Eclipse+EGit Getting Started

Portable Git for Windows: setting the $HOME environment variable to allow complete portability (including SSL keys and configuration for use with GitHub)

November 17, 2010

Portable Git is a completely standalone version of the Git distributed version control system. All you need do to get going is to download a file and unzip it into a folder. In theory, what you end up with is a self-contained Git install which doesn't touch any other part of your Windows system, meaning that you can take your entire, ready configured version control system with you from computer to computer on a portable drive.

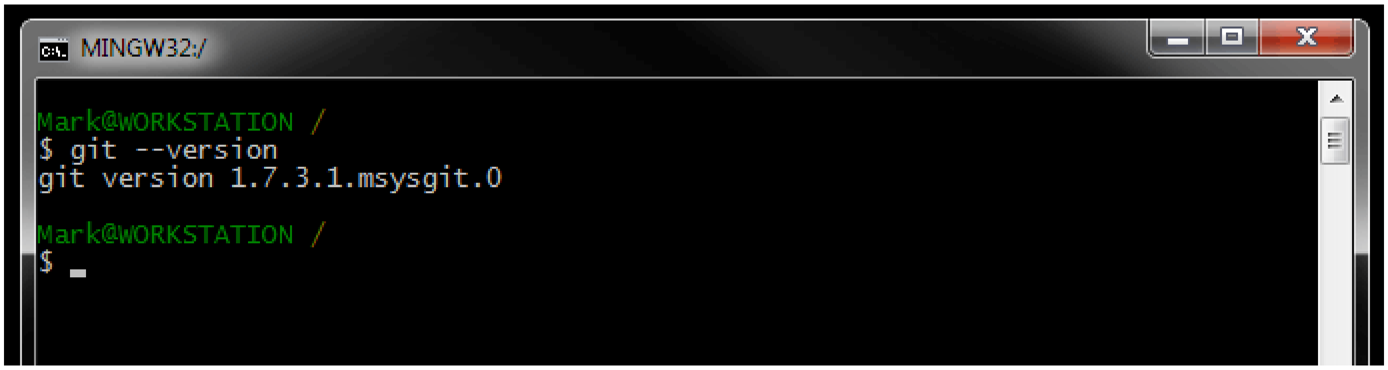
However, there are a couple of issues which make Portable Git not quite as portable as it could be. In this article, I'm going to explain the gotchas and walk you through creating a tweaked Portable Git for Windows install which is truly self contained. Even if you don't need to carry it around with you, it's still useful to keep all your Git configuration files and scripts in one place where you can easily find them.

The first step is to download the zip file containing Portable Git. Once you have the file, unzip its contents into a folder of your choice. I chose to put mine in `E:\Git`.

Now open the new Git folder. Inside you'll see a number of folders and files. Run the batch file named `git-bash.bat` and you should see a shell prompt screen. Type the following:

```
git --version
```

You should see something like this (possibly with a different version number, depending on the version you downloaded):



```
Mark@WORKSTATION /
$ git --version
git version 1.7.3.1.msysgit.0

Mark@WORKSTATION /
$ 
```

By default, Git and MSys (the minimal UNIX system it runs under on Windows) look for configuration items—startup scripts, SSL keys etc—in the current user's home folder, which on my Windows 7 machine is `C:\Users\Mark`.

There's sensible reasoning behind this, because it allows for different users on the same machine to have their own individual Git settings. However, if you install Git on a portable drive, you'll want your settings to travel with the installation—which obviously they won't if it is looking for them in a folder which may not exist on other computers.

So, what we need to do is tell Portable Git to treat a specific location within its own folder as the home folder; that way we can copy the whole Git folder anywhere we like and the settings will travel with it.

Create a folder within the Git folder called `home`. I also created a sub folder within this folder called `mark`, to follow UNIX convention and in case I want to create different settings in the future.

Now we need to set the `$HOME` environment variable, which is how Git knows where to look for the current user's home folder. I found the easiest way to do this was to edit`E:\Git\etc\profile` and override the `$HOME` variable on startup. Find these lines:

```
# normalize HOME to unix path
```

```
HOME="$(cd "$HOME" ; pwd)"
```

And on the line before them add this:

```
HOME="/home/mark"
```

Where the path is the folder you just created, relative to the folder in which you installed Git (thanks to commenter Genesis2001 for the heads-up on this). Note that this is in UNIX format rather than Windows, so `\home\mark` became `/home/mark`. Close any Git shell windows you have open and run `git-bash.bat` again. When the prompt opens, type:

```
echo $HOME
```

You should see the UNIX path you entered: this means Git is now using your new folder as its home folder. Now we'll set the global name and email address which will be associated with Git commits; in order to use GitHub, we also need to generate an SSL key pair for authentication. Enter the following two commands at the shell prompt to set the global variables:

```
git config --global user.name "Your Name"
```

```
git config --global user.email you@yourdomain.com
```

Now that we have pointed Portable Git to the correct folder, you should be able to follow the excellent GitHub Help guide to generating an SSL key pair. Once you’ve done that, change to the new home folder and create a file named profile (no extension).*

Open it up, paste in the following code from this GitHub guide and save:

```
SSH_ENV="$HOME/.ssh/environment"
```

```
# start the ssh-agent
function start_agent {
    echo "Initializing new SSH agent..."
    # spawn ssh-agent
    ssh-agent | sed 's/^echo/#echo/' > $SSH_ENV
    echo succeeded
    chmod 600 $SSH_ENV
    . $SSH_ENV > /dev/null
    ssh-add
}

# test for identities
function test_identities {
    # test whether standard identities have been added to the agent already
    ssh-add -l | grep "The agent has no identities" > /dev/null
    if [ $? -eq 0 ]; then
        ssh-add
        # $SSH_AUTH_SOCK broken so we start a new proper agent
        if [ $? -eq 2 ];then
            start_agent
        fi
    fi
}

# check for running ssh-agent with proper $SSH_AGENT_PID
ps -ef | grep $SSH_AGENT_PID | grep ssh-agent > /dev/null
if [ $? -eq 0 ]; then
    test_identities
# if $SSH_AGENT_PID is not properly set, we might be able to load one from
# $SSH_ENV
else
    . $SSH_ENV > /dev/null
    ps -ef | grep $SSH_AGENT_PID | grep ssh-agent > /dev/null
    if [ $? -eq 0 ]; then
        test_identities
    else
        start_agent
    fi
fi
```

This will ensure that `ssh-agent` is run when you start the Git shell, meaning you don't have to type in your SSL Passphrase every time you authenticate.

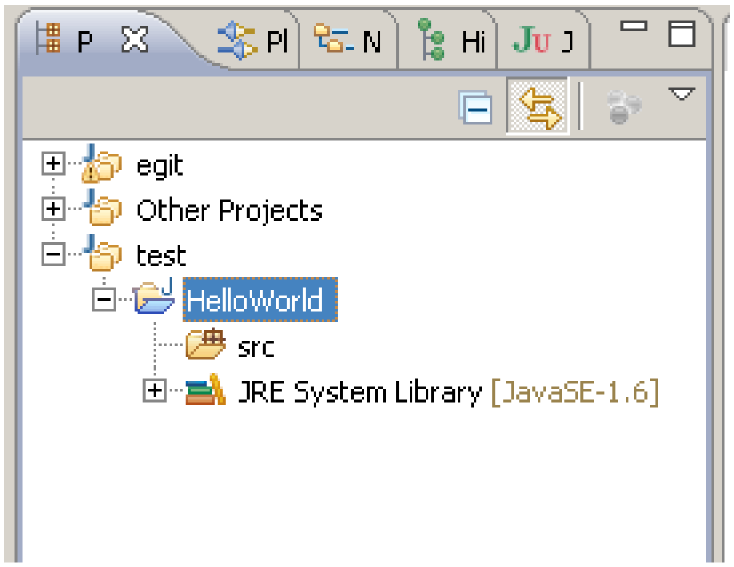
And that's it! You now have a completely standalone Portable Git install. Simply copy the whole Git folder to another machine or portable drive and you can continue working with your Git shell, without having to go through all these configuration steps again in every new situation.

Cautionary Note: if you're carrying your install about on a USB drive, be careful with your private SSL key! If someone gets hold of it, they can potentially authenticate as you until you revoke that key's access from within GitHub.

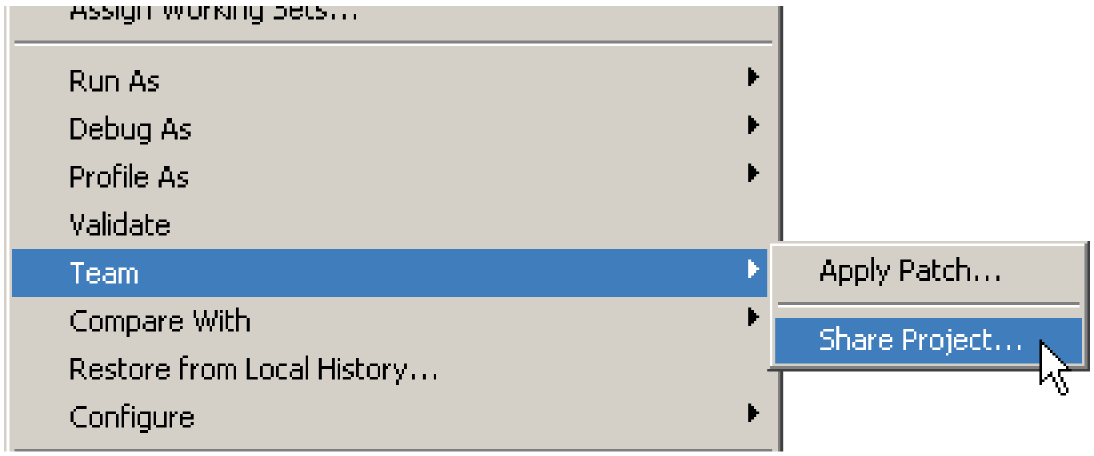
* You're right, this file would normally begin with a period character under UNIX (.profile), but MSYS lets you omit it for ease of working on Windows.

Eclipse & Git

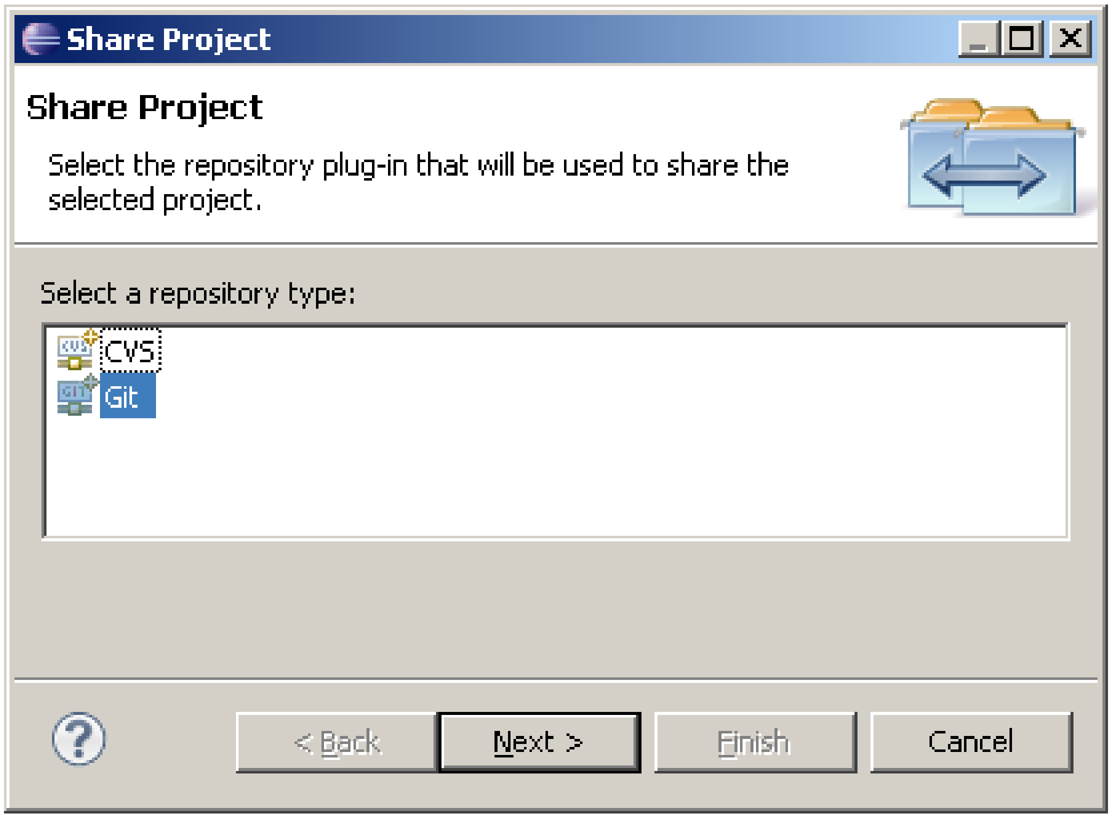
Create a new Java project "HelloWorld"



Select "File" -> "Team" -> "Share Project"



Select repository type "Git" and click "Next"

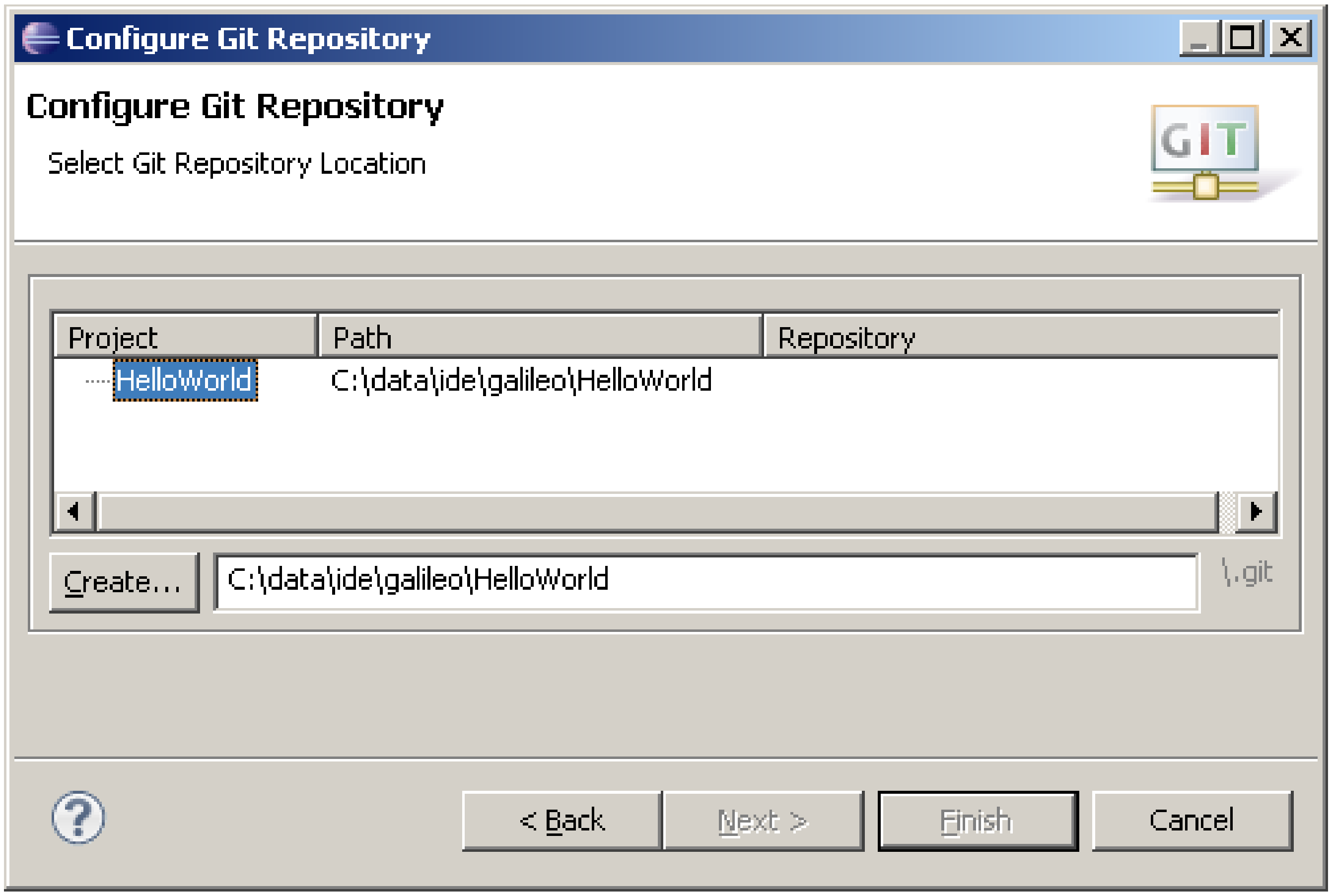

Share Project

Share Project

Select the repository plug-in that will be used to share the selected project.

Select a repository type:

To configure the Git repository select the new Eclipse project HelloWorld


Configure Git Repository

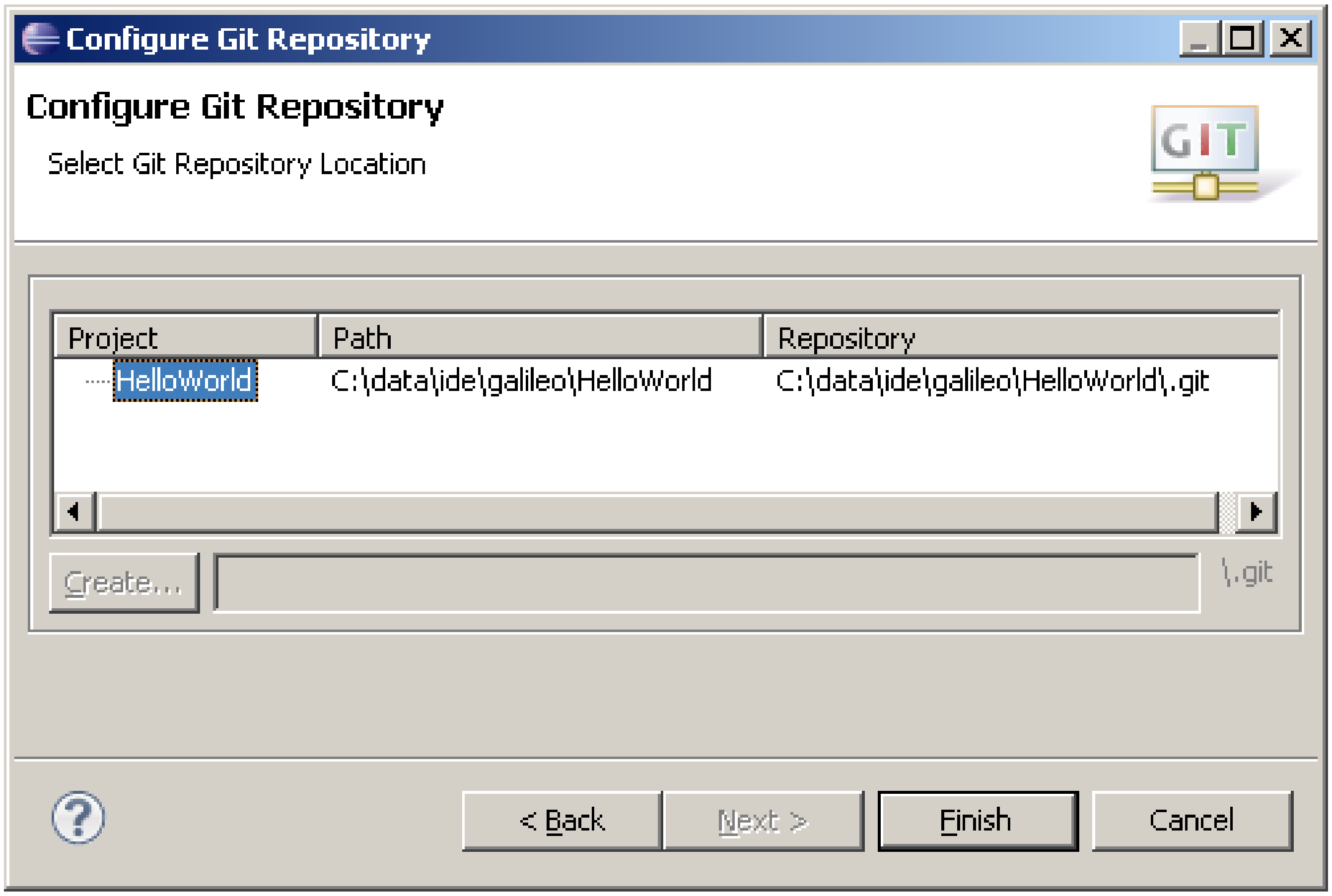
Configure Git Repository

Select Git Repository Location

Project	Path	Repository
HelloWorld	C:\data\ide\galileo\HelloWorld	

Create... C:\data\ide\galileo\HelloWorld \.git

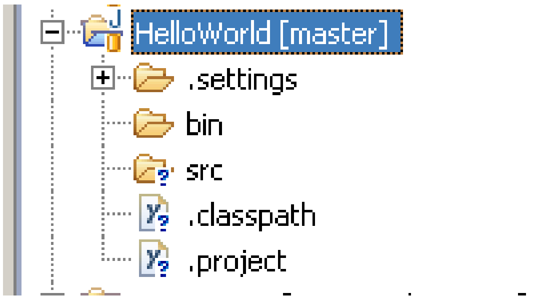
Click "Create" to initialize a new Git repository for the HelloWorld project. If your project already resides in the working tree of an exisiting GIT repository the repository is chosen automatically.



Project	Path	Repository
HelloWorld	C:\data\ide\galileo\HelloWorld	C:\data\ide\galileo\HelloWorld\.git

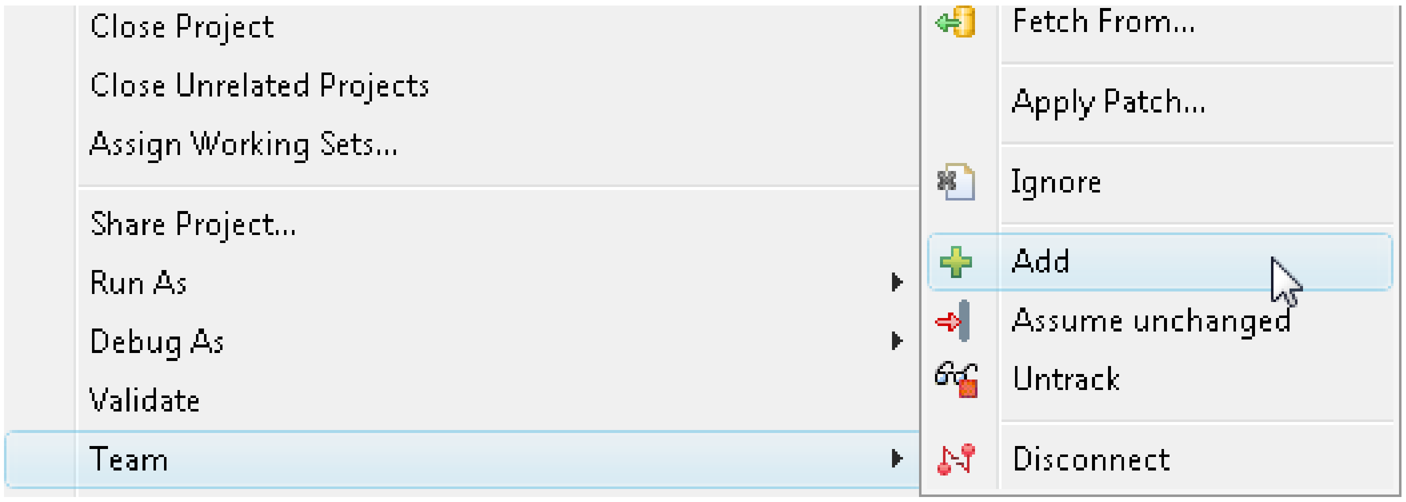
Click "Finish" to close the wizard.

The decorator text "[master]" behind the project shows that this project is tracked in a repository on the master branch and the question mark decorators show that the ".classpath" and ".project" files are not yet under version control



```
HelloWorld [master]
  .settings
  bin
  src
  .classpath
  .project
```

Select "Team" -> "Add to version control" on the project node

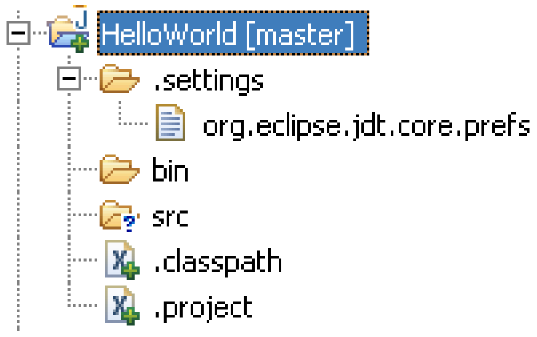


The plus decorators show that now the ".classpath" and ".project" files are added to version control

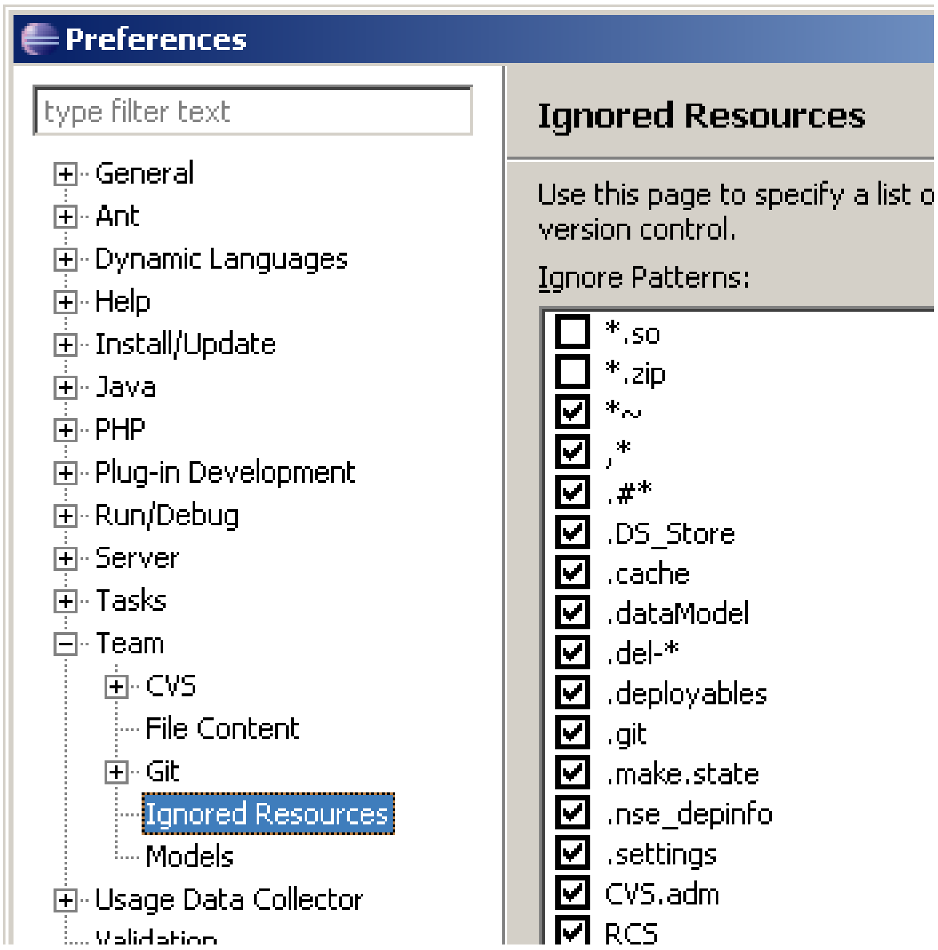
Create a file .gitignore in the project folder with the following content:

bin

This excludes the bin folder from GIT's list of untracked files. Add .gitignore to version control.



The file under ".settings" is not added to version control since it is by default on the list of ignored resources which can be changed in "Preferences" -> "Team" -> "Ignored Resources"


Preferences

type filter text

- General
- Ant
- Dynamic Languages
- Help
- Install/Update
- Java
- PHP
- Plug-in Development
- Run/Debug
- Server
- Tasks
- Team
  - CVS
  - File Content
  - Git
  - Ignored Resources
  - Models
- Usage Data Collector
- Validation

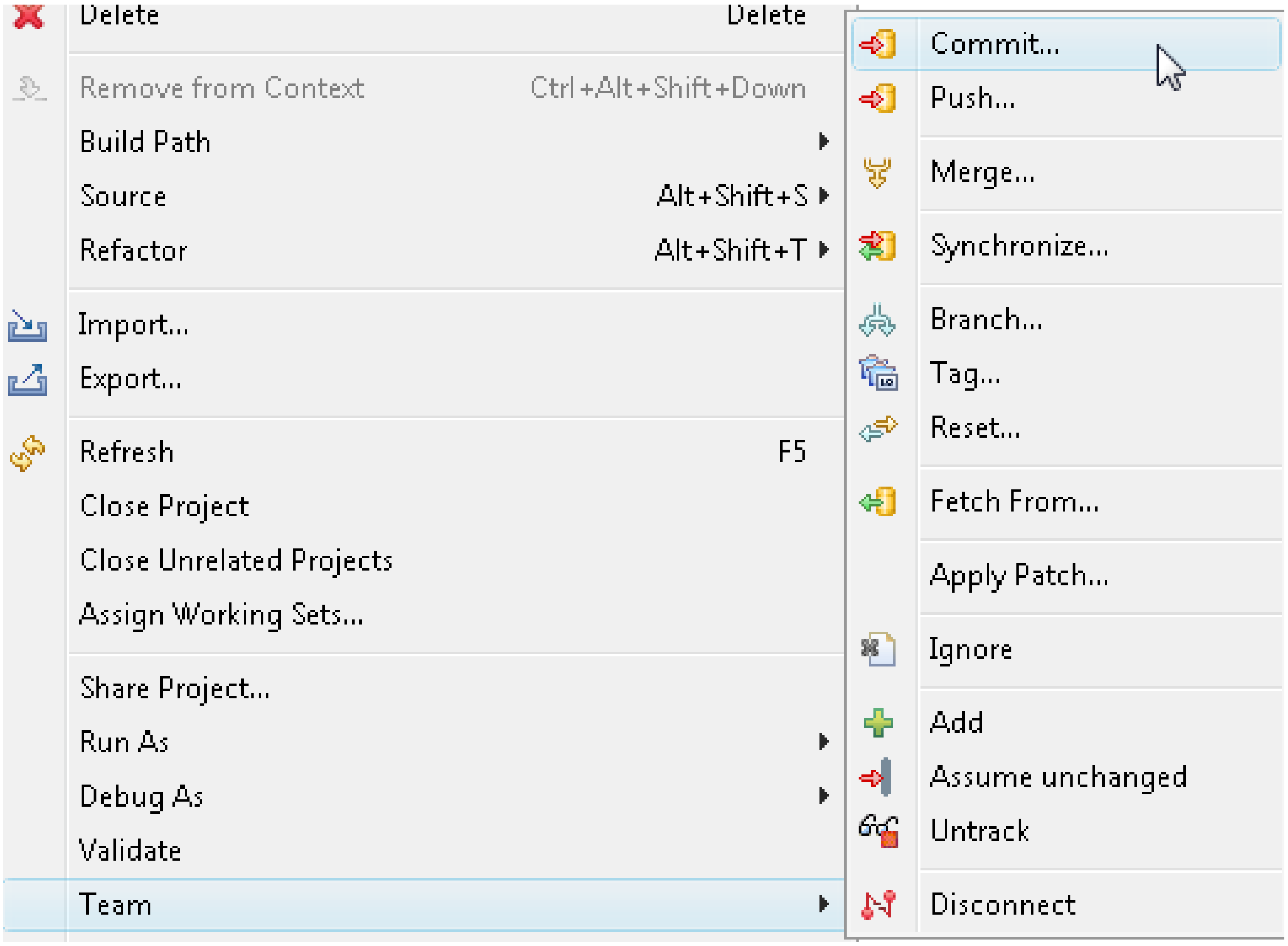
Ignored Resources

Use this page to specify a list o
version control.

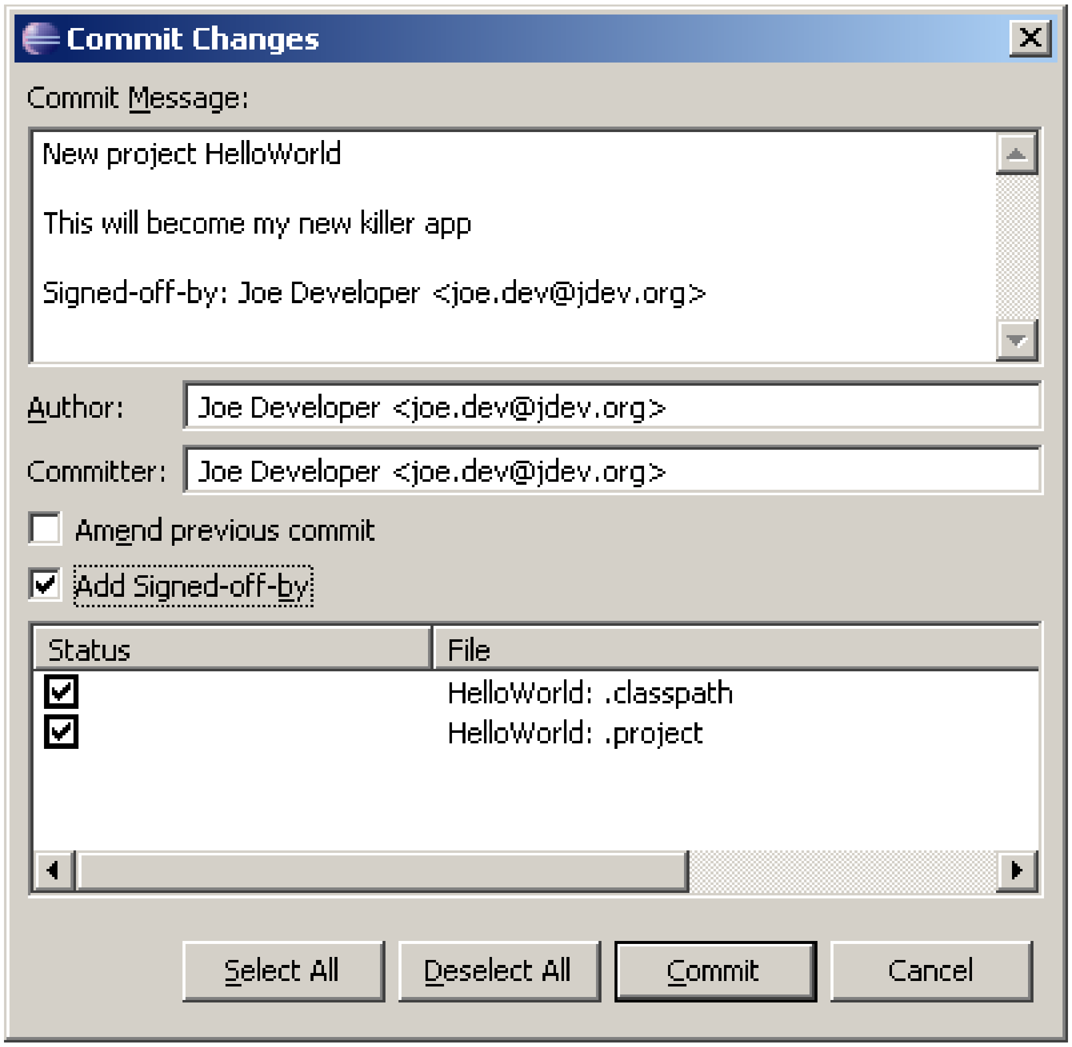
Ignore Patterns:

- [ ] *.so
- [ ] *.zip
- [x] *~
- [x] ,*
- [x] .#*
- [x] .DS_Store
- [x] .cache
- [x] .dataModel
- [x] .del-*
- [x] .deployables
- [x] .git
- [x] .make.state
- [x] .nse_depinfo
- [x] .settings
- [x] CVS.adm
- [x] RCS

Select "Team" -> "Commit" from the context menu on the project



Enter a commit message explaining your change, the first line (separated by an empty line) will become the short log for this commit. By default the author and committer are taken from the .gitconfig file in your home directory. You may check the checkbox "Add Signed-of-by" to add a Signed-off-by tag. If you are committing the change of another author you may alter the author field to give the name and email address of the author. Click "Commit" to commit your first change.



Commit Message:

```
New project HelloWorld

This will become my new killer app

Signed-off-by: Joe Developer <joe.dev@jdev.org>
```

Author: Joe Developer <joe.dev@jdev.org>

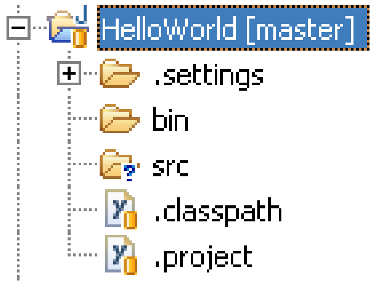
Committer: Joe Developer <joe.dev@jdev.org>

☐ Amend previous commit

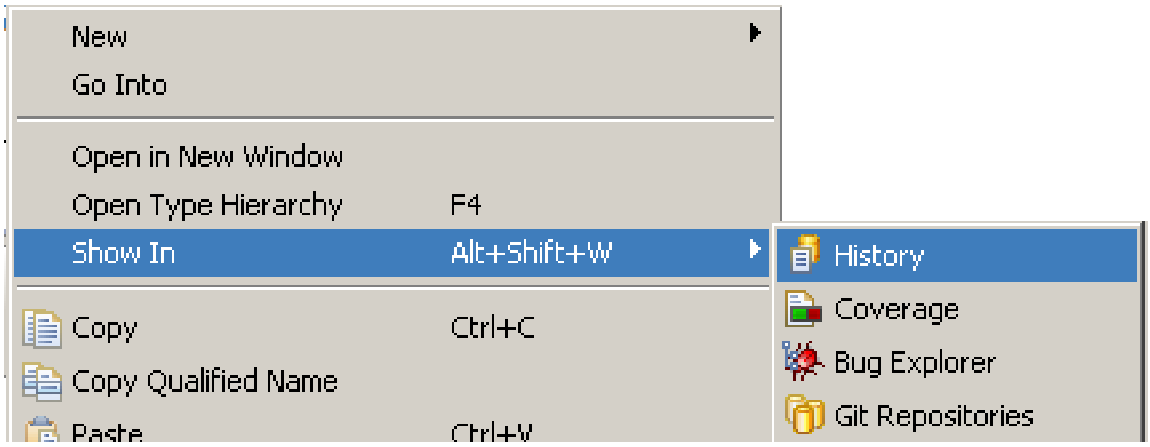
☑ Add Signed-off-by

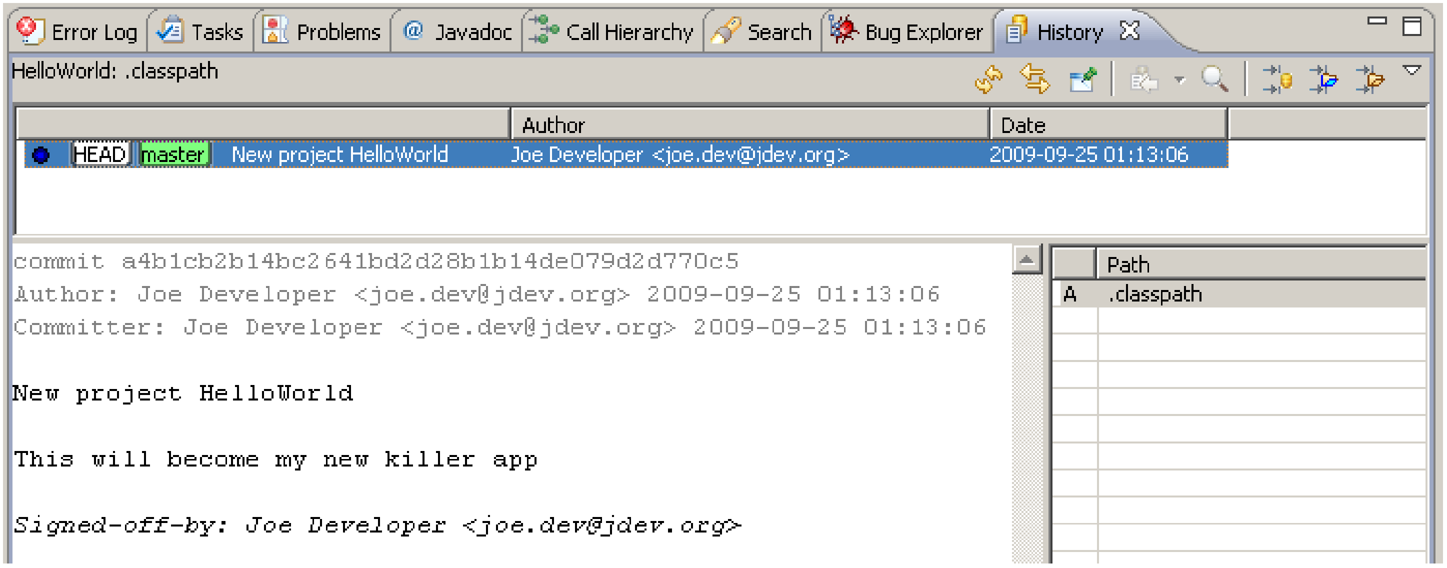
Status	File
☑	HelloWorld: .classpath
☑	HelloWorld: .project

Note that the decorators of the committed files changed.



Select "Team" -> "Show in Resource History" from the context menu to inspect the history of a resource



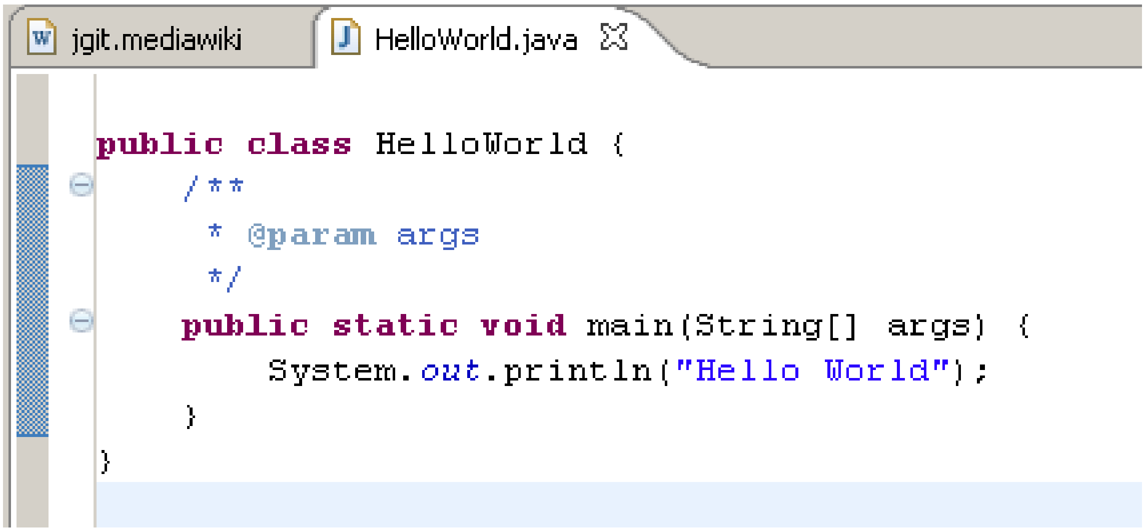


	Author	Date
HEAD master New project HelloWorld	Joe Developer <joe.dev@jdev.org>	2009-09-25 01:13:06

```
commit a4b1cb2b14bc2641bd2d28b1b14de079d2d770c5
Author: Joe Developer <joe.dev@jdev.org> 2009-09-25 01:13:06
Committer: Joe Developer <joe.dev@jdev.org> 2009-09-25 01:13:06

New project HelloWorld

This will become my new killer app

Signed-off-by: Joe Developer <joe.dev@jdev.org>
```

	Path
A	.classpath

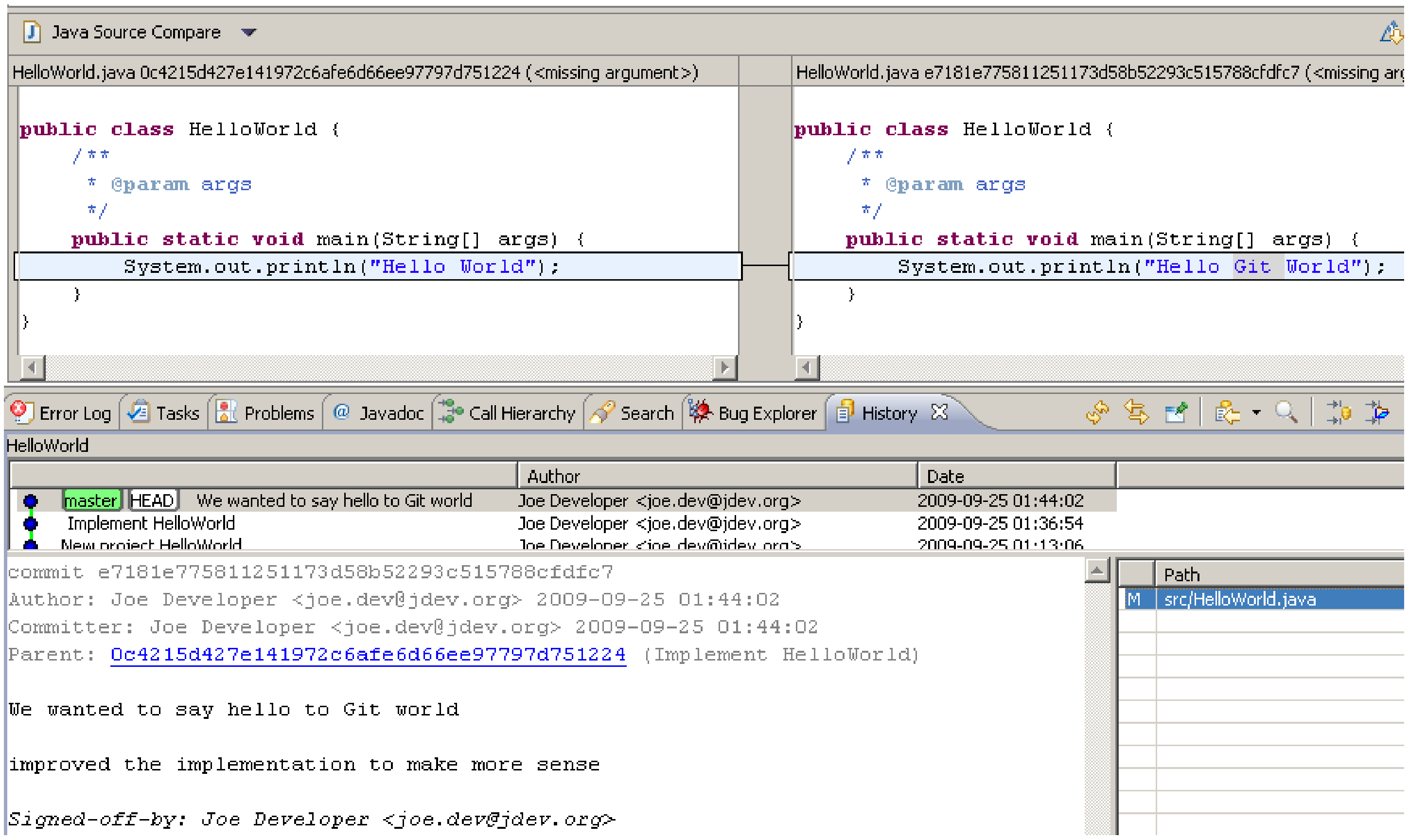
Create a new Java class HelloWorld and implement it, then add it to version control and commit your change. Improve your implementation and commit the improved class, the resource history should now show 2 commits for this class.



```java
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

Double click "src/HelloWorld.java" in the Resource History View to open your last committed change in the

Eclipse compare view



HelloWorld.java 0c4215d427e141972c6afe6d66ee97797d751224 (<missing argument>)

```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

HelloWorld.java e7181e775811251173d58b52293c515788cfdfc7 (<missing ar

```
public class HelloWorld {
    /**
     * @param args
     */
    public static void main(String[] args) {
        System.out.println("Hello Git World");
    }
}
```

	Author	Date
master HEAD We wanted to say hello to Git world	Joe Developer <joe.dev@jdev.org>	2009-09-25 01:44:02
Implement HelloWorld	Joe Developer <joe.dev@jdev.org>	2009-09-25 01:36:54
New project HelloWorld	Joe Developer <joe.dev@jdev.org>	2009-09-25 01:13:06

```
commit e7181e775811251173d58b52293c515788cfdfc7
Author: Joe Developer <joe.dev@jdev.org> 2009-09-25 01:44:02
Committer: Joe Developer <joe.dev@jdev.org> 2009-09-25 01:44:02
Parent: 0c4215d427e141972c6afe6d66ee97797d751224 (Implement HelloWorld)

We wanted to say hello to Git world

improved the implementation to make more sense

Signed-off-by: Joe Developer <joe.dev@jdev.org>
```

	Path
M	src/HelloWorld.java