Lab: Get started with Git in VS Code

Scenario

When working with any source code, it is a good idea to manage it with a source control management tool. Visual Studio Code has built-in support for git. However, it does not have git tools automatically installed, and requires you to manually download, install, and configure git on your machine, before you can use git integration in Visual Studio Code.

In this lab, you will install and configure git, and then use git commands in Visual Studio Code to initialize a repository, connect it to a remote, track some changes, synchronize your changes with a remote, work with branches, and merge conflicts.

Objectives

- Install git
- Clone a git repository

Exercise 1: Enable Git in Visual Studio Code

Exercise Scenario

Visual Studio code supports a number of git commands. When you invoke any of these commands from Visual Studio, the Visual Studio runtime passes those commands to the local git installation on the machine. Therefore, you first must install git.

Task 1: Install Git

High Level Steps

1. Download Git 2.0.0 or newer.
2. Install Git.

Detailed Steps

1. Download Git 2.0.0 or newer.
    a. Open web browser.
    b. Navigate to https://git-scm.com/download
    c. Select download for your operating system.
    d. Save the incoming installation file to a desired location no your filesystem.
2. Install Git.
    a. Locate the file you downloaded in the previous step, and double-click it.

b. If you get a security warning, click **Run** to accept it.
c. In the **Information** page, click **Next**.
d. In the Select Destination Location page click Next.
e. In the **Select Components** page, click **Next**.
f. In the Select Start Menu Folder page, click Next.
g. In the Adjusting your PATH environment page, click Next.
h. In the Choosing HTTPS transport backend page, select the Use the native Window Secure Channel library option, and click Next.
i. In the Configuring the line ending conversions page, click Next.
j. In the Configuring the terminal emulator to use with Git bash page, select the Use Windows' default console window option, and click Next.
k. In the Configuring extra options page, click Install.
l. After the installation procedure completes, in the **Completing the Git Setup Wizard** page, unselect the **View Release Notes** checkbox, and click **Finish**.

Exercise 2: Clone a repository

Exercise Scenario

When working with source control management, you can either clone an existing online git repository to your local machine, or you can initialize a new repository and then publish all of its content in an online git server.

In this exercise, you will clone a git repository, and then open it for editing in Visual Studio code.

Task 1: Clone a Git repository

High Level Steps

1. Restart Visual Studio Code.
2. Clone a repository.

Detailed Steps

1. Restart Visual Studio Code.
   a. If Visual Studio Code is open, close it.
   b. Start Visual Studio Code.
2. Clone a repository.
   a. In Visual Studio Code, press Ctrl+Shift+P to start the command palette.
   b. Enter "Git: Clone".
   c. In the **Repository URL** prompt, enter https://github.com/CloudReadySoftware/crs-al-language-extension

d. In the **Parent Directory** prompt, after the suggested path, enter "\source\repos" and press Enter. At this point, the Source Control icon in the activity bar shows the little clock icon overlaid.

Note: *The repository is now being cloned. Make sure not to close Visual Studio Code. When the clock icon is no longer shown, proceed immediately to the next task.*

Task 2: Open a cloned repository

High Level Steps

1. Open the repository cloned in the previous task.

Detailed Steps

1. Open the repository cloned in the previous task.
   a. In the Would you like to open the cloned repository notification, click Open Repository.

Exercise 3: Initializing a repository locally

Exercise Scenario

Sometimes you will want to enable source control over an existing project. When you have already done some development locally, but that development work was not managed with SCM, you can still enable SCM over it, to start managing and tracking any future changes to a project.

When a situation such as this arises, you can initialize a Git repository locally. Afterwards, you can push that repository to a remote.

In this exercise you will initialize a Git repository over a local folder that already contains some development work. Then, you'll configure this local repository to connect to an empty remote repository.

Task 1: Initialize a local repository

High Level Steps

1. Initialize a repository over an existing demo folder.
2. Make sure that repository is initialized.
3. Open the initialized repository.

Detailed Steps

1. Initialize a repository over an existing demo folder.
   a. Open VS Code.
   b. Access the Command Palette, and then enter "Git: Initialize Repository"

c. Navigate to **C:\Users\student\Documents\AL\Samples\GetAddressService**.

d. Click **Initialize Repository**.

2. Make sure that repository is initialized.

a. Access the Source Control pane (Ctrl+Shift+G)

b. Under the **Source Control Providers** group, locate **GetAddressService Git** provider. It should indicate that there are 8 uncommitted changes in this repository.

3. Open the initialized repository.

a. Click File > Open Folder.

b. Navigate to **C:\Users\student\Documents\AL\Samples\GetAddressService**.

c. Click **Select Folder**. The repository is opened in the current VS Code instance.

d. Make sure that the Source Control icon in the activity bar indicates 8 uncommitted changes.

Task 2: Perform initial commit of changes

High Level Steps

1. Commit changes to repository.

Detailed Steps

1. Commit changes to repository.

a. Access Source Control pane (Ctrl+Shift+G).

b. In the **Message** textbox, enter "Initial commit".

c. Press the checkmark icon next to the provider name on the top of the Source Control pane.

Task 3: Connect the local repository to a remote

High Level Steps

1. Use terminal to connect your local repository to a remote.

Detailed Steps

1. Use terminal to connect your local repository to a remote.

a. Click View > Integrated Terminal.

b. In the terminal pane, enter git remote add origin https://your_git_remote_url

Note: *For this task to work, you must have an account in a Git service provider accessible over Internet. Examples are GitHub, or Visual Studio Team Services. If you don't have that, create a personal account on GitHub.com (it's free)*

and then create a new blank repository. Alternatively, you can use URL git remote add origin https://github.com/vjekob/test4.git but then you will not be able to do any pushes.

This step is not absolutely mandatory. Remaining exercises in this lab can mostly be done without having a working remote.

Exercise 4: Working with branches

Exercise Scenario

In this exercise you will see how branches work and how to create branches, switch between branches, and merge branches.

Task 1: Create a branch

High Level Steps

1. Create a branch over the existing repository.
2. Verify that there is a new branch and that it is selected as current.

Detailed Steps

1. Create a branch over the existing repository.
   a. Access Command Palette (Ctrl+Shift+P);
   b. Enter "Git: Create Branch…" A prompt will ask for the branch name.
   c. Enter "new feature". The new branch named "new-feature" will be created, and selected as a current branch.
2. Verify that there is a new branch and that it is selected as current.
   a. Check the leftmost piece of information in the status bar. It should indicate a source control icon, and show "new-feature" branch.
   b. Click the "new-feature" branch in the status bar. The prompt opens with the list of branches.
   c. Select "master".
   d. Access Command Palette.
   e. Enter "Git: Checkout to…" The same prompt opens with the list of branches.
   f. Select "new-feature".

Note: *All these are different ways of checking if there is a specific branch and to switch the current context to a specific branch. This is called "checkout", and you are always checking out to a specific branch, hence "Git: Checkout to…"*

Task 2: Make changes in the code

High Level Steps

1. Open AddressService.al codeunit file.
2. Make changes in code.
3. Review the changes before committing them.
4. Stage changes.
5. Commit changes.

Detailed Steps

1. Open AddressService.al codeunit file.
   a. Click **Go > Go to file**. Alternatively, press Ctrl+P.
   b. Start typing "AddressService.al". As soon as you start typing the list of files satisfying the search criteria will show.
   c. When AddressService.al is listed, use arrow up and down keys to select it.
   d. With AddressService.al selected, press the right arrow key to open it in the editor.
   e. With AddressService.al opened in the editor, press Esc to close the Go to file prompt.
2. Make changes in code.
   a. At the beginning of the codeunit, locate the YesPleaseTxt constant declaration.
   b. Change the declaration as follows:

```
YesPleaseTxt : TextConst ENU='Yes, please.';
```

Note: *The change is very simple in this example, it merely adds a full stop at the end of the constant. The change itself does not matter, any change is good enough so feel free to introduce any arbitrary change to the code as you see fit.*

3. Review the changes before committing them.
   a. Access the Source Control pane (Ctrl+Shift+G)
   b. Locate the AddressService.al file in the list of changed files.
   c. Select the AddressService.al file. Even though you already had AddressService.al file open in editor, a new editor tab opens which has a left and a right pane. Left pane indicates the last committed state in the current branch, and the right pane indicates the current edited, uncommitted state of the file.
   d. Introduce some arbitrary change in the code in the right pane. You can perform editing operations in the file comparison editor, but only in the right pane. The left pane is read only.

Note: *Either introduce additional changes, or no changes at all. Just make sure not to remove the changes you did in the previous step, because that will make the branch non-dirty again, and there won't be anything to commit.*

4. Stage changes.
    a. In the Source Control pane, locate the AddressService.al file.
    b. To the right of the file name, click the plus (+) icon. The tooltip over this icon says "Stage Changes". After you click it, the new group called "Staged Changes" appears and the AddressService.al file is included in it.

Note: *Staging makes changes known to Git, but does not permanently commit them to a repository. Staging changes is not mandatory, but it helps you distinguish between files you are done changing, and files you are still working on. To make changes permanent, you must commit them.*

5. Commit changes.
    a. Access the Command Palette.
    b. Enter "Git: Commit."
    c. In the prompt, enter "Minor changes".

Note: *This commits the changes to the repositories. The changes are now permanent, and the current branch is clean (non-dirty), and this is indicated with no asterisk next to the branch name in the status bar. Branches must be clean before merging them.*

Task 3: Merge changes

High Level Steps

1. Change context to master branch.
2. Verify that the AddressService.al file does not contain your changes.
3. Merge content from new-feature branch onto the master branch.

Detailed Steps

1. Change context to master branch.
    a. Access the Command Palette.
    b. Enter "Git: Checkout to…", or alternatively click the "new-feature" branch in the status bar.
    c. In the prompt, select the master branch and press Enter.

Note: *You should normally do changes in branches other than master, and then merge changes into the master branch once the work is done. Since you are*

merging onto the current branch, not from the current branch onto another branch, you must be in the context of the branch you are merging into before you run the merge command.

2. Verify that the AddressService.al file does not contain your changes.
   a. Access the AddressService.al file.
   b. Locate the YesPleaseTxt constant.
   c. Verify that it does not include the full stop at the end of the string contents.

Note: *If you introduced other changes, you should notice that you don't see them. Your changes are committed to the new-feature branch, and since you are in the master branch, those changes are not visible.*

3. Merge content from new-feature branch onto the master branch.
   a. Access the Command Palette.
   b. Enter "Git: Merge Branch…" The prompt lists the branches available to merge from.
   c. Select "new-feature" and press Enter.
   d. Verify the AddressService.al file to make sure that the changes you committed to the new-feature branch are now applied to the master branch.

Note: *Really, that's all there is to merging. If there are no conflicts, merge is a simple operation and it results in your target branch containing the changes done to a new branch.*

Task 4: Handle conflicts

High Level Steps

1. Create a new branch.
2. Introduce changes in the "further-changes" branch.
3. Introduce conflicting changes in the "new-feature" branch.
4. Access the master branch and merge from "further-changes" branch.
5. Merge from "new-feature" branch.
6. Solve the conflict.
7. Commit to repository

Detailed Steps

1. Create a new branch.
   a. Follow the steps given above in the **Create a branch** task to create a branch named "further-changes" from the master branch.

2. Introduce changes in the "further-changes" branch.
    a. In the "further-changes" branch (indicated in the status bar), open file Addresses.al.
    b. In the page, change the InsertAllowed and DeleteAllowed properties to True.
    c. Save the file. The "further-changes" branch is now indicated as dirty.
    d. Commit the branch to the repository.
3. Introduce conflicting changes in the "new-feature" branch.
    a. Click the "further-changes" branch name in the status bar, and then check out to "new-feature" branch by selecting it in the list of available branches indicated in the prompt.
    b. Access the Addresses.al page. Notice that InsertAllowed and DeleteAllowed are both False. This is because you introduced your change in a different branch.
    c. Change the ModifyAllowed property to True.
    d. Save the file.
    e. Commit the branch.
4. Access the master branch and merge from "further-changes" branch.
    a. Access the master branch.
    b. Access Command Palette and enter "Git: Merge Branch..."
    c. Select "further-changes" and press Enter. The changes are now merged into the master branch.
5. Merge from "new-feature" branch.
    a. Still in the master branch, access Command Palette, and enter "Git: Merge Branch..." again.
    b. Select "new-feature" and press Enter. A warning is shown letting you know that there is a conflict in the code. Also, the Addresses.al file is open and the conflict is clearly indicated.
6. Solve the conflict.
    a. In the Addresses.al file, click the "Accept Both Changes" action above the conflict indicator. This leaves the changes from both commits. Since this results in invalid AL, you must fix the problem.
    b. In the list of properties, remove any duplicates of InsertAllowed, DeleteAllowed, and ModifyAllowed. You should retain those that have the properties set to True, and remove those where they are set to False.
7. Commit to repository
    a. In the Source Control pane, make sure that the Merge Changes group is visible, and it indicates the Addresses.al file. When there are merge conflicts, the conflicting files are shown here.
    b. Stage changes, then commit changes.