

Working with Git

A distributed versioning control system.

[Case 0: Setting up a project with Git](#)

[Case 0.5: Cloning a repository](#)

[Case 1: Sharing code via remote repository](#)

[Case 2: Handling merge conflicts](#)

[Case 3: Using Branches](#)

[Case 4: Going back to a previous commit](#)

[Case 5: Accidentally committed files to your repository](#)

Version Control - Why Use It?

Case 0: Setting up a project with Git

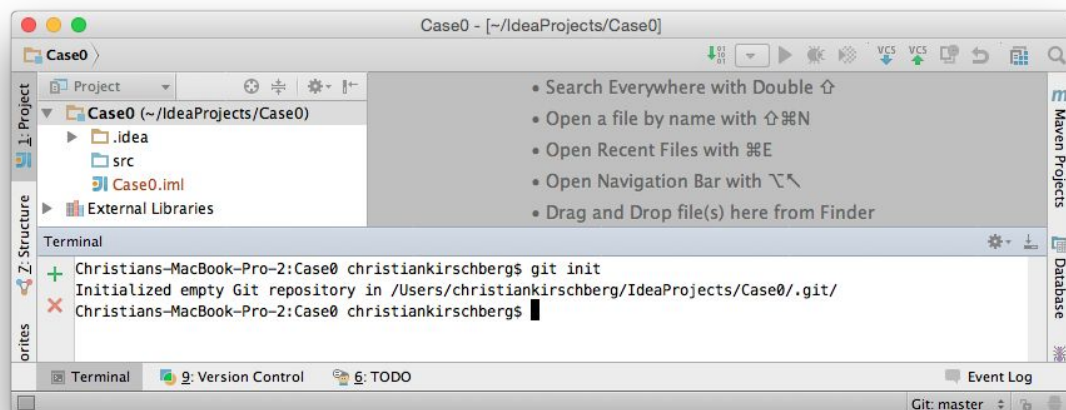
Create a folder on your machine where you would like your new project to exist. If you already have a project, navigate to it.

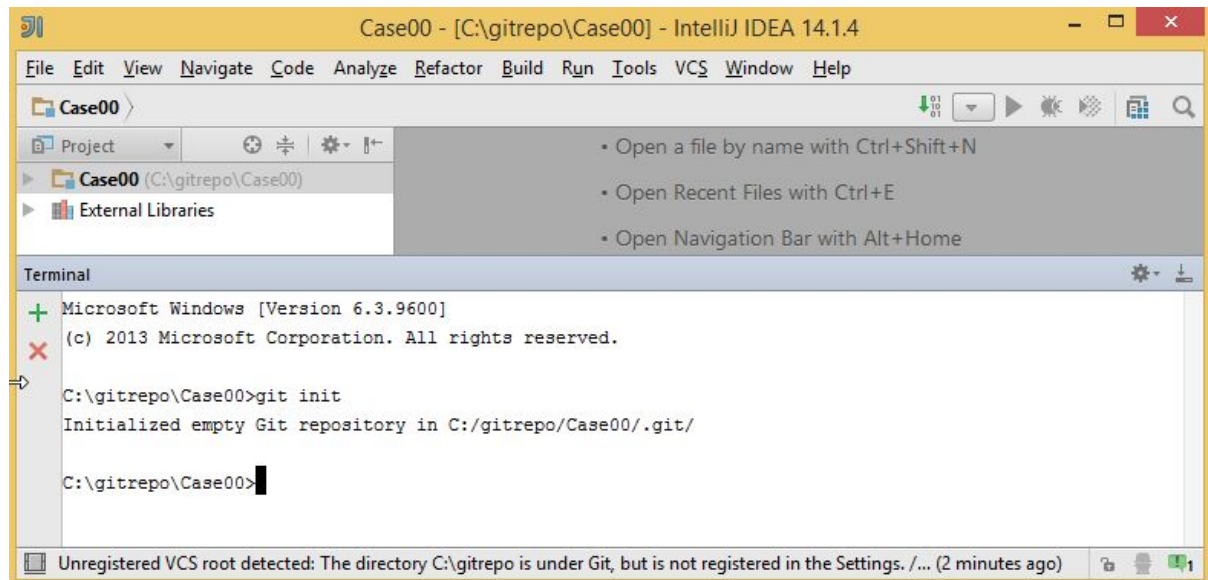
Create the new project.

Write in the terminal, at the root folder of this project:

```
git init
```

If you are using IntelliJ it looks like the following after initializing git.
(If using Mac)





(If using Windows)

Create a .gitignore file to avoid adding irrelevant files to your repository.

We only want to include source code files, so anything else should be excluded in the .gitignore file. You could also include other resources like readme.txt (text files), feature change list, language files or other relevant files. Do not include files which can be generated/compiled from other files, eg. java .class files or similar.

To create the .gitignore file on a Mac

In the terminal write:

```
touch .gitignore
```

```
open .gitignore
```

To create the .gitignore file on Windows

In the command prompt write:

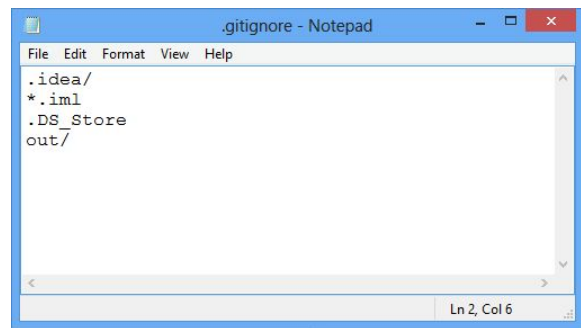
```
notepad .gitignore
```

Insert the files and directories that you want to exclude. Each line in the file is one exclusion eg. a directory with all its files or a file. You can also use * as a wildcard.

In this example our .gitignore file looks like the following:



Mac



Windows

```
.idea  
*.iml  
*.iws  
.DS_Store  
out/
```

Next, we set up a remote repository. In this case we use bitbucket.org but you could also use [github](https://github.com) or other services.

Register as a user at bitbucket.org. All team members must create an account.

Create a repository with Git as a versioning control system.

At the time of writing, creating a repository, [bitbucket](https://bitbucket.org) looks like the following:

Create a new repository

You can also [import a repository](#)

Owner

ChristianKirschberg

Name *

GitTest

Description

Access level

☒ This is a private repository

Forking

Allow only private forks

Repository type

☒ Git
☐ Mercurial

Project management


☐ Issue tracking
☐ Wiki

Language

Java


New to Bitbucket?

Learn the basics of using Git and Mercurial by exploring the Bitbucket 101.



Working in a team?

Create a team account to consolidate your repos and organize your team's work



Repository integrations

HipChat ☐ Enable HipChat notifications

Create repository

Cancel

After creating the remote repository we need to set up our local repository and point it to the remote repository.
To do that we write:

```
git remote add origin  
https://ChristianKirschberg@bitbucket.org/ChristianKirschberg/git  
test.git
```

Replace the URL with your relevant URL. The URL can be found in your bitbucket account at the following location.

Bitbucket

Dashboard

Teams

Repositories

Snippets

Create

Find a repository...

Search

Refresh

Profile

ChristianKirschberg

GitTest

Clone

Create branch

Create pull request

Compare

Fork

Overview

Source

Commits

Branches

Pull requests

Downloads

Overview

Last updated just now

Language Java

Access level Admin

1 Branch

0 Tags

0 Forks

1 Watcher

HTTPS

https://ChristianKirschberg@bitbucket.org/ChristianKirschberg/gittest.git

Share


Invite users to this repo

Send invitation

Recent activity

1 commit
Pushed to ChristianKirschberg/gittest
9de79fb Initial commit
ChristianKirschberg · 14 minutes ago

Repository created
ChristianKirschberg · 2 days ago



THERE ISN'T A README YET
Create one and tell people where to start and how to contribute.

Create a README

Then, we would like to copy (push) our code from the local repository to the remote repository, so other developers can get (pull) the code.

To do that we, first, add all files except those defined in the .gitignore file by:

```
git add --all
```

Then, we commit our new or changed files to the local repository:

```
git commit -m "Initial commit"
```

When you commit for the first time, git might ask you for your username and email. Fill out this information by following the commands git lists.

(notice the commit message here, in the quotes, where we write a message relevant to the changes to the code we have made).

Then, we push our local repository code to the remote repository.

```
git push origin master
```

After this the base project is in the remote repository and other developers can get (pull) the code which we will do in the next section.

If you get an issue with your password, try changing your password in your bitbucket account and push again.

Case 0.5: Cloning a repository

Adding a second (or third or ...) programmer to the project

The programmers who would like to participate in the project must be invited to the repository. (See **Case 0**).

To get the code from the remote repository for the first time, we clone the repository. This copies the code in the repository to your local machine.

Write the git clone command using windows command prompt to get the right path, if you use the terminal in IntelliJ the path will be wrong.

```
git clone https://lasse_conrad@bitbucket.org/ChristianKirschberg/gittest.git
```

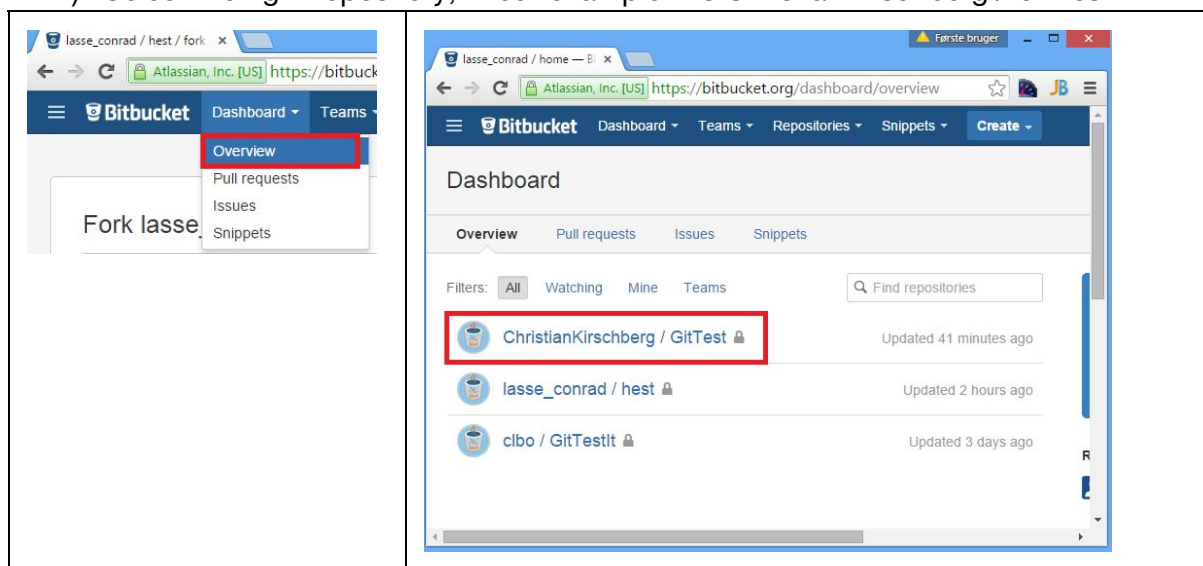
The url is build up in the following way:

```
https://lasse_conrad@bitbucket.org/ChristianKirschberg/gittest.git
```

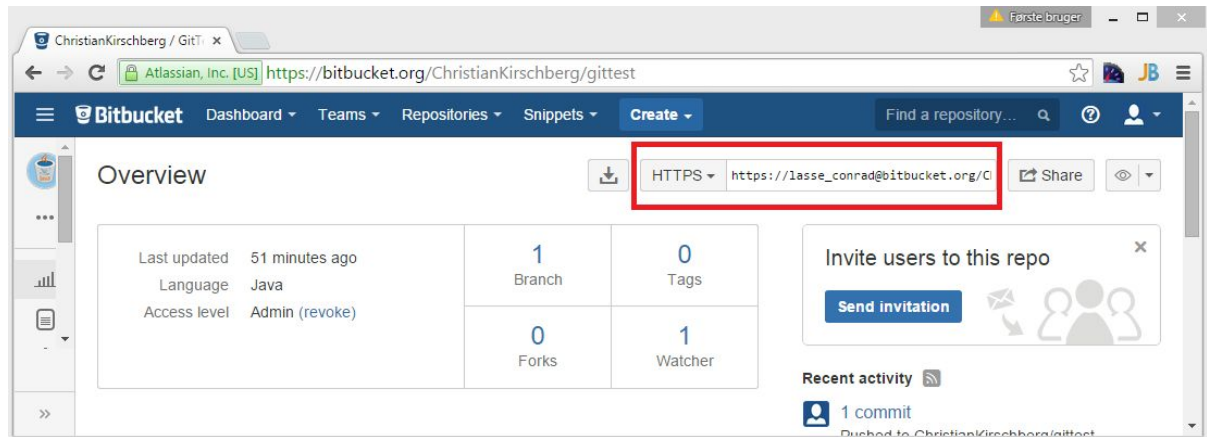
- 1) The username at bitbucket for the programmer who is about to clone the repository
- 2) The username at bitbucket for the owner of the repository which is about to be cloned.
- 3) The repository name

You can cut and paste the right url from Bitbucket using the following steps:

- 1) Select the Dashboard / overview pane.
- 2) Select the right repository, in our example it is ChristianKirschberg / GitTest



3) Cut and paste the url into the terminal after having written the first part of the command i.e. git clone [cut and paste url].



4) Verify the result of the git clone command is similar to this screen output.

```
Terminal
C:\repository>git clone https://lasse_conrad@bitbucket.org/ChristianKirschberg/gittest.git
Cloning into 'gittest'...
Password for 'https://lasse_conrad@bitbucket.org':
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 7 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (7/7), done.
Checking connectivity... done.

C:\repository>
```

Create a new project in your IDE eg. IntelliJ in the folder you cloned to.

After this, you have the code from the shared repository in your local repository and you can start participating in the development of the project, which will work with in the next section.

Case 1: Sharing code via remote repository

Once the repository is setup on multiple developers machines, we would like to share our code extensions. It is important to push (deliver) and pull (receive) code regularly.

There are two scenarios when you want to share your code. Either you get merge conflicts or you get no conflicts. Here we will demonstrate sharing code with no merge conflicts.

Developer 1, in this example called David, adds the following class to the project.

```
public class Car {  
    private String brand;  
    private String model;  
    private double engineSize;  
}
```

Next we would like to share our changes with the other developers, so we first add, commit and push our changes.

Add:

```
git add --all
```

The **git add --all** adds all the new files or changes files to the list of files scheduled to be committed. It does not actually add the changes to the repository but only updates the list of filenames which contains changes or is new.

Commit:

```
git commit -m "Added Person class"
```

the git commit command updates the repository with the changes.

Push:

```
git push origin master
```

Pushes the changes to the shared repository.

Now developer two, in this example Claus, develops the following code:

```
public class Person {  
    private String name;  
    private double height;  
}
```

Claus would like to, also, share this code, so he does the same, add, commit and push.

```
git add --all
```

```
git commit -m "Added Car class"
```

```
git push origin master
```

Notice the output in the terminal after issuing this command.

The push is rejected for Claus because David pushed some changes to the remote repository before Claus pushed his. To solve this Claus must first make a pull to get Davids changes into his repository.

Pull:

```
git pull origin master
```

This is the stage where merge conflicts can occur. In this case there are no merge conflicts.

After pulling the changes Claus can now push his changes to the remote repository. There are no merge conflicts (when Claus pulls Davids changes) because they worked with different classes.

```
git push origin master
```

Now Claus can work with the code shared by David and David can pull Claus changes by issuing a pull command.

In the next example we will see how merge conflicts can occur.

Case 2: Handling merge conflicts

Continuing from the previous case, with no merge conflicts, you can also experience that you get a merge conflict when programmer no. 2, Claus, tries to push his changes.

We will see an example of that in the following:

The following are notes and should be explained like the above cases.

When 2 or more developers work on the same classes, merge conflicts can happen. Here we see this scenario and how to handle it.

Eg. we have the class defined as

```
public class Car {  
    private String brand;  
    private double mph;  
}
```

Programmer1 adds a set method.

```
public class Car {  
    private String brand;  
    private double mph;  
  
    public void setBrand(String brand)  
    {  
        this.brand = brand;  
    }  
}
```

Programmer2 adds a constructor, like:

```
public class Car {  
    private String brand;  
    private double mph;  
  
    public Car(String brand, double mph) {  
        this.brand = brand;  
        this.mph = mph;  
    }  
}
```

Now, programmer1 commits and pushes the changes and that goes well.

Programmer2 commits and pushes the changes and the push is rejected because the new changes created by programmer1 must be pulled first.

Programmer2 makes a pull and gets a merge conflict because Git cannot automatically merge the changes. Programmer 2 must now edit the conflicted files,

add, commit and push them. Programmer 2 decides what to keep and what to delete (if anything).

It is a good idea to push your changes often to avoid handling merge conflicts. It is always the last programmer to push their changes who must handle the merge conflicts (just before he or she is supposed to leave the office for the day, but oh no, merge conflicts to handle...)

Case 3: Using Branches

The following are notes and should be explained like the above cases.

Features should be developed in branches and then merged into the main-development branch when the feature is finished. That way multiple programmers working on the feature can share the code without pushing their (not finished) code into other programmers' (who are working on something else) code base.

Create a branch:

Programmer 1:

```
git checkout -b smart-feature
```

where smart-feature is the name of the new branch you wish to create and checkout.

Then, programmer1, makes some changes to the code, working on the "smart-feature".

After commit code changes, programmer1, uses the following commands. Note the push-command where the new branch name is used.

```
git add --all
```

```
git commit "some message that describes what you did"
```

```
git push origin smart-feature
```

Programmer 2 can get all the latest changes, eg. branch names by running the command:

```
git fetch --all
```

You can see all branches, local and remote, by running the following command:

```
git branch -a
```

To create a new local branch to pull the added code in the smart-feature branch, programmer2 would run:

```
git checkout -b smart-feature
```

creating a new local branch with the same name as the remote branch created by programmer 1.

Then, programmer2, can pull the content by running:

```
git pull origin smart-feature
```

Both programmers are now working in the smart-feature branch.

Extra: (git push --set-upstream origin smart-feature): Sets default when writing git pull or git push.

Merge of master and smart-feature:

```
git checkout master
```

```
git merge smart-feature
```

Delete a feature branch that you do not use anymore:

```
git branch -d smart-feature
```

To delete a remote repo.

```
git push origin --delete smart-feature
```

To update the local database of remote branches

```
git fetch -p
```

Case 4: Going back to a previous commit

...in case you made some changes that you do not want to keep.

The following are notes and should be explained like the above cases.

Programmer 1 makes some changes, but regrets the changes and just want to go back to the latest code base before the changes.

```
git reset --hard
```

(this will delete your work since last commit. **Beware, there is no way to get your code back after this!**)

An alternative to this is that you would like to go back to the last commit, but with the possibility of going back to the code you are working on. (A bit like “undo” and “redo”). In that case you must “stash” your changes, go back by resetting, and then redo by using pop.

First stash:

```
git stash
```

Then go back by using reset

```
git reset --hard
```

Redo by using pop to go back to your stashed changes.

```
git stash pop
```

After this, your stash is removed, so if you want to do it again, you must create a new stash, reset and pop.

Case 5: Accidentally committed files to your repository

At some point you will most likely end up in a situation where you have committed a file or directory to your online repository, which you did not intend to have committed.

If you use the command

```
git rm -r somedir
```

You will delete the online version of the directory file but also the local file as well.

If you want to delete only the shared online version of the file and keep your local copy you will have to use a different approach.

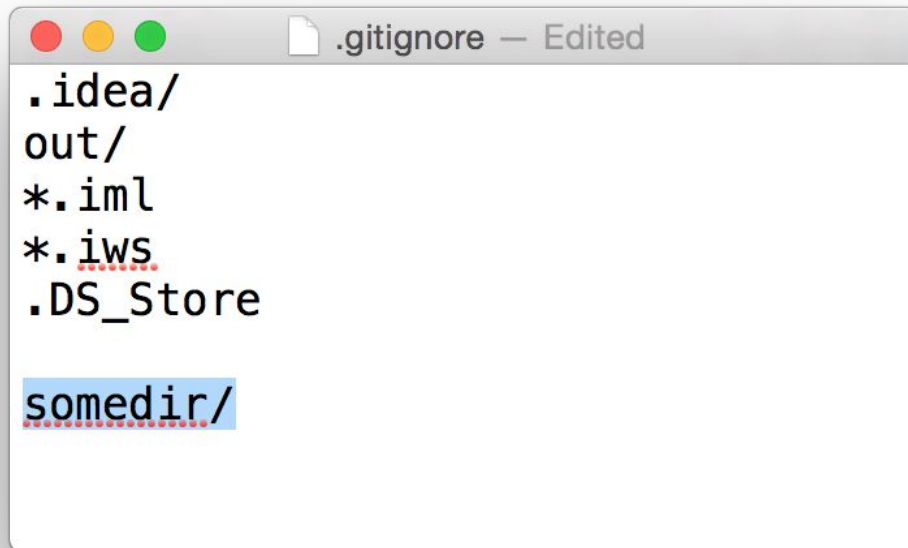
You should remove the file from being monitored by git

```
git rm --cache -r somedir
```

Now the folder is no longer being monitored by git, and it is just a normal folder in your local filesystem.

To prevent the file from being monitored again in the future and from being placed in the online repository you should then modify your .gitignore file so the unwanted file or folder

is no longer being committed



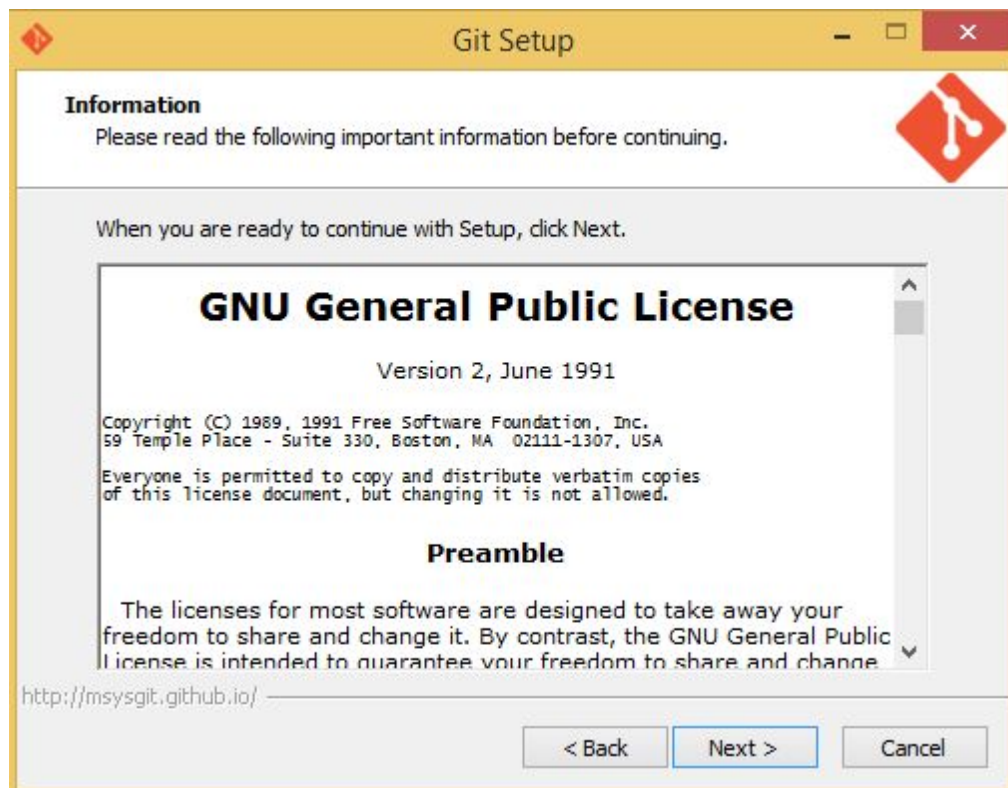
Then add, commit and push

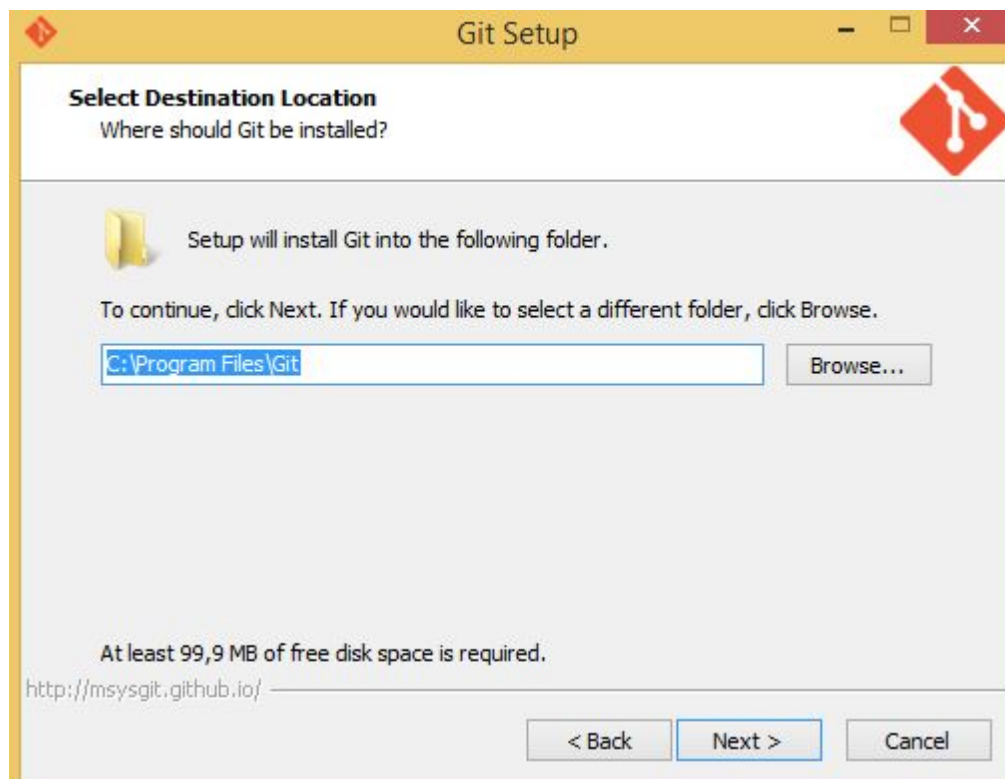
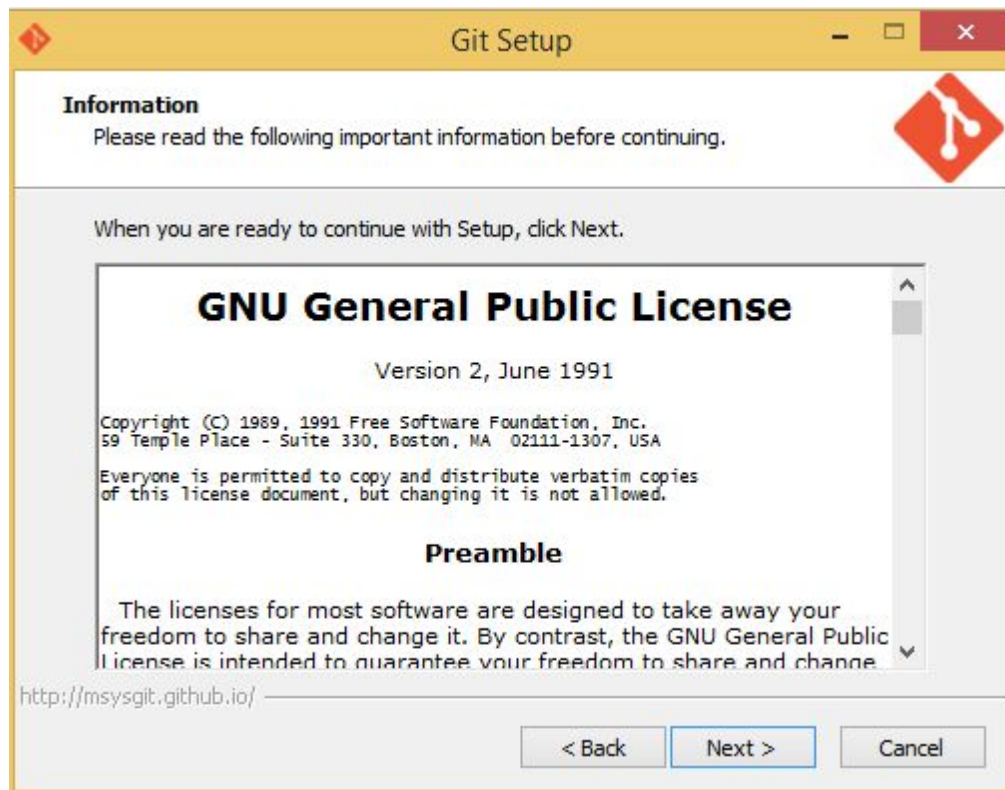
```
git add --all
```

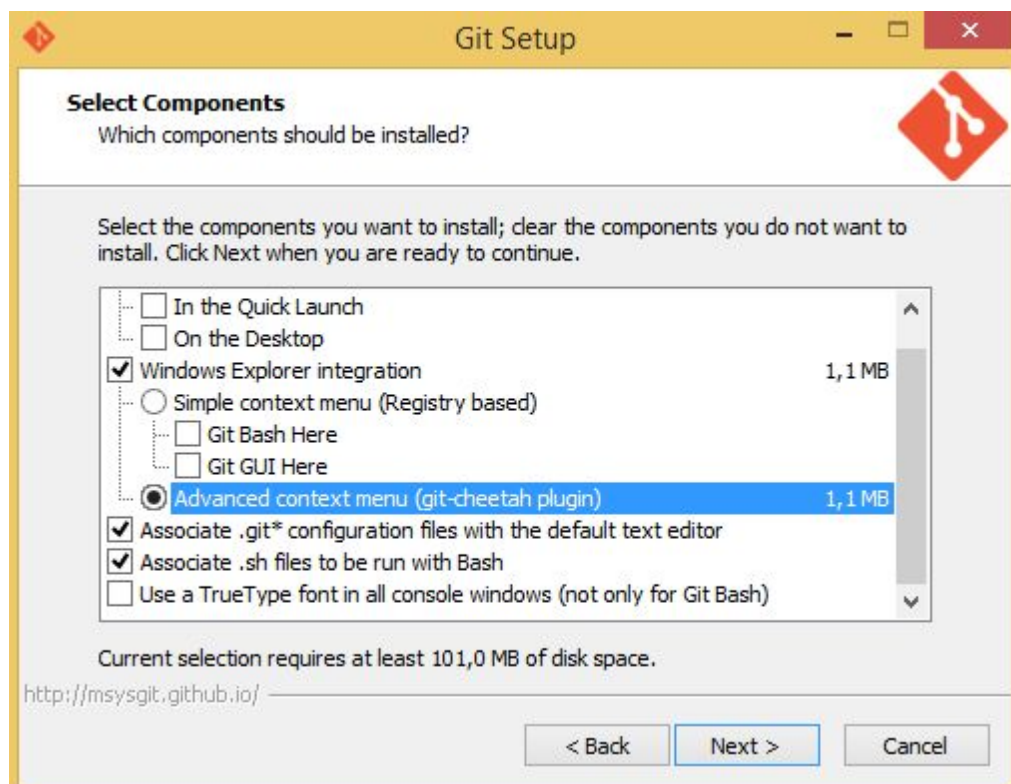
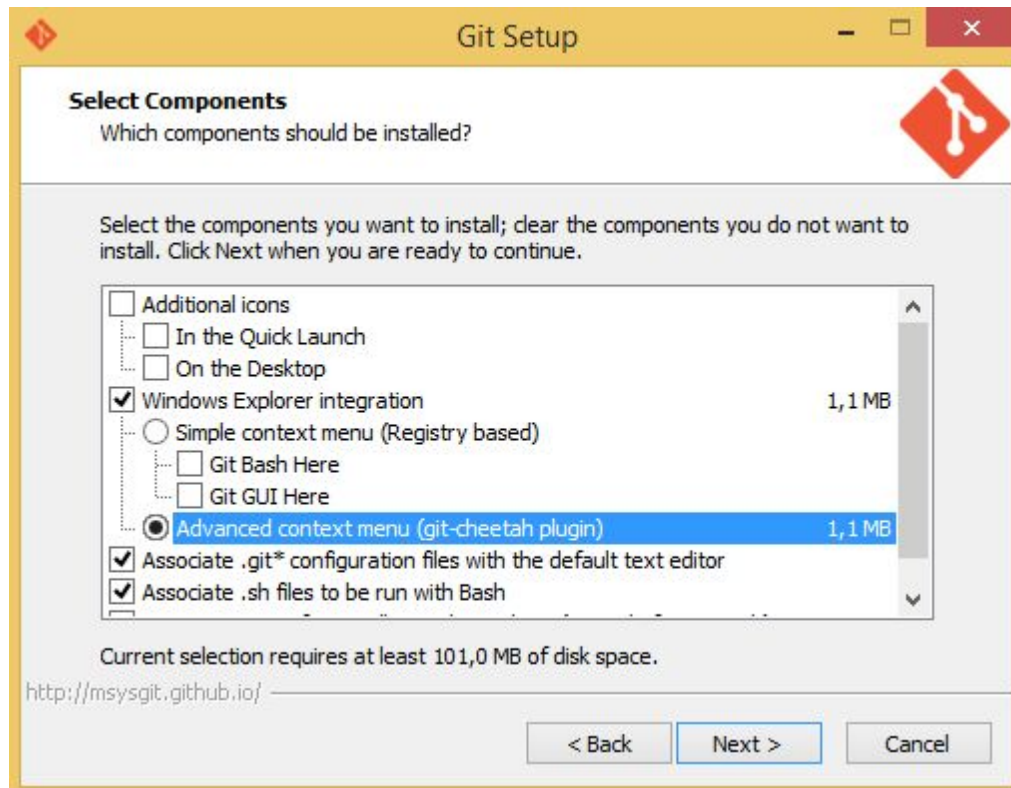
```
git commit -m "removed somedir from online repository"
```

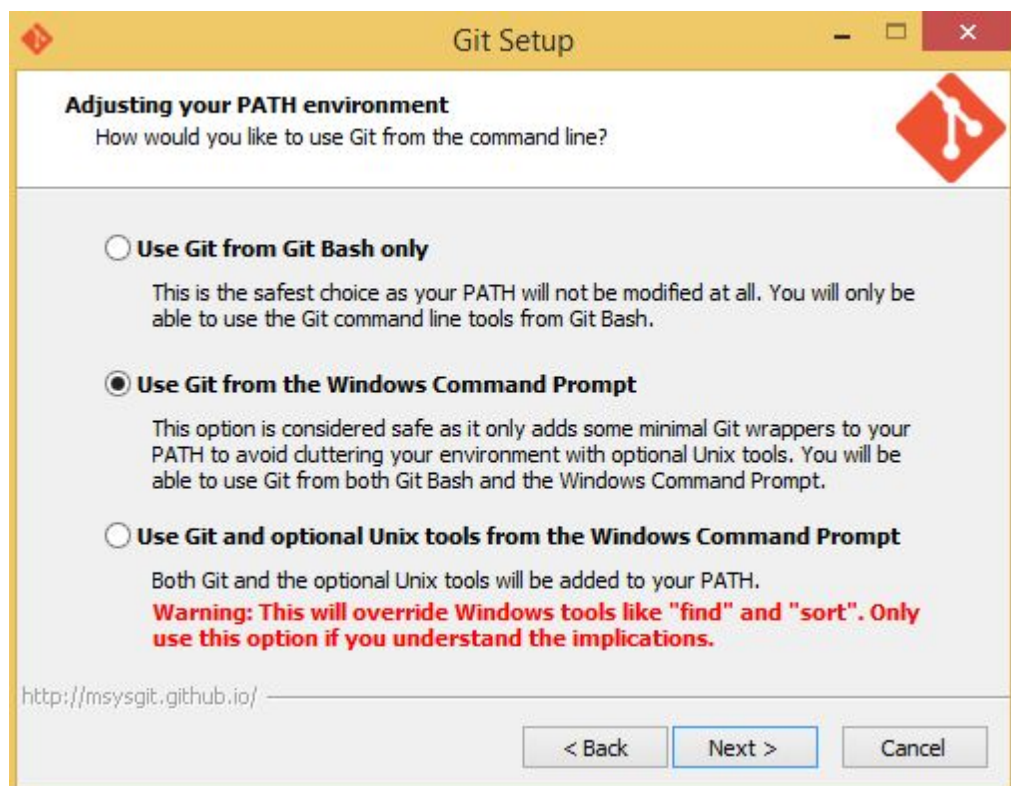
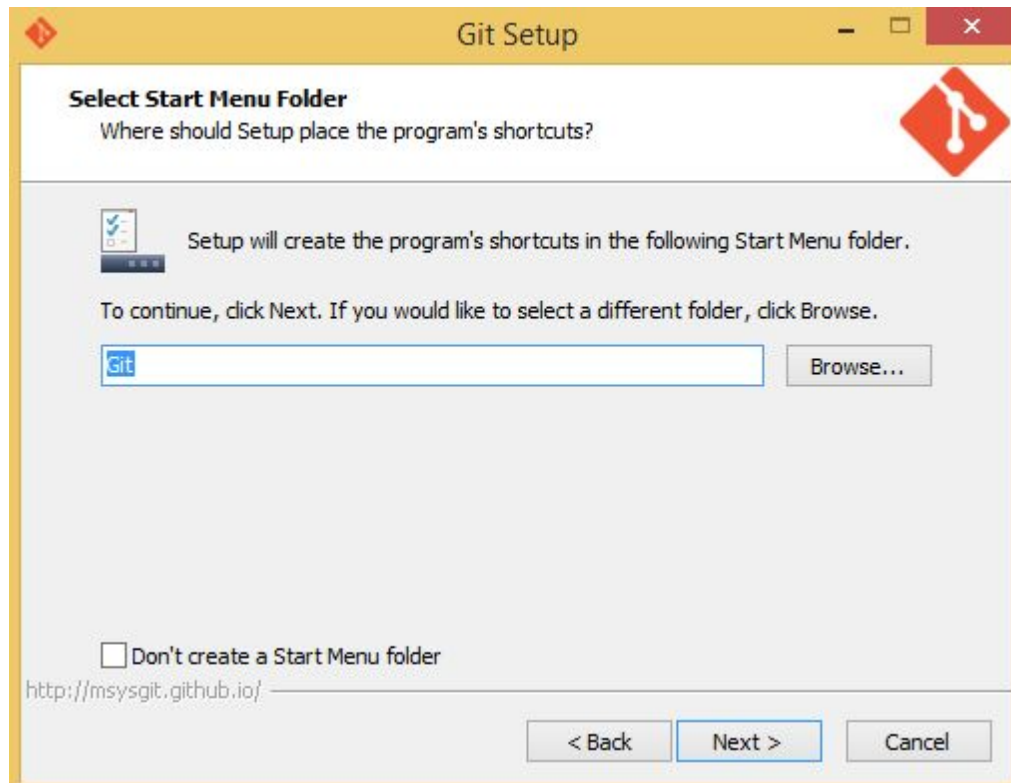
```
git push
```

Now the folder should have been removed online.

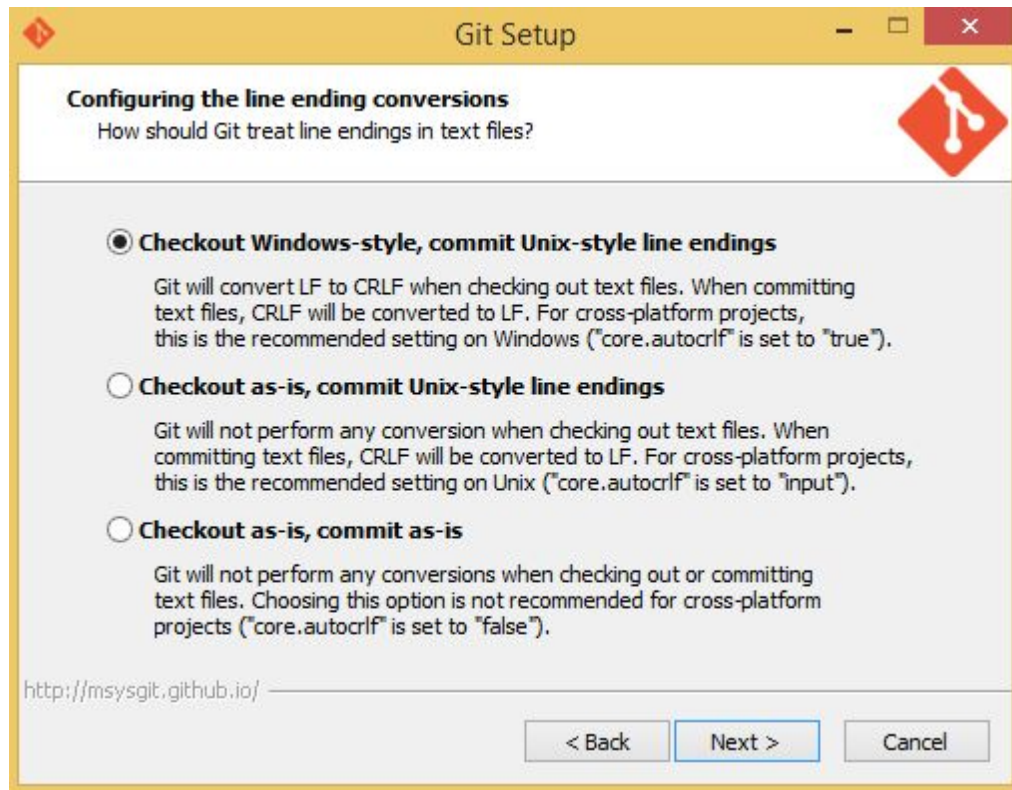








Change the radio button to **Use Git from the windows Command Prompt**. This will make it possible to use the git commands from the Windows Command Prompt.





Uncheck View ReleaseNotes.rtf if you like.


```
Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\laco>cd \
```

```
Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\laco>cd \
C:\>md gitrepo_
```

```
Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\laco>cd \
C:\>md gitrepo
C:\>cd gitrepo
```

```
Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\laco>cd \
C:\>md gitrepo
C:\>cd gitrepo
C:\gitrepo>git init_
```

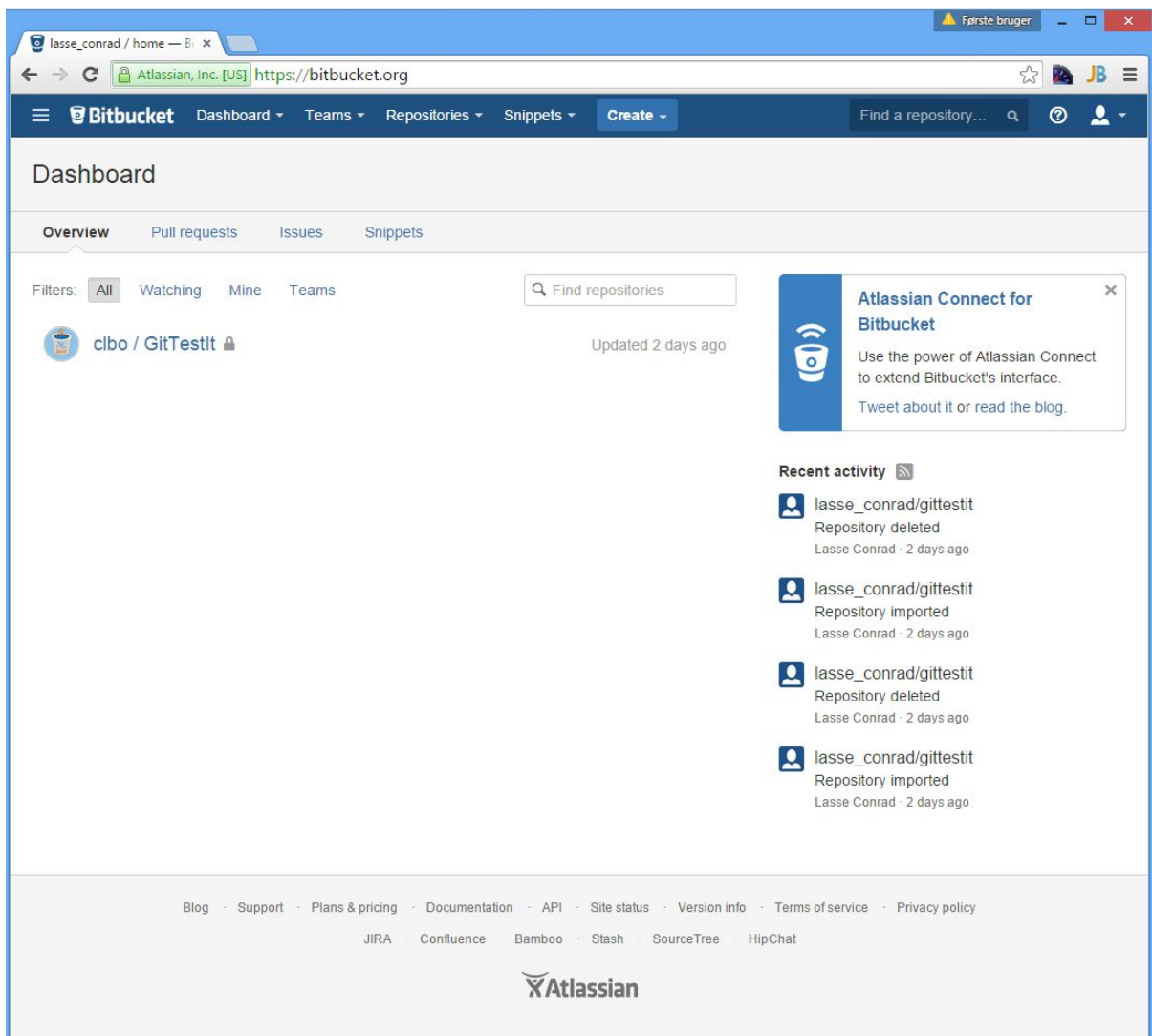
```
Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\laco>cd \
C:\>md gitrepo
C:\>cd gitrepo
C:\gitrepo>git init
Initialized empty Git repository in C:/gitrepo/.git/
C:\gitrepo>
```

```
Command Prompt

C:\gitrepo>git clone https://lasse_conrad@bitbucket.org/clbo/gittestit.git
Cloning into 'gittestit'...
Password for 'https://lasse_conrad@bitbucket.org':
remote: Counting objects: 124, done.
remote: Compressing objects: 100% (110/110), done.
Rremote: Total 124 (delta 33), reused 0 (delta 0)
Receiving objects: 100% (124/124), 17.80 KiB | 0 bytes/s, done.
Resolving deltas: 100% (33/33), done.

Checking connectivity... done.
C:\gitrepo>
```



clbo / GitTestit — Bitbucket

Atlassian, Inc. [US]https://bitbucket.org/clbo/gittestit

Find a repository...

BitbucketDashboardTeamsRepositoriesSnippetsCreate

Overview

Last updated2015-06-29
LanguageJava
Access levelAdmin (revoke)

1Branch
0Forks

0Tags
0Watchers

Invite users to this repo

Send invitation

Recent activity

1 commit
Pushed to clbo/gittestit
53220ea whitespace
Claus Bové · 2 days ago

caa8fed
Commit deleted from clbo/gittestit
ChristianKirschberg · 2 days ago

13d3242
Commit deleted from clbo/gittestit
ChristianKirschberg · 2 days ago

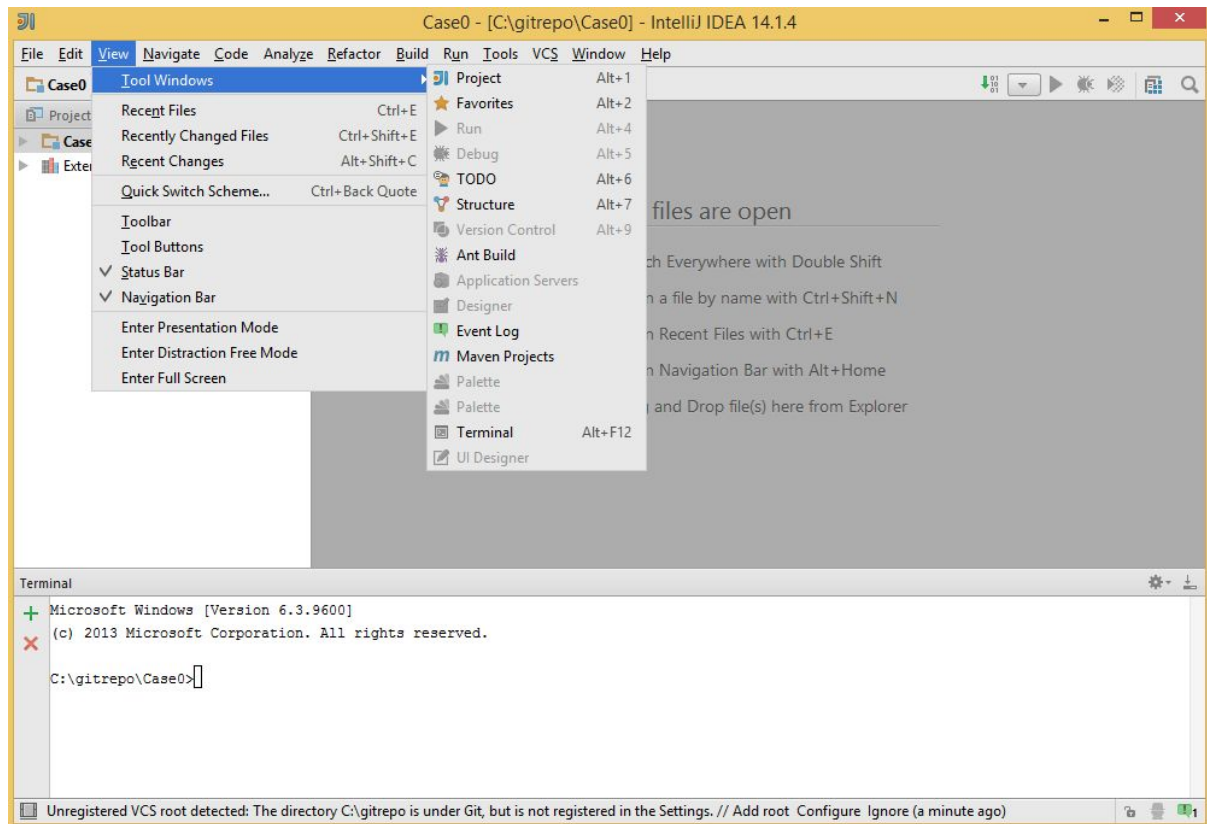
1 commit
Pushed to clbo/gittestit
13d3242 drive added again
Claus Bové · 2 days ago

1 commit
Pushed to clbo/gittestit
caa8fed smart feature done
ChristianKirschberg · 2 days ago

2 commits
Pushed to clbo/gittestit
0d1e58f www
204aee0 test
Claus Bové · 2 days ago

Git Test Exercise

Edit README



Opens terminal view in IntelliJ

Why do we need version control?

TODO:

Add paragraph about: Why do we need version control.

<http://betterexplained.com/articles/a-visual-guide-to-version-control/>