Module four

Working With Branches

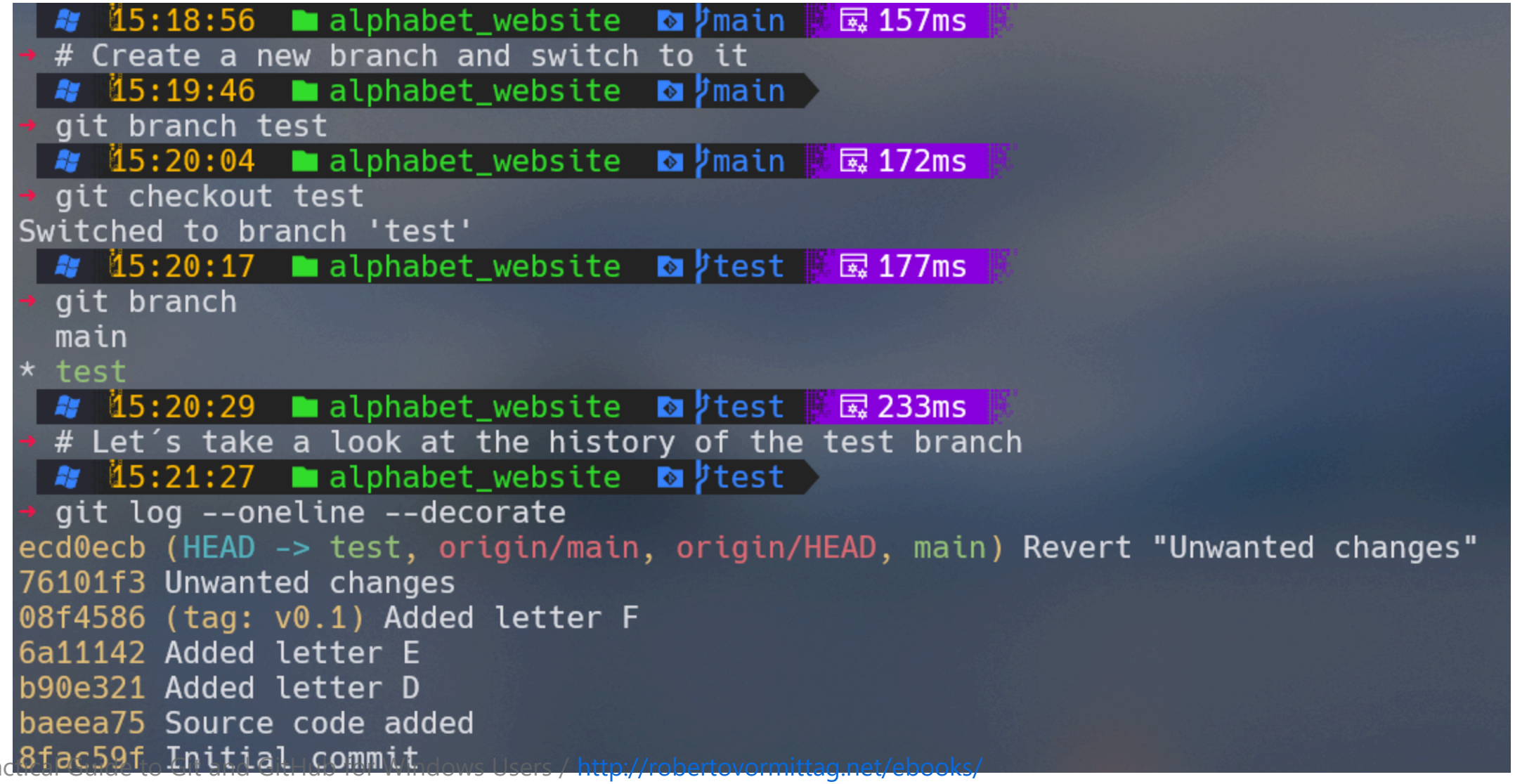
1 - **Moving, deleting and renaming files**

2 - **Switching branches without merging**

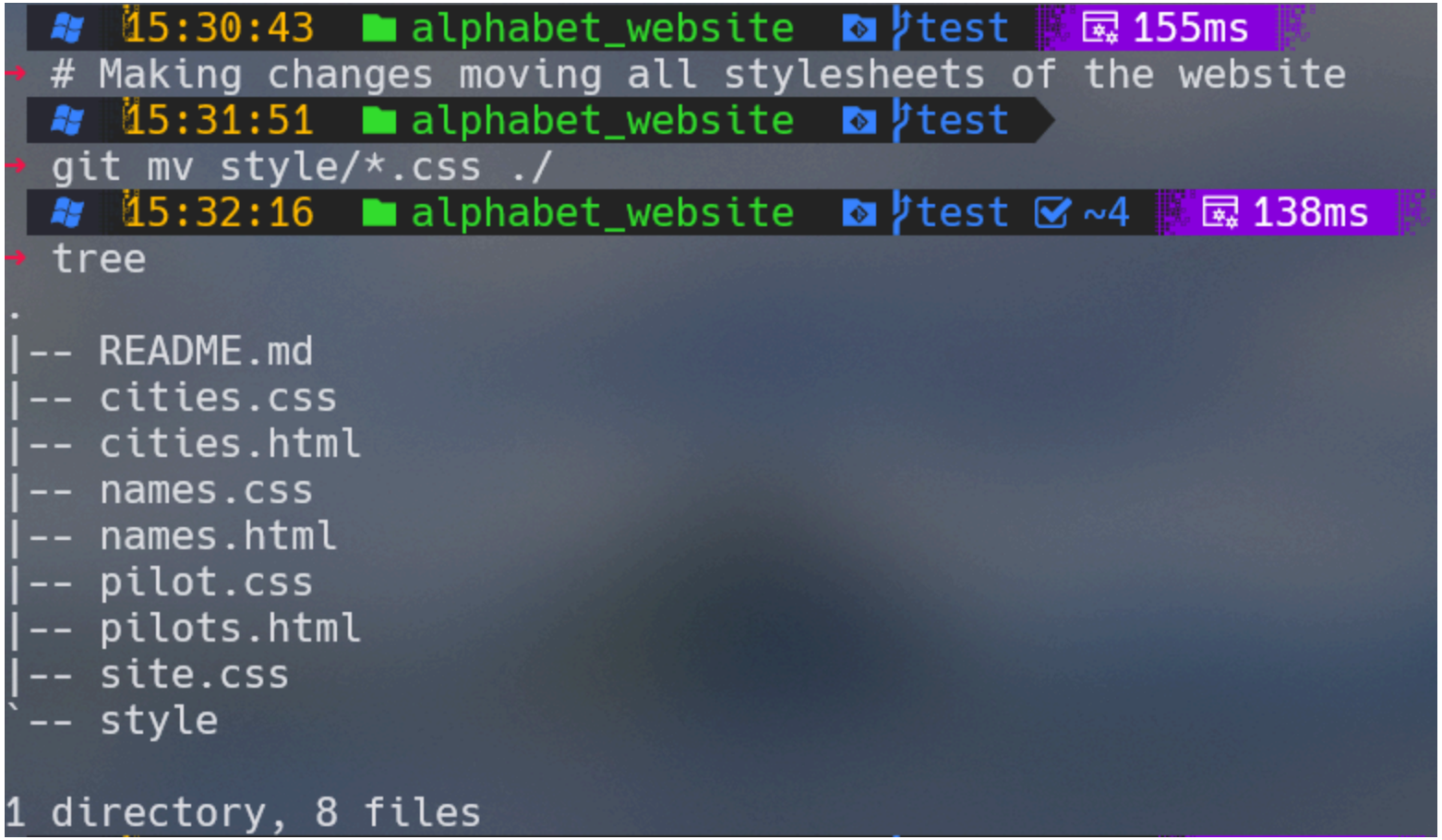
3 - **Merging**

4 - **Resolving conflicts**

Create a new branch



Making changes moving all the stylesheets



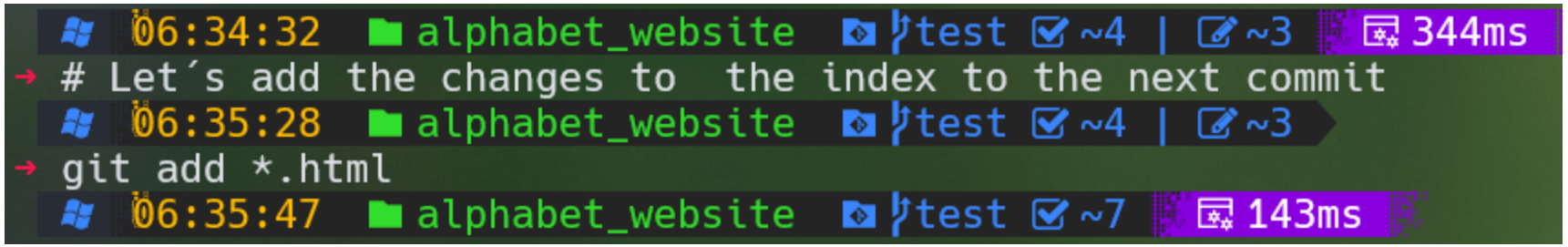
Extract 'style' folder to fix all '*.html files

```
<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>
```

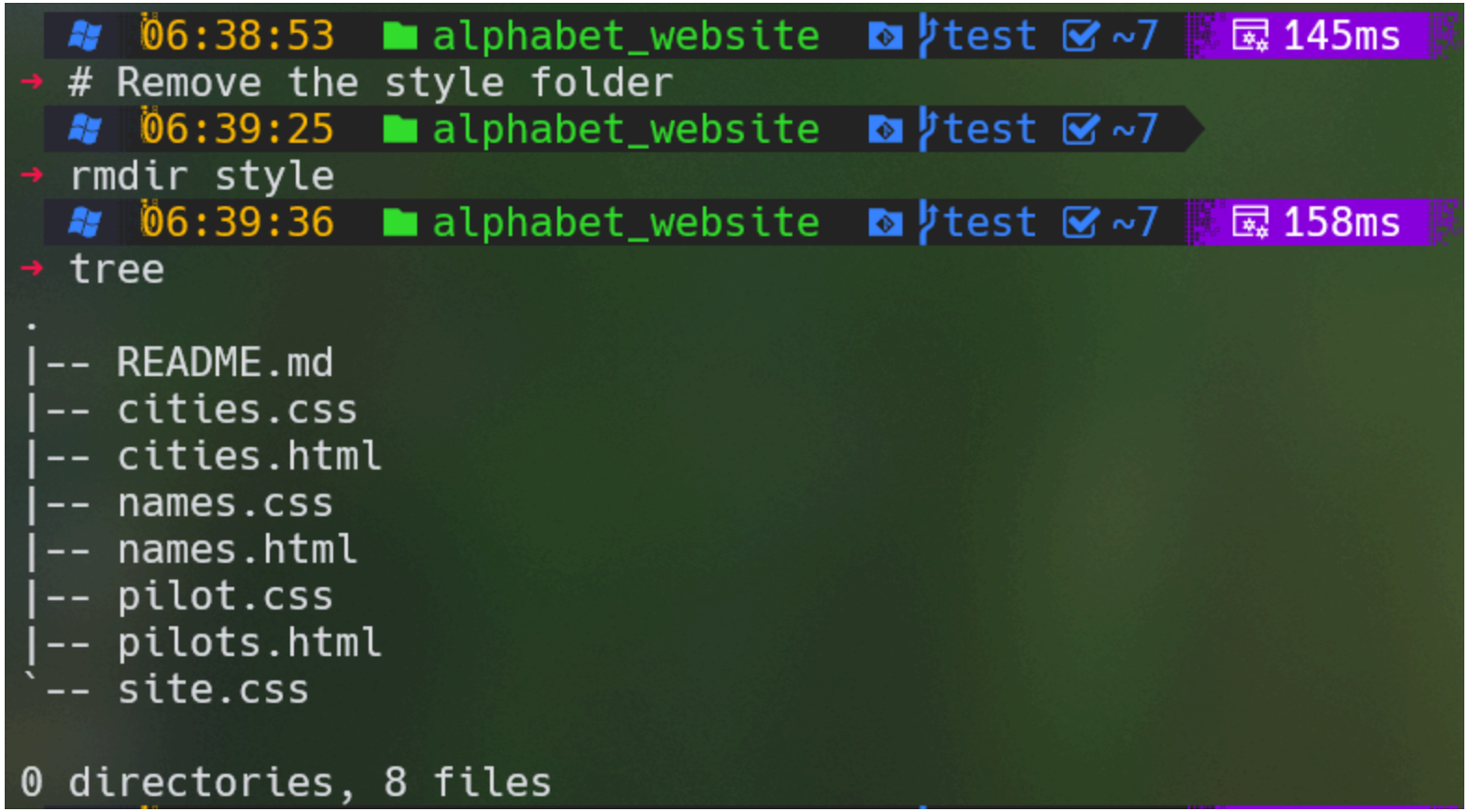
```
<head>
  <title>Pilot's Alphabet</title>
  <meta charset="UTF-8">
  <link href="site.css" rel="stylesheet">
  <link href="pilot.css" rel="stylesheet">
</head>
```

Repeat this changes to cities.html and names.html

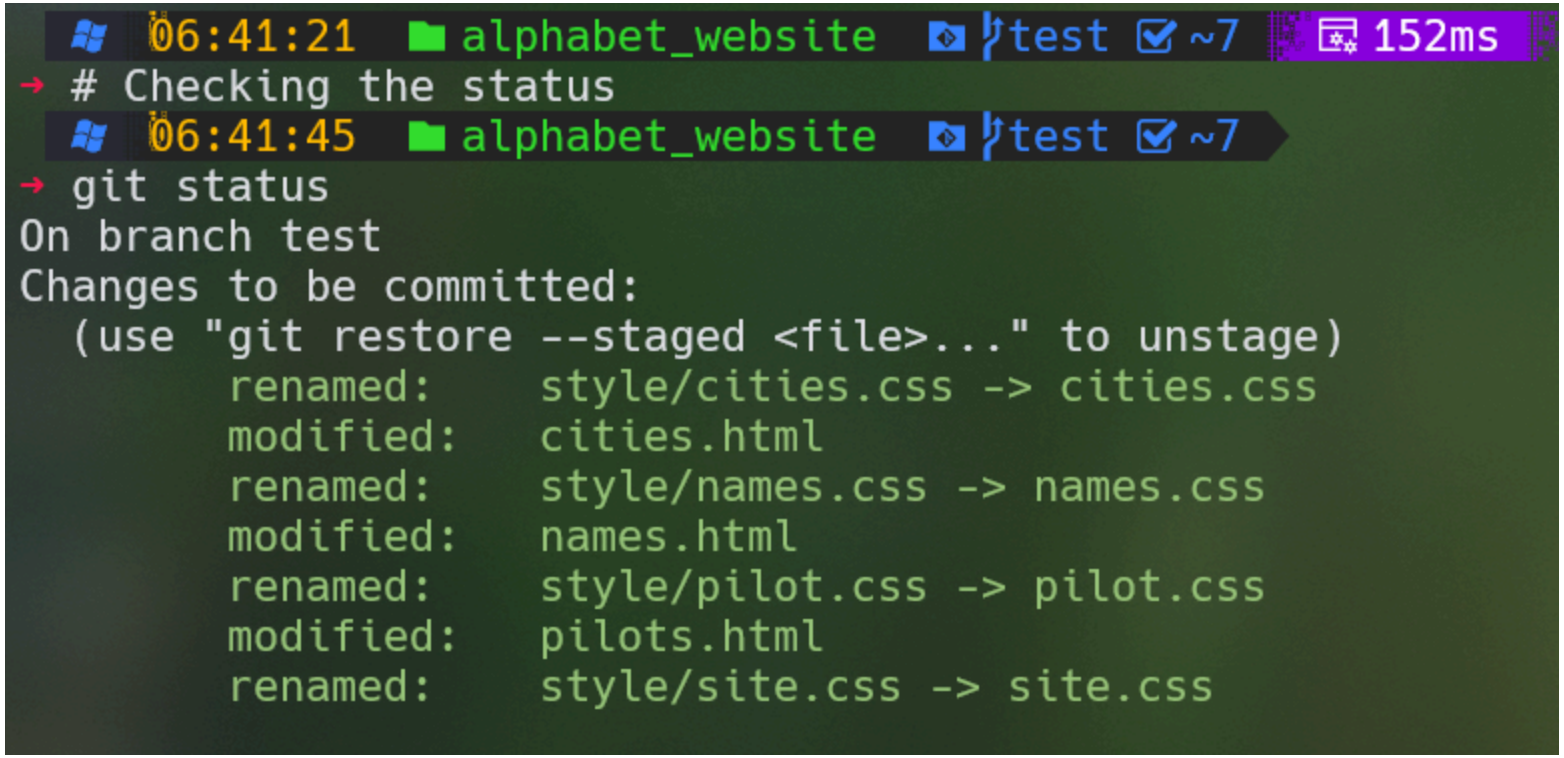
Adding changes to html files



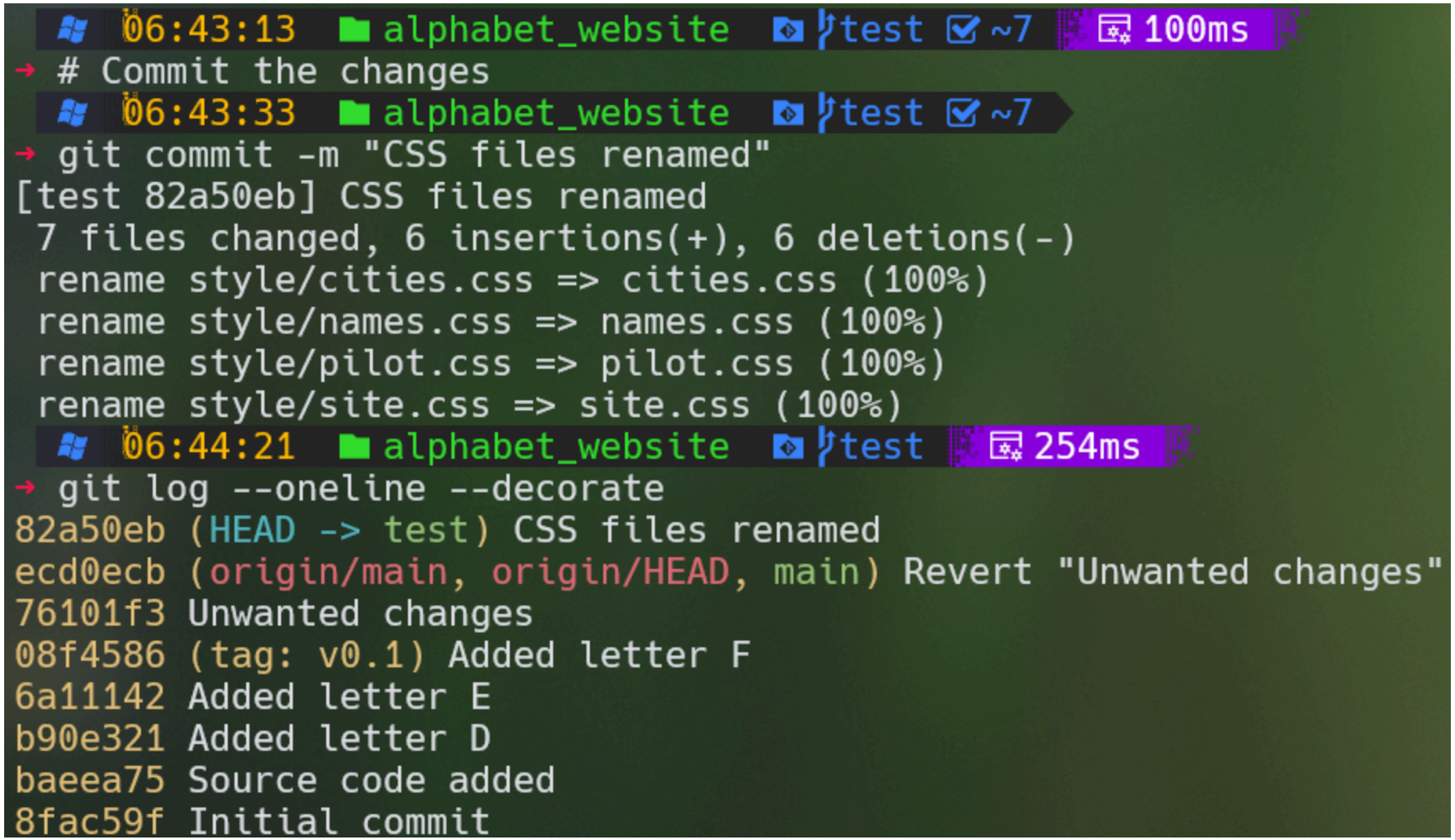
Removing style folder



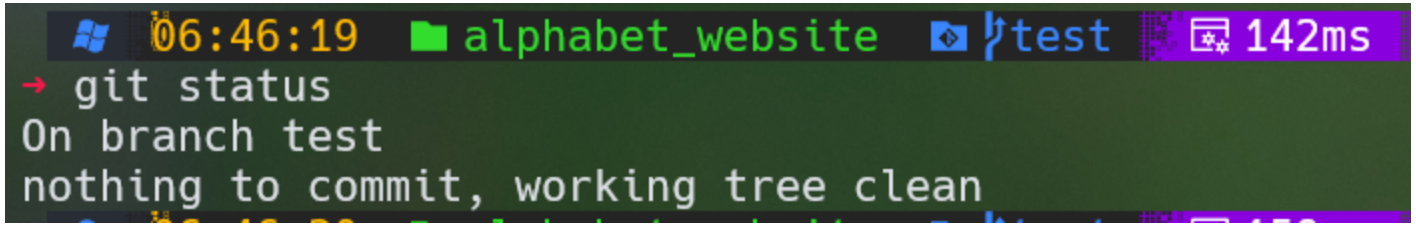
Checking status



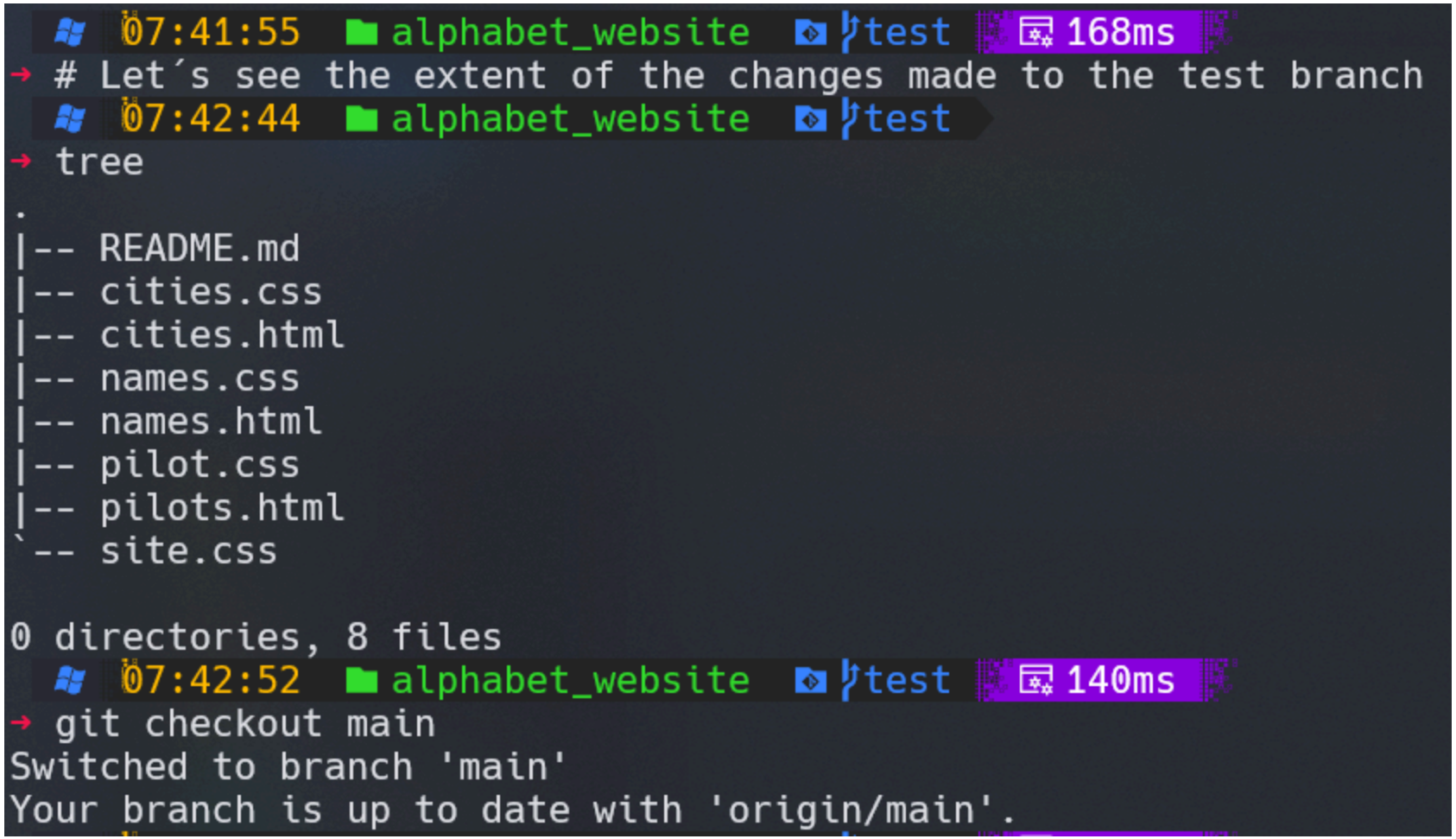
Committing the changes



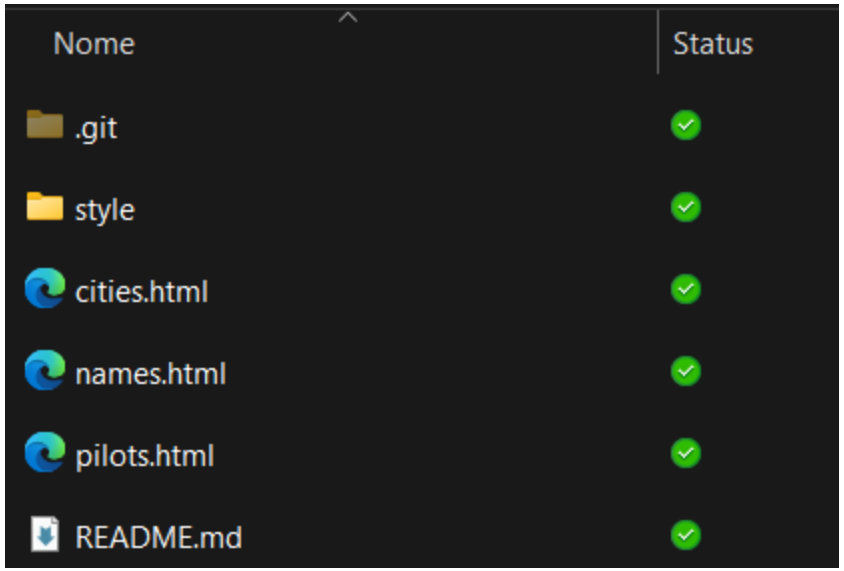
Status up to now



Switching branches without merging



Alphabet_website



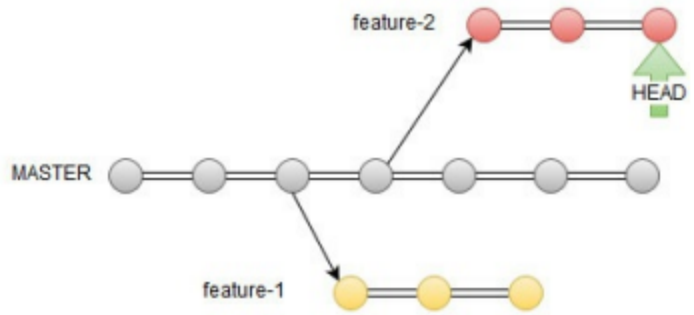
How is that possible?

Every time you switch branches Git will *fetch from the repository database* the contents of the working directory from the last commit on that branch.

When you run the *git checkout <branch>* command the HEAD pointer moves to the last commit of the branch you are checking out, and the content of the working directory reflects what is pointed to by HEAD.

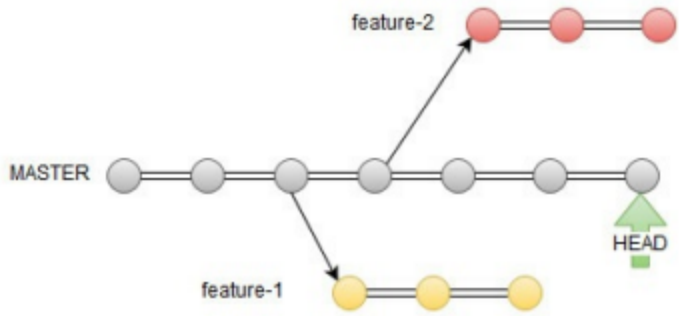
1- Graphic illustration of the switch process

The picture the branch feature-2 is the current branch

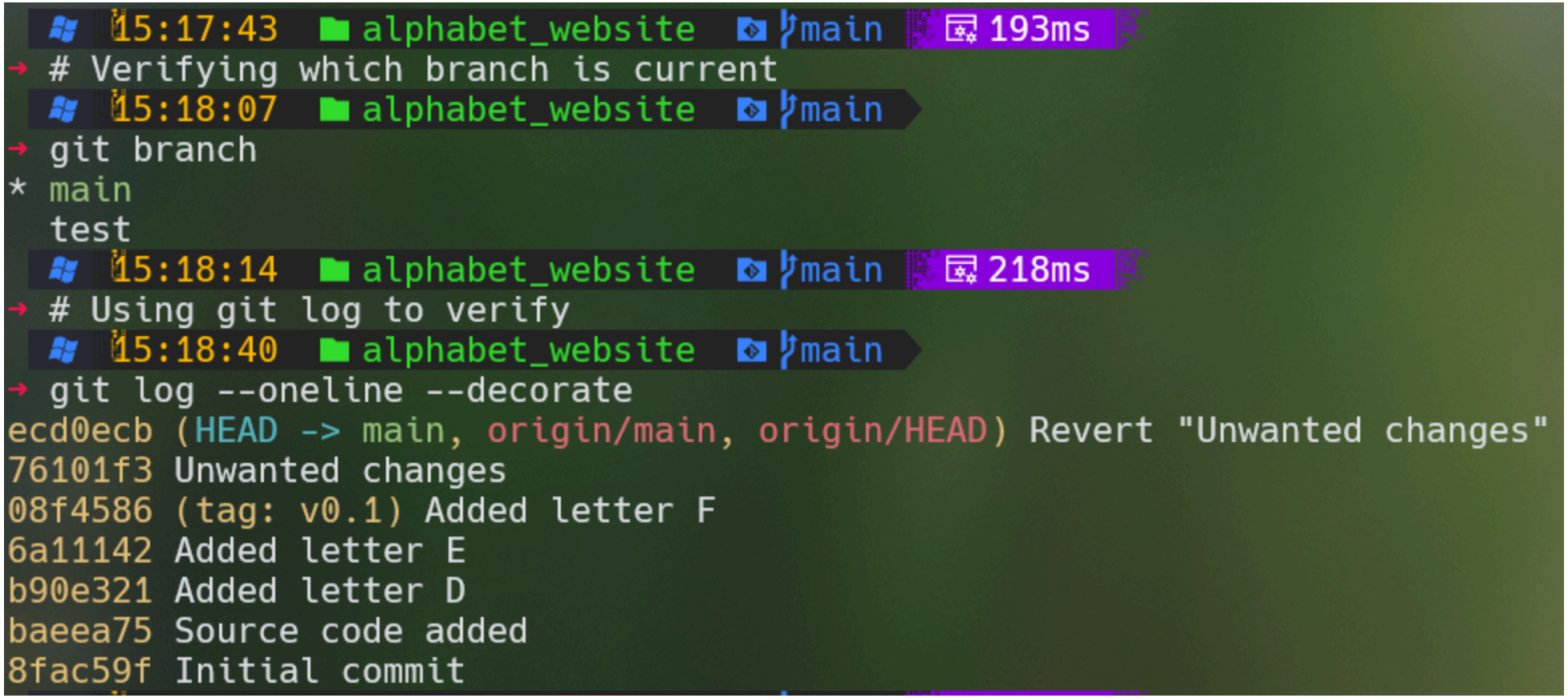


2- Graphic illustration of the switch process

The picture is showing when you command __'git checkout main'__ the HEAD pointer moves to the __last commit of the main branch__ and the contents of the working directory.



You can always verify which branch is current with either git branch or git log.



Special note:

In case you have uncommitted changes when you switch to another branch, Git will try to merge those changes on to the target branch. If the changes are incompatible, then Git will not allow the switch. You can force the switch using the -f option e.g. git checkout -f main if you don't care about loosing the uncommitted changes. It is good practice to switch branches only when the working directory is clean.

Merging

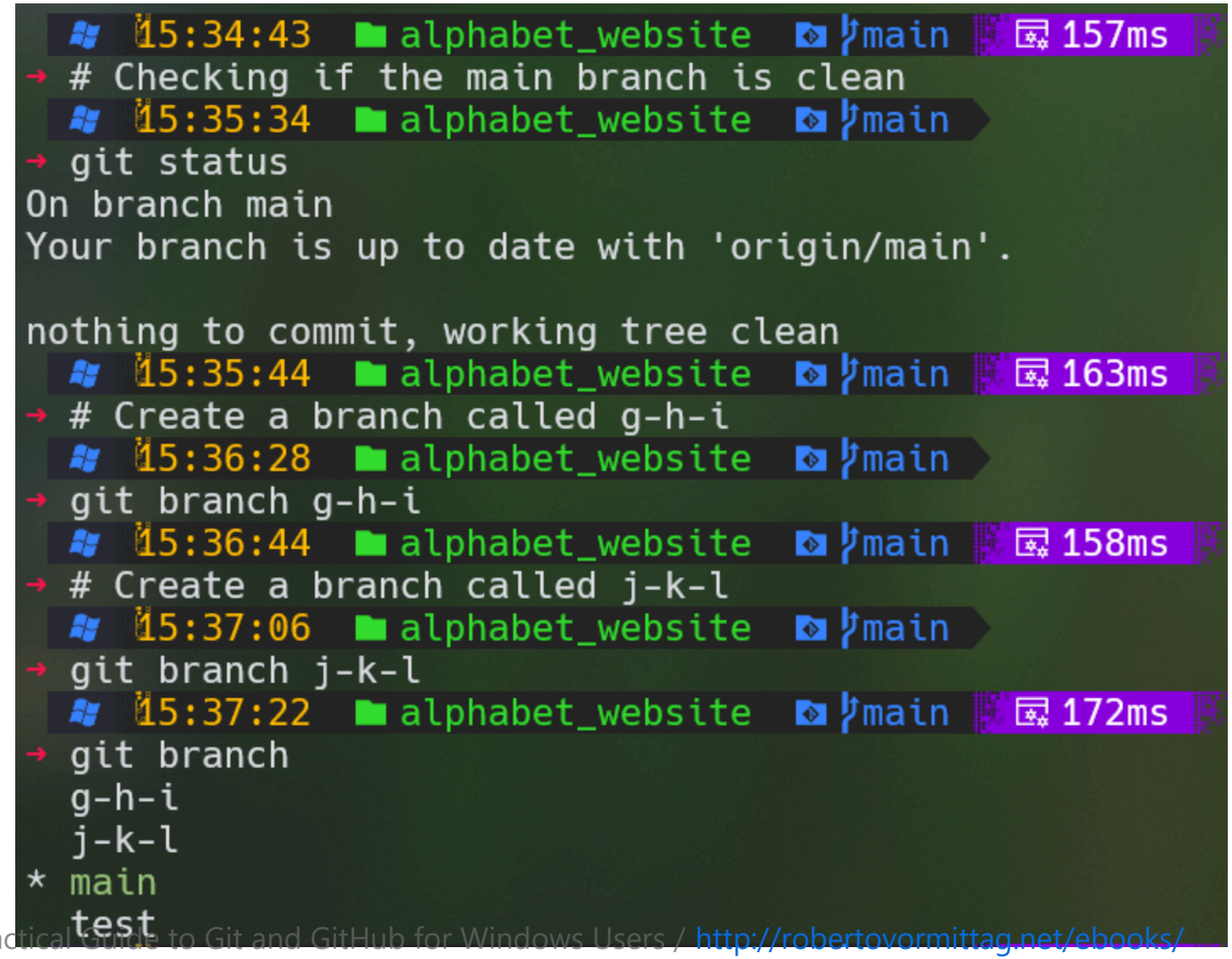
In a typical Git workflow developers normally implement a feature request on a separate branch. This way they can freely experiment with changes without affecting the main code base. The changes are thoroughly tested in the feature branch. Only after successful testing the changes are merged on to the main development branch.

Implementing new features requests

First one is to add the letters G,H,I to the alphabet_website

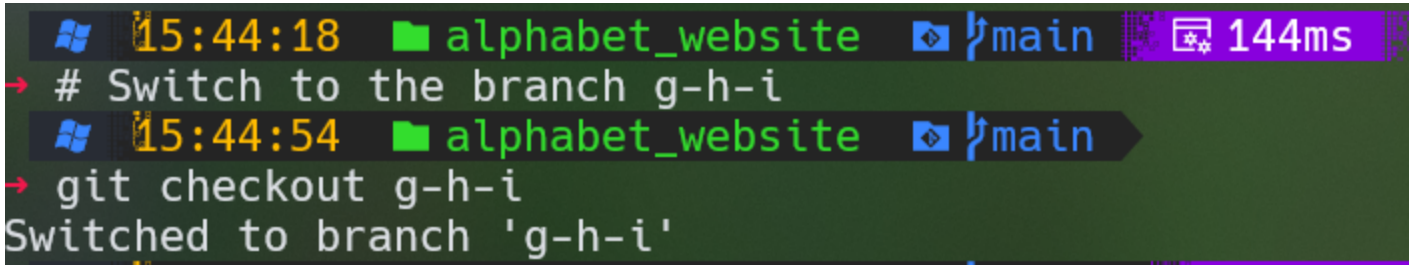
Second one is to add the letters J,K,L to the alphabet_website

Creating new branches



Let´s add the letters G-H-I to the branch g-h-i

To add the letters G-H-I first switch to the g-h-i branch



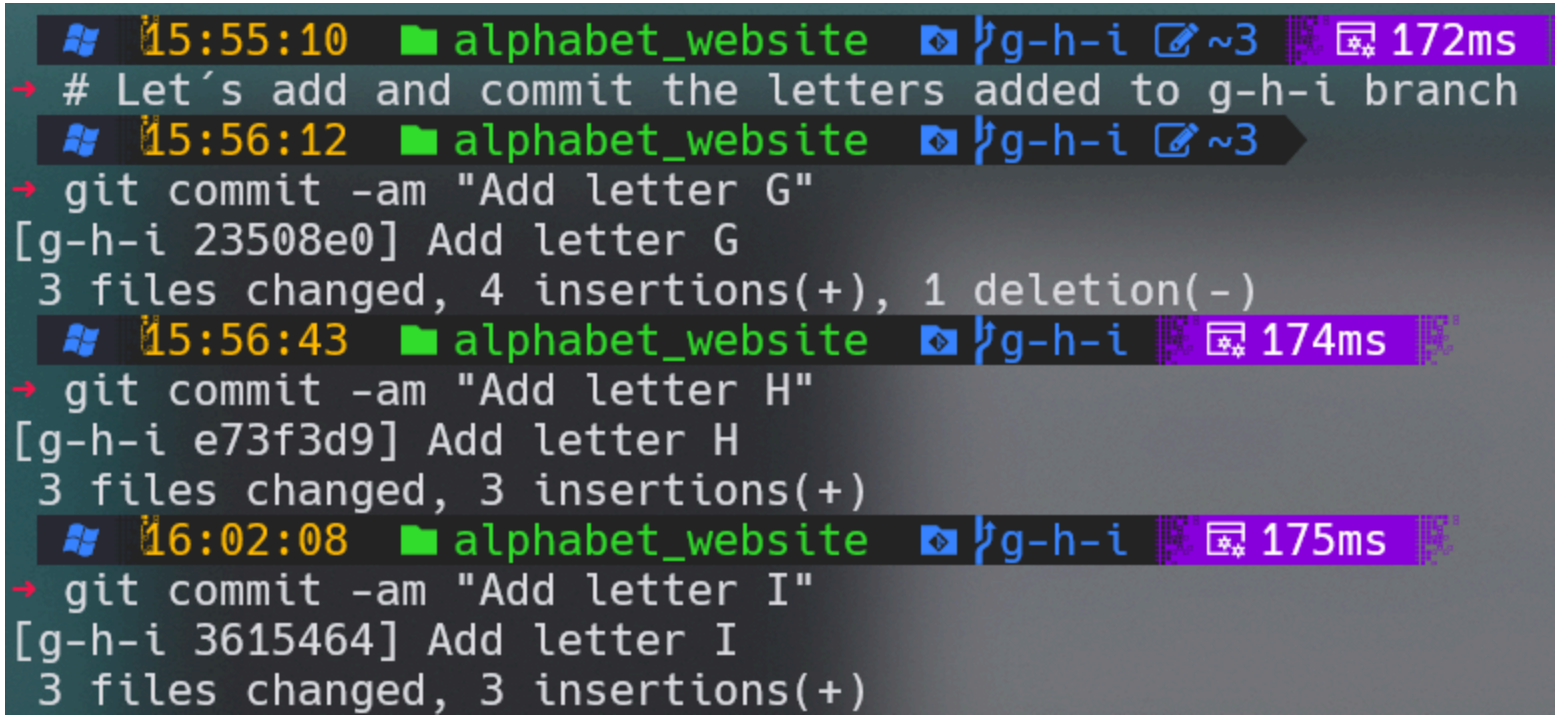
Adding letters G,H,I to the alphabet_website

Add the letter G to pilots.html, cities.html and names.html (e.g. "Golf","Greenville" and "Gloria"). Browse the website to test the changes, stage and commit with the comment "Added letter G".

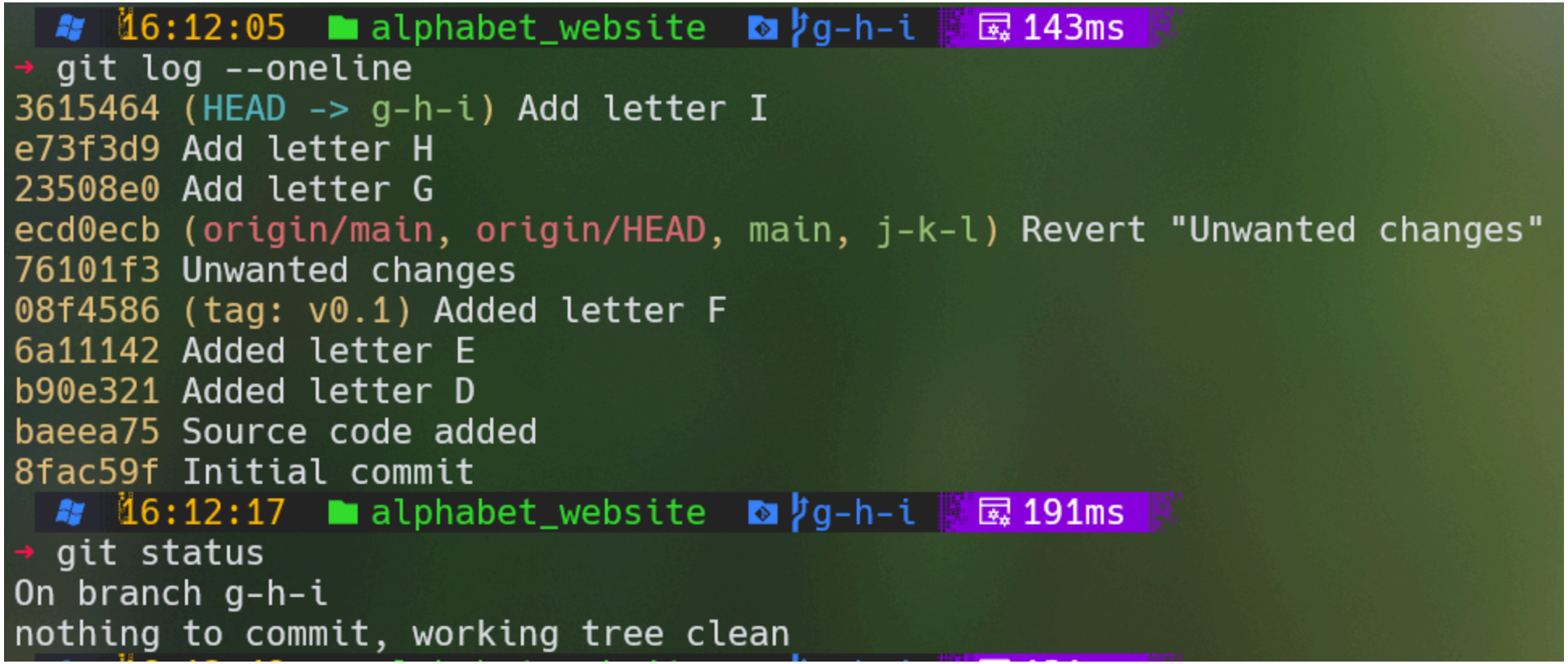
Repeat the same process to add the letter H (e.g. "Hotel", "Houston" and "Henry") then stage and commit.

Repeat the same process to add the letter I (e.g. "India","Illinois" and "Isabel") then stage and commit.

Adding letters G-H-I



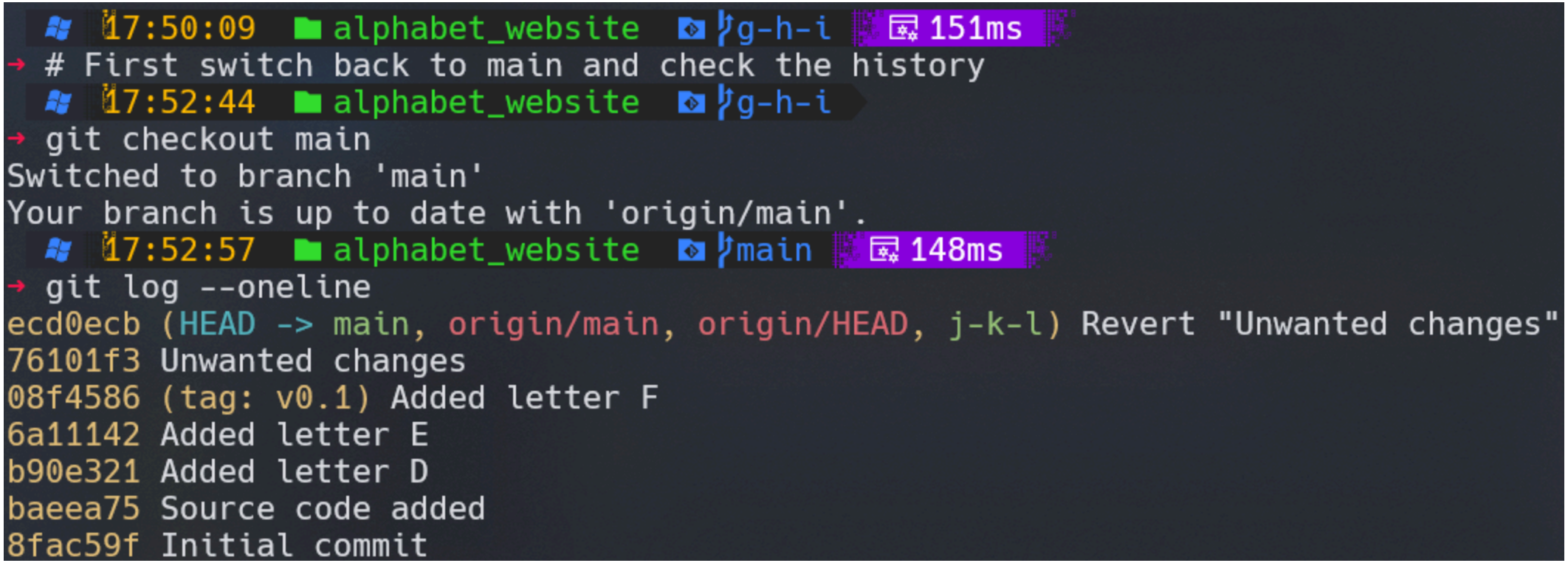
g-h-i branch history look like:



Note that the working directory is clean.

We are ready to merge the changes onto the main branch.

Switch back to main and check the history



Let´s browse the site

Now browse the Phonetic Website and you will find that it is still on the letter F. This will change once we have merged the changes from the g-h-i branch. The git merge command lets you integrate separate timelines of development into a single branch. To incorporate the feature you have implemented in the g-h-i branch into master we just need to run the following command:

```
gti merge g-h-i
```

Merging

```
  18:03:01  alphabet_website  main  213ms
→ # Let´s merge g-h-i
  18:04:23  alphabet_website  main
→ git merge g-h-i
Updating ecd0ecb..3615464
Fast-forward
 cities.html | 3 +++
 names.html  | 3 +++
 pilots.html | 5 ++++-
 3 files changed, 10 insertions(+), 1 deletion(-)
  18:04:38  alphabet_website  main  207ms
→ git log --oneline
3615464 (HEAD -> main, g-h-i) Add letter I
e73f3d9 Add letter H
23508e0 Add letter G
ecd0ecb (origin/main, origin/HEAD, j-k-l) Revert "Unwanted changes"
76101f3 Unwanted changes
08f4586 (tag: v0.1) Added letter F
6a11142 Added letter E
b90e321 Added letter D
baeea75 Source code added
8fac59f Initial commit
```

Resolving conflicts

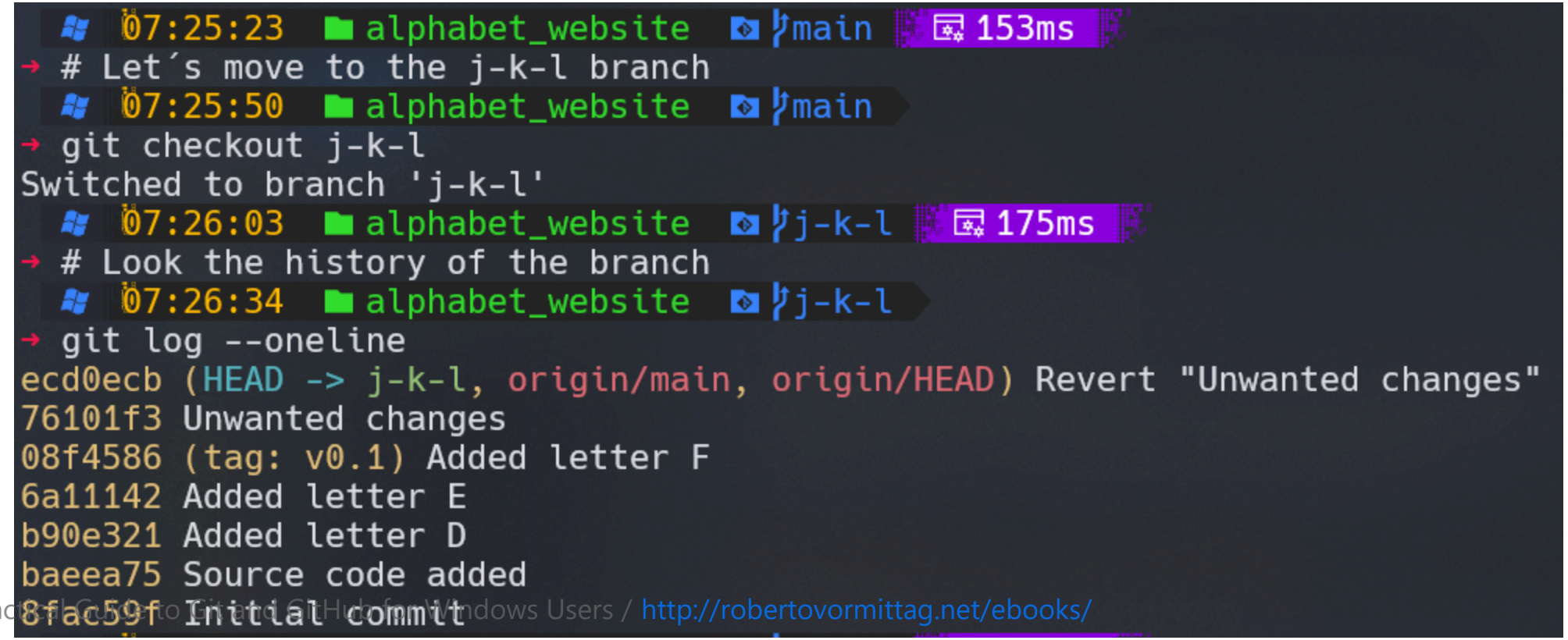
Git does a good job of merging changes in different parts of the same file automatically.

Sometimes a conflict may arise during the merging operation. That happens when the changes you made collide with the existing code in the branch you are merging into.

Adding letters J,K,L to the website

Now we're going to implement the second feature request adding letters J,K,L to the website.

Using the branch j-k-l already created.

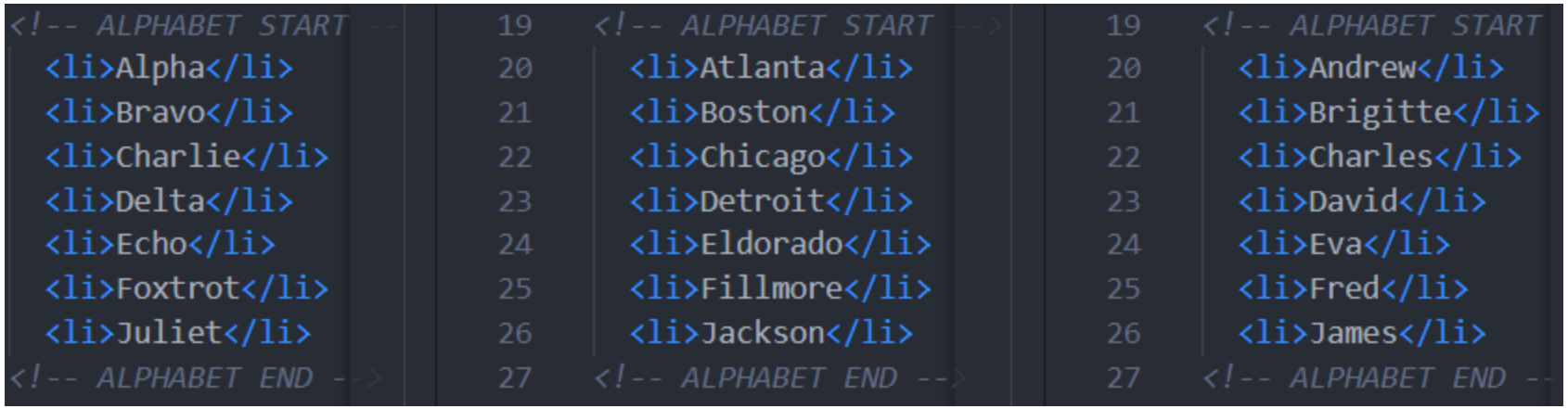


Exercise

Add the letter J to pilots.html, cities.html and names.html ("Juliet", "Jackson" and "James") immediately after the letter F.

There is a gap caused by the missing G, H and I letters. Don´t worry. We will sort out when merging the changes on to the main branch.

Letter J added to the website

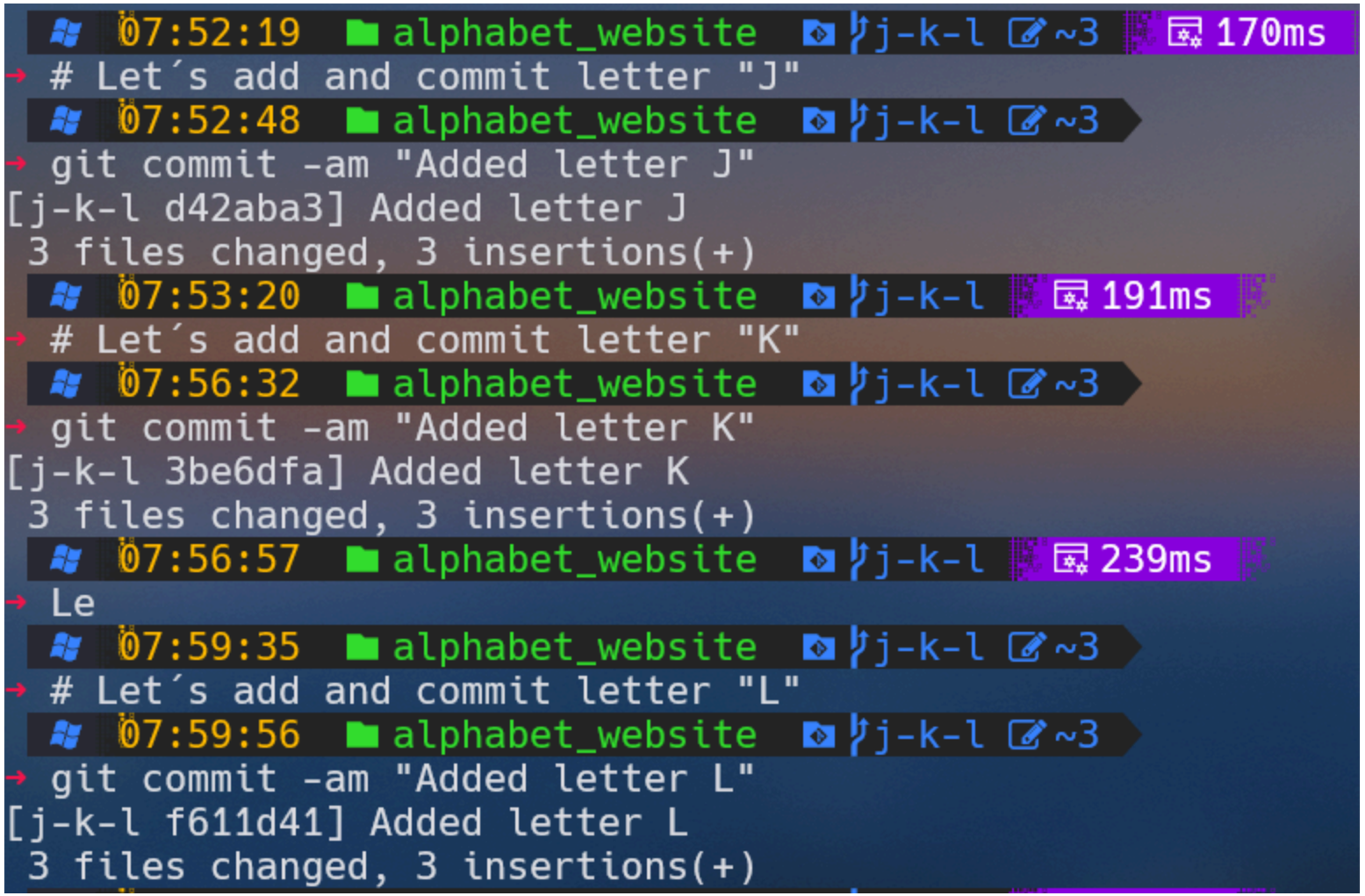


```
<!-- ALPHABET START -->
  <li>Alpha</li>
  <li>Bravo</li>
  <li>Charlie</li>
  <li>Delta</li>
  <li>Echo</li>
  <li>Foxtrot</li>
  <li>Juliet</li>
<!-- ALPHABET END -->
```

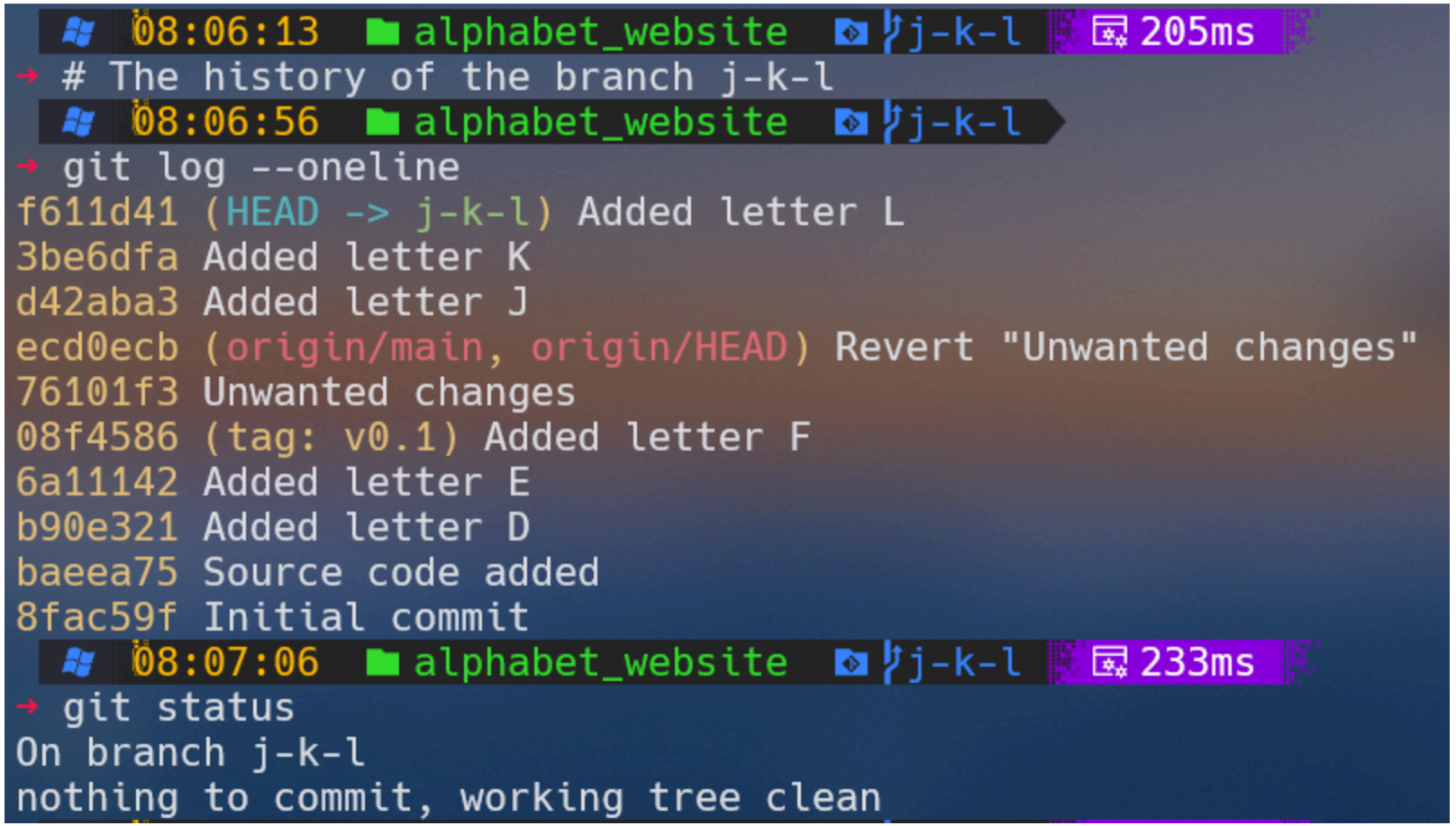
```
19  <!-- ALPHABET START -->
20    <li>Atlanta</li>
21    <li>Boston</li>
22    <li>Chicago</li>
23    <li>Detroit</li>
24    <li>Eldorado</li>
25    <li>Fillmore</li>
26    <li>Jackson</li>
27  <!-- ALPHABET END -->
```

```
19  <!-- ALPHABET START -->
20    <li>Andrew</li>
21    <li>Brigitte</li>
22    <li>Charles</li>
23    <li>David</li>
24    <li>Eva</li>
25    <li>Fred</li>
26    <li>James</li>
27  <!-- ALPHABET END -->
```

Stage and commit the letters J,K,L

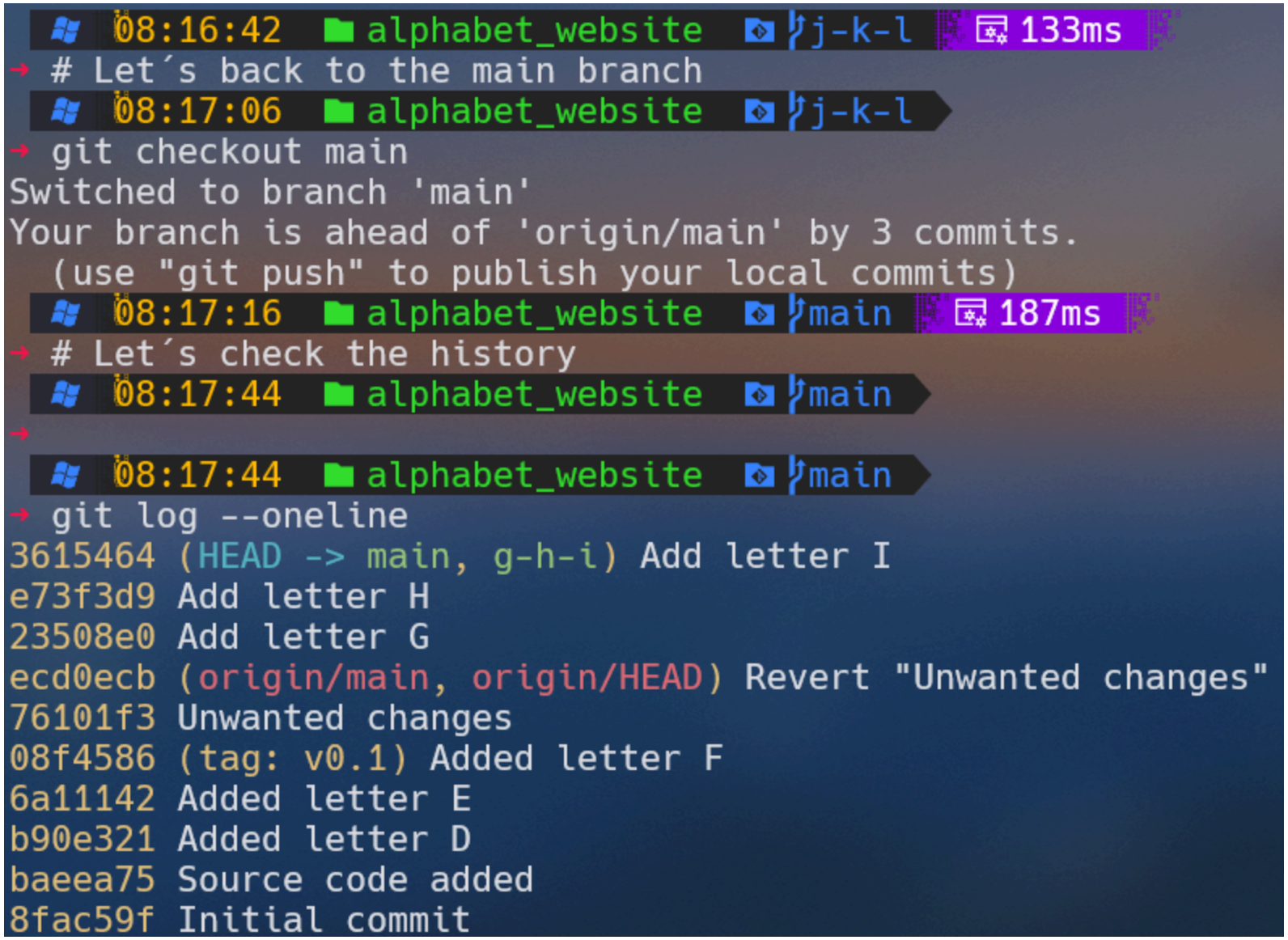


The history of the branch j-k-l

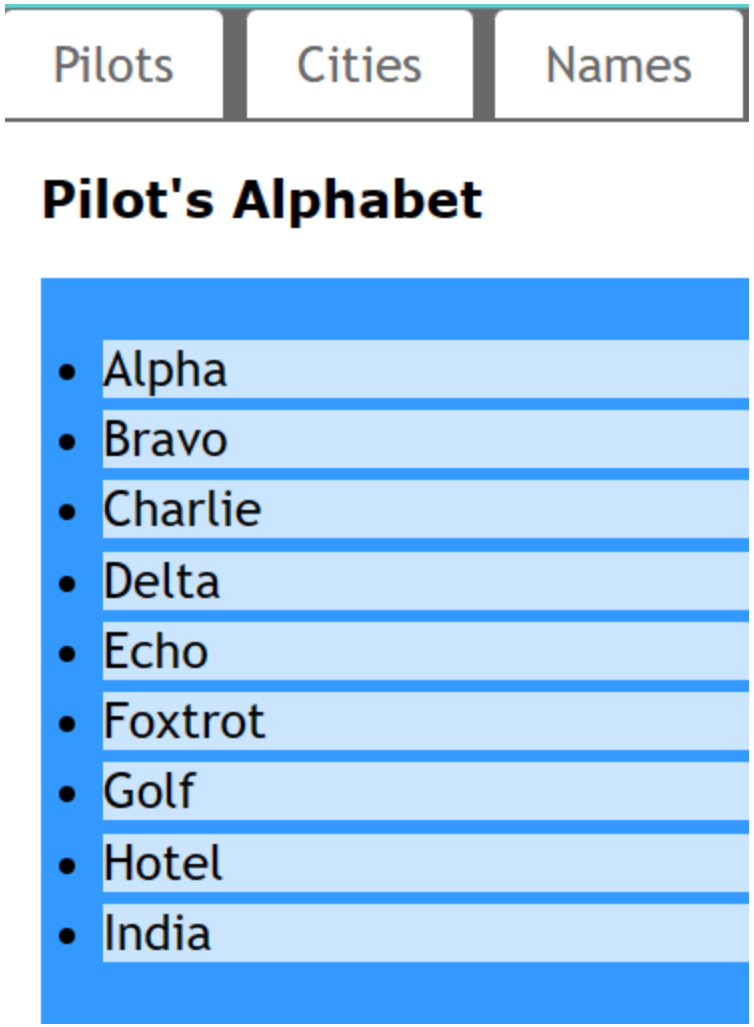


Preparing to merge

We are ready now to merge the changes on to the master branch.First switch back to master and check the history:


08:16:42 alphabet_website j-k-l 133ms
Let´s back to the main branch
08:17:06 alphabet_website j-k-l
git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 3 commits.
(use "git push" to publish your local commits)
08:17:16 alphabet_website main 187ms
Let´s check the history
08:17:44 alphabet_website main
08:17:44 alphabet_website main
git log --oneline
3615464 (HEAD -> main, g-h-i) Add letter I
e73f3d9 Add letter H
23508e0 Add letter G
ecd0ecb (origin/main, origin/HEAD) Revert "Unwanted changes"
76101f3 Unwanted changes
08f4586 (tag: v0.1) Added letter F
6a11142 Added letter E
b90e321 Added letter D
baeea75 Source code added
8fac59f Initial commit

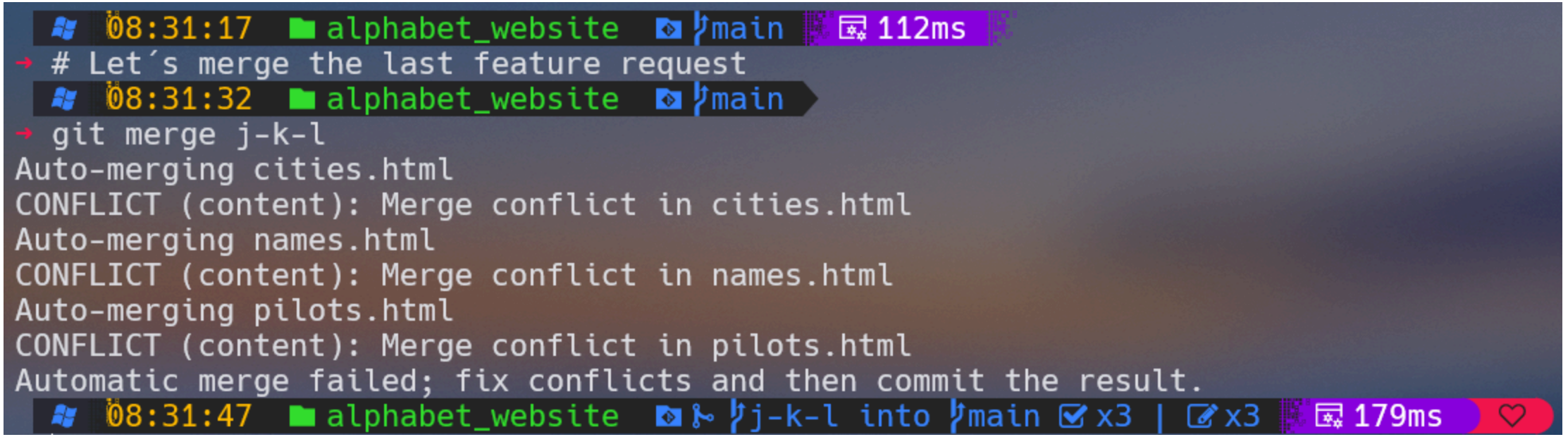
Browsing the website we can find that it is still on letter "I".



and so cities.html and names.html.

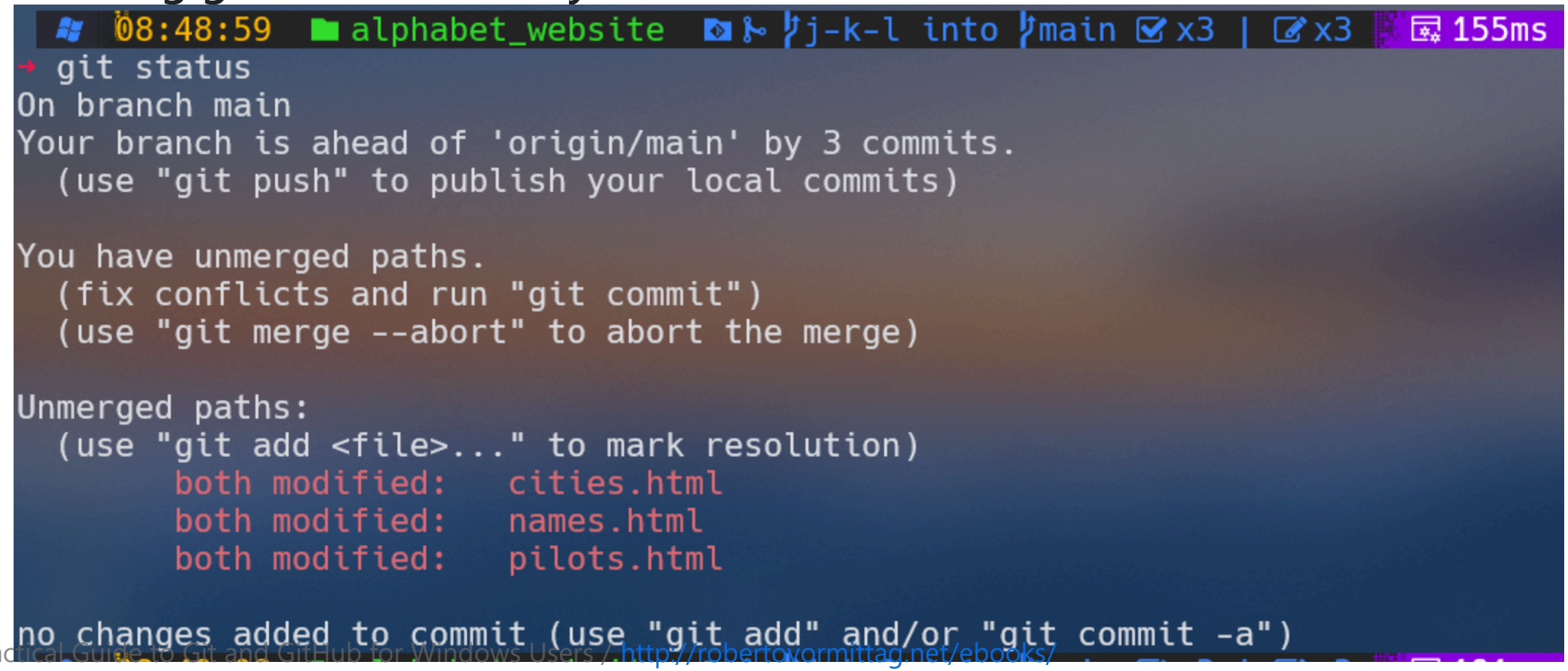
Merge the last feature request

Let's merge the feature we implemented on the j-k-l branch and see what happens:



Understanding the conflit

The git merge output is flagging conflicts on the three HTML files. That was to be expected as we made changes in the same lines of code on both branches. Running git status will tell you that the files have been modified:



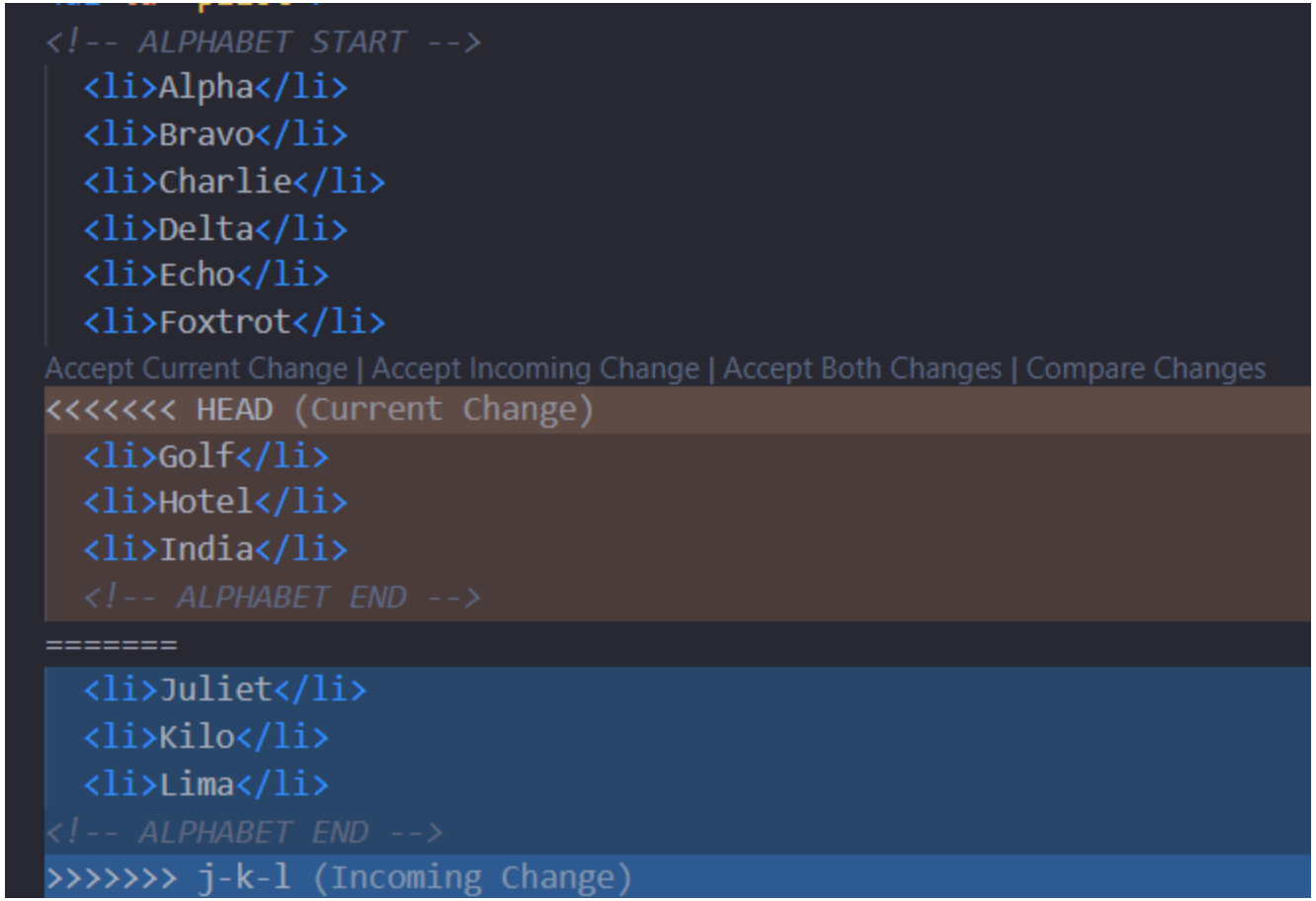
What is the way to resolve the conflicts

In case of conflicts during a merge, Git cleverly highlights the conflicts in the source code in each file keeping both changes (the one in the feature branch and the one in the target branch) leaving it to the developer to resolve the conflicts manually.

Fixing the three 'html' files

Open the files in your favourit text editor. The conflicting changes will be clearly marked by Git.

Let's fix pilots.html first. Open it in your favourite text editor. The conflicting changes will be clearly marked by Git as shown in the next slide.


<!-- ALPHABET START -->
Alpha
Bravo
Charlie
Delta
Echo
Foxtrot
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<<< HEAD (Current Change)
Golf
Hotel
India
<!-- ALPHABET END -->
=======
Juliet
Kilo
Lima
<!-- ALPHABET END -->
>>>>>>> j-k-l (Incoming Change)

Fixing (pilots.html)

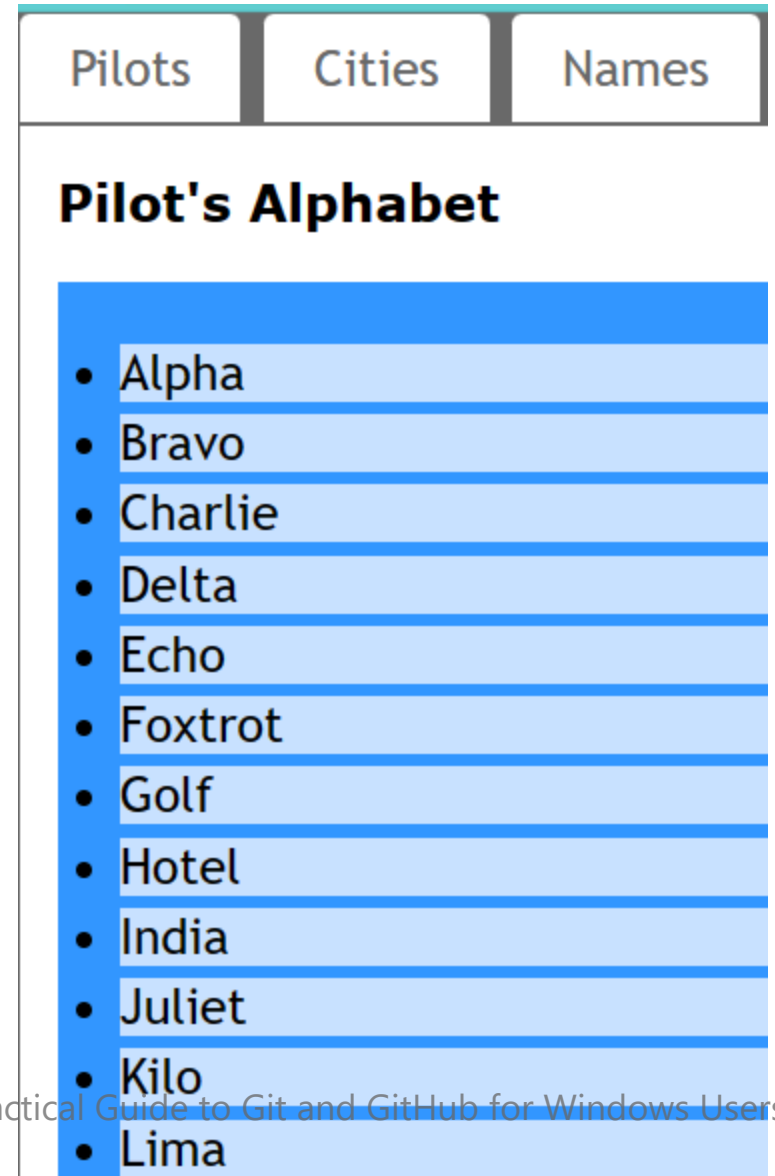
The section of code between <<<<<<< HEAD and ======= shows the code in the current branch. The section of code between ======= and >>>>>>> j-k-l shows the conflicting changes merged from the j-k-l branch. Resolving the conflict in this case is quite easy. As we want to keep both changes we just need to delete the Git markings and retain the sequence of words in alphabetical order.

After fixing

```
<!-- ALPHABET START -->
  <li>Alpha</li>
  <li>Bravo</li>
  <li>Charlie</li>
  <li>Delta</li>
  <li>Echo</li>
  <li>Foxtrot</li>
  <li>Golf</li>
  <li>Hotel</li>
  <li>India</li>
  <li>Juliet</li>
  <li>Kilo</li>
  <li>Lima</li>
<!-- ALPHABET END -->
```

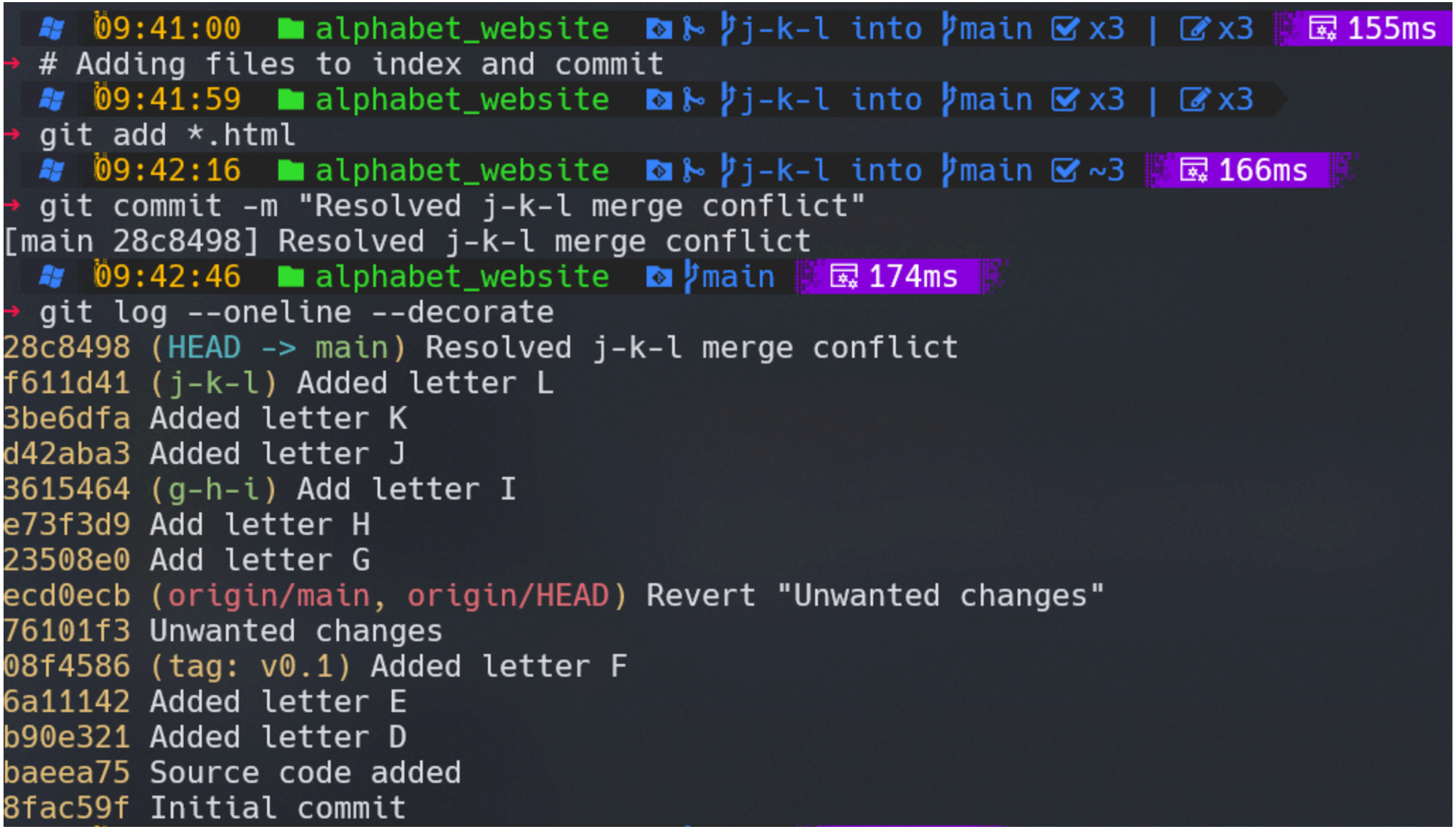
Repeat the process for cities.html and names.html.

Pilots site after fixing



Fixing the others files

Delete also the Git markings in cities.html and names.html to resolve the conflicts and save. Browse the website to test. When all looks good, add the files to the index and commit as usual.


09:41:00 alphabet_website j-k-l into main x3 | x3 155ms
Adding files to index and commit
09:41:59 alphabet_website j-k-l into main x3 | x3
git add *.html
09:42:16 alphabet_website j-k-l into main ~3 166ms
git commit -m "Resolved j-k-l merge conflict"
[main 28c8498] Resolved j-k-l merge conflict
09:42:46 alphabet_website main 174ms
git log --oneline --decorate
28c8498 (HEAD -> main) Resolved j-k-l merge conflict
f611d41 (j-k-l) Added letter L
3be6dfa Added letter K
d42aba3 Added letter J
3615464 (g-h-i) Add letter I
e73f3d9 Add letter H
23508e0 Add letter G
ecd0ecb (origin/main, origin/HEAD) Revert "Unwanted changes"
76101f3 Unwanted changes
08f4586 (tag: v0.1) Added letter F
6a11142 Added letter E
b90e321 Added letter D
baeea75 Source code added
8fac59f Initial commit

Final step

All the commits from the j-k-l branch have been integrated. The last commit is the one you have just executed following the merge conflict resolution. The git merge command is designed to preserve history whenever possible.

Tagging version

To conclude let's tag this version as v0.2 and update the remote repository on GitHub.