Development Environment

Table of Contents

Source Control

The project name, the repository name, and the folder name that contain the source code for the Tax Planning Tools is “TaxTools”. Therefore, you will frequently see the name “TaxTools” used to refer to the Tax Planning Tools in this document.

Source code version control for the Tax Planning Tools is implemented using Git and GitHub. Git is a distributed version control system originally developed for the Linux kernel, but can be use for any project where versioning changes to files is necessary. It is particularly useful in situations where there are multiple people making changes in separate locations. For more information on Git, go to their web page at git-scm.com.

While Git is designed to be a distributed source control system, it is very common to have a central repository that all the developers draw from. GitHub fills this need. It provides a common reference copy of the source code that is always available on the web and always backed up. GitHub provides both free and fee-based hosting for Git repositories. For more information on GitHub, go to their web page at github.com.

Repositories

A repository is a directory that contains the source code for the project and has been initialized for use with Git. When the directory is initialized, Git creates a subdirectory named “`.git`” at the top of the source directory. This is where Git stores all the information it needs to track the changes to the source code.

Since Git is a distributed source management system, a project can have any number of repositories. Typically, each developer has a copy on their system. In addition, there may be one repository that is considered the “main” repository where changes are collected and other repositories can be synchronized to keep up with the changes that have been made by others.

This is the case for the Tax Planning Tools. The main repository is hosted on GitHub at `github.com/bb3413/TaxTools.git`, and each developer has a local copy of the repository. To work on the Tax Planning Tools, you start by cloning the repository on GitHub, which creates a local copy that you can use to do your work. By keeping this local copy synchronized with the main repository stored in GitHub, you’re able to get the latest changes from the main repository and push your updates back to the main repository so others can use them. To “clone” of the official repository, use the following command.

```
$ git clone https://github.com/bb3413/TaxTools.git
```

Authentication

When working with remote systems, it may be necessary to provide some form of authentication. The Tax Planning Tools repository is a public repository, so you can read from the repository (for example, clone) without providing any authentication, but to write to the repository, you will need to be authenticated in one of two ways.

1. You can access the Tax Planning Tools repository using the following HTTPS URL.

   ```
   https://github.com/bb3413/TaxTools.git
   ```

   If you use this URL to write to the repository (for example, `git push`), you will have to provide the account name and a security token each time you write to the repository. This may work well when working from Windows, since the Windows Credential Manager will cache this information and supply it on subsequent commands.

2. Alternatively, you can use the secure shell (SSH) to access the repository. In this case, you use the following Git address.

   ```
   git@github.com:bb3413/TaxTools.git
   ```

This may be more convenient when working from the command line (for example, the Bash shell). However, to use this address, you need to setup the SSH connection as follows.

a. Execute the `ssh-keygen` command and press enter to accept the default for all the questions. This will create a public (`id_ed25519.pub`) and private key (`id_ed25519`) in the directory `$HOME/.ssh`.

```
$ ssh-keygen -t ed25519 -C "comment"
$ cat $HOME/.ssh/id_ ed25519.pub # Show the public key
```

b. Then, you can copy and paste the public key to GitHub. To enter the public key on GitHub, go to Settings->SSH and GPG keys->New SSH key.

The first time you use this address, you will get a warning asking if you trust the remote system, but you will not need to provide any authentication information.

When reading or writing to the repository, you can use either form of address. It only makes a difference in how you are authenticated.

In addition to providing authentication, you need to be added as a "collaborator" on the project by the owner of the repository if you want to make change to the project.

Branches

A branch refers to a separate versions of the same code. The original version continues in one branch and a separate branch is created to keep copies of changes that deviate from the original branch. Creating a branch allows developers to work on changes without affecting each other. When a change is complete, the branches can be merged back together and any conflicts can be resolved.

The Tax Planning Tools repository contains two branches. The main branch, named "main", contains the currently released version of the Tax Planning Tools. Changes to the main branch may be made for small fixes, but generally, the main branch is only changed by periodically merging with the development branch.

The other branch is the development branch. It is named "develop", and it contains the new features and fixes. Once a change is believed to work, it can be added to the development branch to give it more exposure and verify that it is stable. Merging the development branch with the main branch is based on the need for the changes and the perceived stability of the development branch.

Small changes may be made directly in the development branch, but new features should be developed in their own branch off of the development branch. Developing new features in their own branch allows the changes to be checked in (committed) even if they are not complete. The need to check in incomplete changes prevents the need to have "stashes" for each change that is in progress but not complete (stashes are needed to save files that are not checked-in when you need to change to another branch to work on another problem).

Create the Initial Repository

Initially, the Tax Planning Tools consisted of a local directory that contained all of the source code for the tools. This section describes the steps that were used to create the main repository on GitHub from the original source code. This is for information only since the Tax Planning Tools have already been set up to use Git and GitHub to manage the source code.

1. Create a personal access token on GitHub. GitHub requires using a personal access token instead of a password for authentication when using the command line (e.g., `git push` command).
    - In a web browser, go to github.com.
    - Log into account bb3413. Use password for account bb3413.

- Go to Home->Settings->Account->Developer settings->Personal access tokens.
- Select Tokens (classic)
- Click on Generate new token
- Select Generate new token (classic)
  - Enter the account password.
  - Note: Full CLI access to repositories. Use web for everything else.
  - Expiration: 90 days
  - Click on Generate token

2. Create the new, empty repository on GitHub.

   - In a web browser, go to github.com.
   - Log into account bb3413. Use password for account bb3413.
   - Click on the "+" icon in the top-right corner and select "New repository."
     - Repository Name: TaxTools
     - Description: Tools for estimating income tax.
     - Choose Visibility: Public
     - Add README: Off
     - Add .gitignore: No .gitignore
     - Add license: No license
   - Click "Create repository."

3. Convert the source code directory to a Git repository. The `git init` command will convert the current directory into a Git repository where you can use Git commands to track changes to files in the directory. A Git repository has a subdirectory at the top of the source directory named `.git`. This is where Git stores all the information it needs to track changes to the files.

   ```
   $ cd TaxToolsDirectory
   $ git init
   ```

   There may be some files in the directory that are not part of the source code for the project (for example, object files generated by a compiler or files containing private information) and, therefore, you want Git to ignore those files. You can specify these files by making an entry for them in the file `.gitignore`, which you put at the top of the repository. The Git ignore file contains information that is needed by all users of the repository so it should be added to the repository just like a source file. Note: it should only identify temporary files; files that are not ready for release should be left in the source tree so they can be shared when someone clones the repository.

   ```
   $ cat <<-EOF >>.gitignore
          # DreamWeaver status files
          _notes/

          # Editor temporary files
          ~*
          *.swp

          # Modules loaded with NPM
          nodes_modules/

          # NPM status files
          package.json
          package-lock.json

          # ESLint configuration file
          eslint.config.mjs
   EOF
   ```

4. Add the source code files to the repository. The `git add` command adds all the files in the repository to the staging area. Commit the changes in the staging area to the repository. At this point, the source code directory is a Git repository and changes to the files in the directory will be tracked by Git.

```
$ git add .
$ git commit –m "Initial commit."
```

5. The main branch is named "master" by default. Change it to "main".

```
$ git branch -M main
```

6. Mark the current version of the Tax Planning Tools in the main branch with the release number.

```
$ git tag VersionNumber        # e.g., "v2025.08"
```

7. Connect the local repository to the remote repository. The `git remote add` command allows you to specify a name and URL for a remote repository. The name "origin" is the default name used by Git when the name of the remote repository is not specified.

```
$ git remote add origin git@github.com:bb3413/TaxTools.git
$ git remote add originSSH git@github.com:bb3413/TaxTools.git
$ git remote add originHTML https://github.com/bb3413/TaxTools.git
```

8. Populate the remote repository with the files from the local repository. The `-u` option sets "origin" and "main" as the default remote repository and branch. This allows you to later execute the `git push` and `git pull` commands without having to specify these parameters.

```
$ git push –u origin main
$ git push origin --tags
```

9. Create the development branch.

```
$ git branch develop
$ git switch develop
```

10. Update the version information.

```
$ edit Version.js Version.html
$ git add .
$ git commit –m "Create new development version."
```

11. Push the development branch to GitGub. The `-u` option changes the default branch, set above, to now be the "develop" branch. The origin is still set to the Tax Planning Tools repository on GitHub.

```
$ git push –u origin develop
```

12. Go back to GitHub, refresh the page, and make sure the repository looks correct.

Using Git with the Tax Planning Tools

Configuration Settings

Git configuration settings can be set for a user or a repository. User settings apply to the user whenever they are using Git regardless of which repository they are in. Repository settings apply to all users in a specific repository.

The `git config` command allows you to set Git configuration values. If the `--global` option is used, the command sets the value for the user. If the `--global` option is not used, the command sets the value for the current repository.

User configuration settings are stored in the file `$HOME/.gitconfig`, and repository configuration settings are stored in the file `.git/config` at the top of the repository.

User Settings

Before using Git to make changes, you need to set your name and email address in Git. This information is needed to track changes to the files.

```
$ git config --global user.name "First Last"
$ git config --global user.email "Email address"
```

The default name for the initial branch when creating a Git repository is "master". This term is deemed improper so it is recommended that you change the default to "main" for any new Git repositories.

```
$ git config --global init.defaultBranch main
```

When a Git repository is not stored on a local disk (e.g., cloud storage), you need to tell Git that you trust the directory containing the repository.

```
$ git config --global --add safe.directory \
       '/mnt/g/My Drive/Documents/VSCode/TaxToolsDev'
```

Create Local Repository

Since the Tax Planning Tools are available on GitHub, you can make a local copy of the repository (i.e., source code + Git tracking information) by changing directories to where you want to save the repository and execute the following Git command to clone the repository. This will create a subdirectory named "TaxTools" with the most recent version of the source code for the tax tools.

```
$ cd directory
$ git clone https://github.com/bb3413/TaxTools.git
```

Create Account on GitHub

Once you have a local copy of the Tax Planning Tools repository, you can make changes to the files and perform most of the steps described in this section, but you will not be able push your changes to the remote repository so others can see them until you have a GitHub account and have been added as a "collaborator" on the repository on GitHub.

1. Create a GitHub account
2. Ask the owner of the repository on GitHub to be added as a collaborator. The repository owner will need your GitHub account name. The owner can add your account as a collaborator by going to the repository and selecting Settings->Collaborators->Add people.

Synchronize with Remote Repository

Before you make a change to the Tax Planning Tools source code, you want to make sure your local repository contains the most up to date version of the source code file. To synchronize your local repository with the files on the remote repository, use the `git pull` command.

```
$ git switch branch          # Make sure you are in the correct branch
$ git pull
```

Get Status

There are several commands to get information about the repository.

```
$ git status         # Information about the working directory.
$ git show           # List the changes to the current branch.
$ git log            # Show the commit history on current branch.
$ git diff           # See what changed in a file.
```

Example

You can use `git status` to see which files in the working directory have changes. This command can be executed from anywhere in the working directory.

```
$ cd $TAXTOOLS
$ git status
```

From the files that are listed, if they are already being tracked by Git, then you can use the `git diff` command to see which lines in the file have changed.

```
$ cd directory
$ git diff filename
```

Once you are satisfied with the changes, you can add them to the staging area and commit the changes to the repository. When specifying "." as the files to add, that only adds the file from the current directory, so change directories back to the top of the working directory to add all the changes that have not been staged.

```
$ cd $TAXTOOLS
$ git add .
$ git commit -m "Description of change."
```

Or, you can do it one step.

```
$ git commit -a -m "Description of change."
```

Sync with GitHub.

```
$ git push
```

Select a Branch

If you have added, deleted, or modified any files but have not committed those changes, they will remain in your source directory even if you change to a different branch??? For this reason, it is important to make sure you there are no changes that have not been committed before you change branches.

```
$ git status        # Check for changes that have not been committed.
```

After you switch to a new branch, it is also good to make sure the branch is up to date with any changes that have been posted to the remote repository.

```
$ git switch branch
$ git pull          # Make sure it is up to date.
```

Below are some other useful commands for use with branches. The functionality of the `get switch` command is duplicated by the `git checkout` command. The `git checkout` command can switch branches, restore files, and check out commits. The `git switch` command is a newer command intended specifically for switching branches.

```
$ git branch               # List all branches
$ git branch branch        # Create new branch; does not switch to it
$ git branch -d branch     # Delete branch
$ git branch -r            # List all branches in remote repository
$ git checkout branch      # Switch to branch
$ git checkout -b branch   # Create new branch and switch to it
$ git switch branch        # Switch to branch
$ git switch -c branch     # Create new branch and switch to it
```

Create a New Branch

For most changes, you will start at the development branch and create a new branch where you will make your changes. Before you create a new branch, make sure you are in the correct branch.

You can create a new branch and switch to it in two steps.

```
$ git branch branch          # Create new branch
$ git switch branch
```

Or, one step.

```
$ git checkout -b branch     # Create new branch and switch to it
```

Add a New File

To add a new file to the project, you need to create the file, add it to the staging area, and commit the change (i.e., whatever is in the staging area).

1. Before you add a new file, make sure you are in the correct branch and the correct location, then create the file.
2. Add the new file to the staging area.

   ```
   $ git add filename
   ```

3. Commit the changes in the staging area. This will commit the changes in the staging area (i.e., the new file) to the repository. At this point, the new file has been added to the current branch. At some point, you may want to merge the current branch into the development branch.

   ```
   $ git commit -m "Description of change."
   ```

Delete a file

The `git` rm command deletes a file from the working directory and adds the change to the staging area, so after executing the `git rm` command, you can commit the change.

```
$ git rm filename
$ git commit -m "description"
```

Change a File

To change file to the project, you need to edit the file, add it to the staging, and commit the change (i.e., whatever is in the staging area).

1. Before you change a file, make sure you are in the correct branch and that the branch is up to date. Then, change the file.
2. Add the file you changed to the staging area.

   ```
   $ git add filename
   ```

3. Commit the changes in the staging area. This will commit the changes in the staging area (i.e., the modified file) to the repository. At this point, the modified file has been changed in the current branch. At some point, you may want to merge the current branch into the development branch.

   ```
   $ git commit -m "Description of change."
   ```

Revert a Change

To discard a change to a file in the working directory before it has been staged use the `git restore` command. This restores the file to the state it was in after the last time it was committed.

```
$ git restore filename       # Restore the file
$ git restore .              # Restore all files in the current directory
                             # and subdirectories.
```

Stage a File

Any change to the working directory, such as adding, deleting, or modifying a file, needs to be added to the staging area before the change can be committed. Use the `git add` command to stage a file. If more than one file is changed, you can specify all the names to the `git add` command or you can specify dot (".") instead of the file name to add all files that have been added, deleted, or modified anywhere in the current directory to the staging area.

```
$ git add filename
```

If you stage a file and then make another change to the file, you need to re-stage the file; otherwise, only the first change will be committed, but the second change will remain in the file in the working directory.

Instead of the `git add` command, you could also use the `git stage` command. These commands can be used interchangeably.

Revert a Stage

If you make a change and add it to the staging area, but you have not committed the change, you can remove the file from the staging area with the following command. The file in the working directory is not affected.

```
$ git reset filename          # Unstage a file
$ git reset                   # Unstage everything
```

Commit a Change

The `git commit` command will make all the changes in the staging area a permanent change in the repository.

```
$ git commit –m "Description of change."
```

Revert a Commit

```
$ git revert HEAD          # Revert the last commit
```

Push Changes to the Remote Repository

The repository on GitHub is considered the master repository. While Git is capable of distributed source management, it is common for a central repository to be used as a collection point for all changes and for making official releases. For the Tax Planning Tools, the repository on GitHub is that repository. The "local repository" refers to the copy of the repository that you have on your local system, and the "remote repository" refers to the repository on GitHub.

If you have permission to change the remote repository you can use the `git push` command to push committed changes from the local repository to the remote repository.

```
$ git push
```

Make a Change to Development Branch

1. Create a local repository if necessary.
2. Bring the local repository up to date from the master repository on GitHub.

   ```
   $ cd TaxToolsDirectory
   $ git pull
   ```

3. Create a branch off the development branch for the change.

   ```
   $ git checkout develop            # Switch to the development branch
   $ git checkout –b branch          # Create the new branch
   ```

4. Feature development loop.

Edit the files. Edit the version number and version log

```
$ code .      # Run VS Code editor in the current directory.
```

Commit the changes.

```
$ git add .   # Add the changes to the staging area.
$ git diff –staged
$ git commit –m "Description of change"
```

5. Merge the changes into the development branch

```
$ git checkout develop     # Change back to the development branch
$ git merge branch         # Merge the changes
```

6. Update Version Information. Edit the version files directly in the development branch. Increment version number (third number) and update version log.

```
$ edit version.js version.html
$ commit –a "Update version"
```

7. Update the master repository on GitHub.

```
$ git push
```

8. Update the Web page with the new prototype version. Open DreamWeaver and push the folder with the Tax Planning Tools to the web.

Make Change to Main Branch

1. Make sure there are no changes files in local repository. The `git status` command shows which files have been added, deleted, or modified in the local repository.

```
$ git status
```

2. Change to the release branch (main).

```
$ git checkout main
$ git pull
```

3. Edit the files directly in the main branch.

```
$ edit files
```

4. Update version information. Increment the version number (second number) and update version log.

```
$ edit version.js version.html
```

5. Commit the changes in the modified files to the repository.

```
$ git commit –a "Description of change."
```

9. Update the master repository on GitHub.

```
$ git push
```

Merge Development Branch into Main Branch

Merging the development branch into the main branch creates a new release of the Tax Planning Tools. The head of the main branch is always the current release of the Tax Planning Tools.

1. Make sure there are no edited files in the working directory. The `git status` command shows which files have been added, deleted, or modified in the working directory.

```
$ git status
```

2. Change to the development branch (develop).

```
$ git checkout develop
$ git pull                # Make sure the branch is up to date.
```

3. Update Version Information. Set the version number for the next release and make an entry in the log file.

```
$ cd Version
$ edit version.js version.html
$ git commit –a –m "Update version information."     # Add and commit
$ cd ..
$ git push
```

4. Change to the release branch (main).

```
$ git checkout main
$ git pull                # Make sure the branch is up to date.
```

5. Merge the development branch.

```
$ git merge develop
$ git commit –a –m "Update version information."
```

6. Create a Release Tag.

```
$ git tag VersionNumber       # e.g., "v2025.08"
$ git push                    # Push the commits
$ git push --tags             # Push the tags
```

7. Copy the release files to the release directory for pushing to the web page. The current branch should still be in release branch (main).

```
$ cd $TAXTOOLS/..
$ rm –rf TaxToolsNew
$ TaxToolsDev/tools/UpdateRelease       # Copy TaxToolsDev to TaxToolsNew
$ dircmp TaxTools TaxToolsNew           # Compare with previous release
```

8. If everything looks good, make a backup copy and replace the TaxTools directory with TaxToolsNew.

```
$ mv TaxToolsNew TaxTools-VersionNumber
$ tar –cvzf TaxTools-VersionNumber.tgz TaxTools-VersionNumber
$ cp TaxTools-VersionNumber.tgz $ARCHIVE/TaxTools
$ rm –rf TaxTools                       # Remove old release files
$ mv TaxTools-VersionNumber TaxTools    # Replace with new files
```

9. Update the web with the new version of the Tax Planning Tools.

```
$ Open DreamWeaver
      # Copy the new Tax Planning Tools to the web
```

10. Change the local repository back to the development branch.

```
$ cd $TAXTOOLS
$ git checkout develop
$ git pull    # Make sure it is up to date.
```

11. Update Version information for the current development release.

```
$ edit version.js version.html
$ git commit –a "Start new development version."
```

```
$ git push
```

Get a Past Version of TaxTools

Create a Tag

You create a tag with the `git tag` command. When you push commits to the remote repository, it does not push the tags; they must be pushed in a separate step.

```
$ git tag VersionNumber       # e.g., "v2025.08"
$ git push                    # Push the commits
$ git push --tags             # Push the tags
```

You delete a tag with the `git tag` command as well.

```
$ git tag --delete tagName  # e.g., "v2025.08"
$ git push origin --delete tagName
```

Git Help

The following two URLs show the online documentation for Git. The first URL is the online manual page for Git and the second URL is the complete Git documentation.

https://git.github.io/htmldocs/git.html
https://git-scm.com/docs

The Git command also has all of the documentation built into the command so you can access it from the command line without opening a browser.

To show the command line syntax and a list of common Git commands:

```
$ git help
```

To list all of the Git commands:

```
$ git help -a
```

To list guides or tutorials on various topics:

```
$ git help -g
```

For help with a specific command:

```
$ git help command
```

Git Terminology

Commit	Make the changes in the staging area permanent.
Head	Is the most recent commit in the current branch. Since each branch has a head, HEAD in all capital letters refers to the head of the current branch.
Local	Or local repository, refers to the developer's copy of the repository on their local system.
Origin	Is the default name Git uses to refer to a remote repository.
Remote	Refers to the remote repository, which is the same repository on a server somewhere. In our case, it is the Tax Planning Tools repository on GitHub.
Repository	Is any directory that has been initialized by Git using the `git init` command. Once initialized, the directory will have a subdirectory named `.git` where Git stores all the information it needs to track changes to the files in the directory.

Staging Area — Also called the *index* or the *cache*. It is a temporary area where you "stage" changes to be included in the next commit.

Working Directory — The directory containing the source files for the branch you are currently working on. Thus, it is the general term for the source code directory where you do your work. It is also the current branch in the local Git repository.

Editing Tools

VS Code

Visual Studio Code, usually called VS Code, is a popular, free editor for developers. There are many extensions that give it additional capabilities and interoperability with other tools, but it comes with build in support for HTML, CSS, and JavaScript. For more information on VS Code, go to https://code.visualstudio.com.

Installation

1. Go to https://code.visualstudio.com/download
2. Click on the correct install option (for Windows, all the options result in the same installer).
3. Click on the enormous VS Code button to download the installer.
4. Execute the installer
    - Check **Add to PATH**. This is needed to run VS Code from the command line by entering `code`.
    - Check **Add Open with Code action to Windows Explorer file context menu**. This allows you to right click on a file to open the file with VS Code.
    - Check **Add Open with Code action to Windows Explorer directory context menu**. This allows you to right click on a directory to open it with VS Code.
    - Click **Register Code as an editor for supported file types**. This makes VS Code the default editor for the supported file types.

Configuration Settings

Synchronize your configuration settings across all the systems you use.

1. Click on the gear icon and select **Backup and Sync Settings**.
2. You will be prompted to sign in to a GitHub or Microsoft account. Select **Microsoft**.

User Settings

User settings are only for the user, but apply in all projects that the user works on. User settings are stored in the file VS Code settings are stored in the file `C:/Users/`*UserName*`/AppData/Roaming/Code/User/settings.json`.

Editor	Bracket Pair Colorization	?
Terminal	Default Profile Windows	Ubuntu (WSL)
Workbench	Color Theme	Light Modern

Workspace Settings

Workspace settings apply to all users working in the workspace. Workspace settings are stored in the file `.vscode/settings.json`. at the top of the project directory. This is a workspace file are should be checked into the Git repository.

Editor	Format On Save	False	
Editor	Tabsize	4	# Spaces per indentation level
Editor	Insert Spaces	false	# Use tabs for indentation
Editor	Detect Indentation	true	# Match existing indentation
Files	Insert Final Newline	True	
Files	Trim Trailing Whitespace	True	

Extensions

I use the following extensions. VS Code does not support printing, so you need to add an extension to provide the functionality. The Print extension adds print icons to the top banner of the editor panel. The ESLint and Prettier extensions are discussed below.

Print
Prettier - Code Formatter
ESLint

The following extensions may be useful; I haven't decided.

GitLens
Code Spell Checker
Jest
GitHub Copilot
GitHub Copilot Chat
Open in Browser (by TechER)

Command Summary

Ctrl + Shift + P	Command palette
Ctrl + ,	Open settings
Ctrl + S	Save file
Ctrl + K S	Save all
Shift + Alt + F	Format file
Ctrl + G	Go to line
Ctrl + B	Show/hide side bar
Ctrl + J	Show/hide the panel
Ctrl + `	Show/hide terminal
Ctrl + Alt + I	Show/hide Copilot chat window
F11	Toggle full screen

Node and NPM

Note: the tools described in this section do not work when the repository is on cloud storage (e.g., Google Drive).

For now, the following steps should be done in your local repository and not checked into Git.

The Node Package Manager (NPM) is a program that lets you download JavaScript libraries from the NPM registry and use them in JavaScript projects. NPM is bundled with Node, so NPM will be installed when you install Node. Node, or Node.js, is a JavaScript runtime environment and web server, which allows you to test server-side JavaScript code on your local system.

Installation

1. Go to https://nodejs.org
2. Execute the installer
3. In a terminal window, initialize the package manager (NPM) for use with your project. This step only needs to be done once, so this step has already been done. This step creates the file `package.json`, which NPM uses to track which packages have been installed and their dependencies.

   ```
   $ cd TaxToolsDirectory
   $ npm init -y
   ```

4. Other NPM commands:

   ```
   $ npm install packageName
   $ npm list
   $ npm outdated      # List packages that need to be updated
   $ npm uninstall packageName
   $ npm update [packageName]
   ```

Prettier

Prettier - Code Formatter is the code formatter used by the Tax Planning Tools.

1. Install the prettier package. The `save-dev` option indicates that the package is only used during development and will not be distributed with the project.

   ```
   $ cd TaxToolsDirectory
   $ npm install prettier --save-dev
   ```

2. Run Prettier on a file.

   ```
   $ npx prettier filename
   ```

3. In VS Code, install the extension Prettier – Code Formatter and change the following configuration settings. Restart VS Code. Use the keyboard shortcut Shift + Alt + F to format the current document and use Ctrl + Z to undo the change.

Editor	Format On Save	false
Editor	Default Formatter	Prettier – Code Formatter

ESLint

NOTE: This does not work when the source code is hosted on Google Drive. Clone a new repository on the local system and use ESLint there.

ESLint is the code checker used for the Tax Planning Tools.

1. Install the ESLint package. The `save-dev` option indicates that the package is only used during development and will not be distributed with the project.

   ```
   $ cd TaxToolsDirectory
   $ npm install eslint --save-dev
   ```

2. Create a default configuration for ESLint.

   ```
   $ npx eslint --init
   ```

3. Run ESLint on a file.

   ```
   $ npx eslint filename
   ```

4. In VS Code, install the extension ESLint. Restart VS Code.

Git

Git and GitHub are the source code version control tools used for the Tax Planning Tools. This section describes the installation of Git, but there is a complete discussion on how these tools are used with the Tax Planning Tools earlier in this document.

Installation

1. Go to https://git-scm.com/install/windows.
2. Select the Windows tab.
3. Click on **Click here to download**…
4. Execute the installer
    - Git will be installed at: C:/Users/<UserName>/AppData/Local/Programs/Git. You may need to know this if one of the tools is not able to find the Git commands.

Notes

Coding Conventions

- Indentation is 1 tab and tabs are 4 spaces.
- Variable names are all lower case with words separated by underscores.
- Function, property, and method names use camel case.
- Function and method names begin with a lower case letter.
- Class and constructor names begin with a capital letter.
- Use ===, not == whenever possible.
- Do not use var.