GUD Contributors Guide

GUD = GenevaERS User Documentation
GUD repo is hosted on GitHub.com

Windows and Jekyll environment

Version 14 December 2022

Author: Eugene Morrow, emorrow@au1.ibm.com

Document Filename: GUD Contributors Guide 221214.doc

Contents

1 Document overview information 3
1.1 About this document 3
1.2 Summary of changes 3

2 START HERE 4

3 Introduction 5
3.1 The “include” issue 5
3.2 Structure of GUD repo 5
3.3 Levels of folders under “All Markdown Topics” 6
3.4 Overview of includes in GUD 6

4 Install Extension in VSCode 7
4.1 Install Render Line Endings 7
4.2 Configure settings.json 9

5 Install Ruby and Jekyll 11
5.1 Install Ruby 11
5.2 Install Jekyll 15
5.3 Check version of Jekyll and update 16
5.4 Background info on Ruby & Jekyll 17

6 Setup for Prepare-Website (once only) 18
6.1 Once only – setup files to speed your work 18
6.2 Once only – update path environment variable 19
6.3 Once only – set policy for non-native Windows 21

7 How to be a GUD contributor 25

8 How to write Topics 26
8.1 Create a new branch in VSCode (if required) 26
8.2 Topic name and structure 27
8.3 Folder levels for Topics 28
8.4 Link to a topic in the same folder 29
8.5 Link to a topic in a different folder 29
8.6 Link to an entire sub-folder 30
8.7 Insert images in a topic 30
8.8 Create new Level 2 folder of topics 31
8.9 Create new Level 3 folder of topics 33
8.10 How to include stuff 34
8.11 Include coding in a topic 34
8.12 Examples of includes 35
8.13 Finer points of markdown 36
8.13.1 Display some code 36
8.13.2 Control text in a table 37

9 How to write Common Modules 38
9.1 Create a new branch in VSCode (if required) 38
9.2 Just Markdown, that’s all 39
9.3 A common module can include a common module 40
9.4 How to code a common module 40
9.5 Common modules and folder levels 41
9.6 When all common modules and includes are complete 41

10 Run Prepare-Website 42

11 View website on localhost 44
11.1 Run Jekyll serve 44
11.2 View Jekyll site locally 46
12 Commit & merge changes to GUD 47
13 Appendix A: Install Visual Studio Code 52
14 Appendix B: De-install Ruby & Jekyll 56
15 Appendix C: Uninstall Render Line Endings 58
16 Appendix D: Create GUD clone 59
16.1 Create a GUD clone 59
17 Appendix E: Update .gitignore 61
17.1 Update .gitignore 61
17.2 Refresh the cache 62
18 Appendix F: Practice an include 64

1 Document overview information

1.1 About this document

This document is for anyone preparing data for the website "**GenevaERS User Documentation**" on GitHub.com (repo "User-Documentation" under the "GenevaERS" organization). This document describes how to contribute markdown files for this website, and in particular how to do "includes" for those markdown files.

1.2 Summary of changes

Date	Who	Changes
2 Nov 2022	Eugene	First version
9 Nov 2022	Eugene	Updated Section 8.11.1 Display code This now uses html tags <pre><b>.
7 Dec 2022	Eugene	Fixed caution wording in Section 18 Appendix F.
14 Dec 2022	Eugene	'Section 8.10 How to include stuff' is expanded to three sections: Section 8.10 How to include stuff Section 8.11 Include coding in a topic Section 8.12 Examples of includes

2 START HERE

If you want to…	.. then read section:
Clone GUD to your computer	***16 Appendix D: Create GUD clone***
What is this document is solving?	***3 Introduction***
Install software and configure your computer	***4 Install Extension in VSCode*** ***5 Install Ruby and Jekyll*** ***6 Setup for Prepare-Website***
Start work	***7 How to be a GUD contributor***

3 Introduction

3.1 The "include" issue

Problem: the same data need in multiple places on the website.
Two solutions:

1. **Link:** Put data in **Topic A** and all the other topics link to Topic A.
   Markdown provides a way to link between topics.
2. **Include:** Put data in **a common module** and let multiple topics "include" that common file.
   This is a "copybook" situation where multiple places share the same data.
   Markdown does not have a native "include" capability.

The problems with links are:

- One topic may need links to many other topics. Users need to click a link and return a lot.
- When a user links to a topic, there can be unwanted extra stuff. Hassle to find what you want.

Linking to another topic is still valid. For example, consider topic "**Runbook MR91 Control File Generator**".

This topic discusses what MR91 does, how to run it and more. Any topic recommending MR91 can link to this topic. This is a valid use of a link because MR91 is complex and needs a whole topic.

An include is better for something small and specific.

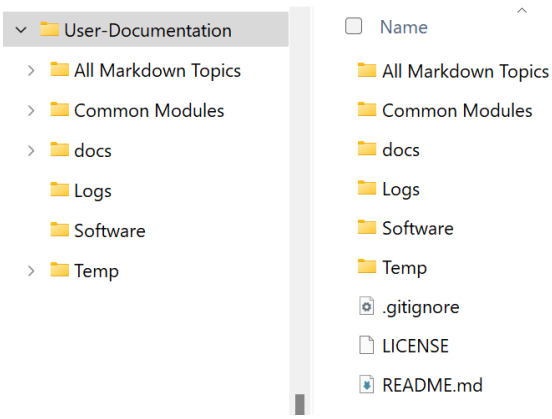
For example, a topic recommends some specific piece of Logic Text.

That topic can include the syntax diagram for that particular piece of Logic Text. The user gets the information without having to click a link and then return.

That topic may also have links to topics that give the full details, including a topic about that Logic Text in general. So both links and includes can be used together.

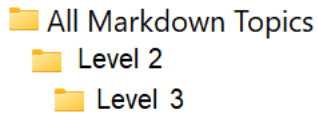
3.2 Structure of GUD repo

The website that Jekyll displays is in the folder "**docs**".



3.3 Levels of folders under “All Markdown Topics”

Most topics are in a Level 2 or Level 3 folder. A few are in “All Markdown Topics” itself which is the only “Level 1” folder.



3.4 Overview of includes in GUD

The include solution is a PowerShell program “**Prepare-Website**”. This software is part of the GUD repo, so the solution is available to everyone.

If a topic has **no includes** Prepare-Website copies it as shown, keeping the same folder level. The folder structure under “All Markdown Topics” and “docs” is the same.

If a topic **HAS includes**, Prepare-Website incorporates the required common modules and places the expanded topic into “docs” in the relevant folder.

Quick summary of what to do:

1. Never touch “**docs**” folder directly.
2. Create a new branch in VSCode.
3. Put all topics under “**All Markdown Topics**” – usually at Level 2 or Level 3.
4. Put any common modules under “**Common Modules**” usually in a folder at Level 2.
5. Run **Prepare-Website**.
6. View your website locally on your computer.
7. Commit and merge your changes in your branch using VSCode and GitHub.

For details see **Section *2 START HERE***.

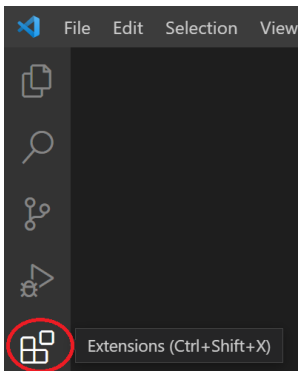
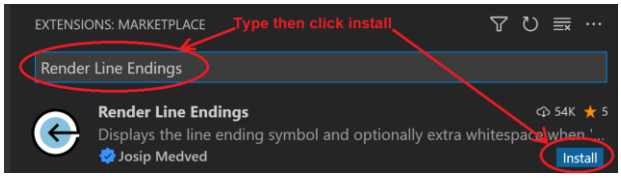
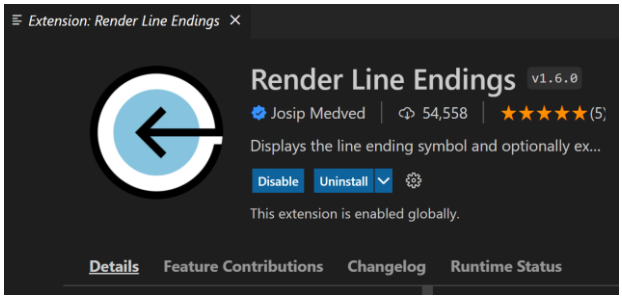
4 Install Extension in VSCode

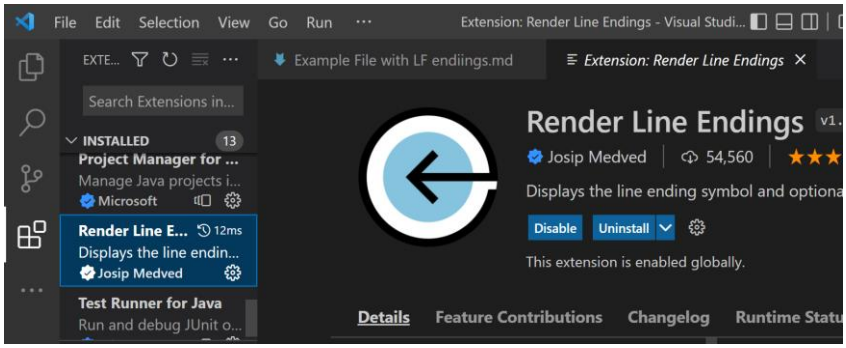
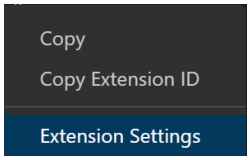
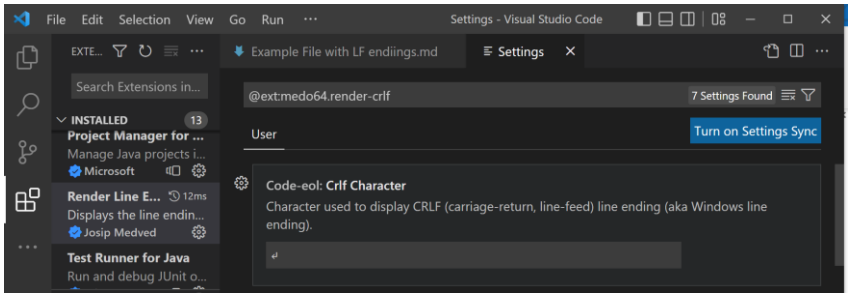
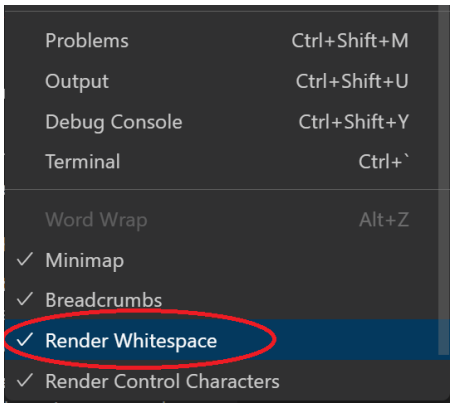
Visual Studio Code has an extension "**Render Line Endings**" that allows you to **easily see blank characters** in markdown files.

If required, use **Section *13 Appendix A: Install Visual Studio Code*** and return here.

Don't like Render Line Endings? Remove with ***Section 15 Appendix C: Uninstall Render Line Endings.***

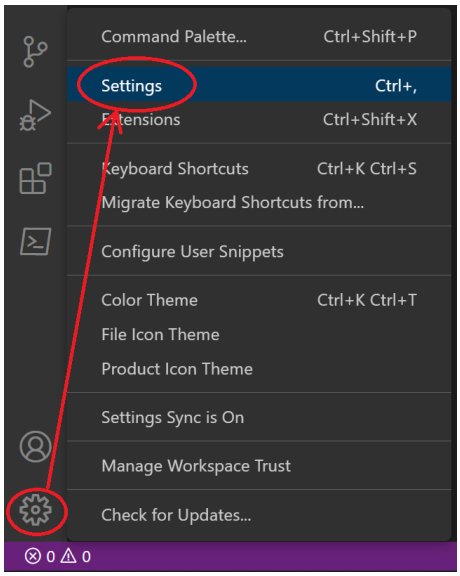
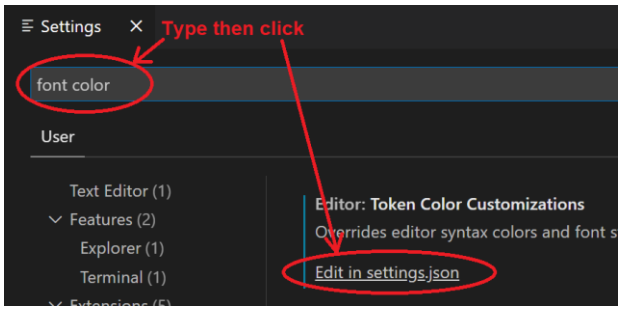
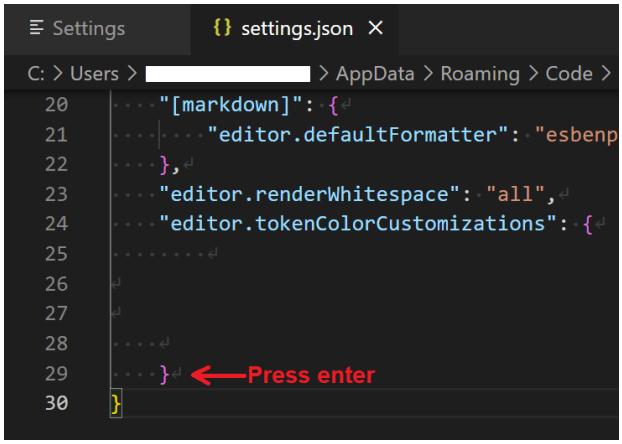
4.1 Install Render Line Endings

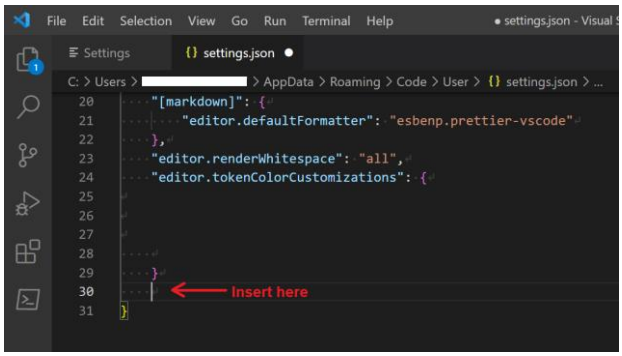
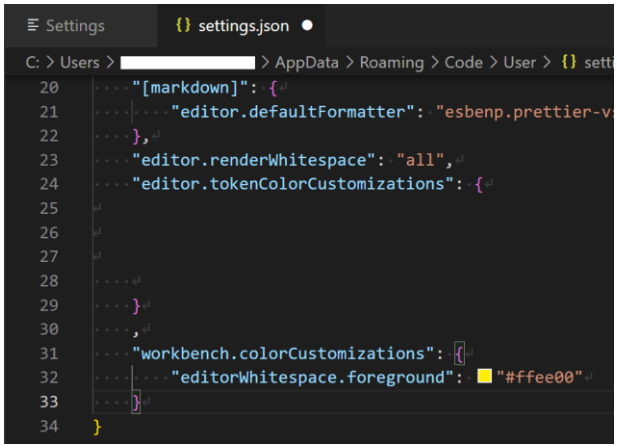
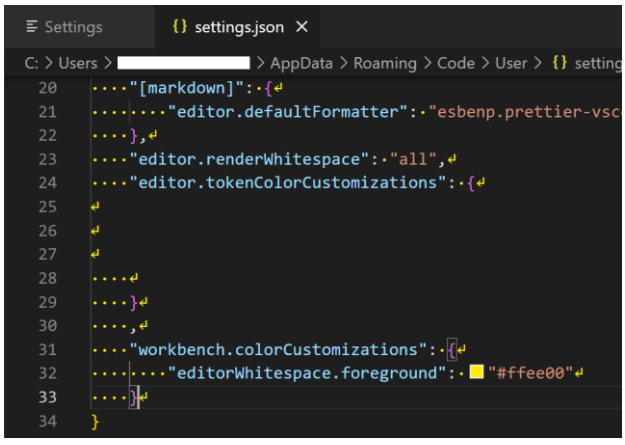
Step	Action	Notes
1	Start **Visual Studio Code**	
2	Click **Extensions** icon If you don't see that icon, click ··· and select **Extensions**	
3	Type **Render Line Endings** in search box and click **Install** beside this item.	
4	Render Line Endings is installed. Click X to close the window at right.	

Step	Action	Notes
5	Ensure you have clicked at left to see the Extensions Click on **Render Line Endings** 	
6	For **Render Line Endings**, click and select **Extension Settings**: 	
7	If you are shown this button, click **Turn on Settings Sync** 	
8	Close windows for Render Line Endings and settings	
9	Click **View** menu in VSCode, scroll to the bottom of the list, and ensure a tick for **Render Whitespace**	
10	Go to next section.	

4.2 Configure settings.json

This makes sure the blank characters are displayed in yellow which is easier to see.

Step	Action	Notes
1	Ensure you are in **Visual Studio Code**	
2	Click Manage icon and select **Settings**	
3	Type **font color** in the search area Click **Edit in settings.json**	
4	Find the position shown – second last (closing) curly bracket. Press **enter**.	

Step	Action	Notes
5	Your cursor should now be in the insert point shown.	
6	Insert this text (can cut and paste): **,** **"workbench.colorCustomizations": {** **"editorWhitespace.foreground": "#ffee00"** **}** 	
7	Click **Ctrl-S** to save settings.json. The file should appear as at right. This shows that Render Line Endings is working – you can see spaces,	

5 Install Ruby and Jekyll

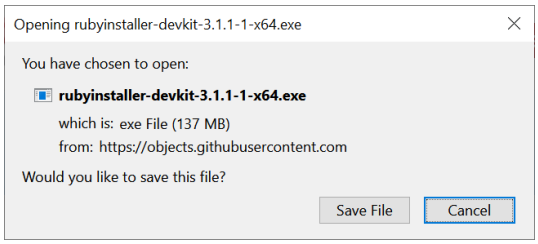
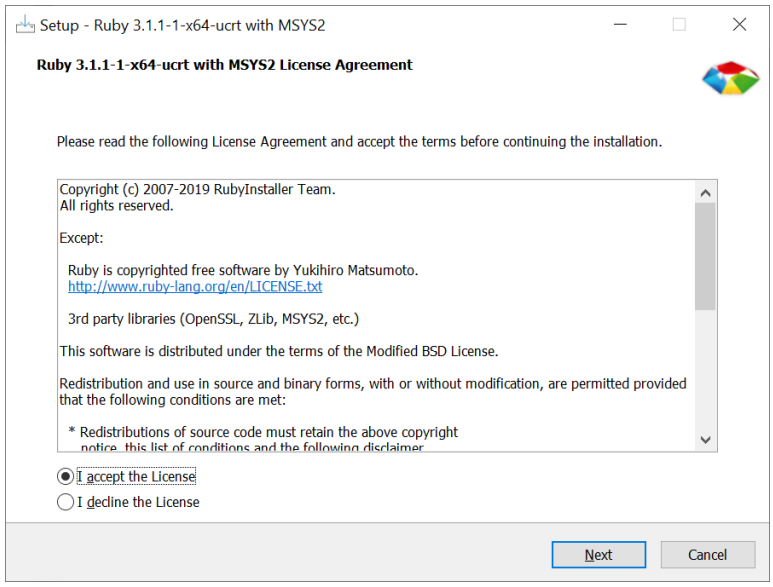
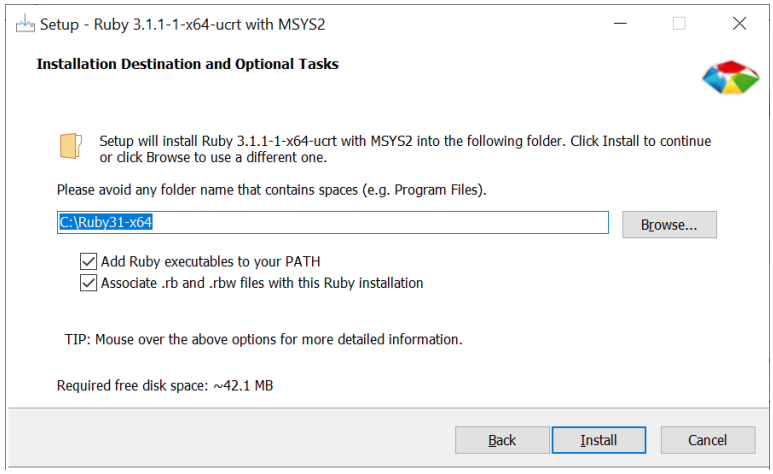
Jekyll is the website software for the GenevaERS User Documentation repo in public GitHub.

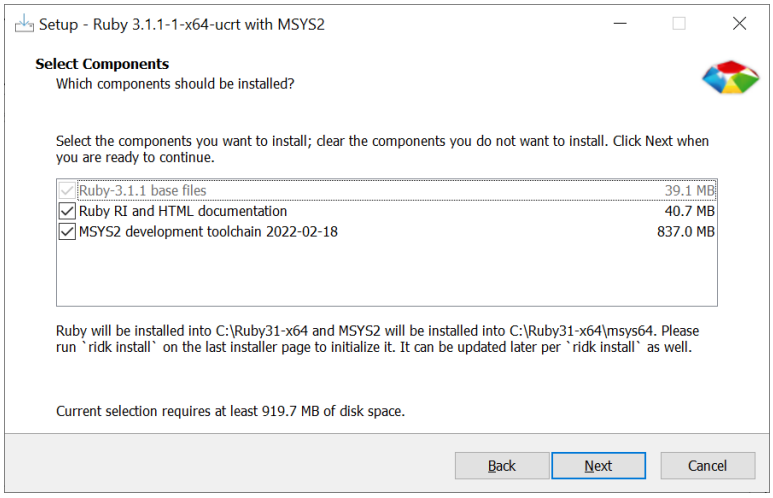
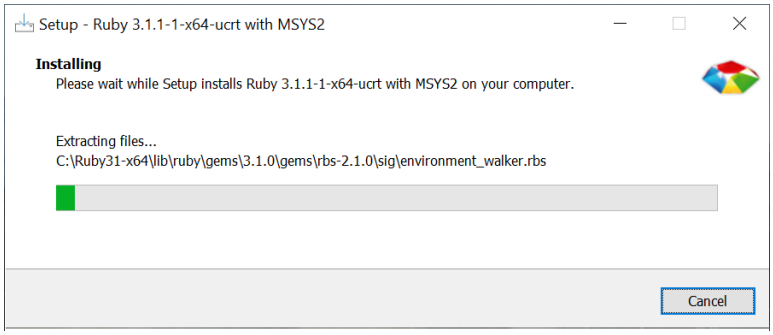
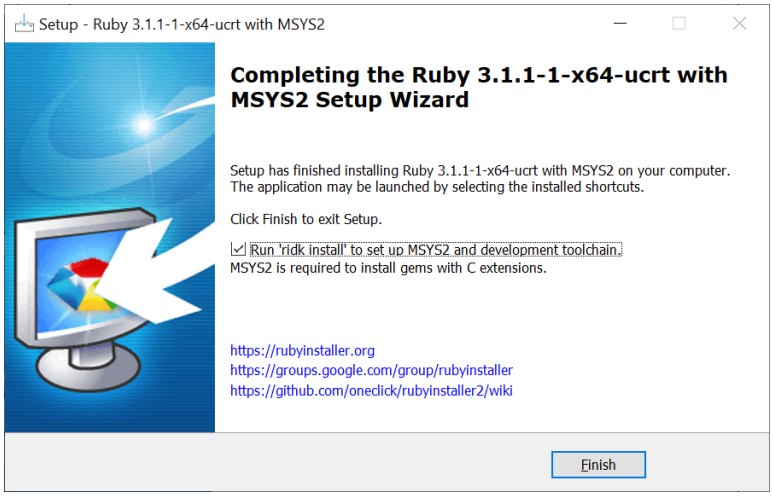
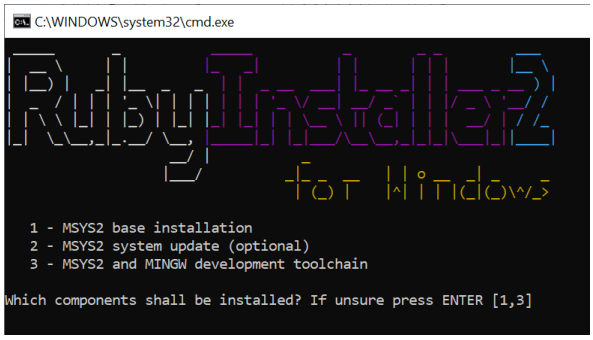
Installing Jekyll is optional and much recommended. With Jekyll you can see how the website looks locally on your computer. You can ensure the markdown looks good before applying the changes to GitHub.

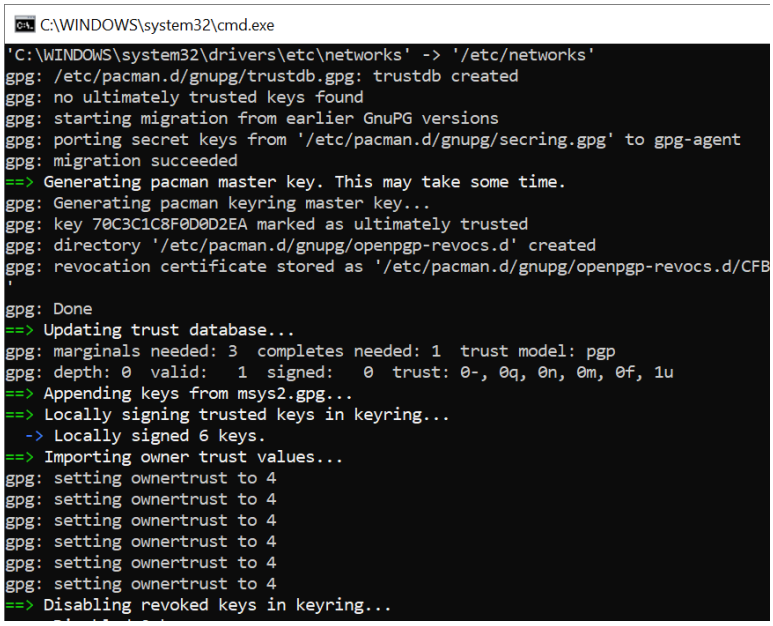
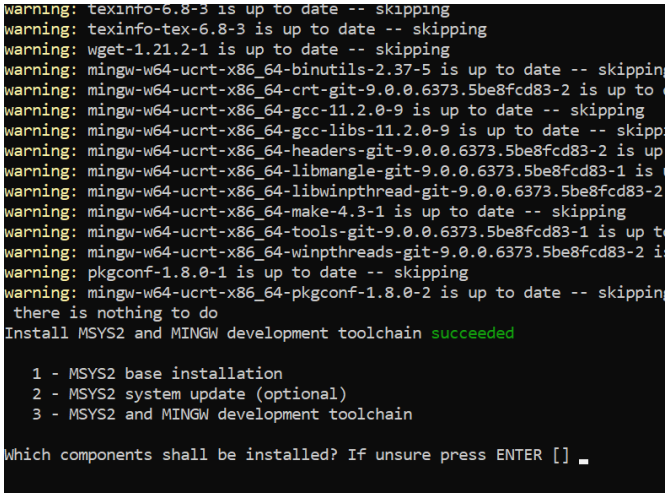
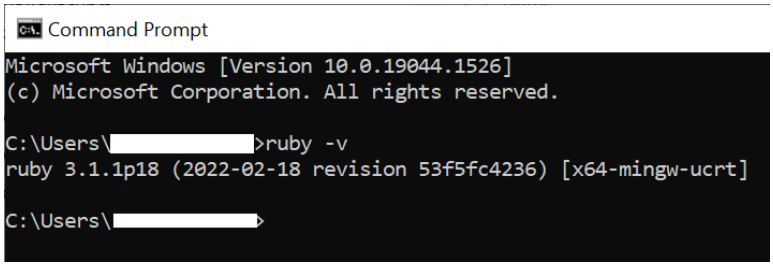
Jekyll is based on Ruby technology, so Ruby is installed first.

5.1 Install Ruby

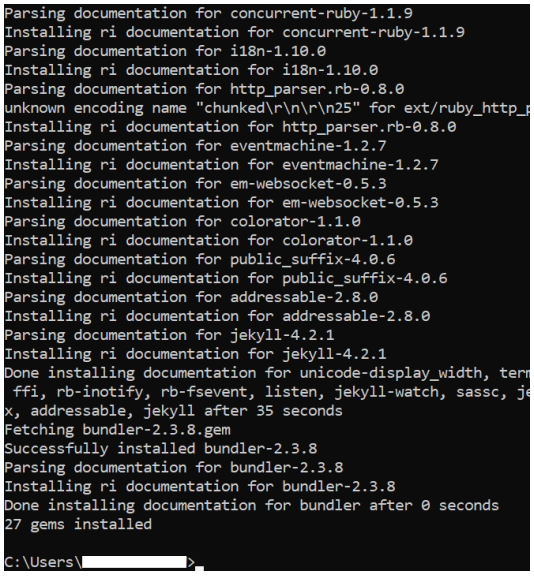
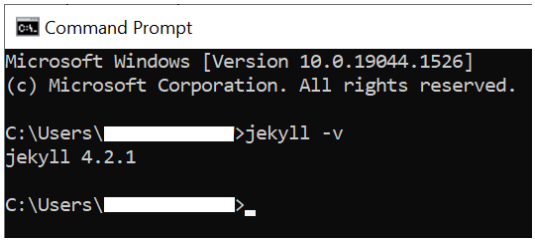
Step	Action	Notes
1	Do you already have Ruby installed? Start a command prompt.	Start, Windows System, Command Prompt
2	Type **ruby -v**	
3	You should see text at right. If so, proceed to the next step. If you see a Ruby version, then you already have Ruby installed and can skip this section.	**'ruby' is not recognized as an internal or external command, operable program or batch file.**
4	Go to URL: https://rubyinstaller.org/	
5	Click **Download**.	

Step	Action	Notes
6	Click the latest version with the DevKit.	
7	Click **Save File**.	
8	Double click the file downloaded.	
9	Click ⊙ **I accept the License** Click **Next**.	
10	Click **Install**.	

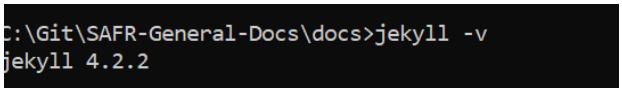
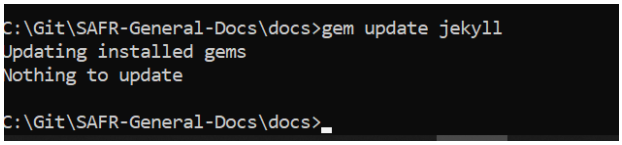
Step	Action	Notes
11	Note MSYS2 – this is the new development kit. Click **Next**.	
12	Install progress appears.	
13	Ensure ☑ **Run 'ridk install' to set up MSYS2 and development toolchain**. Click **Finish**.	
14	Click **Enter**.	

Step	Action	Notes
15	These messages appear for a while but disappear later.	
16	The install messages that remain afterwards are shown at right.	
17	Hit Enter. Install disappears – all complete.	
18	Start a new command prompt.	Start, Windows System, Command Prompt
19	Check Ruby installed: Type **ruby -v**	
20	<u>You should see similar to the screen at right.</u> If so, this section is done. If not, repeat this section.	

5.2 Install Jekyll

Step	Action	Notes
1	Ensure the folder at right has **no files** with these names: **Gemfile** **Gemfile.lock**	C:\Users\(yourname)
2	Do you already have Jekyll installed? Start a command prompt. Type **jekyll -v**	Start, Windows System, Command Prompt
3	You should see text at right. If so, skip the next step and continue.	**'jekyll' is not recognized as an internal or external command, operable program or batch file.**
4	If you see a Jekyll version, then you already have Jekyll installed and can skip this section.	
5	Type **gem install jekyll bundler**	
6		
7	Type **jekyll -v** Jekyll is now installed.	

5.3 Check version of Jekyll and update

Step	Action	Notes
8	Start a command prompt. Type **jekyll -v** As of June 2022, latest is 4.2.2.	
9	To update Jekyll, type **gem update jekyll**	

5.4 Background info on Ruby & Jekyll

This is optional to read.

Ruby is a programming language that is object-oriented and procedural. Developed by Yukihiro "Matz" Matsumoto in Japan in the mid 1990's.

A finished Ruby package is called a "**gem**" and can be installed at the command line. Most gems are libraries. GitHub hosts many Ruby gems.

Jekyll is a Ruby gem that is a static-site generator (effectively providing templates for websites). Jekyll was written by a co-founder of GitHub in 2008. Jekyll is the engine behind GitHub Pages.

Jekyll works with Markdown and other markup languages like Textile.

A jekyll website lives in the "**docs**" folder of a GitHub repo.

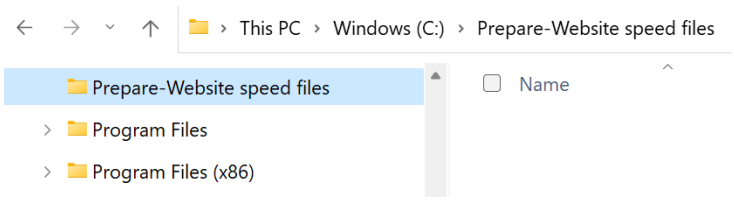
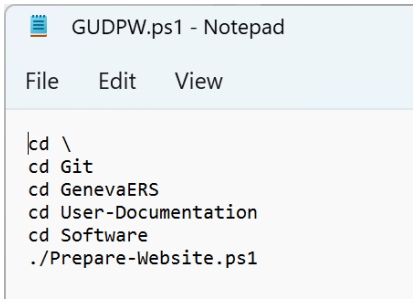
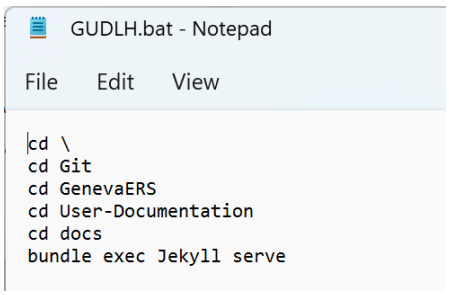
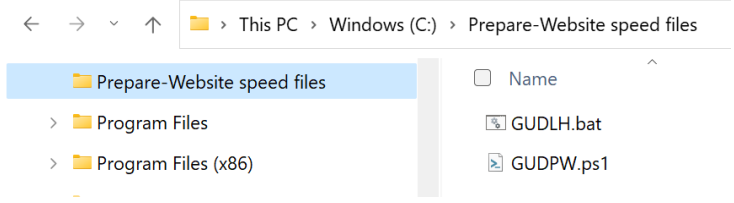
The file **Gemfile** is a key input to Jekyll to specify other gems needed. In rare cases this changes if new versions of gems are required.

The file **_config.yml** provides parameters to Jekyll.

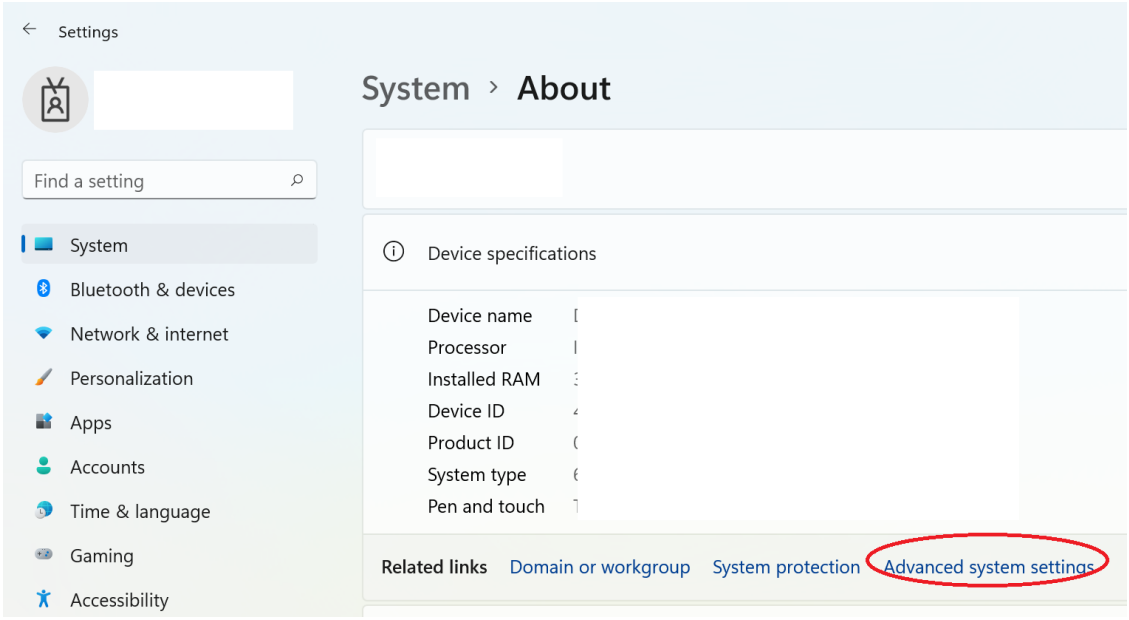
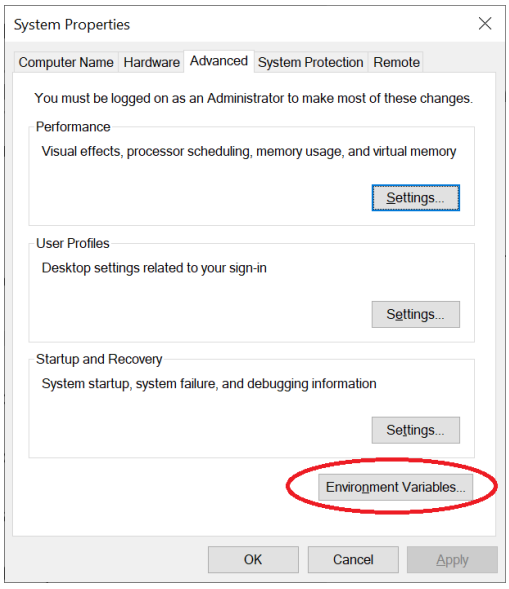
The file Gemfile.lock is effectively a system file – do not touch it.

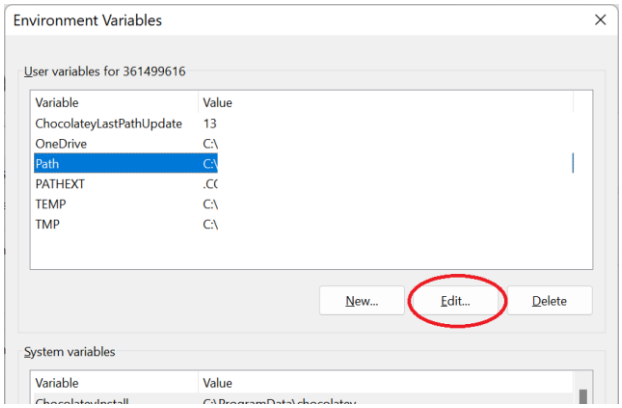
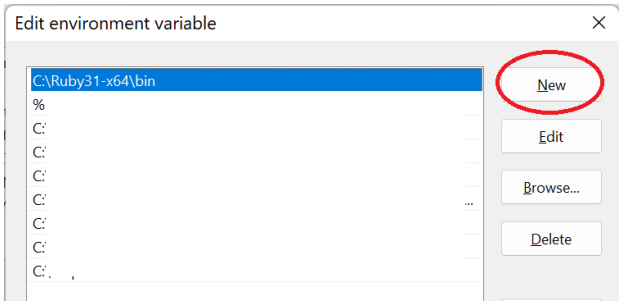
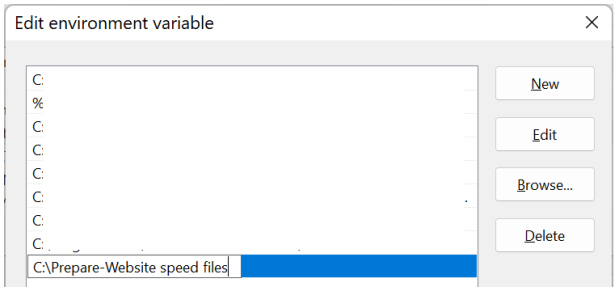
6 Setup for Prepare-Website (once only)

6.1 Once only – setup files to speed your work

Step	Action	Notes
1	Create folder **C:\Prepare-Website speed files**	
2	Copy files **GUDPW.ps1** **GUDLH.bat** as shown at right.	From: **Software** folder of GUD clone To: **C:\Prepare-Website speed files**
3	Edit the file at right. Edit with Notepad or any other editor.	**C:\ Prepare-Website speed files \GUDPW.ps1**
4	The cd commands need to apply to your computer, so that they navigate to the **Software** folder. Change as necessary and **Save**.	
5	Edit the file at right. Edit with Notepad or any other editor.	**C:\ Prepare-Website speed files \GUDLH.ps1**
6	The cd commands need to apply to your computer, so that they navigate to the **Software** folder. Change as necessary and **Save**.	
7	Files are setup in **C:\Prepare-Website speed files**	

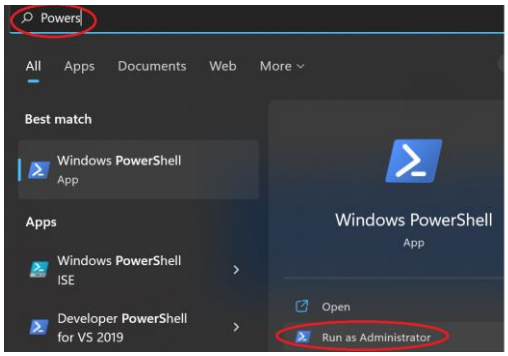
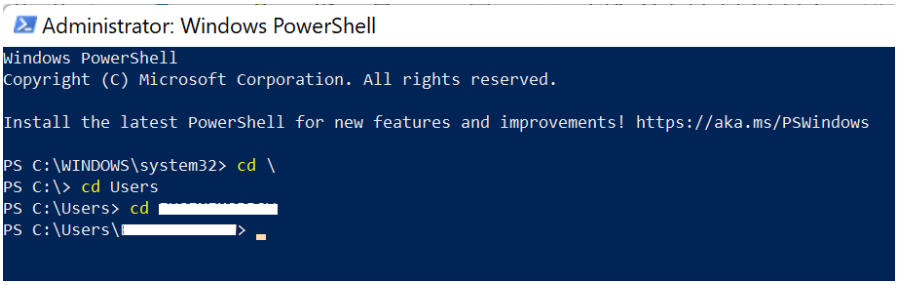
6.2 Once only – update path environment variable

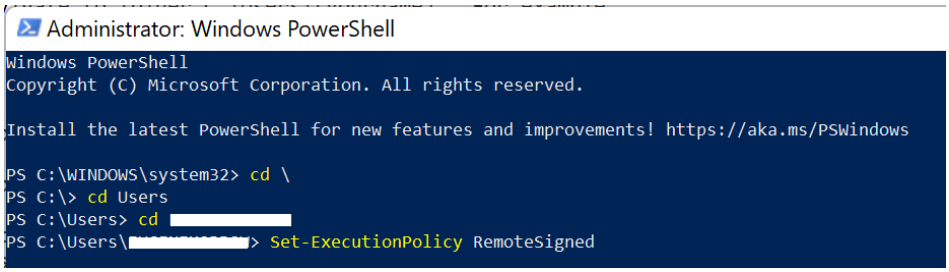
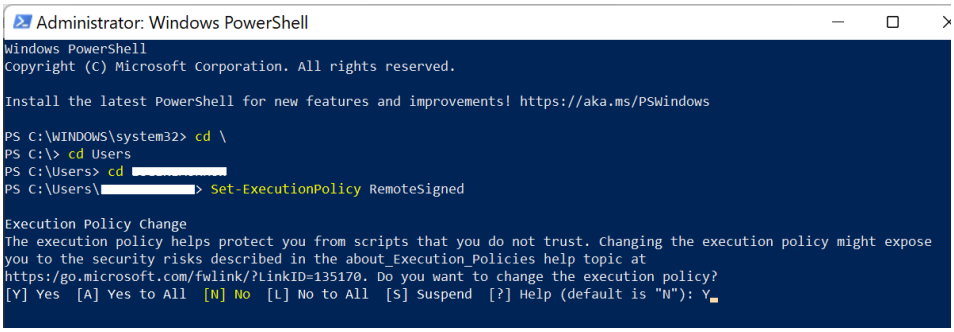
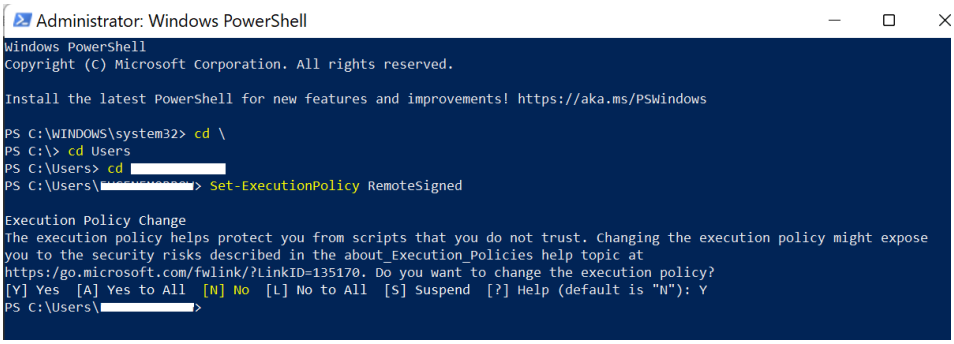
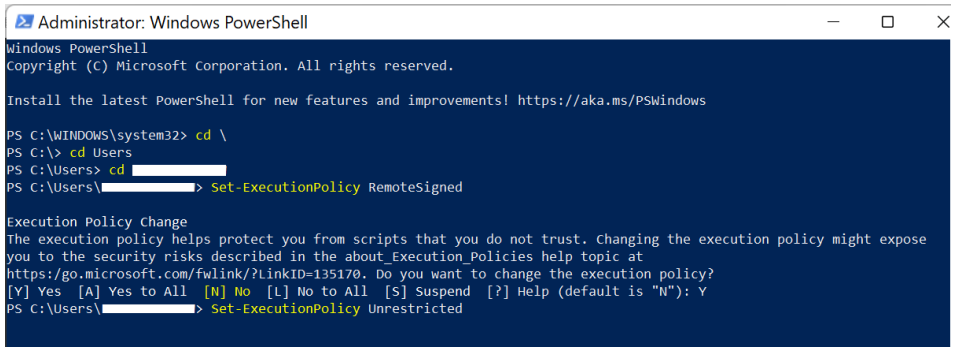
Step	Action	Notes
1	Right click **Start** icon, select **System**	
2	Click **Advanced system settings** 	
3	Click **Environment Variables** 	

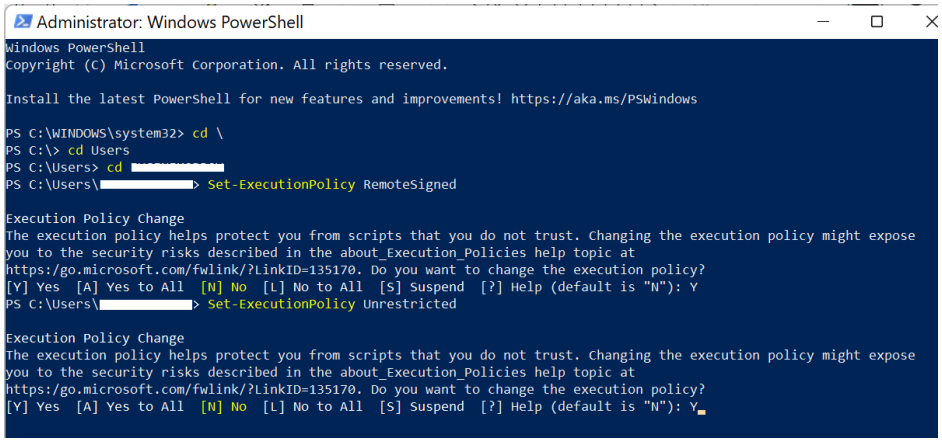
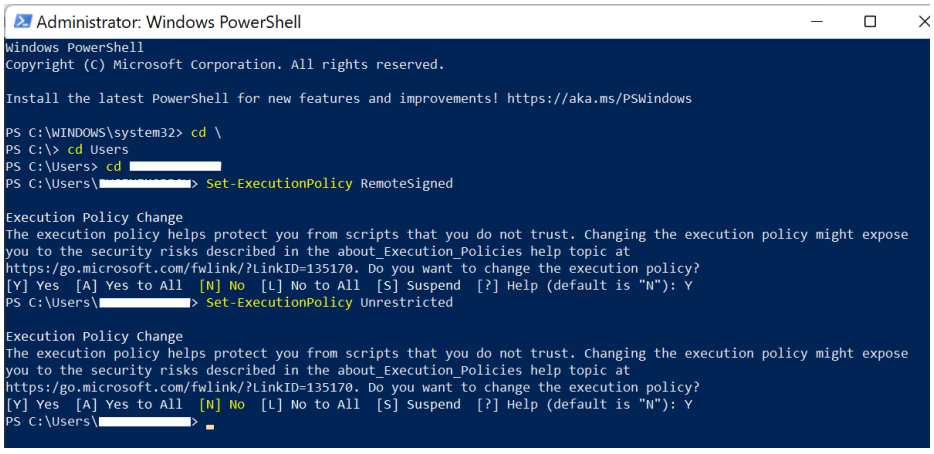
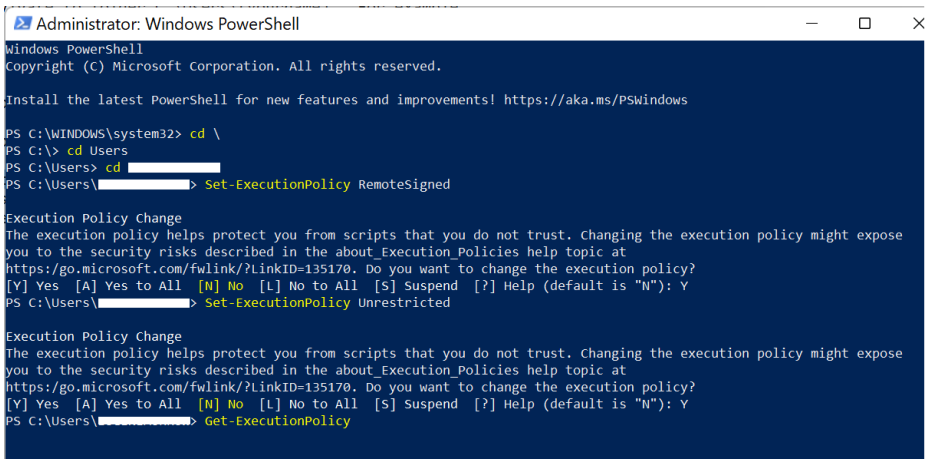
Step	Action	Notes
4	Under **User variables,** Highlight **Path** Click **Edit**	
5	Click **New**	
6	Type **C:\Prepare-Website speed files** Press **Enter.** Click **OK.**	
7	The path is stored. Click **Edit** to check the path. When done, click **OK** twice to exit.	

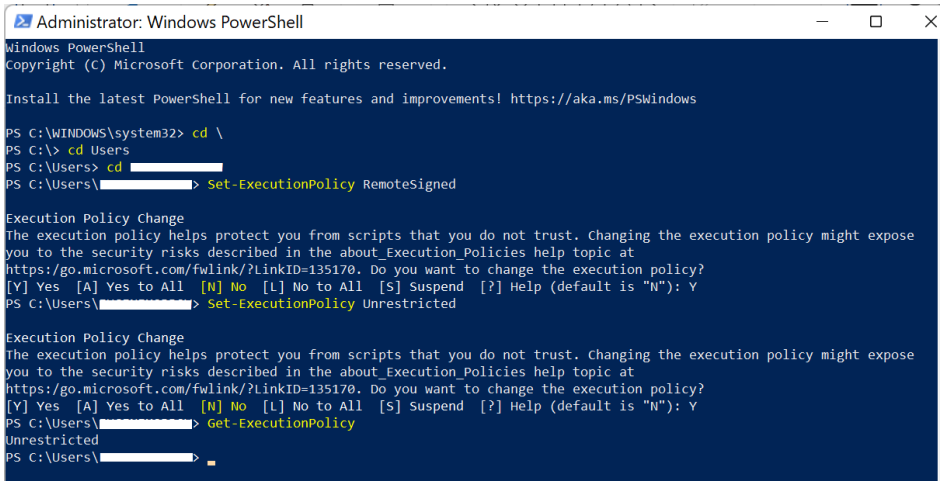
6.3 Once only – set policy for non-native Windows

This applies for Windows running as a guest operating system on some other platform – e.g. Linux or Mac.

Step	Action	Notes
1	Start **PowerShell** Search on "Powers", Under **Windows PowerShell App** at right select **Run as Administrator**	
2	The current folder is **C:\Windows\System** 	
3	The current folder should be **C:\Users\(yourname)** Navigate there, e.g. **cd ** **cd Users** **cd yourname** 	

Step	Action	Notes
4	Type **Set-ExecutionPolicy RemoteSigned** 	
	Press **Y** 	
5	This policy is done. 	
6	Type **Set-ExecutionPolicy Unrestricted** 	

Step	Action	Notes
7	Press **Y** 	
8	This policy is done. 	
9	Type **Get-ExecutionPolicy** 	

Step	Action	Notes
10	This shows the latest policy set: Unrestricted. This is a successful result. 
11	Close the Admin PowerShell window.	

7 How to be a GUD contributor

Perform the steps below **strictly in the order given**. Your changes may seem to have disappeared if the order is not right.

Your computer needs a GUD clone - see **Section *16 Appendix D: Create GUD clone***

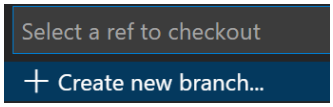
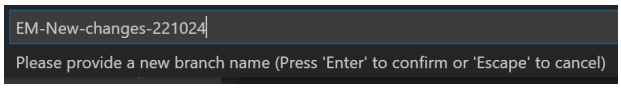
Step	Action	Notes
1	Ensure markdown is placed as follows: 1. **Topics** in folder structure under repo folder **All Markdown Topics** 2. **Common modules** in folder structure under repo folder **Common Modules**. 3. **Images** in folder **All Markdown Topics\images**	For more see **Section 8 *How to write Topics*** **Section *9 How to write Common Modules***
2	Run **Prepare-Website** This is always necessary even if you do not use includes yourself.	For more see **Section *10 Run Prepare-Website***
3	**Recommended: view website on localhost.** See website changes look on our computer before a commit to GitHub.	For more see **Section *11 View website on localhost***
4	**Commit and merge** changes to GitHub.	For more see **Section *12 Commit & merge changes to GUD***
5	View on **GitHub Pages**.	**https://genevaers.github.io/User-Documentation/**

8 How to write Topics

8.1 Create a new branch in VSCode (if required)

The current branch displays at bottom left of VSCode. If the current branch is master, then it is a good idea to create a new branch for the changes you will make (new or updated topics and/or common modules.)

You may already have done this, so ignore this section in that case.

Step	Action	Notes
1	Click in Status bar at bottom left.	
2	Click **Create new branch..**	
3	Type a name for a new branch for your changes. Start with your initials and put dashes instead of spaces.	
4	Check new branch shows in status bar at bottom left.	

8.2 Topic name and structure

Two topics can have the same name if they are in different folders. However, **a topic and a common module cannot have the same name**. This will generate an error message.

Create your topic (say A B C.md) and ensure the following structure:

```
---
layout: default
title: "A B C"
parent: Parent Folder
grand_parent: Grand Parent Folder
nav_order: 01
---
# A B C
{: .no_toc}
TABLE OF CONTENTS
1. TOC
{:toc}
```

Front Matter - must be unique combination of title, parent and grand_parent.

Title displayed inside this topic but not in the TOC for this topic.

TOC for this topic.

The name of topics and folders can have spaces (because this is Windows using Jekyll).

Copy and paste text from an existing topic.

The Front Matter fields are:

Front Matter	Notes
layout	Use "**default**" for most topics. (Use "**home**" for **index.md** in **All Markdown Topics**.)
title	Title displayed on the website.
parent	Folder that contains the topic. **Only set if Level 2 or Level 3 Topic.**
grand_parent	Folder above parent. **Only set if Level 3 Topic.**
nav_order	Relative position in Table of Contents for parent folder.

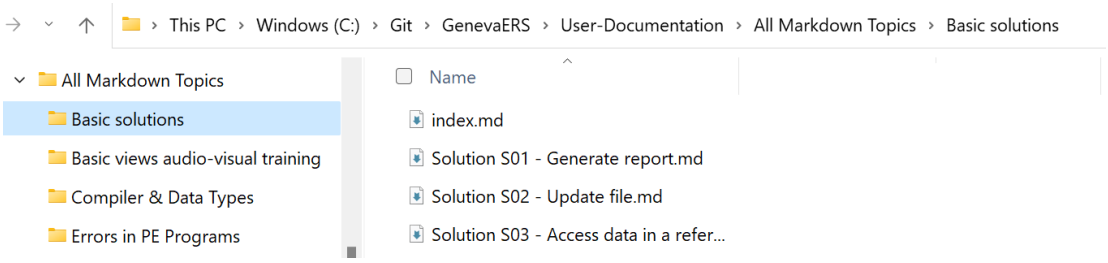
Place your topic either in folder **All Markdown Topics** or in a sub-folder (can be two levels below All Markdown Topics). See existing topics to get the idea of the typical markdown coding used.

When all changes to topics and common modules are complete, see step 2 in **Section *7 How to be a GUD contributor.***

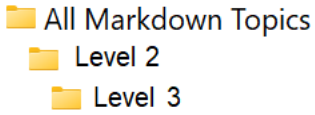
8.3 Folder levels for Topics

Topics are stored in a folder under "**All Markdown Topics**". In rare cases a topic can be that folder itself, but most topics are in a folder somewhere under "All Markdown Topics".

"**Basic solutions**" is at Level 2. This is one level below folder "**All Markdown Topics**"



This website can have folders to Level 3, as shown below:



The Level 3 folder is hence **All Markdown Topics\Level 2\Level 3**

Examples in GUD are:

All Markdown Topics\Lessons in basic skills\Lesson A - Simple report

All Markdown Topics\Workbench Logic Text Syntax\Syntax - functions

8.4 Link to a topic in the same folder

You are in **Topic A.md** in folder **Folder01** that is Level 2 or 3 (under **All Markdown Topics**).

You want to link to Topic B.md in the same folder then here is the coding for Topic A:

See this topic: **Topic B**

The **square brackets are the text that the user sees to click on.** That text does not have to be "Topic B", but most of the time that is the best value to use.

The **round brackets are the target for the link**. In this case Topic B.md is in the same folder.

Do not put ".md" because Jekyll is using html files by the time the website is running.

8.5 Link to a topic in a different folder

You are in **Topic A.md** in folder **FolderAA** (a Level 2 folder). You want to link to **Topic B.md** in another Level 2 folder called **FolderBB**. The coding for some line in Topic A is:

See this topic: **[Topic B](../FolderBB/Topic B)**

The **"../"** is the navigation to go up from Folder01 to All Markdown Topics. The rest directs the link to Topic B in FolderBB.

If **FolderAA** is a Level 3 folder, but FolderBB is still a Level 2 folder, then the coding is:

See this topic: **[Topic B](../../FolderBB/Topic B)**

Above the extra "../" navigates up the extra level .

If these things are true:

1. **FolderAA** and **FolderBB** are both Level 3 folders and
2. **FolderBB** is under a Level 2 folder called **FolderXX**,

then the coding is:

See this topic: **[Topic B](../../FolderXX/FolderBB/Topic B)**

If these things are true:

1. **FolderAA** is Level 2 and
2. **Topic B.md** is in **"All Markdown Topics"**

then the coding is:

See this topic: **[Topic B](../Topic B)**

8.6 Link to an entire sub-folder

You are in **Topic A.md** in Level 2 folder **FolderAA**. You want to link to Level 2 folder FolderBB. Here is the coding for Topic A:

See topics: **[FolderBB](../FolderBB)**

When all changes to topics and common modules are complete, see step 2 in **Section *7 How to be a GUD contributor.***

8.7 Insert images in a topic

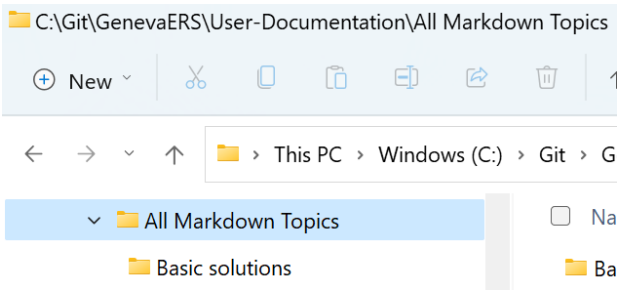
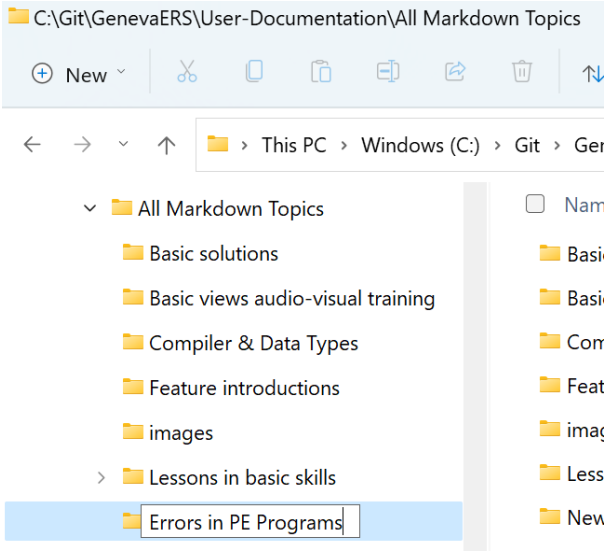
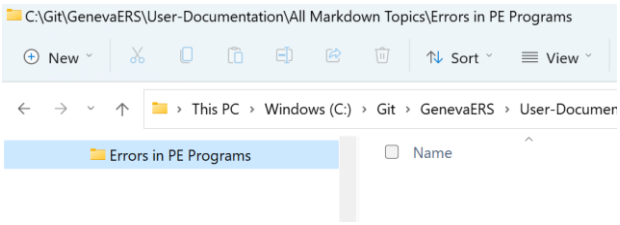
File **ABC.gif** is an image that is usefully part of at least one topic.

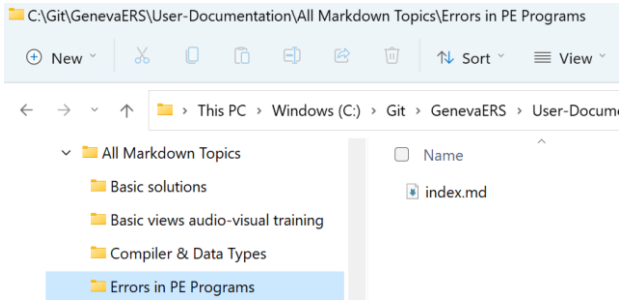
Here is what to do:

Step	Action	Notes
1	Store ABC.gif in the folder at right. That folder is part of the clone of GUD on your computer.	**All Markdown Topics\images**
2	A **topic in a Level 2 folder** uses this coding for the image: **![(image ABC)](../images/ABC.gif)** The **text between the square brackets** is only displayed if the file ABC.gif is unavailable. The text between the round brackets navigates from the Level 2 folder to the file ABC.gif.	
3	A **topic in a Level 3 folder** uses this coding for the image: **![(image ABC)](../../images/ABC.gif)**	
4	A **topic in the Level 1 folder (All Markdown Topics)** uses this coding for the image: **![(image ABC)](images/ABC.gif)**	

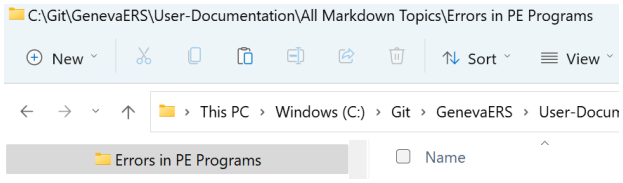
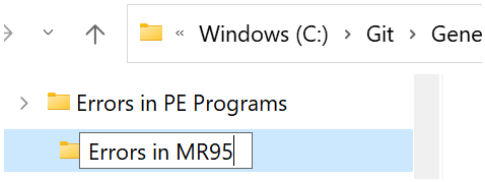
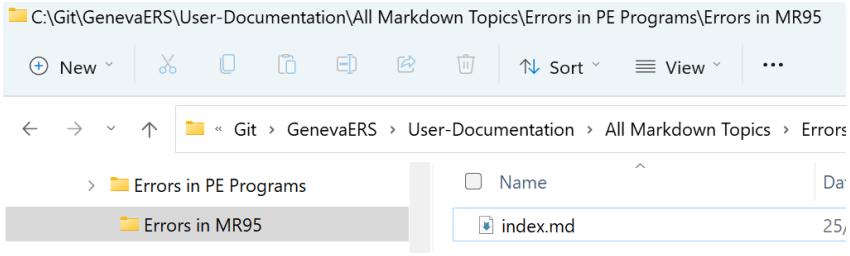
When all changes to topics and common modules are complete, see step 2 in **Section *7 How to be a GUD contributor.***

8.8 Create new Level 2 folder of topics

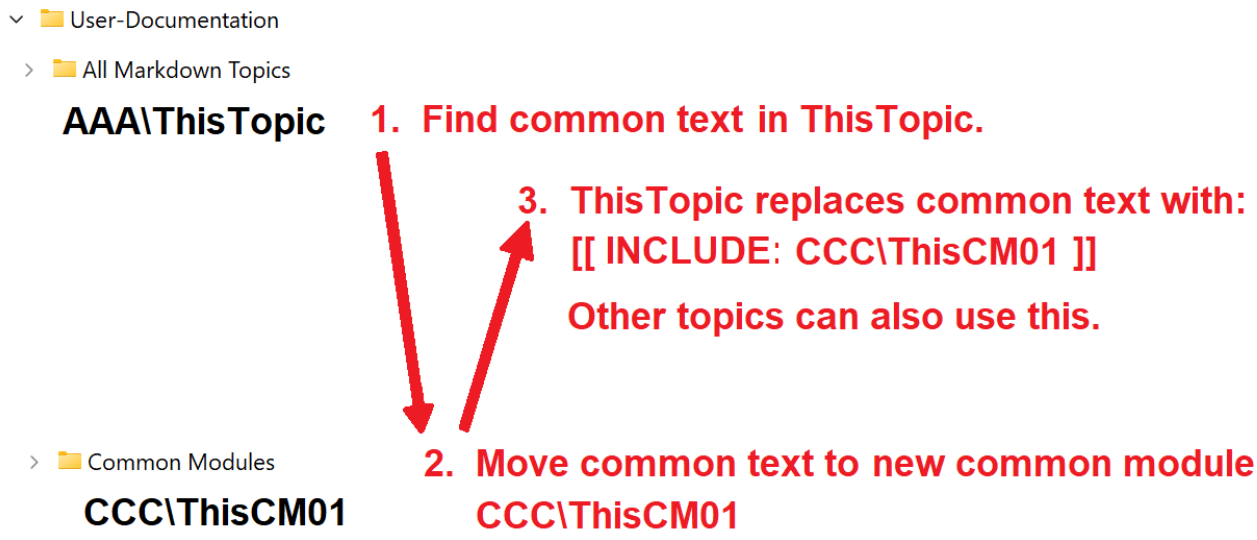
Step	Action	Notes
1	In Windows Explorer, go to folder **All Markdown Topics** in your clone. Typically **C:\Git\GenevaERS\User-Documentation\All Markdown Topics**	
2	Under **All Markdown Topics**, create new folder, e.g. **Errors in PE Programs**	
3	Go to the new folder (currently empty).	
4	Using Notepad or any other editor, create file **index.md** with data as at right. NOTE: field **has_children: true** tells Jekyll that this is a Table of Contents topic for the new folder. Field **nav_order** refers to the position of this folder in the list of folders on the left side of the website. Use a value after checking the "index.md" for other Level 2 folders. There is **NO need for the TABLE OF CONTENTS coding** in general topics – because has_children is true the TOC is automatically generated.	**---** **layout: default** **title: "Errors in PE Programs"** **nav_order: 60** **has_children: true** **---** **# Errors in PE Programs**

Step	Action	Notes
5	The new folder has one topic – **index.md** which is the **"Table of Contents" topic** for the new folder.	
6	You can now place general topics in this folder. Each general topic will have this line in the Front Matter: **parent: Errors in PE Programs**	This is different to the file index.md in this folder. The index.md file appears in the folder above – All Markdown Topics.
7	Recommended: update the home page topic to have a link to the new folder. The home page topic is "**index.md**" in the folder **All Markdown Topics**.	
8	Edit the home page topic. Scroll down and put in coding similar to right. This is a link to the new folder.	**## Errors** **Errors in PE Programs**
9	Save the home page topic.	

8.9 Create new Level 3 folder of topics

Step	Action	Notes
1	In Windows Explorer, in your clone go to the Level 2 folder that will get the new Level 3 folder.	
2	Create a new folder, e.g. **Errors in MR95**	
3	Go to the new folder (currently empty).	
4	Using Notepad or any other editor, create file **index.md** with data as at right. Field **title** is the name of the new folder. Field **parent** is the folder above – the Level 2 folder. Field **nav_order** refers to the position this index topic will appear in the Level 2 folder above. Set to a sensible value compared to the other topics in that Level 2 folder. Field **has_children: true** tells Jekyll that this is a Table of Contents topic for the new folder. There is **NO need for the TABLE OF CONTENTS coding** in general topics – because has_children is true the TOC is automatically generated.	**---** **layout: default** **title: "Errors in MR95"** **parent: Errors in PE Programs** **nav_order: 01** **has_children: true** **---** **# Errors in MR95**
5	The new folder has one topic – **index.md** which is the **"Table of Contents" topic** for the new folder. 	
6	You can now place general topics in this folder. Each general topic will have these lines in the Front Matter: **parent: Errors in MR95** **grand_parent: Errors in PE Programs**	This is different to the file index.md in this folder. The index.md file appears in the folder above – the Level 2 folder.

8.10 How to include stuff



8.11 Include coding in a topic

You have already created file **Common Modules\ComFolder01\Module01.md**

Step	Action	Notes
1	**Open the topic** (or common module) that needs the include shown above.	
2	On the relevant line, type: **[[INCLUDE: ComFolder01\Module01**]] **NOTES:** • Each line of include coding like this is a line by itself. Nothing else on the line (except spaces). • **INCLUDE:** must be in capitals. • Begin "**[[INCLUDE:**" or "**[[INCLUDE:**" • End with **"]]"** (one space before) • Any number of spaces before [[• Any number of spaces after]]. • "**ComFolder01**" is a folder under folder "**Common Modules**" in your clone. There could be two or more folders, e.g. **ComFolder01\ComFolder02\Module01** • **Module01** is a markdown file in the given folder. No need to put "**.md**" (this is assumed). • There may be up to 25 lines each with an include. Such lines can be scattered through your file.	
3	**Save your topic** or (common file).	
4	When all updates are done, proceed to the step at right.	**Step 2 in :** **Section** ***7 How to be a GUD contributor***

8.12 Examples of includes

Examples of includes can be seen in these topics:

1. All Markdown Topics\Workbench Logic Text Syntax\Syntax - functions\ Syntax - function BATCHDATE.md

2. Common Modules\Logic Text\Consolidations\Consolidation Function BATCHDATE Level 3 .md

When all changes to topics and common modules are complete, see step 2 in **Section *7 How to be a GUD contributor.***

8.13 Finer points of markdown

8.13.1 Display some code

Want to display some code (like Assembler)?

Use "**<pre>**" before the code and "**</pre>**" after the code.

This is html coding which can be used in markdown.

For example, see **Common Modules\Logic Text\Descriptions\Description Function PRIOR.md**. The coding is:

PRIOR is typed before the name of the field, for example:

<pre> PRIOR {product_code} </pre>

The result is:

PRIOR is typed before the name of the field, for example:

```
PRIOR {product_code}
```

This text can be seen in the GUD website in topic **Syntax – function PRIOR.md**
in folder **Workbench Logic Text\Syntax – functions.**

This topic is seen at URL https://genevaers.github.io/User-Documentation/Workbench%20Logic%20Text%20Syntax/Syntax%20-%20functions/Syntax%20-%20function%20PRIOR.html

8.13.2 Control text in a table

Text in a cell in a table looks better when you can control these things:

1. Going to new line in that cell
2. Indenting some text in that cell

Create a new line with **
** in your table text.

Indent one space with ** ** in your table text. (The letters nbsp = non blank space?)

This coding ONLY WORKS IN A TABLE.

For example, see **Common Modules\Logic Text\Examples\Example Function PRIOR ECA.md**. The coding is this:

```
|Example logic text|Meaning|
|------------------|-------|
|**IF (CURRENT({field2}) <> PRIOR({field2}))<br>    THEN COLUMN = "PRODUCT: "<br>        ELSE COLUMN = "<br>ENDIF**|If the current record has a different value of field2 from<br>the previous record, set the current column to "PRODUCT: "<br>otherwise set the current column to blank.<br>This assumes the input file is sorted into field2 order.|
```

The result is:

Example logic text	Meaning

IF (CURRENT({field2}) <> PRIOR({field2}))
THEN COLUMN = "PRODUCT: "
ELSE COLUMN = " "
ENDIF
	If the current record has a different value of field2 from
	the previous record, set the current column to "PRODUCT: "
	otherwise set the current column to blank.
	This assumes the input file is sorted into field2 order.

This text can be seen in the GUD website in topic **Syntax – function PRIOR.md** in folder **Workbench Logic Text\Syntax – functions.**

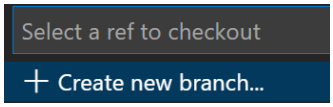
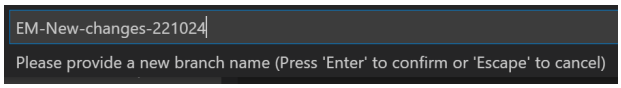
This topic is seen at URL https://genevaers.github.io/User-Documentation/Workbench%20Logic%20Text%20Syntax/Syntax%20-%20functions/Syntax%20-%20function%20PRIOR.html

9 How to write Common Modules

9.1 Create a new branch in VSCode (if required)

The current branch displays at bottom left of VSCode. If the current branch is master, then it is a good idea to create a new branch for the changes you will make (new or updated topics and/or common modules.)

You may already have done this, so ignore this section in that case.

Step	Action	Notes
5	Click in Status bar at bottom left.	
6	Click **Create new branch..**	
7	Type a name for a new branch for your changes. Start with your initials and put dashes instead of spaces.	
8	Check new branch shows in status bar at bottom left.	

9.2 Just Markdown, that's all

Common modules do **NOT have Front Matter** (see **Section *8.2***)

Why? Front Matter is unique to a topic. A common module may be included in many topics, so if a common module has Front Matter it would overwrite Front Matter in all the topics that include it.

Common Modules do **NOT have any Table of Contents coding** (see **Section *8.2)***. This is because a topic creates its own Table of Contents, not any common modules being included.

So a common module has markdown, that's all. The common module has only data that can be included in topics. There can be headings and paragraphs and tables and images and links to other topics.

For example, see common module:

Common Modules\FAQ\Keyboard Shortcuts All WB Screens Common.md

This file contains a large markdown table giving details of the keyboard shortcuts available on (nearly) all workbench screens. The word "Common" in the name ensures that the common module and the topic that includes it have different names (see large box lower on this page).

A common module can have only one line. For example, file:

Common Modules\Logic Text\Syntax diagrams\Syntax Diag Function ALL Level 3.md.

That file has just one line:

![(Function ALL)](../../images/LTSF_ALL_01.gif)

(LTSF stands for Logic Text Syntax Function. The image is a syntax diagram of function ALL.)

This common modules consists of one line: an image reference. The "Level 3" in the name means this common module needs to be included by a topic at Level 3 (see **Section *9.5 Common modules and folder levels***).

This is still a useful include. This file is included in more than one topic. In future, it is planned to replace the image with markdown coding. When that happens, the change occurs in one place – (this common module) rather than in all the topics that refer to this common module.

Common modules must have **unique names** (amongst all common modules). As well, common modules **cannot have the same name as any Topic**. Both these situations generate an error message.

When all changes to topics and common modules are complete, see step 2 in **Section *7 How to be a GUD contributor.***

9.3 A common module can include a common module

A common module include another common module.

Why? This allows common modules to be grouped together in some situations and treated separately on others. This is useful when there are 100 or more includes.

Note that this option has some challenges. The program Prepare-Website meets the following challenges:

1. If Common Module A includes B which includes C which includes A, then there is a potential that a **circular include**. Program Prepare-Website gives an error if this is found and stops processing.
2. If Common Module A includes B which includes C, then we have to action the includes in the reverse order. This means that B must include C first before A includes B. Program Prepare-Website ensures **includes are performed in the correct order.**

9.4 How to code a common module

This topic covers how to create a common module. How a topic or a common module can include that common module is covered in **Section *8.10 How to include stuff.***

Create a common module as below:

Step	Action	Notes
1	Choose a folder under the folder "**Common Modules**" in the clone on your computer.	The folder can be two or more levels deep. It is recommended to keep a simple structure with just one level of folders below folder **Common Modules**.
2	Create a markdown file – e.g. at right.	**C123.md**
3	Type markdown coding. Care needs to be taken with **links and images**. If this applies to your common module, see **Section *9.5 Common modules and folder levels***.	
4	To allow a topic or other common module to include C123.md see the section at right.	**Section *8.10 How to include stuff***

9.5 Common modules and folder levels

Both **markdown images and links** are affected by the level of the folder for the topic. Levels are explained in **Section *8.3 Folder levels for Topics.***

Is the topic at Level 1, Level 2 or Level 3?

If a topic is a certain level, then the common module needs the same coding for that level. The coding for each level is shown in these sections:

Section *8.4 Link to a topic in the same folder*

Section *8.5 Link to a topic in a different folder*

Section *8.6 Link to an entire sub-folder*

Section *8.7 Insert images in a topic*

How to solve this challenge when a common module has images and/or links?

The solution is that there may be different version of your common module – one for Level 1, one for Level 2 and one for Level 3.

For example, if your common module **C123.md** has images and/or links, then you need up to three files:

C123 Level 1.md

C123 Level 2.md

C123 Level 3.md

"Level 3" in the name means "Topics at Level 3 can include this file".

If the only topics that require this common module are at Level 3, then you only need one file. It is recommended that you still name it "C123 Level 3.md". This is because later there may be a need for the Level 2 and Level 1 file versions.

An example of this can be seen in file:
Common Modules\Logic Text\Syntax diagrams\Syntax Diag Function ALL Level 3.md.

That file has just one line:

![(Function ALL)](../../images/LTSF_ALL_01.gif)

Notice that the coding has "../../" which is relevant to a topic at Level 3.

9.6 When all common modules and includes are complete

How a topic or a common module can include a common module is covered in **Section *8.10 How to include stuff.***

When all changes to topics and common modules and include coding is complete, see step 2 in **Section *7 How to be a GUD contributor.***

10 Run Prepare-Website

Do this section every time changes to GUD are finished. This is necessary

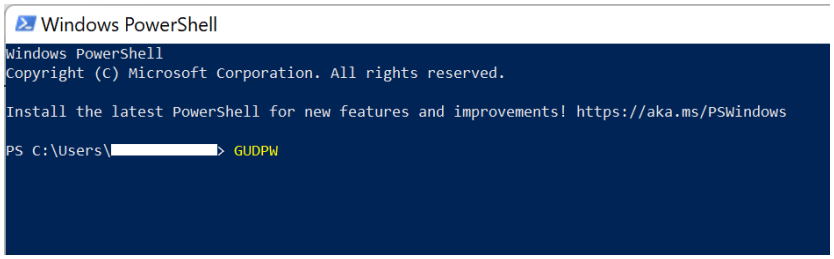
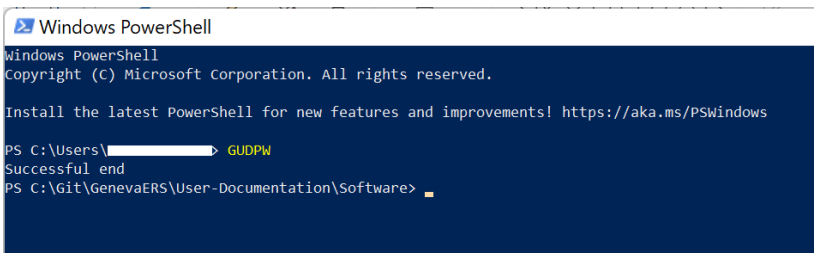
BEFORE testing the changes locally on your computer, and

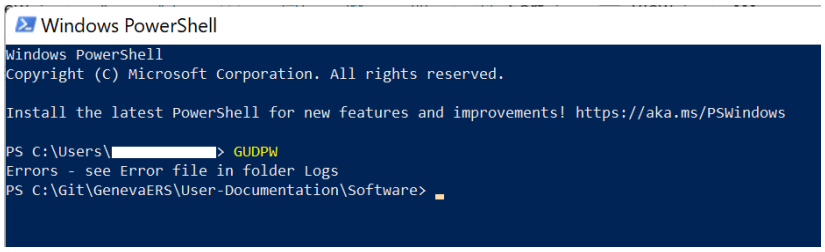
BEFORE commit & merge of the changes to GitHub.

The program is **Prepare-Website.ps1** in repo folder Software.

You have already done all relevant actions in **Section *6 Setup for Prepare-Website***.

To run the program, do the following:

Step	Action	Notes
1	Start PowerShell	Search on "powers", select Windows PowerShell App There is no need to have Admin rights this time.
2	Type "**GUDPW**" and press enter. (This stands for GUD Prepare-Website.) It takes around 20 seconds. 	
3	**If you see "Successful end",** you can see useful information and statistics in the file **REPORT Prepare-Website.txt** in the Logs folder. You can now proceed to the next section. 	

Step	Action	Notes
4	**<u>If you see "Errors – see Error file in folder Logs",</u>** go to the next step. 	
5	Go to **Logs** folder in your clone. View the file **ERRORS Prepare-Websitea.txt**. Fix the problems by changing files in folders All Markdown Topics and Common Modules as required. The files **REPORT Prepare-Website.txt** and **TRACE Prepare-Website.txt** can provide extra information that may sometimes be useful to diagnose the problems. When problem are fixed, return to this section to rerun Prepare-Website.	

11 View website on localhost

11.1 Run Jekyll serve

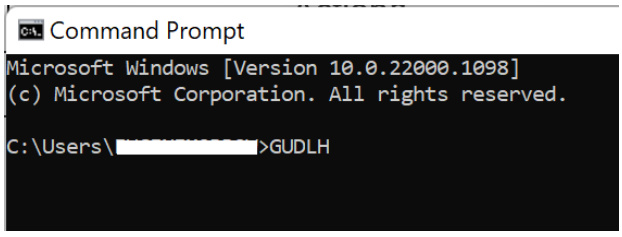
Pre-requisites:

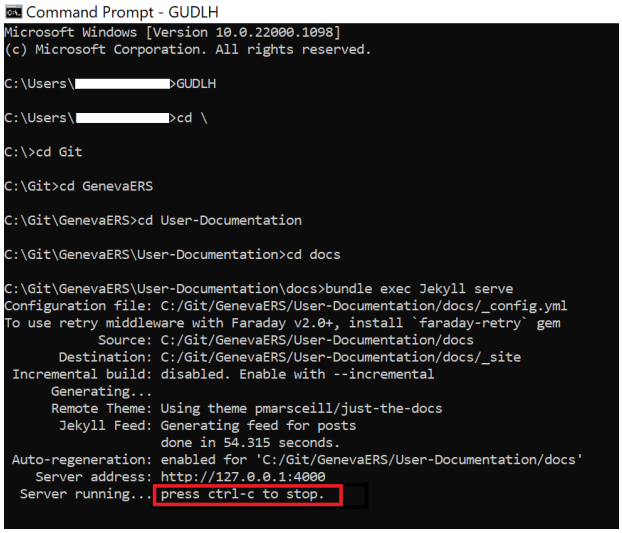
(1) Already installed Ruby & Jekyll (see Section 5).

(2) Already run “Prepare-Website” (see previous section).

While running localhost, did you **make changes and run Prepare-Website again?**

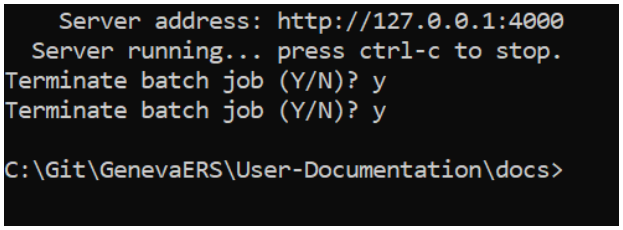
If so, **close the command prompt for localhost** and do the steps below once again. Localhost cannot cope with running Prepare-Website at the same time.

Step	Action	Notes
1	Start a command prompt.	Start, Windows System, Command Prompt
2	Type **GUDLH**. (This stands for GUD LocalHost.)	
3	Wait 75 seconds.	

Step	Action	Notes
4	If you see this: **press ctrl-c to stop.** then **<u>go to next section</u>** to view the website locally on your computer. If there is something different, see the next step below.	
5	You may see a message about needing to run "**bundle update**". If you do see this then go to the next step.	
6	**<u>Press Ctrl-c and then type y enter. Press y enter again.</u>** This stops the localhost process (Jekyll serve). Type "bundle update". This can take 20 minutes to run. There are no shortcuts. Let the bundle update run to completion. When complete, you can run GUDLH again (when required).	This updates your local jekyll environment. This allows you to run the localhost process successfully.

11.2 View Jekyll site locally

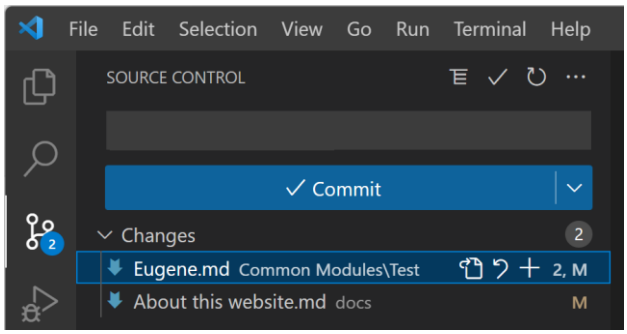
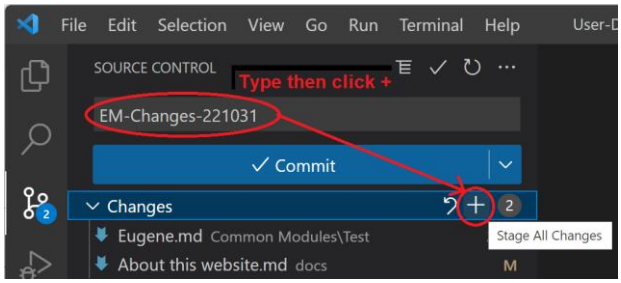
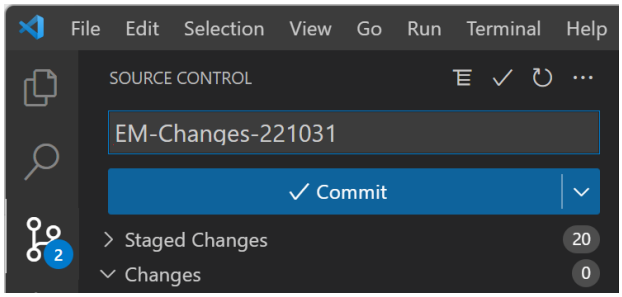
You can see how your changes will look on the website, without committing the change to the actual GUD website on GitHub.

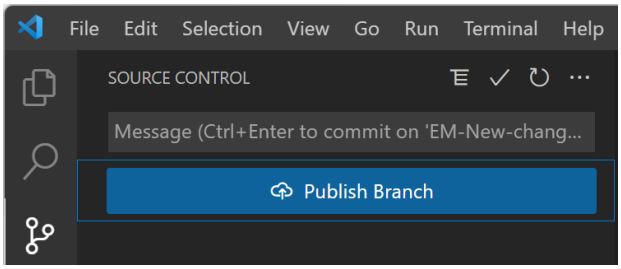
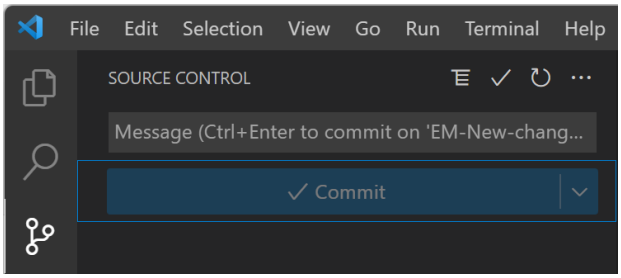
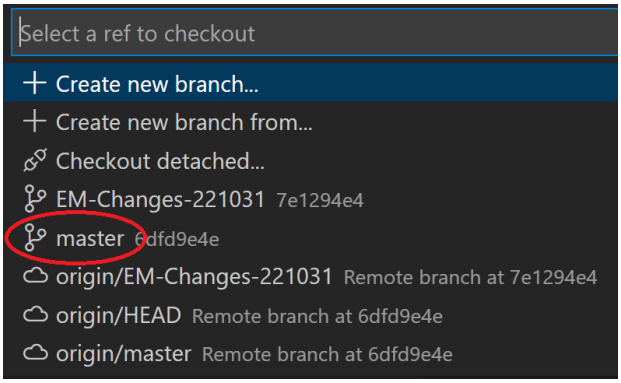
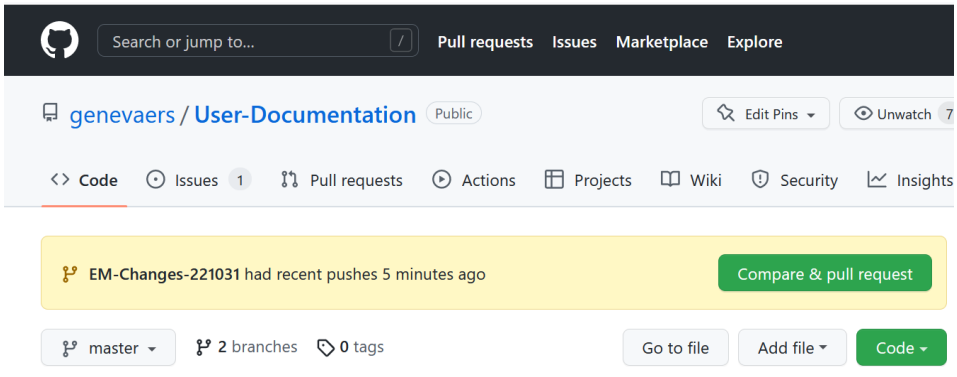
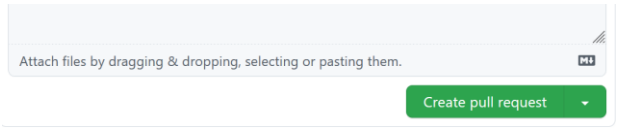
Step	Action	Notes
1	Go to URL: http://localhost:4000/ NOTE: URL ends with **SLASH /**	
2	Your updated website appears.	
3	Review the website and your changes. If you can't see your changes, check: 1 Changes are in the folder structures at right and 2 You ran Prepare-Website. When done looking at your website, close the tab on the browser.	**All Markdown Topics** **Common Modules**
4	**CAN'T FIND YOUR CHANGES?** There are several possible reasons. • **Forgot to run Prepare-Website?** See ***Section 10 Run Prepare-Website*** • **Were there errors reported when you ran Prepare-Website?** These must be fixed before that program will update the website. • **Have you changed the branch in Visual Studio Code?** If the changes were made to the existing clone (master branch) and have **not been staged into a branch yet**, then ensure Visual Studio Code is pointing to the **master** branch. If you have already staged the changes into a branch (but not yet merged), then ensure that Visual Studio Code is pointing to new branch.	
5	To end the local host viewing do the following: • Close the tab in the browser • Back at the command prompt press **ctrl-c** • Press **y** and enter • Press **y** and enter • Close the command prompt.	

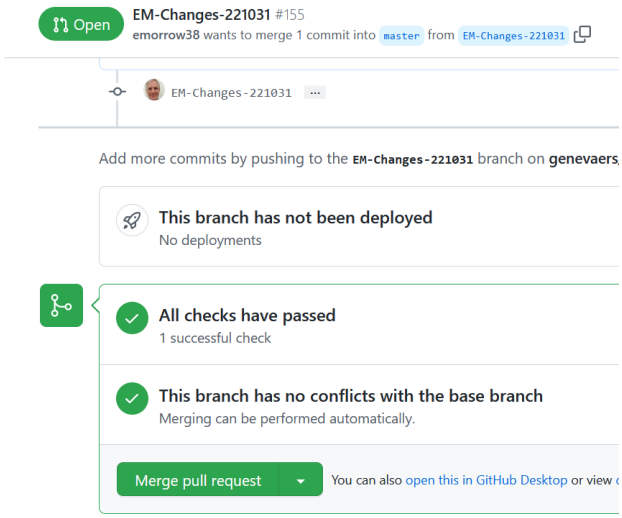
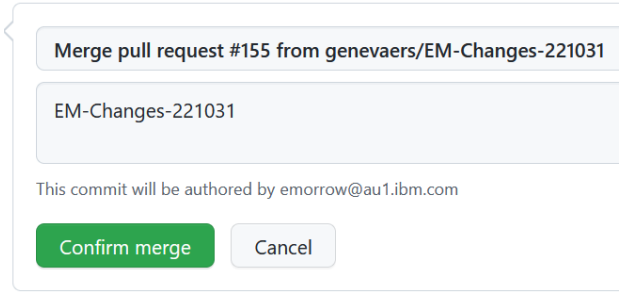
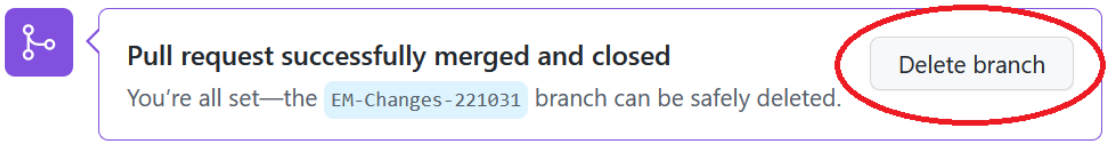
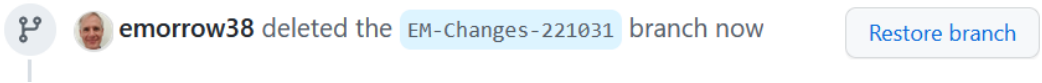
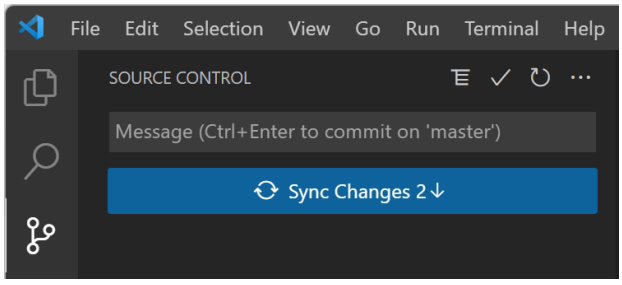
12 Commit & merge changes to GUD

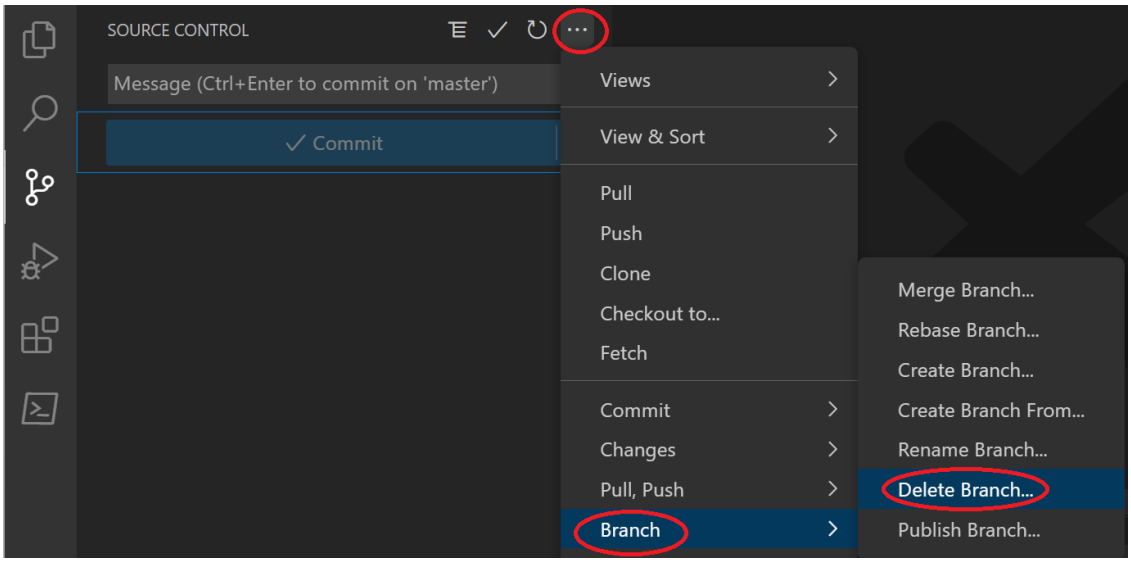
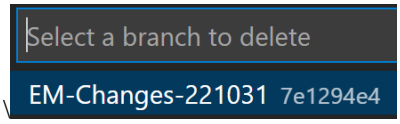
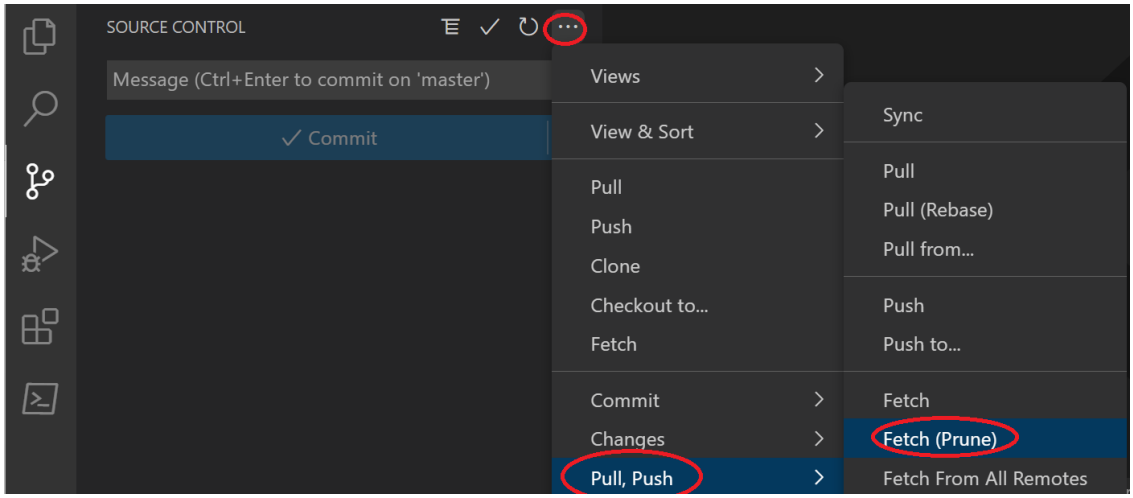
In this example, Eugene changed only one file: **Eugene.md** which is a common module in folder **Test**.

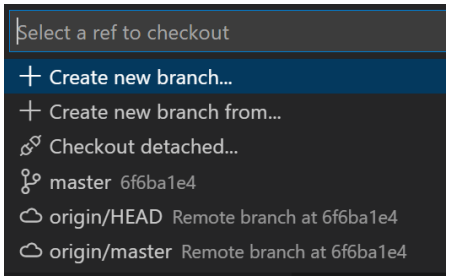
Running **Prepare-Website** means that one topic is affected: **About this website.md** which includes Eugene.md. So the changes are one common module and one topic.

Step	Action	Notes
1	Start **Visual Studio Code**	
2	Open the folder for the GUD clone.	
3	The current branch displays at bottom left in the status bar. An example is at right. If the branch is **master**, then use **Section *8.1*** to create a new one.	
4	Ensure you are in Source Notice how there are two changes listed: the common module and the topic. Eugene only changed the common module, and Prepare-Website changed the topic. So there are two files to provide to GitHub.	
5	Ensure you have typed a message above the Commit button. Click **+ to Stage All Changes.** The + is to the right of **"Changes"**	
6	Your changes are now staged.	
7	Click **Commit**.	

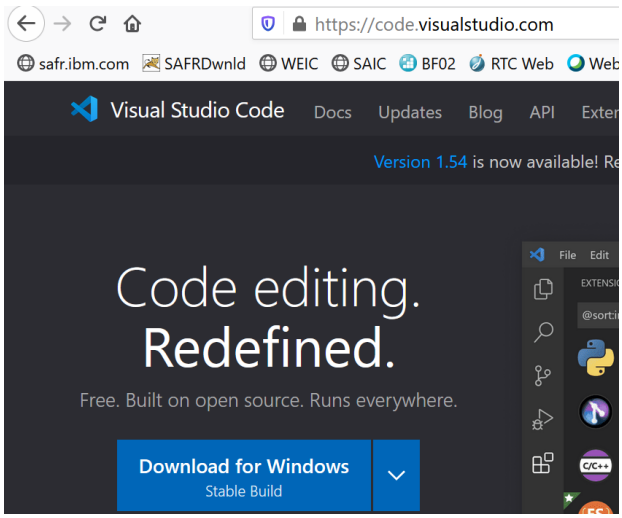
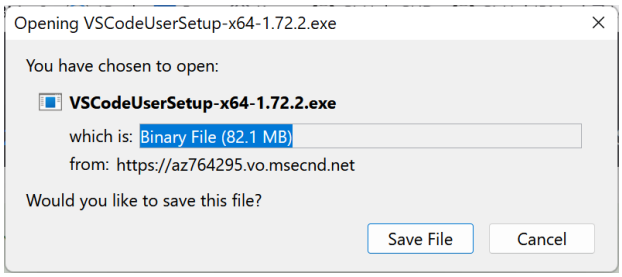
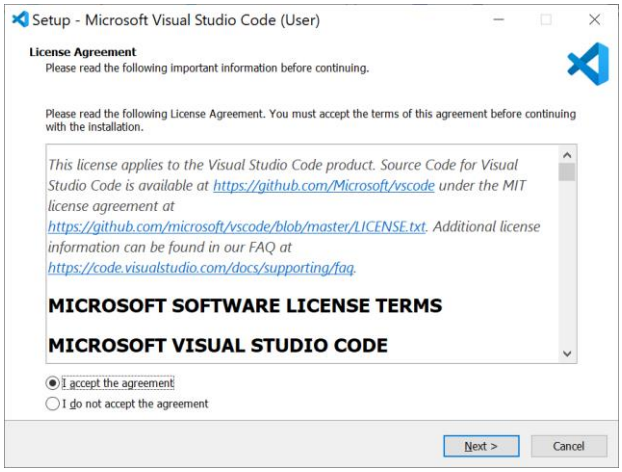
Step	Action	Notes
8	Click **Publish Branch**	
9	Notice that the source icon no longer has a number associated with it. This is normal.	
10	Click in Status bar at bottom left.	
11	Select **master** branch This is because when we return to VSC we will have deleted the new branch, and so it is better to be in branch **master** that is still there.	
12	In a browser, go to URL at right.	https://github.com/genevaers/User-Documentation
13	You must be logged into GitHub and have authority to commit to the GenevaERS User-Documentation repo. Click **Compare & pull request** 	
14	Scroll down a little and click **Create pull request.**	

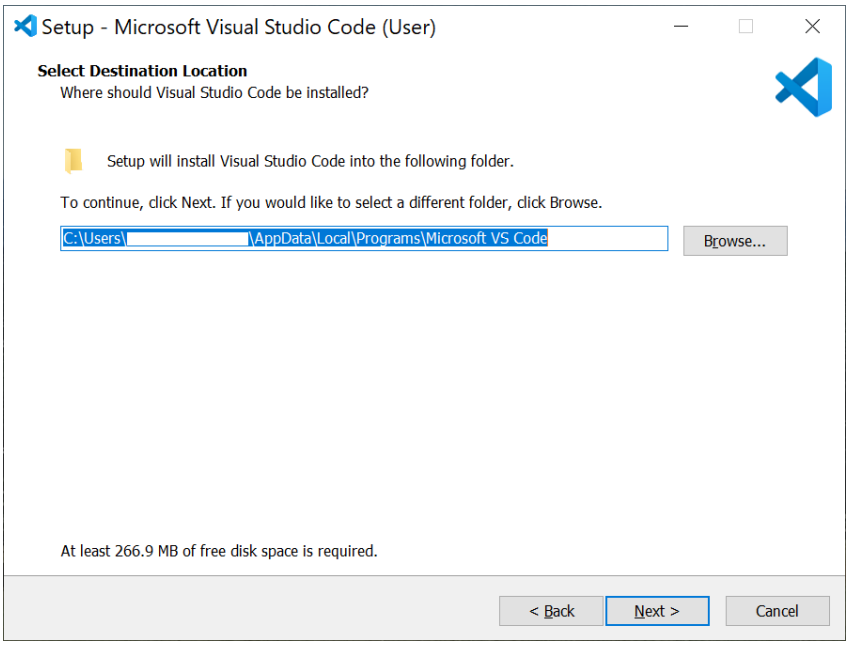
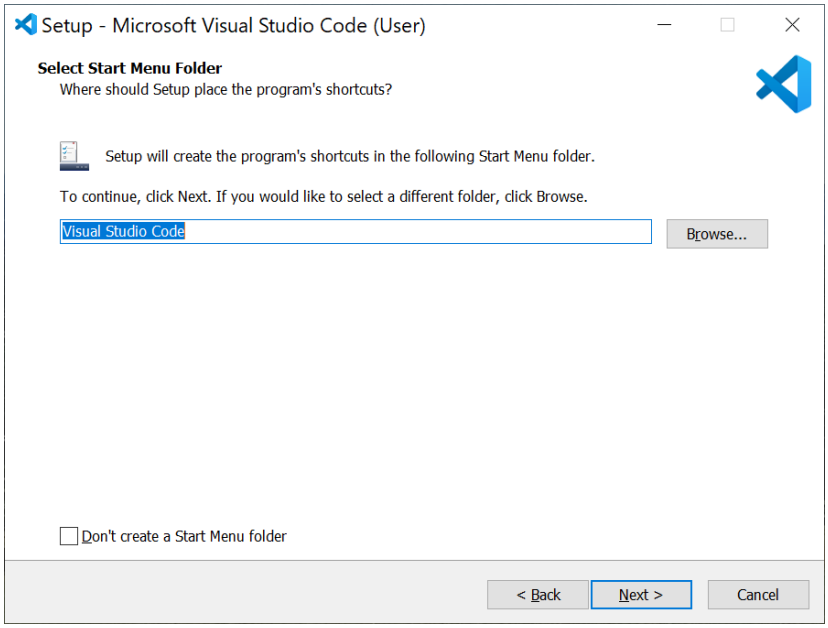
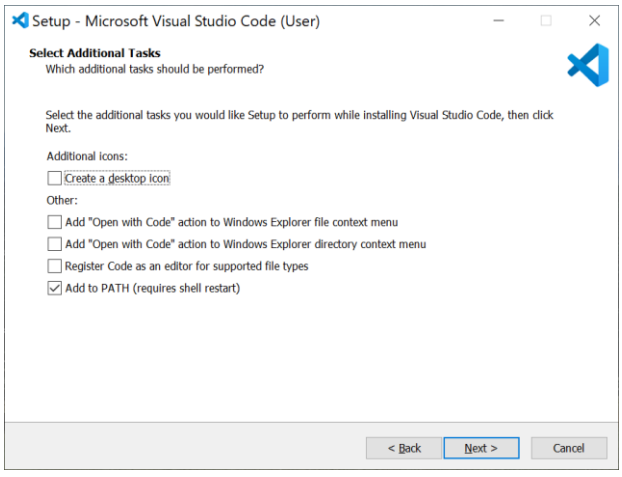
Step	Action	Notes
15	Scroll down a little and click **Merge pull request.**	
16	Click **Confirm merge**	
17	Click **Delete branch**. This is normal because the branch is now merged into master and may suggest the opposite – that it is not merged into master. So deleting the branch is the best idea. 	
18	The branch is now merged into master and then deleted. 	
19	Back in VSCode, ensure you are **in branch "master".** If you are, click **Sync Changes**.	

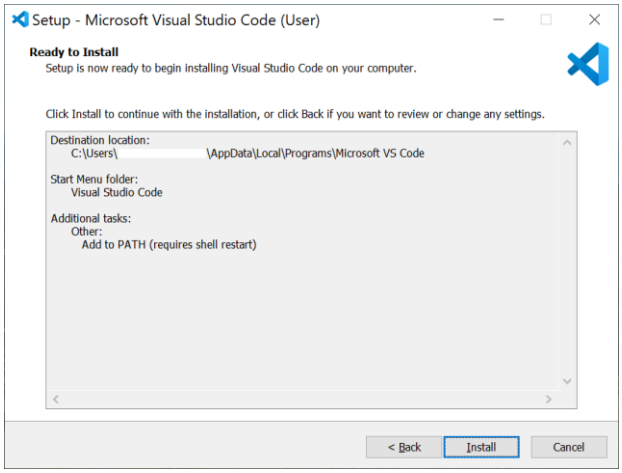
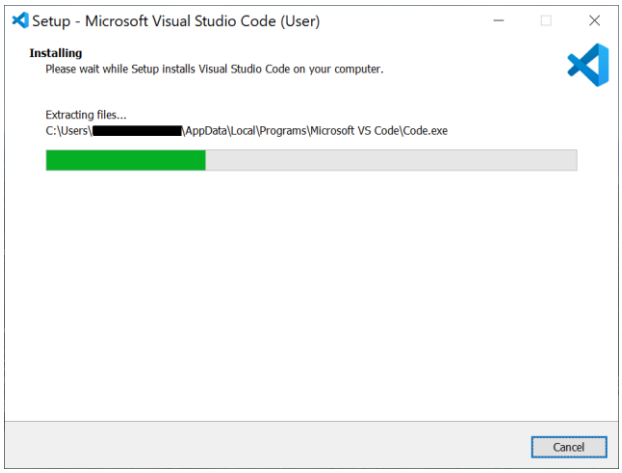
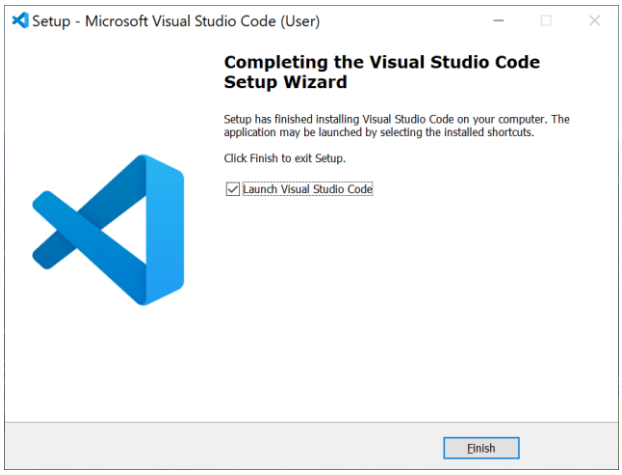
Step	Action	Notes
20	You must have the source icon highlighted at left: Click Select **Branch, Delete Branch** 	
21	Click the branch that is merged already.	
22	Click **Synchronize** icon in Status Bar	
23	You must have the source icon highlighted at left: Click Select **Pull, Push** then **Fetch (Prune)** 	
24	Click **Synchronize** icon in Status Bar	
25	Click in Status bar at bottom left.	

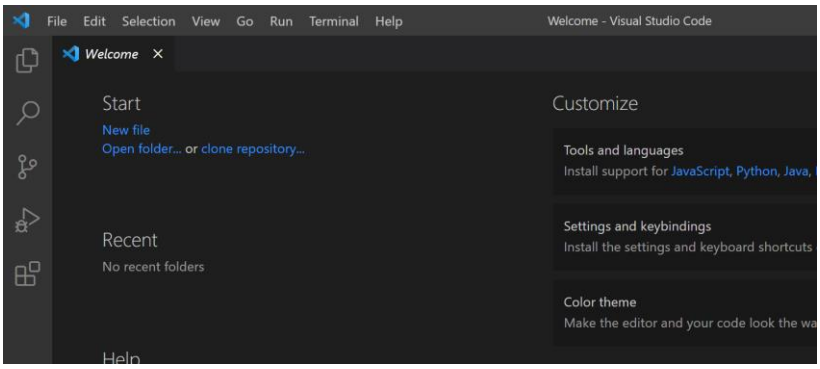
Step	Action	Notes
26	This shows the current branches. **Master** is the only one, which is the normal situation.	
27	Go to this URL:	
28	In a browser, go to URL at right.	https://genevaers.github.io/User-Documentation/
29	After 5 minutes, the changes should appear.	

13 Appendix A: Install Visual Studio Code

Step	Action	Notes
1	Go to the URL at right.	**code.visualstudio.com**
2	Click **Download for Windows**	
3	Click **Save File**	
4	Ensure ⊙ **I accept the agreement** Click **Next**	

Step	Action	Notes
5	Click **Next** 	
6	Click **Next** 	
7	Click **Next**	

Step	Action	Notes
8	Click **Install**	
9	This appears for a while.	
10	Click **Finish**	

Step	Action	Notes
11	You have Visual Studio install. 	

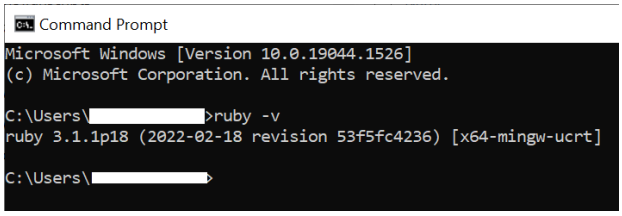
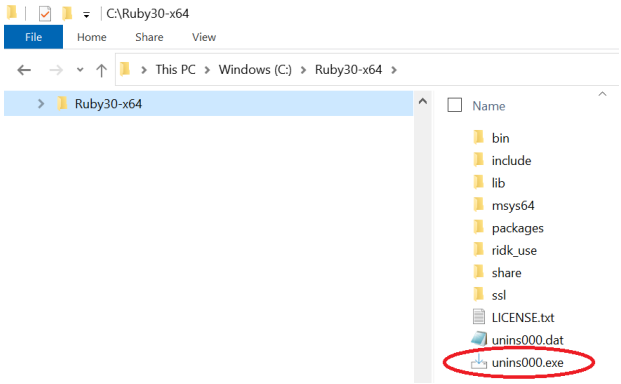
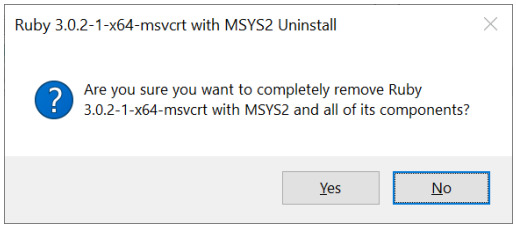
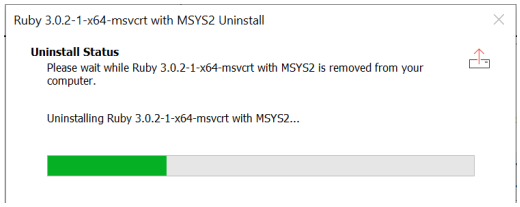
When the install of VSCode is complete, see **Section *4 Install Extension in VSCode***

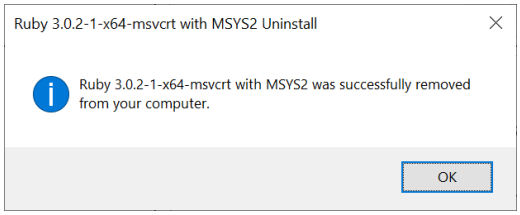
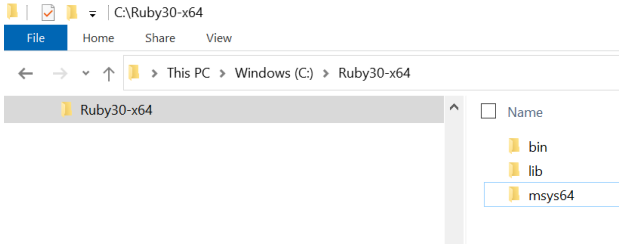
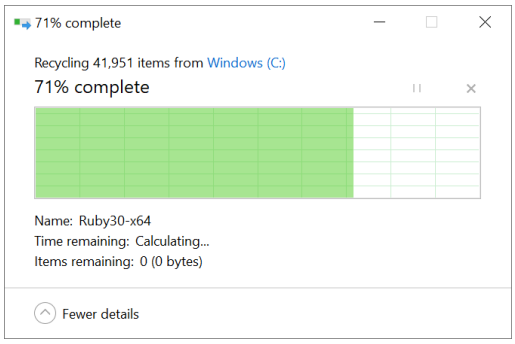
14 Appendix B: De-install Ruby & Jekyll

This is “just in case” – sometimes this comes in handy.

You can de-install and re-install Ruby & Jekyll without damaging your GUD clone or the website.

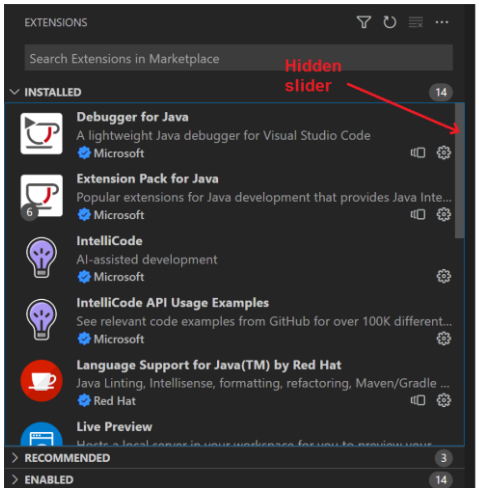
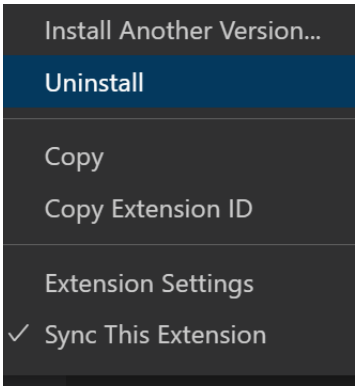
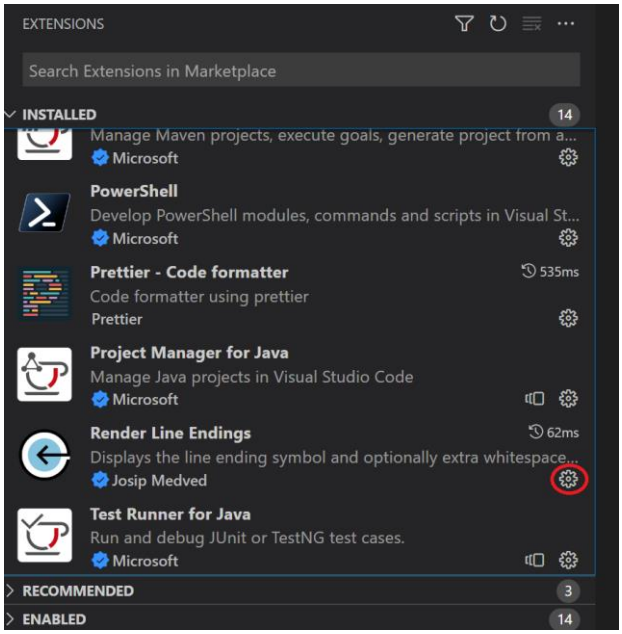
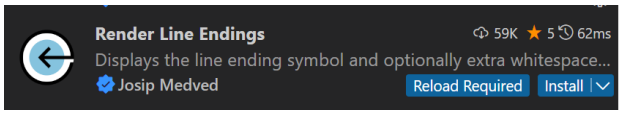
Jekyll is a gem under the control of Ruby, so removing Ruby will also remove Jekyll.

Step	Action	Notes
1	Check Ruby is installed by starting a command prompt.	Start, Windows System, Command Prompt
2	Type **ruby -v**	
3	<u>You should see similar to the screen at right.</u> If so, continue.	
4	Navigate to **C:\Ruby30-x64** or similar folder. Double click **unins000.exe**	
5	Click **Yes**.	
6	This appears briefly	

Step	Action	Notes
7	Ruby and Jekyll are mostly removed.	
8	**Delete** folder **C:\Ruby30-x64**| (and all contents).	
9	Around 44MB of files to delete.	

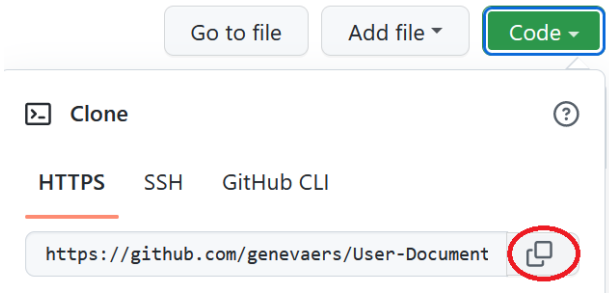
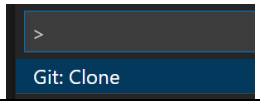
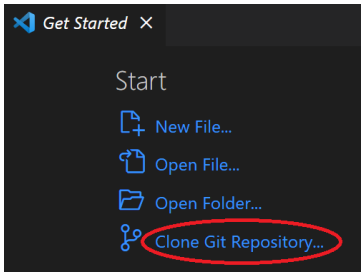
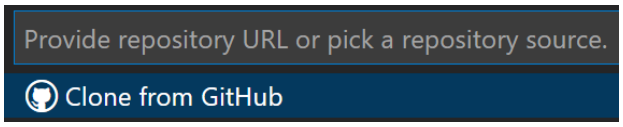
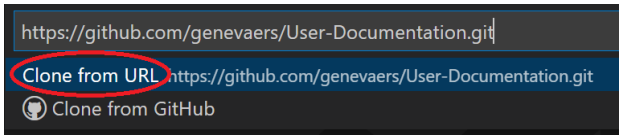
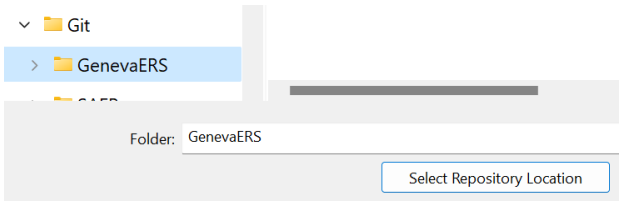
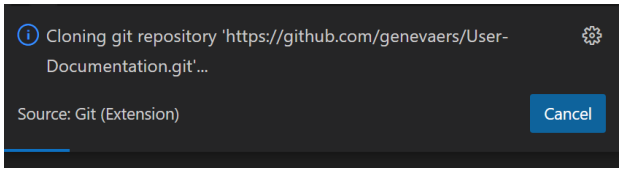
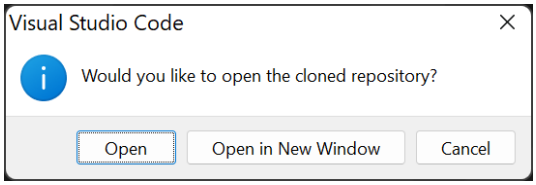
15 Appendix C: Uninstall Render Line Endings

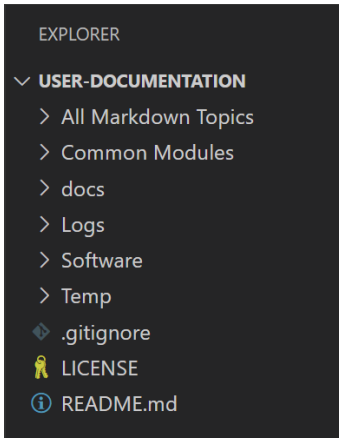
If you don't want this extension in Visual Studio Code, here's what to do.

Step	Action	Notes
1	Ensure you are in **Visual Studio Code**	
2	Click Extensions icon at left:	
3	There is a hidden slider is just to the right of for the first extension. **Scroll to <u>bottom</u>** of "**INSTALLED**" list	
4	Click for Render Line Endings. Select **Uninstall** 	
5	Redner Line Endings is now uninstalled.	

16 Appendix D: Create GUD clone

16.1 Create a GUD clone

Step	Action	Notes
1	Go to URL at right.	https://github.com/genevaers/User-Documentation
2	Click **Code** Click indicated icon	
3	Click **Clone Git Repository** If you can't see this link, do this: • Press **Ctrl-Shift-P** • Click **Git: Clone** 	
4	Paste copied URL into "**Provide**" field	
5	Click **Clone from URL**	
6	Navigate to folder for the clone. Click **Select Repository Location**	
7	A progress window appears.	
8	Click **Open**	

Step	Action	Notes
9	GUD is now successfully cloned. Go to the next section.	

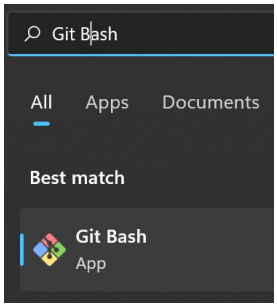
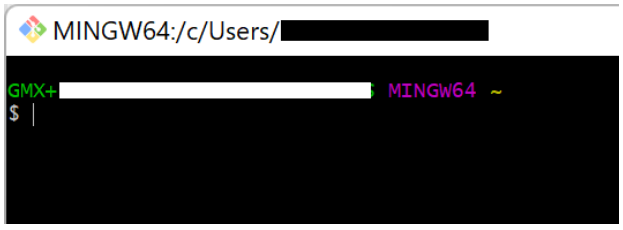
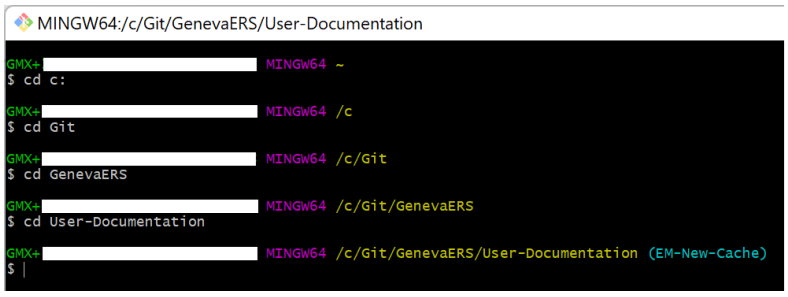
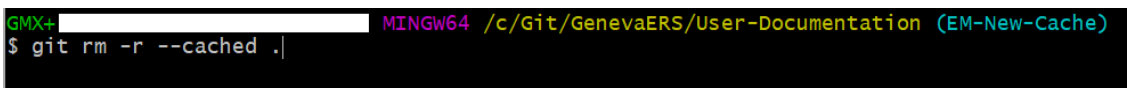
17 Appendix E: Update .gitignore

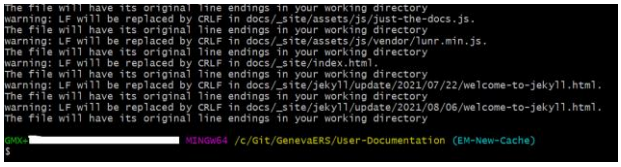
17.1 Update .gitignore

Step	Action	Notes
1	Create a new branch in VSCode	**Section *8.1 Create a new branch in VSCode (if required)***
2	Update file at right. Note the leading full stop.	**User-Documentation\.gitignore**
3	To ignore a **specific file in a folder**, use coding at right.	**docs/search-data.json**
4	To ignore **all files of a certain extension in a folder**, use coding at right.	**docs/*.html**
5	To ignore **all files in a folder,** use coding at right.	**Logs/*.***
6	To ignore **all files in a folder and all subfolders**, use coding at right. (The trailing forward slash indicates a folder.)	**docs/_site/**
7	**Save the changes** in VSCode (Ctrl-S)	
8	**Commit and Merge** the changes to GitHub using the section at right.	**Section *12 Commit & merge changes to GUD***
9	 **The updated ".gitignore" file does nothing if the files in question are already in the "Git cache". The next section refreshes that cache so that the ".gitignore" file starts to work.**	

17.2 Refresh the cache

The Git cache needs refreshing to be in sync with the latest “gitignore” file. It is important that the “.gitignore” file is already updated at this point.

Step	Action	Notes
1	Create a new branch in VSCode	**Section *8.1 Create a new branch in VSCode (if required)***
2	Start **Git Bash** as follows: Click **Start**, in search type **Git Bash**, Select **Git Bash** app.	
3	The Git Bash window opens.	
4	Navigate to the GUD clone on your computer. This is typically: **cd c:** **cd Git** **cd GenevaERS** **cd User-Documentation**	
5	Type **git rm -r --cached .**	
6	A huge amount of output results.	

Step	Action	Notes
7	Type **git add .** GMX+ [redacted] MINGW64 /c/Git/GenevaERS/User-Documentation (EM-New-Cache) $ git add .	
8	A huge amount of output results.	
9	Type **git commit -m “gitignorefixed” --signoff** GMX+ [redacted] MINGW64 /c/Git/GenevaERS/User-documentation (EM-Test-SignOff) $ git commit -m “gitignorefixed” --signoff	
10	There is some output showing files removed from the cache. GMX+ [redacted] MINGW64 /c/Git/GenevaERS/User-documentation (EM-Test-SignOff) $ git commit -m “gitignorefixed” --signoff [EM-Test-SignOff 66710c5] “gitignorefixed” 2 files changed, 58 deletions(-) delete mode 100644 docs/_posts/2021-07-22-welcome-to-jekyll.markdown delete mode 100644 docs/_posts/2021-08-06-welcome-to-jekyll.markdown GMX+ [redacted] MINGW64 /c/Git/GenevaERS/User-documentation (EM-Test-SignOff) $	
11	The gitignore configuration for this clone should now be working. Make changes as usual and commit to GitHub.	

18 Appendix F: Practice an include

Use this to learn how to do includes.

Step	Action	Notes
1	Ensure you have performed the sections at right ONCE ONLY.	**Section *4 Install Extension in VSCode*** **Section *5 Install Ruby and Jekyll*** **Section *6 Setup for Prepare-Website***
2	Ensure you have a GUD clone.	**Section *16 Appendix D: Create GUD clone***
3	Create new branch in VSCode – say **XX-Update-01** where XX is your initials.	**Section *8.1 Create a new branch in VSCode (if required)***
4	Create file **Xxxxxxx.md** in folder **Common Modules\Test** where **Xxxxxxx** is your first name.	The file contains only this text: **The text was included by Xxxxxxx.**
5	Save that file.	
6	Edit the file at right. You can edit in VSCode or Notepad or something else.	All **Markdown Topics\About this website.md**
7	Insert the lines at right at the bottom of the file. The apparently blank line has two spaces. On the INCLUDE: line, there can be spaces before the [[and after the]].	**[[INCLUDE: Test\Xxxxxxx]]**
8	Save the file.	
9	Perform **Section *10 Run Prepare-Website***	This is always necessary when you have finished changes to the website.
10	Perform **Section *11 View website on localhost***	If you have LocalHost running and you make a file change and run GUDPW again, you need to shut down the command prompt and start a new one for GUDLH. Poor old Localhost cannot cope with the changes GUDPW does
11	On Localhost, start the home page and click "**Reference**" and then "**About this website**". You should see your include.	If you don't see your include, then check Step 4 in **Section *11.2***

Step	Action	Notes
12	Perform **Section *12 Commit & merge changes to GUD*** You will see there are three files to commit and merge: **Xxxxxxx.md** Common Modules\Test **About this website.md** All Markdown Topics **About this website.md** docs This is correct – You created the file in **Common Modules** – You updated the file in **All Markdown Topics**. – Prepare-Website updated the file in **docs** (to process the include).	
13	See the updated website at https://genevaers.github.io/User-Documentation/d This takes about 5 minutes after the merge to complete the update.	

END OF DOCUMENT