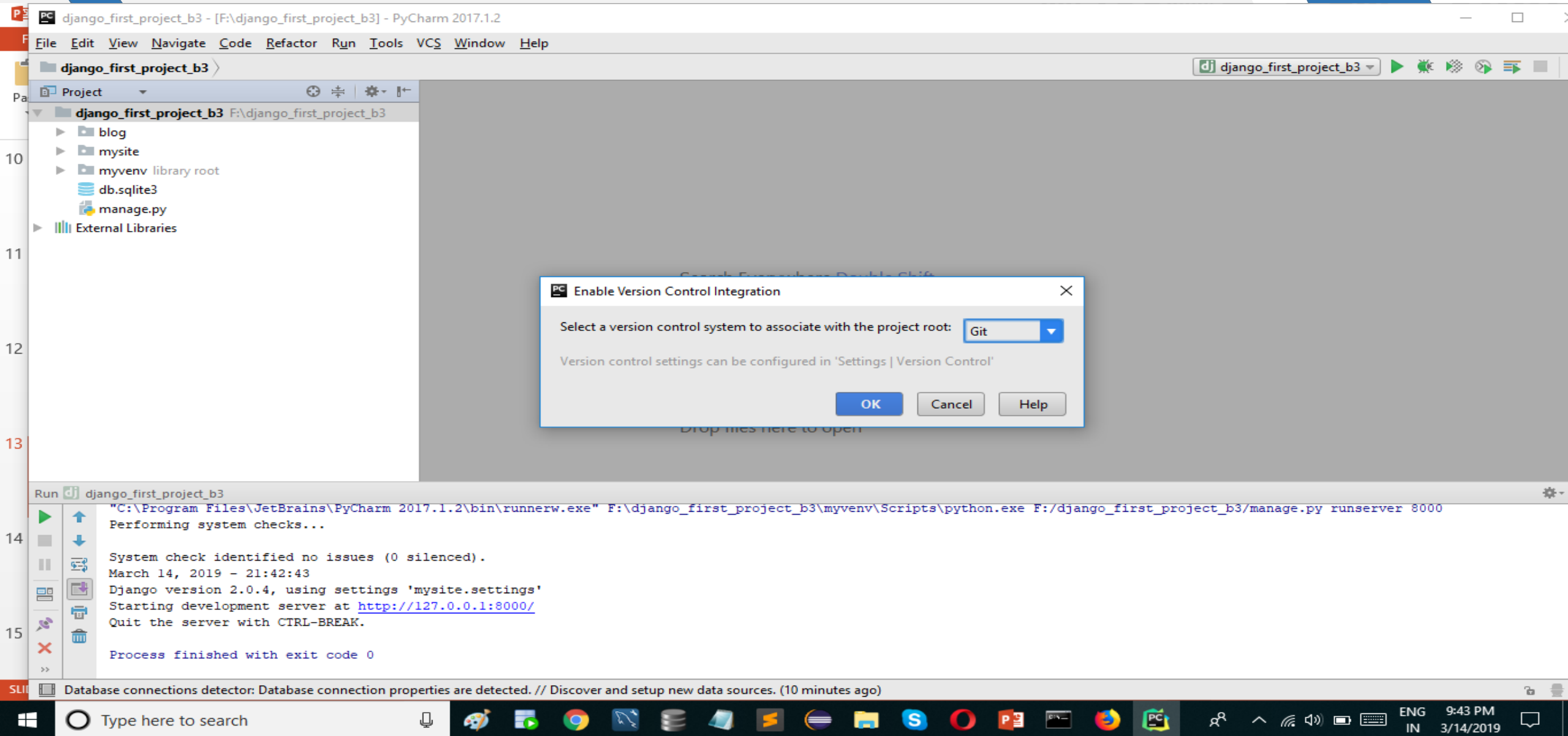
Import an entire project into a single Git repository

- Open the project that you want to put under Git.
- From the main menu, choose VCS | Enable Version Control Integration.
- In the dialog that opens, select Git from the drop-down list and click OK.

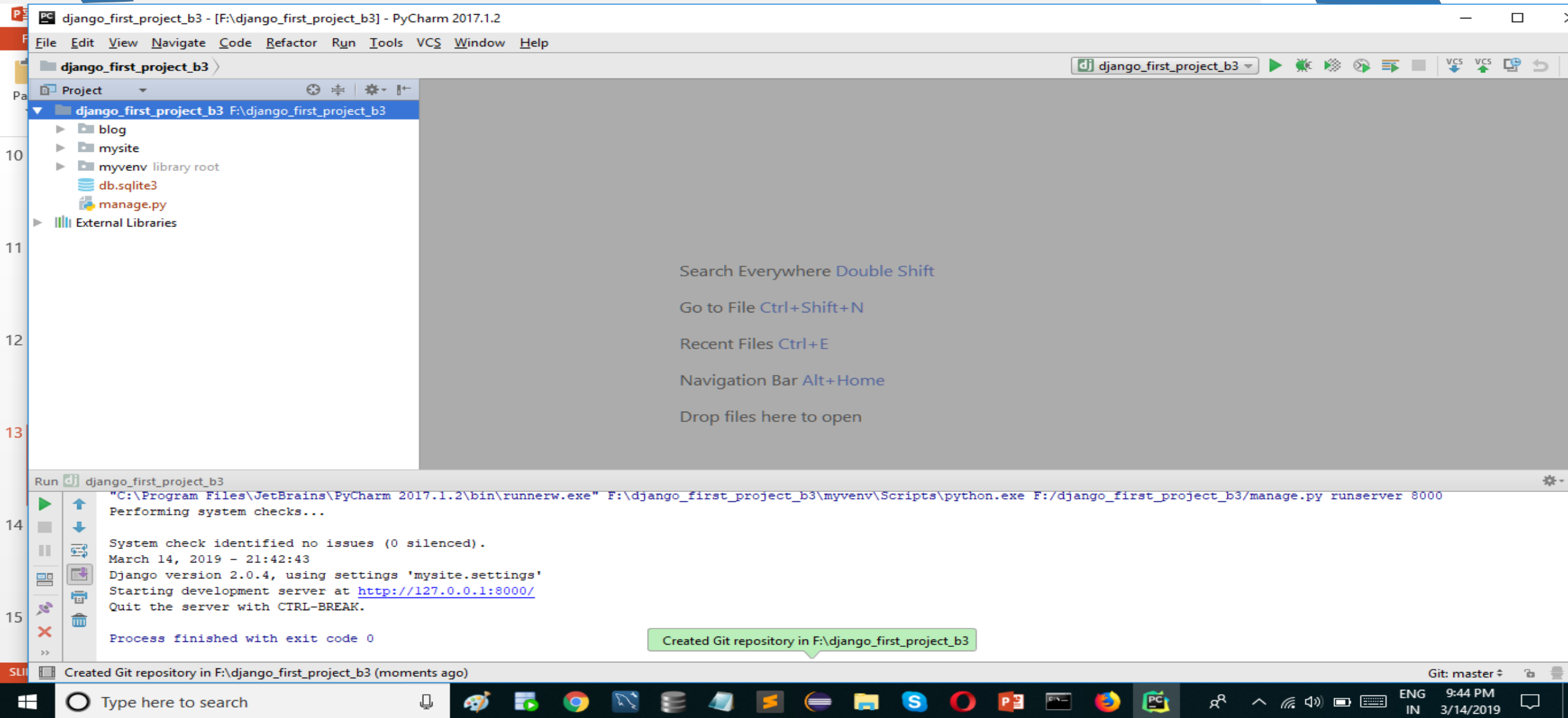
# Put an existing project under Git version control



# Put an existing project under Git version control



# Put an existing project under Git version control



# Add files to the local repository

After you have initialized a Git repository for your project, you need to add project data to it.

Open the Version Control tool window (Alt+9) and switch to the Local Changes tab.

Put any files in the Unversioned Files changelist under version control by pressing Ctrl+Alt+A or selecting Add to VCS from the context menu. You can either add the entire changelist, or select separate files.

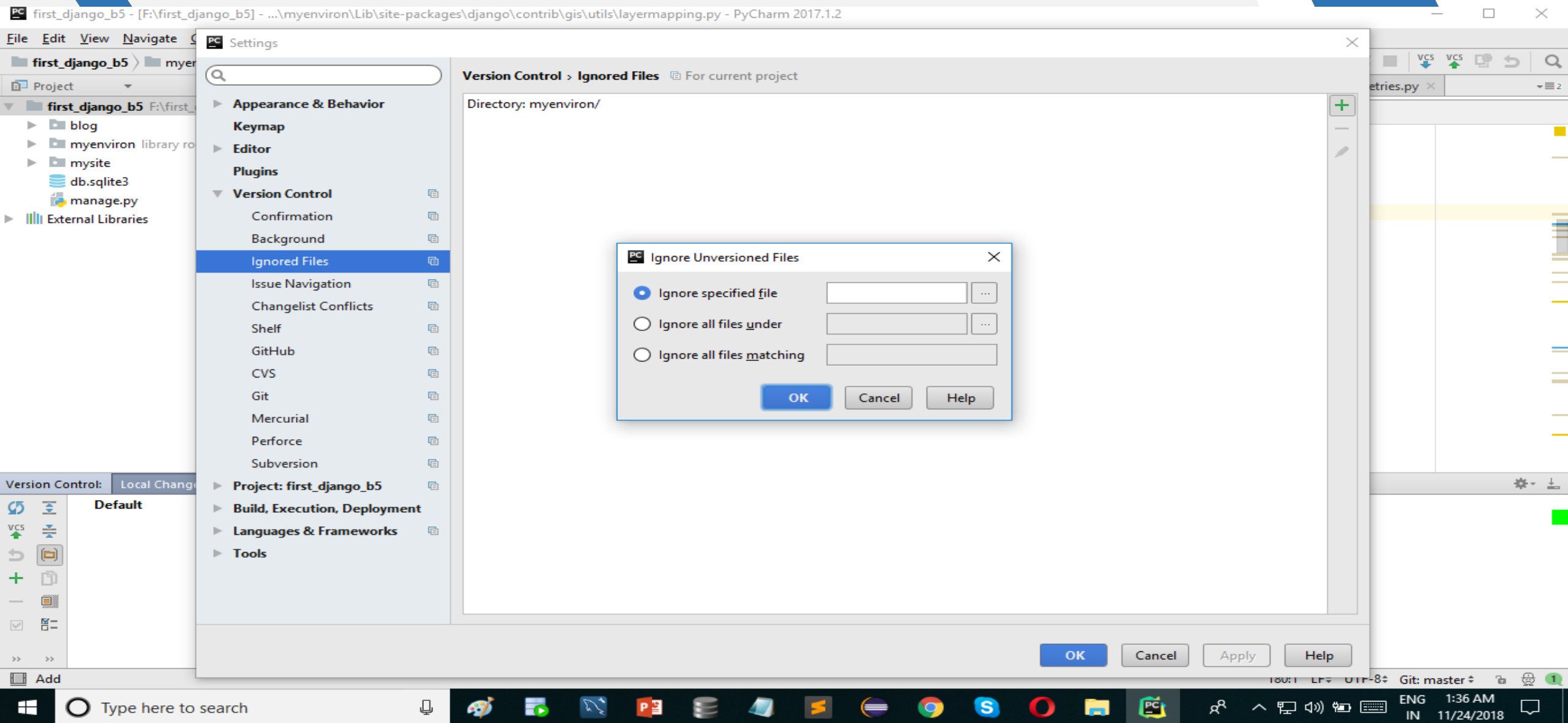
If you have enabled Git integration for your project, PyCharm suggests to add each newly created file under Git version control (you can change this behavior in the Settings dialog (Ctrl+Alt+S) under Version Control | Confirmation). If you want certain files to always remain unversioned, you can configure Git to ignore them.



# Exclude files from version control (ignore)

- **Configure a list of files to be ignored by Git**
- Either:
  - In the Settings/Preferences dialog (Ctrl+Alt+S) select Version Control | Ignored Files in the left pane.
  - Open the Version Control tool window (Alt+9) and switch to the Local Changes tab. Click eye icon on the toolbar and choose Configure Ignored Files.
- Click the Add add button on the toolbar.
- In the Ignore Unversioned Files dialog, specify the files/directories that you want to ignore, or define file name patterns:
  - Ignore specified file: specify the file name relative to the project root.
  - Ignore all files under: specify the directory whose contents should be ignored relative to the project root. The rule is applied recursively to all subdirectories.
  - Ignore all files matching: type the pattern that defines the names of files to be ignored. The rule is applied to all directories under the project root.
  - Two characters can be used as wildcards:
    - \*: to replace any string.
    - ?: to replace a single character.
- For example, \*.iml will ignore all files with the iml extension; \*.\*ml will ignore all files whose extension ends with ml.

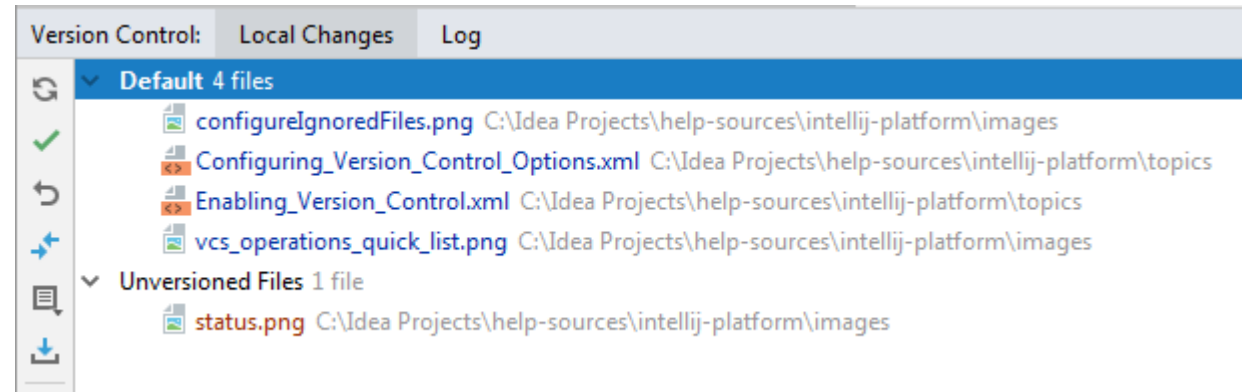
# Exclude files from version control (ignore)



# Check project status

PyCharm allows you to check the status of your local working copy compared to the repository version of the project. It lets you see which files have been modified, which new files have been added to the VCS, and which files are not being tracked by Git.

Open the Version Control tool window (Alt+9) and switch to the Local Changes tab:



The Default changelist shows all files that have been modified since you last synchronized with the remote repository (highlighted in blue), and all new files that have been added to the VCS but have not been committed yet (highlighted in green).

The Unversioned Files changelist shows all files that have been added to your project, but that are not being tracked by Git.

<https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html#put-existing-project-under-Git>

# Add a remote repository

To be able to collaborate on your Git project, you need to configure remote repositories that you fetch data from and push to when you need to share your work.

If you have cloned a remote Git repository, for example from GitHub, the remote is configured automatically and you do not have to specify it when you want to synchronize with it (i.e. when you perform a pull or a push operation).

The default name Git gives to the remote you've cloned from is **origin**.

However, if you created a Git repository based on local sources, you need to add a remote repository for other contributors to be able to push their changes to it, and for you to be able to share the results of your work.

<https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html#put-existing-project-under-Git>

# Adding remotes

Before this, create URL in git hub.

1. From the main menu, choose VCS | Git | Remotes. The Git Remotes dialog will open.
2. Click the Add add button on the toolbar or press Alt+Insert.
3. In the dialog that opens, specify the remote name and URL and click OK.

<https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html#put-existing-project-under-Git>


# Creating URL in git

Create a New Repository

been compromised directly. To increase your security, please [change your password](#) as soon as possible.  
Read our [documentation](#) on safer password practices. See our [blog](#) for more details.

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner:  maruthicse / Repository name:

Great repository names are short and memorable. Need inspiration? How about **improved-fortnight**.

Description (optional)

☒ **Public**  
Anyone can see this repository. You choose who can commit.

☐ **Private**  
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore:  | Add a license:  [i](#)

1

<https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html#put-existing-project-under-Git>

# Creating URL in git

The screenshot shows a web browser window displaying the GitHub repository page for 'maruthicse/djangotest123'. The browser's address bar shows the URL 'https://github.com/maruthicse/djangotest123'. The repository page includes a navigation bar with links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. A security warning banner is visible, stating that the password provided has been reported as compromised. Below the repository name, there are buttons for 'Unwatch', 'Star', and 'Fork'. The 'Code' tab is selected, showing the 'Quick setup' section with options for 'Set up in Desktop', 'HTTPS', and 'SSH'. The 'SSH' option is selected, and the URL 'https://github.com/maruthicse/djangotest123.git' is highlighted. A context menu is open over the URL, showing options like 'Copy', 'Go to', 'Print...', and 'Inspect'. Below the 'Quick setup' section, there is a section titled '...or create a new repository on the command line' with a terminal command block.

maruthicse/djangotest123

Search or jump to... Pull requests Issues Marketplace Explore

⚠ The password you provided has been reported as compromised due to re-use of that password on another service by you or someone else. GitHub has not been compromised directly. To increase your security, please [change your password](#) as soon as possible. Read our [documentation](#) on safer password practices. See our [blog](#) for more details.

maruthicse / djangotest123

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `https://github.com/maruthicse/djangotest123.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [CONTRIBUTING](#) file.

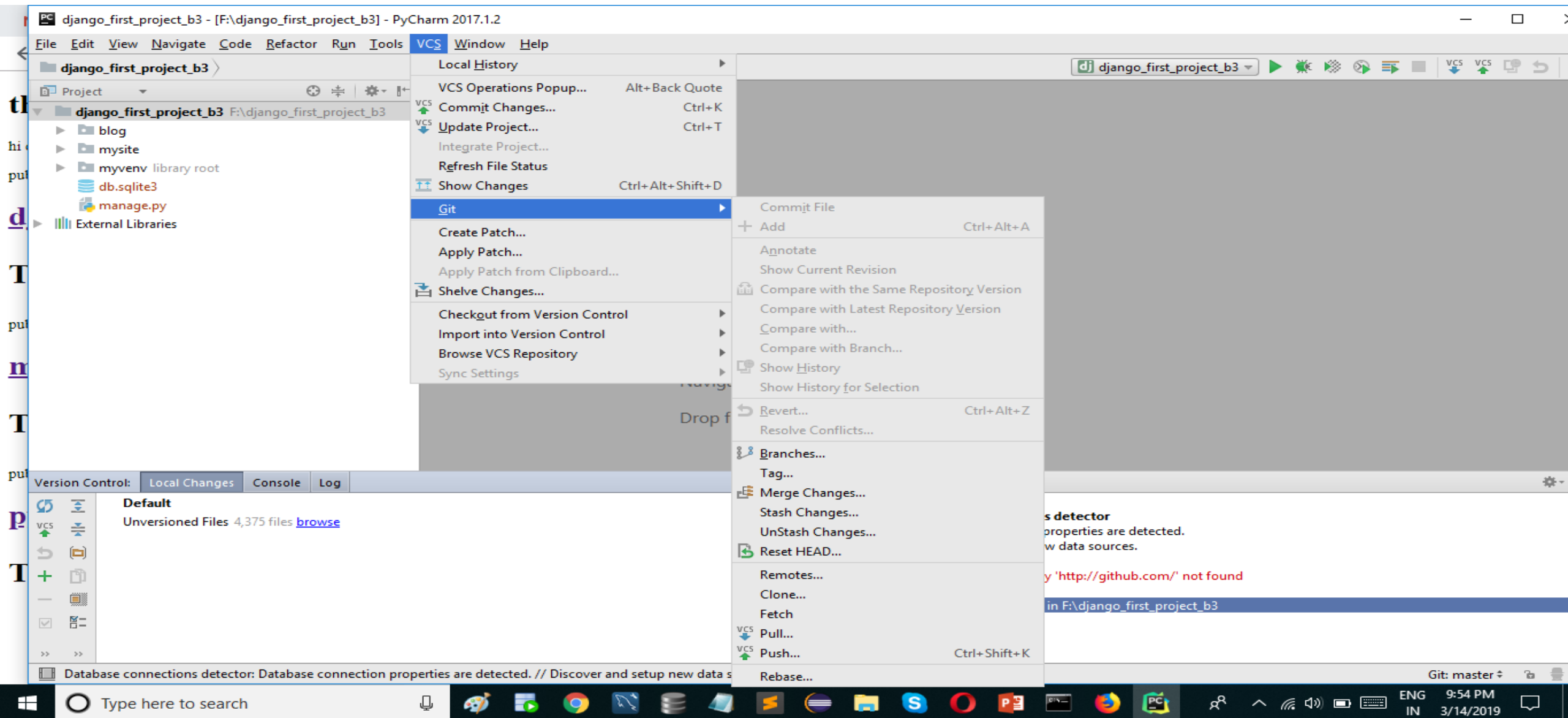
...or create a new repository on the command line

```
echo "# djangotest123" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/maruthicse/djangotest123.git
git push -u origin master
```

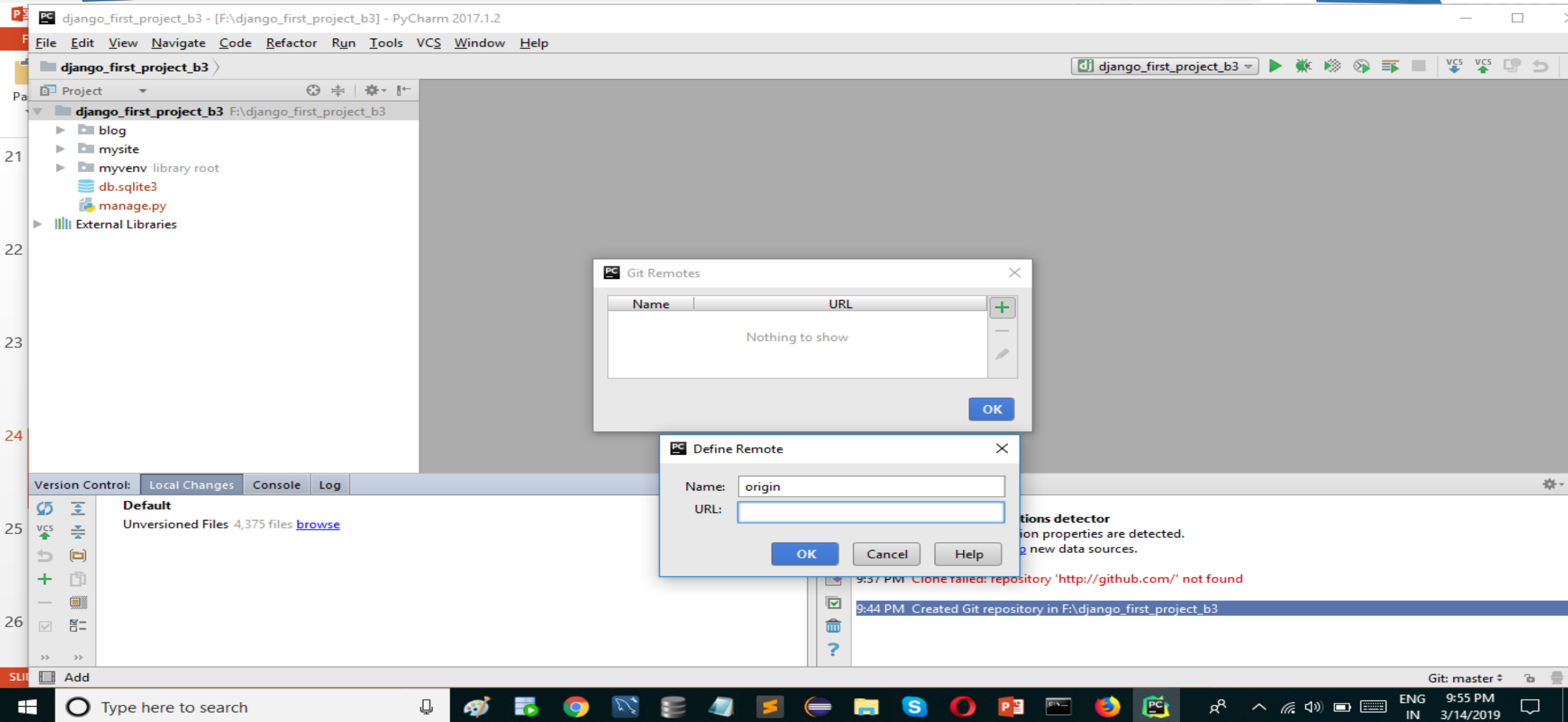
<https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html#put-existing-project-under-Git>



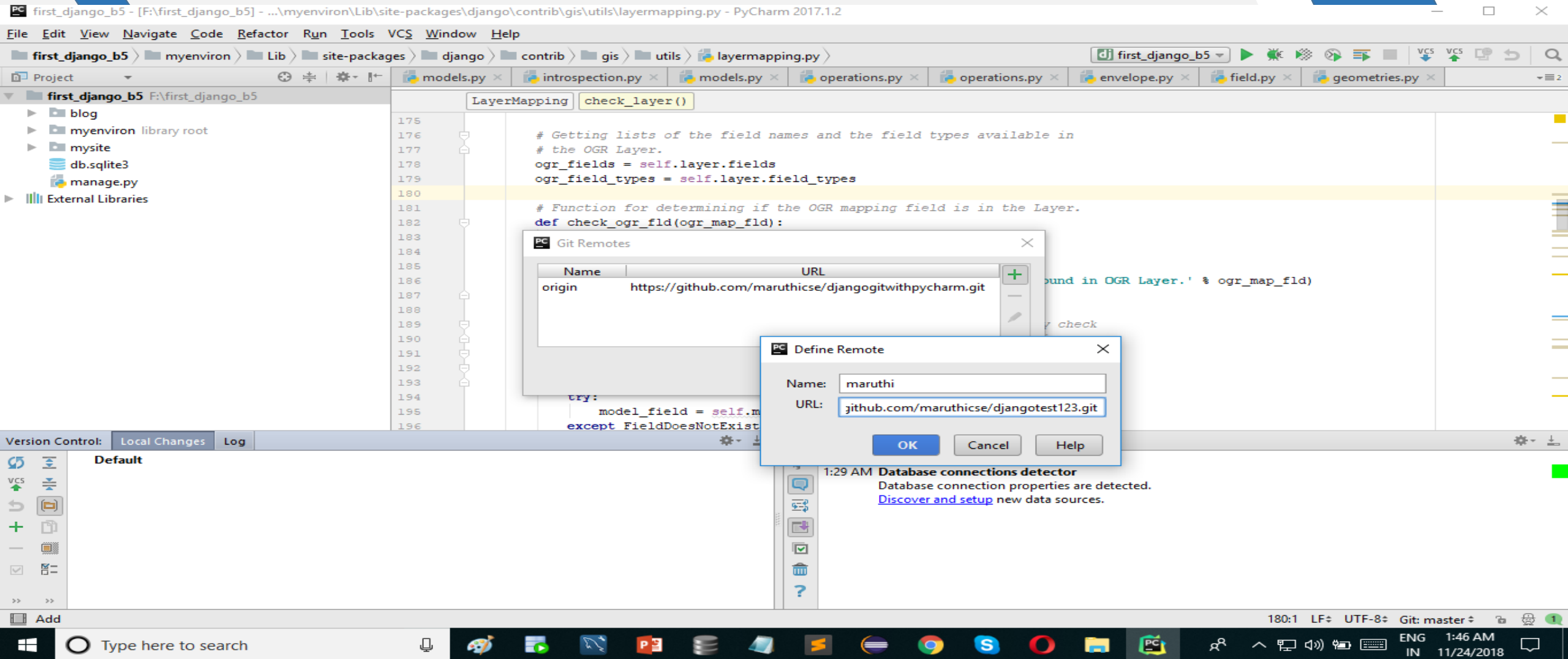
# Creating URL in git



# Creating URL in git

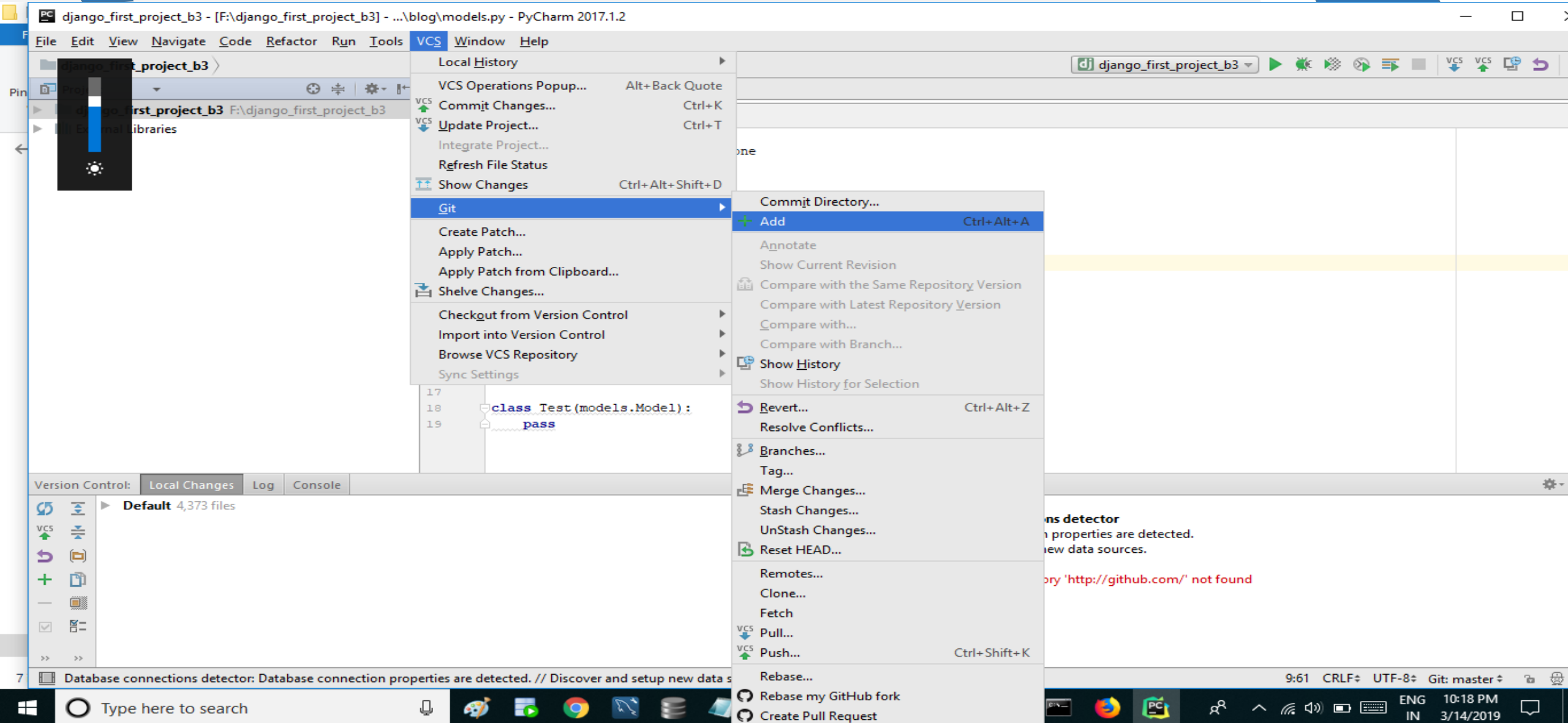


# Adding remotes in pycharm



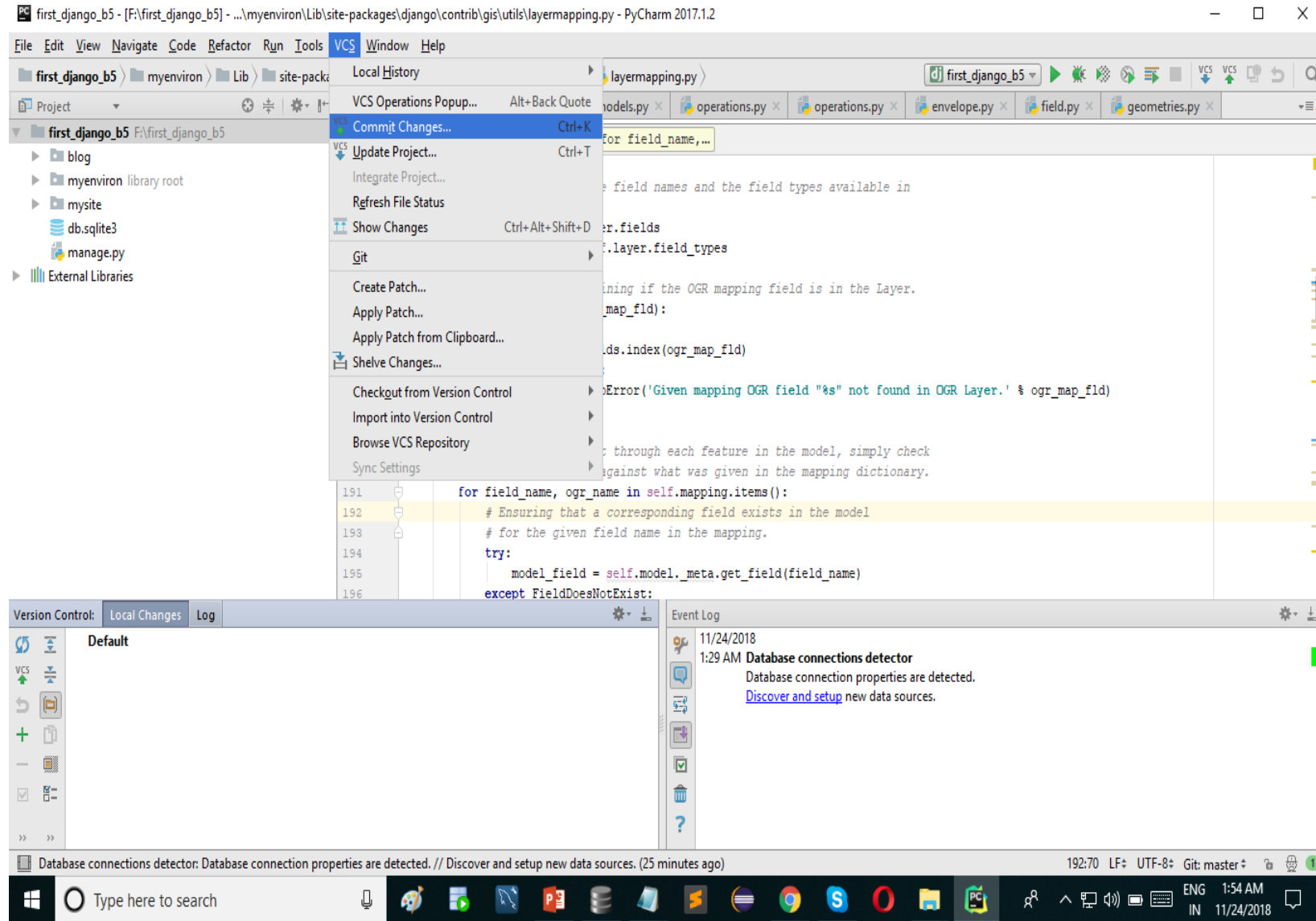
<https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html#put-existing-project-under-Git>

# Adding files in pycharm



# Commit and push changes

After you've added new files to the Git repository, or modified files that are already under Git version control and you are happy with their current state, you can share the results of your work. This involves committing them locally to record the snapshot of your repository to the project history, and then pushing them to the remote repository so that they become available to others.



# Commit and push changes

The screenshot shows an IDE window with the 'Commit Changes' dialog box open. The dialog box has a 'Change list' section with a dropdown set to 'Default'. Below this, a list of files is shown, including 'tests.py' from 'F:/first\_django\_b5/blog'. The 'Commit Message' field contains 'first one'. The 'Diff' section shows a comparison between the current version and the 'Your version'.

**Commit Changes Dialog:**

- Change list:** Default
- Files:** tests.py (F:/first\_django\_b5/blog)
- Commit Message:** first one
- Diff:** 1 difference
- Git Options:**
  - ☐ Amend commit
  - ☐ Sign-off commit
- Before Commit:**
  - ☐ Reformat code
  - ☐ Rearrange code
  - ☐ Optimize imports
  - ☐ Perform code analysis
  - ☒ Check TODO (Show All) [Configure](#)
  - ☐ Cleanup
- After Commit:**
  - Upload files to: (none)
- Buttons:** Commit, Cancel, Help

**Background IDE:**

- Project Explorer:** first\_django\_b5 (F:\first\_django\_b5) contains blog, migrations, templates, \_\_init\_\_.py, admin.py, admin.py.bak, apps.py, forms.py, forms.py.bak, models.py, models.py.bak, tests.py, urls.py, urls.py.bak, views.py, views.py.bak, myenv, library root, mysite, db.sqlite3.
- Version Control:** Local Changes, Log. Default 1 file: F:/first\_django\_b5/blog/tests.py.
- Code Editor:** from django.test import TestCase; # Create your tests here.; # testing the apps

**Taskbar:** Windows taskbar with search bar, task view, and various application icons. System tray shows date and time: 11/24/2018, 2:06 AM.

# Push changes to a remote repository

The screenshot shows the PyCharm 2017.1.2 IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The toolbar shows various icons for running, debugging, and version control. The left sidebar displays the project structure for 'first\_django\_b5', including folders like 'blog', 'migrations', 'templates', and files like 'admin.py', 'forms.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The main editor window shows the 'tests.py' file with the following code:

```
1 from django.test import TestCase
2
3 # Create your tests here.
4 # Testing the apps
5 # hello
```

The line '# hello' is highlighted in yellow. A notification bubble in the top right corner says 'Update available Click here to download ignore'. The bottom status bar shows '1 file committed: first one (moments ago)'. The Event Log on the right side of the bottom bar shows the following entries:

- 11/24/2018 2:02 AM Database connections detector Database connection properties are detected. Discover and setup new data sources.
- 2:12 AM 1 file committed: first one
- 2:13 AM 1 file committed: first one

A blue circle highlights the '1 file committed: first one' message in the Event Log. The Windows taskbar at the bottom shows the search bar and various application icons.



# Commit changes locally

- To invoke the Commit Changes dialog, select the files (or an entire changelist) in the Local Changes view and click icons actions commit svg on the toolbar or choose Commit Changes on the context menu of the selection; or press Ctrl+K. The Commit Changes dialog lists all files that have been modified since the last commit, and all newly added unversioned files.
- Enter a commit message and select the Before Commit actions you want IntelliJ IDEA to perform before committing the selected files to the local repository. You can click icons general messageHistory to choose from the list of recent commit messages.
- Select the following options in the Git section if necessary:
  - Author: if you are committing changes made by another person, you can specify the author of these changes.
  - Amend commit: select this option if you want to add the local changes to the latest commit (see Combine staged changes with the previous commit (amend commit) for details).
  - Sign-off commit: Select this option if you want to sign off your commit, i.e. to certify that the changes you are about to check in have been made by you, or that you take the responsibility for the code in question. When this option is enabled, the following line is automatically added at the end of the commit message: Signed off by: <username>

# Commit changes locally

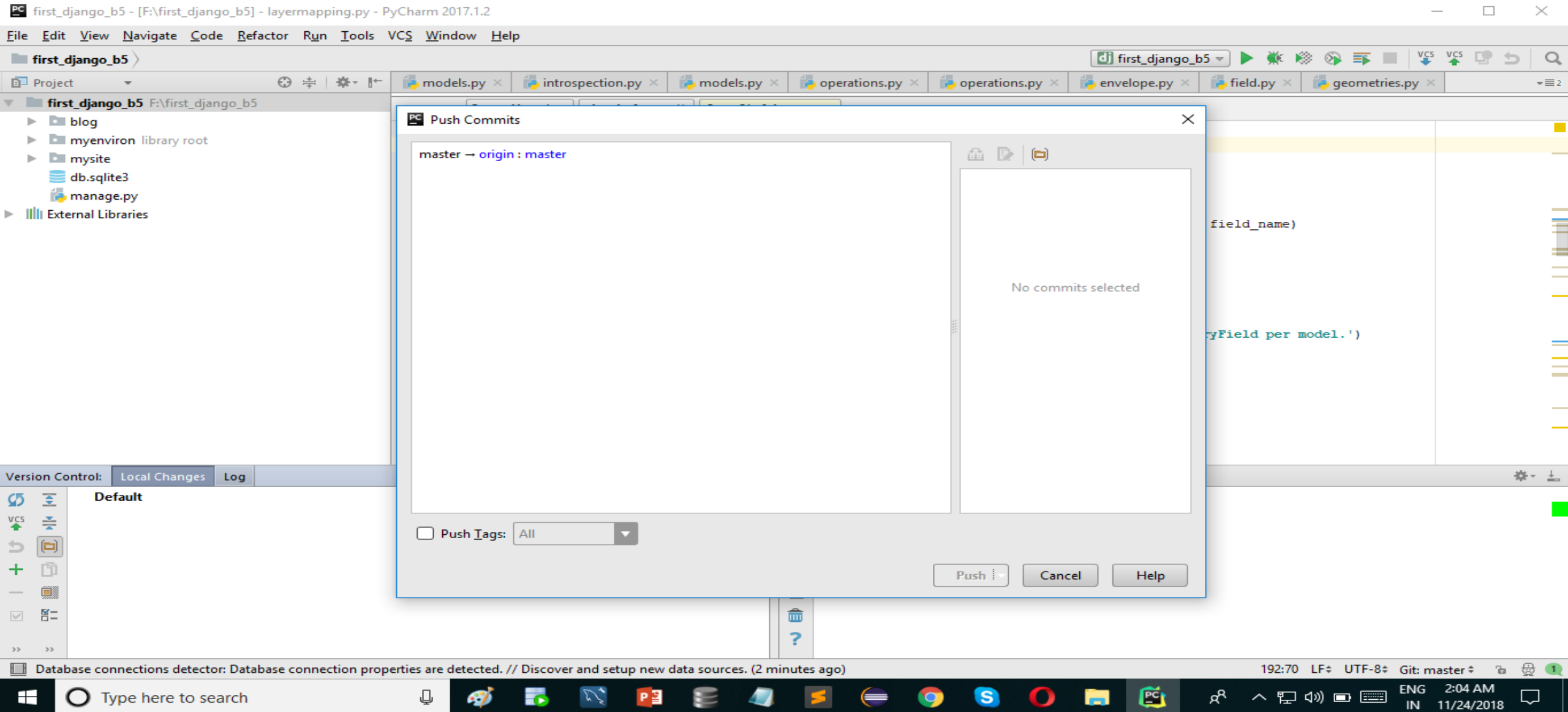
- Click the Commit button or hover the mouse over this button to display one of the following available commit options:
  - Commit and Push: push the changes to the remote repository immediately after the commit.
  - You can also press Ctrl+Alt+K to invoke the Commit and Push action from the Commit dialog.
  - Create Patch: generate a patch based on the changes you are about to commit. In the Create Patch dialog that opens, type the name of the patch file and specify whether you need a reverse patch.
  - Remote Run: run your personal build. This option is only available when you are logged in to TeamCity. Refer to TeamCity plugin documentation for details.

# Push changes to a remote repository

Make sure you have git login credentials.

- Press Ctrl+Shift+K or choose VCS | Git | Push from the main menu. The Push Commits dialog opens showing all Git repositories (for multi-repository projects) and listing all commits made in the current branch in each repository since the last push. If you have a project that uses multiple repositories that are not controlled synchronously, only the current repository is selected by default (for details on how to enable synchronous repositories control, refer to Version Control Settings: Git).
- If there are no remotes in the repository, the Define remote link appears. Click this link and specify the remote name and URL in the dialog that opens. It will be saved and you can edit it later via VCS | Git | Remotes (for details, see Add a remote repository).
- If you want to modify the target branch where you want to push, you can click the branch name. The label turns into a text field where you can type an existing branch name, or create a new branch. You can also click the Edit all targets link in the bottom-right corner to edit all branch names simultaneously. Note that you cannot change the local branch: the current branch for each selected repository will be pushed

# Push changes to a remote repository



# Push changes to a remote repository

If you want to preview changes before pushing them, select the required commit. The right-hand pane shows the changes included in the selected commit. You can use the toolbar buttons to examine the commit details.

Click the Push button when ready and select which operation you want to perform from the drop-down menu: Push or Force push.

These choice options are only available if the Allow force push option is enabled (see Version Control Settings: Git), otherwise, you can only perform the push operation.

# Push changes to a remote repository

The screenshot shows a web browser displaying the GitHub repository page for 'maruthicse/djangogitwithpycharm'. The browser's address bar shows the URL 'https://github.com/maruthicse/djangogitwithpycharm'. The repository page includes navigation tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Insights', and 'Settings'. Below these tabs, a message states 'No description, website, or topics provided.' with an 'Edit' button. A 'Manage topics' link is also present. A summary bar indicates '1 commit', '1 branch', '0 releases', and '1 contributor'. Below this, there are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The commit history shows a single commit by 'maruthicse' with the message 'first one', committed 'an hour ago'. The commit details show a list of files: '.idea', 'blog', 'myenviron', 'mysite', 'db.sqlite3', and 'manage.py', each with a commit message of 'first one' and a timestamp of 'an hour ago'. At the bottom of the repository page, there is a prompt to 'Add a README'.

maruthicse/djangogitwithpycharm

https://github.com/maruthicse/djangogitwithpycharm

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. Edit

Manage topics

1 commit 1 branch 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

maruthicse first one Latest commit 6ef575a an hour ago

.idea	first one	an hour ago
blog	first one	an hour ago
myenviron	first one	an hour ago
mysite	first one	an hour ago
db.sqlite3	first one	an hour ago
manage.py	first one	an hour ago

Help people interested in this repository understand your project by adding a README. Add a README

# Push changes to a remote repository

The screenshot displays the PyCharm 2017.1.2 IDE interface. The main editor window shows the `models.py` file for a Django project named `django_first_project_b3`. The code defines a `Post` model with fields `author`, `title`, `text`, `created_date`, and `published_date`. A `publish()` method is implemented, which sets `published_date` to the current time and saves the instance. A `__str__` method returns the `title` attribute. A `Test` model is also defined.

The bottom panel shows the Version Control tool window with the 'Local Changes' tab selected. It indicates that the changes are ready to be pushed. The Event Log on the right shows the following messages:

- 3/14/2019 10:09 PM Database connections detector: Database connection properties are detected. [Discover and setup](#) new data sources.
- 10:10 PM Clone failed: repository 'http://github.com/' not found
- 10:28 PM 4373 files committed: test

A green tooltip message states: "Push successful. Pushed master to new branch origin/master". The status bar at the bottom confirms: "Push successful: Pushed master to new branch origin/master (moments ago)".



# Push changes to a remote repository

## Check list

1. Make sure you have valid git credentials.
2. Create local repo before you send remote repo
3. Create git url for the project before you add remotes for the pycharm project if you are pushing a new project to git
4. Commit changes before you push the code[ make sure nothing to commit ]
5. Enter commit message while coming changes.
6. Exclude files from version control (ignore) like virtual environment folder.

for more information

**<https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html#put-existing-project-under-Git>**

# Pull source from a remote repository

1. \$ git clone <https://github.com/maruthicse/firstbootcampdjango1>
2. \$ cd firstbootcampdjango1/
3. You can create virtual environment
4. Install required software's
5. Activate it
6. Run the project



# Queries?