

#25. Use Source Control in a Single-Developer Environment



How Do I...?

▶ Part 1: Install Source Control

1. Browse to the ClassFiles folder on your desktop, and then double-click **Git-[version]-preview-[date].exe**.



TIP: You can download these files at <http://www.git-scm.com>.

The Git Welcome screen appears.

2. Click **Next**.

The Information screen appears.

3. Click **Next**.

The Select Startup Menu Folder screen appears.

4. Click **Next**.

The Select Additional Tasks screen appears.

5. Click **Next**.

The Adjusting your PATH environment screen.

6. Click the **Run Git from Windows Command Prompt** option.

This option allows you to run Git from a command prompt if you ever need to set up source control in a multi-developer environment.

The Choosing the SSH executable screen appears.

7. Click **Next**.

The Choosing the CR/LF behavior screen appears.

8. Click **Next**.

The installation starts.

9. When the installation is complete, **Finish**.

► Part 2: Use Source Control

1. From the **Start** menu, point to **All Programs**, point to **Git**, and then click **Git GUI**.
2. Click **Create New Repository**.
3. In the **Directory** box, click **Browse**.

Use the
VFS7.5.2 on the
root of C:\.



4. In the Browse for Folder window, browse to the location of your working project workspace, and then click **OK**.
5. Click **Create**.

The list of Unstaged Changes appears in the top, left corner of the screen. For the initial add to source control, you need to add these files to the Stage Commit area.

6. Select all files in the Unstaged Changes area by clicking first file, and then pressing **Shift** and scrolling down to select last file.
7. From the **Commit** menu, click **Stage To Commit**.
This initial process may take several moments. If a list of line errors appears when the process is complete, close the window. If the window will not close, use the Task Manager to shut down the application, and then restart Git using the existing repository. If prompted to compress, click Yes.
8. In the **Initial Commit Message** box, type a message to indicate what this commit was for, and then click **Commit**.

-----STEPS COMPLETED AFTER SAVING A CHANGE IN THE APPLICATION ARCHITECT-----

9. Each time you are ready to commit, repeat these steps inside Git.
 - a. Click **Rescan**.
Any changes you made that are different from the initial commit appear in the Unstaged Changes pane. Click on one of these items to see the code in the Git workspace.
 - b. Select the new files from the Unstaged Changes pane.
 - c. From the **Commit** menu, click **Stage To Commit**.
 - d. In the **Initial Commit Message** box, type a message to indicate what this commit was for, and then click **Commit**.

SALUTE!

What Did I...?

You can use any source control tool of your choice. In the past we have used Visual Source Safe, which is considered a pessimistic locking system because it requires you to check out each file before opening it. Only after changes are committed and complete will the system unlock the objects so that others may make changes to them.

In this task, we use Git, which is considered an optimistic locking system because it allows you to edit files and merge them when you check them back in without having to lock down all other related objects. Optimistic locking works with the SalesLogix Web development environment well because many of the changes you make to one object are related to several others. For example, when you make a change to a quick form (AccountDetails.main.quickform.xml) you also inadvertently modify another XML file (modelindex.xml).

The screenshot shows the Git GUI interface. The 'Unstaged Changes' pane on the left lists 'Model/Entity Model/SalesLogix A' and 'Model/modelindex.xml'. The main editor shows the XML content of 'modelindex.xml' with a diff view. A red line indicates a deletion of a property, and a green line indicates the addition of a new property:

```

@@ -87,11 +87,11 @@
<Properties>
  <Property name="EntityMappingType" type="EntityMappingType" />
</Properties>
<ControlDefinition>
  <Properties>
-    <Property name="Control" type="Control" />
+    <Property name="Control" type="Control" />
  </Properties>
</ControlDefinition>
</QuickFormElement>
<QuickFormElement>
  <Properties>
@@ -195,10 +195,20 @@
  </Properties>
</ControlDefinition>
</QuickFormElement>

```

After you add a repository to Git, a new .git folder appears inside the source repository directory. This folder contains several files, including a config file.

The screenshot shows a file explorer window for the directory 'C:\WFS7.5.1\.git'. The contents are as follows:

Name	Size	Type
hooks		File Folder
info		File Folder
objects		File Folder
refs		File Folder
config	1 KB	File
description	1 KB	File
HEAD	1 KB	File

The Application Architect provides very useful back-up options without requiring the use of source control. However, using source control in a single-user SalesLogix Web Development environment is useful for maintaining more robust versioning and being able to roll back changes.



#26. Use Source Control in a Multi-Developer Environment



How Do I...?

▶ Part 1: Administrator - Create the Bare Repository

1. From the **Start** menu, click **Run**.
2. In the Open box, type **cmd**, and then click **OK**.
A command prompt appears.
3. Change directories to the root of C:\

```
cd C:\
```
4. Make a new directory for a bare VFS inside of Git. This is the directory to which all of the developers will write their changes.

```
mkdir vfsingit
```
5. Change directories to the new vfsingit folder.

```
cd vfsingit
```
6. Issue the following command to initialize this new Git bare directory:

```
git init --bare
```

You should receive a message when the empty directory is initialized in Git.
7. Close the command prompt.

► Part 2: Developer 1- Create a New Directory for Local Files and Make a Remote Connection to the Bare Repository

In the training environment, you may have already completed these steps. You can skip these and just use the existing repository you set up in the previous task.

→ 1. From the **Start** menu, point to **All Programs**, point to **Git**, and then click **Git GUI**.

2. Click **Create New Repository**.

3. In the **Directory** box, click **Browse**.

4. In the Browse for Folder window, browse to the location of your working project workspace, and then click **OK**.

5. Click **Create**.

The list of Unstaged Changes appears in the top, left corner of the screen.

6. Select all files in the Unstaged Changes area by clicking first file, and then pressing **Shift** and scrolling down to select last file.

7. From the **Commit** menu, click **Stage To Commit**.

This initial process may take several moments. If a list of line errors appears when the process is complete, close the window. If the window will not close, use the Task Manager to shut down the application, and then restart Git using the existing repository.

8. In Git, click click **Remote > Add**.

The Add New Remote window appears.

9. In the **Name** box, type a name for the remote connection (Remote1).

10. In the **Location** box, type the path to the bare directory (C:\vfsingit).

11. Click the **Do Nothing Else Now** option, and then click **Add**.

12. Click **Push**.

The Push window appears, which displays what remote and branch to push to. The active Destination Repository should be remote1.

13. Click **Push**.

The files from Developer 1's repository are pushed to the bare repository.

14. Close Git.

► Part 3: Developer 2 - Create a New Directory for Local Files and Make a Remote Connection to the Bare Directory

1. From the **Start** menu, point to **All Programs**, point to **Git**, and then click **Git GUI**.

2. Click **Create New Repository**.

3. In the **Directory** box, type **C:\VFSDeveloper2**, and then click **Create**.

A new empty directory is created for testing purposes.

4. In Git, click click **Remote > Add**.

The Add New Remote window appears.

5. In the **Name** box, type a name for the remote connection (Remote2).

6. In the **Location** box, type the path to the bare directory (C:\vfsingit).

7. Click the **Do Nothing Else Now** option, and then click **Add**.

► Part 4: Pull and Push Changes

1. From the **Start** menu, click **Run**.
2. In the Open box, type **cmd**, and then click **OK**.
A command prompt appears.
3. Change directories to Developer 2 local repository.

```
cd C:\vfsdeveloper2
```

4. Issue the following command to pull files from the bare repository into Developer 2 repository.

Developer 2
pulling from the
repository →

```
git pull Remote2 master
```

Wait for a moment for files to copy.

5. Using Windows Explorer, browse to C:\vfsdeveloper2.
A Model folder should now appear that was copied from the bare repository.
6. Add a new text file inside the C:\vfsdeveloper2\Model folder (test.txt).
7. Inside Git in the vfsdeveloper2 repository, click **Rescan**.
The new file appears in the Unstaged Changes pane.
8. Click the file, and then click **Commit > Stage to Commit**.
9. Enter a comment in the Commit Message box, and then click **Commit**.
The file is committed to the local repository.

Developer 2
pushing to the
repository →

10. Click **Push**.
The Push window appears. The active Destination Repository should be remote2.

11. Click **Push**.

12. Toggle back to the command prompt, and change directories the Developer 1 local repository.

```
cd C:\VFS7.5.2
```

13. Issue the following command to pull files from the bare repository into Developer 1 repository.

Developer 1
pulling from the
repository →

```
git pull Remotel master
```

14. Using Windows Explorer, browse to C:\VFS7.5.2\Model.
The new test.txt file appears.



What Did I...?

Fetch

A “fetch” creates branch information inside the .git repository. The branch contains data about any changes, but it does not contain the actual files associated with the change.

Pull

A “pull” copies files from the bare repository into the local repository. With our current install, you can only perform a pull through the command prompt by issuing the following command:

```
git pull [RemoteName] [branch]
```

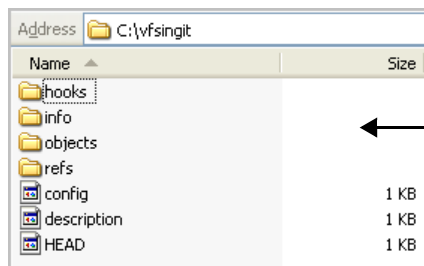
As long as you are inside a git repository directory in the Command Prompt, you can perform commit, fetch, pull, and more functions using the command prompt as well. We used the Git Gui for ease of instruction. If you want to discover what options each command uses, issue a switch after a git command (`git commit -?`) or consult the online documentation with Git.

Commit

A “commit” commits a file to your local .git repository. Commit your changes often, and only push when you know it will work with the remote repository (i.e. “commit often and push only good code”).

Push

A “push” copies all committed files from your local .git repository into the bare repository. If you browse to the bare repository using Windows Explorer, note that you cannot actually see the files you push and pull from.



The bare repository does not contain a visible copy of the actual files pushed and pulled.

It is important to fetch before working on any customizations. Otherwise, if Developer B tries to push changes to a file that Developer A has already pushed since Developer B’s last pull, Developer B would receive a non-fast forward error. In this case, Developer B should pull again and merge the changes before the next commit.