

Table of Contents

1 – Setup Development Environment :.....	2
2 – TeamCity installation :.....	3
3 – Key Management :	4
4 – Using GitExtensions :.....	13
Appendix :	15

1 – Setup Development Environment :

Follow the steps shown below to setup the development environment :

1. <https://store.docker.com/editions/enterprise/docker-ee-server-windows> – Install Docker Windows version.
2. <https://www.eclipse.org/ide/> – Install Java and Eclipse.
3. <https://www.visualstudio.com/> – Install Visual Studio DevOps.
4. <https://jenkins.io/> – Install Jenkins. The CloudBees version is recommended.
5. <https://gitlab.com> – Create a user account on GitLab or GitHub. GitLab provides private repositories so we will select that for this reason.
6. <https://git-scm.com/download/win> – Install Git for windows.
7. <https://desktop.github.com/> – Install GitHub Desktop for windows.
8. <https://sourceforge.net/projects/kdiff3/files/latest/download> – Install kdiff3 code compare tool.
9. <https://docs.gitlab.com/runner/> – Install to run GitLab jobs.
10. <https://git-scm.com/download/gui/windows> – Install Git Gui for easy administration of Git repositories.
11. <https://sourceforge.net/projects/gitextensions/> – Install GitExtensions.
12. <https://www.jetbrains.com/teamcity/download/> – Install TeamCity.
13. <https://marketplace.visualstudio.com/items?itemName=MysticBoy.GitLabExtensionforVisualStudio> – Install GitLab extension for Visual Studio so we can manage GitLab repositories from inside Visual Studio.
14. <https://docs.sonarqube.org/latest/setup/install-server/> – Install Sonarqube.
15. <https://docs.gitlab.com/ee/gitlab-basics/create-your-ssh-keys.html> – Create a public private key pair with RSA based encryption and save the public key in the folder location for current user. Read more about this in the key management section.

Note: Cloud based versions of Eclipse are available. To install TeamCity we will need to setup JDBC drivers for SQL Server access.

2 – TeamCity installation :

TeamCity can be installed to run on IIS with MS Sql Server or Oracle. It runs on JRE (Java Runtime Environment).

Before starting the install, check to see if the database version is supported:

<https://confluence.jetbrains.com/display/TCD9/Supported+Platforms+and+Environments#SupportedPlatformsandEnvironments-SupportedDatabases>

The installer will configure build agents as step 1 of the install.

Property	Value
serverUrl	http://localhost:88
name	Maserati
ownPort	9090
systemDir	C:\TeamCity\buildAgent\system
workDir	C:\TeamCity\buildAgent\work
tempDir	C:\TeamCity\buildAgent\temp
env.TEAMCITY_JRE	C:\TeamCity\jre

env.TEAMCITY_JRE	value should be the JDK home directory (in case you want to run Java builds).
name	value is the name of an agent that will be displayed in the TeamCity user interface.
ownPort	value is a port where the agent listens to the server commands. Please make sure this port is not blocked by firewall.
serverUrl	value is the TeamCity server location.
workDir	value is the working catalogue where the builds are being built.
tempDir	value is the temp catalogue being used by agent.

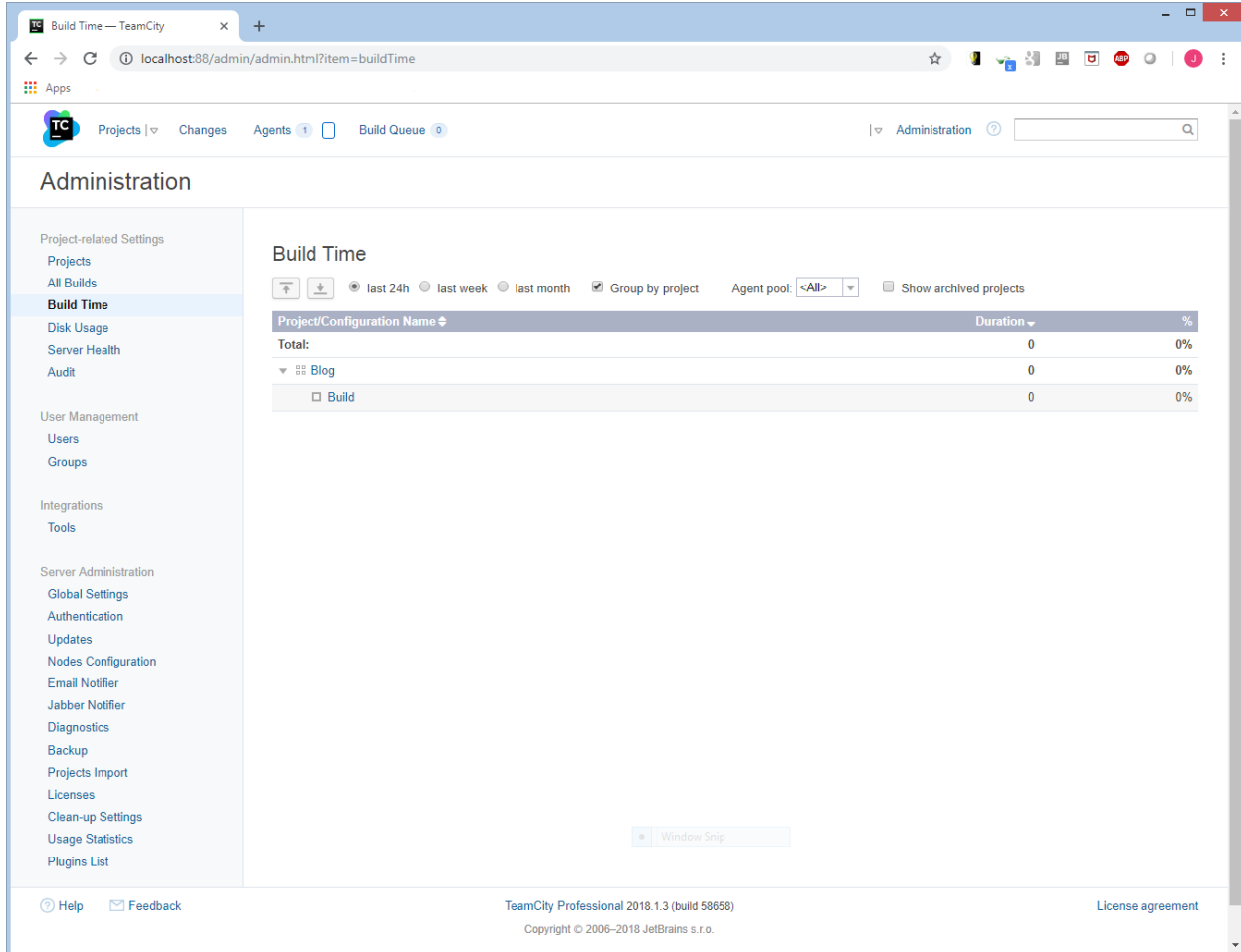
* The Build Agent properties file is stored in the 'conf' directory of the Build Agent installation folder where you can edit it later.

Create a TeamCity database using SQL Management Studio and create a Login to access this database called TeamCity. SQL Server Browser should be running and TCP/IP access to MS Sql Server should be enabled also. For more details reference the setup url below:

<https://confluence.jetbrains.com/display/TCD18/Setting+up+TeamCity+with+MS+SQL+Server>

The browser window should appear and if all goes well it will allow us to install the JDBC drivers and then the installer will initialize the TeamCity database.

Once the database has been setup then we can create a new application project in TeamCity. In our case we will call the project Blog.



The screenshot shows the TeamCity Administration interface. The main content area is titled "Build Time" and displays a table with the following data:

Project/Configuration Name	Duration	%
Total:	0	0%
▼ Blog	0	0%
□ Build	0	0%

The interface also includes a left sidebar with navigation options like "Project-related Settings", "User Management", and "Integrations". The footer shows "TeamCity Professional 2018.1.3 (build 58658)" and "Copyright © 2006–2018 JetBrains s.r.o.".

3 – Key Management :

In part 1 when we installed GitHub Desktop we will get Git Cmd and Git Bash tools with the icon shown below:



To generate a key. Run Git cmd then follow the steps shown below:

```
cd C:\Program Files\Git\usr\bin
ssh-keygen -o
```

Enter Key name as **Test** and Passphrase as **code**. After generating the key save it to the users folder located here C:\Users\JDoe\.ssh so it can be used by GitHub and GitLab etc.

Open a windows command prompt and type this to check the key is valid.

```
type c:\Users\JDoe\.ssh\github_rsa.pub
```

Copy the key value to the windows clip board

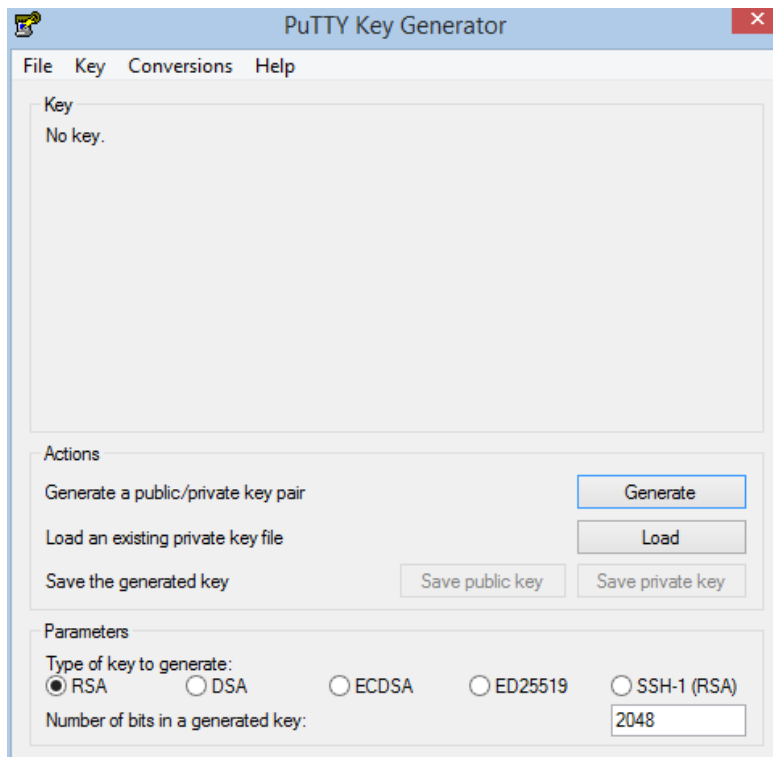
```
type c:\Users\JDoe\.ssh\github_rsa.pub | clip
```

After that we need to paste the Text into GitHub.com and GitLab.com user profiles.

Another way for generating keys using the Putty Tools from Git Extensions Menu.

<https://git-scm.com/book/en/v2/Git-on-the-Server-Generating-Your-SSH-Public-Key>

<https://gitlab.com/help/ssh/README#locating-an-existing-ssh-key-pair>



Also we need to start the Pageant Authentication agent from inside GitExtensions and Add the key we created in it so we can push our code to the remote origin.

When we create the project, it will setup the Version Control Roots Path for the application and TeamCity will monitor this path to see if there are any commits and if it sees a commit then it will notify the build agent.

The screenshot shows the TeamCity Administration interface. The breadcrumb navigation is 'Administration / <Root project> / Blog'. The left sidebar is expanded to 'Build' and then 'Version Control Settings'. The main content area is titled 'VCS Roots' and contains a table with one entry for a Git repository. Below the table is a link to 'Show advanced options'.

Name	Checkout Rules
(git) https://gitlab.com/improvecode/Blog.git#refs/heads/master belongs to Blog Commit hook is inactive Latest check for changes: 15:28 (periodical run by the schedule) Changes checking interval: 1m	Edit Detach Edit checkout rules (0)

On the above screen, click the Build Steps left menu item and select a Build Runner:

The screenshot shows the 'Build Steps' configuration dropdown menu. The menu is open, showing a list of build runner types. The first two items are '-- Choose build runner type --'. The list includes various build runners such as .NET CLI (dotnet), .NET Process Runner, Ant, Command Line, Container Deployer, Docker, Docker Compose, Duplicates finder (Java), Duplicates finder (ReSharper), FTP Upload, FxCop, Gradle, Inspections (IntelliJ IDEA), Inspections (ReSharper), IntelliJ IDEA Project, Maven, MSBuild, MSpec, NAnt, NuGet Installer, NuGet Pack, NuGet Publish, NUnit, PowerShell, Rake, Simple Build Tool (Scala), SMB Upload, SSH Exec, SSH Upload, Visual Studio (sln), and Visual Studio 2003.

Note: TeamCity can run both Java and .NET builds.

Triggers Screen:

Administration / <Root project> / Blog

Run ... Actions Build Configuration Home

Build

- General Settings
- Version Control Settings 1
- Build Steps
- Triggers 1**
- Failure Conditions
- Build Features
- Dependencies
- Parameters
- Agent Requirements
- Suggestions 1
- « Hide unconfigured

Triggers

Triggers are used to add builds to the queue either when an event occurs (like a VCS check-in) or periodically with some configurable interval.

+ Add new trigger

Trigger	Parameters Description
VCS Trigger	Branch filter: +:*

Created 16 hours ago by Jason Azim (view history)

Common Failure Conditions Screen:

The screenshot shows the Jenkins interface for configuring failure conditions. The top navigation bar includes the Jenkins logo, 'Projects | ▾', 'Changes', 'Agents 1', 'Build Queue 0', 'Jason Azim | ▾', 'Administration ?', and a search bar. The breadcrumb trail is 'Administration / <Root project> / Blog'. The main header area contains 'Run ...', 'Actions ▾', and 'Build Configuration Home'. The left sidebar lists settings categories: General Settings, Version Control Settings 1, Build Steps, Triggers 1, Failure Conditions (highlighted), Build Features, Dependencies, Parameters, Agent Requirements, Suggestions 1, and « Hide unconfigured». The main content area is titled 'Common Failure Conditions' with a subtitle 'In this section you can specify when your build should fail. ⓘ'. Under 'Fail build if:', there is a text input 'it runs longer than specified limit in minutes (0 — unlimited) 0'. Below this are four checkboxes: 'one of build steps exited with an error (e.g non-zero exit code)' (checked), 'at least one test failed' (checked), 'an error message is logged by build runner' (unchecked), and 'an out-of-memory or crash is detected (Java only)' (checked). At the bottom of this section are 'Save' and 'Cancel' buttons. The 'Additional Failure Conditions' section has a subtitle 'In this section you can configure build failure depending on various metrics. ⓘ' and a '+ Add failure condition' button. At the bottom left of the sidebar, it says 'Created 16 hours ago by Jason Azim (view history)'.

Build Features Screen:

The screenshot shows the Jenkins interface for configuring build features. The top navigation bar is identical to the previous screen. The breadcrumb trail is 'Administration / <Root project> / Blog'. The main header area contains 'Run ...', 'Actions ▾', and 'Build Configuration Home'. The left sidebar lists settings categories: General Settings, Version Control Settings 1, Build Steps, Triggers 1, Failure Conditions, Build Features (highlighted), Dependencies, Parameters, Agent Requirements, Suggestions 1, and « Hide unconfigured». The main content area is titled 'Build Features' with a subtitle 'In this section you can configure build features. A build feature is a piece of functionality that can affect a build process or reporting its results. ⓘ'. Below this is a '+ Add build feature' button. At the bottom left of the sidebar, it says 'Created 16 hours ago by Jason Azim (view history)'.

Build Dependencies Screen: Useful for setting dependencies on other applications.

TC Projects | Changes Agents 1 Build Queue 0 Jason Azim | Administration Administration / <Root project> / Blog Run Actions Build Configuration Home

Build

- General Settings
- Version Control Settings 1
- Build Steps
- Triggers 1
- Failure Conditions
- Build Features
- Dependencies**
- Parameters
- Agent Requirements
- Suggestions 1
- « Hide unconfigured

Created 16 hours ago by Jason Azim (view history)

Snapshot Dependencies

Build configurations linked by a snapshot dependency will use the same snapshot of the sources. The build of this configuration will run after all the dependencies are built. If necessary, the dependencies will be triggered automatically.

+ Add new snapshot dependency

Artifact Dependencies

Artifact dependency allows using artifacts produced by another build.

+ Add new artifact dependency

Build Parameters Screen: Define custom configuration settings.

TC Projects | Changes Agents 1 Build Queue 0 Jason Azim | Administration Administration / <Root project> / Blog Run Actions Build Configuration Home

Build

- General Settings
- Version Control Settings 1
- Build Steps
- Triggers 1
- Failure Conditions
- Build Features
- Dependencies
- Parameters**
- Agent Requirements
- Suggestions 1
- « Hide unconfigured

Created 16 hours ago by Jason Azim (view history)

+ Add new parameter

There are 0 own parameters defined

Configuration Parameters

Configuration parameters are not passed into build, can be used in references only.

None defined

System Properties (system.)

System properties will be passed into the build (without system. prefix), they are only supported by the build runners that understand the property notion.

None defined

Environment Variables (env.)

Environment variables will be added to the environment of the processes launched by the build runner (without env. prefix).

None defined

Agent Requirements Screen: The server is itself running the builds.

Administration / <>> <Root project> / <>> Blog

Run ... Actions Build Configuration Home

Build

- General Settings
- Version Control Settings 1
- Build Steps
- Triggers 1
- Failure Conditions
- Build Features
- Dependencies
- Parameters
- Agent Requirements**
- Suggestions 1
- « Hide unconfigured

Created 16 hours ago by Jason Azim (view history)

Explicit Requirements

This page lists all requirements that build agents should meet to run your builds.

+ Add new requirement

Agents Compatibility

In this section you can see which agents are compatible with the requirements and which are not.

Compatible agents (1)	Incompatible agents (0)
<ul style="list-style-type: none">▼ Default pool (1)<ul style="list-style-type: none">Maserati	None

When an Agent successfully completes a build it is reported and tracked.

Blog / Build

Run ... Actions Edit Configuration Settings |

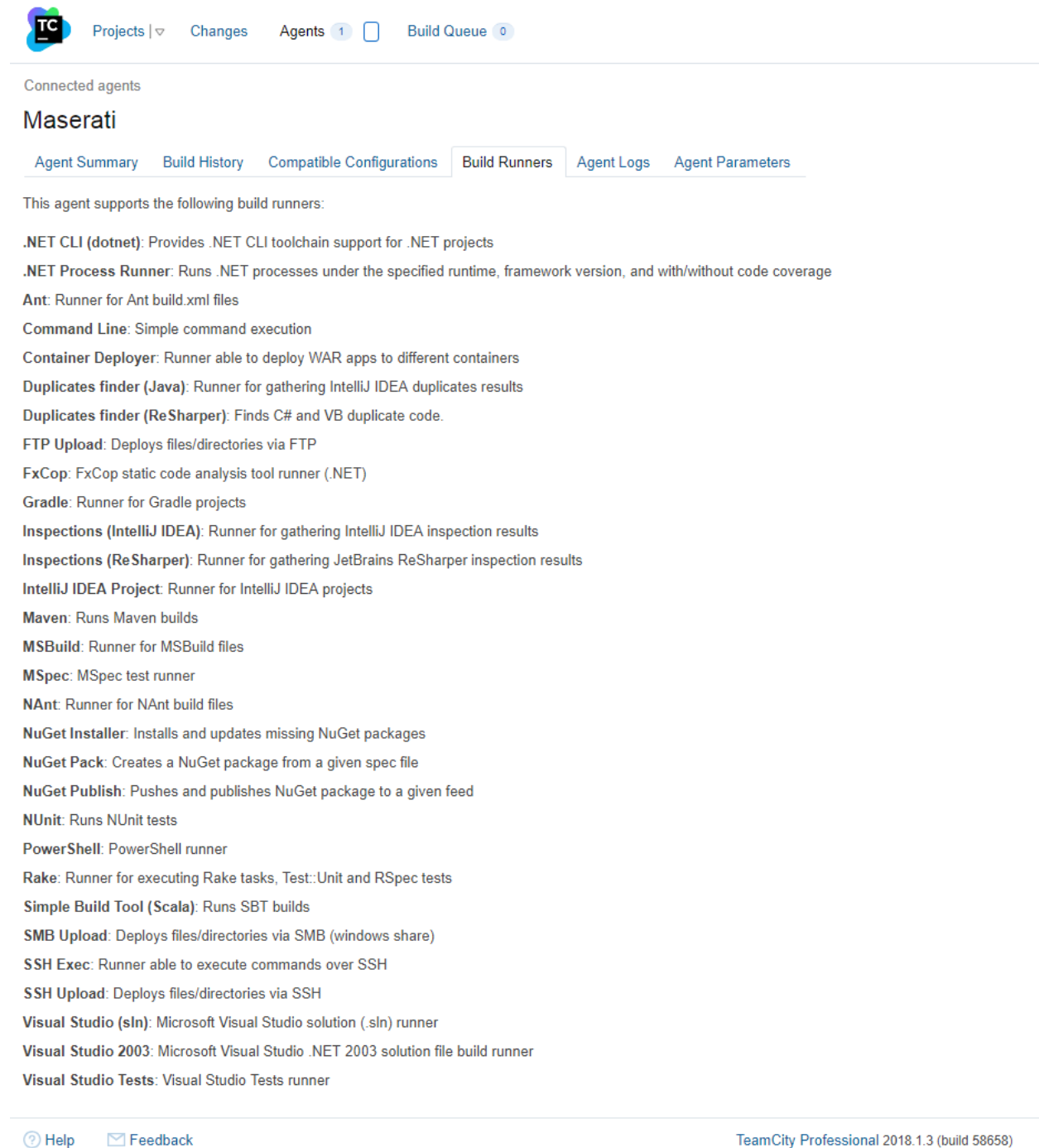
✓ #1 (29 Oct 18 16:47)

Overview Changes 1 Build Log Parameters Artifacts

First recorded build | All history | Last recorded build

Result:	✓ Success	Agent:	Maserati
Time:	29 Oct 18 16:47:22 - 16:47:37 (14s)	Triggered...	Git on 29 Oct 18 16:47

Agents have to be configured with runner build steps. The different types are shown below:



The screenshot shows the TeamCity web interface. At the top, there is a navigation bar with 'Projects', 'Changes', 'Agents' (with a count of 1), and 'Build Queue' (with a count of 0). Below this, the 'Connected agents' section is visible, with 'Maserati' selected. The 'Build Runners' tab is active, displaying a list of supported build runners for this agent.

Connected agents

Maserati

Agent Summary | Build History | Compatible Configurations | **Build Runners** | Agent Logs | Agent Parameters

This agent supports the following build runners:

- .NET CLI (dotnet):** Provides .NET CLI toolchain support for .NET projects
- .NET Process Runner:** Runs .NET processes under the specified runtime, framework version, and with/without code coverage
- Ant:** Runner for Ant build.xml files
- Command Line:** Simple command execution
- Container Deployer:** Runner able to deploy WAR apps to different containers
- Duplicates finder (Java):** Runner for gathering IntelliJ IDEA duplicates results
- Duplicates finder (ReSharper):** Finds C# and VB duplicate code.
- FTP Upload:** Deploys files/directories via FTP
- FxCop:** FxCop static code analysis tool runner (.NET)
- Gradle:** Runner for Gradle projects
- Inspections (IntelliJ IDEA):** Runner for gathering IntelliJ IDEA inspection results
- Inspections (ReSharper):** Runner for gathering JetBrains ReSharper inspection results
- IntelliJ IDEA Project:** Runner for IntelliJ IDEA projects
- Maven:** Runs Maven builds
- MSBuild:** Runner for MSBuild files
- MSpec:** MSpec test runner
- NAnt:** Runner for NAnt build files
- NuGet Installer:** Installs and updates missing NuGet packages
- NuGet Pack:** Creates a NuGet package from a given spec file
- NuGet Publish:** Pushes and publishes NuGet package to a given feed
- NUnit:** Runs NUnit tests
- PowerShell:** PowerShell runner
- Rake:** Runner for executing Rake tasks, Test::Unit and RSpec tests
- Simple Build Tool (Scala):** Runs SBT builds
- SMB Upload:** Deploys files/directories via SMB (windows share)
- SSH Exec:** Runner able to execute commands over SSH
- SSH Upload:** Deploys files/directories via SSH
- Visual Studio (sln):** Microsoft Visual Studio solution (.sln) runner
- Visual Studio 2003:** Microsoft Visual Studio .NET 2003 solution file build runner
- Visual Studio Tests:** Visual Studio Tests runner

[Help](#) | [Feedback](#) | TeamCity Professional 2018.1.3 (build 58658)

Click on Build Agent Link to configure the settings for the Agent:

The screenshot shows the Jenkins web interface. At the top, there is a navigation bar with a logo on the left and links for 'Projects', 'Changes', 'Agents' (with a count of 1), and 'Build Queue' (with a count of 0). Below this, the page title is 'Connected agents' followed by the agent name 'Maserati'. A horizontal menu contains tabs for 'Agent Summary', 'Build History', 'Compatible Configurations', 'Build Runners', 'Agent Logs', and 'Agent Parameters'. The 'Agent Summary' tab is active. Under the 'Status' section, it indicates the agent is 'Connected' since 29 Oct 18 08:11, with a last communication date of 29 Oct 18 16:42. It is 'Authorized' with a comment: 'Locally installed agent is authorized by default' and has a 'Disable agent' button. The 'Enabled' status is shown with a 'Disable agent' button. The 'Details' section lists: Agent name: Maserati, Hostname: Maserati, IP: 192.168.56.1, Port: 9090, Communication protocol: unidirectional, Operating system: Windows 8.1, version 6.3, CPU rank: 540, Pool: Default, and Version: 58658. The 'Miscellaneous' section includes links for 'Clean sources on this agent', 'Open Remote Desktop', 'Reboot agent machine', and 'Dump threads on agent'.

TC Projects | Changes Agents 1 Build Queue 0

Connected agents

Maserati

Agent Summary Build History Compatible Configurations Build Runners Agent Logs Agent Parameters

Status

Connected since 29 Oct 18 08:11, last communication date 29 Oct 18 16:42

Authorized with comment: Locally installed agent is authorized by default [Unauthorize agent](#)

Enabled [Disable agent](#)

Details

Agent name: **Maserati**

Hostname: **Maserati**

IP: **192.168.56.1**

Port: **9090**

Communication protocol: **unidirectional**

Operating system: **Windows 8.1, version 6.3**

CPU rank: **540**

Pool: **Default**

Version: **58658**

Miscellaneous

[Clean sources on this agent](#)

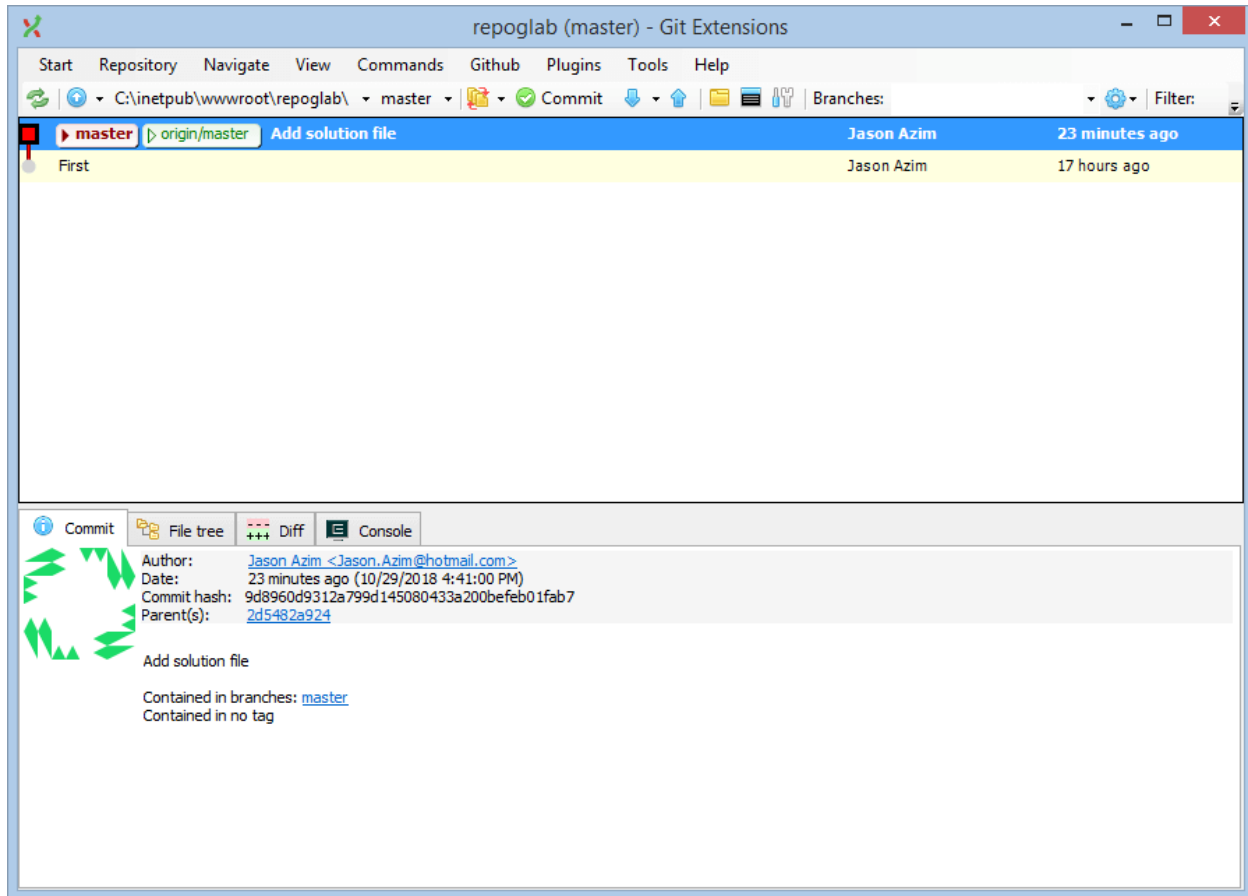
[Open Remote Desktop](#)

[Reboot agent machine](#)

[Dump threads on agent](#)

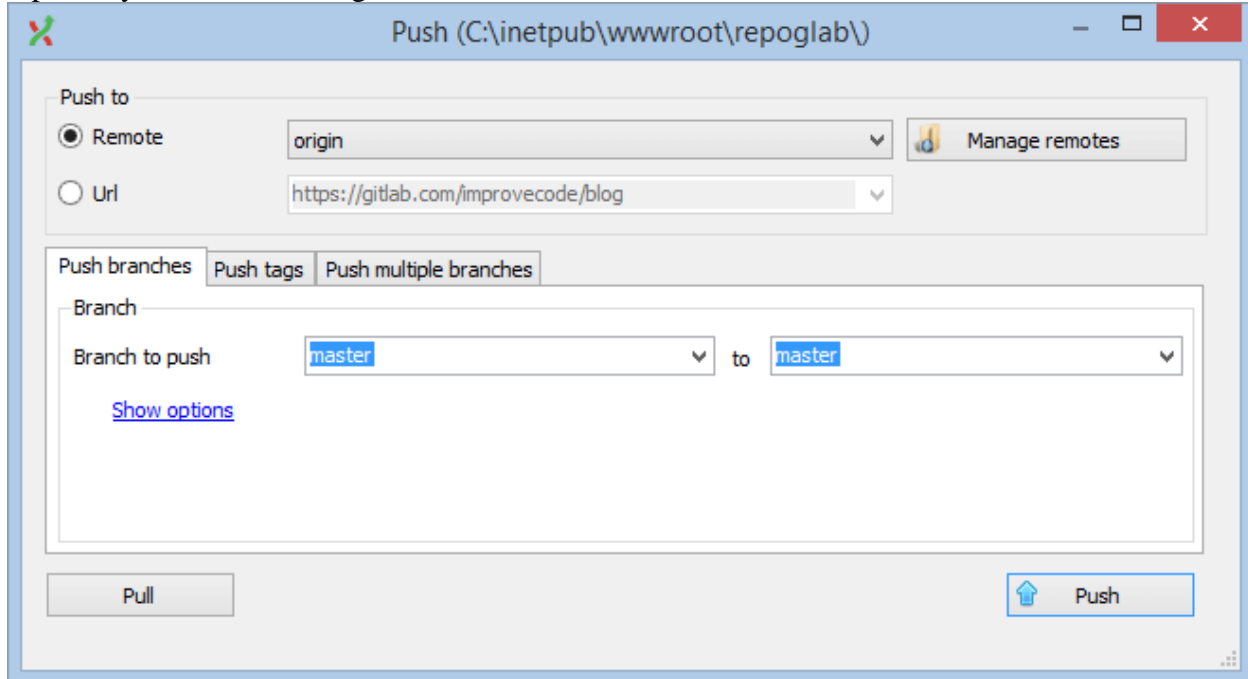
4 – Using GitExtensions :

Open the git repository in GitExtensions. The screen shows the source control graph / timeline.



In the above screen click on the Tools menu to access importing / exporting keys and for running Pageant Authorization Agent.

Click on the Push Up Green Arrow button to push local code changes to remote git http repository. The Push DialogBox is shown:



Confirm the operation by clicking on the Push Blue Button. The code is pushed successfully DialogBox is shown:



Appendix :

Plugins for Team City

<https://plugins.jetbrains.com/search?correctionAllowed=true&pr=teamcity&orderBy=&search=ft%5B>