

Get Started with ProMark

Part 2: Using ProMark

Nicholas Leuci



Before You Begin

This slide deck is the follow-on module from the first slide deck, **'InstallingProMarkWithGitHub.pptx'**

You need to complete ALL the installation tasks in that slide deck BEFORE doing the setup and configuration tasks described here.

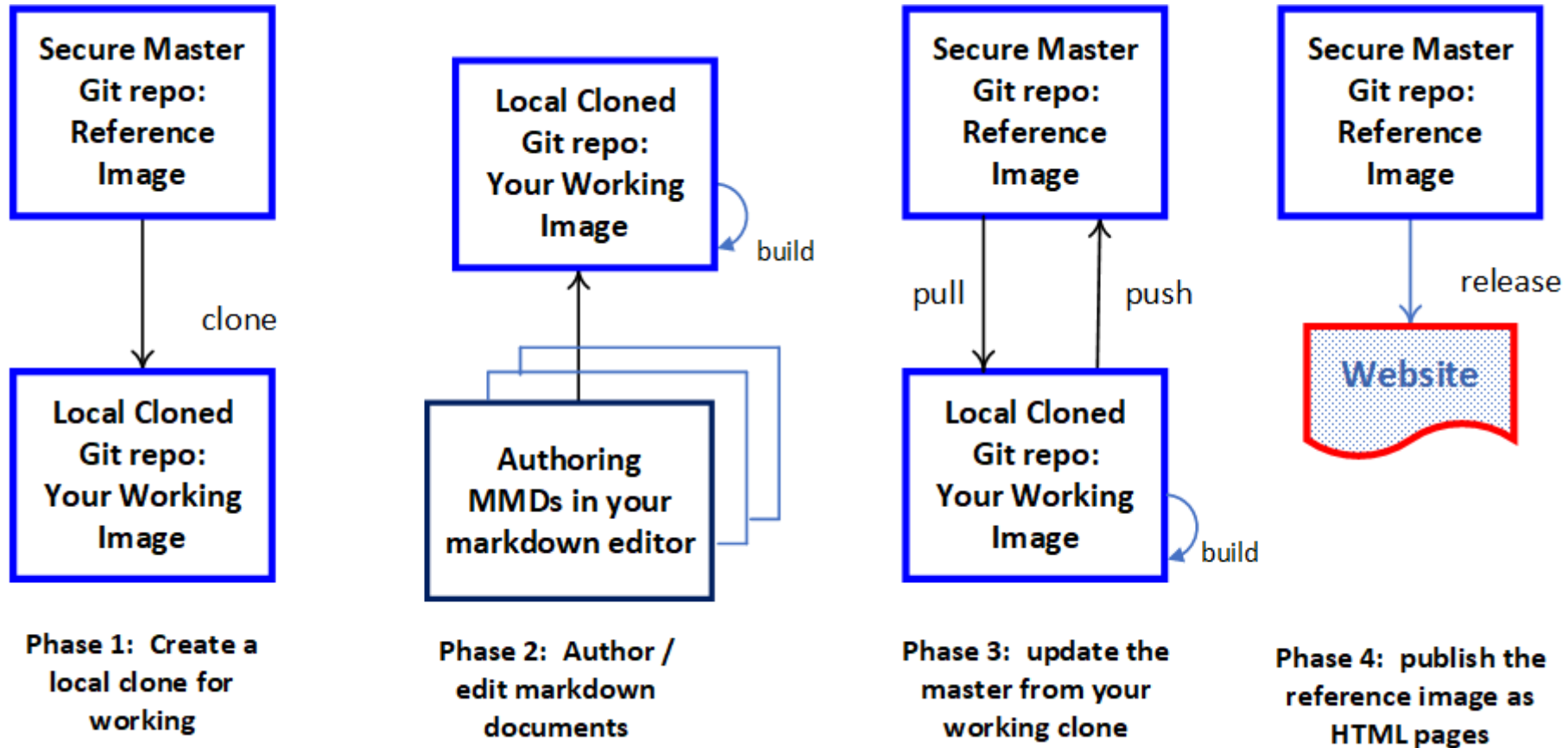
These **prerequisites** must be done and available:

- A completed Git installation, with the **Bash** shell (terminal)
- A working GitHub login account
- A current version of Visual Studio Code installed
- Ready access to the ProMark [PM DOC](#) web site

What's covered in this course

- [Cloning a Repo](#)
- [Converting a Word file to MultiMarkdown using ProMark](#)
- [Editing File Contents](#)
- [Building HTML Files](#)
- [Pushing Local Content to the Repo](#)

The Cloning Paradigm




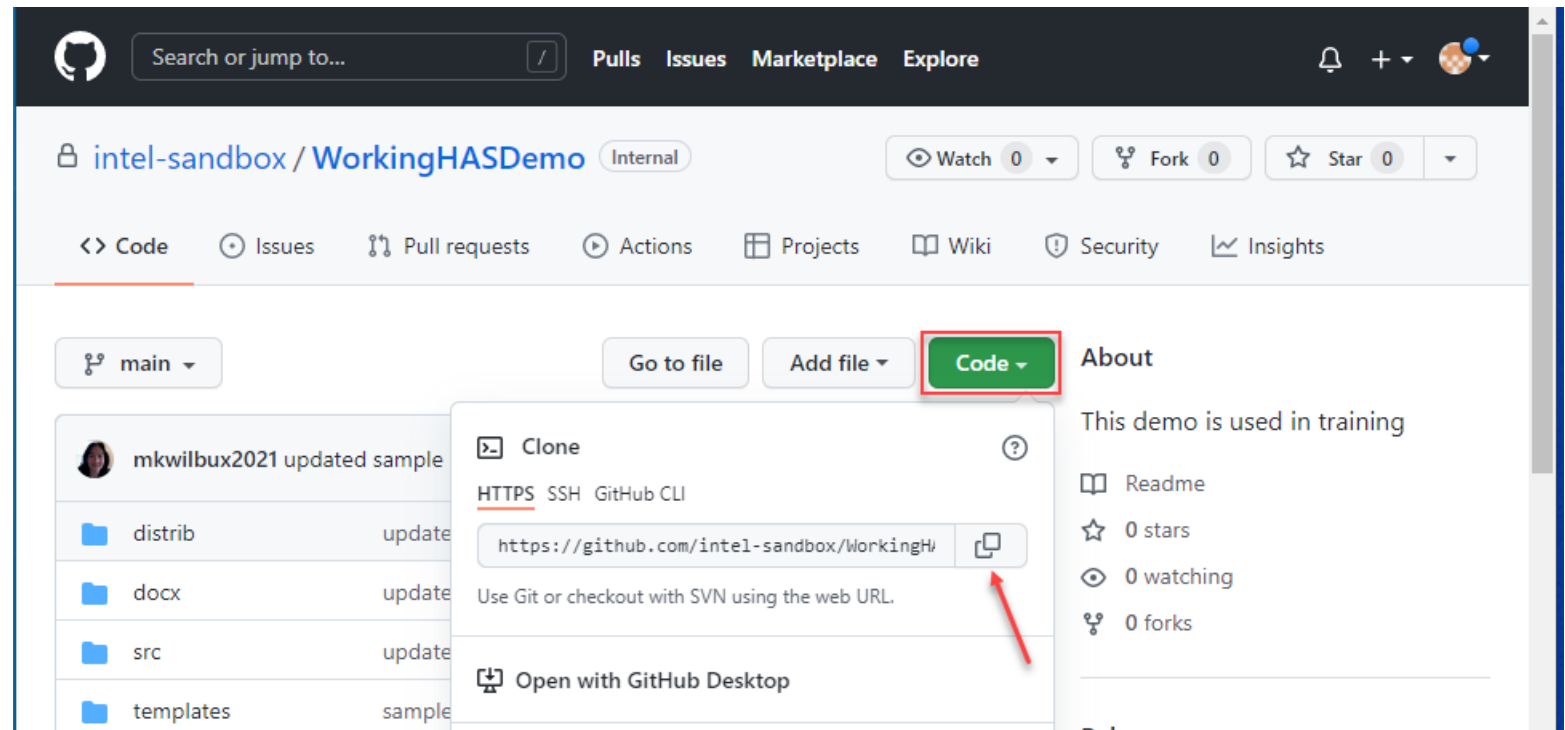
As an author, you work on a cloned image of a master repository (repo). You NEVER work directly on the master. You pull, create and edit documents, do local builds, and then push the documents to the master to synchronize the repo images. An admin / author does a release to publish the master to the website.

Clone a Repo: Use a URL to copy a GitHub repo

A Git repo is available for ProMark training uses on an Intel GitHub site: the 'WorkingHASDemo'. There are several ways to clone a repo. This example uses the HTTPS URL cloning method to select and copy the URL of a remote repo to your local PC. The cloned repo is a mirror image copy on your PC, allowing you to examine and work on the repo locally without affecting the source repo.

You will be required to sign-in to your GitHub account to access the remote repo.

1. Copy this link to your browser and go there:
<https://github.com/intel-sandbox/WorkingHASDemo>
2. Click **Code**:
3. Go to **Clone > HTTPS**
4. Click  to copy the HTTPS URL to your PC's clipboard.
5. The URL link is '**Copied**'

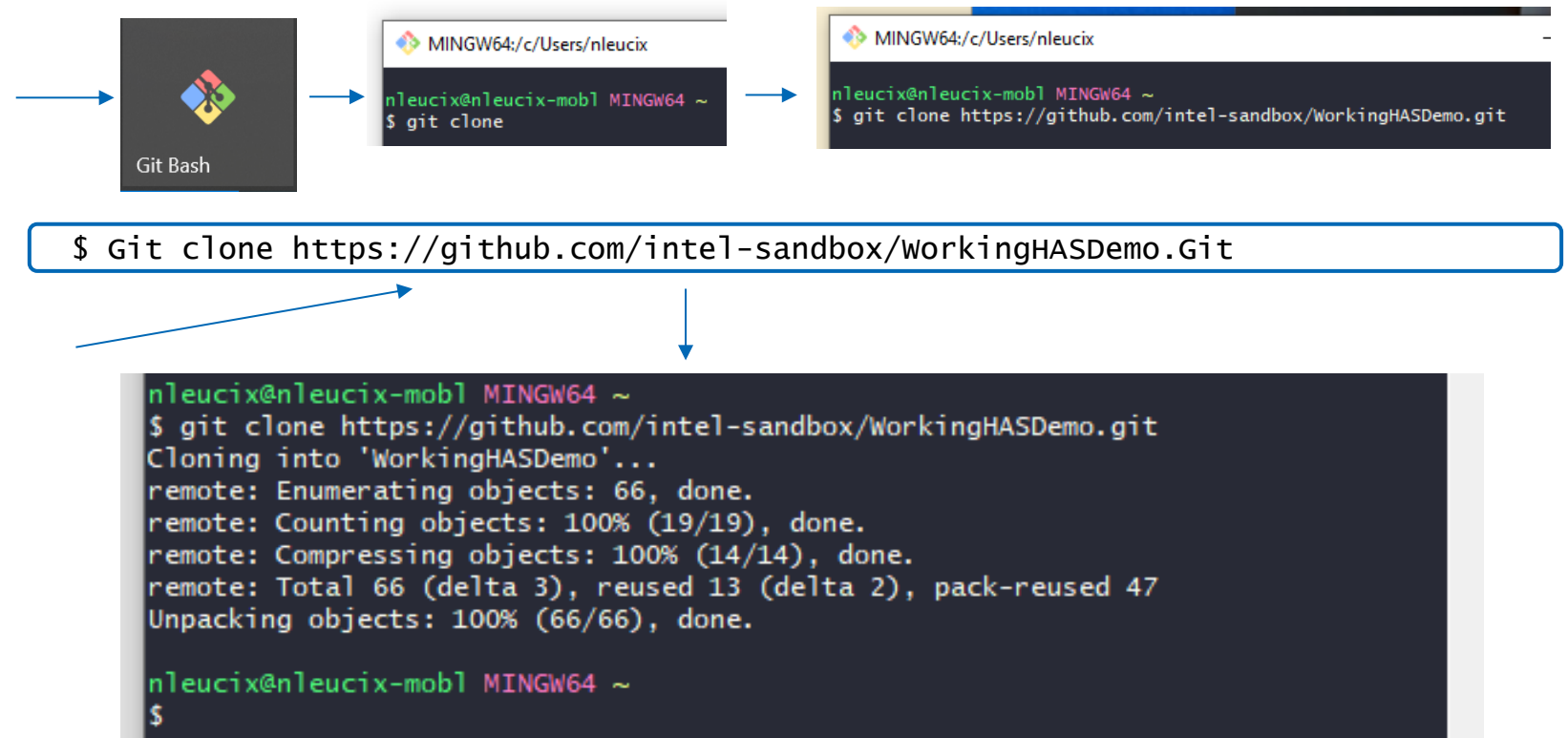


Clone a Repo: Using Git Bash

The Git Bash terminal (or shell) is a command-line utility to perform computer operations. It does take input from the keyboard, but it also responds to mouse events (like button clicks) as well. It is like the MS PowerShell. However, the Git Bash implementation comes with 'hooks' (interfaces) pre-installed to communicate with Git, which makes it very useful for operating on Git / GitHub repos.

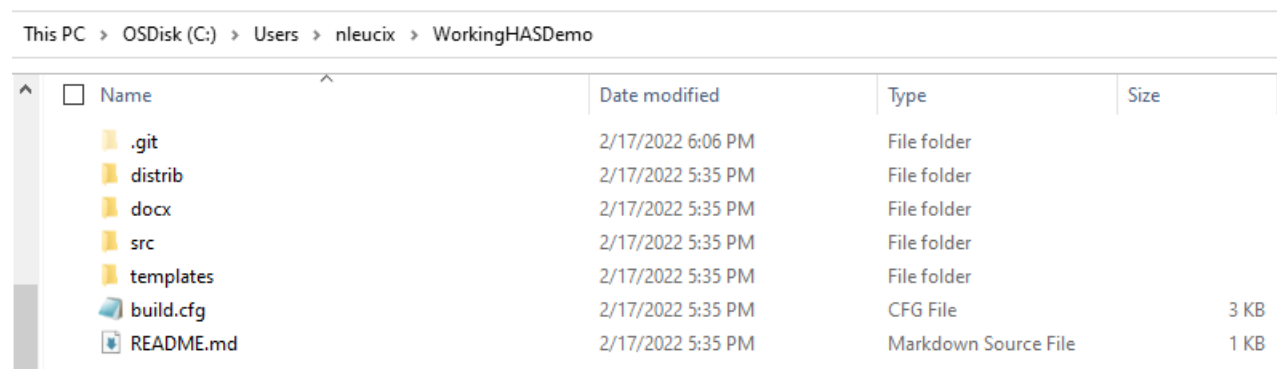
Use the Git Bash terminal to run Git and other commands.

1. Open Git Bash
2. Type: **Git clone**
3. After the entered text **right click** to paste the HTTPS URL from the PC's clipboard to Bash (copied in the previous slide)
4. Press **Enter**
5. The repo is downloaded locally to your machine and your own local image of the repo is created.



Your Local Repo

Your PC now has a local image of the cloned Git repo. You can specify any valid repo location on your PC. Git defaults to the location associated with your account when Git was installed. Typically, this is the path to your Windows user settings: 'C:\Users*youraccountname*\WorkingHASDemo\' , as shown here:



This PC > OSDisk (C:) > Users > nleucix > WorkingHASDemo				
<input type="checkbox"/> Name	Date modified	Type	Size	
.git	2/17/2022 6:06 PM	File folder		
distrib	2/17/2022 5:35 PM	File folder		
docx	2/17/2022 5:35 PM	File folder		
src	2/17/2022 5:35 PM	File folder		
templates	2/17/2022 5:35 PM	File folder		
build.cfg	2/17/2022 5:35 PM	CFG File	3 KB	
README.md	2/17/2022 5:35 PM	Markdown Source File	1 KB	

The cloned repo is a hierarchical tree of files and settings. Your local work can be done here, using Git Bash and Visual Studio Code. However, do not alter this tree's organization from Windows, which must match the originating repo's organization. If you add / change / remove files or directories, you must use Bash and Visual Studio Code to synchronize these tree changes with the originating master repo.

This presentation will show you how to setup a local working repo in any valid location.

Markdown Authoring On Your Local Repo

You can operate directly on your local repo's tree of files. This is NOT recommended. ProMark has been designed and developed to do this. However, ProMark is a publishing tool, not an authoring tool.

The best way to work on your repo is to use a robust authoring tool with ProMark. This authoring tool should support **Markdown** and **MultiMarkdown (MMD)**. ProMark supports both. Markdown is an opensource specification for creating web pages using only text with no HTML tags. Markdown content is authored and maintained as text meant to be easily rendered as HTML pages.

The paradigm is: **Text** as source content (**markdown**) → viewed as rendered **HTML** pages (**markup**).

Because markdown is a specification and not an adopted standard, there are some differences in the way editors and browsers support markdown. There is a fundamental core that most markdown tools support. Use this link to see and understand these basic core markdown features:

<https://www.markdownguide.org>

MultiMarkdown is a superset enhancement of markdown with advanced features such as tables, citations, and footnotes, and non-HTML features like 'smart' typography. These added features all support electronic authoring and publishing. Click here for more information on [MMD](#).

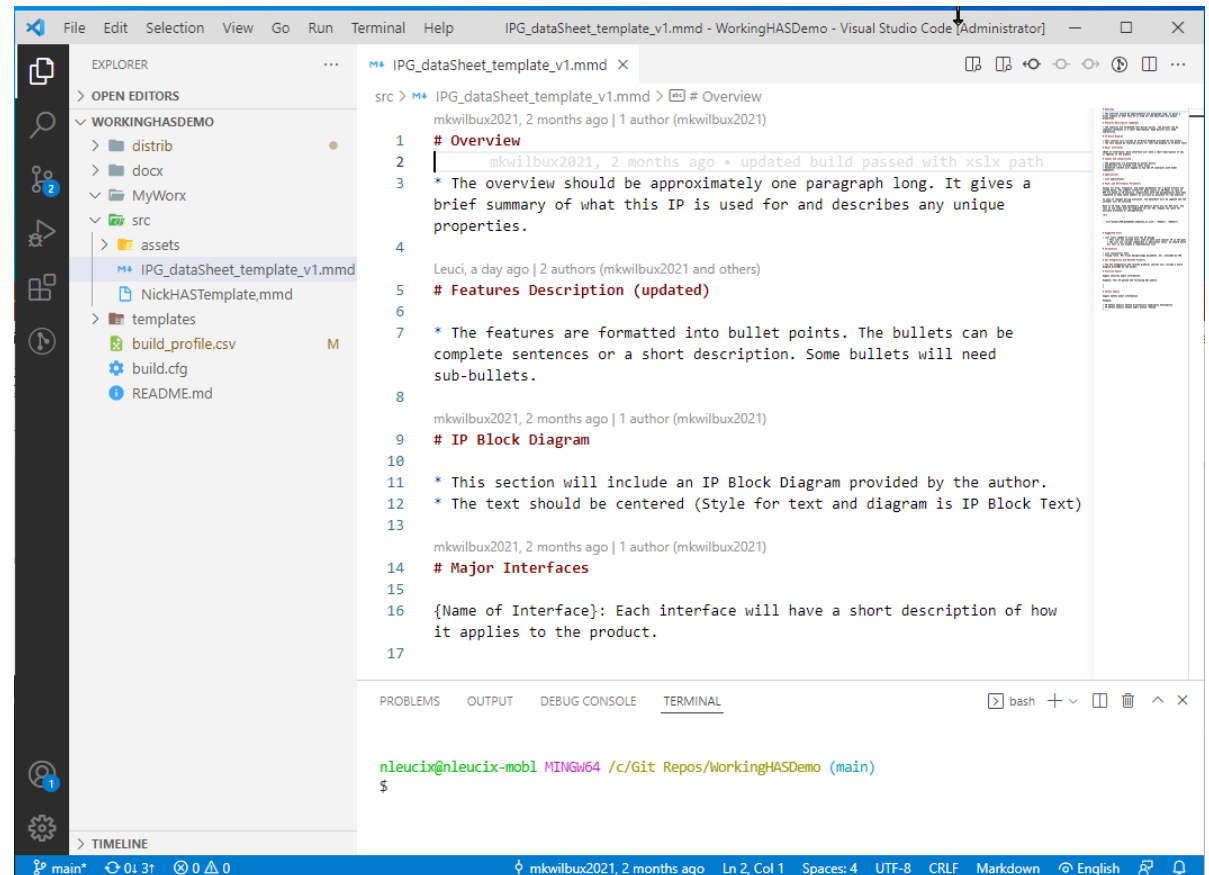
Visual Studio Code for ProMark Authoring

You can use any markdown / MMD editor with ProMark. ProMark recommends Visual Studio Code (aka VS Code or VSC). This is an opensource tool from Microsoft, and it is widely used with many extensions including MMD. It also has built-in support for working with Git / GitHub repos. Launch VS Code now.

Here is an example of Visual Studio Code after it was connected to a repo, with an MMD file open for editing. Note these key features:

- The Menu Bar in the top part, with some info
- The Side Panel on the left side, showing the repo's file hierarchy
- The Tabbed Editing Pane, open for work
- The Thumbnail Preview on the right pane
- The Terminal Panel in the lower part
- The Status Bar on the bottom part

Note that you can set your viewing preferences using the menu **File:Preferences:Color Theme** settings. A **Light** or **Dark** setting will not alter the content, only how it is shown in the tool.



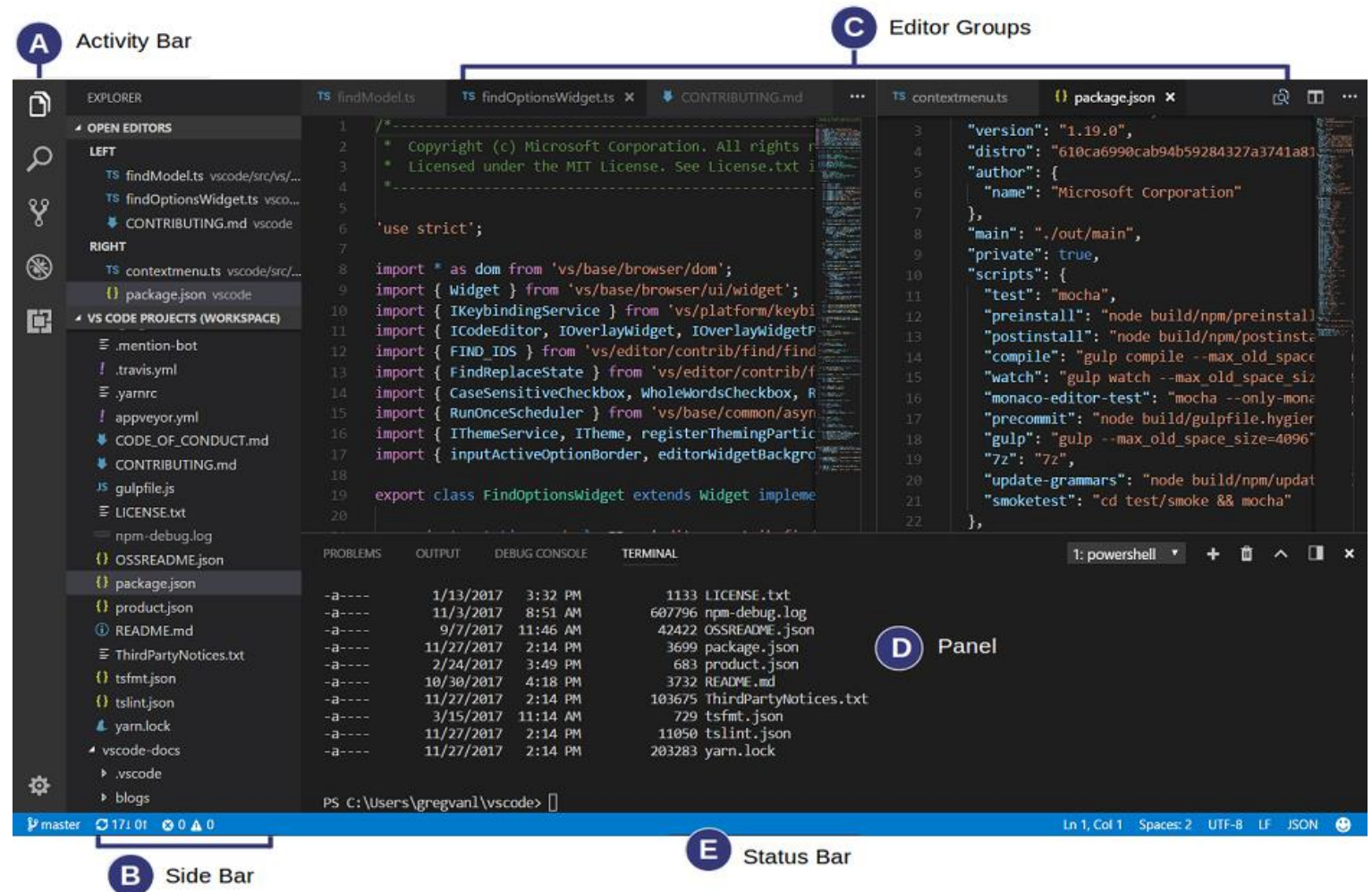
Visual Studio Code: Dark Preference

Here is an image of VS Code used with a **Dark** preference. Compare this to the **Light** preference in the previous slide

The **File:Preferences:Color Theme** only affects the appearance of the tool, not its functionality.

When you first launch VS Code you need to try out the Light vs. Dark palettes available to you. The tool will remember the setting, and you can change it any time.

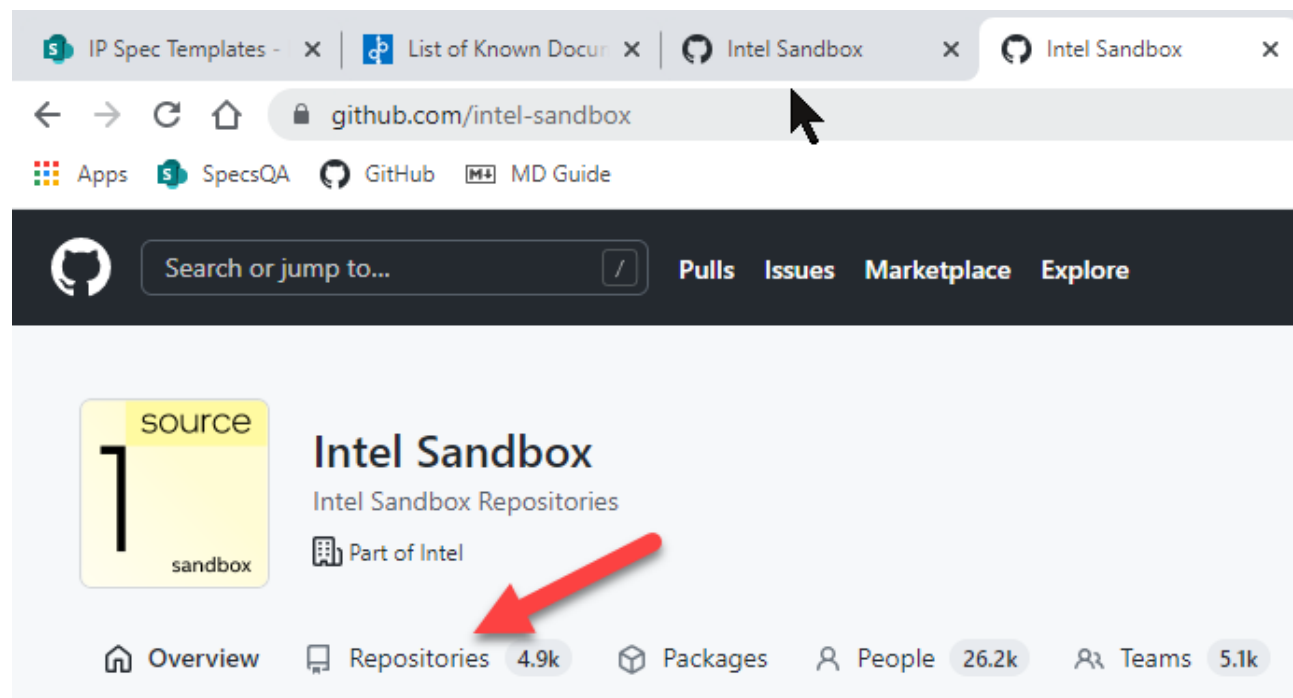
There are also external color theme extensions available for VS Code providing more specific settings to suit your needs and tastes.



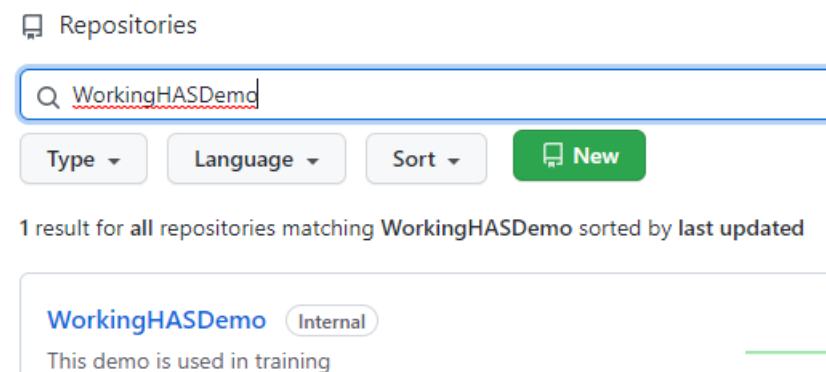
Create a Workspace to Clone a Git Repo

If you download a repo from GitHub using default settings, your local repo will be created relative to your Window's 'user' settings. This is not an optimal working environment. A better approach is to create a dedicated working area for your repos.


In the following slides, a workspace was created in drive C: at the root named 'Git Repos'. The path to this location is 'C:\Git Repos'. To follow this example, you need to use your valid Intel Git Hub login account.

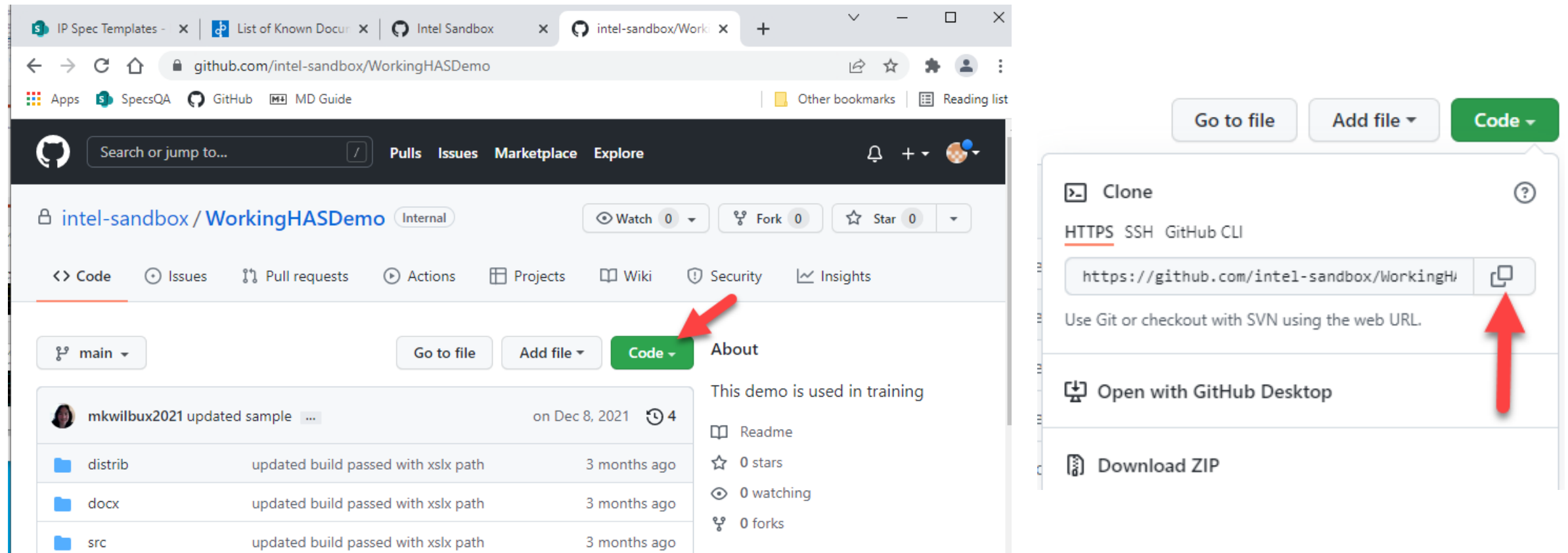


First, navigate to your [Git Hub Intel Innersource page](#). Click on **Repositories**. In the search tool, enter the name of the repo you want to clone and click on it. We will use the previous repo 'WorkingHASDemo':



Cloning A Git Repo

Click the green **Code** button. A pop-up '**Clone**' window will come up. Click on the 'copy' icon. s will copy the URL repo to your Windows clipboard. This is the ProMark preferred cloning method.




The screenshot shows a web browser with multiple tabs, including 'IP Spec Templates', 'List of Known Docu...', 'Intel Sandbox', and 'intel-sandbox/Worki...'. The active tab displays the GitHub repository page for 'intel-sandbox / WorkingHASDemo'. The repository is marked as 'Internal' and has 0 watches, 0 forks, and 0 stars. The 'Code' button is highlighted with a red arrow. A 'Clone' pop-up window is open, showing the repository URL 'https://github.com/intel-sandbox/WorkingH/'. The 'copy' icon next to the URL is also highlighted with a red arrow. The pop-up window includes options for cloning via HTTPS, SSH, or GitHub CLI, and buttons for 'Go to file', 'Add file', and 'Code'. Other options in the pop-up include 'Open with GitHub Desktop' and 'Download ZIP'.


Go to file Add file **Code**


Clone

HTTPS SSH GitHub CLI

`https://github.com/intel-sandbox/WorkingH/` 

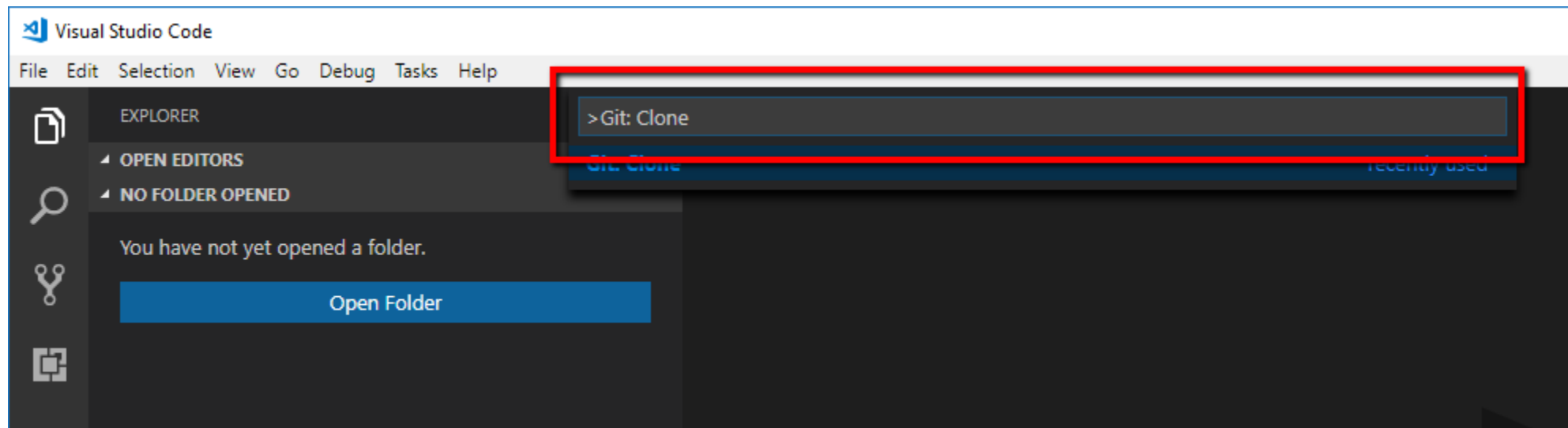
Use Git or checkout with SVN using the web URL.

 Open with GitHub Desktop

 Download ZIP

Cloning A Git Repo with Visual Studio Code

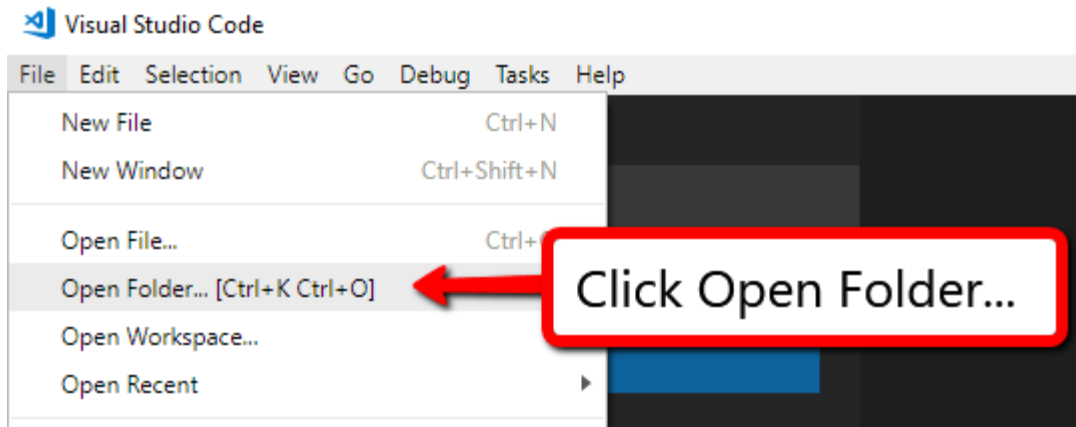
Launch Visual Studio Code. Use the '**Command Palette**' to complete the repo cloning. The command palette can be brought up in two ways: either by hitting **F1** or by clicking **View -> Command Palette...** from the menu options. Once the command palette is open, type **Git Clone** in the box and then hit **Enter**.



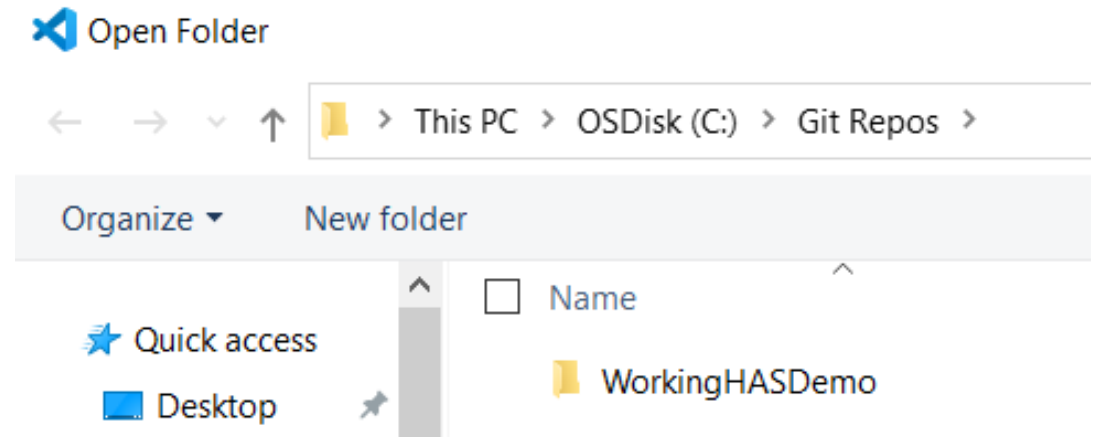
You will now be prompted to enter the repository URL that you copied from the previous step: 'https://github.com/intel-sandbox/WorkingHASDemo.git'. Paste in this string and then hit **Enter**. A folder selection box will pop-up asking you to specify where to store your local copy of the repo. Navigate to your work area and then click **Select Repository Location** at the bottom of the pop-up.

Cloning A Git Repo with Visual Studio Code 2

Open the folder that you used to clone your repo in the previous slide. From the menu bar, click **File -> Open Folder...**

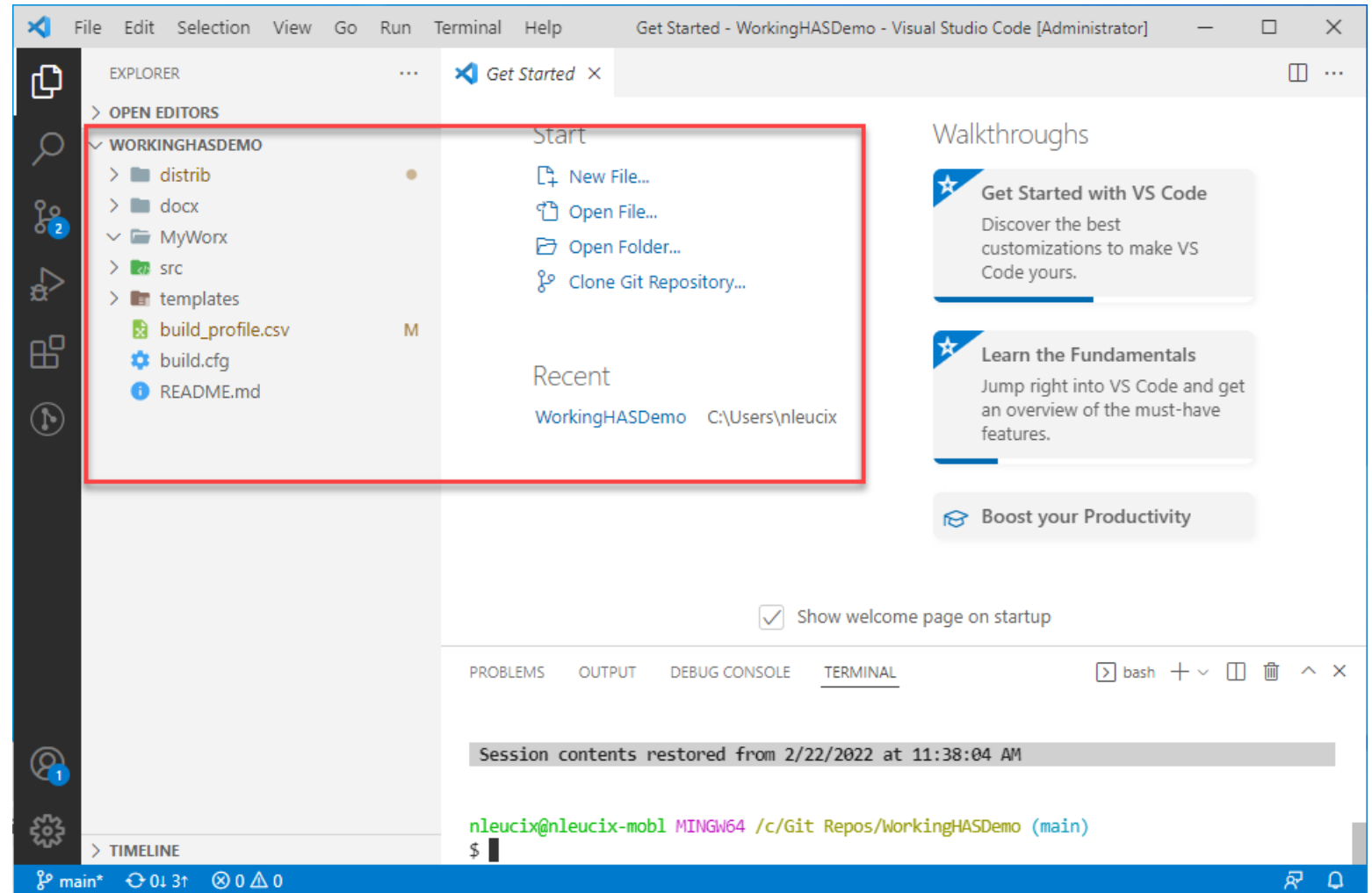
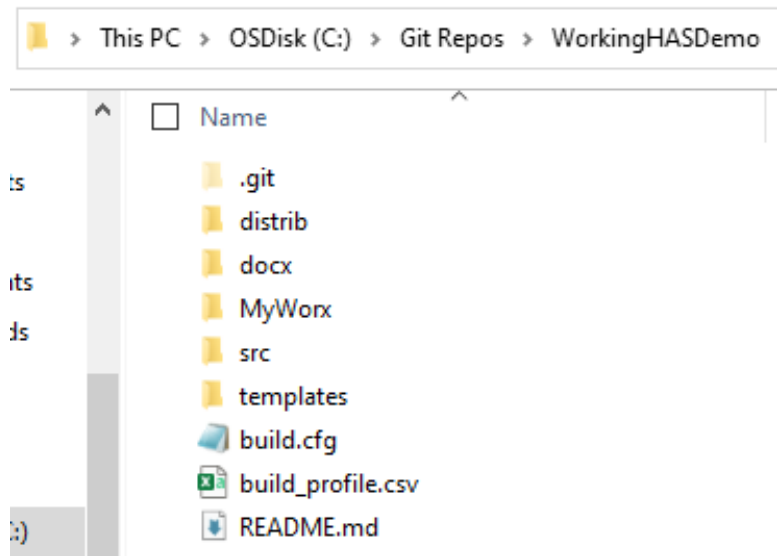


This will load the repo you cloned to your local work area into Visual Studio Code, ready for you to work.



Cloning A Git Repo with Visual Studio Code 3

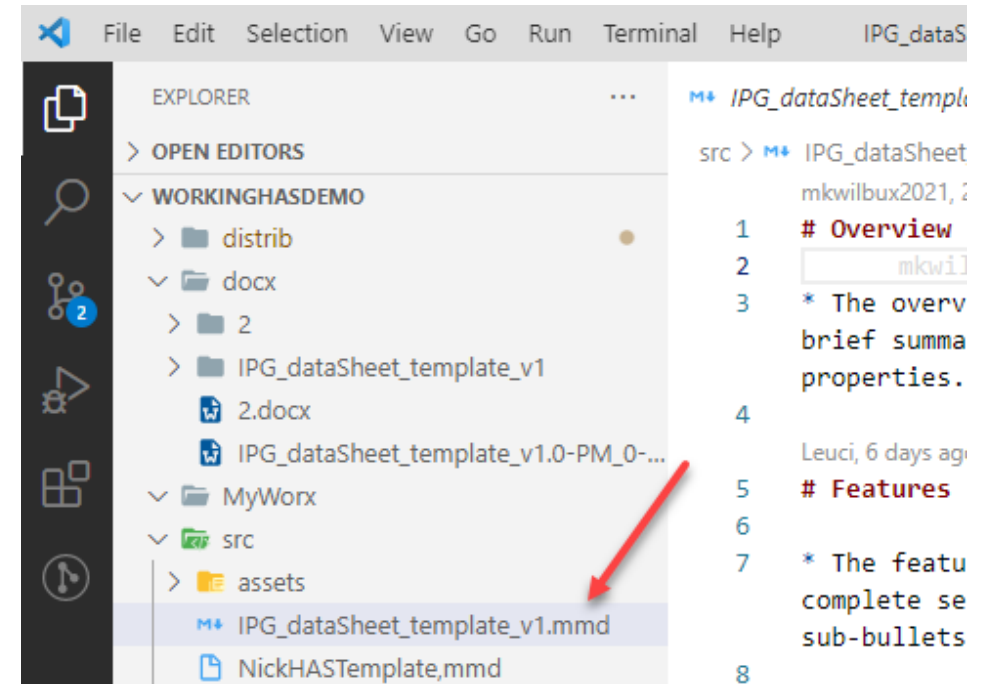
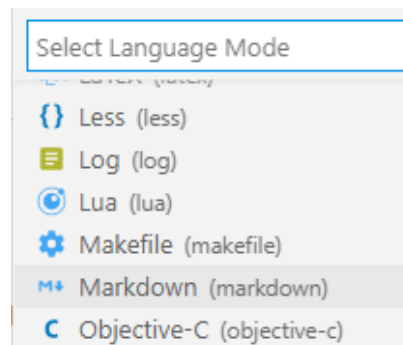
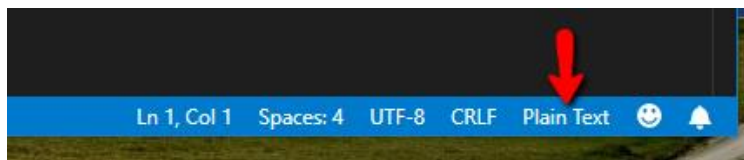
The Explorer pane shows the contents of the repo you just opened. This is derived from your PC's repo work area.



Associate *.mmd Files to Markdown

Navigate and click on the 'src' directory to expand its file list. Click on the first '*.mmd' file. This is the file extension for a markdown file. Click on this file and it will open in the first tabbed editing panel.

Initially, Visual Studio Code does not have a default handling association with a '*.mmd' file. It defaults to text. In the right corner of the status bar, it shows the file type in the opened editor: 'Plain Text'. Click on this indicator to **Select Language Mode** for the .mmd file. A list will pop-up. Scroll down the list and click on **Markdown**.

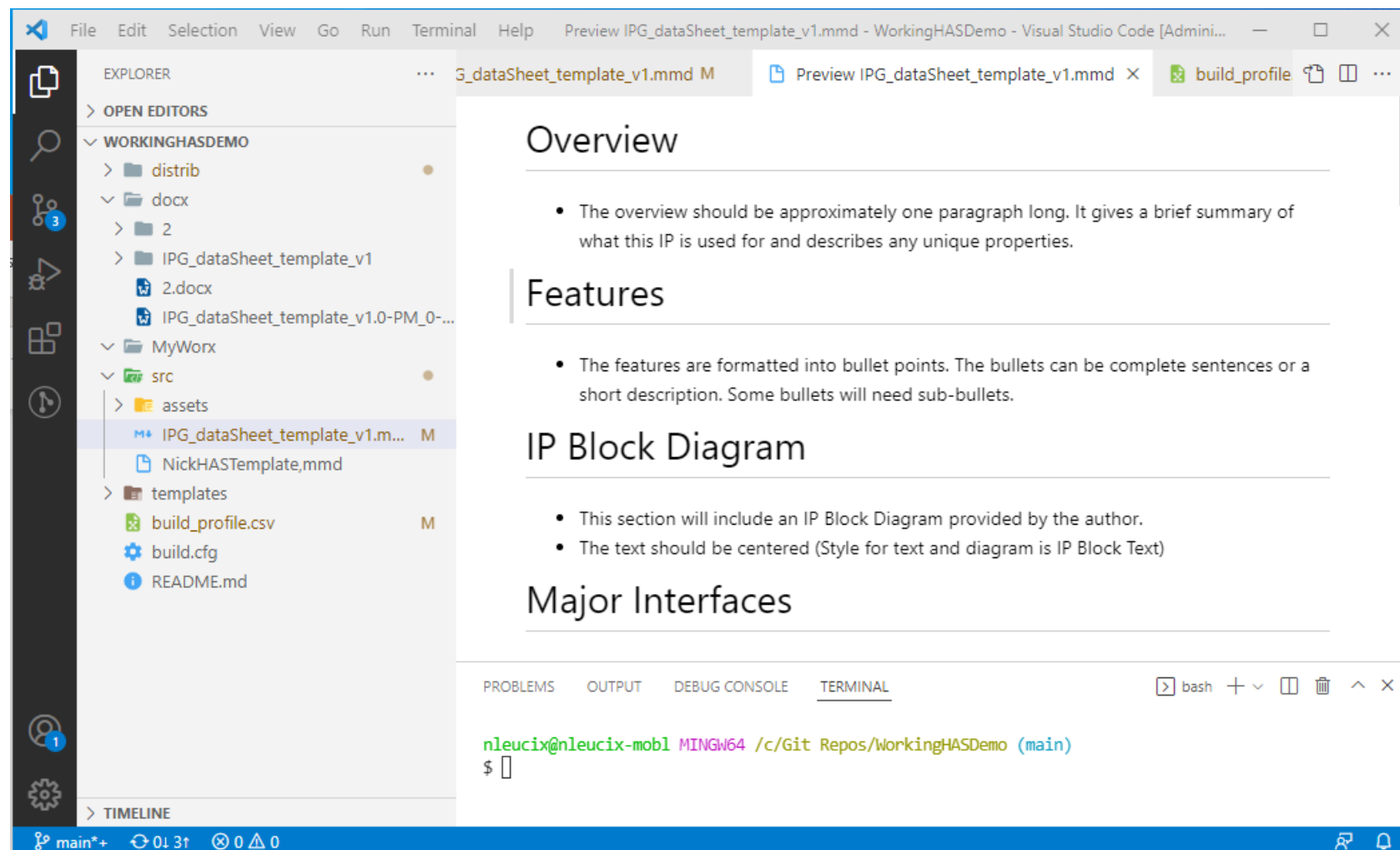


Visual Studio Code now recognizes the file as a Markdown file



Preview Markdown in Visual Studio Code

Visual Studio Code allows you to edit the text of a Markdown file, and to then **preview** how it would render as an HTML page. Now that you have associated the .mmd extension as a Markdown file, Visual Studio Code can preview it. Right click on the tab's file name and select **Open Preview** in the pop-up menu, or enter the key combination **CTRL+SHIFT+V**. The editor is now previewing (rendering) in a new tab. No editing can be done in a preview tab.

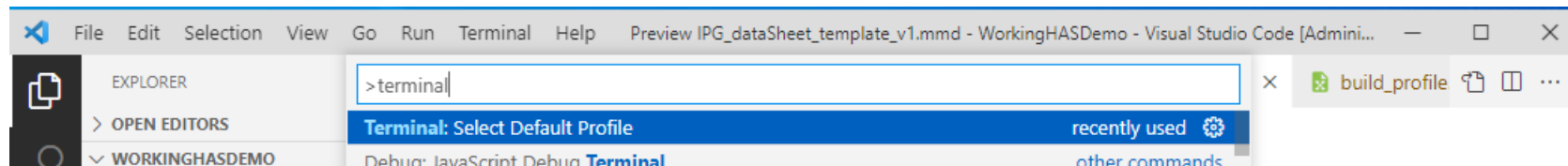


Git Bash in Visual Studio Code

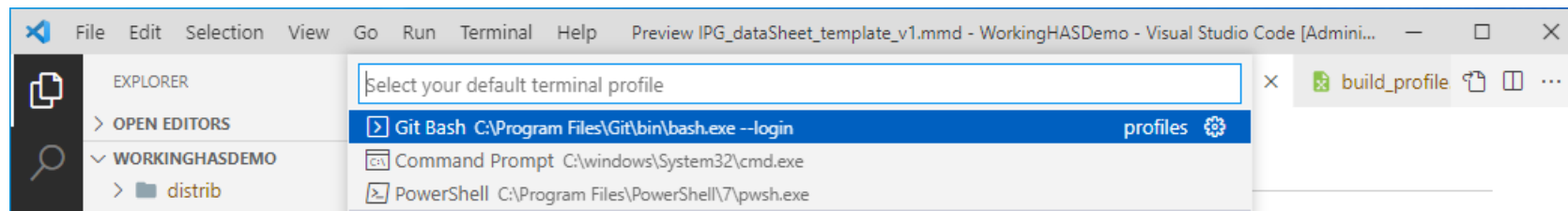
The Bash terminal (shell) is the preferred method to operate on a Git repo, as it has built-in Git hooks. Visual Studio Code has an integrated terminal to interact with your OS or repo.

The terminal is a command-line tool that enables communication with the OS and your repo. By default, this terminal operates as a PowerShell. You can change this setting to operate as a Git Bash terminal in Visual Studio Code.

1. Open the Command Palette (press F1 or Ctrl+Shift+P) and start to type **Terminal: Select Default Profile**. When this option is displayed, click it.



2. Select **Git Bash** from the drop-down option list



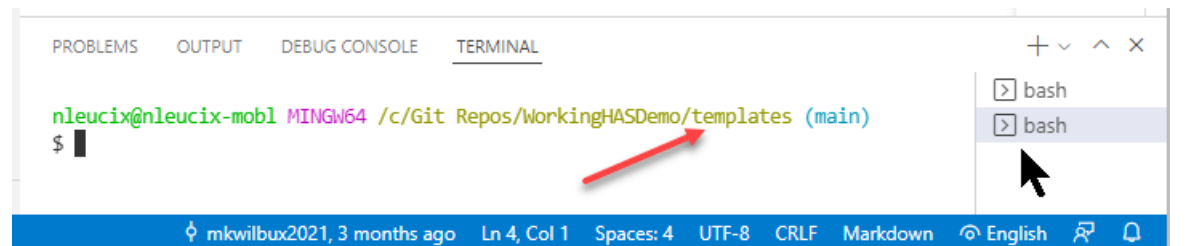
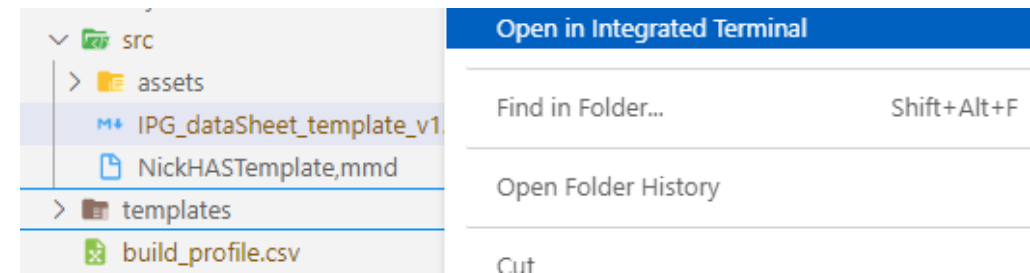
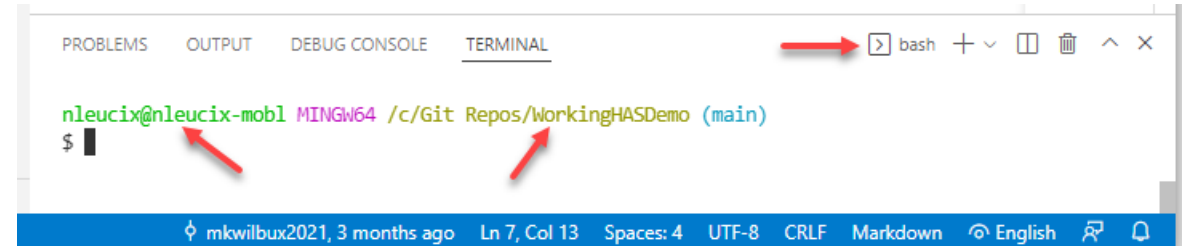
Use Git Bash To Navigate

The terminal area of Visual Studio Code now has the Bash '\$' prompt and displays your linked name. Adjacent to your name is your current path in Visual Studio. The word '**(main)**' shows that your path is at the root of your local repo.

Notice the '**bash**' terminal indicator in the upper right corner of the terminal area. When this area has the focus, you can type commands at the '\$'.

You can use Visual Studio Code to navigate through the repo, and you can easily open multiple Bash terminals while working in your repo. Use your mouse to find an area in the repo. Right click on the name and select **Open In Integrated Terminal**.

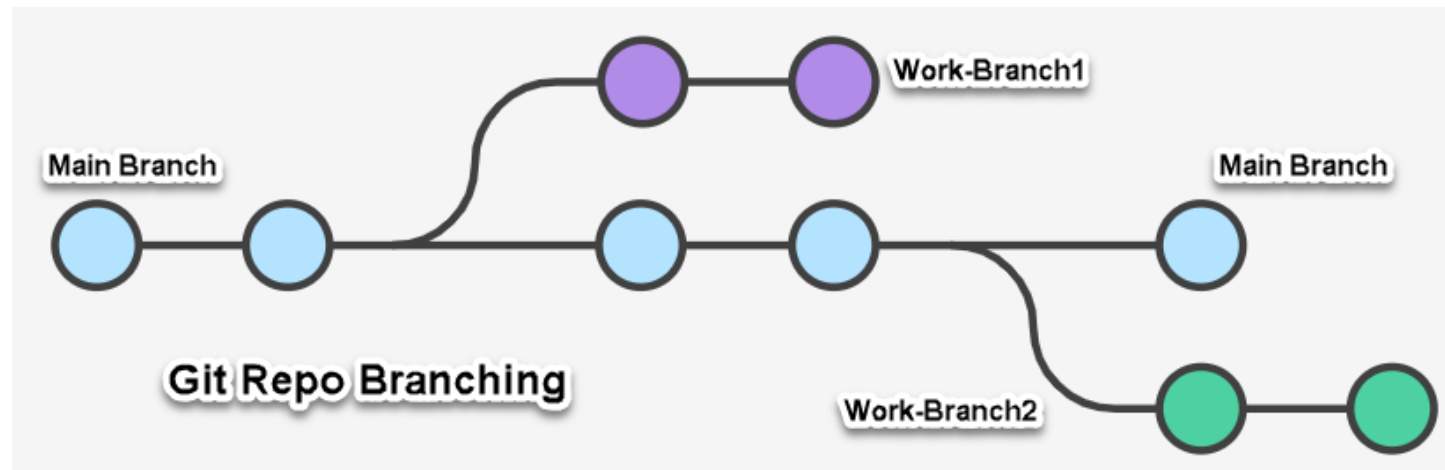
The changed path is for the second Bash terminal's selected work area. The terminal indicator shows that two Bash shells are now in use and selectable.



Git Repo Branching

When you clone a Git repo, your local clone is a replicated copy of the Git master repo, with a default **branch** named '**main**'. A Git branch is a working area. However, your **main** should not be used as a working area. The **main** branch is always used to synchronize your local repo with the master repo.

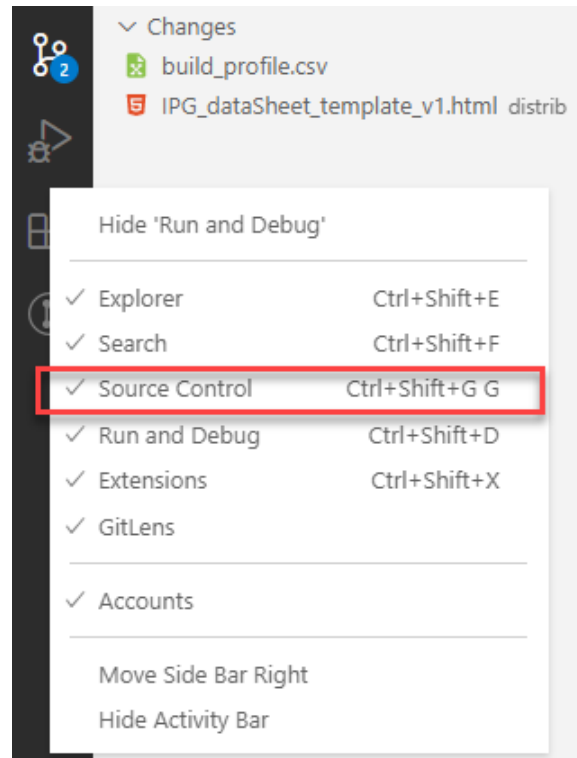
To author and edit, you need to create a new working area: you need to create a branch. You work in a named branch, use it to stage updates to the master, and when synchronized, your named branch's changes are not only in the master repo, but also in your main branch. You can have multiple working branches active and easily switch between them. Each branch is started as a replicated copy of your main branch. A good recommended practice is to use each branch for a specific work task.



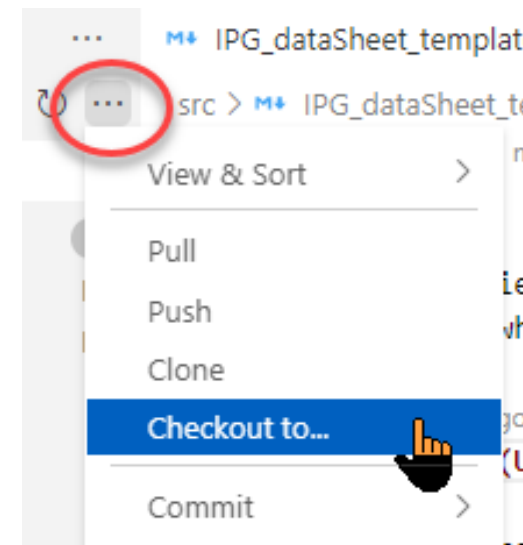
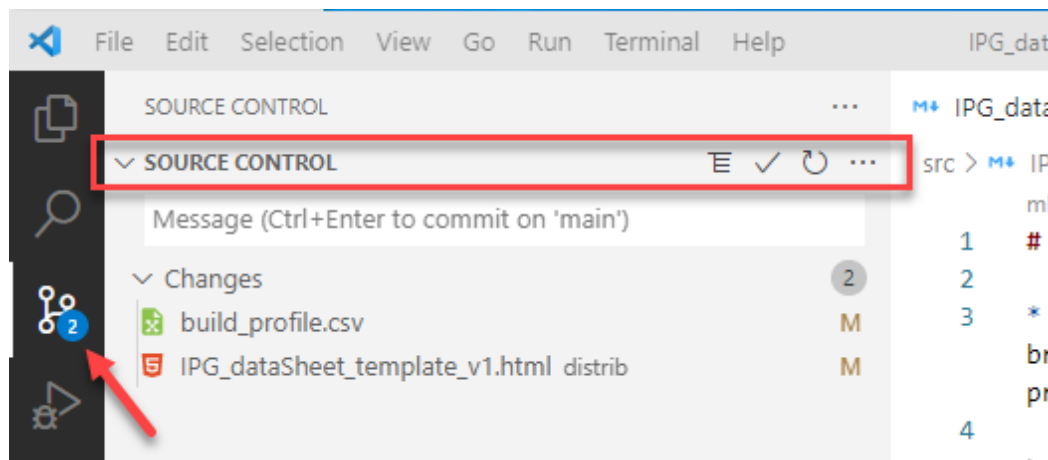
Git Repo Branching 2

You can create a branch using the terminal '`git checkout -b <YourNewBranchName>`' command. However, the easier way to create a branch is to take advantage of the functionality inherent in VS Code.

The left edge of VS Code contains Activity Bar icons for frequently used functional areas. Go to the **Source Control** icon. If you don't see the Source Control icon, right click on the Activity Bar to enable it.

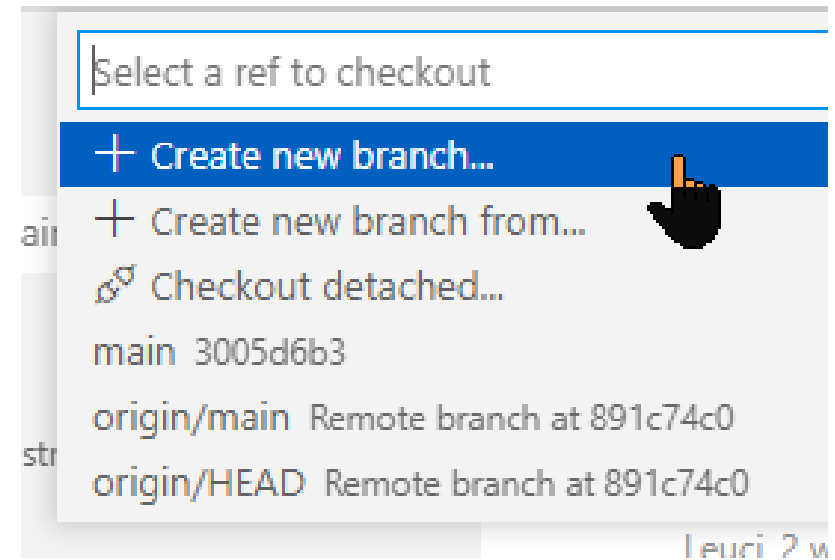


Left click on the Source Control icon, as shown below. This brings up the Source Control submenu. At the right of this, left click on the three ellipsis dots (...) to bring up a Selection list. Left click on '**Checkout to...**'.



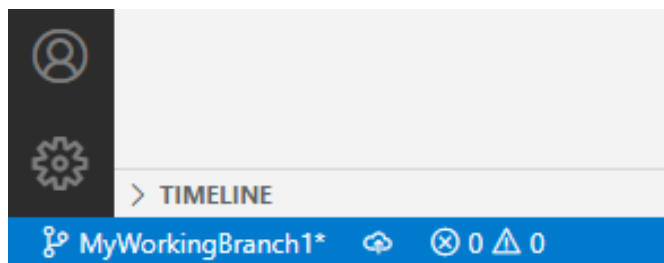
Git Repo Branching 3

This brings up the Command Area at the top of VS Code enabling you to select a branch. You can pick one of the existing branches in the scrollable list. Your **'main'** branch is shown too. If you select an existing branch, VS Code will switch to that branch for continued work there. Click **Create new branch...** This selection allows you to name and create a new branch as a working area:



Please provide a new branch name (Press 'Enter' to confirm or 'Escape' to cancel)

By creating a new branch, VS Code copies your local repo to that branch and enables work there. The lower left corner shows the new branch as your working area. This is also shown in your Bash terminal:



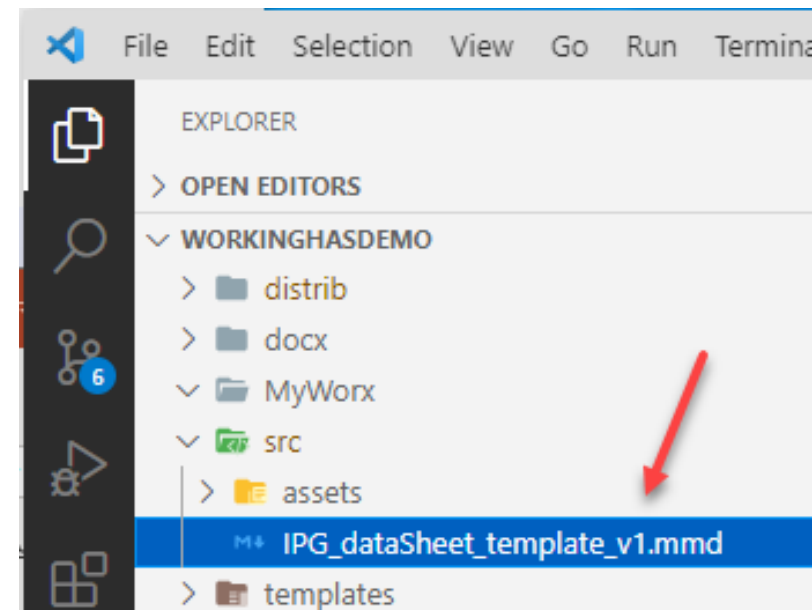
Basic Markdown Editing

ProMark uses a cloned Git repo on your local PC to create, edit and publish Intel HAS documents. The most basic activity in this process is using VS Code (Visual Studio Code) to edit markdown. Be sure to create or switch to a working branch first!

Navigate to the 'src' directory of the repo and expand the list. Select the 'assets' subdirectory. Click on the first Markdown (.mmd) file so that it opens in an editing tab. In our repo WorkingHASDemo this is 'IPG_dataSheet_template_v1.mmd'.

A Markdown file is a text file: it is composed solely of ASCII characters. There are no binary elements or data structures in it. The formatting that you see is not in the file, but in how VS Code interprets its content.

The content displayed in the tab can be edited. Move your mouse next to 'Features' and type in the text ' (Updated) '. Right-click on the file name in the tab and select 'Open Preview' or enter the key combination **CTRL+SHIFT+V**.



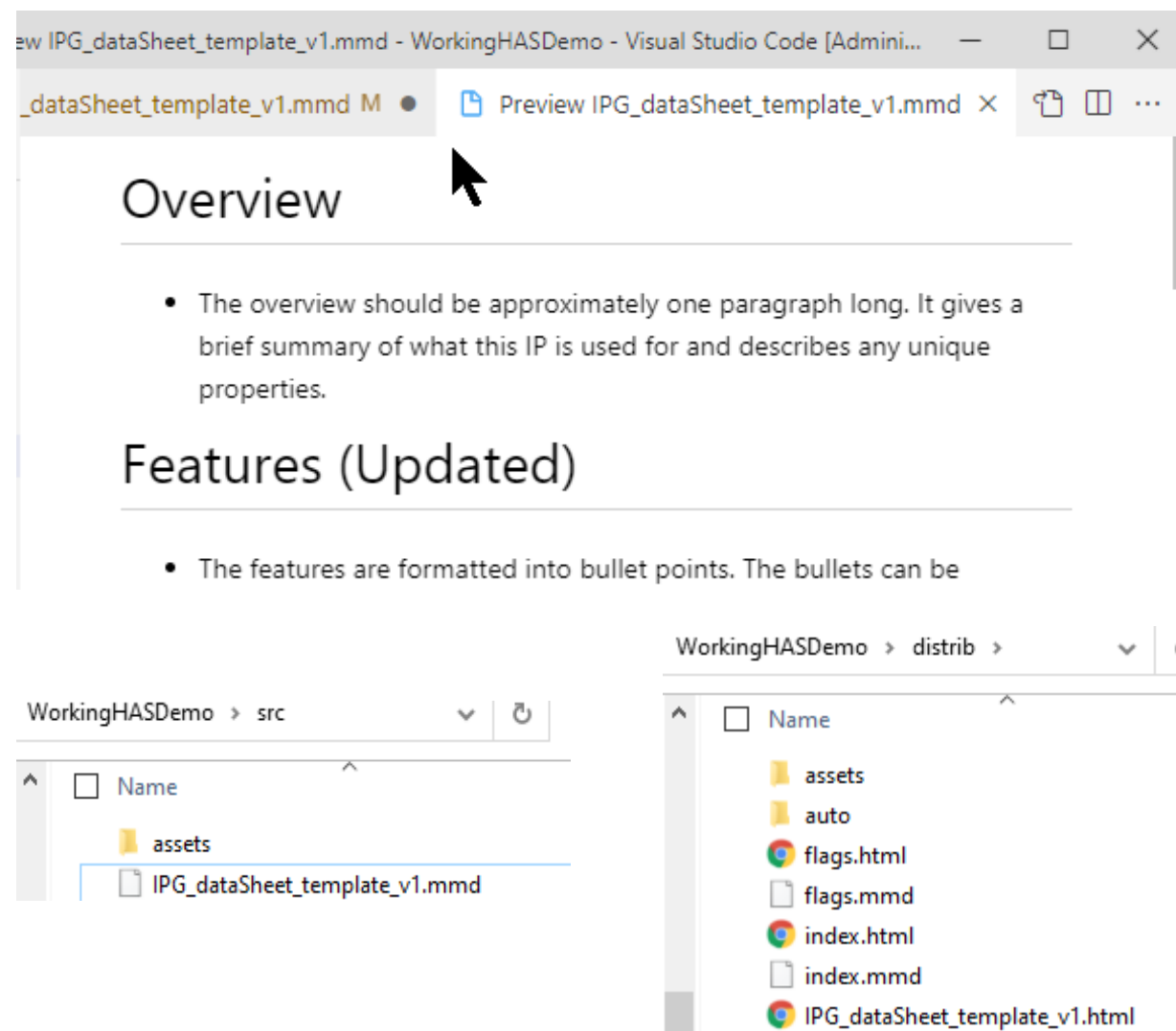
```
M+ IPG_dataSheet_template_v1.mmd M X
src > M+ IPG_dataSheet_template_v1.mmd > [abc] # Overview
mkwilbux2021, 3 months ago | 1 author (mkwilbux202
1 # Overview mkwilbux2021, 3 month
2
3 * The overview should be approximatel
long. It gives a brief summary of wha
for and describes any unique properti
4
You, seconds ago | 2 authors (mkwilbux2021 and othe
5 # Features
6
```

Basic Markdown Editing 2: Preview

The preview of the file is opened in a second, adjacent tab. This preview shows your Markdown rendered as HTML, suitable for publishing as a web page. The file's appearance is visibly changed.

The VS Code paradigm is to author and edit in the Markdown tab, and then preview the contents in rendered HTML. This HTML preview is **not** editable. All editing is done in the Markdown tab.

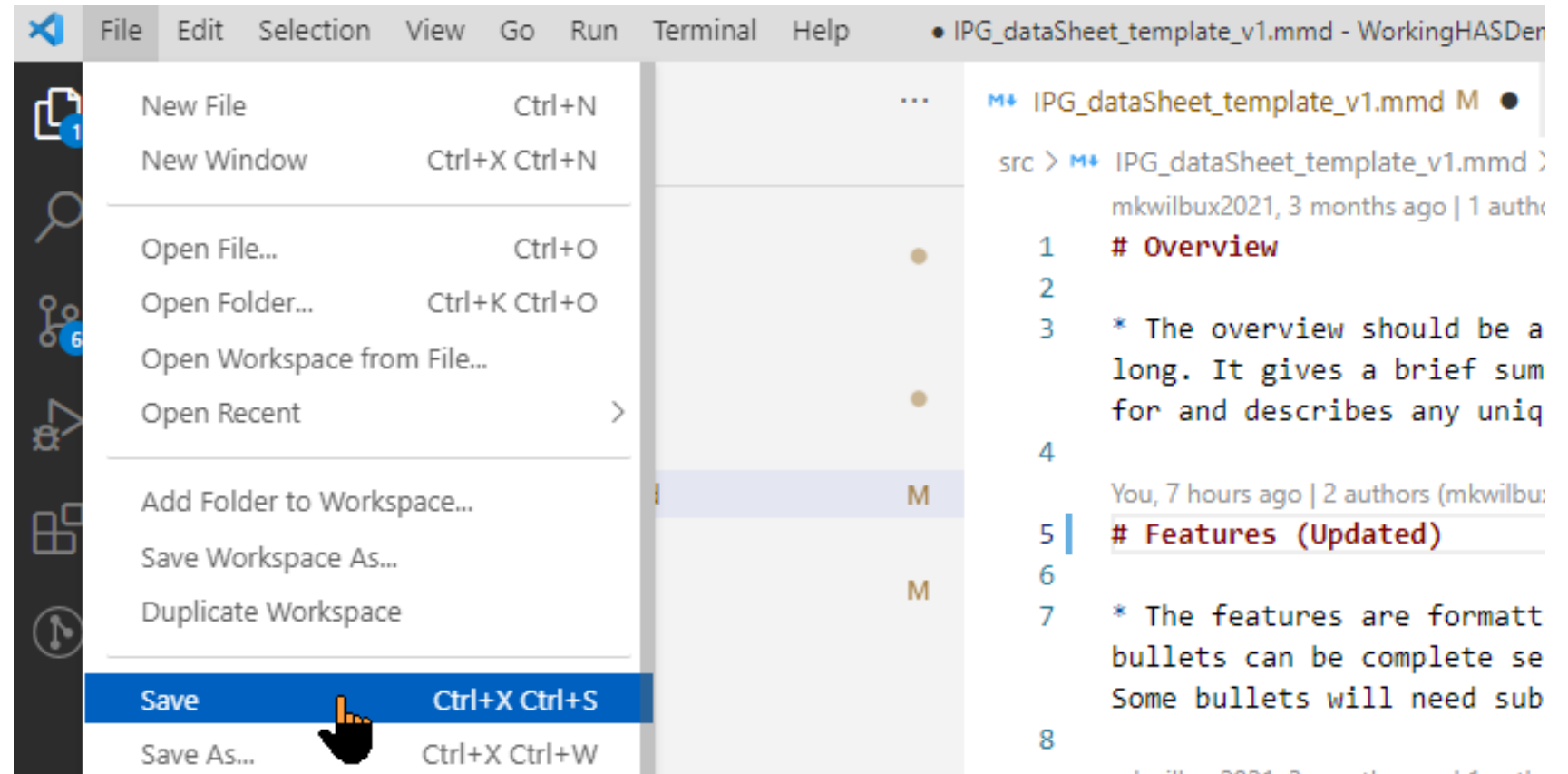
This change is not yet captured in the repo. You need to take a series of steps to make this change permanent. Go to the actual .mmd file on your PC. Its text contents are unchanged, nor is the HTML version of the file in '**distrib**' changed.



Basic Markdown Editing 3: Save

Go to your Bash terminal and at the \$ type `git status`. Press the **Enter** key. Git reports that a file has been edited but not saved: **Modified: src/IPG_dataSheet_template_v1.mmd**

The first step in preserving the edit is to save the file in VS Code. VS Code has visual cues that the file needs to be saved: the 'M' to the right of the file name, and the large dot on the Markdown tab. Select (highlight) the name of the file, and then go to the top **File** menu. Left click and choose **Save**, or enter **Ctrl+X Ctrl+S**



Basic Markdown Editing 4: Add, Commit, Build

Next, you need to propagate the change to your local Git repo. Go to your Bash terminal and at the \$ enter `git add --all`. Enter `git status` again. Git still reports that a file has been edited but not saved: **Modified: src/IPG_dataSheet_template_v1.mmd** However, the VS Code shell message color has changed, indicating a safer state. This step **Adds** the pending change to the repo.

The last step is to **Commit** the pending change to the repo, making it persistent in your local repo. At the \$ enter `git commit -m "New content"`. This is a script command that preserves all pending file changes, insertions and deletions. Notice the **M** displayed after the file name is now gone. Enter `git status` again. Git now reports that there is **nothing to commit, working tree clean**. There are no further pending changes for your local repo.

It is crucial to understand that VS Code is ProMark's **development environment**. You author and edit Markdown in VS Code to publish HAS content. Now that we have newly updated content, it is necessary to construct the HTML output to reflect the updated Markdown file.

At the \$ enter `build`. This is a script command that launches the construction process. If successful, Git will report: **SUCCESS | Build PASSED** Go to your PC's repo **WorkingHASDemo/distrib** subdirectory and double click on the HTML version of the file. Your change will load in your web browser.

Basic VS Code and Git Review

- Use VS Code to ‘**clone**’ a GitHub ProMark repo, copying an image of the repo to your PC.
- You operate on your repo from VS Code, not directly from your PC.
- Edit ProMark ‘.mmd’ Markdown files in the **Edit tab**, and view changes in the **Preview tab**
- Use the Bash Terminal (shell) in VS Code to enter Git commands at the **\$** prompt; All Bash commands are terminated with the **Enter (Return Key)**
- You work on your local repo, not the GitHub master repo. Upcoming: synchronizing with the master

Operation / Task / Git Command	Comment / Note
Associate ‘.mmd’ files as Markdown type	Necessary to edit / preview content
Set Bash as your VS Code Terminal	Allows VS Code to issue Git commands
<code>git status</code>	Returns current state of repo operations
<code>CTRL+SHIFT+V</code>	Open HTML preview of Markdown file
<code>CTRL+X CTRL+S</code>	To save edits in VS Code (or use File::Save menu)
<code>git add --all</code>	Signals your repo that update(s) are ready
<code>git commit -m "update reason"</code>	Sends your update(s) to your repo from VS Code
<code>build</code>	Reconstructs your repo with the new update(s)

Converting a Word File to ProMark

If your content is in Word format, you can convert your Word files to MultiMarkdown (MMD) using ProMark.

1. Open Git Bash (terminal).
2. Add the path (if needed) to the Word file.
3. Type: `doc2mmd Filename.docx`

Example: `doc2mmd IPG_dataSheet_template_v1.0-PM_0-1.docx`

A directory is created with the generated MultiMarkdown (MMD) and HTML files.

```
mkwilbux@mkwilbux-mobl MINGW64 ~/github/HAS-demo-training/docx (main)
$ ls
IPG_dataSheet_template_v1.0-PM_0-1.docx

mkwilbux@mkwilbux-mobl MINGW64 ~/github/HAS-demo-training/docx (main)
$ doc2mmd IPG_dataSheet_template_v1.0-PM_0-1.docx
19:41:51 INFO : Creating output directory 'C:\Users\mkwilbux\github\HAS-demo-training\do
cx\IPG_dataSheet_template_v1'
19:41:51 INFO : Parsing docx document
19:41:52 INFO : Converting tables to IPG_dataSheet_template_v1.xlsx
19:41:52 INFO : Saving markdown
19:41:52 INFO : Compiling to HTML
```

Converting a Word File to ProMark

A directory is created with the generated MultiMarkdown (MMD) and HTML files.

```
mkwlbux@mkwlbux-mobl MINGW64 ~/github/HAS-demo-training/docx (main)
$ ls
IPG_dataSheet_template_v1.0-PM_0-1.docx

mkwlbux@mkwlbux-mobl MINGW64 ~/github/HAS-demo-training/docx (main)
$ doc2mmd IPG_dataSheet_template_v1.0-PM_0-1.docx
19:41:51 INFO : Creating output directory 'C:\Users\mkwlbux\github\HAS-demo-training\do
cx\IPG_dataSheet_template_v1'
19:41:51 INFO : Parsing docx document
19:41:52 INFO : Converting tables to IPG_dataSheet_template_v1.xlsx
19:41:52 INFO : Saving markdown
19:41:52 INFO : Compiling to HTML

mkwlbux@mkwlbux-mobl MINGW64 ~/github/HAS-demo-training/docx (main)
$ ls
IPG_dataSheet_template_v1/  IPG_dataSheet_template_v1.0-PM_0-1.docx

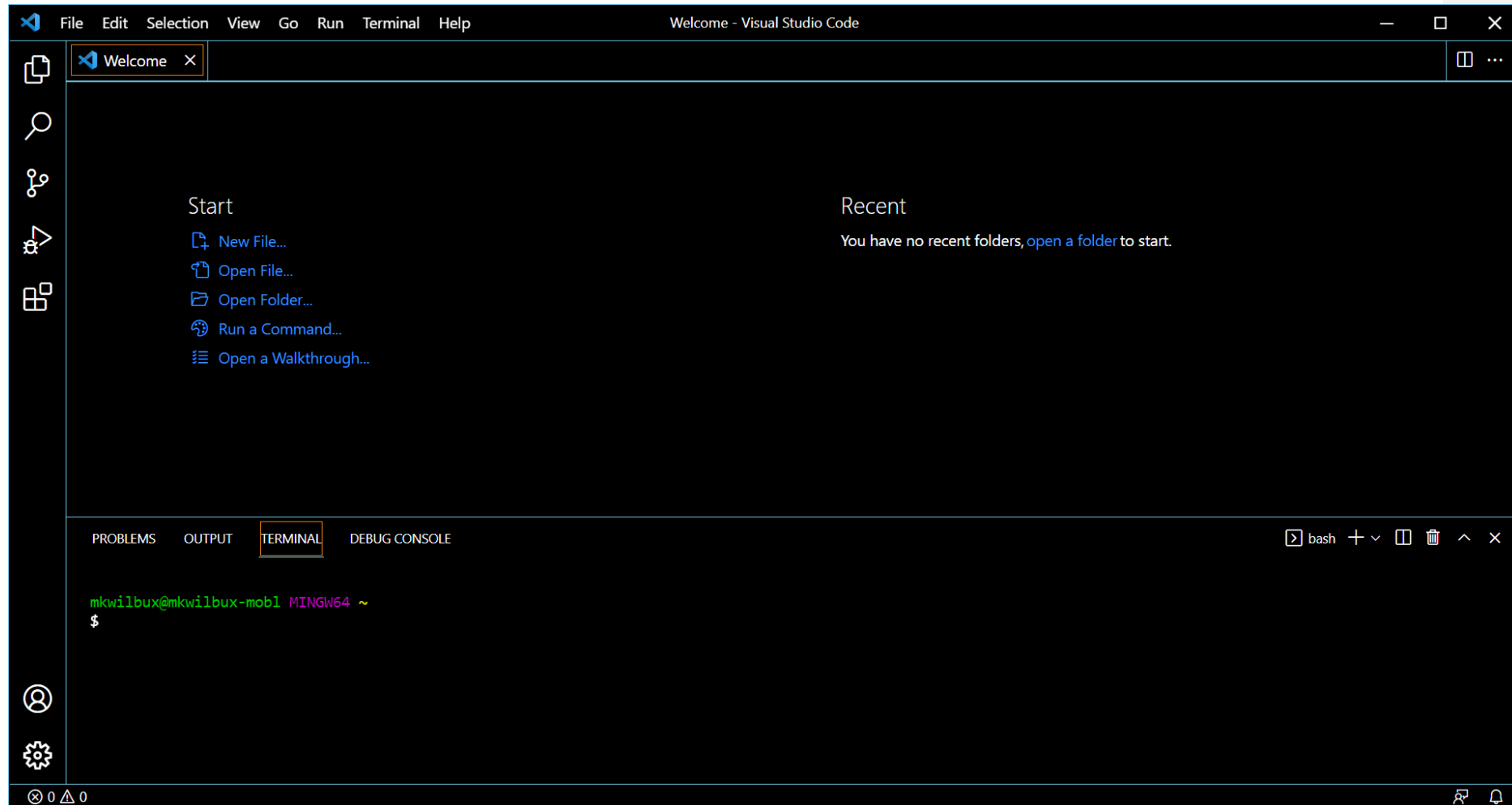
mkwlbux@mkwlbux-mobl MINGW64 ~/github/HAS-demo-training/docx (main)
$ cd IPG_dataSheet_template_v1

mkwlbux@mkwlbux-mobl MINGW64 ~/github/HAS-demo-training/docx/IPG_dataSheet_template_v1 (main)
$ LS
IPG_dataSheet_template_v1.html  IPG_dataSheet_template_v1.mmd  assets  auto
```

Using Visual Studio Code (VS Code)

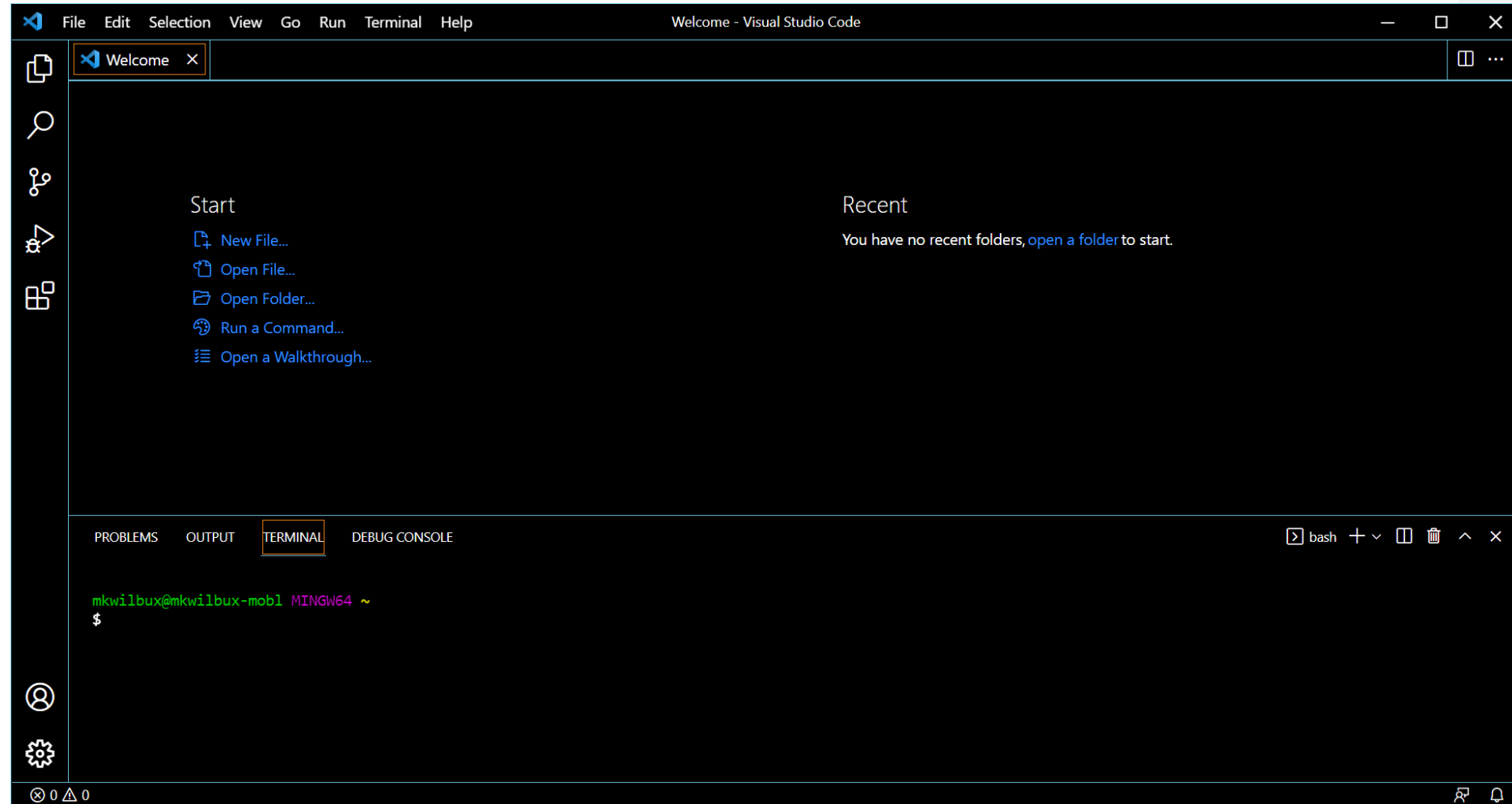
ProMark offers the flexibility to Bring Your Own Editor (BYOE). The editor recommended by ProMark is Visual Studio Code (VS Code).

https://pmdoc.tools.gitlab-pages.devtools.intel.com/pmdoc_tools/Installation/VSCode%20for%20PM_DOC.html



Editing File Contents: Visual Studio Code

1. Open VS Code
2. Go to Terminal in the menu
3. Select New Terminal
4. A terminal will display
5. Change the default terminal to Bash
 1. Select the Drop Down Menu
 2. Select Default Profile
 3. Select Bash as the default



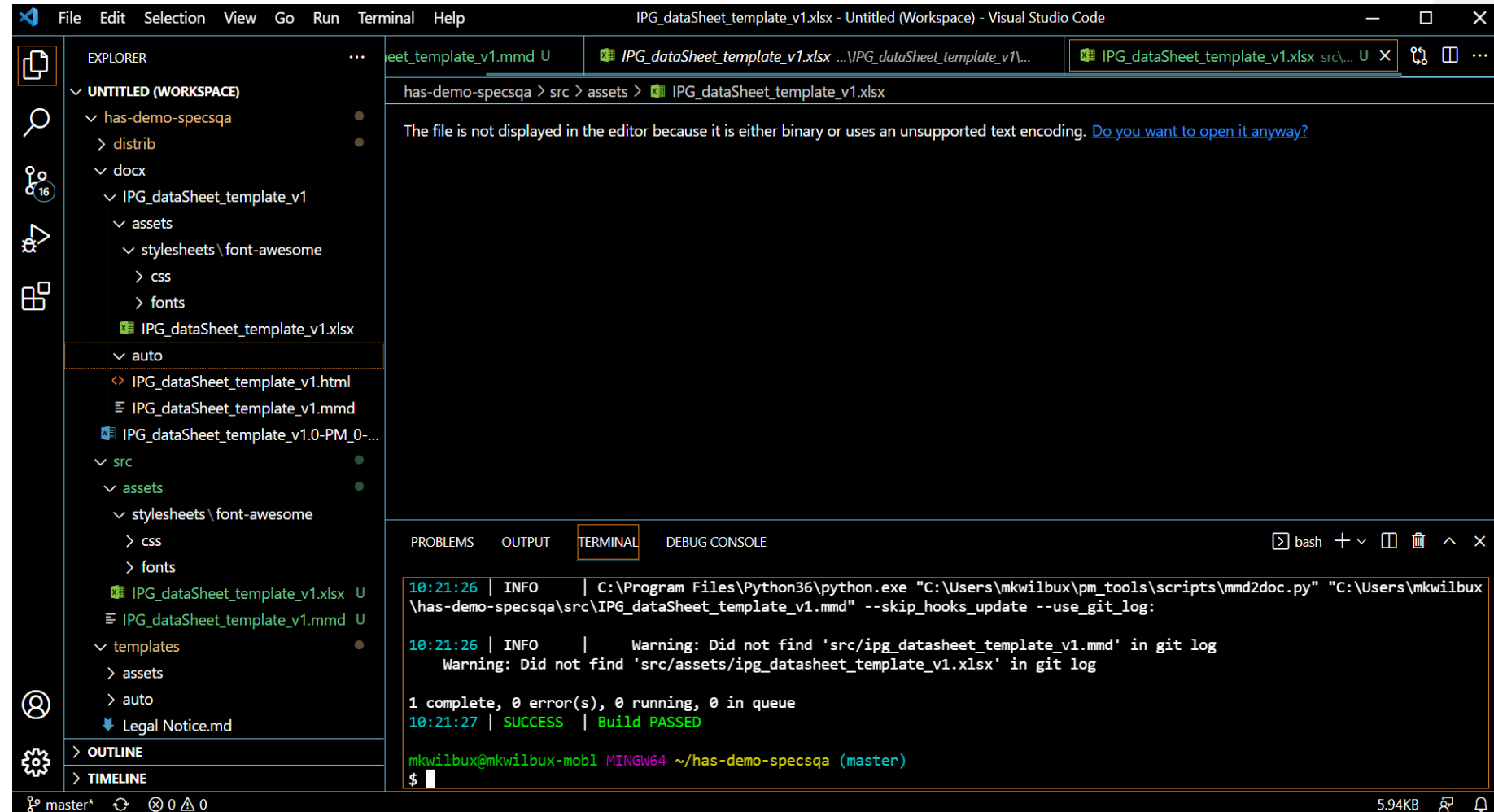
Editing File Contents: Visual Studio Code

Now we will add the file folder for the document

To view files and directories on the left panel, go to View>Explorer

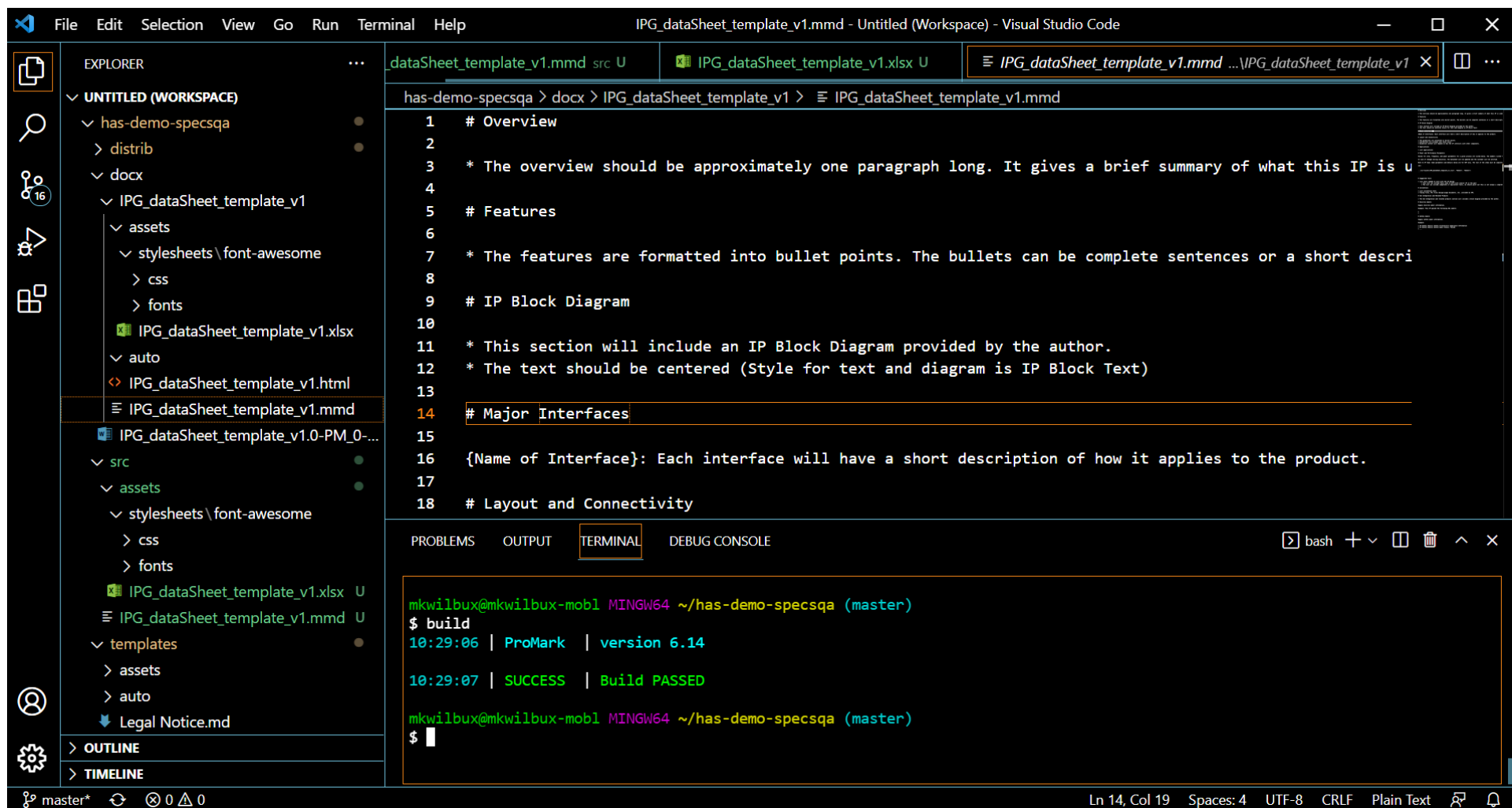
1. File > Add new folder
2. Navigate to the folder
3. Click Add

The directory for the document displays on the left side panel of VS Code. Here you can easily navigate and view files for the document.



Editing File Contents: Local

Make changes to the content. Save the file.



The screenshot displays the Visual Studio Code interface with a workspace titled "IPG_dataSheet_template_v1.mmd - Untitled (Workspace)". The Explorer sidebar on the left shows a project structure with folders like "has-demo-specsqa", "docx", and "IPG_dataSheet_template_v1". The file "IPG_dataSheet_template_v1.mmd" is selected and open in the editor. The editor content shows a template for a data sheet with sections for Overview, Features, IP Block Diagram, Major Interfaces, and Layout and Connectivity. The terminal at the bottom shows the command "build" being executed, resulting in a successful build of ProMark version 6.14.

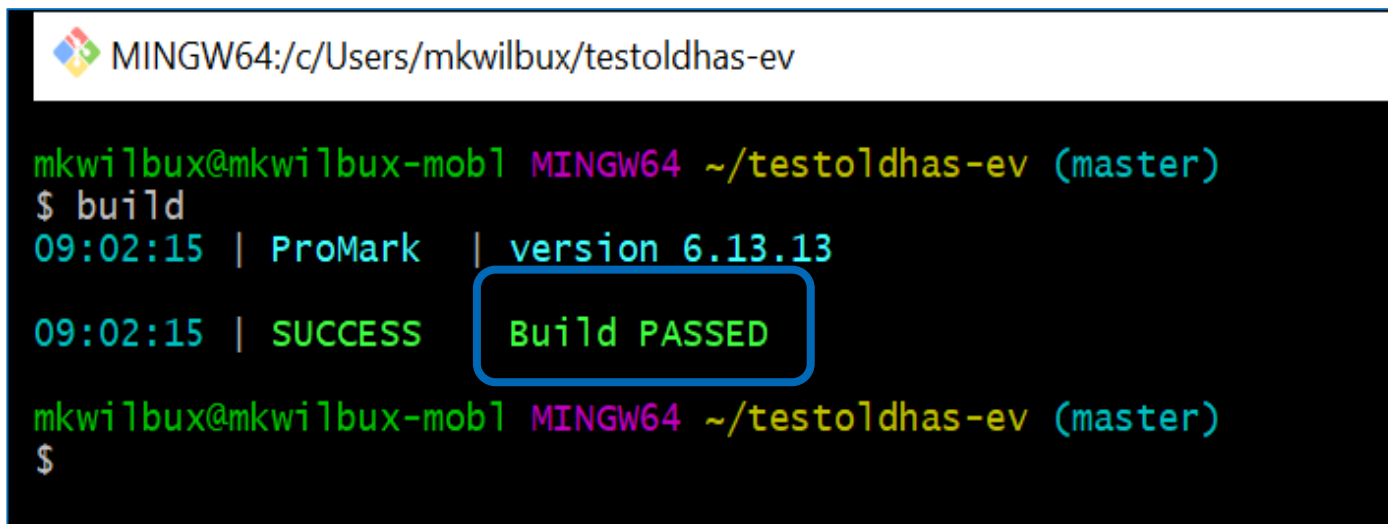
```
has-demo-specsqa > docx > IPG_dataSheet_template_v1 > IPG_dataSheet_template_v1.mmd
1  # Overview
2
3  * The overview should be approximately one paragraph long. It gives a brief summary of what this IP is u
4
5  # Features
6
7  * The features are formatted into bullet points. The bullets can be complete sentences or a short descri
8
9  # IP Block Diagram
10
11 * This section will include an IP Block Diagram provided by the author.
12 * The text should be centered (Style for text and diagram is IP Block Text)
13
14 # Major Interfaces
15
16 {Name of Interface}: Each interface will have a short description of how it applies to the product.
17
18 # Layout and Connectivity
```

```
mkwilbux@mkwilbux-mobl MINGW64 ~/has-demo-specsqa (master)
$ build
10:29:06 | ProMark | version 6.14
10:29:07 | SUCCESS | Build PASSED
mkwilbux@mkwilbux-mobl MINGW64 ~/has-demo-specsqa (master)
$
```

Building HTML Files

1. In the terminal, type **build** to generate HTML files from the markdown.
2. If the build is successful, the message “**Build PASSED**” is displayed.

The generated files are stored in the **distrib** directory.

A terminal window with a black background and white text. The title bar at the top shows a Windows logo icon followed by the path "MINGW64:/c/Users/mkwilbux/testoldhas-ev". The terminal content shows a user prompt "mkwilbux@mkwilbux-mob1" followed by "MINGW64 ~/testoldhas-ev (master)". The user enters the command "\$ build". The output shows a progress bar: "09:02:15 | ProMark | version 6.13.13". Below that, another progress bar shows "09:02:15 | SUCCESS | Build PASSED", where "Build PASSED" is highlighted with a blue rounded rectangle. The terminal ends with the user prompt "\$".

```
MINGW64:/c/Users/mkwilbux/testoldhas-ev

mkwilbux@mkwilbux-mob1 MINGW64 ~/testoldhas-ev (master)
$ build
09:02:15 | ProMark | version 6.13.13
09:02:15 | SUCCESS | Build PASSED
mkwilbux@mkwilbux-mob1 MINGW64 ~/testoldhas-ev (master)
$
```

Building PDF Files – **** PDFs for Internal Use Only ****

1. To generate a PDF file for **Internal use only**, open Git Bash (terminal).

2. In the terminal, type:

```
build --pdf filename.mmd
```

3. If the build is successful, the message “**Build PASSED**” is displayed.

4. The generated files are stored in the **distrib** directory.

```
mkwilbux@mkwilbux-mobl MINGW64 ~/newclass/has-demo-specsqa (master)
$ ls
build.cfg  build_profile.csv  distrib/  docx/  readme.md  src/  templates/

mkwilbux@mkwilbux-mobl MINGW64 ~/newclass/has-demo-specsqa (master)
$ build --pdf src/IPG_TRM_template_v1.mmd
09:47:19 | ProMark | version 6.16.3

09:47:19 | INFO | Building templates/Legal\ Notice.md
09:47:19 | INFO | Building templates/Title.md
09:47:19 | INFO | Building 2 document(s)
2 complete, 0 error(s), 0 running, 0 in queue 09:48:17 | SUCCESS | Build PASSED

mkwilbux@mkwilbux-mobl MINGW64 ~/newclass/has-demo-specsqa (master)
$ ls
build.cfg  build_profile.csv  distrib/  docx/  readme.md  src/  templates/

mkwilbux@mkwilbux-mobl MINGW64 ~/newclass/has-demo-specsqa (master)
$ cd distrib/

mkwilbux@mkwilbux-mobl MINGW64 ~/newclass/has-demo-specsqa/distrib (master)
$ ls
IPG_TRM_template_v1.htm1  IPG_TRM_template_v1.pdf  IPG_TRM_template_v1.pdf.htm1  assets/  auto/  flags.htm1  flags.mmd  index.htm1  index.mmd
```

Building PDF Files – **** Internal Use Only ****

Sample output

IPG_TRM_template_v1.pdf - Intel Enterprise PDF Reader

FileHomeCommentFill & SignViewFormProtectShareHelp

Find

Bookmarks

1 About this Document

2 Introduction

3 Functional Description

4 Feature(s) <One per Subsection>

5 Programmers Model

1 About this Document

1.1 Audience

The information in this document is intended for an integration team that is integrating this IP into an SoC.

1.2 References

If you need more information on this IP, you may find these documents or websites helpful:

[Table: Sheet1 \(source\)](#)

Document Name	Link / Location
IP_Name Integration Guide	doc directory of the release
IP_Name Datasheet	doc directory of the release
IP_Name TRM	doc directory of the release
IP_Name Release Notes	doc directory of the release
IP_Chassis_FuSa_HW_Architecture_Guideline	goto/doctracker and search For "Functional Safety (FuSa) - Chassis 2.2 "

1 / 6

121.03%

Pushing Local Content to the Repo

In this step, push the local sources (your edits) to the master (remote repo). You use several different Git commands to push your content to the repo.

<code>git status</code>	Check the status of your cloned repo – which branch, what changed
<code>git add .</code>	Add files to be pushed. The '.' Indicates the present working directory.
<code>git commit -m "Message here"</code>	The message describing this push.
<code>git push</code>	Push the local files (working source) to the remote repo (master or other). The push synchronizes your update(s) with the master. There may be a conflict if someone else has pushed a change to the same file(s). These conflicts must be resolved by the authors. Git has no mechanism to resolve conflicts.

Pushing Local Content to the Repo

Use the terminal and press **Enter** after all Git commands.

1. Before pushing content to the remote repo, check the current branch:

```
git status
```

Results are displayed.

- If you need to change branches:

```
git checkout branchname
```

2. Stage all files in the current directory:

```
git add .
```

3. Add a commit message:

```
git commit -m "commit message here"
```

Note: Commit messages are required and should clearly list the changes and why the changes were made.

4. Push the files to the remote master repo from your local cloned repo

```
git push
```

Pushing Local Content to the Repo

Push the files to the remote repo from your local cloned repo:

git push

```
mkwlbux@mkwlbux-mobl MINGW64 ~/github/HAS-demo-training (main)
$ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean

mkwlbux@mkwlbux-mobl MINGW64 ~/github/HAS-demo-training (main)
$ git push
Enumerating objects: 49, done.
Counting objects: 100% (49/49), done.
Delta compression using up to 8 threads
Compressing objects: 100% (44/44), done.
Writing objects: 100% (48/48), 545.87 KiB | 10.30 MiB/s, done.
Total 48 (delta 9), reused 0 (delta 0)
remote: Resolving deltas: 100% (9/9), done.
```

Release

Repository Owners publish the document to the final destination. For more information about repository administration, refer to the ProMark [Repository Owner Guide](#).

To publish the content, open a terminal and type the ProMark script command:

release

Additional Resources

Topic	Link
Overview	ProMark QuickStart
GitLab access	GitLab access
Editor training	Editor Overview
Git Basics	Link to GitLab Cheatsheet
PDF build	PDF Syntax
Pre-Requisites	Before You Begin
Markdown (MD) basics	Markdown Basics
MultiMarkdown (MMD) basics	MultiMarkdown Basics
Repo Admin	Repo Admin Basics
Merge Request	Gitlab
Word2mmd	doc2mmd
Troubleshooting	Troubleshooting

Troubleshooting

TOPIC	Recommendation
Exception: Java not found	Uninstall and re-install Adopt OpenJDK. Check PATH.
Fatal error during Git clone	<p>This can occur during the first install when Git Bash was not previously installed.</p> <p>Ensure that Git Bash is installed and try running the installer again with only "Generate/show open SSH key" checked. If you receive this error again, reboot your machine and re-run the previous step.</p>

Terminology

Terminology	Definition
Git Bash	Terminal (shell)
ProMark	Intel Internal documentation and publishing system
GitLab	Repository
VS Code	Recommended MMD markdown editor for use with ProMark
SSH Key	Encrypted Key generated during install to securely identify the user
SSH URL	URL used to clone a repository
MMD	MultiMarkdown (a superset of markdown)
HTML	Hypertext markup language
Repo	Repository – location of documentation project files

ProMark Support

For more information and support from the ProMark team, visit:

[ProMark Office Hours](#)

[Yammer](#)

[Documentation](#)

[Support Requests](#)

Questions?

