Identifying Vendored Files Using Custom Completion Conditions

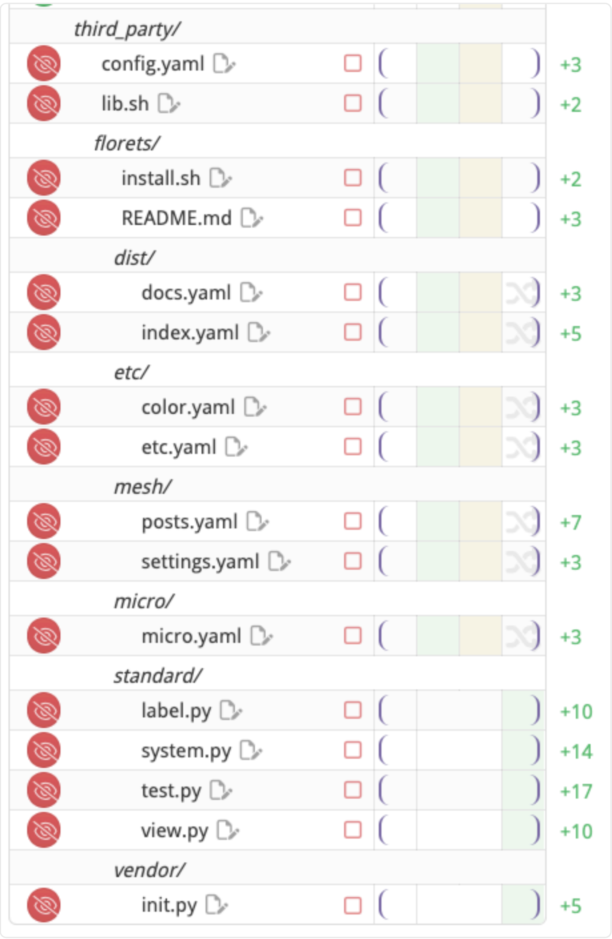
By: Alvin Charity

Published: Friday, July 5, 2024

As coding projects grow in complexity your team may need to commit your project dependencies alongside your code. Sometimes these dependencies are stored in a specific folder like `node_modules` or `vendor`. To streamline your code reviews, Reviewable automatically identifies these folders and automatically marks them as vendored. This saves you and your team time and energy, allowing you to move on to reviewing more important parts of the commit.

However, if your team is using a custom folder for these dependencies or resources, identifying these files might be tricky and cost your team a lot of valuable time. The good news is that Reviewable allows you to specify where these vendored files live in your project by identifying these folders by name using a custom completion condition that changes the state of the `review` object related to your pull request.

For example, the screenshot below shows that we have a folder called `third_party` in our project that contains dependencies and other scripts necessary to develop our application. In the Reviewable file matrix, these files will appear alongside any other files requiring review.



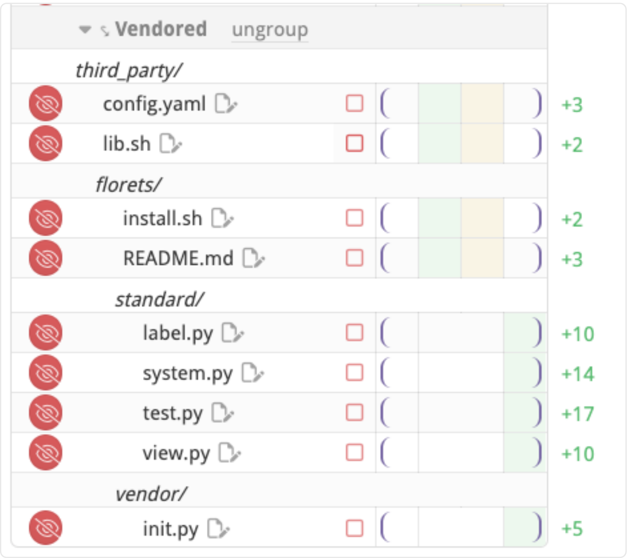
third_party/	
config.yaml	+3
lib.sh	+2
florets/	
install.sh	+2
README.md	+3
dist/	
docs.yaml	+3
index.yaml	+5
etc/	
color.yaml	+3
etc.yaml	+3
mesh/	
posts.yaml	+7
settings.yaml	+3
micro/	
micro.yaml	+3
standard/	
label.py	+10
system.py	+14
test.py	+17
view.py	+10
vendor/	
init.py	+5

Finding Vendored Files

To use a custom completion condition to find these files we have to identify the folder we would like to mark as reviewed. To do this, we will add a regular expression to match the folder name itself so that it will update their status in the file matrix. The code snippet below will treat any file in the `third_party` directory as vendored.

```
_.forEach(review.files, (file) => {

  if (/third_party/i.test(file.path)) {
    file.vendored = true;
  }

});

return {
  files: review.files,
};
```

After adding this snippet to the repository settings in Reviewable and applying the changes, we can head back to our file matrix to see that these files in the file matrix are now grouped in a category titled "Vendored".



Vendored ungroup

third_party/

config.yaml +3

lib.sh +2

florets/

install.sh +2

README.md +3

standard/

label.py +10

system.py +14

test.py +17

view.py +10

vendor/

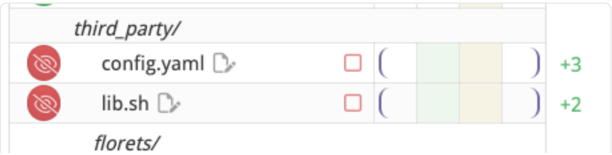
init.py +5

Automatically Review Vendored Files

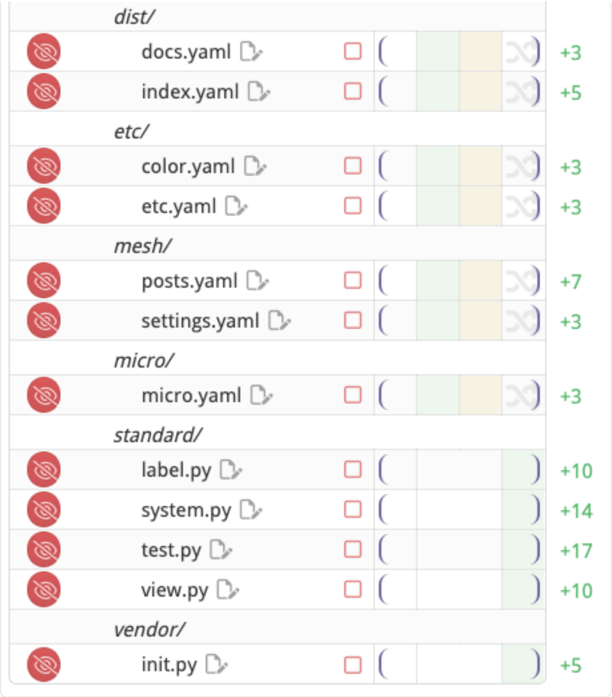
If you would like to skip the review process for these files and tell Reviewable to automatically mark them as "reviewed", you can add a line of code that will update the review status for each file to `true`:

```
_.forEach(review.files, (file) => {

  if (/third_party/i.test(file.path)) {
    file.vendored = true;

    // The line below will mark each vendored file in the
`third_party` directory as "reviewed"
    _.forEach(file.revisions, (rev) => { rev.reviewed = true; });
  }

});

return {
  files: review.files,
};
```

Now all of our files in the `third_party` directory have been identified as vendored, grouped into the "Vendored" category, and marked as reviewed.



```
third_party/
  config.yaml   □ ( )  +3
  lib.sh        □ ( )  +2
  florets/
```



```
    dist/
      docs.yaml      □ ( )  +3
      index.yaml     □ ( )  +5
    etc/
      color.yaml     □ ( )  +3
      etc.yaml       □ ( )  +3
    mesh/
      posts.yaml     □ ( )  +7
      settings.yaml  □ ( )  +3
    micro/
      micro.yaml     □ ( )  +3
    standard/
      label.py       □ ( )  +10
      system.py      □ ( )  +14
      test.py        □ ( )  +17
      view.py        □ ( )  +10
    vendor/
      init.py        □ ( )  +5
```

This example shows how easy it is to use a custom completion condition to automatically identify vendored files and mark them as reviewed, allowing your team to focus on more critical work. Reviewable's completion conditions provide a way for your team to tailor your code reviews to how you work best, and provides fine-grained control of your review process, with many example conditions to help you get started.