Process for editing UI copy

This process involves working directly in a code repository to locate copy and make edits. This approach is direct and efficient. Content can be updated without involving creating copy specs, passing specs across tickets and among multiple people. It's timely because the copy changes can be made more quickly versus waiting for capacity later on. Errors can be avoided and consistency can be provided as the UX content team can make more changes, more quickly, and more broadly.

To work this way, you gain access to front-end code repositories in either GitHub or GitLab, or both, and work just as developers do:

1. Use a GitHub client to access code files that you need to work on.
2. Create a branch where you do your work locally, off to the side.
3. Commit (finalize) changes to your branch.
4. Create a pull request for your branch that is reviewed and approved, and added to the main repository for deployment.

You will have to work with your development team to determine which files are okay to change in order to make copy changes in the UI -- you may be modifying html templates, strings.js files, or other items. Over time you should have a good sense which files are involved in affecting the front-end UI, email templates, and other content.

GitHub or GitLab setup

Cb Defense has code repositories on both github.com (external to Carbon Black) and on GitLab (an on-premises installation of the product named GitLab). Front-end UI for Defense is probably on GitHub. One example of code on GitLab is a set of email templates.

To work with GitHub, you need to create an account on github.com and request access to the Carbon Black repository.

To work with GitLab, you'll use your existing Carbon Black internal username and password, but you'll need to request access to gitlab.bit9.local.

With both repositories, you can install a GitHub client on your local system. You can use one client and switch between working with repositories on GitHub or GitLab.

GitHub Account

1. Create an account on github.com or use an existing account if you have one.

2. Ensure your account is set up with two-factor authentication. On github.com, log in and visit Settings - Security for the two-factor setup steps. You'll need to choose a two-factor method, such as using an app like Google Authenticator on your phone.
3. Use Zendesk to create an IT ticket requesting access to Carbon Black on github. Tell them your user name and ask for the /defense-ui repository.

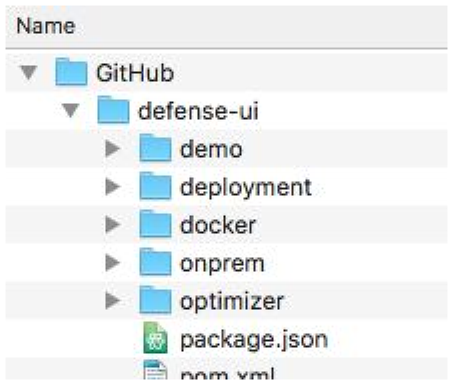
GitLab Access

Use Zendesk to create an IT ticket requesting access to gitlab.bit9.local.

GitHub Desktop setup

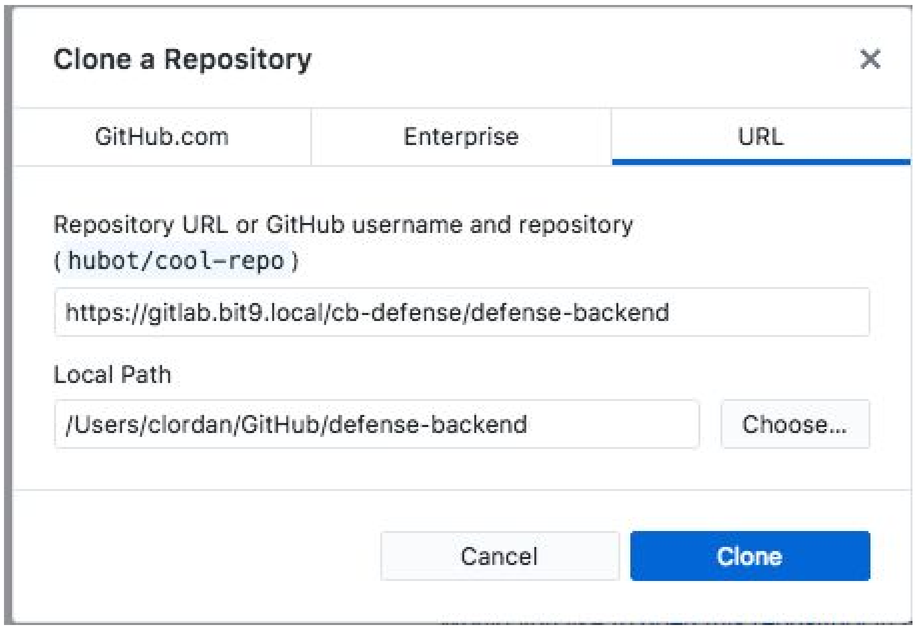
GitHub Desktop allows you to create local clones of repositories so you can create working branches, and do all your copy work locally before committing changes.

1. Install GitHub Desktop (https://desktop.github.com/) and use your github.com account to sign in to the client.
2. To clone the /defense-ui repository in your client, visit https://github.com/carbonblack/defense-ui and click the green Clone or Download button. Choose Open in Desktop and your GitHub Desktop client should open automatically with the options to create a local clone. Unless you change the defaults, all the all the defense-ui files are stored on your machine in a GitHub folder, for example:



```
Name
▼ GitHub
    ▼ defense-ui
        ▶ demo
        ▶ deployment
        ▶ docker
        ▶ onprem
        ▶ optimizer
          package.json
          pom.xml
```

3. To clone a GitLab repository, open GitLab in your browser and navigate to the repository you need, for example: https://gitlab.bit9.local/cb-defense/defense-backend
4. Copy the URL path from your browser.
5. In GitHub Desktop, choose File - Clone Repository, and paste the URL path onto the URL tab:


Clone a Repository

Repository URL or GitHub username and repository
(hubot/cool-repo)

https://gitlab.bit9.local/cb-defense/defense-backend

Local Path

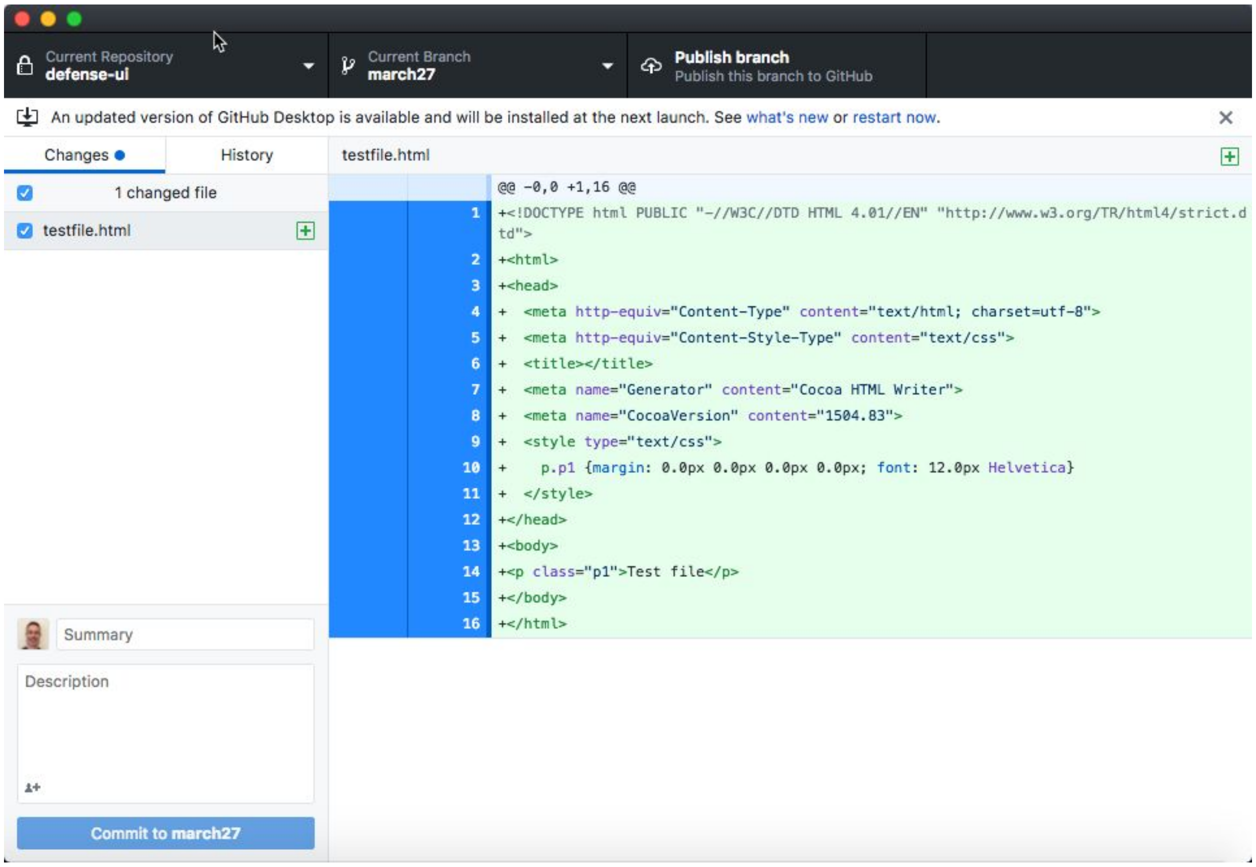
/Users/clordan/GitHub/defense-backend

Using GitHub Desktop

GitHub Desktop lets you work on making changes with a code repository.

The high-level way to work is this:

1. Set your current repository.
2. Click Fetch Origin to make sure you have all the latest changes on your machine. This pulls down all the files from the remote repository so you have the latest changes.
3. Create your own branch so you can do your work. You can name the branch whatever is meaningful, perhaps with a date, Jira ticket number, whatever makes sense to identify the work.
4. Use a text editor (explained in the next section) to update files. As you make and save changes in your text editor, you can use the Changes tab in GitHub Desktop to a running record of the changes you're making in your branch, for example:



An updated version of GitHub Desktop is available and will be installed at the next launch. See what's new or restart now.

```
@@ -0,0 +1,16 @@
+<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.d
td">
+<html>
+<head>
+  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
+  <meta http-equiv="Content-Style-Type" content="text/css">
+  <title></title>
+  <meta name="Generator" content="Cocoa HTML Writer">
+  <meta name="CocoaVersion" content="1504.83">
+  <style type="text/css">
+    p.p1 {margin: 0.0px 0.0px 0.0px 0.0px; font: 12.0px Helvetica}
+  </style>
+</head>
+<body>
+<p class="p1">Test file</p>
+</body>
+</html>
```

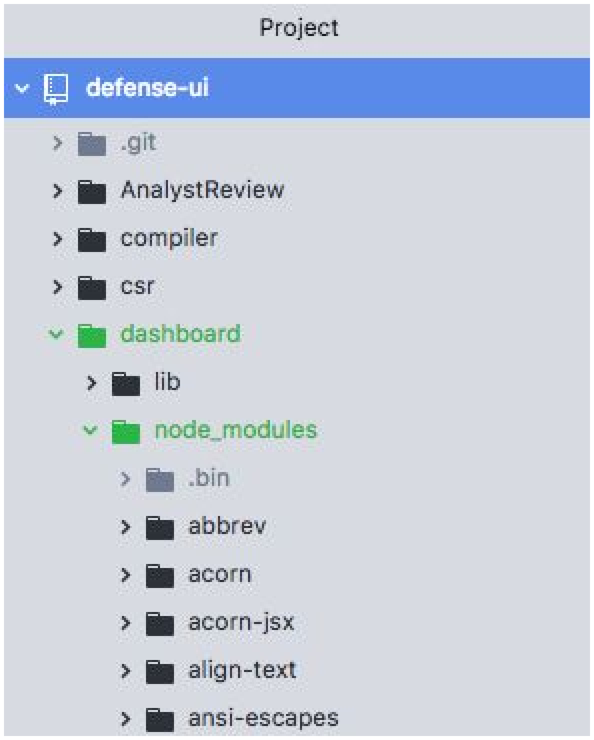
5. Once you're finished with a set of work, you commit the changes. You're required to enter a Summary and a Description (in the respective fields) and then click the Commit to [branch] button.
6. After you commit to the local branch, click Publish Branch to send the changes up to the main repository.
7. Choose Branch - Pull Request to create a pull request within the main repository. This is a request that needs to be reviewed and approved, with the approver typically being the person who also merges your code into the repository. Your code does not affect the main repository until the pull request is approved and merged -- which is good because you can't damage anything.

   Typically your pull request should be in coordination with a known body of work, such as a JIRA ticket, so that your team is expecting a set of work and an approver is ready and willing to merge your pull request.

Text editor (Atom)

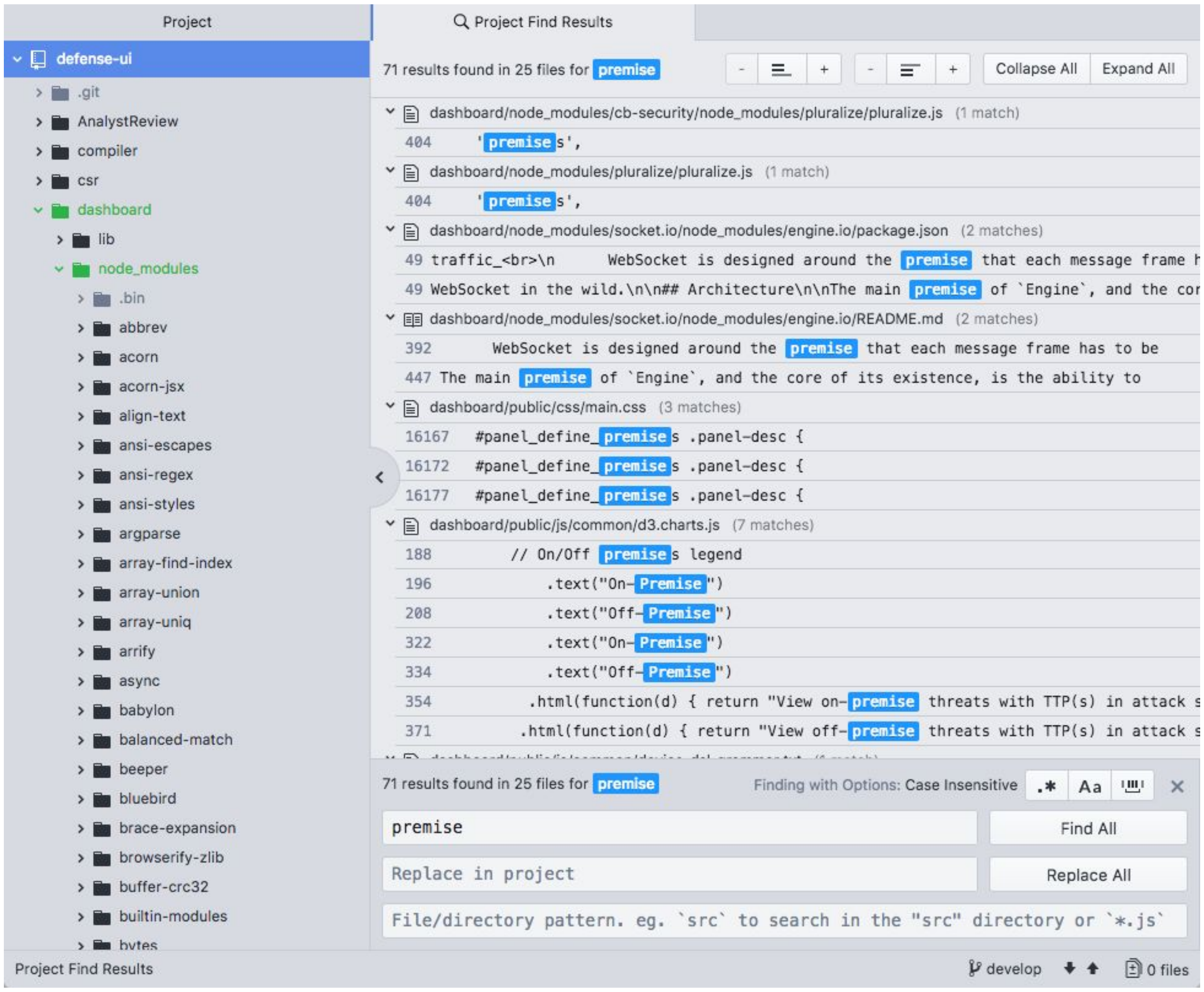
You can use any editor to make copy/string changes. Atom is a good choice for a text editor because it works well with formatting code and it has a useful feature to let you search within an entire folder, or project.

1. Install Atom (https://atom.io/)
2. Within Atom, choose File - Add Project Folder to add the local folder of the repository. For example, if you add the GitHub\defense-ui folder, it appears like this:



Locating strings

With the repository added, you can use the Find - Find in Project features to locate strings in the files, for example, if you wanted to find and change the word “premise,” Atom displays all the instances in all the files in the project:



```
defense-ui
  .git
  AnalystReview
  compiler
  csr
  dashboard
    lib
    node_modules
      .bin
      abbrev
      acorn
      acorn-jsx
      align-text
      ansi-escapes
      ansi-regex
      ansi-styles
      argparse
      array-find-index
      array-union
      array-uniq
      arrify
      async
      babylon
      balanced-match
      beeper
      bluebird
      brace-expansion
      browserify-zlib
      buffer-crc32
      builtin-modules
      bytes
```

```
71 results found in 25 files for premise

dashboard/node_modules/cb-security/node_modules/pluralize/pluralize.js  (1 match)
  404      'premises',
dashboard/node_modules/pluralize/pluralize.js  (1 match)
  404      'premises',
dashboard/node_modules/socket.io/node_modules/engine.io/package.json  (2 matches)
  49 traffic_<br>\n      WebSocket is designed around the premise that each message frame h
  49 WebSocket in the wild.\n\n## Architecture\n\nThe main premise of `Engine`, and the cor
dashboard/node_modules/socket.io/node_modules/engine.io/README.md  (2 matches)
  392      WebSocket is designed around the premise that each message frame has to be
  447 The main premise of `Engine`, and the core of its existence, is the ability to
dashboard/public/css/main.css  (3 matches)
  16167   #panel_define_premises .panel-desc {
  16172   #panel_define_premises .panel-desc {
  16177   #panel_define_premises .panel-desc {
dashboard/public/js/common/d3.charts.js  (7 matches)
  188          // On/Off premises legend
  196              .text("On-Premise")
  208              .text("Off-Premise")
  322              .text("On-Premise")
  334              .text("Off-Premise")
  354            .html(function(d) { return "View on-premise threats with TTP(s) in attack s
  371           .html(function(d) { return "View off-premise threats with TTP(s) in attack s

71 results found in 25 files for premise     Finding with Options: Case Insensitive   .*  Aa
premise
Replace in project
File/directory pattern. eg. `src` to search in the "src" directory or `*.js`

develop    0 files
```

You can simply make the edits, collect the changes in your local branch, and then create a pull request when you are finished to get the changes included.