

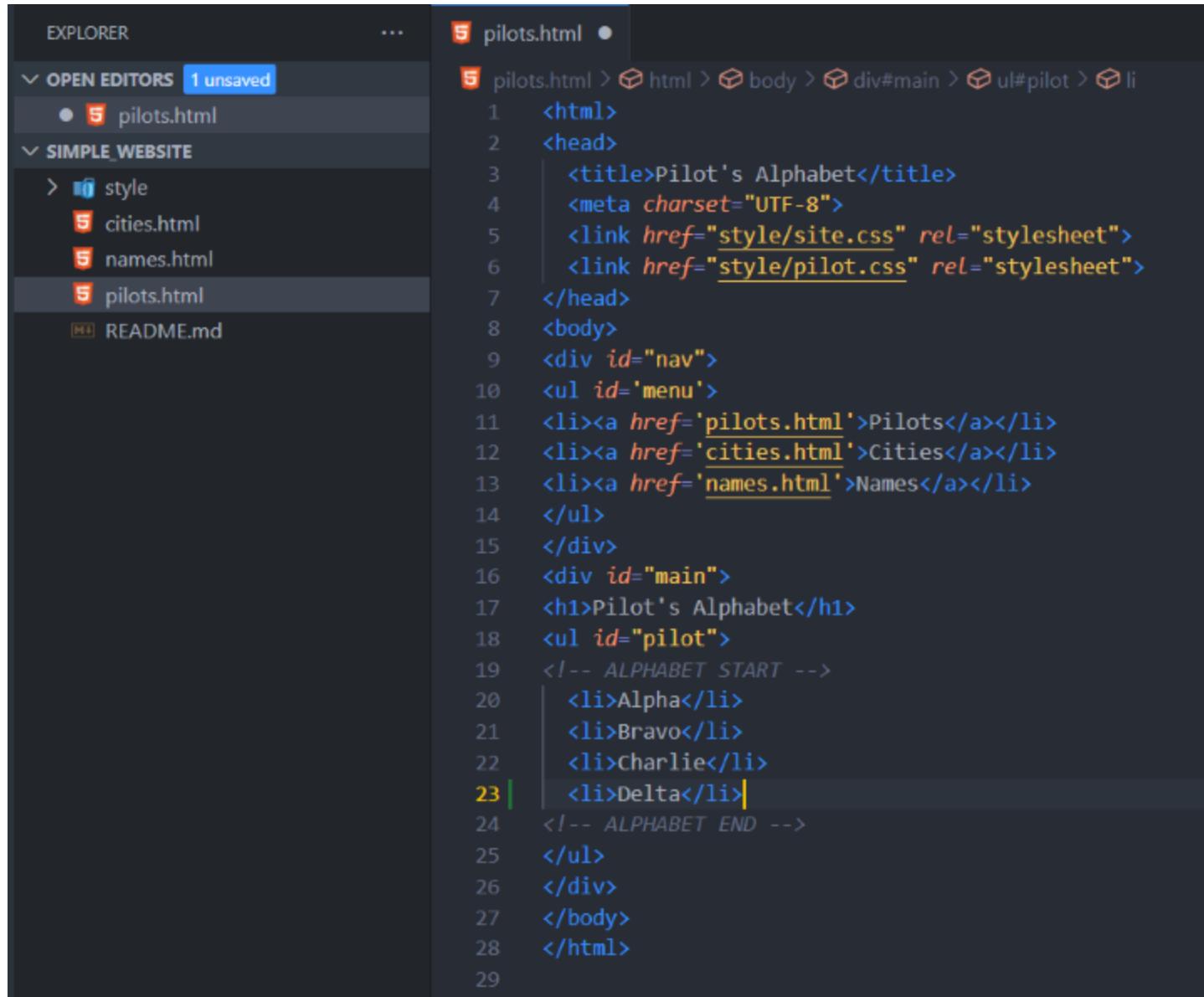
## Project Version Control with Git

During the life of a project you will implement new features and continuously make changes to the working directory by adding, deleting, renaming and editing source code and other files.

### Implementing a feature request

A feature request is a requirement to modify or add functionality to an application or website.

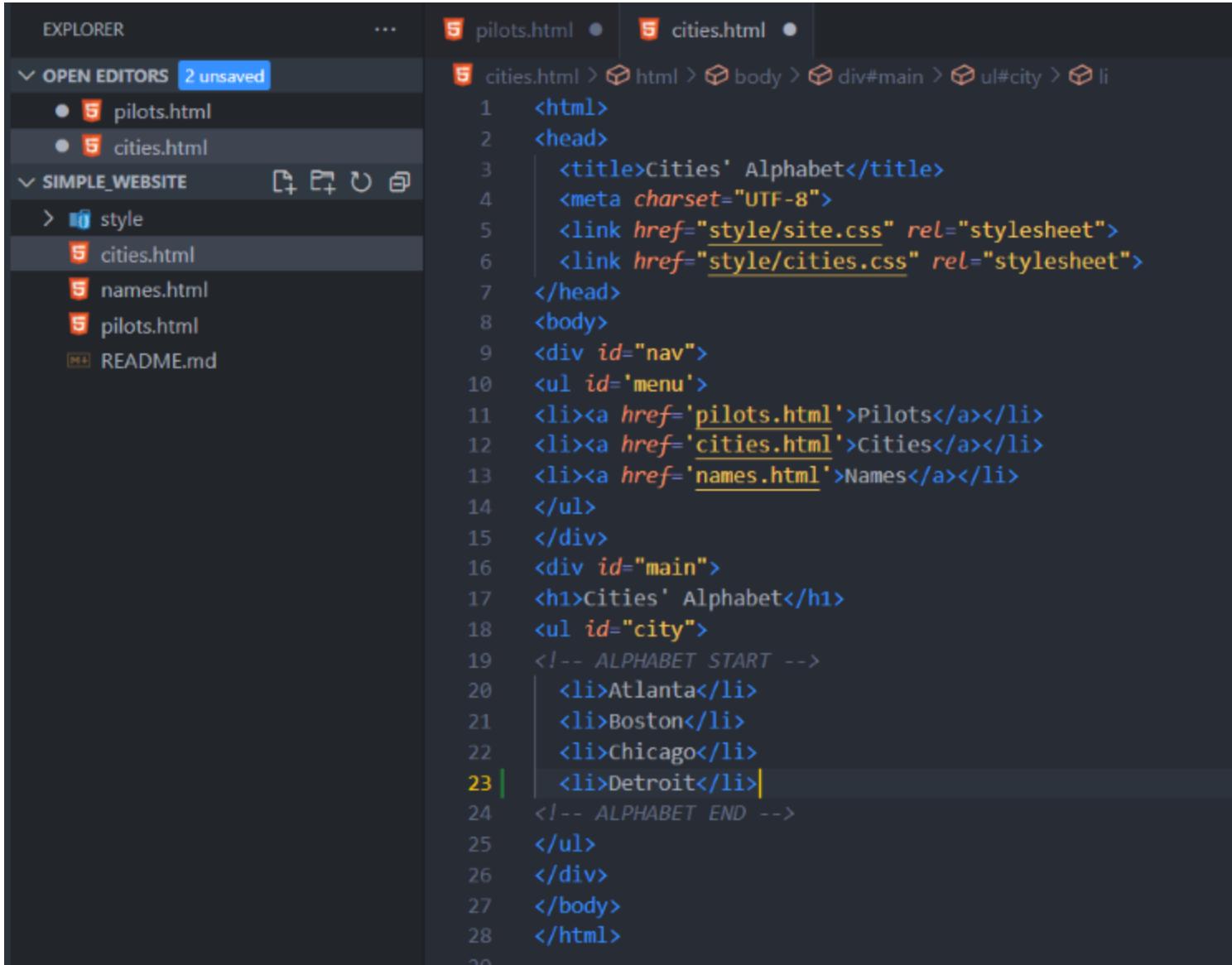
# Open pilots.html to add "Delta"



The screenshot shows the VS Code interface with the 'pilots.html' file open in the editor. The file contains an HTML structure with a navigation menu and an alphabet section. The word 'Delta' is being typed into the alphabet list.

```
<html>
<head>
    <title>Pilot's Alphabet</title>
    <meta charset="UTF-8">
    <link href="style/site.css" rel="stylesheet">
    <link href="style/pilot.css" rel="stylesheet">
</head>
<body>
    <div id="nav">
        <ul id='menu'>
            <li><a href='pilots.html'>Pilots</a></li>
            <li><a href='cities.html'>Cities</a></li>
            <li><a href='names.html'>Names</a></li>
        </ul>
    </div>
    <div id="main">
        <h1>Pilot's Alphabet</h1>
        <ul id="pilot">
            <!-- ALPHABET START -->
            <li>Alpha</li>
            <li>Bravo</li>
            <li>charlie</li>
            <li>Delta</li>
            <!-- ALPHABET END -->
        </ul>
    </div>
</body>
</html>
```

# Open cities.html to add "Detroit"

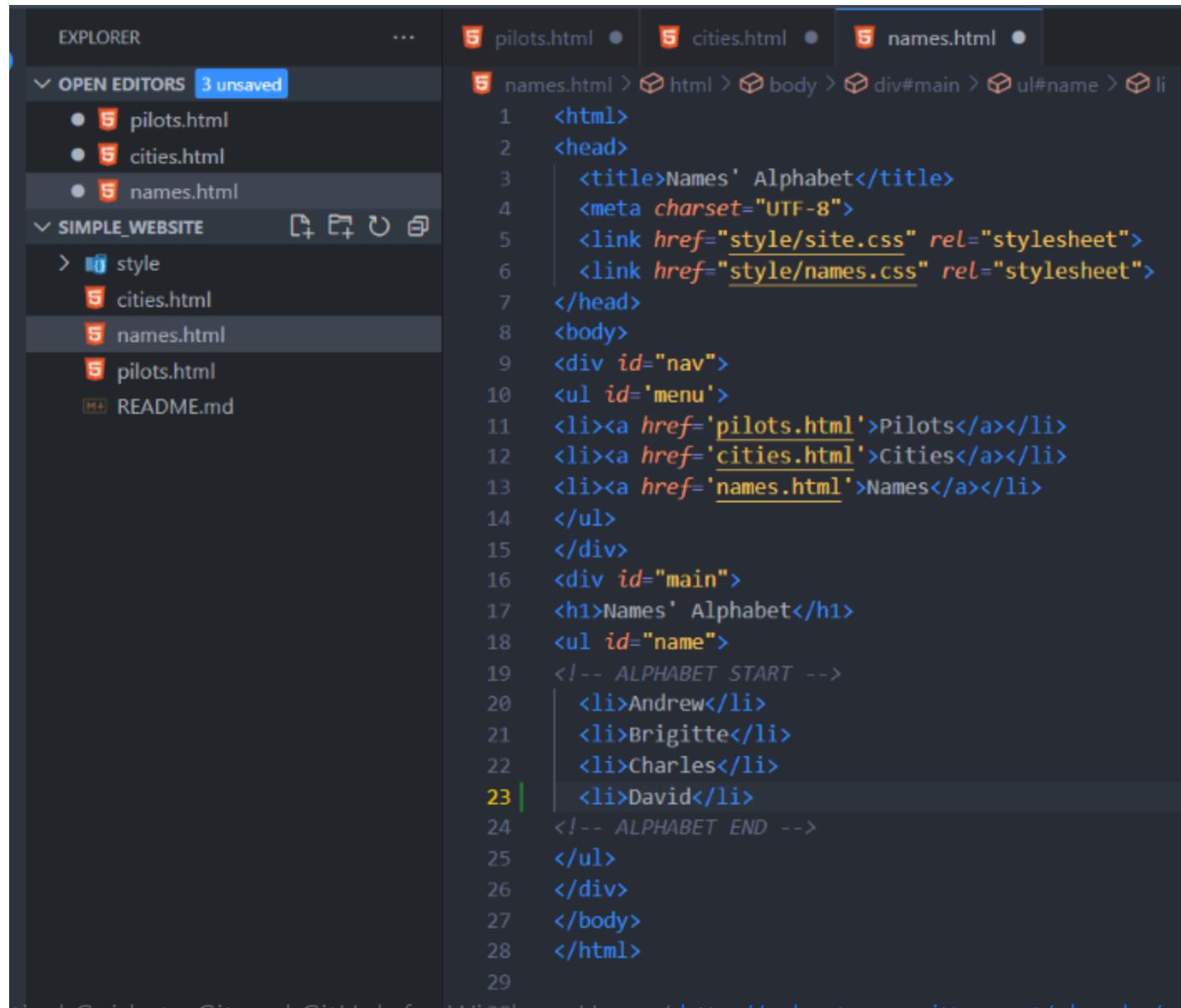


The screenshot shows a code editor interface with the following details:

- EXPLORER** panel on the left:
  - OPEN EDITORS**: 2 unsaved
  - pilots.html
  - cities.html** (highlighted)
- SIMPLE\_WEBSITE** folder:
  - style
  - cities.html** (highlighted)
  - names.html
  - pilots.html
  - README.md
- Editor Area**:

```
cities.html > html > body > div#main > ul#city > li
1   <html>
2     <head>
3       <title>Cities' Alphabet</title>
4       <meta charset="UTF-8">
5       <link href="style/site.css" rel="stylesheet">
6       <link href="style/cities.css" rel="stylesheet">
7     </head>
8     <body>
9       <div id="nav">
10      <ul id='menu'>
11        <li><a href='pilots.html'>Pilots</a></li>
12        <li><a href='cities.html'>Cities</a></li>
13        <li><a href='names.html'>Names</a></li>
14      </ul>
15    </div>
16    <div id="main">
17      <h1>Cities' Alphabet</h1>
18      <ul id="city">
19        <!-- ALPHABET START -->
20        <li>Atlanta</li>
21        <li>Boston</li>
22        <li>Chicago</li>
23        <li>Detroit</li> | (cursor here)
24        <!-- ALPHABET END -->
25      </ul>
26    </div>
27  </body>
28</html>
```

# Open names.html to add "David"



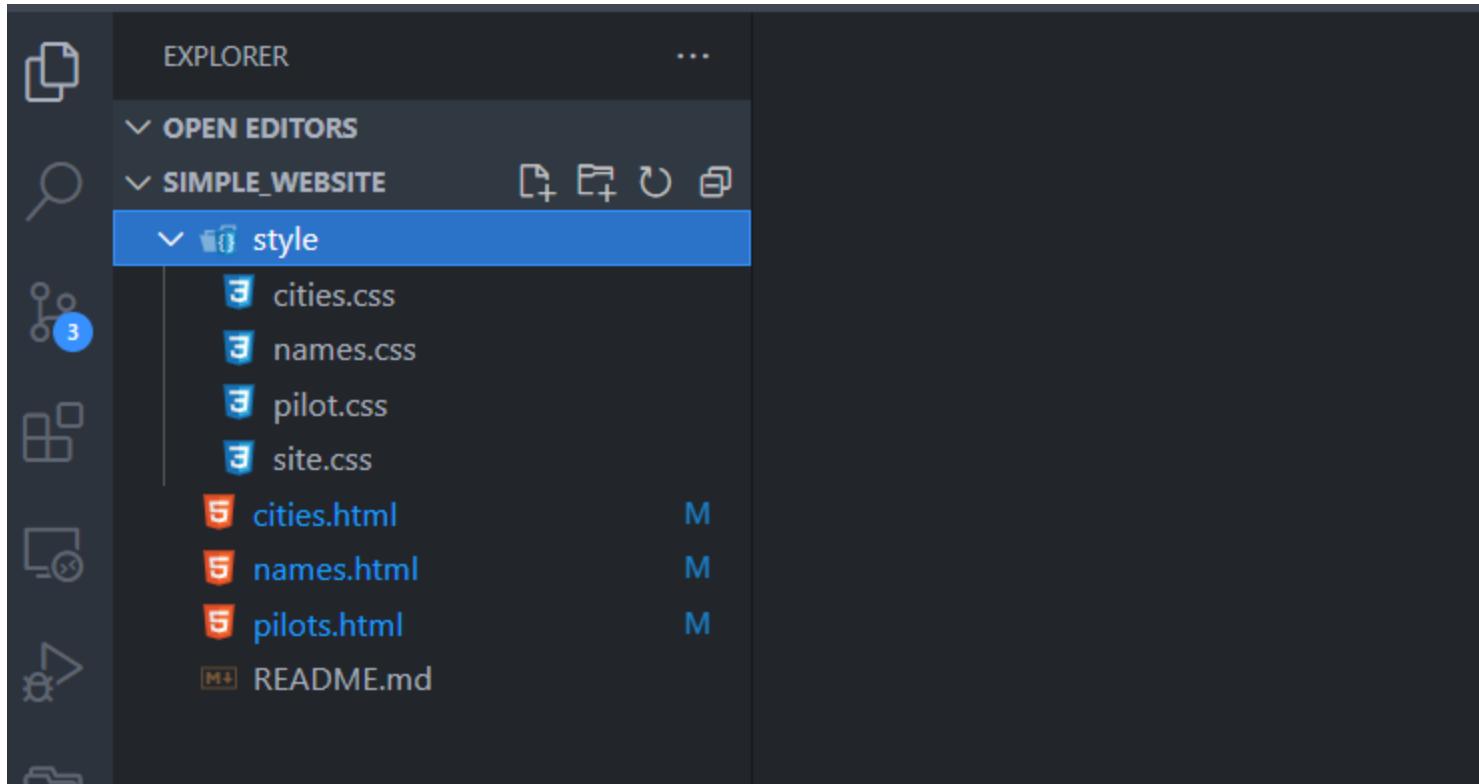
The screenshot shows the VS Code interface with the 'names.html' file open in the editor. The file contains an HTML structure with a navigation menu and a main content area displaying a list of names. The name 'David' is currently being typed at line 23.

```
<html>
<head>
<title>Names' Alphabet</title>
<meta charset="UTF-8">
<link href="style/site.css" rel="stylesheet">
<link href="style/names.css" rel="stylesheet">
</head>
<body>
<div id="nav">
<ul id='menu'>
<li><a href='pilots.html'>Pilots</a></li>
<li><a href='cities.html'>Cities</a></li>
<li><a href='names.html'>Names</a></li>
</ul>
</div>
<div id="main">
<h1>Names' Alphabet</h1>
<ul id="name">
<!-- ALPHABET START -->
<li>Andrew</li>
<li>Brigitte</li>
<li>Charles</li>
23 | <li>David</li>
24 <!-- ALPHABET END -->
25 </ul>
26 </div>
27 </body>
28 </html>
29
```

## Updating the local repository

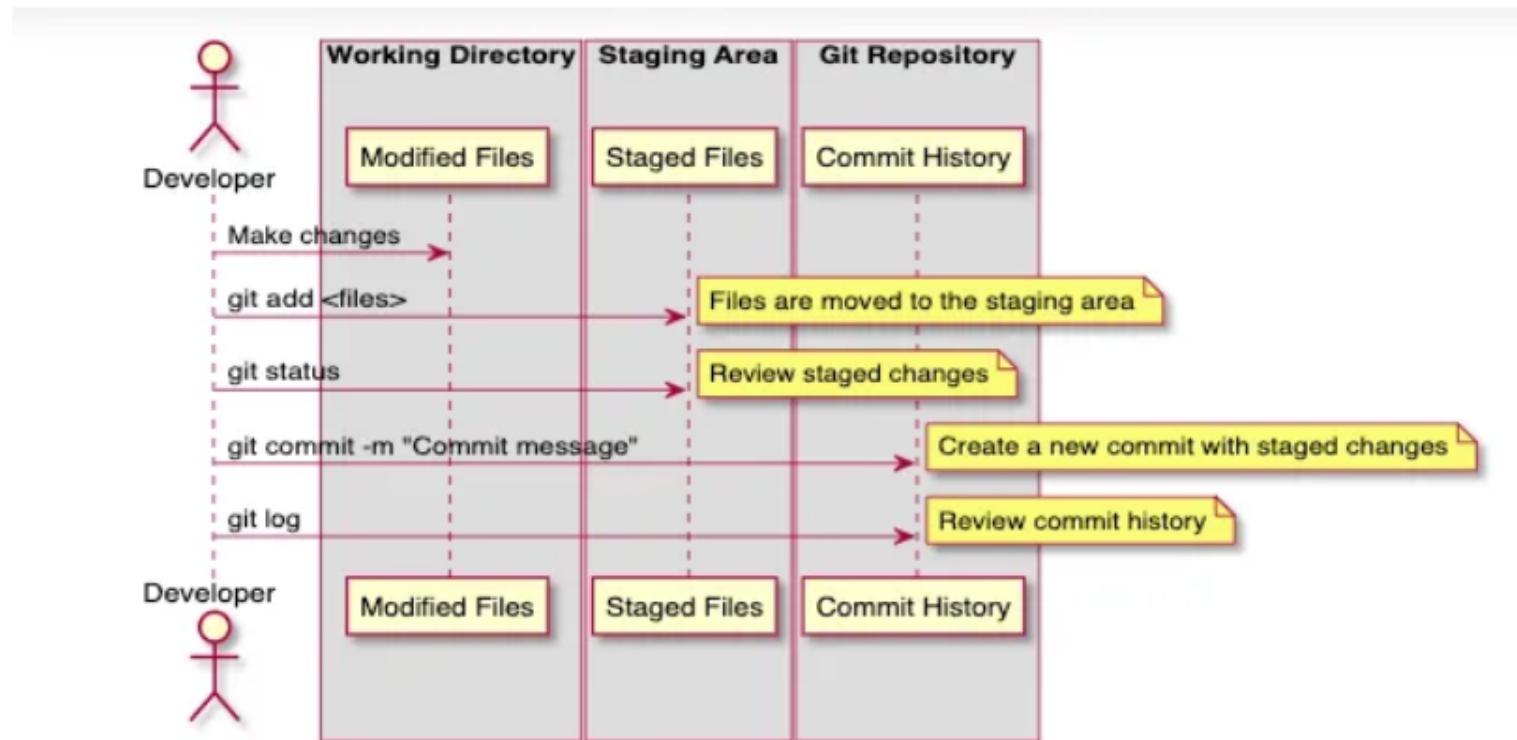
The reason why Git uses two phases - staging and committing - to update repository is that by staging first you can *group all related changes into a single commit* and give it a meaningful comment.

# View from Visual Studio Code



# A graphical vision of Git commit

## git commit process



# Committing letter D

```
14:39:21 simple_website main ↵~3 152ms
→ # Update the local repository
14:39:51 simple_website main ↵~3
→ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   cities.html
    modified:   names.html
    modified:   pilots.html

no changes added to commit (use "git add" and/or "git commit -a")
14:39:57 simple_website main ↵~3 157ms
→ # Let's using 'git commit -am "Added letter D"' to add and commit together
14:41:04 simple_website main ↵~3
→ git commit -am "Added letter D"
[main aaabd6b] Added letter D
 3 files changed, 3 insertions(+)
14:41:23 simple_website main ↵~3 190ms
→ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

# Viewing Project History

```
14:43:45 simple_website main 158ms
→ # Now we're monitor the evolution of project - its history by 'git log'
14:48:36 simple_website main
→ git log
commit aaabd6b12bc31e5deea07f8b54642352e3cdd264 (HEAD -> main)
Author: linduarte <clldscotti@gmail.com>
Date:   Mon Aug 21 14:41:23 2023 -0300

    Added letter D

commit 7cdb449d7826b230aa80a2e8ef69272abc846aef (origin/main, origin/HEAD)
Author: linduarte <clldscotti@gmail.com>
Date:   Mon Aug 21 13:54:37 2023 -0300

    Source code added

commit 0af44e4047a94e671b425da2b564062c7ed13986
Author: linduarte <140444382+linduarte@users.noreply.github.com>
Date:   Mon Aug 21 13:31:49 2023 -0300

    Initial commit
14:48:43 simple_website main 234ms
→ git log --oneline --decorate
aaabd6b (HEAD -> main) Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
14:50:38 simple_website main 156ms
→ git log --oneline --decorate --author=linduarte
aaabd6b (HEAD -> main) Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
```

## What is a Branch?

A branch represents an independent line of development in a project with its own separate history of commits.

You can list the branches in the local repository by running the `git branch` command. The `--remote` switch shows the local copy of the branches in the remote repository on GitHub.

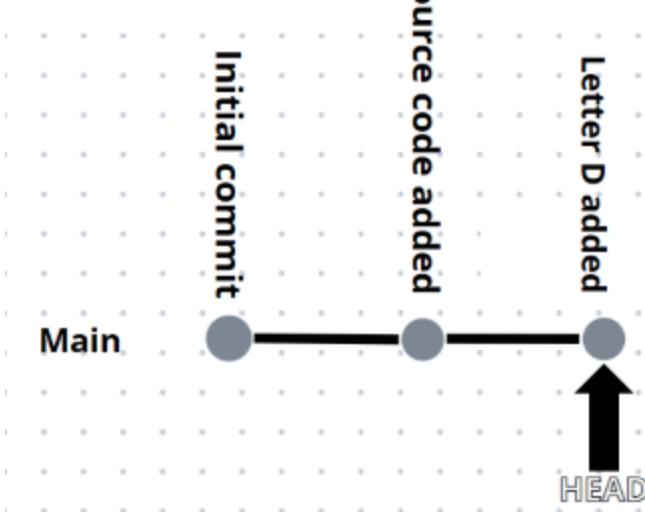
```
14:55:13 simple_website main 176ms
→ # What's a branch?
14:56:59 simple_website main
→ # We can list branches in local repository by running 'git branch'
14:58:19 simple_website main
→ git branch
* main
14:58:31 simple_website main 224ms
→ git branch --remote
origin/HEAD -> origin/main
origin/main
```

## Branch

Right now our repository has only a single branch called **main** both locally and remotely.

When you first setup a repository Git creates the **main** branch automatically for you.

All Git projects have a main branch by default.



The image shows a 'main' trunk representing the main branch.

Each commit is a circle. The arrow symbolizes the HEAD pointer.

The branch pointed to by HEAD is the current or checked out branch.

HEAD -> main It means that HEAD is currently pointing to the main branch. The contents of the working directory reflect the last commit on the checked out branch plus any changes you have made.

## Comparing Versions

It is important at this point to note that there are *several versions* in Git project.

- 1-The **working directory** version (the one you use for editing)
- 2-The **staged** version(after you run **git add** to add the file to the index for the next commit)
- 3-The **committed** versions (one version for each commit)

The **git diff** command shows changes between the working directory, index and commit versions.

## Exercise

Feature request: ***add the letter E*** to the phonetic website.

1 pilots.html

- ***Echo***

2 cities.html

- ***Eldorado***

3 names.html

- ***Eva***

# Check status and add files to index

```
14:18:41 simple_website main 108ms
→ # Adding letter 'D'
14:22:00 simple_website main ~3
→ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   cities.html
    modified:   names.html
    modified:   pilots.html

no changes added to commit (use "git add" and/or "git commit -a")
14:22:11 simple_website main ~3 191ms
→ # Adding the files to the index
14:23:27 simple_website main ~3
→ git add *.html
14:23:36 simple_website main ~3 191ms
→ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   cities.html
    modified:   names.html
    modified:   pilots.html
```

## Exercise

Feature request: *add the letter F* to the phonetic website.

1 pilots.html

- *Foxtrot*

2 cities.html

- *Fillmore*

3 names.html

- *Fred*

## Let's check the status after letter 'F' added

```
14:49:46 simple_website main ~3 | ~3 168ms
→ # Letter F added
14:50:08 simple_website main ~3 | ~3
→ git status
On branch main
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   cities.html
    modified:   names.html
    modified:   pilots.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   cities.html
    modified:   names.html
    modified:   pilots.html
```

After checking the status we have now two different versions of the HTML files.

One version contains the staged changes (letter D). The other one contains the unstaged changes we did adding letter 'F'.

And of course we also have the committed versions as show by **git log** in the next slide.

## Let's check `git log` and viewing unstaged changes by `git diff`

```
15:00:24 simple_website ⚡ main ✘ ~3 | 🖍~3 114ms
→ git log --oneline --decorate
aaabd6b (HEAD -> main) Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
15:00:50 simple_website ⚡ main ✘ ~3 | 🖍~3 186ms
→ # Using git diff to view the differences between the varios versions
15:04:37 simple_website ⚡ main ✘ ~3 | 🖍~3
→ git diff pilots.html
diff --git a/pilots.html b/pilots.html
index fb4318f..32706f8 100644
--- a/pilots.html
+++ b/pilots.html
@@ -22,6 +22,7 @@
    <li>Charlie</li>
    <li>Delta</li>
    <li>Echo</li>
+   <li>Foxtrot</li>
    <!-- ALPHABET END -->
  </ul>
</div>
```

Interpreting the output of the `git diff pilots.html` command line by line.

1. `diff --git a/pilots.html b/pilots.html`: This line indicates that the `git diff` command is comparing two versions of the file `pilots.html`. The `a/pilots.html` represents the original file, while `b/pilots.html` represents the modified file.
2. `index fb4318f..32706f8 100644`: This line provides information about the file index. It shows the commit hashes (`fb4318f` and `32706f8`) of the original and modified versions of `pilots.html`. The `100644` represents the file mode, which is typically associated with regular files.

3. `---` `a/pilots.html`: This line indicates the start of the section related to the original version of the file (`a/pilots.html`).
4. `+++ b/pilots.html`: This line indicates the start of the section related to the modified version of the file (`b/pilots.html`).
5. `@@ -22,6 +22,7 @@`: This line is the unified diff header. It provides information about the line numbers and context for the changes that follow:
  - `-22,6`: This means that the change begins at line 22 in the original file and affects 6 lines.
  - `+22,7`: This means that the change begins at line 22 in the modified file and affects 7 lines. The `+` indicates that a line has been added.

## Viewing changes between the index and the last commit

For this we have to use the switch --cached .

```
15:26:22  simple_website  main ~3 | ~3  160ms
→ # Viewing changes between the index and the last commit
15:27:02  simple_website  main ~3 | ~3
→ git diff --cached pilots.html
diff --git a/pilots.html b/pilots.html
index ce31f88..fb4318f 100644
--- a/pilots.html
+++ b/pilots.html
@@ -21,6 +21,7 @@
    <li>Bravo</li>
    <li>Charlie</li>
    <li>Delta</li>
+   <li>Echo</li>
    <!-- ALPHABET END -->
  </ul>
</div>
```

## Viewing changes since the last commit

```
15:35:27 simple_website main ~3 | ~3 161ms
→ # Viewing changes since last commit
15:35:54 simple_website main ~3 | ~3
→ git diff HEAD pilots.html
diff --git a/pilots.html b/pilots.html
index ce31f88..32706f8 100644
--- a/pilots.html
+++ b/pilots.html
@@ -21,6 +21,8 @@
    <li>Bravo</li>
    <li>Charlie</li>
    <li>Delta</li>
+   <li>Echo</li>
+   <li>Foxtrot</li>
    <!-- ALPHABET END -->
  </ul>
```

## Viewing changes between any two commits

```
15:53:19  simple_website main ~3 | ~3  174ms
→ # Viewing changes between any two commits
15:53:41  simple_website main ~3 | ~3
→ git diff HEAD~1 HEAD pilots.html
diff --git a/pilots.html b/pilots.html
index 9dd516f..ce31f88 100644
--- a/pilots.html
+++ b/pilots.html
@@ -20,6 +20,7 @@
<li>Alpha</li>
<li>Bravo</li>
<li>Charlie</li>
+ <li>Delta</li>
<!-- ALPHABET END -->
</ul>
</div>
```

We can use the short hash obtained from `git log` to view the difference between any two commits

```
 16:02:30 simple_website main ~3 | ~3 167ms
→ # Git log to view the difference between any two commits
 16:11:40 simple_website main ~3 | ~3
→ git log --oneline --decorate
aaabd6b (HEAD -> main) Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
 16:11:58 simple_website main ~3 | ~3 196ms
→ git diff 7cdb449 aaabd6b pilots.html
diff --git a/pilots.html b/pilots.html
index 9dd516f..ce31f88 100644
--- a/pilots.html
+++ b/pilots.html
@@ -20,6 +20,7 @@
    <li>Alpha</li>
    <li>Bravo</li>
    <li>Charlie</li>
+   <li>Delta</li>
    <!-- ALPHABET END -->
  </ul>
  </div>
```

# Let's commit letter 'E', git add for letter 'F' and commit.

```
16:38:24 simple_website main ~3 | ~3 174ms
→ # Let's commit letter 'E'
16:39:25 simple_website main ~3 | ~3
→ git commit -m "Added letter E"
[main d51985c] Added letter E
  3 files changed, 3 insertions(+)
16:40:00 simple_website main ~3 258ms
→ #Let's add and commit letter "F"
16:40:31 simple_website main ~3
→ git commit -am "Added letter F"
[main d14a0b3] Added letter F
  3 files changed, 3 insertions(+)
16:40:58 simple_website main 181ms
→ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
16:41:11 simple_website main 138ms
→ git log --oneline --decorate
d14a0b3 (HEAD -> main) Added letter F
d51985c Added letter E
aaabd6b Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
```

## What if you want to undo changes

One of the reasons to track a project's history is to have the ability to **undo** changes.

It is a common situation in software development: you change something, you test the change and it does not quite work the way you expected.

You then decide to revert the code back to the previous version.

With Git you can easily accomplish that.

## **There are three possible scenarios in Git**

- 1 - Undoing changes in the working directory before staging
- 2 - Undoing changes after staging and before committing
- 3 - Undoing committed changes

## Exercise : Unwanted Change

Let's demonstrate the three scenarios using the Phonetic Website.

Open `pilots.html` and delete all words except "Alpha".

```
<!-- ALPHABET START -->
| <li>Alpha</li>
> <!-- ALPHABET END -->
```

Now we will learn how to recover the lost words using **Git undo** features.

The idea is that you have to change dozens of lines of code in various parts of the source file.

# Undoing Changes Before Staging

To recover the lost words in `pilots.html` all you have to do is to revert the file to its last 'modified' version using the `git checkout` command:

```
05:16:40 simple_website \main 222ms
→ # Undoing changes before staging
05:45:35 simple_website \main ~1
→ git checkout pilots.html
Updated 1 path from the index
05:47:02 simple_website \main 156ms
→ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

**Browse the website to verify that the words have been recovered.**

```
<!-- ALPHABET START -->
<li>Alpha</li>
<li>Bravo</li>
<li>Charlie</li>
<li>Delta</li>
<li>Echo</li>
<li>Foxtrot</li>
<!-- ALPHABET END -->
```

# Undoing Changes After Staging

Open `pilots.html` and delete all words except "Alpha".

```
06:04:12 simple_website ⚭main ✎~1 158ms
→ # Undoing changes after staging
06:04:53 simple_website ⚭main ✎~1
→ git add pilots.html
06:05:15 simple_website ⚭main ✅~1 166ms
→ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   pilots.html
```

After running `git status` it shows that `pilots.html` is ready to be committed.

It also suggests to run `git reset HEAD (file)` to un-stage the change.

```
06:16:18 simple_website main ~1 150ms
→ # Run git reset HEAD to un-stage the change
06:17:22 simple_website main ~1
→ git reset HEAD pilots.html
Unstaged changes after reset:
M     pilots.html
06:17:59 simple_website main ~1 188ms
→ git status
On branch main
Your branch is ahead of 'origin/main' by 3 commits.
  (use "git push" to publish your local commits)

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   pilots.html

no changes added to commit (use "git add" and/or "git commit -a")
```

## Git restore x Git reset

Both `git restore` and `git reset` are used to undo changes in your repository. However, they work differently.

`git restore` is used for restoring files in the working tree from the index or another commit. This doesn't update the branch. On the other hand, `git reset` is used to update your branch.

So, if you want to undo changes that you have made to a file in your working directory but haven't staged yet, you can use `git restore`. If you want to undo changes that you have made to a file and have already staged it, you can use `git reset`.

The **git reset** command has removed the file from the index (staging area) however the unwanted change is still in the working directory.

To recover the lost words, you still need to repeat the process for undoing un-staged changes and run **git checkout** to restore the last 'modified' version.

```
06:59:43 simple_website ↴main ~1 153ms
→ # Recovering the lost words
07:00:20 simple_website ↴main ~1
→ git checkout pilots.html
Updated 1 path from the index
```

Browse the website to verify that the words have been recovered.

## Undoing Committed Changes

Open `pilots.html` and delete all words except "Alpha".

Now we're going to commit the 'unwanted change'.

# Committing "unwanted changes"

```
07:03:11 simple_website ↴main 158ms
→ # Undoing committed changes
07:15:35 simple_website ↴main ~1
→ # Commit the unwanted changes
07:16:18 simple_website ↴main ~1
→ git add pilots.html
07:17:02 simple_website ↴main ~1 125ms
→ git commit -m "Unwanted changes"
[main 032aca5] Unwanted changes
 1 file changed, 5 deletions(-)
07:17:31 simple_website ↴main 172ms
→ git log --oneline --decorate
032aca5 (HEAD -> main) Unwanted changes
d14a0b3 Added letter F
d51985c Added letter E
aaabd6b Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
```

## Undoing the committed change running `git revert` using the short hash of the unwanted commit.

```
 07:23:34  simple_website main 164ms
→ git log --oneline --decorate
032aca5 (HEAD -> main) Unwanted changes
d14a0b3 Added letter F
d51985c Added letter E
aaabd6b Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
 07:23:38  simple_website main 190ms
→ # Undoing the committed change
 07:24:20  simple_website main
→ git revert 032aca5 --no-edit
[main 6d4e190] Revert "Unwanted changes"
Date: Wed Aug 23 07:25:24 2023 -0300
1 file changed, 5 insertions(+)
```

## Explaining the `git revert 032aca5 --no-edit` command:

- `git revert` : This is a Git command used to create a new commit that undoes the changes introduced by a previous commit. It's a way to reverse changes without removing commit history.
- `032aca5` : This is the commit hash or identifier of the commit you want to revert. In this case, you are specifying a commit with the hash `032aca5` as the target.

## Continue

- `--no-edit`: This is an option used with `git revert`. When you use `--no-edit`, it tells Git to create the revert commit without opening the default text editor for you to edit the commit message. Instead, Git will automatically generate a commit message for the revert based on the commit you're reverting.

So, when you run `git revert 032aca5 --no-edit`, Git will create a new commit that undoes the changes introduced by the commit with the hash `032aca5`, and it will do so without asking you to edit the commit message. Git will generate a default commit message indicating that this new commit is a revert of the specified commit. This is useful when you want to quickly create a revert commit without modifying the commit message.

## Checking the project history

```
07:38:54 simple_website main 185ms
→ # Let's check the project history
07:39:18 simple_website main
→ git log --oneline --decorate
6d4e190 (HEAD -> main) Revert "Unwanted changes"
032aca5 Unwanted changes
d14a0b3 Added letter F
d51985c Added letter E
aaabd6b Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
```

# Tagging Versions

You can use Git to attach a tag to easily identify a stable version of a project with **git tag** command.

## 1- Create a Tag:

To create a tag, you can use the **git tag** command followed by a tag name. For example, if you want to tag a version 1.0 of your Python project, you can do:

```
git tag 1.0
```

## 2- List Tags:

To see a list of tags in your repository, you can use:

```
git tag
```

### 3- Tagging Commits:

You can also tag specific commits by specifying the commit's SHA-1 hash:

```
git tag -a v1.1 <commit-SHA-1>
```

This creates an annotated tag with a message. Annotated tags are recommended for versioning.

## Tag - Continue

### 4 - Push Tags:

Tags are not automatically pushed to remote repositories when you push changes. To push tags, you can use:

```
git push origin --tags
```

This sends your tags to the remote repository.

### 5 - Checkout Tags:

You can check out a specific tag to view the code at that version:

```
git checkout v1.0
```

## 6 - Delete Tags:

To delete a tag, use the -d option:

```
git tag -d 1.0
```

## 7 - To delete a tag on the remote repository:

```
git push origin --delete 1.0
```

# Tagging sample

```
09:05:37 simple_website ⚡main 106ms
→ # Tagging version
09:06:08 simple_website ⚡main
→ # Let's find out the short hash for the commit you want to version
09:07:17 simple_website ⚡main
→ git log --oneline --decorate
6d4e190 (HEAD -> main) Revert "Unwanted changes"
032aca5 Unwanted changes
d14a0b3 Added letter F
d51985c Added letter E
aaabd6b Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
09:07:32 simple_website ⚡main 200ms
→ # Let's use the "Added letter E" - d51985c
09:09:02 simple_website ⚡main
→ git tag -a v0.1 d51985c -m "v0.1"
09:10:22 simple_website ⚡main 174ms
→ git log --oneline --decorate
6d4e190 (HEAD -> main) Revert "Unwanted changes"
032aca5 Unwanted changes
d14a0b3 Added letter F
d51985c (tag: v0.1) Added letter E
aaabd6b Added letter D
7cdb449 (origin/main, origin/HEAD) Source code added
0af44e4 Initial commit
```

# Checking status and update GitHub

```
09:24:52 simple_website main 231ms
→ git status
On branch main
Your branch is ahead of 'origin/main' by 5 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
09:26:45 simple_website main 172ms
→ git push origin main
Enter passphrase for key 'C:/users/clldu/onedrive/documentos/ssh_keys/.ssh/id_ed25519':
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 16 threads
Compressing objects: 100% (19/19), done.
Writing objects: 100% (19/19), 1.67 KiB | 569.00 KiB/s, done.
Total 19 (delta 12), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (12/12), completed with 3 local objects.
To github.com:linduarte/simple_website.git
  7cdb449..6d4e190  main -> main
```

# Commit Messages for Effective Version Control

Commit types			
Commit Type	Title	Description	Emoji
feat	Features	A new feature	✨
fix	Bug Fixes	A bug Fix	🐛
docs	Documentation	Documentation only changes	📝
style	Styles	Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)	💡
refactor	Code Refactoring	A code change that neither fixes a bug nor adds a feature	📦
perf	Performance Improvements	A code change that improves performance	🚀
test	Tests	Adding missing tests or correcting existing tests	⚠️
build	Builds	Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)	🛠
ci	Continuous Integrations	Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)	⚙️
chore	Chores	Other changes that don't modify src or test files	♻️
revert	Reverts	Reverts a previous commit	⤵

## Special script to add to '.bashrc' file

```
# Your existing configuration settings

# Start SSH agent if not already running
```
if [ -z "$SSH_AUTH_SOCK" ]; then
    eval $(ssh-agent -s)
    ssh-add
fi
```

Other configurations and aliases if applicable