

REVERB

CREATED BY SINGLESTONE



GITHUB FOR ENGINEERS

MAY 21, 2018

1. TODAY'S PROJECT

OVERVIEW

Today you will work in small teams to create ReverbRVA.com — a local's guide to Richmond. The guide is organized by neighborhoods:

- The Fan
- Church Hill
- Jackson Ward
- Scott's Addition
- Northside
- Shockoe Bottom

Each team will create a local's guide for one neighborhood by writing and committing code to GitHub. We'll combine all guides to produce the final ReverbRVA.com guide.



Northside



The Fan



Scott's Addition

Northside

GOOD FOOD

THE BEST ENTERTAINMENT

LEARNING OBJECTIVES

Today you will learn to use:

- GitHub's code management features including cloning, forking, branching, committing, pull requests, merging, rebasing, squashing and releases to collaboratively create your neighborhood guide.
- GitHub's publishing capabilities with GitHub pages.

TEAM ORGANIZATION




- Students will work in one of five teams seated around a table.
- Teams will have three sprints to create their neighborhood guides. At the end of each sprint teams will integrate their code and publish an updated ReverbRVA.com site.
- At the end of today, each team will present their neighborhood guide and teach us something new about Richmond!

CODE REPOSITORY

<https://github.com/ReverbTraining/Git-Intermediate-Exercise>

We will use this repository for all activities. You may want to bookmark it in your web browser.

PREREQUISITES

-  A valid username registered with <http://github.com>
-  Download and install Atom from <http://atom.io>
-  Install Git. For Windows, download from <https://git-scm.com>. For MacOS, install with command 'xcode-select --install' in Terminal. For Linux, install with your package manager.

2. CONFIGURE GIT CLI

(10 MIN)



In this exercise you will complete the first-time configuration so that you can use Git on your computer.

- 1 On Windows, open Git-Bash. On any other platform, open a terminal.
- 2 Every change committed to Git is logged. The log includes the name and email address of the person (or system) making the change. Before you can use Git to commit any changes you need to tell it who you are. Run the following commands to set your name and email address:

```
git config --global user.name "Your Name"  
git config --global user.email "<your.name@example.com>"
```

- 3 **[Optional]** If you are using Atom as your editor (recommended) you can configure Git CLI to use it for commit messages:

```
git config --global core.editor "atom --wait"
```

[Optional] If you are using Visual Studio Code as your editor you can configure Git CLI to use it for commit messages:

```
shell git config --global core.editor "code --wait"
```

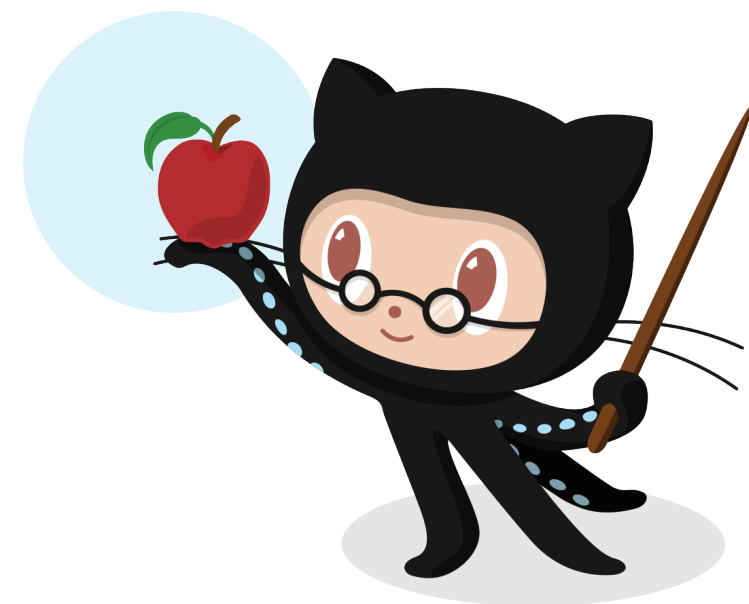
- 4 Note how a .gitconfig file was created in your home directory and updated with your user configuration:

```
cat ~/.gitconfig
```

Git will use the values in this file to get its configuration.

You can manually view and edit that file, but using the `git config` command to write configurations is safer as it prevents you from syntax errors. To view your configuration with `git config` run:

```
git config --global --list
```

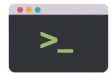


3. SET UP SSH KEYS

(10 MIN)

Git usually uses Secure Shell (SSH) or HTTPS to interact with remote repositories (such as those on GitHub). For this course, we'll use SSH.

To be able to use SSH with GitHub, you need a keypair. You keep the private key on your computer and provide a copy of the public key to GitHub.



- 1 On Windows, open Git-Bash. On any other platform, open a terminal.

- 2 See if you have an existing keypair:

```
ls ~/.ssh/
```

If you have files named `id_rsa` and `id_rsa.pub` then you can skip the next step.

- 3 Run the following command to generate a private/public keypair:

```
ssh-keygen -b 4096 -C "your.name@example.com"
```

Use the email address you have associated with your GitHub.com account. Accept all of the defaults when prompted by pressing for each prompt. When the process completes it will create `~/.ssh/id_rsa` and `~/.ssh/id_rsa.pub`.

- 4 Copy the contents of your public key (`id_rsa.pub`) into your clipboard.

Windows

```
cat ~/.ssh/id_rsa.pub | clip
# Piping the output to the clip command will place the output in
your clipboard.
```

macOS

```
cat ~/.ssh/id_rsa.pub | pbcopy
# Piping the output to the pbcopy command will place the output
directly in your clipboard. (The "pasteboard" in Mac parlance.
Think different.)
```

If all else fails

```
cat ~/.ssh/id_rsa.pub
# Then copy the output; it will start with "ssh-rsa" and end with
your email address.
```



- 5 Login to GitHub.com and click on your avatar in the top-right corner. Choose "Settings" from the menu.

- 6 On the left side, choose "SSH and GPG keys".

- 7 Click the green "New SSH key" button in the top-right corner.

- 8 Give your key a title. A good title might be the name of the computer you're using, so you can distinguish between keys from different machines.

- 9 Paste the contents of your public key (already in your clipboard) into the "Key" box and click the green "Add SSH key" button.



- 10 Test your access by running the following command from your terminal:

```
ssh -T git@github.com
```

Select "yes" if prompted to continue. If everything is set up correctly you should get the following message back:

```
Hi <GitHub username>! You've successfully authenticated, but
GitHub does not provide shell access.
```

4. CONFIGURE GIT ALIASES

(5 MIN)



You can configure Git with aliases to make it easier to run commonly-used commands.

- 1 On Windows, open Git-Bash. On any other platform, open a terminal.
- 2 Create an alias for the Git status and checkout subcommands:

```
git config --global alias.st status
git config --global alias.co checkout
```

- 3 View the contents of the global .gitconfig file to see the alias configuration:

```
cat ~/.gitconfig
```

5. SPRINT 1: GIT CLI BASICS

(50 MIN)

In Sprint 1 we'll be building the initial ReverbRVA.com neighborhood guide. Each member of your team will choose a category (e.g. Restaurants, Attractions, etc.) and based on your research will add content to a JSON file. Then you will use Git CLI commands in a basic GitHub Flow workflow to contribute your work.

Organize Your Work Before you begin building the guide, you first need to organize your work. Each team member should open a web browser to <http://ReverbRVA.com>, select their neighborhood from the home page and review the available categories (e.g. Restaurants, Attractions, etc.).

In your team, divide up the categories so each person has one category to work on. We will use GitHub Issues to track your work. Issues help document the scope of your work and communicates to others who is working on what.

In your team, each person should create a new Issue to track your work in Sprint 1. From GitHub.com, create a new Issue. The title should include your neighborhood and category.

Title: Northside Attractions Research

Description: Research the best attractions in Northside. For each identify:

- name
- neighborhood
- address
- note

Assign it to yourself and associate with a Project that is named for your neighborhood. Save your new issue and make note of the issue number (you'll need this later).

Research Your Neighborhood Using your web browser, each team member spend 10 minutes researching your selected category for your neighborhood. You may find it helpful to create a temporary text file to paste content as you go. When done, this is the initial content for your guide.

5. SPRINT 1: GIT CLI BASICS

(50 MIN)

CLONE THE REPOSITORY



- 1 First, you need to clone the repository from GitHub to your computer. On GitHub, click the green "Clone or download" button and make sure the pop-up says "Clone with SSH". (If it doesn't, click the "Use SSH" link in the top-right corner of the pop-up.) Click the clipboard icon next to the URL to copy the link to your clipboard.



- 2 On Windows, open Git-Bash. On any other platform, open a terminal.
- 3 Create a repos directory inside your home directory to store all of your Git repositories and move into it.

```
# Create a 'repos' directory inside your home directory
mkdir ~/repos
# Move into the 'repos' directory
cd ~/repos
```

- 4 Use the git clone command to clone the repository from GitHub (replacing REPO_NAME with the specific values used for this project).

```
git clone git@github.com:ReverbTraining/REPO_NAME.git
```

Git will connect to GitHub using SSH and pull down the repository contents into a new directory on your computer.

- 5 Once the clone is complete, move into the repository directory.

```
cd REPO_NAME
```

- 6 Now that you're in your cloned repository, check the repository's status with the git status command.

7

Whenever you run git clone it will automatically create a remote named origin that points to the source that you originally cloned. You can confirm this by running git remote --verbose. (Omitting the verbose flag will just return the names of your remotes and not their URLs).

CREATE A FEATURE BRANCH

8

Now that you have the repository cloned locally, you need to create a branch to isolate your changes. First, see what branches currently exist in your local repository. The asterisk (*) indicates the branch that you are currently on.

```
git branch
```

9

Next, create a branch. Name the branch after the GitHub issue number that you're working on.

```
# Use your actual issue number
git checkout -b issue-1
```

10

Verify that you are on your new branch by running git branch again.

CREATE NEIGHBORHOOD CONTENT

To publish your guide, you need to first add your content to a JavaScript file in JSON formatted arrays. Each array is an entry in your neighborhood guide, such as a specific attraction or restaurant.



11

Open the repository directory in your editor of choice.

5. SPRINT 1: GIT CLI BASICS

(50 MIN)

- 12 Navigate to your neighborhood's directory. Find the JavaScript (*.js) file for the category you've selected and open it.
- 13 Add your JSON array as the value for the data variable. It should look something like this:

```
var data = [
  {
    "name": "establishment name",
    "neighborhood": "neighborhood name",
    "address": "street address",
    "note": "information about the establishment"
  },
  {
    // Add as many objects as you need based on your
    researched data.
    // You should have at least four.
  }
];
```

- 14 Validate that your JSON array is syntatically valid by checking it with [JSONLint](#). Paste the square brackets ([]) and everything between them. If your JSON does not validate, correct any syntax errors before continuing.
- 15 Save the JavaScript file.
- 16 Now that the state of your repository has changed, run git status. Git will show you what file has changed.
- 17 Git also tracks what lines in a file changed. Run git diff to show the differences between the file's last committed state and the copy you're working on. (Press q to exit the diff view.)

- 18 To stage a commit, use the git add command along with a path argument. You can stage an individual file, or stage all of the changes in your current directory (recursively) by specifying a period.

```
git add .
```

- 19 Running git status again will show that the changed JavaScript file has been staged for commit.
- 20 Commit your change by running git commit. Running this command without any other arguments will open your text editor (as defined by the core.editor property in your Git configuration) so that you can write a commit message.

Use the below example as a guide for your commit message, inserting your issue number in place of XX. Manually wrap your body text lines at 72 characters. Note how you can use Markdown in the body of your message:

```
Add data for <NEIGHBORHOOD> establishments
```

```
Adds an array of JSON objects that contains information
about neighborhood
establishments in <NEIGHBORHOOD>. This data has been
- researched, and
- syntatically validated
```

```
Fixes #XX.
```

When you've finished entering your commit message, save the file and close your editor. Git will finish the commit.

5. SPRINT 1: GIT CLI BASICS

(50 MIN)

MAKE ANOTHER COMMIT

- 21 Open the Markdown file for your assigned category.
- 22 Edit the page's title to include the neighborhood. For example, "Restaurants" should become "Church Hill Restaurants".
- 23 The "Get a suggestion" button is created by the `<input>` tag. The button currently does nothing when clicked. Add an `onClick` action that calls the `printSuggestion()` function and passes the JSON data you just added. Your result should look like this:

```
<input type="button" name="getSuggestion" value="Get a
suggestion" onClick="printSuggestion(data)">
```

Save the file once you're done.

- 24 Open the JavaScript file for your category and add a new line above the data variable. On that line, add a comment describing the data. For example:

```
// Suggestion data for <NEIGHBORHOOD> <CATEGORY>
var data = [
  ...
];
```

Save the file once you're done.

- 25 Use `git status` and `git diff` to see what's changed since the last commit.

- 26 Stage your changes and commit them. Use the below example as a guide for crafting your commit message.

Add page title and action for suggestion button

Adds an 'onClick' action for the suggestion button that calls the 'printSuggestion()' function.

Also adds a descriptive comment to the data.

Fixes #XX.

USE THE LOG

- 27 View the commit log for your branch by running `git log`. The most recent commit will be the first one listed.
- 28 The `git log` command accepts many arguments to change the format of the log view. One useful argument is `--oneline` which shows only the summary of each commit. Try it with `git log --oneline`.
- 29 Use `git show` on each of your commits (by passing the commit ref, or at least the first few characters of it, as an argument to the command). For example, `git show 018ab7c`.

UNDERSTAND WHAT CHANGED

- 30 Run `git log --oneline` and make a note of the top two commit refs. The top commit is your second, the one below that is your first.
- 31 Run `git diff <first>..<second>` replacing the placeholders with your commit refs. The output will show what changed from the first commit to the second one.
- 32 To compare your feature branch (`issue-XX`) to the master branch run `git diff master issue-XX`. If you omit the second branch name it will show a diff between master and the branch you currently have checked out.

5. SPRINT 1: GIT CLI BASICS

(50 MIN)

REVISIT GIT CHECKOUT

In this exercise you will switch between commits and branches to see how Git changes the disk contents of the repository to reflect whichever ref is checked out. You'll need the SHA-1 hashes from your first commit ("first") from the diff two commits section above.

- 33 Ensure you have checked out your feature branch by running `git checkout issue-XX` where XX is your issue number.
- 34 Open your Markdown and JavaScript files in Atom or Visual Studio Code. If you're using another editor (like vim) open and close the files after each check out.
- 35 Check out your first commit on your branch with `git checkout <first>` replacing the placeholder with your commit ref.
- 36 Note how your changes to Markdown and JavaScript files have disappeared, since they were committed after your first commit.
- 37 Run `git log` and note how the log doesn't include your second commit.
- 38 Now `git checkout master` to check out your master branch. Note how all of your changes to both files are gone.
- 39 Return to your feature branch with `git checkout issue-XX` where XX is your issue number.

SUBMIT AND INTEGRATE YOUR WORK

- 40 To push your branch to the repository on GitHub, use the `git push` command and pass it the name of your remote (origin) and the name of the branch that you're pushing to on GitHub (issue-XX):

```
git push --set-upstream origin issue-XX
```



- 41 Open GitHub in your browser and go to the repository page. A yellow notification banner includes a button to compare your changes and open a pull request. Click it.
- 42 Write a good commit message and include a tag to close your issue.
- 43 Assign the pull request to a person on your team and add that person as a reviewer.
- 44 Submit the pull request.
- 45 Review the pull request that your teammate has assigned to you and approve their changes.
- 46 Finally, accept the pull request. GitHub will merge your feature branch (issue-XX) into the master branch in GitHub's repository.

UPDATE YOUR LOCAL REPOSITORY



- 47 Since the master branch on GitHub is what's changed (as a result of merging your pull request), your local copy of the master branch is now behind. Run `git checkout master` to check out your local master branch.
- 48 Run `git pull` to have Git fetch the changes from the origin remote (GitHub) and merge them into your local master branch.
- 49 Run `git log` to confirm that your local branch has all of the expected commits.

CLEAN UP OLD BRANCHES

- 50 Check out the master branch.
- 51 Delete your local feature branch by running `git branch -d issue-XX`.
- 52 Delete your remote feature branch by running `git push origin :issue-XX`. The colon prefixing the branch name signals a deletion.

6. SPRINT 2: FORKING

(50 MIN)

Update Styling In Sprint 1 we collectively built our initial guide. In this Sprint we will improve the style of our guide by updating CSS files. We will also introduce how to use the forking workflow to make these changes.

Organize Your Work Just like in Sprint 1, you will use an Issue to track your work. From GitHub.com, create a new Issue. The title should include your neighborhood and category.

Title: Update Style for Northside Attractions

Description: Improve the styling for Northside attractions guide.

Assign it to yourself and associate with a Project that is named for your neighborhood. Save your new issue and make note of the issue number (you'll need this later).

CREATE AND CLONE A FORK

To begin, you need to perform the one-time steps of forking a repository, cloning the fork to your computer, and adding a remote back to the original upstream repository so you can keep your fork in sync.



- 1 Begin by navigating to the repository page in GitHub.
 - 2 Click the "Fork" button in the upper-right of the page. The number next to the button is the number of times the repository has already been forked.
-
- 3 Clone the forked repository to your computer. Click the green "Clone or download" button and make sure the pop-up says "Clone with SSH". (If it doesn't, click the "Use SSH" link in the top-right corner of the pop-up.) Click the clipboard icon next to the URL to copy the link to your clipboard. Refer back to the CLI Basics exercise if you need detailed instructions.

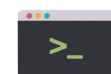


- 4 From your terminal, change into the repository directory and run `git remote --verbose` to get a list of the configured remotes. You should see only the origin remote.

```
origin@github.com:GITHUB_USER/GIT_REPO.git (fetch)
origin@github.com:GITHUB_USER/GIT_REPO.git (push)
```



- 5 Navigate back to the original GitHub repository page (the one that you forked) and click the green "Clone or download" button and make sure the pop-up says "Clone with SSH". (If it doesn't, click the "Use SSH" link in the top-right corner of the pop-up.) Click the clipboard icon next to the URL to copy the link to your clipboard.



- 6 In your terminal, add an "upstream" remote that points back to the original project that you forked.

```
git remote add upstream git@github.com:ReverbTraining/GIT_REPO.git
```

- 7 Run `git remote --verbose` again and confirm that both remotes are now configured.

```
origin@github.com:GITHUB_USER/GIT_REPO.git (fetch)
origin@github.com:GITHUB_USER/GIT_REPO.git (push)
upstream    git@github.com:ReverbTraining/GIT_REPO.git (fetch)
upstream    git@github.com:ReverbTraining/GIT_REPO.git (push)
```

CREATE A FEATURE BRANCH

- 8 Now that you have the repository cloned locally, you need to create a branch to isolate your changes. First, see what branches currently exist in your local repository. The asterisk (*) indicates the branch that you are currently on.

```
git branch
```

- 9 Next, create a branch. Name the branch after the GitHub issue number that you're working on.

```
git checkout -b issue-1
```

- 10 Verify that you are on your new branch by running `git branch` again.

6. SPRINT 2: FORKING

(50 MIN)

UPDATE A CSS FILE



- 11 Open the repository directory in your editor of choice.
- 12 In your neighborhood's directory, create a new CSS file for your category. For example, `restaurants.css`.
- 13 In your CSS file add styling to your page. You can use the following as an example, or customize it:

```
.vcard {
  font-family: serif;
  margin-left: 35%;
  border-radius: 10px;
  border-style: solid;
  border-color: silver;
  border-width: thin;
  padding: 3%;
}

.org {
  font-size: larger;
  font-variant: small-caps;
  margin-top: 5%;
  margin-bottom: 3%;
  border-bottom-style: solid;
  border-bottom-color: black;
  border-bottom-width: thin;
}

.note {
  font-style: italic;
  margin-top: 2%;
}
```

- 14 In your Markdown file, add a line after line three and load the stylesheet for your category:

```
<link rel="stylesheet" href="<category>.css">
```

- 15 Now that the state of your repository has changed, run `git status`. Git will show you what files have changed. Use `git diff` to verify the differences between the old and new versions of the files. Note that it only compares files that are already tracked by Git.
- 16 Stage and commit your changes. Remember to follow The Seven Rules of a Great Commit Message when writing your commit messages.

USE GIT LOG TO COMPARE BRANCHES



- 17 View the commit log.

```
git log

# If the log doesn't show branch names, run
git log --decorate=short
```

- 18 Note how your feature branch is at the top of the log with the most recent commits. The upstream/master, origin/master, and master branches are behind. All three of these master branches are on your computer. When you fetch from the upstream and origin remotes those branch markers will move to reflect any new commits that appeared on GitHub.

PUSH TO YOUR FORK AND OPEN A PULL REQUEST

- 19 To push your branch to your fork on GitHub, use the `git push` command and pass it the name of your remote (origin) and the name of the branch that you're pushing to on GitHub (issue-XX):

```
git push --set-upstream origin issue-XX
# 'origin' here refers to your fork; run 'git remote -v' for a reminder
```


6. SPRINT 2: FORKING

(50 MIN)



- 20 Open GitHub in your browser and go to your fork's repository page. A yellow notification banner includes a button to compare your changes and open a pull request back to the upstream repository. Click it.
- 21 GitHub will automatically use the commit message (if you only have one commit) to populate the pull request form. Assign someone on your team as a reviewer.
- 22 Submit the pull request. The pull request will be visible in the upstream repository.
- 23 Review the pull request that your neighbor has assigned to you and approve their changes.
- 24 Finally, accept your pull request. GitHub will merge your feature branch (issue-XX) into the master branch in upstream repository that you forked.

UPDATE YOUR LOCAL REPOSITORY

At this point it's important to understand the state of the three repositories you're working with.

Local repository

- Contains master and issue-XX branches
- The master branch exists only to mirror upstream's master branch
- Has a remote called origin that points to your GitHub fork
- Has a remote called upstream that points to the original GitHub repository that you forked

GitHub Fork

- Contains master and issue-XX branches
- Is a fork of the original GitHub repository
- Is private to you; no one else has access
- Exists primarily to host feature branches that are the sources for pull requests
- The master branch is not used

Original GitHub repository

- Contains a master branch that includes the commits from the issue-XX branch (these were merged via your accepted pull request)
- Is the canonical repository for the project
- Is publicly accessible (at least to a group of people)
- Maintained by a few (or one) maintainers
- You have read access but might not have write access



- 25 To retrieve the current state of the original repository, use the `git fetch` command against the upstream remote. This will update the local `upstream/*` branches you have such as `upstream/master`.

```
git fetch upstream
```

- 26 Use `git log --branches --remotes` to see where `upstream/master` is relative to the other branches (`master`, for example). The commits between them in the log indicate commits that are present on GitHub (and now in your local `upstream/master` branch) but not yet in your master branch.
- 27 You can use `git diff` to see the differences between the two branches. Simply run `git diff master upstream/master`.
- 28 Checkout your master branch and merge `upstream/master` into it. This will bring your local master branch in sync with the state of the master branch on GitHub (as of the last time you ran `git fetch`).

```
git checkout master
git merge upstream/master
```

- 29 Run `git log` again and note that both `upstream/master` and `master` are now at the same commit.

6. SPRINT 2: FORKING

(50 MIN)

You can optionally update your GitHub fork's (origin) master branch by now running `git push origin master`. In practice, your fork's master branch is never used—you simply update your local repository from the upstream repository, and push your feature branches to your fork (so you can create pull requests).

CLEAN UP OLD BRANCHES

Once your pull request has been accepted and merged into upstream's master branch and you've updated your local repository, you can safely delete your feature branch from your local repository and from your GitHub fork.

- 30 Check out the master branch.
- 31 Delete your local feature branch by running `git branch -d issue-XX`.
- 32 Delete your remote feature branch by running `git push origin :issue-XX`.

Close Issue Verify the status of your issues from the first two Sprints. If your commit message contained "fixes XX" where XX is your issue, it should be closed. If not close it manually.



7. RESOLVING MERGE CONFLICTS

(10 MIN)

Many software repositories have a **MAINTAINERS** file that lists the people responsible for maintaining the code. In this exercise you will create a **MAINTAINERS** file for your neighborhood directory and put your name in it.

CREATE AN ISSUE

Create a GitHub issue that describes the need to have you listed in the **MAINTAINERS** file for your neighborhood and assign it to yourself. Tag yourself in the issue description by prefixing your GitHub username with `@`. Note the issue number, as you will use it to name your feature branch.

UPDATE YOUR LOCAL REPOSITORY



- 1 Run `git checkout master` to check out your local master branch.
- 2 Fetch from your remotes and merge upstream changes into your master branch.
- 3 Run `git log` to confirm that your local branch has all of the expected commits.

CREATE A MAINTAINERS FILE

- 4 Create a branch named `<your initials>-maintainers`.
- 5 On your new branch a file named **MAINTAINERS** in your neighborhood's directory.

```
touch <neighborhood>/MAINTAINERS
```

7. RESOLVING MERGE CONFLICTS

(10 MIN)



6 Open the file in your editor and add your name on the first line.

7 Save, stage, and commit your change.



8 Push your branch to GitHub and open a Pull Request. Assign someone on your team as the reviewer.

RESOLVE MERGE CONFLICT



9 Pull the latest master branch code from GitHub into your local feature branch so you can view the conflict and resolve it.

```
git checkout issue-XX
git pull upstream master
```

10 Use `git status` to confirm which file has conflicts. In this exercise only the **MAINTAINERS** file is shown, but in practice you may need to resolve conflicts across several files.

11 Open the **MAINTAINERS** file in your editor. Note the conflict markers that Git has inserted into the file. The lines above `=====` represents the changes introduced by your feature branch. The lines below represent the current state of the master branch.

12 Edit the file so that there is one name per line. Be sure to remove all of the conflict markers. Save the file.

13 Use `git add` to indicate that the conflict has been resolved.

14 Run `git commit` to commit the **MAINTAINERS** file with the resolved conflict.

15 Review the git log to visualize the branch history.

16 Now `git push` your feature branch to GitHub. This will add your new commit to the Pull Request.



17 Use GitHub's `@username` notation in the PR's comments to inform your reviewer that your PR is ready for re-evaluation.

8. SPRINT 3: REBASE AND SQUASH

(50 MIN)

In Sprint 3 we will add a new feature to the ReverbRVA.com guide: Sponsored Content. We will do this by adding a new attribute "sponsored" to the JSON arrays we created in Sprint 1.

Organize Your Work Just like in the previous Sprints, you will use an Issue to track your work. From GitHub.com, create a new Issue. The title should include your neighborhood and category.

Title: Add Sponsored Content for Northside Attractions

Description: Implement the new sponsored content feature

Assign it to yourself and associate with a Project that is named for your neighborhood. Save your new issue and make note of the issue number (you'll need this later).

ADD SPONSORED ATTRIBUTE TO DATA

- 1 Open your category JavaScript file for your neighborhood.
- 2 Select any two suggestions in your array of JSON objects and add a "sponsored": true attribute to each of them. For example:

```
[
  {
    "name": "Restaurant A",
    "address": "123 Main Street",
    "neighborhood": "Ginter Park",
    "note": "Great burgers",
    "sponsored": true
  },
  {
    "name": "Restaurant B",
    "address": "123 Franklin Street",
    "neighborhood": "Bellvue",
    "note": "Try the calamari"
  }
];
```

- 3 Validate that your JSON array is syntactically valid by checking it with JSONLint. Once validated, commit your change.

REMOVE THE EXISTING printSuggestion() FUNCTION

Your existing printSuggestion() function will be superseded by a new function shared across the entire website.



- 4 In your category JavaScript file, delete the entire printSuggestion() function. Your file should only contain

```
var document = "";
var data = [...];
```

- 5 Commit your change.

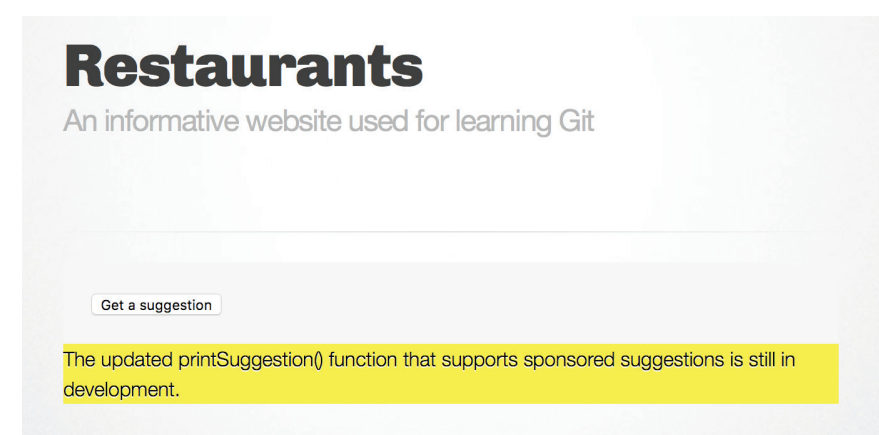
SOURCE THE SHARED printSuggestion() FUNCTION

- 6 In your Markdown file, source the new function by adding the following line at the end of the file:

```
<script type="text/javascript" src="../../suggestion.js"></script>
```

- 7 Commit your change.
- 8 Use git log to examine the branch's commit history. At this point, your feature branch should have three new commits.
- 9 Push your branch to your GitHub fork.

Because the shared function has not yet been implemented, if you were to publish the site as-is users would see the following warning if they try to get a suggestion.



8. SPRINT 3: REBASE AND SQUASH

(50 MIN)

REBASE TO INCORPORATE UPSTREAM CHANGES

- 10 Run `git fetch upstream` to retrieve changes from the upstream project repository.
- 11 Use `git log` again to examine the current commit history. You may need to run it against both your feature branch and upstream/master. Note how there is a new commit on upstream/master. Make note of that commit's SHA-1 hash.
- 12 Run `git show` to examine the new commit.
- 13 From your feature branch, run `git rebase upstream/master`. This will replay your feature branch's commits onto the latest commits from upstream's master branch.
- 14 Run `git log` again to review your branch's new history. It should include the new upstream/master commit followed by all of the commits you've made in your branch.
- 15 Push your rebased branch to your branch on your GitHub fork. Since rebasing rewrites history, you need to force-push.

```
git push --force origin <branch name>
```

SQUASH COMMITS

- 16 Run `git rebase --interactive upstream/master` to trigger an interactive rebase against upstream's master branch.
- 17 When prompted, choose to pick the first/top commit and squash all of the others. Save and close the file.

- 18 After Git performs the rebase and squash, you will be prompted to compose a commit message for your new squashed commit. Compose a good commit message that closes your issue.
- 19 Use `git log` to examine your branch history. Note how the three unique commits in your branch have been reduced to one commit that comprises all of the changes you've made.
- 20 Force-push your branch to your fork on GitHub.



- 21 Open a Pull Request and add someone from your team as a reviewer.

REVIEW AND MERGE

- 22 Review the Pull Request you have been assigned. Once you are satisfied with the content, accept and merge the Pull Request.
- 23 Check out your work on the site!





TRAINING DESIGNED TO MAKE WAVES



(833) 473-8372

reverb.singlestone.io
reverb@singlestoneconsulting.com



We are a tech consulting company that focuses on Cloud, DevOps, Application Development, Digital Engagement and Customer Experience. For 20 years, the idea that there's a smarter way to do business and a better way to serve customers has driven our mission-oriented approach to consulting. Learn more about our approach, our leadership and the way we think by visiting our website at singlestoneconsulting.com.