Stanford Code the Change Guides Documentation

Stanford Code the Change

May 16, 2021

TECHNICAL GUIDES

CHAPTER

ONE

TESTING ANDROID APPS

1.1 Why Test?

Automated testing has benefits both for the overall organization and for individual developers. For example, tests help catch problems before code is shipped to users, lowering the costs of fixing. A solid set of tests that thoroughly check all functionality also let developers experiment more and take risks, safe in the knowledge that they won't break any existing functions without a test letting them know.

1.2 Writing Tests

1.2.1 Types of Testing

Classifications of Tests:

- Unit Tests: Small, fast tests that check a tiny bit of functionality. For example, a unit test for a calculator might check that *1 + 1 = 2*. These are usually run in an unrealistic environment so that a small part of the product can be tested in isolation. This means that these types of tests can miss problems that arise only when different small parts of the product interact with each other. The narrowness of unit tests also make it easy to pin down a bug, since you know exactly what is tested by each test.
- Integration and End-to-End Tests: Take longer, but test the entire product together. This catches more subtle bugs, but those bugs can be hard to catch if they aren't caught by a unit test.

Unit Tests

Testing in Android, and in Java more generally, is usually done with JUnit. While the latest version is JUnit 5, the Android documentation refers to JUnit 4 as the most recent. The testing directory is `[module*name]/src/test/java/`, and it is created automatically whenever Android Studio creates a new project. The following should be included in your dependencies in `build.gradle`:

```
dependencies {
    // Required -- JUnit 4 framework
    testImplementation 'junit:junit:4.12'
    // Optional -- Mockito framework
    testImplementation 'org.mockito:mockito-core:1.10.19'
    // Optional -- Hamcrest matchers like is()
    testImplementation 'org.hamcrest:hamcrest-library:1.3'
    // Optional -- Robolectric and Android support library needed for JSON
    testImplementation 'org.robolectric:robolectric:3.8'
    testImplementation 'com.android.support.test:runner:1.0.2'
```

(continues on next page)

(continued from previous page)

```
    testImplementation 'com.android.support.test:rules:1.0.2'
}
```

This will import the dependencies JUnit and Mockito.

Create one or more testing classes in the testing directory, and fill them with methods that describe your tests. Methods with tests should be prefixed with `@Test` and use `assertThat` statements to check that the expected results are obtained. Below is one example:

```
import org.junit.Test;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;

public class CalculatorTest {

    @Test
    public void calculator_one_isNumber() {
        assertThat(Calculator.isNumber(1), is(true));
    }
}
```

Note that the `is(true)` statement is a Hamcrest matcher that serves only to make the statement more readable.

Handling Dependencies

Many times, the functionality you want to test relies on code elsewhere in your project. You could just call that code, but then your test will also fail if *that* code has a bug. This destroys the value of unit tests, namely that when they fail you know exactly where the bug is. There are two tools to handle this: Mockito and Robolectric. Mockito lets you define exactly how you expect the other code to behave, but it can get cumbersome to mock complex objects like those that are part of Android. For those, Robolectric contains logic stubs that emulate how Android behaves while still running faster than test would on an emulator.

These considerations are especially important if you need to use any Android classes because Android Studio will execute your tests against a dummy version of the `android.jar` library that throws exceptions in response to any call. This forces you to replace all components of the Android libraries you use with either mocking from Mockito or Robolectric.

To run your unit tests, right-click the directory of tests and select `Run tests`.

For more information, see https://developer.android.com/training/testing/unit-testing/local-unit-tests#java

Mockito

With Mockito, you have to specify how you expect the dependency code to behave. This is done with `when([dependency function call]).thenReturn([expected result])` calls. For example, imagine if the calculator example from before returned a resource string that specified the boolean result like so:

```
public class Calculator {
    private Context context;

    public Calculator(Context inContext) {
      context = inContext;
    }
```

(continues on next page)

(continued from previous page)

```
    public boolean isNumber(int num) {
        // Any int is a number
        return context.getString(R.string.true)
    }
}
```

We could mock the Android Context like this:

```
import org.junit.Test;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertTrue;
import org.junit.runner.RunWith;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;
import android.content.SharedPreferences;


@RunWith(MockitoJUnitRunner.class)
public class CalculatorTest {

    private static final String FAKE_TRUE = "TRUE";

    @Mock
    Context mockContext;

    @Test
    public void calculator_one_isNumber() {
        when(mockContext.getString(R.string.true)).thenReturn(FAKE_TRUE)
        Calculator calc = new Calculator(mockContext);
        assertThat(calc.isNumber(1), is(FAKE_TRUE));
    }
}
```

If you forget to mock something, you will get an error saying that the method you call is not mocked. To solve this, mock the method as shown above.

Guidelines for Well-Structured Unit Tests

- Long, descriptive names are perfect for tests since the method name is often what is displayed if the test fails. The name should include the conditions, action, and expected output for the test. For example,

```
public class Tests {
  public static boolean whenLoggedOut_givenViewProfile_showGuestView() {
    // Log-out the user
    // Open up a profile
    // Check that the guest version of the profile was returned
  }
}
```

- Tests should be focussed on a particular piece of functionality. Leave testing of other parts to separate tests. This will help make it easy to debug when tests fail. Given any method name of a test that failed, you should know exactly where in the codebase the problem is.

1.3 Test-Driven Development

To take testing even further, you can make writing tests the first thing you do before writing any code. This forces you to think through how you want your product to behave before you even start coding. This makes it clearer what the code needs to do. Developers usually have to do this anyway, but by doing it through tests instead of while coding, they both avoid writing code that is later discarded and save time by writing the tests and pinning down the requirements simultaneously.

Importantly, make sure that the tests you write *fail* at first. If they don't, there is a bug in the tests.

1.4 References

1. https://developer.android.com/training/testing/ The embedded video in particular
2. https://developer.android.com/training/testing/fundamentals

CHAPTER

TWO

GUIDE TO PYTHON FLASK UNIT TESTING

2.1 Introduction to Unit Testing Concepts

2.1.1 Why Test?

Testing is generally a best-practice in software development. Some of its benefits include:

- **More robust software**: Tests help catch bugs in your product. If you create new tests as you discover mistakes, you know you won't make the same mistake again.
- **Faster debugging**: If you have lots of small tests covering all of your code, you can track down bugs based on which tests are failing. This can quickly isolate a problem without putting `print` statements everywhere.
- **Automatic documentation**: Good tests can even replace much of your documentation. Instead of reading your documentation, someone can read your tests and see what kinds of input and output your code expects.
- **Freedom to experiment**: With comprehensive testing, you can be assured that a given version of your code works as expected. This lets you add new features or experiment with a new implementation of a feature while knowing that your project still works overall.

2.1.2 Kinds of Testing

There are 3 general classes of testing:

- Unit Testing: Tests small components of the program (units) in isolation. For example, you might test each individual function.
- Integration Testing: Tests larger parts of the program that consist of multiple *units*. This is a little vague, so it might be most helpful to think of as between unit and end-to-end testing.
- End-to-End Testing: Tests that the program as a whole works as expected. For a webapp, this might mean programmatically clicking buttons on the site to complete a common user task. Selenium is a common tool for this.

For this guide, we will focus on unit testing.

2.1.3 Unit Testing

Unit testing takes a reductionist approach by focusing on the parts of a program rather than the whole. As a gross generalization, the idea is:

> Break the program into all its little pieces, each of which has well-defined behavior. If we are sure that each little piece behaves as expected, then when we put the pieces all back together, the program should also behave as expected. Therefore, we write tests to verify that each piece–each *unit*–of the program behaves correctly.

Of course, this is incredibly optimistic. Bugs can often hide in how parts of the program interact, and it is quite time-consuming to actually test each part thoroughly.

This doesn't mean that unit testing isn't *useful* though, just that it's not a silver bullet. We hope that across all three types of testing, nearly all the bugs will be caught. Unit testing has benefits besides bug-catching, though. It is the kind of testing that can replace many of those function-level comments (e.g. docstrings or javadoc), and it is the kind of testing that can help isolate bugs. If you make a change that breaks your program, you can check which unit tests are failing. The code those tests cover is likely where the bug is hiding.

2.2 Unit Testing with Python Flask

2.2.1 Python Flask

Python Flask is a framework that makes it easy to create web apps with Python. This guide will use a Flask app as an example and walk you through creating unit tests for it. Even if you don't use Flask, the unit-testing concepts illustrated are generally applicable.

The app used is a stripped-down version of the CultureMesh FFB app created by Stanford Code the Change. If you have experience working with that code, it may be helpful to know that the following features remain:

- Home Page: `/`
- Viewing Post: `/post/?id=<Post ID Here>`

All other pages may no longer function properly. For example, clicking on the `About` link on the home page will yield an error page. The other pages were removed to simplify this guide and so that we can concentrate on writing unit tests.

2.3 Getting Started

1. Clone this repository by running

```
$ git clone https://github.com/Code-the-Change/flask_tests_workshop.git
```

2. Install python from https://python.org or via your favorite package manager
3. Install `virtualenv`

```
$ pip3 install virtualenv
```

4. If you get a note from `pip` about `virtualenv` not being in your `PATH`, you need to perform this step. `PATH` is a variable accessible from any bash terminal you run, and it tells bash where to look for the commands you enter. It is a list of directories separated by `:`. You can see yours by running `echo $PATH`. To run `virtualenv` commands, you need to add python's packages to your `PATH` by editing or creating the file `~/.bash_profile` on MacOS. To that file add the following lines:

```
PATH="<Path from pip message>:$PATH"
export PATH
```

5. Then you can install dependencies into a virtual environment

```
$ cd flask_tests_workshop
$ virtualenv venv
$ source venv/bin/activate
$ pip install -r requirements.txt
```

6. Make the start script executable

```
chmod 700 run.sh
```

7. Start the app

```
./run.sh
```

   You'll see something like this on the terminal:

```
$ python run.py
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: XXX-XXX-XXX
 * Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
```

   You can then head over to your browser and type in http://127.0.0.1:8080/ on the address bar. You should now see the home page of the app.

Note: By default, the website (even if running locally) really communicates with the live CultureMesh API.

---

8. You can use any editor you like, but this guide will point out some shortcuts for PyCharm (Community Edition) users.

9. The `master` branch of this repository stores the starter code you should start with and make changes to as you follow along with the guide. Completed example code is included in this document. If you would like to see "solution" code, switch to the repository's other branches. For example, to see the finished version, checkout the `tests_written` branch.

```
$ git checkout tests_written
```

   To see all available branches:

```
$ git branch --all
```

2.4 Writing Tests

2.4.1 Set-up Directories

Create the following structure of Python modules:

```
flask_tests_workshop/
    test/
        unit/
            webapp/
                __init__.py
            __init__.py
        __init__.py
```

PyCharm will automatically add the `__init__.py` files if you create each module by right-clicking on the parent directory and selecting `New > Python Package`. Such a deep nesting of directories is unnecessary for this small example, but it is helpful to separate tests into folders like this for your own sanity when you have many more tests.

2.4.2 Create `client` Fixture

We will use a `pytest` feature called "fixtures" to turn our web app into a Python object we can run tests against. Copy the following code into `flask_tests_workshop/test/unit/webapp/__init__.py`. This will make `client` available to all tests under the `webapp` directory. In fact, `pytest` will automatically provide us with an instance of `client` for each test.

```
import pytest
from culturemesh import app

"""Initialize the testing environment

Creates an app for testing that has the configuration flag ``TESTING`` set to
``True``.

"""

# The following function is derived from an example in the Flask documentation
# found at the following URL: http://flask.pocoo.org/docs/1.0/testing/. The
# Flask license statement has been included below as attribution.
#
# Copyright (c) 2010 by the Pallets team.
#
# Some rights reserved.
#
# Redistribution and use in source and binary forms of the software as well as
# documentation, with or without modification, are permitted provided that the
# following conditions are met:
#
#     * Redistributions of source code must retain the above copyright notice,
#       this list of conditions and the following disclaimer.
#     * Redistributions in binary form must reproduce the above copyright
#       notice, this list of conditions and the following disclaimer in the
#       documentation and/or other materials provided with the distribution.
#     * Neither the name of the copyright holder nor the names of its
#       contributors may be used to endorse or promote products derived from
#       this software without specific prior written permission.
```

(continues on next page)

(continued from previous page)

```
#
# THIS SOFTWARE AND DOCUMENTATION IS PROVIDED BY THE COPYRIGHT HOLDERS AND
# CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
# PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR
# CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
# EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
# PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR
# BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER
# IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
# ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE AND DOCUMENTATION, EVEN IF
# ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


@pytest.fixture
def client():
    """Configures the app for testing

    Sets app config variable ``TESTING`` to ``True``

    :return: App for testing
    """

    #app.config['TESTING'] = True
    client = app.test_client()

    yield client
```

The `@pytest.fixture` annotation tells `pytest` that the following function creates (using the `yield` command) an app for testing. In this case, the function doesn't do too much, but it could also configure temporary database files or set configurations for testing (e.g. the commented-out `app.config` line).

2.4.3 Simple Test: Test the Landing Page

`pytest` identifies our tests by searching for files prefixed with `test` and functions starting with `test` within those files. Therefore, create a file called `test_root.py` in the `webapp` directory.

To use the `client` fixture we created before, we need to import it. PyCharm will claim that the import is unused, but `pytest` actually needs it. At the top of the newly created `test_root.py` file, add an import statement:

```
from test.unit.webapp import client
```

Test for the Right Landing Page

Let's first make sure that we are getting the landing page we expect. First, create a function named `test_landing` that takes a parameter `client`. The function declaration should look like `def test_landing(client):`. The `client` parameter will be filled with the fixture we created.

Loading Page into Test

To see the landing page, try pointing a web browser to http://127.0.0.1:8080/. To load this page in our test, add the following lines to the function we just created:

```
landing = client.get("/")
html = landing.data.decode()
```

The first line executes a `GET` request against our app at the URL `/`. This is equivalent to pointing your browser to http://127.0.0.1:8080/. The app doesn't care about the base of the URL (the `http://127.0.0.1:8080` part), so the fact that we don't ask for anything more than that is all that matters (and is expressed as just a `/`). A `GET` request is a type of web request and the type used when you just go to a website and *get* the contents of the page to view.

The second line takes the response to that request, gets the data out, and decodes it from binary back to normal text. This yields the HTML code your web browser renders into the page you see.

Specifying Expectations for Page

Now we can add tests for what we expect the landing page to include. `pytest` uses `assert` statements for this. If the statement following the `assert` keyword is false, an exception is raised, causing the test to fail. Therefore, we can add statements like these to test important attributes of the landing page:

```
# Check that links to `about` and `login` pages exist
assert "<a href=\"/about/\">About</a>" in html
assert " <a href=\"/home/\">Login</a>" in html

# Spot check important text
assert "At CultureMesh, we're building networks to match these " \
       "real-world dynamics and knit the diverse fabrics of our world " \
       "together." in html
assert "1. Join a network you belong to." in html
```

We can also check that the request was successful (indicated by a response code of 200):

```
assert landing.status_code == 200
```

Test for Landing Page Aliases

In this app, the landing page can be accessed either at `/` or at `/index/`. To confirm that this is working as expected, we can add a test like this:

```
def test_landing_aliases(client):
    landing = client.get("/")
    assert client.get("/index/").data == landing.data
```

Finished Test File

In the end, your `flask_tests_workshop/test/unit/webapp/test_root.py` should look like this:

```
from test.unit.webapp import client


def test_landing(client):
    landing = client.get("/")
    html = landing.data.decode()

    # Check that links to `about` and `login` pages exist
    assert "<a href=\"/about/\">About</a>" in html
    assert " <a href=\"/home/\">Login</a>" in html

    # Spot check important text
    assert "At CultureMesh, we're building networks to match these " \
           "real-world dynamics and knit the diverse fabrics of our world " \
           "together." in html
    assert "1. Join a network you belong to." in html

    assert landing.status_code == 200


def test_landing_aliases(client):
    landing = client.get("/")
    assert client.get("/index/").data == landing.data
```

Running Tests

When running the tests, we will need to specify some environment variables that the app expects to be available. This can be automated using a script `test.sh` which will work just like `run.sh`. Fill `test.sh` with the following code:

```
#!/usr/bin/env bash

export CULTUREMESH_API_KEY=1234
export WTF_CSRF_SECRET_KEY=1234
export CULTUREMESH_API_BASE_ENDPOINT=dummy

python -m pytest
```

The variables have dummy values because the app shouldn't actually interact with the server. All those interactions should be mocked, as we will soon see.

You can execute the tests by executing the `test.sh` file

```
$ ./test.sh
```

All tests should pass.

2.4.4 More Complicated Test: Test Viewing a Post

The landing page was relatively simple to test since it was already an isolated unit. When viewing a post, on the other hand, the app normally retrieves information from CultureMesh's servers. In a unit test, we want to isolate our code from CultureMesh's servers. If something goes wrong, then we'll know whether it is our code or the servers that are to blame. For this test, create a `test_posts.py` file under the `webapp` directory.

Understanding Flask Code

In order to isolate our code from the server, we need to understand what our app does when we try to view a post. For an example of viewing a post, start the app and point a web browser to http://127.0.0.1:8080/post/?id=626. When the web browser sends Flask this request, Flask runs one of the app's functions based on the URL from the browser and returns to the browser whatever the function returns. In this case, we can see in `flask_tests_workshop/culturemesh/__init__.py` that the lines

```
from culturemesh.blueprints.posts.controllers import posts

app.register_blueprint(posts, url_prefix='/post')
```

tell Flask that any URLs starting with `/post` (excluding the base `http://127.0.0.1:8080`) should be handled by the `culturemesh.blueprints.posts.controllers` module. As the import statement suggests, this module is defined in `flask_tests_workshop/culturemesh/blueprints/posts/controllers.py`. At the top of that file, you should see

```
@posts.route("/", methods=['GET'])
def render_post():
```

This tells Flask that any URLs with nothing after the `/post` prefix (we know the URL has that prefix since we have already been routed to this file) should be handled by the `render_post()` function (the requests also have to be GET requests).

In the `render_post()` function, a `Client` object is instantiated and then used repeatedly. For example,

```
c = Client(mock=False)
post = c.get_post(current_post_id)
```

The `Client` object is a layer of abstraction that handles all of the app's interactions with the server. To isolate our code from the server, we need to somehow replace all the calls to `Client` with dummies.

Mocking

The canonical way to handle this problem is by *mocking* the `Client` calls. We will replace the called functions with *mocks* that, instead of contacting the server, just return hard-coded objects that we expect to receive from the server in response to the call we expect the mocked function to receive. In our test, we can test that the mocked function was called with the parameters we expect. Thus, with a passing test case, we can say

> We know that the mocked function was called correctly, and we know that it returned what we would expect the real call to return. The end result of the function was as expected, so under the assumption that the mocked function works correctly, our function works correctly.

While it might seem strange to assume something works correctly in a test case, that is what it means to isolate part of the program. When we isolate our code from the server, we are testing whether, under the assumption that the server works correctly, our code works correctly. This lets us test *only* our code and ignore the server.

Python comes with built-in mocking functionality through the `mock` library. `pytest` integrates with `mock` using the `pytest-mock` library. These let us either create our own dummy function or let Python create one for us. We will

use both techniques. To use these features and the fixture from earlier, include these import statements at the top of the `test_posts.py` file:

```
from test.unit.webapp import client
import mock
from mock import call
```

Getting Expected Output

So we know we need to mock all the `c.` statements, but first we need to know what they normally return. If you were writing a new feature you might already know, but in this case we need to find out. It's clunky, but `print` statements are one way to do this. Print out the result of each `c.` call in the `render_post` function, for instance like this:

```
post = c.get_post(current_post_id)
print('---------- c.get_post ----------')
print(c.get_post(current_post_id))
print('----------')
```

Note: Doing this might seem strange. Of course the tests pass since we set the expected output based on the actual output! If you were writing this code from scratch, you would know what outputs to expect and this would all make more sense. However, in this case, we are dealing with existing code that doesn't have tests but that we know works from manually using the app. Writing these tests is useful because it automates what would otherwise be manual testing. In other words, we know that the app works now, and we want to automate the process of making sure it always works.

In the end, your function should look something like this:

```
@posts.route("/", methods=['GET'])
def render_post():

    current_post_id = safe_get_query_arg(request, 'id')

    user_id = current_user.get_id()
    c = Client(mock=False)
    post = c.get_post(current_post_id)
    print('---------- c.get_post ----------')
    print(c.get_post(current_post_id))
    print('----------')

    post['network_title'] = get_network_title(c.get_network(post['id_network']))
    print("---------- c.get_network ----------")
    print(c.get_network(post['id_network']))
    print('----------')
    post['username'] = c.get_user(post["id_user"])["username"]
    print('---------- c.get_user ----------')
    print(c.get_user(post['id_user']))
    print('----------')
    post['time_ago'] = get_time_ago(post['post_date'])

    # NOTE: this will not show more than the latest 100 replies
    replies = c.get_post_replies(post["id"], NUM_REPLIES_TO_SHOW)
    print('---------- replies ----------')
    print(replies)
    print('----------')
```

(continues on next page)

(continued from previous page)

```
    replies = sorted(replies, key=lambda x: int(x['id']))

    error_msg = None

    for reply in replies:
      reply['username'] = c.get_user(reply["id_user"])["username"]
      print('---------- c.get_user(reply) ----------')
      print(c.get_user(reply['id_user']))
      print('----------')
      reply['time_ago'] = get_time_ago(reply['reply_date'])

    new_form = CreatePostReplyForm()

    return render_template(
      'post.html',
      post=post,
      replies=replies,
      num_replies=len(replies),
      curr_user_id=user_id,
      form=new_form,
      error_msg=error_msg
    )
```

Now, run the server and go to http://127.0.0.1:8080/post/?id=626 again. The post should appear in the browser, and in your terminal you should see output like this:

```
 * Running on http://127.0.0.1:8080/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 655-024-032
---------- c.get_post ----------
{'id': 626, 'id_network': 1, 'id_user': 157, 'img_link': None, 'post_class': 'o',
↪'post_date': 'Sun, 26 Aug 2018 22:31:04 GMT', 'post_original': None, 'post_text':
↪"Hi everyone! I'm hoping to move here soon, but I'd like to get a better sense of␣
↪the local community. Would anyone be willing to take a few minutes to talk with me␣
↪about there experiences living here, particularly after leaving home? Thanks!\n",
↪'vid_link': None}
----------
---------- c.get_network ----------
{'city_cur': 'Palo Alto', 'city_origin': None, 'country_cur': 'United States',
↪'country_origin': 'United States', 'date_added': 'Tue, 12 Jan 2016 05:51:19 GMT',
↪'id': 1, 'id_city_cur': 332851, 'id_city_origin': None, 'id_country_cur': 47228,
↪'id_country_origin': 47228, 'id_language_origin': None, 'id_region_cur': 55833, 'id_
↪region_origin': 56020, 'img_link': None, 'language_origin': None, 'network_class':
↪'rc', 'region_cur': 'California', 'region_origin': 'Michigan', 'twitter_query_level
↪': 'A'}
----------
---------- c.get_user ----------
{'about_me': "I'm from Michigan", 'act_code': '764efa883dda1e11db47671c4a3bbd9e',
↪'company_news': None, 'confirmed': 0, 'events_interested_in': None, 'events_upcoming
↪': None, 'first_name': 'c', 'fp_code': None, 'gender': 'n', 'id': 157, 'img_link':
↪'https://www.culturemesh.com/user_images/null', 'last_login': '0000-00-00 00:00:00',
↪ 'last_name': 's', 'network_activity': None, 'register_date': 'Sun, 02 Dec 2018␣
↪16:33:20 GMT', 'role': 0, 'username': 'cs'}
----------
---------- replies ----------
```

(continues on next page)

(continued from previous page)

```
[{'id': 465, 'id_network': 1, 'id_parent': 626, 'id_user': 157, 'reply_date': 'Sun,␣
↪02 Dec 2018 18:20:40 GMT', 'reply_text': "This is a test reply, but I'd be happy to␣
↪talk to you.  "}, {'id': 461, 'id_network': 1, 'id_parent': 626, 'id_user': 172,
↪'reply_date': 'Tue, 18 Sep 2018 16:09:13 GMT', 'reply_text': 'This is another test␣
↪reply.  Do not mind me, but welcome to Palo Alto! Hope you like it here'}, {'id':␣
↪460, 'id_network': 1, 'id_parent': 626, 'id_user': 171, 'reply_date': 'Tue, 18 Sep␣
↪2018 16:07:16 GMT', 'reply_text': 'This is only a test reply.  But I am sure␣
↪someone else here can help you out.'}]
----------
---------- c.get_user(reply) ----------
{'about_me': 'I like to cook and watch movies.  I recently made some clam chowder and␣
↪it was amazing :D.  Originally from Mexico, now living in the bay area.', 'act_code
↪': '', 'company_news': None, 'confirmed': 0, 'events_interested_in': None, 'events_
↪upcoming': None, 'first_name': 'Alan', 'fp_code': None, 'gender': None, 'id': 171,
↪'img_link': None, 'last_login': '0000-00-00 00:00:00', 'last_name': 'Last name',
↪'network_activity': None, 'register_date': 'Thu, 20 Sep 2018 10:30:04 GMT', 'role':␣
↪0, 'username': 'aefl'}
----------
---------- c.get_user(reply) ----------
{'about_me': 'Live and learn', 'act_code': '', 'company_news': None, 'confirmed': 0,
↪'events_interested_in': None, 'events_upcoming': None, 'first_name': 'Alan 2.0',
↪'fp_code': None, 'gender': None, 'id': 172, 'img_link': None, 'last_login': '0000-
↪00-00 00:00:00', 'last_name': 'Lastname', 'network_activity': None, 'register_date
↪': 'Wed, 19 Sep 2018 22:15:15 GMT', 'role': 0, 'username': 'aefl2'}
----------
---------- c.get_user(reply) ----------
{'about_me': "I'm from Michigan", 'act_code': '764efa883dda1e11db47671c4a3bbd9e',
↪'company_news': None, 'confirmed': 0, 'events_interested_in': None, 'events_upcoming
↪': None, 'first_name': 'c', 'fp_code': None, 'gender': 'n', 'id': 157, 'img_link':
↪'https://www.culturemesh.com/user_images/null', 'last_login': '0000-00-00 00:00:00',
↪ 'last_name': 's', 'network_activity': None, 'register_date': 'Sun, 02 Dec 2018␣
↪16:33:20 GMT', 'role': 0, 'username': 'cs'}
----------
127.0.0.1 - - [07/Jan/2019 09:42:11] "GET /post/?id=626 HTTP/1.1" 200 -
127.0.0.1 - - [07/Jan/2019 09:42:11] "GET /static/css/culturemesh-style.css HTTP/1.1"␣
↪200 -
127.0.0.1 - - [07/Jan/2019 09:42:11] "GET /static/css/bootstrap.css HTTP/1.1" 200 -
127.0.0.1 - - [07/Jan/2019 09:42:11] "GET /static/fonts/font-awesome/css/fontawesome-
↪all.min.css HTTP/1.1" 200 -
```

Now you know what the functions we will mock normally return! They are all JSON objects or lists of JSON objects, which Python represents as dictionaries or lists of dictionaries, respectively. You can copy-and-paste them straight into your test file and assign them to variables. The top of your test file should include some of those objects like this:

```
view_post_post = {'id': 626, 'id_network': 1, 'id_user': 157, 'img_link': None,
                  'post_class': 'o',
                  'post_date': 'Sun, 26 Aug 2018 22:31:04 GMT',
                  'post_original': None,
                  'post_text': "Hi everyone! I'm hoping to move here soon, but "
                               "I'd like to get a better sense of the local "
                               "community. Would anyone be willing to take a "
                               "few minutes to talk with me about there "
                               "experiences living here, particularly after "
                               "leaving home? Thanks!\n", 'vid_link': None}
view_post_net = {'city_cur': 'Palo Alto', 'city_origin': None,
                 'country_cur': 'United States',
```

(continues on next page)

(continued from previous page)

```
                  'country_origin': 'United States',
                  'date_added': 'Tue, 12 Jan 2016 05:51:19 GMT', 'id': 1,
                  'id_city_cur': 332851, 'id_city_origin': None,
                  'id_country_cur': 47228, 'id_country_origin': 47228,
                  'id_language_origin': None, 'id_region_cur': 55833,
                  'id_region_origin': 56020, 'img_link': None,
                  'language_origin': None, 'network_class': 'rc',
                  'region_cur': 'California', 'region_origin': 'Michigan',
                  'twitter_query_level': 'A'}
view_post_replies = [{'id': 465, 'id_network': 1, 'id_parent': 626,
                      'id_user': 157,
                      'reply_date': 'Sun, 02 Dec 2018 18:20:40 GMT',
                      'reply_text': "This is a test reply, but I'd be happy "
                                    "to talk to you.  "},
                     {'id': 461, 'id_network': 1, 'id_parent': 626,
                      'id_user': 172,
                      'reply_date': 'Tue, 18 Sep 2018 16:09:13 GMT',
                      'reply_text': 'This is another test reply.  Do not mind '
                                    'me, but welcome to Palo Alto! Hope you '
                                    'like it here'},
                     {'id': 460, 'id_network': 1, 'id_parent': 626,
                      'id_user': 171,
                      'reply_date': 'Tue, 18 Sep 2018 16:07:16 GMT',
                      'reply_text': 'This is only a test reply.  But I am sure '
                                    'someone else here can help you out.'}]
```

When we mock functions, we can specify these objects as the object to return and Python will handle creating the dummy function automatically.

However, the `c.get_user` function is different because it is called more than once. This means we can't specify a single return value when we mock it. Instead we will have to write the dummy function ourselves and return the user object whose ID matches the parameter. Here is an example:

```
def mock_client_get_user(id):
    users = [
        {'about_me': "I'm from Michigan",
         'act_code': '764efa883dda1e11db47671c4a3bbd9e',
         'company_news': None, 'confirmed': 0,
         'events_interested_in': None, 'events_upcoming': None,
         'first_name': 'c', 'fp_code': None, 'gender': 'n', 'id': 157,
         'img_link': 'https://www.culturemesh.com/user_images/null',
         'last_login': '0000-00-00 00:00:00', 'last_name': 's',
         'network_activity': None,
         'register_date': 'Sun, 02 Dec 2018 16:33:20 GMT', 'role': 0,
         'username': 'cs'},
        {'about_me': 'I like to cook and watch movies.  I recently made some '
                     'clam chowder and it was amazing :D.  Originally from '
                     'Mexico, now living in the bay area.',
         'act_code': '', 'company_news': None, 'confirmed': 0,
         'events_interested_in': None, 'events_upcoming': None,
         'first_name': 'Alan', 'fp_code': None, 'gender': None, 'id': 171,
         'img_link': None, 'last_login': '0000-00-00 00:00:00',
         'last_name': 'Last name', 'network_activity': None,
         'register_date': 'Thu, 20 Sep 2018 10:30:04 GMT', 'role': 0,
         'username': 'aefl'},
        {'about_me': 'Live and learn', 'act_code': '', 'company_news': None,
         'confirmed': 0, 'events_interested_in': None, 'events_upcoming': None,
```

(continues on next page)

(continued from previous page)

```
        'first_name': 'Alan 2.0', 'fp_code': None, 'gender': None, 'id': 172,
        'img_link': None, 'last_login': '0000-00-00 00:00:00',
        'last_name': 'Lastname', 'network_activity': None,
        'register_date': 'Wed, 19 Sep 2018 22:15:15 GMT', 'role': 0,
        'username': 'aefl2'}
    ]
    for user in users:
        if user['id'] == id:
            return user
    raise ValueError("User ID {} is unknown to mock_client_get_user".format(id))
```

Now that we can specify the expected outputs of the mocked functions, we can use `@mock.patch` annotations to actually do the mocking. This looks like

Performing Mocking

```
@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_post',
            return_value=view_post_post)
@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_network',
            return_value=view_post_net)
@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_user',
            side_effect=mock_client_get_user)
@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_post_replies',
            return_value=view_post_replies)
def test_view_post(replies, user, net, post, client):
```

The first argument to each `@mock.patch` call is a string that specifies the function to mock–to replace with a dummy function. Importantly, it specifies the function based on where it is *used*, not where it is *defined*. In the top case, the `Client.get_post` function is defined at `culturemesh.client.Client.get_post`. However, it is used within the `culturemesh.blueprints.posts.controllers` file, which imports `Client`.

Note: The distinction between where a function is defined and where it is used is a common source of problems and easy to get wrong. Some trial and error may be necessary here.

Each `@mock.patch` statement causes another object (the mocked function) to be passed as an argument when the test is run. This is where the `replies, user, net, post` arguments come from. Note that they are ordered from left to right to match the order of the `@mock.patch` calls from bottom to top. This is strange, but it results from the order in which Python applies annotations.

Specifying Return Values

Each `@mock.patch` call also specifies how Python should determine the return value. For those functions that are only called once, a single value can be specified by the `return_value` keyword argument. Python will then handle creating the mock function for us. For functions that are called more than once, we have to create the mock function ourselves and use the `side_effect` keyword argument instead.

Adding Test Cases

Now, we can add assertions like before.

```
result = client.get('/post/?id=626')
html = result.data.decode()

# Check that replies are displayed
assert "This is another test reply.  Do not mind me, " in html
assert 'This is only a test reply.' in html

# Check that reply author username displayed
assert 'aefl' in html

# Check that post text displayed
assert 'Hi everyone!' in html

# Check that post author username displayed
assert 'cs' in html

# Check that network name displayed
assert 'From Michigan, United States in Palo Alto, California, ' \
       'United States' in html
```

In addition, we can check that the mocked functions were called as expected. Instead of the `assert` keyword, we can call functions on the mock functions provided as parameters to our test. `assert_called_with` tests whether the mock function was called with the arguments you pass to the assertion. To test for multiple calls in a particular order, use `has_calls` instead and provide a list of `call` objects. When creating each `call` object, pass the parameters you expect the mocked function to be called with. Combining these techniques might result in test code like this:

```
replies.assert_called_with(626, 100)
user.assert_has_calls([call(157), call(171), call(172), call(157)])
net.assert_called_with(1)
post.assert_called_with('626')
```

If you didn't know what arguments to expect, you could take a guess randomly and run the test. `pytest` will report the failing test case and show you what arguments the mocked function actually received.

Finished Test Function

In the end, you should have a `test_posts.py` file that looks like this:

```
from test.unit.webapp import client
import mock
from mock import call


view_post_post = {'id': 626, 'id_network': 1, 'id_user': 157, 'img_link': None,
                  'post_class': 'o',
                  'post_date': 'Sun, 26 Aug 2018 22:31:04 GMT',
                  'post_original': None,
                  'post_text': "Hi everyone! I'm hoping to move here soon, but "
                               "I'd like to get a better sense of the local "
                               "community. Would anyone be willing to take a "
                               "few minutes to talk with me about there "
                               "experiences living here, particularly after "
```

(continues on next page)

(continued from previous page)

```
                                "leaving home? Thanks!\n", 'vid_link': None}
view_post_net = {'city_cur': 'Palo Alto', 'city_origin': None,
                 'country_cur': 'United States',
                 'country_origin': 'United States',
                 'date_added': 'Tue, 12 Jan 2016 05:51:19 GMT', 'id': 1,
                 'id_city_cur': 332851, 'id_city_origin': None,
                 'id_country_cur': 47228, 'id_country_origin': 47228,
                 'id_language_origin': None, 'id_region_cur': 55833,
                 'id_region_origin': 56020, 'img_link': None,
                 'language_origin': None, 'network_class': 'rc',
                 'region_cur': 'California', 'region_origin': 'Michigan',
                 'twitter_query_level': 'A'}
view_post_replies = [{'id': 465, 'id_network': 1, 'id_parent': 626,
                      'id_user': 157,
                      'reply_date': 'Sun, 02 Dec 2018 18:20:40 GMT',
                      'reply_text': "This is a test reply, but I'd be happy "
                                    "to talk to you.  "},
                     {'id': 461, 'id_network': 1, 'id_parent': 626,
                      'id_user': 172,
                      'reply_date': 'Tue, 18 Sep 2018 16:09:13 GMT',
                      'reply_text': 'This is another test reply.  Do not mind '
                                    'me, but welcome to Palo Alto! Hope you '
                                    'like it here'},
                     {'id': 460, 'id_network': 1, 'id_parent': 626,
                      'id_user': 171,
                      'reply_date': 'Tue, 18 Sep 2018 16:07:16 GMT',
                      'reply_text': 'This is only a test reply.  But I am sure '
                                    'someone else here can help you out.'}]


def mock_client_get_user(id):
    users = [
        {'about_me': "I'm from Michigan",
         'act_code': '764efa883dda1e11db47671c4a3bbd9e',
         'company_news': None, 'confirmed': 0,
         'events_interested_in': None, 'events_upcoming': None,
         'first_name': 'c', 'fp_code': None, 'gender': 'n', 'id': 157,
         'img_link': 'https://www.culturemesh.com/user_images/null',
         'last_login': '0000-00-00 00:00:00', 'last_name': 's',
         'network_activity': None,
         'register_date': 'Sun, 02 Dec 2018 16:33:20 GMT', 'role': 0,
         'username': 'cs'},
        {'about_me': 'I like to cook and watch movies.  I recently made some '
                     'clam chowder and it was amazing :D.  Originally from '
                     'Mexico, now living in the bay area.',
         'act_code': '', 'company_news': None, 'confirmed': 0,
         'events_interested_in': None, 'events_upcoming': None,
         'first_name': 'Alan', 'fp_code': None, 'gender': None, 'id': 171,
         'img_link': None, 'last_login': '0000-00-00 00:00:00',
         'last_name': 'Last name', 'network_activity': None,
         'register_date': 'Thu, 20 Sep 2018 10:30:04 GMT', 'role': 0,
         'username': 'aefl'},
        {'about_me': 'Live and learn', 'act_code': '', 'company_news': None,
         'confirmed': 0, 'events_interested_in': None, 'events_upcoming': None,
         'first_name': 'Alan 2.0', 'fp_code': None, 'gender': None, 'id': 172,
         'img_link': None, 'last_login': '0000-00-00 00:00:00',
         'last_name': 'Lastname', 'network_activity': None,
```

(continues on next page)

(continued from previous page)

```
         'register_date': 'Wed, 19 Sep 2018 22:15:15 GMT', 'role': 0,
         'username': 'aefl2'}
    ]
    for user in users:
        if user['id'] == id:
            return user
    raise ValueError("User ID {} is unknown to mock_client_get_user".format(id))


@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_post',
            return_value=view_post_post)
@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_network',
            return_value=view_post_net)
@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_user',
            side_effect=mock_client_get_user)
@mock.patch('culturemesh.blueprints.posts.controllers.Client.get_post_replies',
            return_value=view_post_replies)
def test_view_post(replies, user, net, post, client):
    result = client.get('/post/?id=626')
    html = result.data.decode()

    # Check that replies are displayed
    assert "This is another test reply.  Do not mind me, " in html
    assert 'This is only a test reply.' in html

    # Check that reply author username displayed
    assert 'aefl' in html

    # Check that post text displayed
    assert 'Hi everyone!' in html

    # Check that post author username displayed
    assert 'cs' in html

    # Check that network name displayed
    assert 'From Michigan, United States in Palo Alto, California, ' \
           'United States' in html

    replies.assert_called_with(626, 100)
    user.assert_has_calls([call(157), call(171), call(172), call(157)],
                          any_order=False)
    net.assert_called_with(1)
    post.assert_called_with('626')
```

2.5 Licensing and Attribution

This work was initially created for a workshop at Stanford Code the Change.

CHAPTER

THREE

GITTING AROUND WITH ROBERT FROST: A GUIDE TO GETTING STARTED WITH GIT

To learn Git properly, visit the actual documentation: https://git-scm.com/doc

3.1 Setup

First, install Git here: https://git-scm.com/downloads

Next, create a GitHub account: https://github.com/join

To make sure that you git works for you, open up a terminal window and try the git command:

```
$ git
usage: git [--version] [--help] [-C <path>] [-c <name>=<value>]
       [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
       [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
       [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
       <command> [<args>]
... Even more console output ...
```

After that, config Git with your name and email from your GitHub account:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Next, we will have to clone (download) our starter project, and move the `poem.txt` file to a new directory.

```
$ git clone https://github.com/codethechange/git_workshop
$ mkdir git_practice
$ mv git_workshop/poem.txt git_practice
$ cd git_practice
$ cat poem.txt
Stopping By Woods On A Snowy Evening

    By Robert Lee Frost
...
```

Irrelevant note: Definitely check out Steve Jackson's version of this poem.

3.2 Motivation - Too Many Drafts

Let's pretend that you came up with this poem in `poem.txt`. This stanza took more than one take, however. In reality, you pored over each line many times. Now, imagine you want to make some improvements to the poem. You want to save this version of the draft, but don't want to lose your previous draft. What do you recommend? A common approach is to save a new file with a new name, perhaps `poemv2.txt`. As you become increasingly involved in the writing process, you will realize that this mode of saving drafts will become unwieldy quickly. The directory will quickly balloon to include `poemv25.txt`, `poemfinal.txt`, `realpoemfinal.txt`, `poemfinalv2.txt`, and so on. All of these files will have lots of redundant information, and it is tricky to organize and label your files so that you know both the chronological ordering of each version and what you accomplished in each version.

This is where Git begins to save the day.

3.3 The Necessities: Committing, Branching, and Merging

3.3.1 Getting Started

Git stores all version histories of your project directory (which we will now call a **repository**) while still allowing you be viewing files in only one version at a time.

To make a your directory a repository, run the following command:

```
$ ls -a
.           ..              poem.txt
$ git init
Initialized empty Git repository in ...
$ ls -a
.           ..              .git            poem.txt
$ git log
fatal: your current branch 'master' does not have any commits yet
```

`git init` created a hidden directory named `.git` in your folder. Thus, you have a repository. Congratulations! Unfortunately, we haven't added any versions yet.

DO NOT touch that .git folder. EVER. Your .git folder is like a museum :-)

3.3.2 Committing

Each version is called a **commit**. In fact, Git stores only the *changes* between each version in a commit. As such, you can even choose which changed files you want to include in a commit. This is called **staging**. For now, we will add all changed files into the staging area for a commit. To learn about how to ignore files, read this.

```
$ git add .
```

This adds all files that have been changed (which is poem.txt, since we are creating art from the void) were added into the staging area. Let's commit them now.

```
$ git commit -m "first commit"
[master (root-commit) 7487f1b] first commit
1 file changed, 25 insertions(+)
create mode 100644 poem.txt
$ git log
commit 25e813bb0f8d4250c207af099700359e57709e30 (HEAD -> master)
```

(continues on next page)

(continued from previous page)

```
Author: DrewGregory <djgregny@gmail.com>
Date:   Thu Jan 31 02:07:31 2019 -0800

    First commit
```

Now, our first revision. To be even more descriptive, make the last line "And miles to go before I sleep well."

Don't think that Git didn't notice:

```
$ git status
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
    modified:   poem.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

In fact, we can even see the changes that were made line-by-line:

```
$ git diff
diff --git a/poem.txt b/poem.txt
index 33148d6..5b6adeb 100644
--- a/poem.txt
+++ b/poem.txt
@@ -22,4 +22,4 @@ Of easy wind and downy flake.
The woods are lovely, dark, and deep,
But I have promises to keep,
And miles to go before I sleep,
-And miles to go before I sleep.
+And miles to go before I sleep well.
```

Let's stage (`git add`) and commit (`git commit`) this change:

```
$ git add .
$ git commit -m "new ending"
[master 7a665a6] new ending
1 file changed, 1 insertion(+), 1 deletion(-)
```

Notice that the commit only stores what files are changed and which lines are changed for each file.

Now, we are going to experiment with this poem. We will tackle two approaches: changing the tone and accompanying the poem with some ASCII art. We want to work on each experiment separately and somewhat asynchronously ...

3.3.3 Branching

A branch is a sequence of commits representing some approach to a project. In general, a branch represents some feature that you want to add to a project. By default, every repository has a *master* branch. Don't know which branch you're on? Just check:

```
$ git status
On branch master
nothing to commit, working tree clean
```

Now, let's create a new branch:

```
$ git checkout -b italian
Switched to a new branch 'italian'
```

`git checkout` is a command for switching branches. This will modify all the (tracked) files in your repository to reflect the version you are checking out. The `-b` flag signifies that we want to make a new branch with the following name. In this case, our new branch's files will be duplicates of the branch we were on before until we make our first commit on this branch. To experiment with a change in tone, replace "snow" with "dough" and "woods" with "pizzas".

```
$ cat poem.txt
Stopping By Pizzas On A Doughy Evening

    By Robert Lee Frost



Whose pizzas these are I think I know.
His house is in the village, though;
He will not see me stopping here
To watch his pizzas fill up with dough.

My little horse must think it's queer
To stop without a farmhouse near
Between the pizzas and frozen lake
The darkest evening of the year.

He gives his harness bells a shake
To ask if there's some mistake.
The only other sound's the sweep
Of easy wind and downy flake.

The pizzas are lovely, dark, and deep,
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep well.
$ git add .
$ git commit -m "snow->dough, woods->pizzas"
[italian bc452ff] snow->dough, woods->pizzas
1 file changed, 5 insertions(+), 5 deletions(-)
```

Let's now explore what ascii art can do to the poem. We want to experiment with art separately from our food-centered shift, however, so we want to branch off the original poem.

```
$ git checkout master
Switched to branch 'master'
$ git checkout -b ascii_art
Switched to a new branch 'ascii_art'
```

Now, insert this ascii art:

```
$ cat poem.txt
Stopping By Woods On A Snowy Evening ...

    By Robert Lee Frost
            \/
        _\_\/\/_/_
          _\_\/_/_
        __/_/\_\__
```

(continues on next page)

(continued from previous page)

```
         / /\/\ \
            /\


Whose woods these are I think I know.
His house is in the village, though;
He will not see me stopping here
To watch his woods fill up with snow.

My little horse must think it's queer
To stop without a farmhouse near
Between the woods and frozen lake
The darkest evening of the year.

He gives his harness bells a shake
To ask if there's some mistake.
The only other sound's the sweep
Of easy wind and downy flake.

The woods are lovely, dark, and deep,
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep well.
$ git add .
$ git commit -m "added a snowflake"
[ascii_art 18b593a] added a snowflake
1 file changed, 7 insertions(+), 2 deletions(-)
```

Fantastic! After much thought, you think the poem could both use some italian integration and some ascii eye candy (perhaps the snowflake goes with 'Frost?'). Since the `master` branch should generally represent production-ready code, let's **merge** the `ascii_art` and `italian` branches into the `master` branch so that commits from both branches end up in the master branch. In order to do so, first `git checkout` *into* the branch that you want the code to end up, and `git merge` *from* the branch that you want to get code. In this example, we want text from our feature branches to end up in the `master` branch.

```
$ git checkout master
Switched to branch 'master'
$ git merge ascii_art
Updating 7a665a6..18b593a
Fast-forward
poem.txt | 9 +++++++--
1 file changed, 7 insertions(+), 2 deletions(-)
$ cat poem.txt
Stopping By Woods On A Snowy Evening ...

    By Robert Lee Frost
            \/
        _\_\/\/_/_
        _\_\/_/_
        __/_/\_\__
        / /\/\ \
            /\

Whose woods these are I think I know.
His house is in the village, though;
He will not see me stopping here
```

(continues on next page)

(continued from previous page)

```
To watch his woods fill up with snow.

My little horse must think it's queer
To stop without a farmhouse near
Between the woods and frozen lake
The darkest evening of the year.

He gives his harness bells a shake
To ask if there's some mistake.
The only other sound's the sweep
Of easy wind and downy flake.

The woods are lovely, dark, and deep,
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep well.
$ git log
commit 18b593a108fea601d9213fb9e9146f161710d9a3 (HEAD -> master, ascii_art)
Author: DrewGregory <djgregny@gmail.com>
Date:   Thu Jan 31 23:45:39 2019 -0800

    added a snowflake

commit 7a665a68bbc8aa8634940f5e2b7f9b0237860ec9
Author: DrewGregory <djgregny@gmail.com>
Date:   Thu Jan 31 13:25:53 2019 -0800

    new ending

commit 25e813bb0f8d4250c207af099700359e57709e30
Author: DrewGregory <djgregny@gmail.com>
Date:   Thu Jan 31 02:07:31 2019 -0800

    First commit
```

If you have tried merging before, that may have felt *too* easy. `master` suddenly has the 'added a snowflake' commit from `ascii_art`, andthe file was updated. The developers of Git were quite smart, so Git will handle most of file merging fort you. How do they merge changes on the same line, you ask? Well, let's see.

```
$ git merge italian
Auto-merging poem.txt
CONFLICT (content): Merge conflict in poem.txt
Automatic merge failed; fix conflicts and then commit the result.
```

In other words, Git can't. You have to resolve them yourself. I don't know you, but when I first encountered this console response, my first response was . . .

Panic. Give up. Believe that understanding Git is hopeless. Just walk away.

If this was your original response as well, I understand, but don't worry. Let's just see what happened to poem.txt.

```
$ cat poem.txt
<<<<<<< HEAD
Stopping By Woods On A Snowy Evening ...
=======
Stopping By Pizzas On A Doughy Evening
>>>>>>> italian
```

(continues on next page)

(continued from previous page)

```
    By Robert Lee Frost
            \/
        _\_\/\/_/_
        _\_\/_/_
        __/_/\_\__
        / /\/\ \
            /\


Whose pizzas these are I think I know.
His house is in the village, though;
He will not see me stopping here
To watch his pizzas fill up with dough.

My little horse must think it's queer
To stop without a farmhouse near
Between the pizzas and frozen lake
The darkest evening of the year.

He gives his harness bells a shake
To ask if there's some mistake.
The only other sound's the sweep
Of easy wind and downy flake.

The pizzas are lovely, dark, and deep,
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep well.
```

Note the only weird modification is the top part of the file. Both branches modified the first line, so Git merely displays which branch had which line and expects you to modify the file manually to choose the version you want. The area between `<<<<<<< HEAD` and `=======` is the section that the our current branch (`master`) had, and the area between `=======` and `italian` is what the `italian` branch contained. Again, *manually* edit the file to produce the combined result you would like. In other words:

```
$ cat poem.txt
Stopping By Woods On A Doughy Evening ...

    By Robert Lee Frost
            \/
        _\_\/\/_/_
        _\_\/_/_
        __/_/\_\__
        / /\/\ \
            /\


Whose pizzas these are I think I know.
His house is in the village, though;
He will not see me stopping here
To watch his pizzas fill up with dough.

My little horse must think it's queer
To stop without a farmhouse near
Between the pizzas and frozen lake
```

(continues on next page)

(continued from previous page)

```
The darkest evening of the year.

He gives his harness bells a shake
To ask if there's some mistake.
The only other sound's the sweep
Of easy wind and downy flake.

The pizzas are lovely, dark, and deep,
But I have promises to keep,
And miles to go before I sleep,
And miles to go before I sleep well.
$ git diff
diff --cc poem.txt
index fe9857e,5d4494f..0000000
--- a/poem.txt
+++ b/poem.txt
@@@ -1,18 -1,13 +1,18 @@@
diff --cc poem.txt
index fe9857e,5d4494f..0000000
--- a/poem.txt
+++ b/poem.txt
@@@ -1,18 -1,13 +1,18 @@@
- Stopping By Woods On A Snowy Evening ...
-Stopping By Pizzas On A Doughy Evening
++Stopping By Woods On A Doughy Evening ...
    By Robert Lee Frost
-
+            \/
+        _\_\/\/_/_
+         _\_\/_/_
+        __/_/\_\__
+         / /\/\ \
+            /\
- Whose woods these are I think I know.
+ Whose pizzas these are I think I know.
His house is in the village, though;
He will not see me stopping here
$ git add .
$ git commit -m "merged successfully"
[master f9dca7a] merged successfully
$ git log
commit f9dca7a219e1e6341195d244bc6fbc428a13c724 (HEAD -> master)
Merge: 18b593a bc452ff
Author: DrewGregory <djgregny@gmail.com>
Date:   Fri Feb 1 00:42:13 2019 -0800

    merged successfully

commit 18b593a108fea601d9213fb9e9146f161710d9a3 (ascii_art)
Author: DrewGregory <djgregny@gmail.com>
Date:   Thu Jan 31 23:45:39 2019 -0800

    added a snowflake

commit bc452fff1fb50f69721285979883 84f4c90054af (italian)
Author: DrewGregory <djgregny@gmail.com>
Date:   Thu Jan 31 14:27:08 2019 -0800
```

(continues on next page)

(continued from previous page)

```
    snow->dough, woods->pizzas

commit 7a665a68bbc8aa8634940f5e2b7f9b0237860ec9
Author: DrewGregory <djgregny@gmail.com>
Date:   Thu Jan 31 13:25:53 2019 -0800
```

To visualize this, there's a really cool graph command (credits to U8NWXD for showing me this):

```
$ git log --graph --abbrev-commit --decorate --all
*   commit f9dca7a (HEAD -> master)
|\  Merge: 18b593a bc452ff
| | Author: DrewGregory <djgregny@gmail.com>
| | Date:   Fri Feb 1 00:42:13 2019 -0800
| |
| |     merged successfully
| |
| * commit bc452ff (italian)
| | Author: DrewGregory <djgregny@gmail.com>
| | Date:   Thu Jan 31 14:27:08 2019 -0800
| |
| |     snow->dough, woods->pizzas
| |
* | commit 18b593a (ascii_art)
|/  Author: DrewGregory <djgregny@gmail.com>
|   Date:   Thu Jan 31 23:45:39 2019 -0800
|
|       added a snowflake
|
* commit 7a665a6
| Author: DrewGregory <djgregny@gmail.com>
| Date:   Thu Jan 31 13:25:53 2019 -0800
|
```

3.4 The Necessities: Pushing, Pulling, and Pull Requests with GitHub

3.4.1 Remotes

With your newfound Git skills, you may find some other project buddies. Indeed, merging and branching only locally may feel somewhat unnecessary when most ordinary humans work on only one task at a time. Git branching and merging achieves its full form in the context of collaborating on software projects.

First, let's host this repo (slang for repository: yeah I'm cool) on GitHub. You should have an account by this point, so sign in and create a new repository. Next, let's add a **remote** to your local repository on your computer, which allows you to associate your local repository with your hosted repo on GitHub.

```
$ git remote -v
$ git remote add origin https://github.com/<GitHub username>/<repo name>.git
$ git remote -v
origin      https://github.com/<GitHub username>/<repo name>.git (fetch)
```

3.4.2 Pushing

Now, we can send commits on branches (**push**) to our hosted GitHub repo.

```
$ git push -u origin master
Counting objects: 15, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (10/10), done.
Writing objects: 100% (15/15), 3.54 KiB | 906.00 KiB/s, done.
Total 15 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To https://github.com/DrewGregory/laughing-dollop.git
* [new branch]      master -> master
Branch 'master' set up to track remote branch 'master' from 'origin'
```

Let's break down this command: `git push <remote> <branch>`. If the branch does not already existed on the hosted repository, you have to add the `-u` flag (which means set upstream). That's about it! You should be able to see your commits on the hosted GitHub repo.

3.4.3 Pulling

Have a friend push some commits on your branch (alternatively, clone the repo yourself in a separate directory, commit your changes, and push them). To get the commits from the remote branch. Run this command:

```
$ git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/DrewGregory/laughing-dollop
f9dca7a..e2da0b7  master     -> origin/master
Updating f9dca7a..e2da0b7
Fast-forward
poem.txt | 2 +-
1 file changed, 1 insertion(+), 1 deletion(-)
```

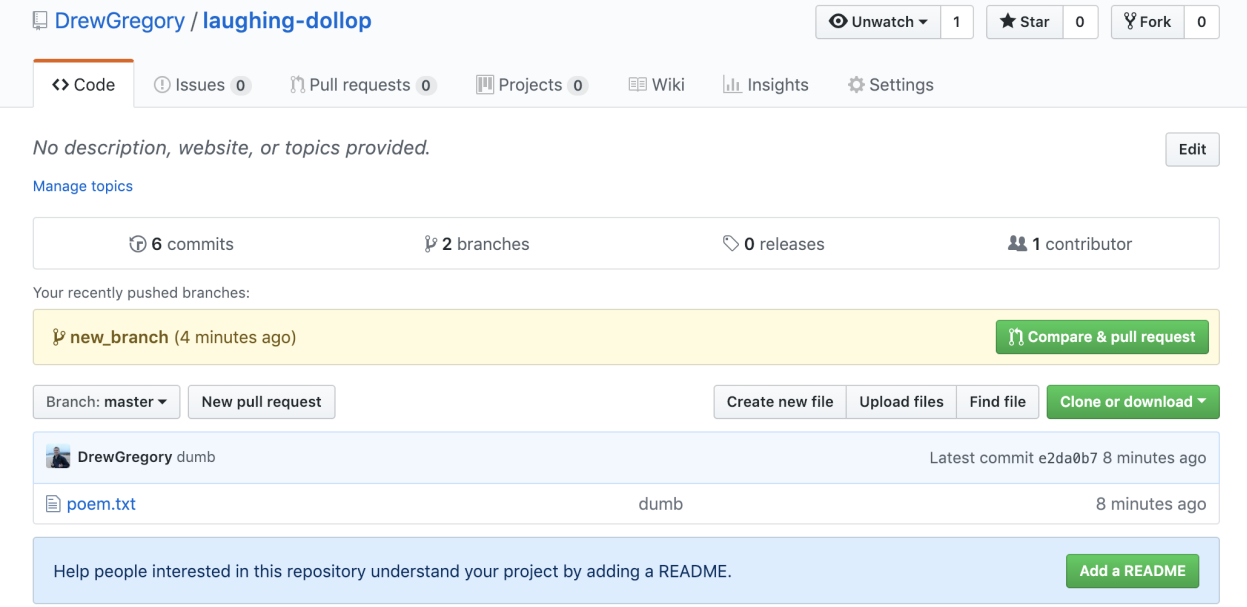
What if you want to checkout a branch on the remote repo that does not yet exist locally? Use `git fetch`:

```
$ git fetch
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (1/1), done.
remote: Total 3 (delta 1), reused 3 (delta 1), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/DrewGregory/laughing-dollop
* [new branch]      new_branch -> origin/new_branch
$ git checkout new_branch
Branch 'new_branch' set up to track remote branch 'new_branch' from 'origin'.
Switched to a new branch 'new_branch'
```

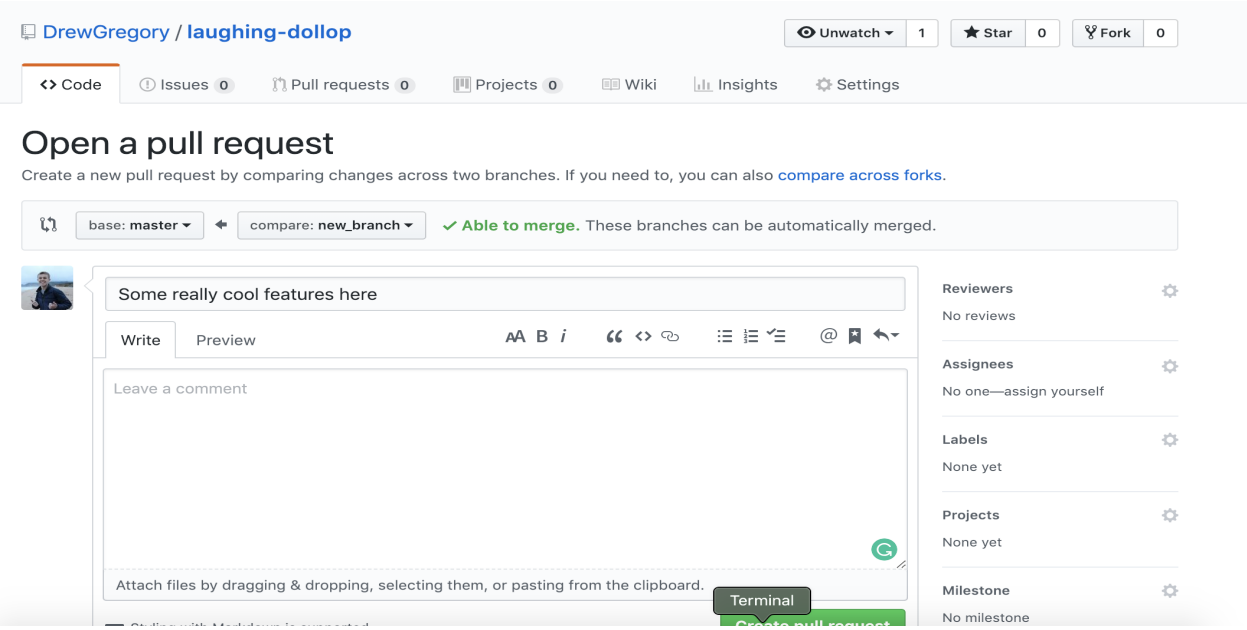
3.4.4 Pull Requests

First, let me say that Pull Requests have nothing to do with `git pull`. There. I said it. Now let's move on.

A great model for collaboration is to work on individual features (which means make a new branch from `master` or some other major branch) and then submita a pull request to merge the feature branch back into the major branch. GitHub even expects me to do so:



Now, you can elaborate on what your PR (other cool slang for Pull Request) accomplishes.



Ideally, other members will review your pull requests, give you feedback, and merge when finished.

Sometimes the branches will have merge conflicts. GitHub has an interface where you can manually resolve them in the same way we did locally. alternatively, you can merge the major branch (`master` in this case) into your feature

branch (`new_branch`) *before* submitting the PR, which will guarantee that there will be no conflicts on GitHub. Yes, this merging is the inversion of the Pull Request merge.

3.5 Fancy Tricks

3.5.1 Undoing Your Mistakes

To be frank: this post will explain undoing things in Git better than I can.

3.5.2 Signing Commits

Another link.

3.6 Licensing and Attribution

This work was initially created for a workshop at Stanford Code the Change.

CHAPTER

FOUR

PASSWORDS: CRACKING, HASHING, SALTING

4.1 Motivation

- https://www.theguardian.com/technology/2019/jan/17/breached-data-largest-collection-ever-seen-email-password-hacking
- https://www.consumerreports.org/digital-security/stolen-emails-and-passwords-surface-online/
- https://motherboard.vice.com/en_us/article/evegxw/collection-one-data-breach-password-hack-what-to-do

Unfortunately, websites get hacked quite frequently. This can leak the authentication data of millions of users. Because of this, one must take precautions regarding how we store this authentication data in the first place.

We will largely be using this phenomenally written article as a resource: https://crackstation.net/hashing-security.htm

This guide will serve as an interactive tutorial to see for yourself the effects of different representations of account passwords.

4.2 Storage

First, install pipenv if you have not done so already. Clone https://github.com/codethechange/password-storage-workshop.git, and then run `pipenv install` followed by `pipenv shell`. Next, generate a passwords file by running `python password-geneartor.py`. That should create a `passwords.json` file. This is a simulated dump of username-password pairs, consisting of a fair proprtion of the most popular passwords coupled with passwords comprised of concatenated dictionary words as well as random strings. Imagine as an adversary that this is a dump of passwords you acquired.

4.3 Plaintext

Alas, if you store your passwords in plaintext, it is automatically game over. If the wrong eyes reach this password dump, they can appear as any user on your platform. Let's try to avoid this.

4.4 Hashing

Run `python hash-passwords.py` to generate `hashed-passwords.json`. Imagine that this is the dump you, as an adversary, maliciously fetched from some poor web platform. To log in, the server hashes your provided password and see if the resulting hash matches the entry in the database corresponding to your username. This may seem secure out first glance, but we can easily concoct a lookup table. Run `python hash-dictionary.py` to generate a lookup table for all passwords that are dictionary words. Lookup tables cover many other variations of passwords, like common passwords in general and words with common substitutions. Next, run `python crack-passwords.py` to find passwords for many users! Also note that most lookup tables are too large to fit in memory, so rainbow tables are used instead for a time/space complexity tradeoff.

4.5 Salting

How could we prevent this lookup table? Well, we have to essentially require a unique lookup table for each password. We can produce a unique, random salt for each username-password pair.

Write this code in `hash-salt-passwords.py`:

```
import pandas as pd
from hashlib import sha256
import hashlib, binascii
import json
from random import choice
with open('passwords.json') as f:
    d = json.load(f)
from string import printable
chars = [c for c in printable if c.isalnum()]
for i in range(len(d['users'])):
    salt = ''.join([choice(chars) for _ in range(8)])
    m = sha256()
    to_hash = salt + '||' + d['passwords'][i]
    m.update(to_hash.encode("utf-8"))
    d['passwords'][i] = salt + '$' + m.hexdigest()
with open('salted-passwords.json', 'w') as f:
    json.dump(d,f)
```

Then we will have the passwords stored in a more secure dump: `salted-passwords.json`.

Now, try cracking this password with `crack-salts.py`:

```
# Load table
words = []
with open('dictionary.txt') as f:
    for line in f:
        line = str(line).replace('\n','')
        words.append(line)
# See if we can crack any passwords
import json
with open('salted-passwords.json') as f:
    d = json.load(f)
cracked_users = []
from hashlib import sha256
for i in range(len(d['users'])):
    salt, hash_val = tuple(d['passwords'][i].split('$'))
    if i % 10 == 0:
```

(continues on next page)

(continued from previous page)

```
        print(i)
    # Generate salted hash for each word
    for word in words:
        m = sha256()
        m.update(str(salt + word).encode('utf-8'))
        if m.hexdigest() == d['passwords'][i]: # the hash exists! We have found a␣
↪collision
            cracked_users.append((d['users'][i], word))
            break
print('Cracked ' + str(len(cracked_users)) + ' passwords!')
print(cracked_users[-10:])
```

Not that this takes significantly more time to crack than with no salt.

There are two quick improvements to our salting: a key derivation function, where we can control the computational difficulty of each resulting hash value, and a cryptographic PRG.

```
import pandas as pd
from hashlib import sha256
import hashlib, binascii
import json
from secrets import choice
with open('passwords.json') as f:
    d = json.load(f)
from string import printable
chars = [c for c in printable if c.isalnum()]
for i in range(len(d['users'])):
    salt = ''.join([choice(chars) for _ in range(8)])
    val = binascii.hexlify(hashlib.hmac('sha256', d['passwords'][i].encode('ascii'),␣
↪salt.encode('ascii'), 1000000))
    d['passwords'][i] = salt + '$' + val.decode('ascii')
    print(d['passwords'][i])
with open('salted-passwords.json', 'w') as f:
    json.dump(d,f)
```

With that being said, check out this article for more info about the preferred key derivation functions.

CHAPTER

FIVE

GUIDE TO SLACK NOTIFICATIONS

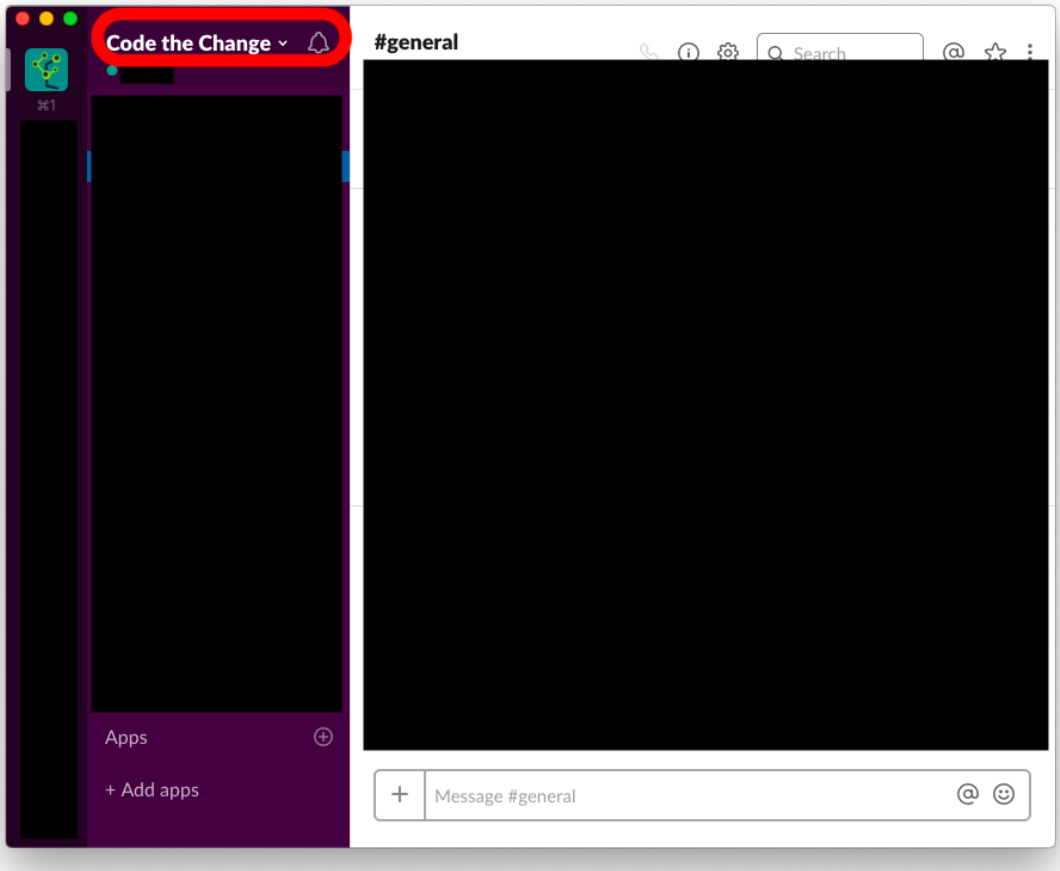
Slack notifications can be annoying to work with because Slack silences many notifications by default. This guide will walk you through how to change these settings.

5.1 Activating Notifications on Desktop

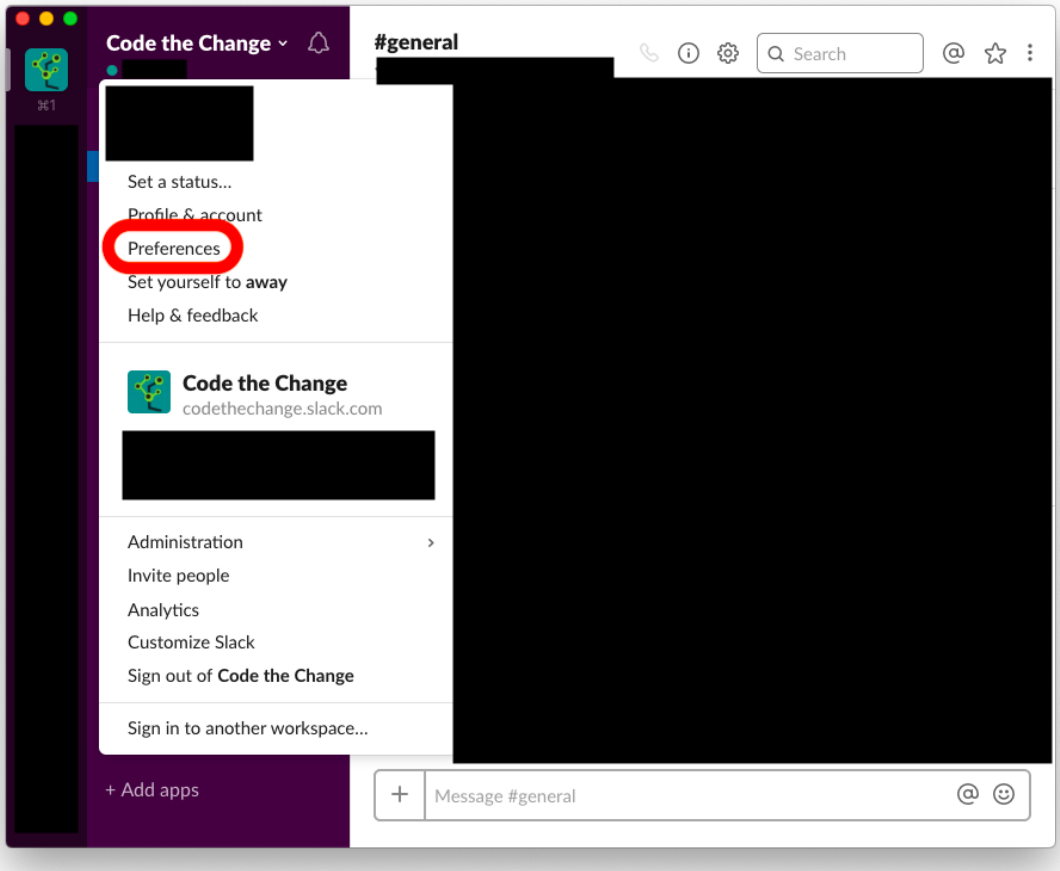
You will need to have the Slack app installed on your computer. You can download it here.

5.1.1 For MacOS

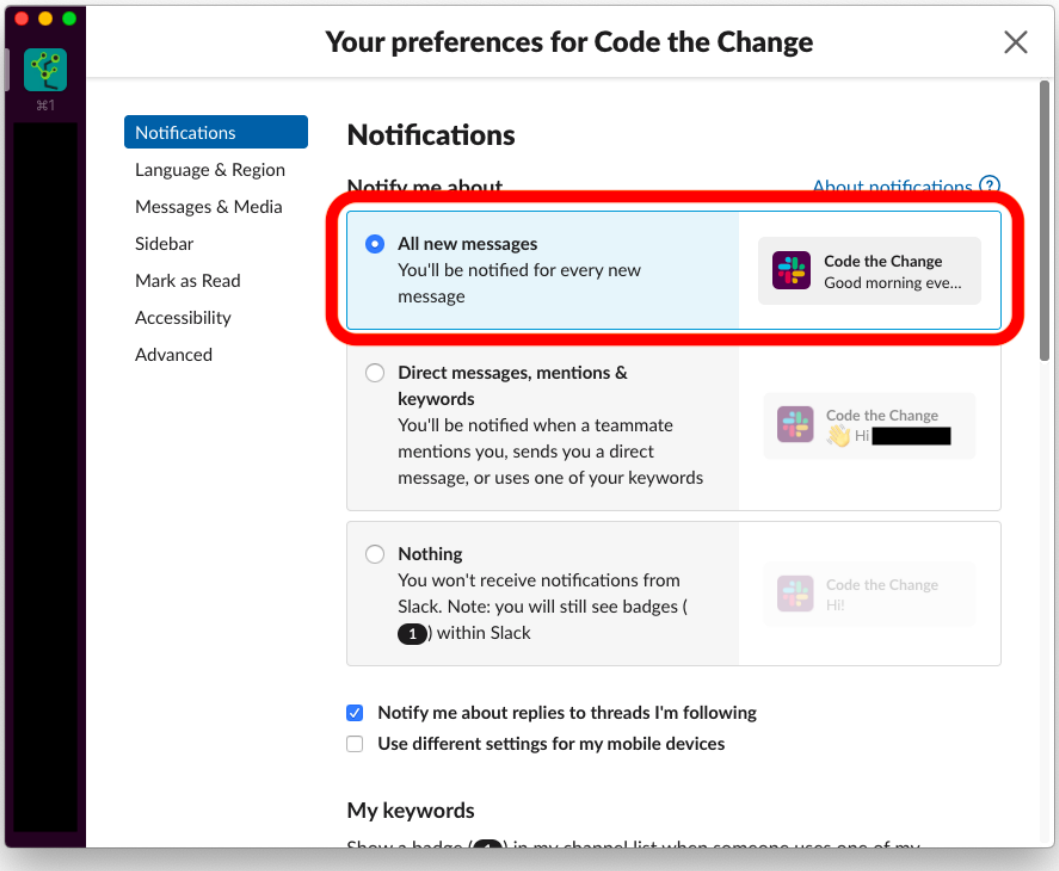
1. Open the Code the Change workspace from the left sidebar.
2. Select the title bar at the top of the left panel. The title should read `Code the Change`. The title is outlined in red in this image



3. In the drop-down menu that appears, select `Preferences`, outlined in red here



4. In the preferences window that appears, under `Notify me about`, choose `All new messages`, outlined in red here



5.1.2 For Windows and Linux

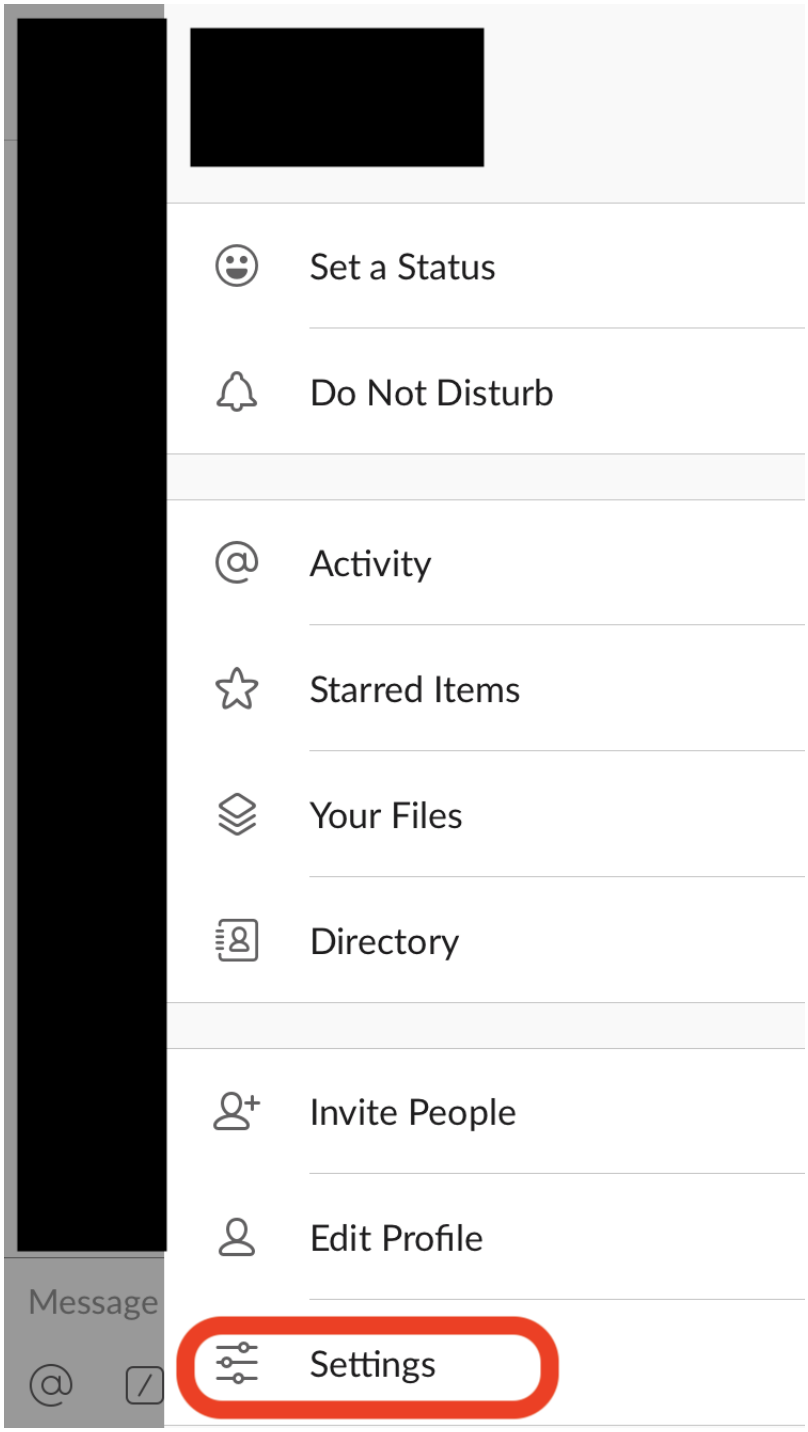
The instructions are likely the same as they are for MacOS. If not, see Slack's help page on notifications.

5.2 Activating Notifications on Mobile

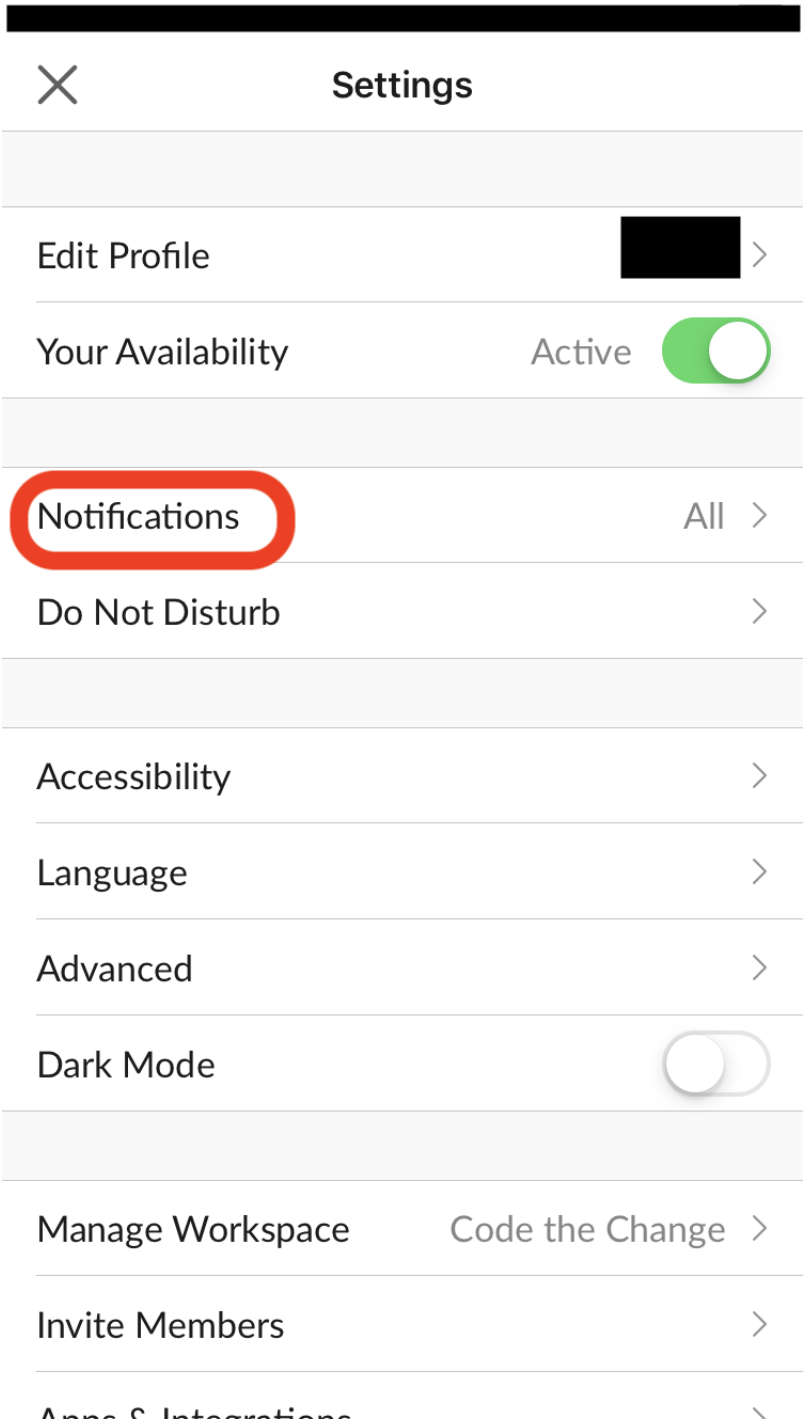
You will need to have the Slack app installed on your computer. You can download it from your phone's app store.

5.2.1 For iPhones

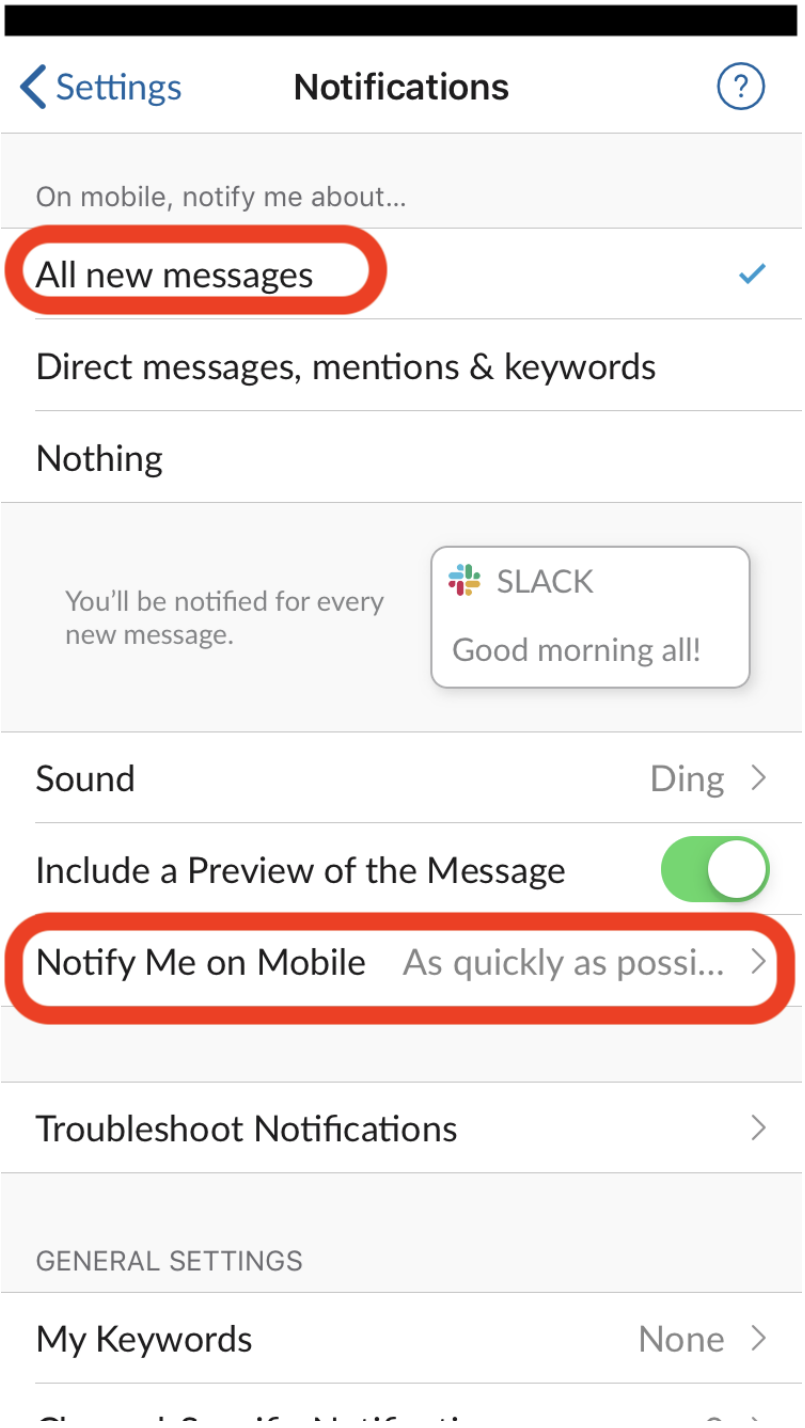
1. Open the Slack app and select the Code the Change workspace.
2. Swipe right to reveal the right sidebar.
3. Select `Settings`, outlined in red here



4. Select `Notifications`, outlined in red here



5. Choose to be notified for `All new messages` and to be notified on mobile `As quickly as possible`. These options are outlined in red here


Settings
Notifications
On mobile, notify me about...
All new messages
Direct messages, mentions & keywords
Nothing
You'll be notified for every new message.
SLACK
Good morning all!
Sound
Ding
Include a Preview of the Message
Notify Me on Mobile
As quickly as possi...
Troubleshoot Notifications
GENERAL SETTINGS
My Keywords
None

5.2.2 For Other Phones

The instructions are likely similar to those for iPhones. If not, see Slack's help page on notifications.

5.3 Further Reading

Much of this guide came from the Slack help page on notifications. Check it out for more details!

5.4 Licensing and Attribution

This work was initially created for a workshop at Stanford Code the Change.

CHAPTER

SIX

AN INTRODUCTION TO THE UNIX COMMAND LINE

6.1 Introduction

The original Unix system was developed by Bell Laboratores in the 1970s. It has since become the foundation of many operating systems, including Linux systems and macOS. Luckily for us, while all these operating systems may have very different graphical interfaces (GUIs), they share a common command-line interface (CLI). In this introduction, we will cover some of the most useful tools provided by this interface.

Note: With macOS Catalina, Apple switched the default terminal shell from bash to zsh. This guide focusses on bash, so you may need to run `bash -s` to get a bash shell.

Note: This guide is written for Mac users, but it should largely apply to any Unix system. I try and point out the parts that are Mac-specific.

6.1.1 Motivation

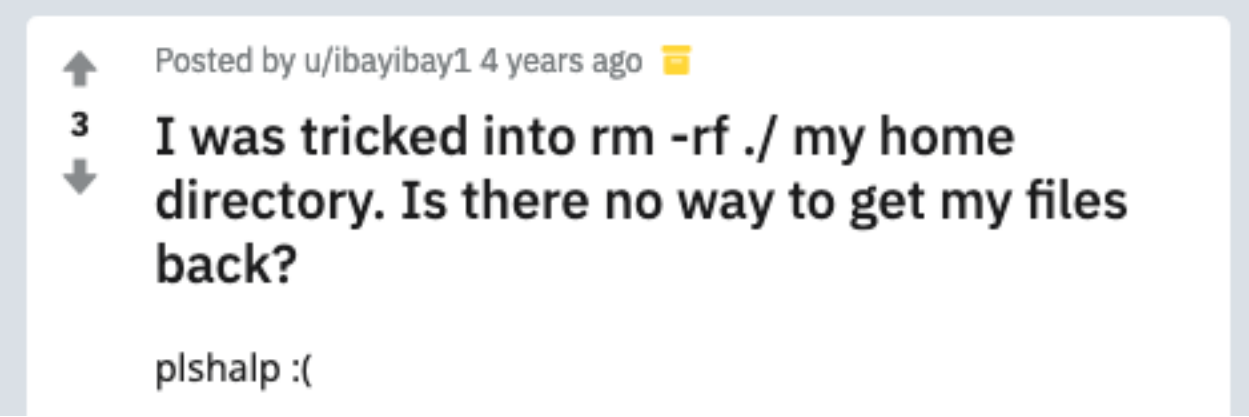
Once upon a time, the Unix command line was the only way to use a develop software. Now, however, there exist a plethora of integrated developer environments (IDEs) and graphical text editors. So why bother learning Unix commands at all? Here are a few of my reasons.

Unix is the Language of Coders

Even if *you* don't need to know the command line, you still will have to work with other people who live and breathe it. Take a moment to skim this page from the developer's guide for installing Oppia on a Mac. It is full of command-line instructions! Even if you can do all your work without the CLI, other people are going to assume you are fluent in Unix.

Knowledge Keeps you Safe

Even if you didn't understand the terminal commands in the Oppia installation instructions, you could always copy-and-paste them, right? Don't. **Never run commands you don't understand!** Otherwise you might end up like this poor soul who accidentally deleted their home folder:



Warning: `rm -rf <some folder>` is dangerous! It deletes the specified folder and all subfolders recursively. It also does so without asking any questions (because of the `-f`) regardless of files that are marked as not to be deleted. The deleted files are not moved to the trash, they are immediately deleted.

The CLI Helps Diagnose Bugs

I'm going to wax a bit philosophical for a moment, so bear with me. When you are trying to track down a bug, you want to assume that as little code as possible works. For example, if you have a function in a program that is buggy, are you going to debug by running the whole program or just the function? I'd hope you'd (barring unusual cases) choose to run just the function so that you don't have to worry about bugs elsewhere in your code. The same principle applies here. If you are debugging a problem with some software, you don't want to have to assume that a full GUI is working properly. Better to go with a command-line tool that has been used by so many programmers, it's about as bug-free as we are likely to get.

Sometimes graphical programs also hide important details. For example, some programs assume that files end with a newline character. A graphical text editor might not show a file as missing a trailing newline, but the command line will.

Overview

This introduction is centered around a program I wrote. We will work through downloading and using the program, just like you might when installing a project you want to contribute to. Along the way, we'll use command-line tools, and I'll explain what they do. By the end, you should:

1. Be able to navigate, modify, and search the file system.
2. Be able to use command-line editors to work with text files.
3. Be comfortable working with text files through the terminal.
4. Understand how executables work in Unix.
5. Know how to find more information about Unix commands.

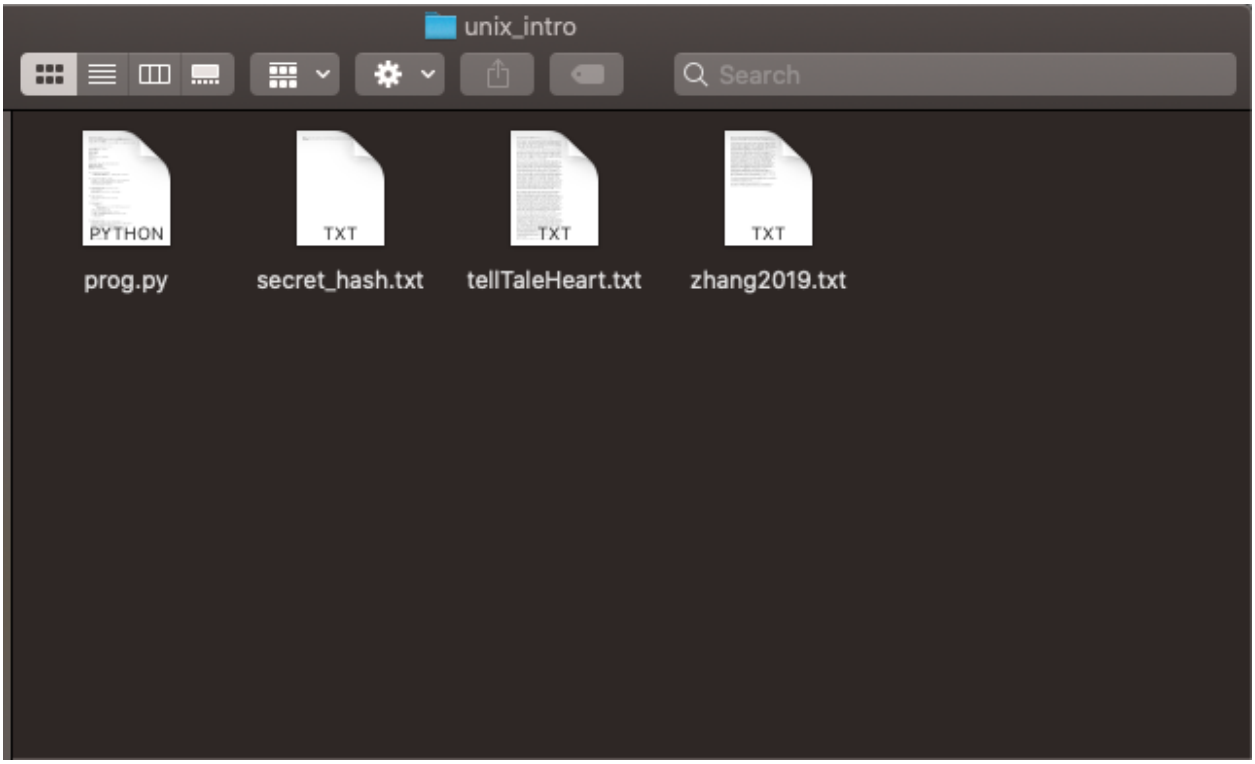
6. Have a basic understanding of how to work with command outputs.

Let's get started!

6.2 Getting Acquainted with the Unix File System

6.2.1 Moving Around

You are probably familiar with the files and folders of your computer looking something like this:



In the command line, we work with the same files and folders using Unix commands. Start by opening your terminal. On a Mac, launch the `Terminal` application from your applications folder. Then move to your home folder by running:

```
$ cd ~
```

Note that the `$` here stands for the command prompt your terminal shows when it's awaiting instructions. The tilde (`~`) is a short-hand for your home directory, and `cd` is a command that moves you to the path you give it as an argument. In this example `~` is the argument. A neat trick is that running `cd` without any arguments is the same as providing the tilde as the argument.

To see where you are in your file sysstem, you can print your *current working directory* like this:

```
$ pwd
/Users/<your_username>
```

Here, I've included a line without the command prompt. This is an example of the kind of output you might see from running the command that immediately precedes it. I will use this notation throughout this guide, and it is commonly used for Unix systems.

Now let's explore a little. `-` is a short-hand that tells `cd` to return to your previous working directory, and `..` refers to your parent directory. `/` refers to the root of your file system. On a Mac, this is usually called `Macintosh HD`. Here's how these work:

```
$ pwd
/Users/<your_username>
$ cd ..
$ pwd
/Users
$ cd /
$ pwd
/
$ cd -
$ pwd
/Users
```

Note that these paths might be different if you're not on a Mac. Now let's see what this journey looks like in the GUI. Open your home folder in your computer's file explorer (e.g. Finder on a Mac). If you right-click on the folder name in the top middle of the window, you can see the `Users` folder we saw from `pwd`. If you click it, that's just like running `cd ..`. If you open `Macintosh HD`, that's like running `cd /`. If you click the back button, that's like running `cd -` (only the first time! `cd` doesn't save history the way the back button does).

In the GUI, we constantly see the contents of our current directory in the Finder window. In the CLI, we can see the same contents like this:

```
$ cd ~
$ ls
Desktop  Downloads  ...
```

The `ls` command lists the contents of your current directory. We can see the equivalent GUI display by opening our current directory in the finder:

```
$ open .
```

`open` is a Mac command that opens files in the appropriate application, and `.` is a short-hand for your current directory.

6.2.2 Setting Up the Program

Now let's setup the program we will work with for the rest of this guide. Start by creating a folder somewhere on your filesystem to store this work. You now know how to move around in the terminal, so put it anywhere you like! Here's a useful trick: dragging a file or folder onto the Terminal window pastes into your command line the path to that file or folder.

```
$ cd /your/desired/path
$ mkdir unix_guide
```

The `mkdir` command creates a folder in your current directory with the name you specify. Now, we can download the program:

```
$ cd unix_guide
$ git clone https://github.com/U8NWXD/unix_intro.git
```

This isn't a guide about `git`, so I'm going to assume you know how it works. If not, it's also not that important here. You can also download the zip file from the GitHub page.

Quick sidenote here. The `tree` command is awesome, but it isn't installed by default on Macs. You can install it using the also awesome HomeBrew. If you install it, `tree` will show a hierarchy of files from your current directory.

6.2.3 Permissions and File Metadata

Permissions

Now that we have downloaded the code, let's run the program! Enter the folder that git downloaded and show the files present:

```
$ cd unix_intro
$ ls
prog.py           secret_hash.txt   tellTaleHeart.txt zhang2019.txt
```

There are a number of files here we won't use until later, but let's start with the program itself, `prog.py`. To run a program (called an executable) in Unix, all we have to do is type a path to it. However, to help us prevent running executables by accident, Unix requires that we inclue `./` for executables in our current working directory. Recall that `.` stands for your current directory. Let's run the executable like this:

```
$ ./prog.py
-bash: ./prog.py: Permission denied
```

Oops, we get a permission error! Each Unix file has permissions that specify who can read, edit, and execute it. To see these permissions, run `ls` with the `-l` flag, which tells `ls` to show additional information, including permissions, about each file.

```
$ ls -l
-rw-r--r-- 1 cs  staff   4052 Jan  8 09:26 prog.py
...
```

The `-rw-r--r--` part specifies the file's permissions. It's made up of four parts:

`-` This specifies the file type, for example whether the file is a directory.

`rw-` This indicates the permissions of the user that owns the file, which we can see from the output above is me (`cs`). You'll see your username there. The `rw` means we can read and write, but the `-` means we cannot execute.

`r--` This indicates the permissions assigned to the group that owns the file. They have only read permissions. In my case, the group `staff` owns the file, as you can see from the output above.

`r--` This indicates the permissions assigned to all other users. They can only read.

Unix uses octal to abbreviate these permissions. Each of the last three groups above can be thought of as sets of three bits. In binary, these bits are `(110)(100)(100)`, where I have used parentheses to separate the three sets of bits. Recall that to convert binary to octal, we can take each group of three bits, evaluate each as a decimal number, and then concatentate the resulting digits. In this case, this procedure yields the octal number 644, with each digit specifying the user (who owns the file), group (who owns the file), or world (everyone) permissions.

With that in mind, what permissions do we want the file to have so that we can execute it?

So let's make our program executable:

```
$ chmod [octal permissions number] prog.py
```

substituting in the permissions number you found above. Now if we view the permissions again, you should see an extra `x` indicating you have execution permissions:

```
$ ls -l
-rwxr--r--  1 cs  staff   4052 Jan  8 09:26 prog.py
```

Last Modified Time

Now we can run the program! First, let's use its help feature to see what it can do:

```
$ ./prog.py -h
usage: prog.py [-h] {edited,diff,pass,hash,secret} ...

A program to help you experiment with some UNIX commands

positional arguments:
  {edited,diff,pass,hash,secret}

optional arguments:
  -h, --help            show this help message and exit
```

There are a bunch of features here that we'll get to later, but let's start with the `edited` feature. It can tell us how long ago a file was modified. Let's create a file using the `touch` command and then see how long ago it was modified:

```
$ touch foobar
$ ./prog.py edited foobar
0:00:03.692538
This program is part of unix_intro (https://github.com/U8NWXD/unix_intro)
```

Great! We modified our file (by creating it) about 3 seconds ago. Now let's try some other ways of editing the file:

- Using `touch`: `touch foobar`.
- With a text editor: Open the vim editor with `vim foobar`, then type `i` to enter insert mode. Make some changes (navigate with arrow keys), and then leave insert mode by pressing escape. Then, save and exit by typing `:wq`.
- Renaming it: `mv foobar foo` will "move" the file `foobar` to the file `foo`, which effectively renames it.

Shebangs

But how does the terminal know how to execute the program? It has the `.py` extension, but it could still be python 2 or 3. In fact, the file extension is irrelevant. Instead, the terminal relies on a *shebang*, which is a comment at the top of the file. Take a look using `head`, which shows the first few lines of a file:

```
$ head -n 2 prog.py
#!/usr/bin/env python
# This file is part of unix_intro (github.com/U8NWXD/unix_intro),
```

See that `#!/usr/bin/env python`? That tells the terminal to use whatever executable is run when you type `python` into the terminal.

As a quick aside, `head` has a complementary command called `tail` which does the same thing, only from the end of the file.

6.3 Looking Up Commands

We can also see this metadata ourselves. Remember the `ls -l` command? We can lookup what the rest of its output means in its manual page:

```
$ man ls
```

Look for the `-l` entry and read where the modification timestamp is shown.

Note: The `man` command is invaluable. If you need to figure out what a command does, it should be your first stop. Online documentation is great, but sometimes commands have slightly different syntax on different operating systems. The manual pages from `man` should always be accurate.

We can also see where the `ls` executable is on the system:

```
$ which ls
/bin/ls
```

This is super helpful when you want to know which version of a command you are running. For example, try:

```
$ which python
```

This will tell you which python installation you are running now. If you haven't already discovered, there can be many python installations on a single system!

6.4 Comparing Files

Let's move on to another of the program's features, `diff`, which checks whether two files are identical. Let's try it out by making a copy of Edgar Allen Poe's *Tell Tale Heart*:

```
$ cp tellTaleHeart.txt copy.txt
```

As you may have guessed from this example, `cp` copies the file at the first argument to the second argument. Now, let's see if they are the same:

```
$ ./prog.py diff tellTaleHeart.txt copy.txt
True
This program is part of unix_intro (https://github.com/U8NWXD/unix_intro)
```

We get `True`, indicating that the files are the same, as we'd expect. Now go ahead and edit `copy.txt` in vim and try again:

```
$ ./prog.py diff tellTaleHeart.txt copy.txt
False
This program is part of unix_intro (https://github.com/U8NWXD/unix_intro)
```

Now they're different! We can find the difference using `diff`:

```
$ diff tellTaleHeart.txt copy.txt
```

The output identifies the lines that are different between the two files. If you've used `git diff` before, this is similar.

6.5 Environment Variables

Notice the pesky line that the program always prints out telling us it is part of `unix_intro`? There is a way to hide it, but instead of using an argument, it uses another common way to configure Unix programs: environment variables.

Environment variables are variables accessible to every program running in your terminal. You can see all your current variables with `printenv`:

```
$ printenv
SHELL=/bin/bash
PWD=<your current directory>
LANG=en_US.UTF-8
OLDPWD=<your previous directory>
...
```

Notice that `OLDPWD` shows your previous directory and `PWD` your current one. This is how `cd -` and `pwd` work!

To disable that annoying line (I'll call it an epilog), we need to set an environment variable called `UNIX_INTRO_DISABLE_EPILOG` to 1 (which will mean true for Python). Set it like this:

```
$ export UNIX_INTRO_DISABLE_EPILOG=1
$ ./prog.py diff tellTaleHeart.txt copy.txt
False
```

See, no epilog!

Let's say we want to avoid typing that long variable name each time. We can save it to a file and then load that file instead. To do this, open a new file in vim called `environ`:

```
$ vim environ
```

and write in it `export UNIX_INTRO_DISABLE_EPILOG=1`. Now, you can set the variable just by using the `source` command, which loads the contents of a file into your terminal:

```
$ source environ
```

Go ahead and try it in a new terminal window!

6.6 Handling Large Command Outputs

Our program can also generate passwords by selecting random words (obligatory xkcd reference). Try it out:

```
$ ./prog.py pass 3
Cainian
urceolate
neighboress
```

Of course your words will be different (it's random!). Now let's make a really strong password:

```
$ ./prog.py pass 100
teetotalism
semiprimigenous
rhyacolite
...
```

That's a lot of output! Let's use `less` to make the output scrollable and avoid cluttering up our terminal history:

```
$ ./prog.py pass 100 | less
```

Notice we used a fancy new operator here, the pipe (`|`) operator. It takes the output from the left-hand command and sends it ("pipes it") to the right-hand command. This is very useful for stringing commands together.

We can also send our password to a file using >, another cool operator:

```
$ ./prog.py pass 100 > password.txt
```

Go ahead and take a look at `password.txt` in vim. Now let's add another 100 words. If we use > again, we'll overwrite the file, but if we use >>, we can append to it:

```
$ ./prog.py pass 100 >> password.txt
```

Now that we have a lot of words, we can do some other cool things to them. First, you need to know that the `cat` command prints out the contents of a file. With that, we can sort them:

```
$ cat password.txt | sort | head
```

and we can remove any duplicates:

```
$ cat password.txt | sort | uniq | wc -l
```

Note: The *uniq* command cnly compares adjacent lines, so you have to sort the file first!

Here I used the `wc` command, which counts. With `-l`, it counts the number of lines it is provided. Were there any duplicated words?

6.7 Deleting Files

Warning: Be careful deleting files from the command line! There's no trash, so once you run the command, your files are gone.

Remember that story at the top about the poor fellow who erased their home directory? Now we can learn what that command did. We delete files using `rm`. For example, delete the password file:

```
$ rm password.txt
```

Now the fellow from earlier ran `rm -rf ~`. The `~` means the home directory, but `rm` won't delete directories so easily. The `-r` flag tells `rm` to delete the specified folder and any subfolders recursively, removing the entire file tree. Lastly, there are some files that are protected from being deleted (git makes a few of these). `-f` tells `rm` to ignore those protections and delete anyway.

6.8 Searching

Now maybe you noticed that the program has one other feature, called `secret`. Let's run it:

```
$ ./prog.py secret
There is a file in /usr/ called 'words'.
Count the words in that file that have the 'not' prefix.
Assume the prefix takes the forms: il-, ir-, im-, in-.
Password:
```

First off, you for some reason haven't been returned to the command prompt! That's because the program is waiting for you to enter the password. Since we don't know it yet, quit the program by holding the control and `c` keys together.

Note: Using ctrl-C to kill running programs is very useful when things go wrong. Don't forget it!

Now, our first task is to find a file in `/usr/` called `words`. If we look in `/usr/`, it doesn't look like `words` is going to be easily found:

```
$ ls /usr
X11         X11R6       bin         lib         libexec     local
sbin        share       standalone
```

Instead, let's use the `find` command:

```
$ find /usr -name words
find: /usr/sbin/authserver: Permission denied
/usr/share/dict/words
```

There it is! (We can ignore the permission error.) `find` is actually quite powerful, but I'll let you explore it's abilities on your own.

Now we need to find all the words starting with `im`, `il`, `in`, or `ir`. This is a great job for a regular expression. I'll leave explaining the details of regular expressions to this Digital Ocean post but I'll tell you that we want to find all words that match `^i[lmnr]`. The `^` represents the start of the word, and `[lmnr]` stands for any of the 4 letters. We can use `grep` to filter out only the words that match this regular expression:

```
$ cat /usr/share/dict/words | grep "^i[lmnr]" | wc -l
```

Now we can run the program again and enter the password. Notice that when you enter the password, your typing is not shown. This is a security feature.

As a side note, this is also more secure than passwing a password in as an argument, because any other program you're running and any administrator can see the arguments to any running program.

That's an odd message. Let's see if we can't find out what it means by searching for that last unusual word in the `unix_intro` folder. It turns out that with the `-r` flag to search files recursively and the `-i` flag to ignore case, we can do this with `grep`:

```
$ grep -ri <the unusual word> .
```

In the results, you should see a bunch of results from `zhang2019.txt`. Go ahead and take a look at this file:

```
$ cat zhang2019.txt | less
```

Believe it or not, this abstract was published on bioRxiv, which biologists use to share their papers before official publication. That this nonesense got in caused quite a storm among the academic Twitter community! You can see the abstract on bioRxiv here.

6.8.1 Hidden Files

Lastly, you might be wondering where the secret message was hiding. We can find it using the `-a` flag that shows hidden files:

```
$ ls -a
.                 .git            .secret           prog.py
tellTaleHeart.txt ..              .gitignore        secret_hash.txt
zhang2019.txt
```

Notice `.secret`, which is where the message is! The `.` at the beginning causes it to be hidden most of the time.

6.9 Conclusion

I hope you now are more comfortable using the Unix command line and figuring out how to use new commands. If you're interested in learning more, try looking up these:

- Admin Rights: `su`, `sudo`
- Downloading Files: `curl`, `wget`
- Archives: `tar`, `gzip`, `unzip`
- Process Management: `ps`, `top`, `kill`, `killall`
- Jobs
- String Manipulation: `awk`, `sed`
- Cryptography: `gpg`
- Clock: `date`

6.10 Licensing and Attribution

This work was initially created for a workshop at Stanford Code the Change.

CHAPTER

SEVEN

INTRODUCTION TO DATABASE AND STRUCTURED QUERY LANGUAGE (SQL)

7.1 Introduction

A database serves as a platform to store and manage data. Database systems provide an environment for storage and retrieval of both structured and semi-structured data. Structuring the database tailored to one's needs is as important as designing a visually appealing and easily navigate-able UI/UX front-end. A good database would have advantages in terms of scale, speed, stability, evolution, reliability, cost efficiency, etc.

The objective of this guide is to provide an overview of relational databases, non-relational databases, some of the prominent languages used to query the databases, and a deep dive into Structured Query Language (SQL), which is a standard language for relational databases.

7.2 Relational Database

A relational database refers to any database that follows the relational model provided by traditional relational database management systems. What is meant by *relational* is that each object in the database is linked to some other object in certain ways.

One keyword that is associated with relational databases is **table**. Relational databases store data in **tables** and rows, built on the concepts of relational algebra.

Relational databases are perhaps the most straightforward way of representing the data. Adopting this scheme, each row in the table is a record with a unique ID called the *key*, while the columns of the table hold attributes of the data.

7.2.1 Advantage of Relational Databases

Relational databases have a number of advantages, which include but are not limited to the following:

1. It is easy to remove redundant data through a process called **normalization**. The normalization process entails the organization of the columns (also called as attributes) and tables (also called as relations) of a database to ensure that their dependencies are properly enforced by database integrity constraints.

2. Relational databases provide the ability to deal with larger datasets, as long as data are structured coherently and they abide by conventions.

3. Relational databases offer almost infinite number of possible permutations to slice and manipulate the data, based on one's needs, which turns out to be a huge advantage in data analysis.

4. Relational databases have reputations as a standard and mature tool for the design, collection, and storage of data, first introduced and widely utilized since the 1970s.

7.2.2 Sidenote: Types of Relational Databases

Examples of relational databases are: MySQL, PostgreSQL, SQLite3, etc.

Question: Is there a need for all these different relational databases?

Answer: There is! These database systems are designed to meet the needs of a particular task or a certain industry. Although the underlying scheme is relational, their specifics do differ.

7.2.3 When to Use Relational Databases

Oracle provides the following guideline in the section *What to Look for When Selecting a Relational Database*, duplicated below with some degree of abridgement:

Several factors can guide your decision when choosing among database types and relational database products. Ask yourself the following questions.

- What are our data accuracy requirements? [Will data storage and accuracy

rely on business logic? Does our data have stringent requirements for accuracy?]

- Do we need scalability? [What is the scale of the data to be managed,

and what is its anticipated growth?]

- How important is concurrency? [Will multiple users and applications need

simultaneous data access? Does the database software support concurrency while protecting the data?]

- What are our performance and reliability needs? [Do we need a high-performance,

high-reliability product? What are the requirements for query-response performance?]

7.3 Non-relational Database

A non-relational database refers to any any database that does not follow the relational model provided by traditional relational database management systems.

In non-relational databases, you would not expect to see a table. They instead utilize a storage model optimized for the specific requirements of the types of data being stored. Very commonly, data are stored as simple key/value pairs, or even more often than that, data are stored as JSON documents.

It is worthwhile noting the term NoSQL, which refers to database systems that do not use SQL for queries (hence NoSQL), but uses other programming languages to achieve this purpose. For instance, MongoDB uses Javascript, which has a distinguished advantage in parsing and handling JSON data. Many non-relational databases also use Python as their querying language.

7.3.1 Advantage of Non-relational Databases

Relational databases have a number of advantages, which include but are not limited to the following:

1. Non-relational databases are better in storing and processing large amounts of unstructured data. Unlike relational databases, it is not obligatory to relate an object to other ones. Thus, NoSQL databases are more flexible.

2. As a corollary to the first advantage, the fact that non-relational databases contain data in an unstructured manner makes the iteration and code pushes very fast.

3. Non-relational databases use object-oriented programming paradigm, which is easy to use and is widely applicable.

7.3.2 Sidenote: Types of Non-relational Databases

Examples of relational databases are: MongoDB, DynamoDB, etc.

MongoDB stores data as a JSON file; DynamoDB is a AWS product, usually lauded for its built-in security and in-memory caching for faster interactions. Major companies tend to use DynamoDB a lot.

7.4 Structured Query Language (SQL)

Our main purpose is to explore SQL, a standard language for relational database management systems, which include Oracle, Sybase, Microsoft SQL Server, Ingres, etc. SQL is a relatively straightforward language, and it is built on relational logic, which makes it extra useful and easier to learn.

RDBMS, which stands for Relational Database Management System, is the underlying form of SQL, and as we have seen above, data in an RDBMS are stored in objects called **tables**, which might look as follows:

```
+----+----------+-----+-------------+-------------------+
| ID | NAME     | AGE | DEPARTMENT  | FAVORITE_NUMBER   |
+----+----------+-----+-------------+-------------------+
|  1 | Keith    |  32 | CS          |  137              |
|  2 | Chris    |  54 | CS/Ling     |  1                |
+----+----------+-----+-------------+-------------------+
```

This table can have a name! We can call it `PROFESSORS`. (It is a convention to name the table with uppercase letters only.)

A table has smaller entities called **fields**. `PROFESSORS` table consists of ID, NAME, AGE, DEPARTMENT, and FAVORITE NUMBER fields.

Before getting into the syntax of SQL, we need to know a special value that can appear in a table: NULL. NULL signifies that the value/entry in the field is blank or NaN. Note that `NULL != 0` or `NULL != (empty string)`.

7.4.1 Syntax of SQL

CREATE statement

`CREATE` allows us to create a new table in the database.

```
CREATE TABLE professors (
    id INTEGER,
    name TEXT,
    age INT,
    department TEXT,
    fav_num INT
);
```

SELECT statement

`SELECT` allows us to fetch data from a database. You can either select column(s) from a particular table, or can select all columns from a table.

```
SELECT column1, column2, ..., columnn FROM professors;
SELECT * FROM professors;
```

INSERT statement

`INSERT` allows us to insert a new **row** into a table.

```
INSERT INTO PROFESSORS (id, name, age, department, fav_num) VALUES (3, MTL, 61, Dean,␣
↪100);
```

ALTER statement

`ALTER` allows us to add a new column to a table.

```
ALTER TABLE PROFESSORS ADD COLUMN fav_book TEXT;
```

UPDATE statement

`UPDATE` allows us to edit/change a row in a table.

```
UPDATE PROFESSORS SET fav_book = 'C++ Programming Language Guide' WHERE name = 'Keith
↪';
```

DELETE statement

`DELETE` (or more specifically, `DELETE FROM`) allows us to delete one or more rows from a table.

```
DELETE FROM PROFESSORS WHERE favorite_book IS NULL;
```

WHERE keyword

`WHERE` allows us to restrict our query results to a certain condition

```
SELECT * FROM PROFESSORS WHERE fav_num > 100;
```

The common comparison operators (=, !=, >, <, >=, <=) are used in SQL.

AND/OR keywords

`AND` operator allows us to combine multiple conditions where both conditions must be met, used in tandem with ``WHERE` keywords.

```
SELECT * FROM PROFESSORS WHERE fav_num > 100 AND age BETWEEN 20 AND 60;
```

Likewise, `OR` operator allows us to combine multiple conditions where at least one of the conditions must be met, used in tandem with `WHERE` keywords.

```
SELECT * FROM PROFESSORS WHERE fav_num > 100 OR age BETWWEEN 20 AND 60;
```

ORDER BY keyword

`ORDER BY` allows us to sort the results, either alphabetically or numerically.

```
SELECT * FROM PROFESSORS ORDER BY name;
SELECT * FROM PROFESSORS ORDER BY name DESC;
```

LIMIT keyword

`LIMIT` allows us to specify the maximum number of rows the result will have.

```
SELECT * FROM PROFESSORS LIMIT 10;
```

COUNT function

`COUNT()` function allows us to count the number of **non-empty** values in a column. The input is the name of a column.

```
SELECT COUNT(*) FROM PROFESSORS;
```

SUM function

`SUM()` function allows us to return the sum of all the values in the specified column.

```
SELECT SUM(AGE) FROM PROFESSORS;
```

MAX/MIN functions

`MAX()` function allows us to return the largest value in the specified column, and `MIN()` the smallest value.

```
SELECT MAX(AGE) FROM PROFESSORS;
SELECT MIN(AGE) FROM PROFESSORS;
```

AVG functions

`AVERAGE()` function allows us to return the average of the values in a specified column.

```
SELECT AVG(AGE) FROM PROFESSORS;
```

GROUP BY statement

`GROUP BY` statement allows us to arrange identical data into **groups**, used in tandem with `SELECT`.

```
SELECT age, COUNT(*) FROM PROFESSORS GROUP BY age;

// identical to, for instance:

SELECT COUNT(*) FROM PROFESSORS WHERE age = 20;
SELECT COUNT(*) FROM PROFESSORS WHERE age = 40;
SELECT COUNT(*) FROM PROFESSORS WHERE age = 60;
```

DISTINCT keyword

`DISTINCT` keyword allows us to filter duplicate values and return rows of specified column.

```
SELECT DISTINCT fav_book FROM PROFESSORS;
```

INNER JOIN keyword

`INNER JOIN` allows us to select records that have matching value in **two or more** tables. Assume that there is another table called `CSFACULTY` that contains faculty members in the Computer Science Department.

```
SELECT name FROM PROFESSORS INNER JOIN CSFACULTY ON PROFESSORS.name = CSFACULTY.name;
```

7.5 Licensing and Attribution

This work was initially created for a workshop at Stanford Code the Change.

CHAPTER

EIGHT

ZERO KNOWLEDGE WHAT? AN INTRODUCTION TO ZERO KNOWLEDGE

8.1 An Introduction to Zero Knowledge Proofs

One of my instructors shared that he would have a funny daily conversation with his dad:

- Dad: "How was school today?"
- Middle Schooler: "Fine."

Understandably, a father might be frustrated with the same response each day. Why? One could argue that this father might not be learning much about his son's day. Indeed, this idea captures a fundamentally important idea in complexity theory and cryptography: zero-knowledge proofs.

8.1.1 Proof Systems

First, let's define how proofs work in computer science. We will define two parties: a *prover* P and a *verifier* V. P and V can communicate and send messages to each other. V can output "accept" if V is convinced by P and "reject" otherwise. V can only run in polynomial in the length of the statement in question. P and V can also use randomness. Here are some example statements a prover may try to prove to the verifier:

- Is a boolean formula satisfiable?
- Can a graph's nodes be assigned three different colors such that no two adjacent nodes are the same color?
- Given a set of integers, is there a subset whose sum is zero?
- Given a group $\mathbb{G}$ with a generator $g \in \mathbb{G}$, what is the discrete logarithm of g^x for some $x \in \mathbb{G}$?

Note: The first three examples are all NP-Complete problems, so it is widely believed that V will not be able to determine the answer in polynomial time themselves. This is why P is needed in the first place.

Note: This fourth problem is a different form than the previous three. In fact, it is a *search problem* ("what is the answer") as opposed to a decision problem ("is this true or false"). This problem doesn't have as strong computational hardness as the previous three but is also assumed to be hard for classical (non-quantum) computers. If you have a way to solve this problem efficiently for elliptic curve groups with classical computers, contact your nearest cryptographer immediately.

Note: For budding complexity theorists, this definition of a proof system loosely encapulates the complexity class IP.

8.1.2 Definining Zero Knowledge

In what sense can a proof system be zero knowledge? At first glance, proving information without giving away information sounds like a paradox. Let's say in English that a zero knowledge proof should prove that a statement is true **without giving away information beyond the fact that the statement is true**.

Let's ground this idea with some examples:

- Prove that a graph is 3-colorable without giving away any information about how to color it.
- Prove that a boolean formula has a satisfying assignment without giving away anything about the assignment.
- Prove that some cryptocurrency was given to you without giving away anything about who previously owned that token.
- Prove that you know the discrete logarithm of g^x without giving away anything about x.

Note: Similarly to our decision versus search problem discussion, the first three proofs are proofs of the truth of a statement. The last proof isn't about whether a statement is true but instead is defined as a "zero-knowledge proof of knowledge."

8.1.3 A Cool Application

Let's see which of your friends know you super well: Who knows your birthday?

Now, imagine that you were with a group of friends who can all listen in. Some of your friends claim that they know the answer. You would like to allow these arithmetic pros to prove to you that they know the answer without giving away the answer to your other friends who are listening. Even more, you want to allow your friends that *know* your birthday to convince your friends that *don't* know your birthday that they know your birthday without giving away your birthday.

The crazy thing: this can be done!

8.1.4 Schnorr's Sigma Protocol

First, let's encode your birthday as an integer: MMDDYYYY. For example, if you birthday is Jan 10, 1938, then your encoding would be 01101938. Choose some super large cyclic group $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ where $p = 2q + 1$ for some *huge* primes q and p (let's say around 2^{2048}). Since our P and V are humans, let $p = 2000000579$.

Note: If you want to know how I picked p, check out this page. This careful choice of p is to make the discrete logarithm for this group difficult.

Choose any $g \neq 0, 1 \in \mathbb{Z}_p$ as our generator for $\mathbb{Z}_p$. Next, let x the encoding of our birthday. Publish $h = g^x$ to your friends. We assume it is hard for them to be able to find x given g^x beyond brute forcing the solution. If your appearance doesn't give away your age, let's assume that there are about $36,500$ possibilities and that a human thus would really struggle to brute force guess the answer.

Note: In this toy example, brute forcing the answer (guessing a's and cheking if $g^a = g^x$) via a computer is easy. Most computer science problems have a much larger output space such that brute forcing a solution would be infeasible. As such, our goal will be to make brute forcing be a forgetful friend's best strategy to figure out the answer.

In this case, your friend is the prover, and you are the verifier. In fact, even your friends who **don't** know the answer could be the verifier themselves! Here is the protocol:

$P(x, h = g^x)$		$V(h = g^x)$
$r \rightarrow^R \mathbb{Z}_{\|}$		
$u \leftarrow g^r$	$\rightarrow u$	
	$c \leftarrow$	$c \rightarrow^R \mathbb{Z}_{\|}$
$z \leftarrow r + cx$	$\rightarrow z$	
		"Accept" if $g^z == u * h^c$

8.1.5 How to Try It Using the Python Interpreter

Note: This tutorial is an instructive toy example. This code should not be used for any real application. For example, this code does not protect against side channel attacks and stores private values in terminal memory.

Pick some friends and try this out! You can easily do all this arithmetic using the Python Interpreter, which we will demonstrate below.

For the person who wants to poll who knows their birthday, they should compute their public value as follows (without anyone looking at their screen):

```
>>> g = 5 # public parameter
>>> p = 2000000579 # public parameter
>>> x = 1101938 # Don't let anyone peek! Jan 10, 1938
>>> h = pow(g,x,p) # more efficient than (g**x) % p
1880666247
```

For the friend P that wants to prove that they know your birthday, have them begin with this code (do not let anyone take a peek at your screen!).

```
>>> from random import SystemRandom
>>> gen = SystemRandom()
>>> g = 5 # public parameter
>>> p = 2000000579 # public parameter
>>> r = gen.randrange(p)
>>> u = pow(g,r,p)
>>> print(u) # send to V
1706406692
```

Send u to the verifier friend V. V should then run this code on their computer.

```
>>> from random import SystemRandom
>>> gen = SystemRandom()
>>> g = 5 # public parameter
>>> p = 2000000579 # public parameter
>>> u = 1706406692 # From prover, let's keep this for later
>>> c = gen.randrange(p)
>>> print(c) # send to P
107041050
```

Next, P should receive c and run this code:

```
>>> c = 107041050 # from V
>>> z = r+c*x
>>> print(z) # Send to V
1181831844243911
```

Finally, V should perform this final check:

```
>>> h = 1880666247 # from person who's birthday I should know but don't.
>>> z = 1181831844243911 # from prover
>>> print("They know that person's birthday" if pow(g,z,p) == (u * pow(h,c,p)) % p␣
↪else "They're lying! They don't know their birthday")
They know that person's birthday
```

8.1.6 Zero Knowledge Formalisms

This is a more formal treatment of a zero knowledge proof. Feel free to skip this section. Let (P, V) be an interactive proof system for a language L (a "language" is a complexity theory formalism. Essentially, treat $x \in L$ as "x is true"). This proof system is *zero-knowledge* if it satisfies the following properties $\forall x$:

- ***Completeness.*** $x \in L \implies Pr_{\textbf{P,V's randomness}}[V \textbf{ accepts running}(P, V)] = 1$ In English, this means that an honest prover and verifier should cause the verifier to accept if x is true.
- ***Soundness.*** $\forall P^*(x \notin L \implies Pr_{\textbf{P,V's randomness}}[V \textbf{ accepts } (P^*, V)]) < \frac{1}{3}$**.** In English, this means that malicious provers should only be able to trick an honest verifier that a statement is true with low probability.
- ***Perfect Zero Knowledge.*** $\forall V^* \exists$ **efficient S such that** $\forall x \in L : \{Sim(y)\} \approx \{View_{V^*}((P, V^*)(x))\}$ In English, this means an efficient algorithm should be able to generate the transcript of the protocol without even knowing whether x is true or not. If the transcript can be generated without the knowledge of the answer, then the transcript must give away zero knowledge! In order to preserve completeness and soundness, the resulting proof system **must** be randomized.

There are many variants of this definition of zero knowledge:

- ***Honest-Verifier ZK*** only requires the simulator to produce identical distributions for the honest verifier V for the protocol. In other words, the verifier is expected to behave honestly, but malicious verifiers may be able to learn information by deviating from the protocol.
- ***Statistical ZK*** relaxes the requirement that the transcript/view distributions are identical and instead makes the similarity function allow for some negligible difference (as a function of some security parameter) in probability for any given state in the distribution.
- ***Computational ZK*** requires that all polynomially bounded algorithms should be unable to distinguish the transcript/view distributions (relying on some security assumptions).

8.1.7 An Aside About Completeness and Soundness for Schnorr's Protocol

Completeness and soundness properties are what define an interactive proof system, so I will briefly explain the analysis for Schnorr's protocol. Note that if P is honest, then $g^z = g^{r+cx} = g^r g^{cx} = u(g^x)^c = u * h^c$, satisfying completness. This part is not vital fo the idea of zero-knowledge, so you can skip this section.

Soundness is somewhat trickier. In fact, our ZK formalisms don't quite apply here, as those definitions were for decision problems, not search problems. Instead, we could say that soundness means "a poly-time adversary without x should only be able to produce z where $g^z = u * h^c$ with negligible probability." We could prove this definition of soundness by using an adversary that would break soundness for Schnorr's Protocol to create an adversary that would break some widely held security assumption, such as the Decisional Diffie Hellman assumption or discrete log assumption. This proof format is a security reduction and is widely used in cryptgraphy.

Similarly, one could define a variant of soundness for Proof of Knowledge systems. Namely, the scheme should satisfy a proof of knowledge requirement. Formally, this means that an efficient algorithm E called the "extractor" can, with black box access to P, determine the hidden value. In our setting:

$\forall x, h, P^* Pr[g^x = h : x \leftarrow E^{P^*}(h)] \geq Pr[(P, V)(h) = 1] - \epsilon$ where ϵ is considered the *knowledge error*.

Here would be our extractor algorithm:

Run P^* to determmine u.
Send $c_1 \rightarrow^R \mathbb{Z}_\text{l}$ to P^* and get response z_1
Rewind the prover P^* to its state after the first message.
Send $c_2 \rightarrow^R \mathbb{Z}_\text{l}$ to P^* and get response z_2
Output $\frac{z_1 - z_2}{c_1 - c_2} \in \mathbb{Z}_p$

Here, the algorithm fails if $c_1 - c_2 = 0$ which happens with probability $\frac{1}{p}$. As a result, the knowledge error is $\frac{1}{p}$ for provers that always convince the verifier.

8.1.8 Proving that Schnorr's Sigma Protocol is Zero Knowledge

Our goal is to construct an efficient algorithm that will produce a distribution identical to a verifier's view of the protocol. We will only prove HVZK: the non-interactive version of Schnorr's Protocol will automatically become strongly zero knowledge against even malicious verifiers because the protocol transcript will not depend on the verifier at all.

Consider the following algorithm S:

$z \rightarrow^R \mathbb{Z}_\text{l}$
$c \rightarrow^R \mathbb{Z}_\text{l}$
$u \leftarrow \frac{g^z}{h^c}$
Output (u, c, z)

Note that we perform the protocol in reverse to guarantee that the verifier's constraints are satisfied without having to perform difficult discrete logarithms. We only satisfy honest-verifier ZK because we implicitly assume that the verifier will generate c with uniform randomness. Next, note that we arrive at strong zero-knowledge because the transcript/view distributions are identical: $\forall g, h \in \mathbb{G} : \{Sim(y)\}$ which equals $= \{(u, c, z) : c, z \leftarrow^R \mathbb{Z}_p, u = \frac{g^z}{h^c}\}$ which equals $= \{(g^r, c, z) : r, c \leftarrow_R \mathbb{Z}_p, z = r + cx\}$ which equals $= \{View_V((P, V)(g, h))\}$

This holds because each tuple (u, c, z) is uniformly random such that (u, c, z) satisfies the constraints $g^z = u * h^c$.

8.1.9 Licensing and Attribution

This tutorial is heavily inspired by the 2019 CS355 course and lecture notes: https://crypto.stanford.edu/cs355/19sp/about/, particularly for the details of Schnorr's Protocol and Proof of Knowledge systems. Special thanks to Floran Tramèr, Dima Kogan, and Henry Corrigan-Gibbs for being such fantastic instructors.

I also found Oded Goldreich's ZK: A Tutorial to be immensely helpful in understanding different variations of zero knowledge. I highly recommend starting there to learn more about zero-knowledge!

CHAPTER

NINE

DISTRIBUTED HASH TABLES WITH KADEMLIA

9.1 Motivation: A Book-Lending Thought Experiment

Imagine we enjoy reading physical books and do not want to read books online. How would we go about finding books to read?

Libraries are fantastic. They have enormous selections of books, and it is quite easy to find the book you are looking for in the library with the Dewey Decimal System. At the same time, it can be fun to physically own books and share books between our friends.

This alternative approach to reading, while fun, has some complications. For example, how do we determine who owns which book? Most people don't bother maintaing a central list of who owns what. That would also be a pain to maintain (at least pre-Google Docs). Imagine we didn't care who owned which book and just wanted to make sure your system is easy to follow and resistant against new and volatile friendships. How would we design our system?

In particular, how would we:

- Determine who owns which books?
- Communicate with our friends to fetch our desired book?
- Handle the scenario when you have a falling-out with a friend and that person decides to leave the group suddenly?
- Scale to a *super* (1,000,000) large friend group while still letting this system remain functional?

This contrived example is somewhat related to the notion of peer-to-peer file sharing. A data structure for a peer-to-peer filesharing system is a Distributed Hash Table (DHT).

9.2 Peer-to-Peer File Sharing in Action: IPFS

A cool product that uses DHTs under the hood is the Inter-Planetary File System (IPFS). Definitely consider trying IPFS out for yourself.

We will now explore how DHTs work.

9.3 Distributed Hash Tables

Suppose we want to store data in a way such that we can easily find what we are looking for. One data structure that allows that is our traditional map, where we store elements as <key, value> pairs. Maps support two operations: PUT(key, value), which inserts a new element, and GET(key), which returns the value of the element corresponding to that key. For example, if we wanted to store a bunch of words and definitions on our computer, we save (PUT) each definition (value) under the word (key) it defines. When we want to know what a word means, we get its definition by looking it up (GET) in the map.

While maps are useful, they do have one major limitation: they are only stored locally on a single computer. What happens if our elements have super large values, like a file that contains a 2-hour long movie? This is an issue because the more movies we store, the more space we take up in the computer, and it can become difficult to store our entire map on one computer. Alternatively, what if we have an enormous number of elements? One way to solve this problem is to store separate parts of the map on separate computers. We could connect these parts by designating a centralized computer, or coordinator, which would coordinate which keys are stored on which computer. This map would not be very resilient, however, to when the coordinator goes offline.

A Distributed Hash Table (DHT) is a kind of data structure stored on multiple computers that aims to address this issue. We still want our basic operations PUT(key, value) and GET(key), like we had in our map, but we want our data structure to still stand even if some computer leaves our network. In designing our DHT, we want it to be:

- *Scalable*. When more computers participate, we want the DHT to split up the elements evenly among the computers.
- *Fault-Tolerant*. If a computer leaves the network, we want to be able to access the elements that it previously held.

There are some fundamental limitations here. If all computers leave at once, we have nowhere to store anything – thus, we will need to assume that computers leave at a slow enough rate. Intuitively, we will probably need to replicate keys across different computers so that key-value pairs will be recoverable even if some of those computers leave at once.

9.4 A Decentralized But Organized Partition of the Map

Our goal will be to design a protocol that divides up portions of the map somewhat evenly among participating computers without having a centralized coordinator. As usual, randomness can come to the rescue. We can have participating computers take on an identifier and then have elements be assigned to the computer whose id is closest to the key. These identifiers (ids) and keys can be part of some shared key space of integers, for example all 160-bit integers. This idea has several advantages:

1. The actual choice of determining an identifier does not require any centralized coordinator.
2. Depending on the notion of closeness and space, a uniformly random selection of identifiers will likely distribute keys somewhat evenly among computers.
3. Adding a new computer requires merely shifting the few keys that were previously close to another identifier but are now closest to our new computer.

One may be queasy that our keys are limited to 160-bit integers. Don't worry: any key can be hashed into this 160-bit keyspace with a collision resistant hash function. For historical reasons, the original authors used SHA-1.

PUT and GET operations then become simple: find the closest computer to a key, and either store or query for that key. Of course, we still have a major problem left to address: how do we find the closest computer to a given key?

Normal maps are often represented in two general ways: as a hash table and as a search tree. First, we will briefly explore how we could use a hash table to motivate a geometry for segmenting the map's keyspace.

9.4.1 Hash Table Partitions: Chord

Since our keys are integers (which we could interpret as non-negative), we could interpret our hash table as an indexable array. We could then break up the array into contiguous blocks and assign these blocks to participating computers. In a distributed view, computers will implicitly claim contiguous blocks by choosing a random id and then claim the closest keys to that computer where the distance between a key k and a computer with id is simply $k - id$ (where the binary representation is interpreted as a non-negative integer).

These participating computers could be viewed as forming a ring structure in increasing order going in a clockwise direction. Keys assigned to a given computer will then be those that are in the space between a computer id and the id of the computer after it. This geometry is also called Consistent Hashing, and is the base layer for how Chord, a DHT, works (Figure 1).

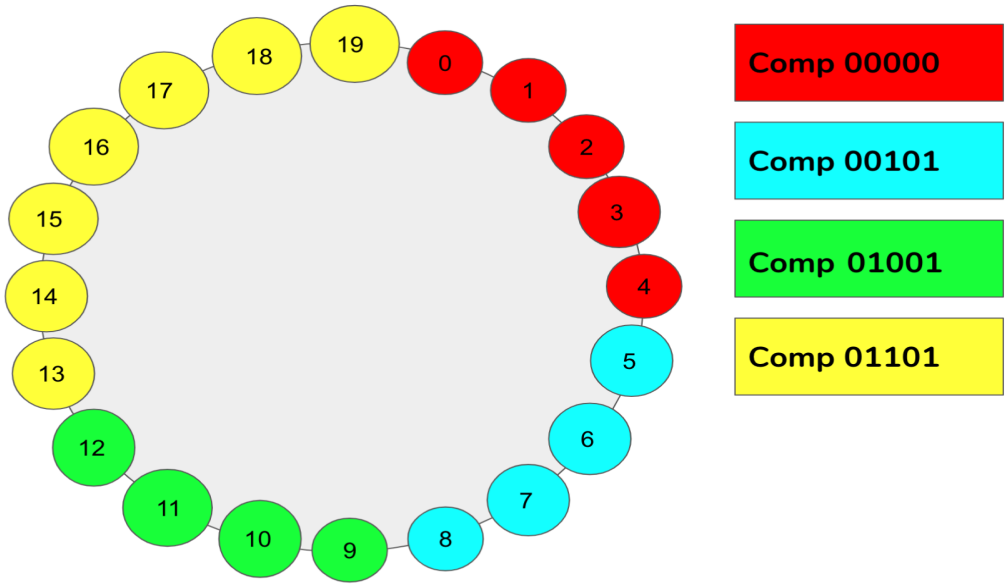


Fig. 1: Figure 1: Illustrated here is the geometry of ring consistent hashing. Displayed are four computers. Namely, we have Computer 00000, Computer 00101, Computer 01001, and Computer 01101. That is, computers 0, 5, 9, and 13, respectively. These numbers are randomly assigned. We can see by the color code that computer 5 contains all IDs between 5 and the value of the next computer (9). This pattern continues around the ring.

This fulfills the advantages mentioned above by making the ring become denser and denser as more and more computers participate. Similarly, the key spaces between computer ids will likely shrink and become relatively uniform as more computers insert themselves into the network with random identifiers. Let $successor(k)$ return the computer with the closest id in the clockwise direction in the ring at point k. A computer with id can then store the addresses of computers of the form $successor((id + 2^i)twoupton$, where n is the number of computers in the ring (Figure 2).

Lookups can consist of nodes recursively calling on the node that they know of that is closest to the key of interest. Node lookup runtime takes $O(log(n))$, but we will not focus on analyzing the runtime, correctness, nor details of this structure for this article. However, if your intuition tells you that DHTs can be represented as segmented hash tables, then you are right!

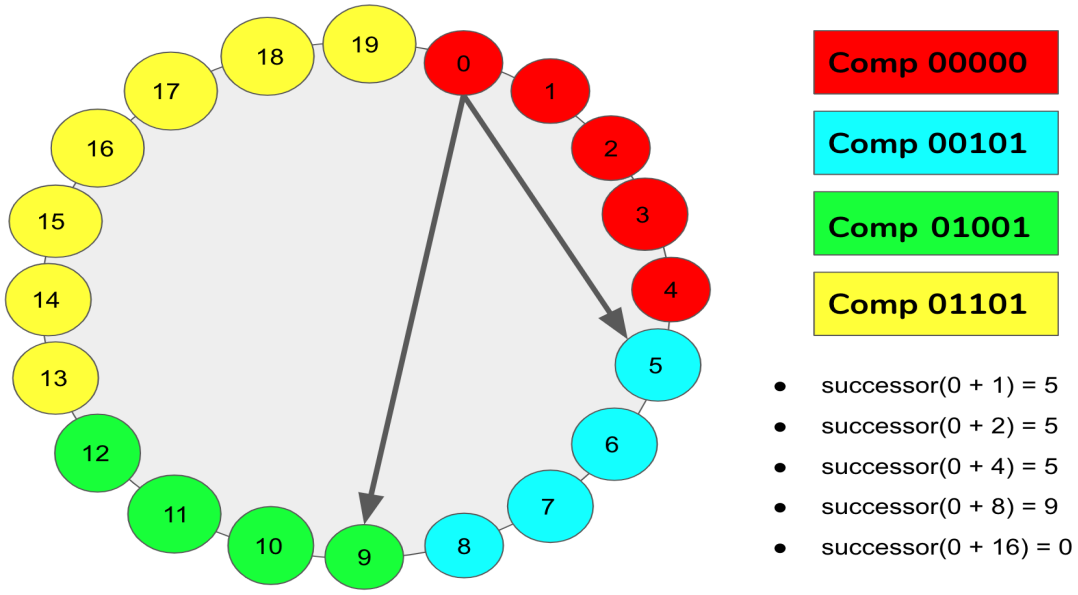


Fig. 2: Figure 2: Depicted above is the same set of computers and IDs as in Figure 1. However, now we are showing the computers visible from ID 0. That is, we start at 0 and add successive powers of 2 up to the number of nodes in the ring. In our case specifically, we have 20 computers. This means we will add the numbers 1, 2, 4, 8, and 16 to our start node (0), and we will call successor(k) on each of these nodes. Our result is a list of computers visible from node ID 0.

9.4.2 Search Tree Partitions: Kademlia

Our main focus will be in examining how to partition a binary trie. As with most things, it is best to discover how to segment the tree with a visual example.

Let us say that all ids and keys are in a keyspace of $[0, 1, \ldots 2^3 - 1]$ and are represented in binary. We can represent this space as a complete binary tree where each leaf node is a key. Circled leaves are ids that correspond to a participating computer in the network. In the example above (Figure 3), three computers are participating in the protocol with ids of 000, 110, and 111 respectively.

For Chord, we partition keys into contiguous blocks of a circular array/ring. How should we partition the leaves of this tree among the circled leaves? A natural segmentation could be to assign a key to the node with the lowest common ancestor. We can then color code the diagram as follows, where leaves with the same color as a computer leaf mean that keys should be stored on that computer:

Note that we also annotated the edges of the trie. This shows the isometry between the path from the root of the tree to a leaf and the binary representation of the leaf node. Common ancestors then become common prefixes of all leaf descendants!

Note the coloration of 100 and 101. Unfortunately, there was a tie among computer nodes for the lowest common ancestor: 110 and 111 both have a common ancestor representing a common prefix of '1__'. Thus, it was a somewhat arbitrary choice to assign those keys to the computer 110.

How could we better formalize a rule for breaking this tie? In fact, a more optimal solution would be to evenly divide 100 and 101 amongst 110 and 111. Why not generalize the notion of longest common prefix?

If two computers tie for the longest common prefix, we can instead look at the most significant bit index b where they differ. We can then check bit b of the key and assign the key to the computer whose bit b in their id equals bit b of the key.

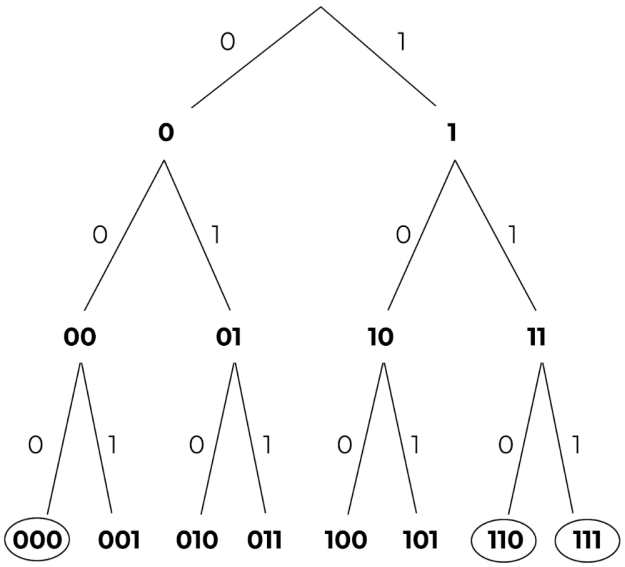


Fig. 3: Figure 3: A binary tree with height 3. The circled leaves represent ids of computers.

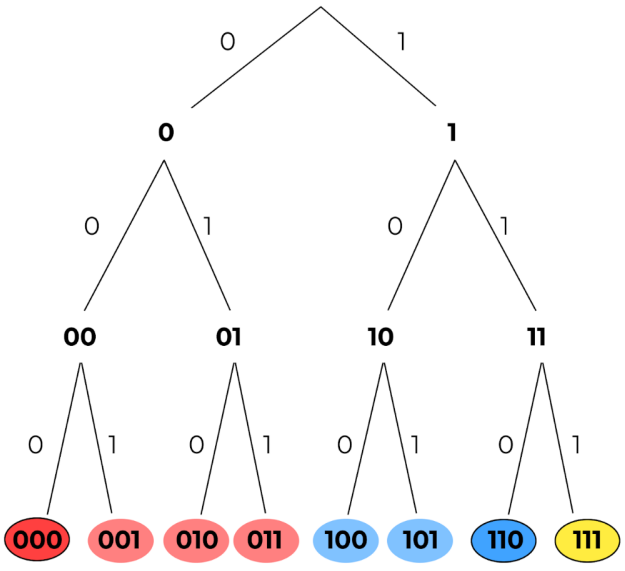


Fig. 4: Figure 4: Leaves are color coded to show that the left 4 bits have a lowest common ancestor with 000 and thus are assigned to that computer. Similarly, 100 and 101 are assigned to the computer with id 110, and 111 has nothing assigned. This is actually erroneous, but it will motivate an "improvement" of closeness later.

Notationally let 100_i be the ith bit of 100, 0-indexed from left to right. For example, $100_0 = 1$. Next, consider key 100 and computer ids 110 and 111. Both 100, 110, and 111 share a common prefix of 1. However, bit 1 of 110 and 111 are equal: 110_1. Thus, we consider the first bit at which they differ: bit 2. Since 110_2 but $111_2 \neq 100_2$, 100 should be assigned to the computer whose id is 110.

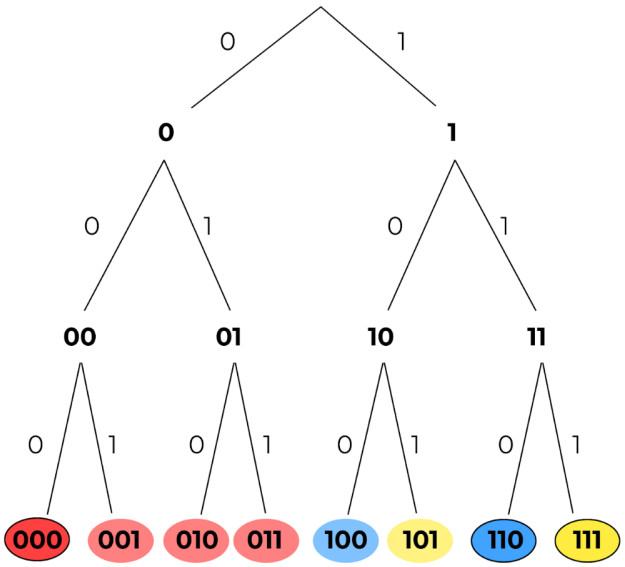


Fig. 5: Figure 5: Here is the same tree, but with the correct assignments of **100** to **110** and **101** to **111**.

9.5 Defining Distance for Keys in a Binary Trie

We have now a mechanical definition for determining where a key should be assigned:

Mechanical Definition of Closeness Between Keys and Computer IDs:

Look at the binary representation of the key and computer ids. Now, start with a candidate set of all the computers' IDs. While the candidate set has more than one computer id remaining, go bit index by bit index from the most significant bit to the least significant bit. If at least one candidate id has the same bit as the key at our current bit index, remove all candidate ids that differ from the key at that bit index. Once we have one ID left, that computer ID is the closest ID.

We could extend this and compare "closeness" by seeing how many iterations a computer id will remain in the candidate set before it is trimmed out on a certain bit index. Alternatively, we could determine the ith closest computer by removing the first $i - 1$ closest computers from our candidate set and then find the closest computer out of the remaining ids. While interesting and complete, we would like to translate this *mechanical* definition into an *operational* definition.

Intuitively, we want to define closeness as some scalar value that strictly prioritizes similarity among the most significant bits over less significant bits but still allows for comparison even if all computer ids differ from the key at hand on significant bits. In other words, we would like to be able to add penalties to ids that differ from the key at a significant bit so that they are not as close to ids that equal the key at that same significant bit index. We would like to add decreasing degrees of penalties such that ids that were penalized are permanently excluded from ids that were penalized later: we want to mimic this strictly shrinking candidate set from our mechanical description.

In this sense, "penalties" suggest that defining distance may be more intuitive than defining closeness. One could consider a key and a computer id and look bit by bit. If the bits are the same, no penalty should be applied. If bits

differ, this earlier penalty should be applied so that all other later penalties would still be less than this penalty. Powers of two capture this idea of penalties perfectly: since $2^i > \sum_{j=0}^{i-1} 2^j$, one could add a higher power of two for more significant bits if they differ from the key.

Mechanical Definition of Distance Between a Key and Computer ID: Initialize the distance value at 0. Compare the key and computer ID bit by bit. If the key and computer ID differ at the ith least significant bit (with the least significant bit having an index i=0), add a penalty of 2^i to the distance value.

One may recognize this process: this is exactly what the XOR operation does! We just need to interpret the XOR result as a non-negative integer to allow for this bucketing of penalties to strictly order ids that have higher bits in common under ids that differ from ids that do not have that significant bit ancestor in common.

Operational Definition of Distance Between Keys and Computer IDs: The distance between a key and a computer id is the XOR of their values interpreted as an integer.

Thus, a key should be stored on the computer that is closest to the key, or has the lowest XOR value with the key relative to all other computer ids.

From this point forward, distance will be defined as the XOR between two ids/keys interpreted as a non-negative integer, and the "closest" computer to a key is the computer with lowest XOR distance between its id and the key.

Further, XOR is a distance metric. If someone is interested, please email someone at Code the Change, and we can include an appendix proving that XOR is a distance metric.

9.6 Tree Partitions as Routing Tables

Now that we have established which keys should be stored with which computers, we need to determine how computers can find keys from other computers. Intuitively, we would like to only store the addresses of a small amount of computers but still be able to find keys relatively quickly. One possible goal is that every query for a computer should get us at least "one bit closer" to the computer that actually contains the key of interest. This could roughly give us a runtime of $O(log(n))$. Let's assume that our participating computers all exist at the beginning and never leave, so they can be represented by the tree shown above.

We could therefore store a logarithmic number of computer ids and their corresponding IP addresses in an organized contact book. Which computer ids should we store? We could look back at our complete tree diagram for inspiration.

For now, let's ignore *how* a computer develops its address book and instead think about *what* we want the address book to look like. Let's consider the view of computer 111. Note that we could just have computer 111's "view" of the map be solely the encircled region above. In these sibling nodes, we could merely store any computer whose id is a descendant of that sibling node. For example, for the node that represents prefix '0', we could store '000'. There are no computers that begin with prefix '10', so that internal node will be empty. For the node that represents prefix '110,' we could store the address of computer '110.' Each of these sibling nodes is called a "k-bucket" for reasons mentioned later (for now, k=1).

9.7 Key/Computer Lookups

Let's now consider a far away key: 011. How would a **GET(011)** work for computer 111? This key would be stored in computer 010. However, '010' is not listed in our node 111's address book. The closest computer to 010 that we know of is computer 000. We could then ask computer 000 to tell us whether it has key 011 or if computer 000 knows of any closer computers to 011. Computer 000 should in turn have 010 in its address book, so 000 could return the IP address of 010 to 111, and then 111 could ask 010 for key 011, which it should have.

We can formalize this process as follows:

1. Find the closest computer in your routing table (via XOR).

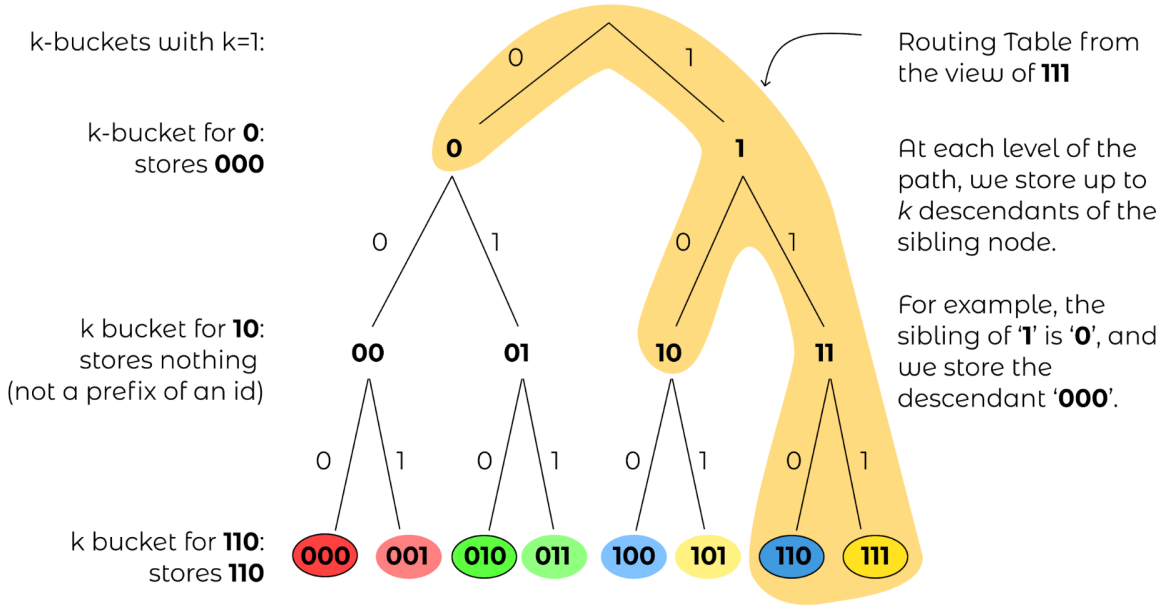


Fig. 6: Figure 6: The routing table from the view of computer with id **111**, with the corresponding k-buckets for k=1.

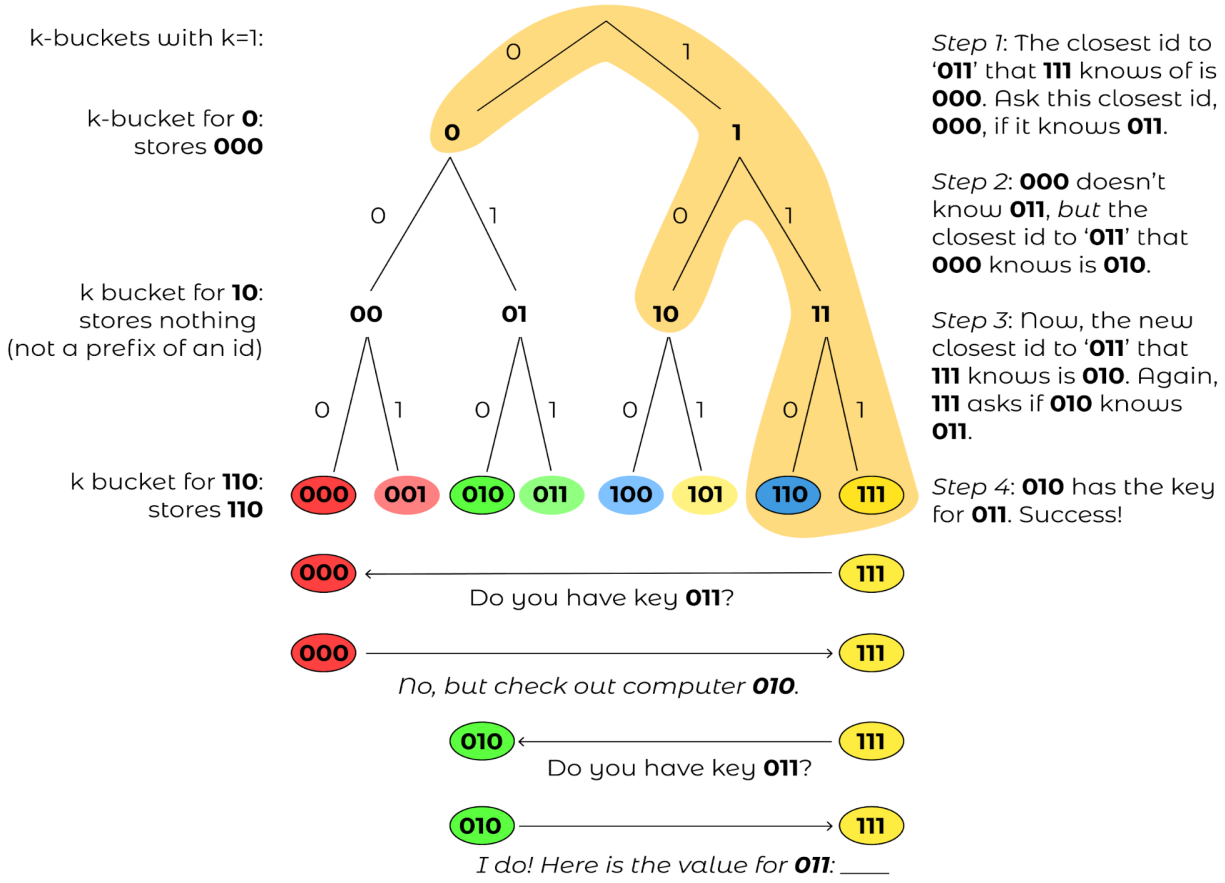


Fig. 7: Figure 7: 111 wants to perform GET(011).

2. **While the closest computer you know of does not have the key and has not already responded:**
    a. Ask the closest computer you know of for the key or a closer computer
    b. If the closest computer responds with a closer computer, update our closest computer variable.

Now, let's analyze the correctness assuming that each computer routing table is correctly populated with at least one computer id per k-bucket if such a computer exists within that range for that k-bucket. Let id_{global} be the identifier of the closest computer to the key by XOR across all computers.

We will prove that, for each step in the node lookup when a query computer q queries a computer c, the returned id_{close} will be in the same k-bucket as id_{global} from computer c's perspective.

Consider id_{global}'s k-bucket in c and let id_k be the computer id stored in that k-bucket. Let p be the binary prefix represented by the internal node for id_{global}'s k-bucket.

If id_{close} is in a different k-bucket than id_{global}, then id_{close} has a different prefix than p for those first $|p|$ bits. Given that more significant bits always contribute more to distance than all the successive bits combined, and that id_{close} is closer than id_k, then no identifier with prefix p can be closer than id_{close} to the key. But id_{global} has prefix p and is the closest identifier globally to the key, so this cannot be true. Thus, the closest computer cannot be inserted into any k-bucket other than the same k-bucket as id_{global}. For a visual understanding of what we are trying to prove, see figure 8 for a description of this proof by contradiction: we are trying to show that that $id_k = id_{close}$.

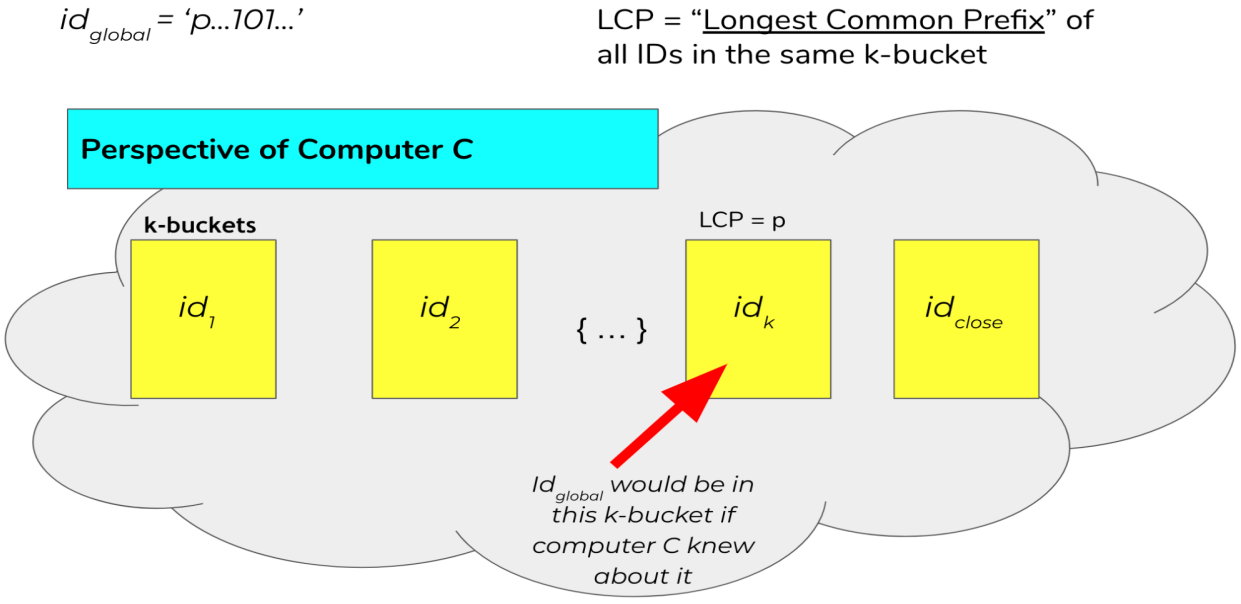


Fig. 8: Figure 8: The figure above illustrates the routing table of computer c. Each of these k-buckets contains an ID with a certain longest common prefix. For example, the k-bucket on the far left contains id_1 and represents all ids that differ from c's id on the first bit. Note that because id_{close} is in a different bucket as id_k, this would imply that id_{close} is closer to our goal than id_{global}, which is a contradiction.

This invariant over each lookup will guarantee that we will eventually find id_{global} by having us increase the length of our common prefix with id_{global} in each lookup iteration. Since there are only $log(n)$ unique bits in id_{global}, our lookup runtime and number of hops should take expected time $O(log(n))$.

We could visualize these network requests as "traveling down the trie." Since we are guaranteed to always return identifiers that are in the same k-bucket as where id_{global} would be, we are guaranteed that each iteration of our closest known id will have a common prefix with id_{global} that is at least 1 bit larger than the previous id. This will guarantee that if the key space is 160 bits, then we will find id_{global} in 160 iterations. If the tree is sufficiently balanced

enough, which will be with high probability given that computers generate their ids uniformly at random, we should instead find id_{global} in $O(log(n))$ iterations, where n is the number of computers participating in the network.

We have found a way to do lookups given a static network! Woohoo! Note that since the key space equals the computer-id space, looking up computers and looking up keys are equivalent procedures: instead, we look for the closest computer to a given computer id.

Finally, we can simplify this highly unbalanced routing table trie somewhat. We will never have computers in the k-bucket corresponding to the same 158-bit prefix. Thus, why should we store those k-buckets at all? Instead, a computer should consider the k-buckets up to and including the computer ids that are closest to it that it knows of.

9.8 Supporting Dynamic Leaves and Joins

The gap from our current state to a fully dynamic DHT is not as far away as it seems. Let us first consider how we can extend this protocol to support computers sporadically leaving. Let us assume that we have some known parameter k that represents the number of computers such that it is extremely unlikely that all of those computers will leave the network in the same hour. For example, the authors of the original Kademlia paper experimentally determined that their k should be 20.

Next, our k-bucket should, instead of storing just 1 computer, actually store k computers within that k-bucket's range. Thus, with high likelihood, there will always be at least one computer online in each of a computer's k-buckets.

How do computer lookups change? Our new procedure is as follows:

1. Find the k closest computers in your routing table (via XOR).
2. **While you have not heard responses from all the k closest computers:**
    a. Ask the new k closest computers for their own k closest computers in their respective routing tables, potentially including themselves.
    b. Update our list of the k closest computers we know of given the newly returned closer computers.

Next, how do we keep our k-buckets up to date? If a computer c in a k-bucket has not been queried over the past hour, our current computer current should ping c to make sure it is still online. If c does not respond, it can be evicted from the k-bucket table. Therefore, computers can leave the network at any point during the protocol without notifying other computers: this computer will just be cleaned out of a computer's k-buckets.

If we learn of new computer ids and their IP addresses by doing lookups and a k-bucket has additional space, we can insert this (id, IP addr) pair into the k-bucket. Note that this eviction/insertion paradigm favors older computers in a k-bucket. This decision is intentional: while this means that computers that have stayed longer in the network get an unfair burden of network traffic, the authors experimentally found that computers that have been around the longest in the network are much more likely to continue staying on than new computers. Thus, old computers are favored to maximize connectivity in the network from experimental observations of how peer-to-peer network participants behave.

Next, how do we become aware of new computers that could fit into what are unfilled k-buckets? We could perform lookups on random ids within a k-bucket's id range (the range is defined by all leaves that are direct descendants of the k-bucket's corresponding internal node in the complete tree) and thus learn about computers within that k-bucket if they exist. This is called a bucket refresh.

Finally, we support computer joins. A computer j must know the id and IP address of at least one other computer c. j will then perform a computer lookup on itself to get the closest known computers to j. Then, j will perform bucket refreshes for all k-buckets from the closest known computers to j. This will populate the routing table of computer j based on the correctness of the lookup algorithm and the fact that the routing tables of other computers are sufficiently populated. In addition, all computers that are being queried by j will become aware of j in the process and thus will have the opportunity to insert j into their own k-buckets if their corresponding bucket is not already full.

Finally, whenever a computer *current* learns of a new computer c, current can also check if c is closer to any keys currently stored on current are actually closer to c. If that is the case, it can issue PUT requests to :math:`c`for those keys. fThis will allow keys to be automatically moved to new but closer computers.

9.9 Walkthrough of a Kademlia Network Genesis

Every Kademlia network comes from modest beginnings. Consider the computer 000, which will store its own keys at the beginning. For simplicity, we will consider the case where $k = 2$ on our familiar 3-bit keyspace.

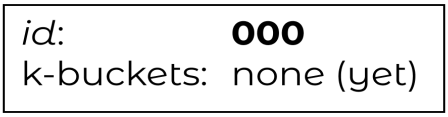


Fig. 9: Figure 9: A single computer with id 000.

Next, a new computer 010 joins, performing a computer lookup of itself on 000. Computer 000 will only respond with nothing, so both computers will end their routing table with just each other. Since there is only a k-bucket of the form between 00 and the root, 010 will only perform bucket refreshes within 1__ and 0__.

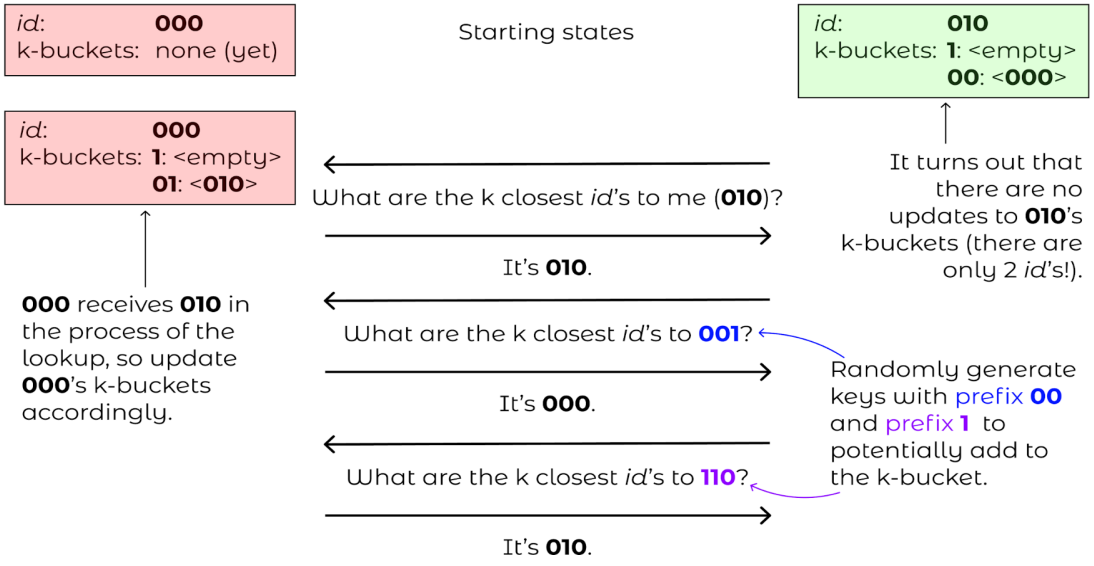


Fig. 10: Figure 10: Routing tables for **000** and **010**, with **010** asking **000** for 1) the closest ids to itself (**010**), 2) the closest ids to some random id with prefix **00** and 3) closest ids to some random id with prefix **1**.

Finally, consider computer 111 joining while only knowing computer 000. 111 will issue a computer lookup on itself. It will start by asking "what are the closest computers to myself" for 000, receiving 010 in the process. The closest node 111 knows to itself is 010, so 111 will only have a k-bucket for prefix **0** which will contain 000, 010. 111 will then ask "what are the closest computers to myself" to 010, receiving 010 and 000 again in the process, so 111 will terminate its lookup procedure. 010 and 000, after hearing requests from 111, will populate 111 in their prefix **1** k-buckets.

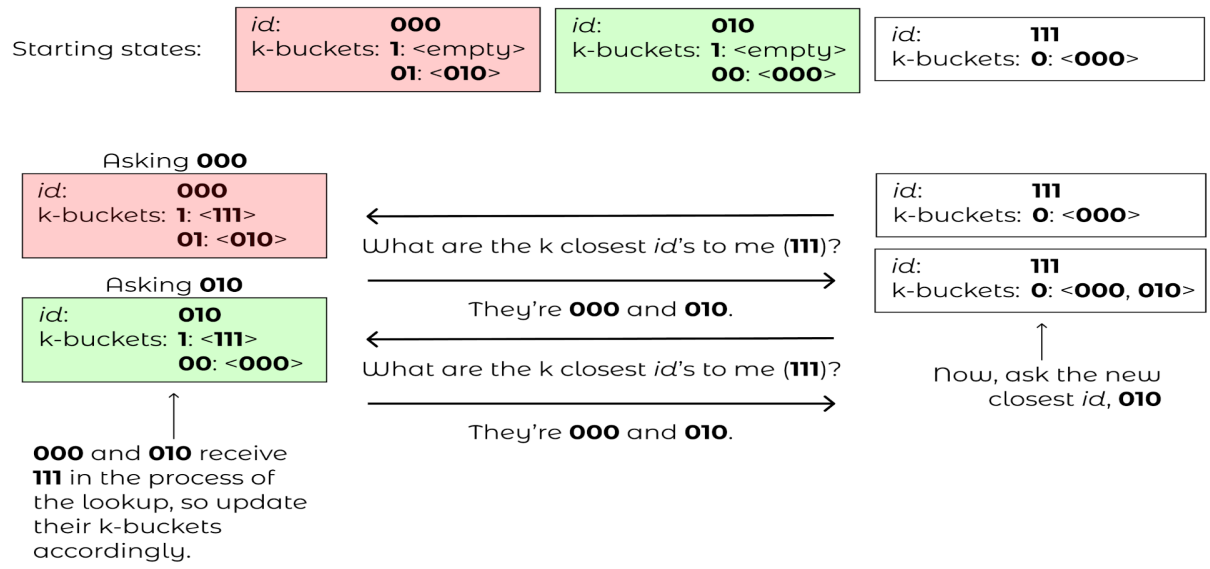


Fig. 11: Figure 11: The lookup process when 111 is added to the network.

9.10 Piecing Together the DHT

With key and computer lookups down, all that remains is to more formally define the network remote procedure calls (RPCs).

A computer can be asked **FIND_COMP(id)** call and will return k of the closest computer ids in its routing table and their IP addresses.

A computer can receive a **FIND_VALUE(key)** call and will return the value if the (key,value) pair is stored locally on the machine. If the key is not stored locally, the computer will respond as if it received a **FIND_COMP(key)** call.

A computer can receive a **STORE(key, value)** and will just store the key-value pair in a local map of its choice.

A computer can receive a **PING** call to verify that the computer is still online.

To ensure that keys remain in the network, the caller who stored or requested a resource is required to re-issue a **STORE** call within a given time frame, such as every 24 hours. Otherwise, computers will automatically evict old key-value pairs to reduce bloat.

To speed up computer lookups, **FIND_COMP/FIND_VALUE** calls can be done asynchronously for some asynchronous parameter α. In the original paper, $\alpha = 3$. In other words, we could issue **FIND_COMP** calls in parallel to 3 different computers at a time during our computer lookup procedure.

9.11 Conclusion

At last, we have our DHT! A **GET(key)** operation is a computer lookup at key followed by a **FIND_VALUE()** call for the k closest computers to key by id. A **PUT(key, value)** operation is a computer lookup at key followed by a **STORE(key)** operation at the k closest computers (which should be repeated within some interval if you want the (key,value) pair to stay there).

The original paper mentioned various improvements to the DHT. For example, to reduce traffic for often-accessed keys, one could store the keys on a wider group of closer computers each time it is accessed. To increase computer discovery for a newly inserted low-depth computer id (in the case of unbalanced trees), the authors recommended storing subtree of k-buckets instead of a single k-bucket in the routing table to increase network discovery of this newly inserted computer id that now commands a large region of the key space.

Note that we made a key assumption: the computers would choose their ids at random and follow the protocol honestly. Often, distributed systems such as Kademlia operate in a wild, trustless environment. S/Kademlia extends Kademlia to prevent various sybil attacks with cryptographic hash function hardness as a form of proof-of-work.

9.12 More Resources

Of course, it would be super valuable to read the original paper introducing Kademlia.

In addition, Kelsey Chan's interactive visualization for various Kademlia operations is both quite visually appealing and helpful for understanding how Kademlia works.

9.13 Licensing and Attribution

CHAPTER

TEN

AN INTRODUCTION TO WEB SECURITY

Danger: Web security is a huge field with far more nuances and details than I can cover here. If you are interested in learning more or are responsible for making security decisions for a web application, see the resources at the end of this guide. **Understanding this guide is insufficient experience for doing real security work!**

10.1 Introduction

10.1.1 Motivation

Many of the protocols that underpin our modern internet were created without modern security concerns in mind. I actually think this was pretty reasonable at the time. When the ARPANET, the precursor to the modern internet, was first created in 1969, it looked like this:

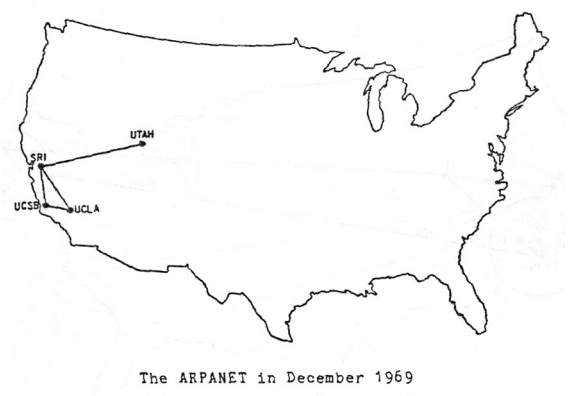


Fig. 1: From J. Noel Chiappa at MIT, 2014-11-07. Images comes from the "ARPANET Technical Information: Geographic Maps" page

When the only people on the network were four universities, you just didn't have to worry about as many security problems as you do now, when less than 30% of the internet looks like this:

The fundamental insecurities of many web protocols make cybersecurity on the internet difficult. Still, there are lots of steps we as developers can take to build more secure applications.

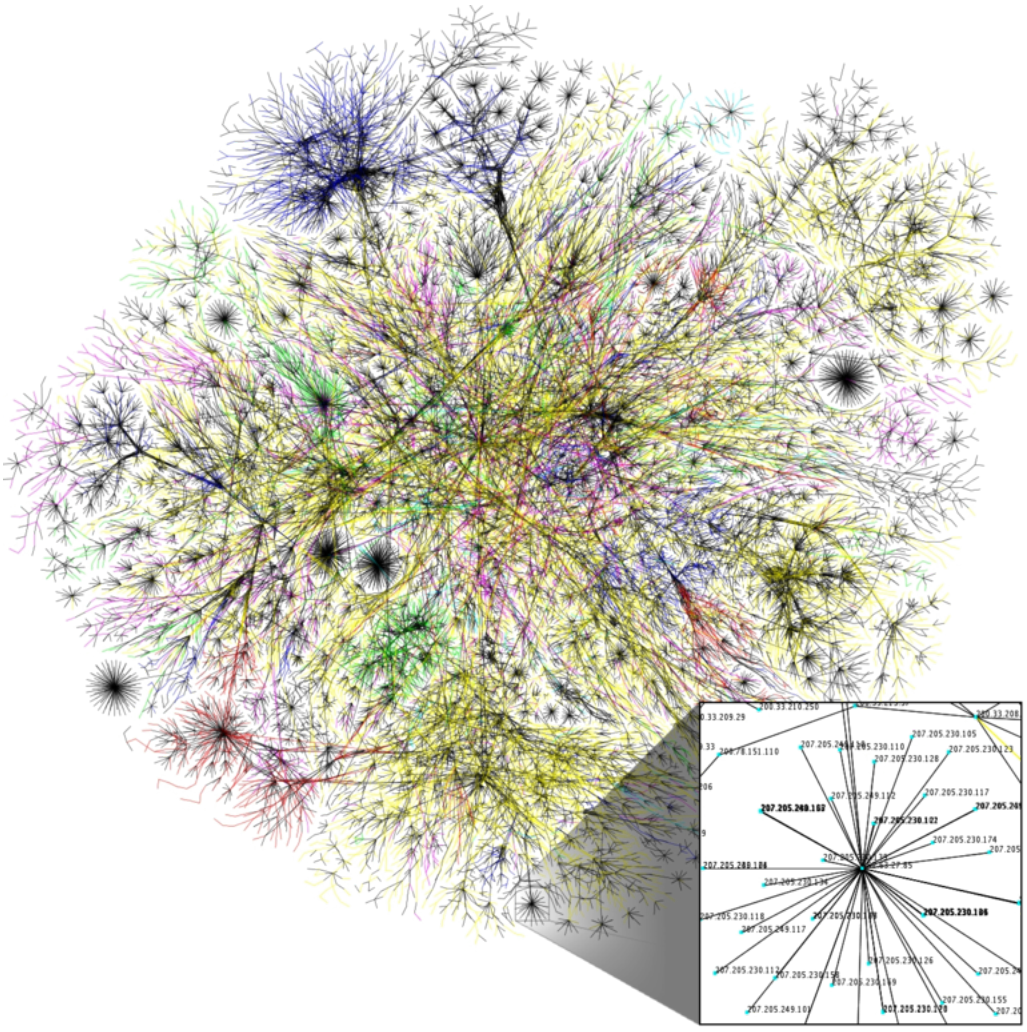


Fig. 2: By The Opte Project, 2006. Sourced from Wikimedia at this page.

10.1.2 Overview

In this guide, we will cover some common security vulnerabilities that affect web apps. I've put together a simple web app for you to run locally that has a secure version and a vulnerable version. We'll experiment with the vulnerable version to see how these attacks work, and then we'll see how the secure version fixes the vulnerability.

10.1.3 Goals

By the end of this guide, I hope you will have a high-level understanding of how the following attacks work:

- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)
- Monkey-in-the-Middle (MitM)
- HTTPS Downgrade
- SQL Injection

I also hope you'll have a high-level understanding of how the following defensive strategies work and how they prevent various attacks:

- Escaping user input
- CSRF tokens
- HTTP Strict Transport Security (HSTS)
- Authenticating users on every page
- Two-Factor Authentication

Lastly, I hope that by the end you find cybersecurity interesting enough to go and learn some more on your own! This is a fascinating field, and there are tons of great resources for learning more. I'll link to some at the end of the article.

10.2 Setting Up PwnMe

In case you haven't seen it before, "pwn" comes from online gaming culture where it means to defeat (i.e. to "own") an opponent. In hacker circles it refers to compromising a target. You can find more information at wikitionary.

10.2.1 Download

I created a simple web application, PwnMe, that we can use to experiment with security vulnerabilities. It's a Flask app that you can run locally on your computer. Don't worry, the app is only visible from your computer. It cannot be accessed by other computers on your network unless you pass `--host=0.0.0.0` to the `flask run` command we'll discuss below.

> **Danger:** Please do not run PwnMe as a publicly accessible website. I purposely chose to have you run it locally to ensure no one exploited its vulnerabilities maliciously. If you expose it publicly, there's always a chance that an unsuspecting person will stumble upon it and fall victim to its vulnerabilities.

PwnMe is available on GitHub at https://github.com/U8NWXD/pwnme. Clone the repository to your local computer like this:

```
$ git clone https://github.com/U8NWXD/pwnme.git
$ cd pwnme
```

10.2.2 Install Dependencies

Now, let's set up a virtual environment and install the app's dependencies.

Note: You might need to replace `python` with `python3` in the below commands depending on how you installed Python. This app requires Python 3.

I like to use Python's `venv` module for virtual environment, but you're selcome to use whatever solution you like. You can even forgo the virtual environment together and install these requirements globally, though I don't recommend it since doing this can muck up your system with unnecessary packages.

```
$ python -m venv --prompt pwnme venv
$ source venv/bin/activate
$ pip install -r requirements.txt
```

10.2.3 Initialize PwnMe

Configuration

To configure PwnMe, we need to generate a configuration file at `instance/config.py` like this:

```
$ mkdir instance
$ python pwnme/generate_config.py > instance/config.py
$ cat instance/config.py
SECRET_KEY = "<secret key>"
```

Instead of `<secret key>` you should see 64 hex characters. This is a 256-bit secret key that Flask will use for security operations like signing cookies and generating CSRF tokens (more on this later).

Initialize Database

PwnMe stores its data in a SQLite database at `instance/pwnme.sqlite`. To initialize this database, run:

```
$ source environment
$ flask init-db
Initialized the database
```

Notice that we told `Flask` about our app using the environment variables in `environment`.

10.2.4 Run PwnMe

Now, you can run the web app:

```
$ flask run
  * Serving Flask app "pwnme" (lazy loading)
  * Environment: development
  * Debug mode: on
  * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
  * Restarting with stat
  * Debugger is active!
  * Debugger PIN: <pin>
```

Don't worry about the value of `<pin>`; we won't use it.

10.2.5 PwnMe Structure

When you first launch PwnMe and go to http://127.0.0.1:5000, you'll get a page with links to two versions of PwnMe: a safe version and a vulnerable version. The vulnerable one is susceptible to many of the attacks we'll discuss, while the safe one has been hardened against them.

10.3 A Note on Responsible Bug Hunting

Many of the vulnerabilities in this guide are commonly found in production today. Many companies run bug bounty programs to encourage people to responsibly disclose these bugs so they can be fixed. If you choose to participate in these programs, **you must closely adhere to responsible disclosure policies.** For example, take a look at hackerone's policy. I even have one for my own website.

These policies are important because by default, testing for vulnerabilities in websites is usually a federal crime under the Computer Fraud and Abuse Act (CFAA). Most vulnerability disclosure policies include safe harbor provisions that promise not to press charges against you so long as you follow the policy.

Note: I'm not a lawyer, and these policies can be pretty complicated. If you decide to start looking for vulnerabilities in real websites, you should carefully consult the site's policies and bring any questions to a qualified lawyer.

For this guide, we're going to be avoiding these issues by testing our exploits on our own web app that's running locally on our computers.

10.4 Common Web Vulnerabilites and Their Mitigations

Danger: **Never** look for vulnerabilities in websites without permission. Doing so may be a federal crime and carry both civil and criminal charges under the Computer Fraud and Abuse Act (CFAA). It's frequently a felony!

10.4.1 Cross-Site Scripting (XSS)

Cross-site scripting was originally used by Microsoft engineers to describe an attack where a malicious website loaded a target app and got it to execute malicious JavaScript. However, it has now come to refer to many other kinds of code

injection, which makes the name pretty confusing. XSS is when an attacker injects client-side code into a web page and gets that code to run in another users's browser when they visit the page.

Note: If you aren't familiar with the idea of client-side code, here's a quick overview. In the web, code can run in two different places: on the server (server-side) or on the machine of the client accessing the server (client-side). For example, when you send an email, the email is transmitted to your recipient by server-side code, but the layout of the message box you are typing into is defined by client-side code. This distinction is important. For example, if your server has secret keys that no user should know, you'd better not put them in client-side code! Similarly, any good password manager will encrypt a user's passwords client-side so that the server never has access to them. PHP and Python are server-side languages, while HTML and JavaScript generally run client-side. (Node can use server-side JavaScript.)

The Vulnerability

Let's try out XSS on PwnMe. Launch PwnMe with `flask run` and navigate to http://127.0.0.1:5000. Then load the vulnerable site and click on the `Who Am I` link in the navigation bar. If you type your name into the box, the app says hello to you!

Use your browser's developer tools to take a look at the page's HTML. You can generally access these by right-clicking on the page. Find where your name is being shown. You should see code like this:

```
<section class="content">
  <header>

  <h1>Hello</h1>

  </header>



  <!--The |safe filter disables autoescaping, allowing reflected XSS-->
  <h1>Hello, YOUR_NAME!</h1>


</section>
```

Take a moment to think about how this program works. It takes in your name and puts that into an HTML template. How might this be vulnerable to XSS? Don't scroll down until you've thought about it!

There are lots of possible attacks here, but I'm going to just show you one to give you a sense for what's possible. Try pasting `<script>alert('You have been hacked!');</script>` into the field for your name. When you submit the form, an alert pops up! Here we just made an alert pop up, but client-side JavaScript can do a lot. For example, you might be able to impersonate the user and submit commands to the server. If the user has a lot of permissions on the server, you could do a lot of damage.

Notice that the name you submit ends up in the URL. This means you could send someone an email with a link that includes your malicious code. If they open it in a browser where they're signed in, your code has free reign to mess with their account.

The Fix

Now, switch to the safe version of PwnMe and try again. Instead of an alert popping up, you should see `Hello, <script>alert('You have been hacked!');</script>!` displayed as the name. It's definitely still weird, but now your injected JavaScript is being displayed as text instead of being executed.

Let's take a look at the code to understand how the safe version prevents XSS. PwnMe uses Jinja2 HTML templates, which can be configured with parameters from our Python code. If you compare at the HTML templates `pwnme/templates/hello_vuln.html` and `pwnme/templates/hello_safe.html` you'll notice that the vulnerable template has an extra `|safe`:

```
<h1>Hello, {{ name|safe }}!</h1>
```

The `|safe` disables escaping. Flask configures Jinja2 with automatic escaping turned on by default, but `|safe` overrides that. Jinja2 is nice in that it handles escaping automatically, but we could do it with another tool too.

Important: In any web application, make sure that any untrusted content is properly escaped before being sent to users. Untrusted content includes user-provided content, but it may also include third-party content you aren't sure is safe.

You may be tempted to try doing the escaping yourself. Just don't. These security operations are difficult to get right, and you're better off using code that's already been reviewed by experts.

Important: **Never** try and implement security code on your own. Security primitives like escaping content, hashing passwords, and encrypting data are available as reputable packages or built-in functions for nearly all modern programming languages. Use them! They're much more likely to get all the tricky details right than we are.

Even with this fix, are there still ways an attacker could abuse this site? Here's one way: what if an attacker set the `name` parameter to `Valued Customer. Your account has been hacked! Call (555) 555-5555 for immediate assistance`? In a sense this is an injection attack, only instead of code we're injecting a statement the legitimate website doesn't actually want to make. There aren't really blanket mitigations for this kind of attack except making very clear in your UI what is user-provided and what is not.

10.4.2 Cross-Site Request Forgery (CSRF)

CSRF attacks exploit the fact that web operations are mostly stateless. When you go to a web form, you use a GET request to see the form. Then when you submit the form, a POST request sends the contents of your form to the server. Like all HTTP requests, these operations are independent. You could have submitted the POST request directly from your terminal. The only reason for the form in the web page is as a user-friendly tool to create the POST request.

Take a few moments to think about how this could be abused. When you have some thoughts, scroll down to see how an exploit can take advantage of this independence of HTTP requests.

Imagine you're logged into your bank account in one tab, but you're browsing the web in another tab. If you open up a malicious site, say evil.example.com, you should be fine since evil.example.com can't access the data on your bank web pages. However, what if evil.example.com tricked you into submitting a form that sent a POST request to your bank? Since you're logged in already, that POST request looks exactly the same as the one that would come from the form on your bank's website. You might have thought you were asking to see 100 cat photos from evil.example.com, but the form actually transferred 100 dollars from your bank account!

The Vulnerability

Even more insidiously, forms can be invisible and submit automatically. Take a look at `malicious/csrf_vuln.html`:

```
<body onload="document.forms[0].submit()">
  <form action="http://127.0.0.1:5000/vuln/withdraw/" method="POST">
        <input type="hidden" name="amount" value="100">
  </form>
</body>
```

This form submits automatically and withdraws 100 dollars from your bank account at PwnMe. Let's see this in action.

Navigate to the vulnerable PwnMe site and register for an account. Log in, and check your balance. You should have 0 dollars. Now go to withdraw money and withdraw -100 dollars. Now you have 100 dollars.

Note: Obviously you shouldn't be able to withdraw negative amounts of money, but this works for what we need. If you're interested in cool exploits when user input isn't properly validated, check out buffer overflow and numeric overflow attacks.

Now, let's pretend you click on a link in an email that sends you to a website with `malicious/csrf_vuln.html`. To simulate this, open `malicious/csrf_vuln.html` in the same browser where you're logged in to PwnMe. If you check your balance again, you should see that your money has disappeared!

The Fix

To fix this vulnerability, we need a way to tell whether a POST request comes from our website or another site. The standard way to do this is with CSRF tokens. These tokens are essentially unguessable passwords given to a web form and passed along when the user submits the form. Since a malicious site doesn't have these passwords, it can't submit a valid request to drain your bank account.

CSRF tokens are a pretty simple idea, so you might be tempted to implement them yourself. **Don't.** There are other reputable projects that implement CSRF more reliably than we can. The Google Cloud Platform's flask-talisman project recommends Flask-SeaSurf. Flask-SeaSurf operates on an entire app, so to only apply CSRF protection to the safe version of PwnMe, I used Flask-WTF instead.

Switch to the safe version of PwnMe. If you go the withdraw funds page and inspect the HTML, you'll see a form like this:

```
<form method="post">
    <input id="csrf_token" name="csrf_token" type="hidden" value=
↪"IjE1YjM2NTQwZTIzZTM1MjI1ZDBkM2ZjZWU3M2MyZTYyNmQyYjBhMWQi.X-gT3g.
↪EPXLTXLzO4CBozBRsJx0rOykafY">
    <label for="amount">Amount</label> <input id="amount" name="amount" required=""␣
↪type="text" value="">
    <input type="submit" value="Withdraw Funds">
</form>
```

Notice the CSRF token. Now let's simulate navigating to a site that attempts a CSRF attack against the safe version by loading `malicious/csrf_safe.html`. Now you see a CSRF validation error, and your money is safe!

Important: Every authenticated endpoint on your site that changes state should be a POST endpoint and implement CSRF protections.

10.4.3 Monkey-in-the-Middle (MitM)

A monkey-in-the-middle attack is where a malicious server sits between the legitimate server and the user. You can imagine it like this:

```
+------+         +----------+         +--------+
| User | <-----> | Attacker | <-----> | Server |
+------+         +----------+         +--------+
```

The attacker impersonates the user when talking to the server, and they impersonate the server when talking to the user. This lets them fully control the conversation. For example, they could alter the server's response to the user to give the user bad information, or they could manipulate a user's request to cause the server to send money to the attacker's account instead of the user's.

Note: There are actually legitimate uses for MitM! Sometimes organizations want to monitor and filter internet traffic. For example, schools might want to keep kids from accessing illegal content at school, and companies might want to block phishing sites. To do this, they act as a MitM. To do this, they have to configure their users' devices to trust root CAs under the organization's control. Unfortunately, some governments try to surveil their citizens this way. In late July, 2019, researchers revealed that Kazakhstan was instructing its citizens to configure their browsers to trust a root CA run by the Kazakhstan government so they could act as a MitM. In response, both Chrome and Firefox blacklisted the CA.

The Vulnerability

Actually, both the safe and vulnerable versions of PwnMe are susceptible to MitM attacks because we've been working with them over insecure connections. Notice how the URLs start with `http://` instead of `https://`. That means we aren't using TLS (Transport Layer Security), a security protocol that encrypts web traffic and authenticates the server to the user.

Note: You might have heard of the SSL protocol. It's the precursor to TLS, though people often still use "SSL" when talking about TLS.

The Fix

To prevent MitM attacks, we need to secure those connections with TLS. This is handled at the level of the web server, not the web app, so we won't demonstrate it here. However, we can still discuss how it works.

When you visit a TLS-secured website, the website presents a cryptographically signed certificate that proves the site contains a secret key. It also contains a signature from a certificate authority (CA) that. This signature certifies that the CA has confirmed that the controller of the secret key is also the proper owner of some domain. Your browser comes with a list of CAs it trusts. These are called root CAs, and they have to follow strict regulations and undergo regular audits.

As an example of how seriously browsers take violations by root CAs, let's consider Symantec. Symantec was one of the largest and oldest certificate authorities, but a number of instances were discovered from 2013 to 2017 where Symantec issued certificates improperly. As a result, Google, Apple, and Mozilla all stopped trusting certificates issued by Symantec. This was disruptive for server operators, who had to get new certificates, and for Symantec, who had a lot of unhappy customers. Browsers don't mess around with violations by CAs!

Since the CA ecosystem is so tightly secured, it's very hard to get a fraudulant certificate. This means that if your website uses TLS, an attacker won't be able to impersonate you, preventing a MitM attack.

At one point, many websites didn't use TLS because it was too expensive to get certificates. Now, you can use Let's Encrypt, a free service that gives out TLS certificates that are trusted by all major browsers. They verify website ownership just like other CAs, so they're still secure.

Important: Every website should use TLS for everything. No exceptions.

Note: You might notice that for some sites, particularly banking websites, the name of the website's company appears next to the lock icon in your browser. This signifies that the website is providing an Extended Validation (EV) certificate. While normal certificates just confirm that the holder owns the website's domain, EV certificates prove which legal entity controls the certificate and that the legal entity is the proper owner of a website.

10.4.4 HTTP Downgrade

The Vulnerability

Unfortunately, not all websites use TLS, so browsers have to allow unsecured HTTP connections. Similarly, not all browsers support TLS, so websites usually allow unsecured connections too. This means that a MitM attacker can bypass the protections of TLS by tricking the user into thinking the server is HTTP-only and tricking the server into thinking the user doesn't support TLS. This is an HTTP Downgrade attack.

The Fix

HTTP Public Key Pinning (HPKP)

A now-discouraged solution to HTTP downgrade attacks was HTTP Public Key Pinning (HPKP). This involved setting a header with the server's response that specified the public key of one of the certificates verifying the website's identity. The browser would then only allow TLS connections with the website, and it would only allow TLS connections with that particular certificate.

This meant that if a rogue certificate authority issued a certificate for your website, that certificate wouldn't be accepted because it would have the wrong public key.

HPKP has now been deprecated because it was very easy for websites to pin the wrong certificate, permanently blocking browsers from connecting to them. This also meant an attacker who managed to set headers on your requests could do the same. Some legacy browsers support it still, but you shouldn't use it.

Important: Don't use HPKP.

Instead, browsers have adopted Certificate Transparency, which provides additional oversight for the certificates issued by CAs to detect bad behavior. I won't go into it much here, but it's a very cool protocol. I encourage you to check it out!

HTTP Strict Transport Security (HSTS)

HSTS is a somewhat softer version of HPKP that is recommended for use today. It is a header you send with your requests that tells browsers they should only ever contact you over TLS. From then on, the browser will refuse to connect over plain HTTP, preventing downgrade attacks.

You can even set a `preload` option in your HSTS header that will get your site included in a list shipped with major browsers. Then users don't have to visit you at all to know that they should only connect with you over TLS.

Important: Use HSTS to prevent downgrade attacks.

10.4.5 SQL Injection

SQL injection is similar to XSS in that it relies on getting a computer to execute what's supposed to be just data as code. Instead of executing JavaScript, SQL injection uses SQL commands.

The Vulnerability

Let's try this out on the vulnerable version of PwnMe. Log in and go to the `Lookup` page. This page lets you look up a user's ID from their username. For a bank, this might be how you look up someone's account number to send them money. Since we store user data in a database, we must be executing some kind of SQL command to look up the user ID. Maybe it's something like this:

```
SELECT * FROM user WHERE username = "user_input";
```

How might we exploit this? Scroll down after you've thought about it a little.

What if you provided `robert"; DROP TABLE user;--` as the username? Then the command would become:

```
SELECT * FROM user WHERE username = "robert"; DROP TABLE user;--";
```

This command is valid because `--` makes the last quote a comment. First we'll look up a user ID, but then we'll delete the entire `user` table!

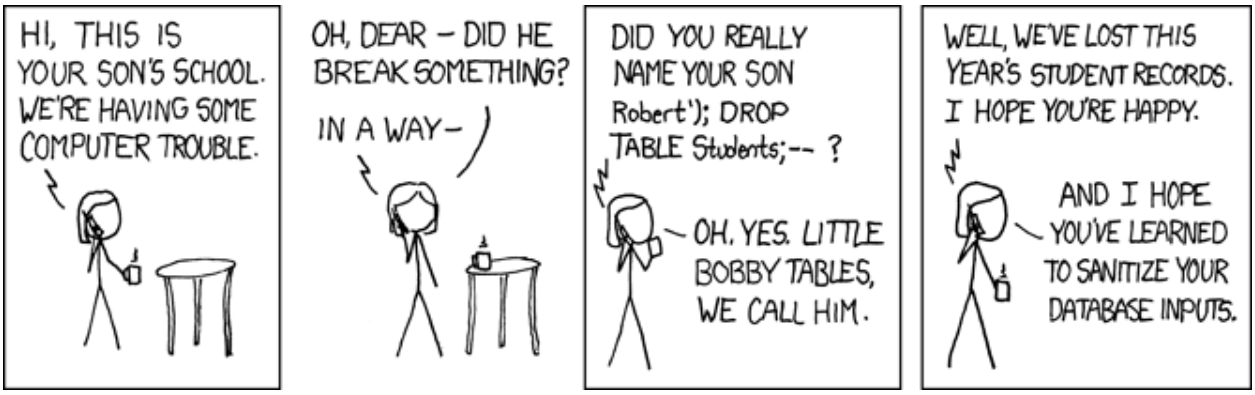


Fig. 3: This is an XKCD comic titled "Exploits of a Mom" available under a CC-BY-NC 2.5 license. Importantly, this means that this comic is not licensed for commercial use.

If we look at the code in `pwnme/vuln.py`, we see two problems:

```
user = db.executescript(
    f'SELECT * FROM user WHERE username = "{username}"'
).fetchone()
```

First, we used `db.executescript`, which allows for multiple commands separated by semicolons. This is unnecessary for this lookup function. Second, we put the user-provided username directly into the SQL command with no filtering or escaping.

The Fix

Instead, we should have used code like in `pwnme/safe.py`:

```
user = db.execute(
    'SELECT * FROM user WHERE username = ?', (username,)
).fetchone()
```

Here we use `db.execute()`, so only one command can run. More importantly, though, we let the `execute()` function handle putting the username into the command. This lets the `sqlite3` library do proper escaping. This is called parameterizing commands.

Go ahead and try this exploit on the safe version of the site. It should behave as if no user was found because your input was properly escaped.

Important: Never put untrusted data into a SQL command without proper escaping. As a general practice, you should parameterize all your SQL commands to let robust libraries handle this for you.

10.5 Useful Web Security Practices

10.5.1 Authenticating Users at Every Endpoint

In PwnMe, you'll see a `Secret Message` option when you log in. If you click it, you'll be able to view a top secret note. Now, copy the URL of the note and try to load it when you're signed out. In the vulnerable site, this still works! What we should happen, and what does happen on the safe site, is that you get directed to the login page instead.

If you look in the code at `pwnme/safe.py` you'll see that the `hidden()` function is decorated by `@login_required`. This decorator is missing in `pwnme/vuln.py`.

Important: Just hiding the link won't stop hackers. You need to actively check that user's are authenticated at *every* secured endpoint.

10.5.2 Two-Factor Authentication (2FA)

You're probably all familiar with two-factor authentication. While it is really more of an authentication technique, which we haven't discusssed much here, I wanted to highlight it since it's so useful.

2FA, which marketing teams the world over have alternately named two-step verification, security codes, verification codes, two-step authentication, and more, deals with a simple problem: passwords suck. Users can't remember very strong ones. When websites force users to have strong ones, they tend to write them down on sticky notes at their desks that then get picked up by camera crews.

2FA mitigates many of these problems by checking that you control some physical object to let you sign in. This could be your phone (e.g. an SMS code, Duo, or the TOTP protocol), a physical key that generates numbers (banks seem to like these), or a security key (e.g. FIDO2). These are much harder and riskier for hackers to steal since they're physically near you.

Here are some common 2FA methods:

- Transmitted Codes: Numeric codes are often sent over SMS or email. These are among the least secure kinds of 2FA because SMS and email accounts can be compromised remotely (and sometimes easily). For example SIM swap scams can let hackers steal your phone number and get your SMS 2FA codes. One exception to this is Apple's 2FA with trusted devices, which sends codes to trusted devices.
- Codes Generated from Pre-Shared Key: Numeric codes are generated from a secret key that the server gave you when you first set up 2FA. You probably scanned a QR code that contained the key. From then on, your device and the server can both generate the same code, which the server uses to check that you have your device. Sometimes this code changes based on the time (TOTP), and other times it changes based on how many times you've asked for codes before (HOTP). Since the secret key is securely stored on your device, it's very difficult to steal remotely. This scheme is also quite easy to implement on your own because of many open-source implementations and no need for a separate communications channel like SMS.
- Notifications to Trusted Devices: Duo and Google Prompts fall into this category. When you sign in, the server sends a notification to an app on your phone that then prompts you to allow or reject the request. This is harder to implement than TOTP or HOTP because you have to build the notification infrastructure, but it similarly benefits from the difficulty in stealing trusted devices.
- Security Keys: These are the most secure forms of 2FA I've seen. A security key is usually a USB device you plug into your computer to authenticate. That key generates an authentication token that is only valid for the site you're visiting and for that sign-in attempt. This makes it the only method here that prevents phishing.

Phishing is where an attacker gets a user to visit a malicious site that looks like a trusted site. For example, it might mimic PwnMe's login page. When the user attempts to sign in, they are inadvertantly giving their credentials to the attacker. Any numeric codes from 2FA can also be phished and quickly used to sign the attacker in before they expire (usually in 30 seconds or so). Even notification-based 2FA can be phished because if the attacker triggers a log-in attempt on the legitimate site immediately after the user puts their credentials into the phishing site, the user will probably approve it thinking they're on the legitimate site.

Security keys, and particularly the FIDO2 protocols, prevent phishing by tying authentication tokens to the pages at which they are generated. Even if the user puts in their security key at the phishing site, the generated token is only valid at the phishing site. It won't work at the legitimate site, so the attaker can't log in. Google uses security keys exclusively for their Advanced Protection Program, and it found that when it gave security keys to its employees in 2017, it completely eliminated employee account takeovers.

Password managers also provide a powerful defense against phishing since they usually check what website you're on before auto-filling your password. This requires that users be disciplined about not copy-pasting passwords into sites when their password manager won't auto-fill.

10.5.3 Security Headers

There are lots of other security headers you can add, some of which are demonstrated by `add_response_headers()` in `pwnme/safe.py`. Flask provides a handy list of security changes, including headers, you can make to your Flask app.

10.6 Resources

For learning web security:

- Stanford's CS253 course
- The Security section of Google's Web Fundamentals guides
- Mozilla's Web Security Guidelines
- The Security Considerations section of Flask's documentation
- Hacker101

For learning cybersecurity more broadly:

- Fedora's Defensive Coding Guide

Reference material for secure coding:

- The Open Web Application Security Project (OWASP), particularly their reference guide and Security Knowledge Framework

NIST also has a list of free and low-cost cybersecurity lessons that you may find helpful.

10.7 Licensing and Attribution

This work was initially created for a workshop at Stanford Code the Change.

CHAPTER

ELEVEN

SECURING THE OPEN SOURCE SOFTWARE SUPPLY CHAIN

11.1 Introduction

11.1.1 Motivation

Case Study: SolarWinds

Sometime before March 2020, Russia's Foreign Intelligence Service (SVR) compromised SolarWinds, a software company based in Texas that makes the Orion network management tool. Note that Russia has denied involvement, but US officials have said the attack has the hallmarks of an SVR operation.

The SVR used its access to add malicious code (the payload) to Orion. Orion is used by thousands of organizations all over the world, and nearly eighteen thousand of them installed a compromised version of Orion. Among these eighteen thousand were US Pentagon, Treasury, Department of Homeland Security, and National Institutes of Health, not to mention other governments and companies.

While we won't know the full impact of this hack for years, this hack highlights how vulnerable the software supply chains we all rely on are.

To read more about the SolarWinds hack and see the articles from which I drew the information about the hack in this guide, see this blog post from Yubico, this essay from Bruce Schneier, and this NPR story.

Other Supply-Chain Hacks

While the SolarWinds hack is among the most recent and damaging supply-chain attack, there have been many others. Here are some from the Linux Foundation's "Open Source Software Supply Chain Security" publication from February 2020:

- In 2015, developers were tricked into using a malicious version of Xcode that inserted malware into their apps.
- In 2017, attackers introduced malicious libraries to the Python Package Index (PyPI). They chose library names that were similar to legitimate, popular packages so developers would install them by accident.
- In 2019, attackers compromised the Ruby account of the developer of the popular Ruby package `strong_password`. The attackers used this access to deploy a new version of the package that would download and execute arbitrary code of the attacker's choice on the developer's machine.
- Later in 2019, 11 backdoored Ruby packages were found to have been infected with malicious code. Four were infected by compromising a developer's account.

Responsibility of Software Developers

When users run our software, they are trusting us to ensure it is free of malicious code. In this guide, we'll discuss how software projects get compromised and what you can do to keep your projects safe.

11.2 How Projects Get Compromised

At a high level, you can think of your project as a mixture of code from you and your dependencies:

```
Dependency -----> +---------------+
Dependency -----> | Your Project | -----> Users
Dependency -----> +---------------+
                          ^
                          |
                          |
                  Code You Write
```

From this view, we see that your project can be compromised in a few ways:

- Vulnerable dependencies
- Intentionally-introduced malicious code from you
- Accidentally-introduced security bugs from you

Let's take a look at each of these and how you can mitigate the risks they pose.

11.2.1 Vulnerable Dependencies

According to GitHub's Octoverse 2020 Security Report, an active repository in a supported package ecosystem like PyPI or NPM had a 59% chance of getting a Dependabot alert about a dependency with a security vulnerability in 2020. This means your repository will probaby have a dependency with a security vulnerability.

Luckily for us, GitHub will automatically send us Dependabot alerts whenever a dependency has patched a vulnerability. By staying on top of these alerts, you can keep your dependencies up-to-date.

Important: Act on Dependabot alerts quickly to upgrade vulnerable dependencies.

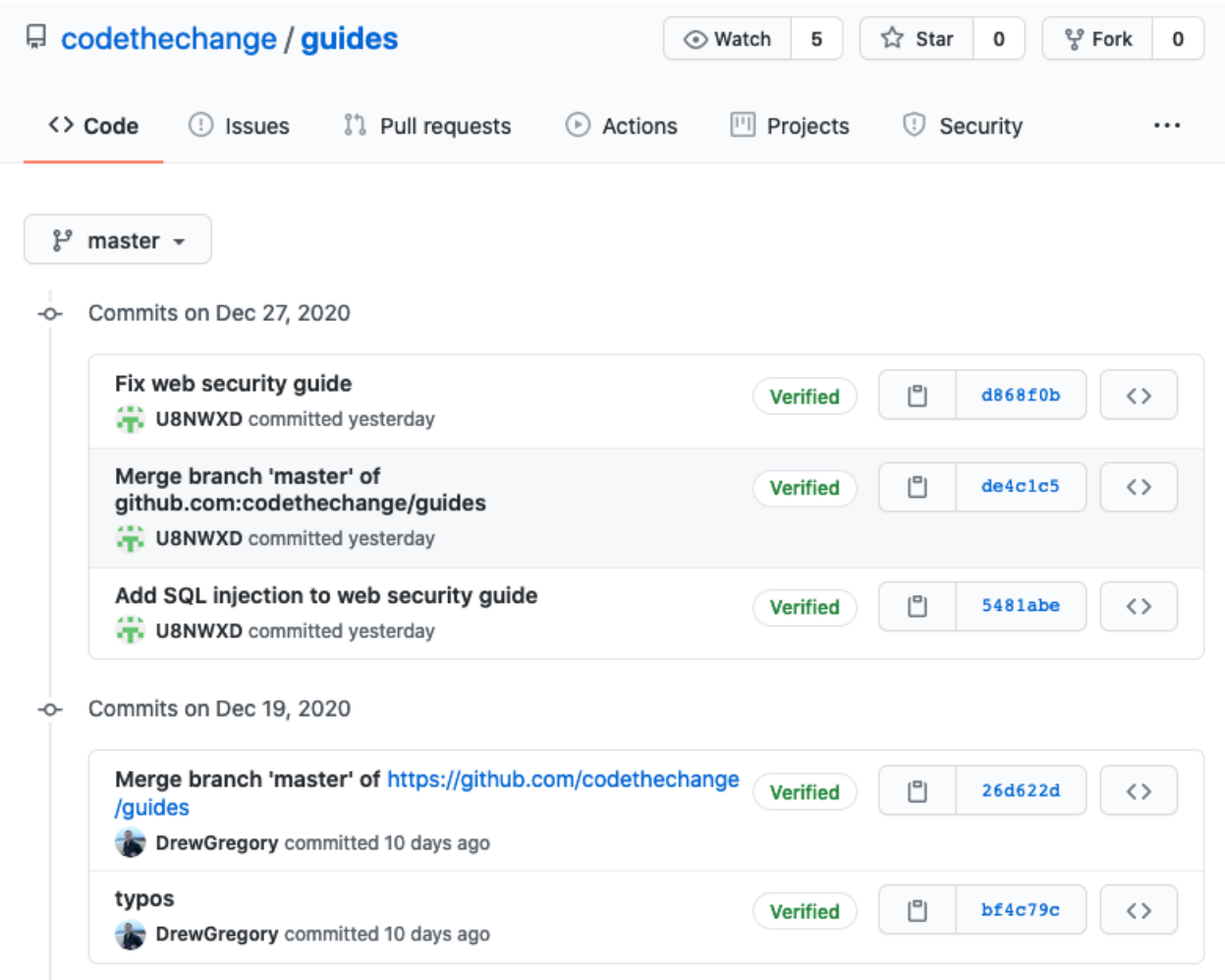
11.2.2 Intentional Malicious Code from You

You might first think about one of your developers intentionally introducing security bugs. In my opinion, this is pretty unlikely in the small teams we work on. What small risk remains is pretty effectively mitigated by the fact that `git` lets us track who introduced what changes.

A bigger threat is an attacker compromising one of your developers' accounts. Recall that many of supply-chain attacks we discussed earlier involved compromising a developer's account. Here are some steps you as developers can take to secure your accounts:

- Use a strong, unique password for your GitHub account. I *highly* recommend using a password manager, for example KeePassXC or Bitwarden.
- Enable two-factor authentication on your GitHub account. Here are instructions. This is one of the easiest and most impactful steps you can take to improve your personal security. Do it right now!

As a bonus step, I strongly support cryptographically signing your git commits. GitHub has instructions you can follow. For example, if you look at our guides repository at Code the Change, you'll see a `Verified` label next to our commits:



This label indicates that our commits have each been signed by the author with a secret key they alone control. This makes it easy to identify commits introduced by a hacker–they will be unsigned.

Important: Secure your GitHub account with strong, unique passwords and two-factor authentication. As a bonus steps, start signing your commits.

11.2.3 Accidental Malicious Code from You

According to the 2020 Octoverse Security Report, only 17% of security vulnerabilities on GitHub were intentionally malicious, and these vulnerabilities caused just 0.2% of Dependabot alerts. Most software vulnerabilities are just mistakes.

Note: Most security vulnerabilites are mistakes. Unfortunately, they also go unnoticed for 4 years on average.

Unfortunately, there are no easy solutions to avoiding security bugs. GitHub has code scanning features that may be

useful, though I haven't used them yet.

More generally, we stop security bugs the same ways we stop other bugs: defensive coding, thorough testing, and code review. You can ensure these steps aren't bypassed by setting up protected branches on GitHub.

Important: Enforce code review and passing automated tests for all code changes. Make sure your tests are comprehensive.

11.3 Helping Users Trust You

Just keeping vulnerabilities out of your code isn't enough. You also have to help users trust that the code they download is secure.

11.3.1 Signing Software

All software you ship should be digitally signed so your users can verify that it hasn't been tampered with. I find it easiest to create a signed git tag for each release. If you then create a release on GitHub, you'll get a `Verified` label just like with commits. Here's an example from one of my projects:

Important: Securely sign all software releases to verify their authenticity.

11.3.2 Version Documentation

If your repository is ever compromised, you want to be able to tell your users something like

> "version 1.1.2 was compromised, so please make sure you have upgraded to version 1.1.3."

This is only possible if you release numbered versions of your software. I recommend following the Semantic Versioning numbering system, as it has become standard in open source software.

To get the most out of versions, you should also document the changes in each version using a changelog. In fact, the first compromise of a Ruby package in 2019 that we discussed above was caught when a user of the package noticed that a new version had been released without an update to the changelog.

11.3.3 Transparency

Finally, you need to communicate to your users all the security practices you have in place to keep them safe. This could be in your documentation or README file. The important thing is to let your users know what practices you follow (e.g. signing releases) so they can spot anomalies (e.g. an unsigned release) that might indicate a compromise.

I also recommend letting people know how they can securely report security problems to you. Many of the hacks we discussed at the The earlier Ruby package compromise in 2019 was caught when a user reported the problem to the developer. This required the developer to have published secure ways to reach them. This should all go into your security policy, for instance a SECURITY.md file on GitHub.

Important: Be transparent about your security policies and how people can report vulnerabilities to you.

11.4 Licensing and Attribution

This work was initially created for a workshop at Stanford Code the Change.

CHAPTER

TWELVE

CREATING ACCESSIBLE WEBSITES

12.1 Introduction

12.1.1 Motivating Example: Developing a Website

Design for People Like You

Imagine you are a software developer creating a new website. When you browse the web, you:

- Use a laptop computer.
- Enter text into forms with a keyboard.
- Click on buttons and scroll with a trackpad.
- Consume web content visually.
- Are adept at reading and understanding lots of text.

When you create the website, you'll probably most naturally design for users like you–users who browse the web the same way you do. I know I probably would. After all, I understand my own experience better than I can understand anyone else's.

Design for People Unlike You

If you want other people to use your website, though, you'll have to design it for people who browse the web differently. Thankfully, web technology supports this. For example, most major website you view today will be compatible with the smaller screen sizes of mobile devices. If you are viewing this guide online with a desktop or laptop computer, try making your browser window narrower and watch the site adjust.

While most developers hear about designing websites for mobile devices, designing websites for disabled users is much less prominent. Far too many websites today are inaccessible for users with disabilities. This may be why only 54% of disabled Americans use the internet, compared with 81% of Americans without a disability, according to the Pew Research Center.

If your website isn't accessible, disabled people might be forced to look for alternative websites. According to the World Bank, 15% of the world have a disability. That's a billion people who your website might not serve effectively if it isn't accessible.

Note: Don't let your website be one that is unusable for the many people with disabilities.

Just as you can design for people who use mobile devices, you can also design for people with disabilities. For example, you can support:

- People with auditory disabilities by including transcriptions and captions for audio.
- People with cognitive or learning disabilities by clearly structuring your content and replacing long blocks of instructions with form labels.
- People with physical disabilities by letting users navigate with just a mouse or just a keyboard.
- People with speech disabilities by supporting text-based interactions as an alternative for any voice interactions.
- People with visual disabilities by providing text descriptions of visual content and ensuring your content is clear based on the text alone.

For more examples of barriers people face when using the web, see the Diverse Abilities and Barriers page of the Web Accessibility Initiative (WAI).

12.1.2 Theories of Disability

There are many theories of disability, but here we'll focus on two common ones:

- The medical model treats disability as a condition disabled people suffer from. Disabilities then are problems to be cured, often with adaptive technology. For example, the medical model would view blindness from a cataract as a medical problem to be fixed with cataract surgery.
- On the other hand, the social model treats disabilities as equally valid ways of being. From this perspective, the difficulties a blind person experiences interacting with the world are not because of any medical condition but rather a built environment that is not designed to accommodate them.

In this guide we'll focus on the social model and designing websites to accommodate people who interact with the web in many different ways. However, you should be aware that this is not the only way to think about disabilities.

A Small Reflection

I had been hearing argumenets from both the medical and social model perspectives for years before writing this guide, but I didn't have a good mental model for those ideas until I took COMM 230A. There I came to see the medical and social models as two prominent lenses through which one can view disability. I also came to appreciate how societies in different places and at different times have accepted one of these models more.

Note: If you're a Stanford student, I highly recommend COMM 230A: Digital Civil Society. It discusses the very broad intersection between digital technologies and the civil society discourse around social change. I think that it's important for all of us to understand the social implications of our work, and this is especially true of software development work since technology can scale to affect so many people.

Check out this Mixtape podcast, which includes a discussion on disability theories if you're interested in learning more. It includes strong arguments for both lenses, which illustrates how important it is for us all to be able to see the world through both.

12.1.3 Universal Design

Once we accept the social model, our goal in designing accessible website becomes that of universal design. That is, we want to create a website that is universally accessible no matter how someone browses the web. This may seem like a tall order, but disability rights advocates have worked hard to facilitate accessible design. Because of their work, you can make your website much more accessible using features already built into internet standards like HTML. The WAI has a fantastic guide for developing accessible websites that you should definitely take a look at.

This guide is far from a substitute for the full Web Content Accessibility Guidelines (WCAG), but it will highlight some best practices.

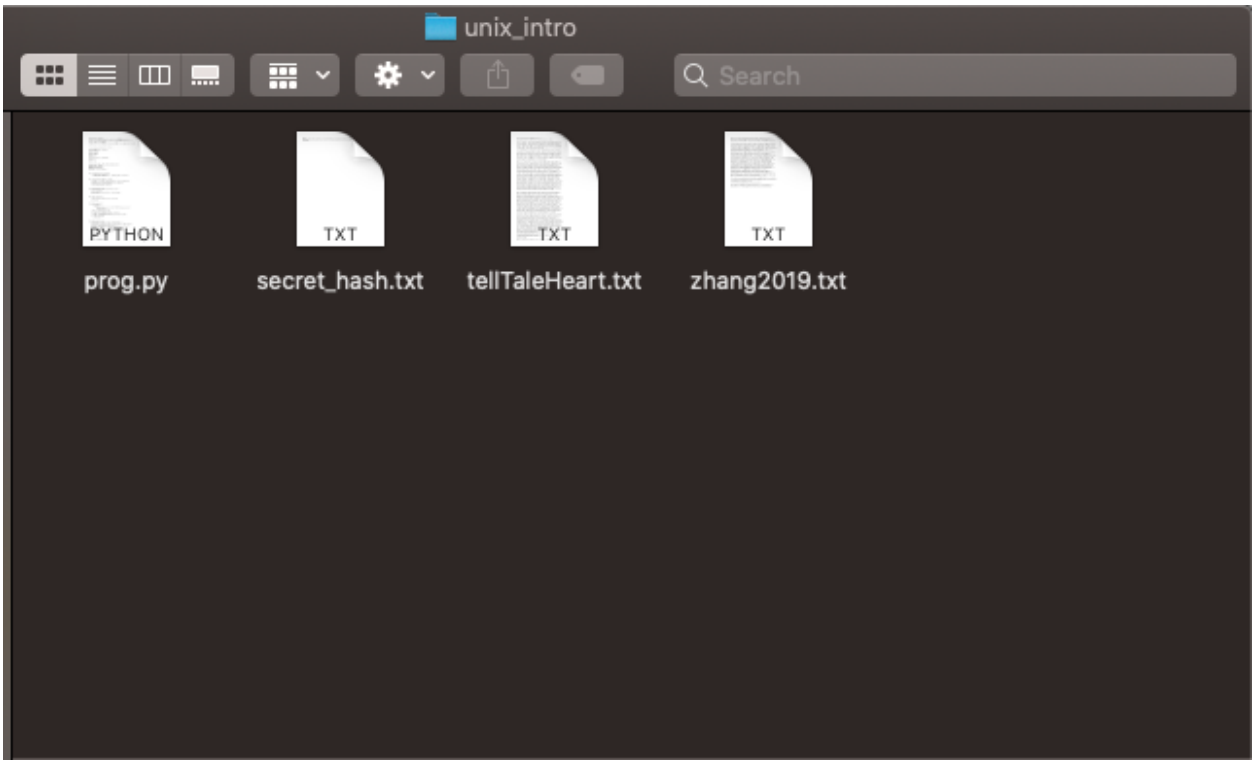
12.2 Examples of Accessibile Web Design

12.2.1 Alternative Text for Images

Whenever you include an image in your website, include alternative text for users who access your page with a sreen reader. HTML supports alternative text like this:

```
<img src="_images/gui_filesystem.png"
     alt="A screenshot of a Mac Finder window showing some file icons within a folder.
↪">
```

This image will be displayed like this:



If you use your browser's HTML inspection feature (e.g. right-click and choose `Inspect Element` in Firefox), you can see the alternative text.

You can find more guidance for alternative text on images in the WAI Images tutorial.

12.2.2 Form Labels

HTML includes a lot of features to help make your website accessible, but you have to use them! For example, you can use `for` and `id` atributes in `label` and `input` elements to help people navigate your forms. For example:

```
<label for="phone">Phone Number</label>
<input id="phone" type="text" pattern="^[0-9-() ]*$" name="phone">
```

Here's what that looks like once the HTML is rendered:

Note: The rendered HTML above might not appear if you are viewing an offline version of this guide. When rendered, the HTML creates an input box next to a `Phone Number` label. If you type letters into the box, the border turns red to indicate invalid text was entered.

The `for` and `id` link the label to the input element to help tell people which label goes with which element. As an added bonus, the `pattern` attribute lets you warn the user if they enter something invalid. This can help users with cognitive or learning disabilities fill out your forms correctly.

See the WAI Labels tutorial for more guidance.

12.3 Conclusion

The standards that underpin the web include many features to make it accessible to disabled people, but it's up to us as developers to actually use them. I hope you leave this guide inspired and empowered to make your websites accessible for everyone.

12.4 Resources

- This Mixtape podcast on artificial intelligence and disability includes a helpful discussion on different theories of disability and how they have waxed and waned over time.
- The Web Accessibility Initiative (WAI) has great resources to help you make your website more accessible.

12.5 Licensing and Attribution

This guide was informed by the World Wide Web Consortium (W3C) Web Accessibility Initiative (WAI) Developing for Web Accessibility guide and Diverse Abilities and Barriers page.

This work was initially created for a workshop at Stanford Code the Change and as a project for COMM 230A at Stanford University.

CHAPTER

THIRTEEN

LEADING INDIVIDUAL FEEDBACK SESSIONS

13.1 Introduction

13.1.1 Audience

This guide is aimed at team leads, but that's not to say that it's not helpful for everyone to think about doing real-time feedback. It's how we get better!

13.1.2 Why Do Real-Time Feedback?

Feedback in general is a critical part of learning. Leading without getting feedback would be like doing problem sets and exams without getting a grade, written feedback, or an answer key later. Sure you might learn a little just by trying, but you'd learn a whole lot more if you had a sense of what you were doing well and what could be better. That's what feedback is!

Feedback forms are great, but I find that real-time feedback sessions are invaluable for learning about how your members are feeling and generating ideas for how to do things better. I think that the more you can build conversations with your members, the better.

In my experience, if you form good working relationships with your members, they will actually by quite candid with you even if there's no anonymous feedback form. Conversations like feedback sessions also help build this relationship by showing that you care about making their experience better.

13.2 How to Lead a Feedback Session

The first rule for this is that every team is different, and you know your team best. Therefore, you should absolutely adapt and ignore this advice as appropriate for your team. That said, here are some ideas for leading a feedback session that have been useful for me.

13.2.1 Conversation Outline

- Open by thanking them for the work they've done or expressing your appreciation for their contributions. This kind of arranged, individual feedback conversation is probably new and strange for them, so it's helpful for you to open and set the example and tone. Try and make your comments as specific to the individual as possible, and model the kind of feedback you hope they give you. (Only model the positive though, because critiques at the start can be weird.)
- Start narrow and concrete
  - What have you been doing well as a team lead?
  - What could you improve upon?
- Broaden slightly to the team
  - How do they feel about progress so far?
  - What is the team environment like? Collaborative? Fun? Stressful? Annoying?
  - What do they like about the team? What do they wish were different?
- Broaden to the club
  - What about the club do they want to continue? What would they change?
  - How is the club environment?
  - What are the club officers doing well?
  - What could the club officers to better?
- Reflect on themselves. This is out of order in the progression, but it's often the hardest to talk about.
  - What are they proud of?
  - What would they like to target for improvement? How might you be able to help them with that?
- Wrap up
  - Do they want to continue next quarter?
  - Anything else?

13.2.2 After Getting Feedback

During Conversation

Be careful about pushing back, even on feedback you disagree with. It's fine to tell the member you disagree, but you should make clear that you are still taking their opinions into account.

You should also probably take notes on what your members say. You don't just want to remember general themes, you want to remember the specific suggestions your members make so that you can act on them intentionally.

After Conversation

Short Term

After you've gotten feedback from your members, take some time to reflect on what they said. Discuss what you learned with other team leads and club officers, and identify action items you want to work on.

Long Term

Turn the feedback into changes and action. Sometimes these will be super simple changes, for example one of the suggestions I got once was to have 3 1-hour sessions a week instead of 1 3-hour session. I implemented that change a few days after the call.

Other times, you'll get feedback that takes longer to act on. For example, I have heard that the club could benefit from more cross-team social interactions. That's what drove work on having more social events in the following quarters.

13.3 Closing Thoughts

Throughout this process, keep the club officers and other team leads in the loop about how things are going. The club officers in particular are happy to help!

13.4 Licensing and Attribution

This work was initially created for Stanford Code the Change.

CHAPTER

FOURTEEN

TEAM LEADERSHIP GUIDE

This guide contains advice, resources, and suggestions for leading teams at Stanford Code the Change. This guide is by no means complete (and it never will be), so please add your own knowledge to it so we can all learn to be better leaders!

14.1 General Leadership Principles

These ideas are applicable to leading in many contexts, including CtC teams.

14.1.1 Listening

Sometimes it might seem as if "listening" is trumpted as the cure for everything, but I think it is critical for leaders to be acutely aware of and sensitive to their team members' needs, triumphs, and feelings.

Needs

Many of my mistakes in leadership were the result of underestimating my team's needs or overestimating their abilities. For example, I have given my teams too much autonomy and responsibility before they were ready for it, and the result was confusion and wasted time.

To avoid this, you first need to be aware of your members' needs. This might mean starting off each meeting checking in on how they are doing, both with respect to the project and more broadly as a person. If they tell you they are confused or lack direction, *listen* to and take seriously that feedback. For more long-term needs, I like to have quarterly feedback forms and live (e.g. in person or over the phone) conversations with each member to gauge what they are comfortable with and what they need help with. Once you know your members' needs, you can take actions to address them.

Triumphs

Be wary of only thinking about the negatives–what your members are *failing* to do because of unmet needs. Also pay attention to when they succeed. When a member submits a pull request, for example, make sure to comment on the things they did well, not just the problems that need fixing. You don't want to overwhelm them with negative feedback, and knowing what to do again is also super helpful. Also, you want to know not only when you are not giving enough help, but also when you are providing too much help. As teams grow more confident and competent, they will be able to move faster and with less guidance. Let them! Your team won't be able to get better if you're micromanaging the details they already have mostly handled. They instead need you to be exposing them to bigger-picture ideas like project planning and module organization.

Feelings

Even in a technical team, each member's emotional state has direct impacts on the team's work. Stressed-out members, interpersonal conflicts, and a loss of motivation can all prove fatal to a team, so you should pay attention to them. For example, once a team lead was worried about one of their members not feeling like they belonged in the club. Their concern is why our feedback surveys ask about belonging now, and it was also a driving force behind our efforts to recruit members with backgrounds that aren't well represented in our club.

Response to Feedback

It's not enough just to listen, though. You also have to *do* something with what you learn. In particular, it is often helpful to be open about the actions you are taking in response to feedback so that people know their suggestions will go somewhere. Be careful though not to violate the implicit confidentiality of feedback given in private. Breaking the trust of your members can also destroy a team.

14.1.2 Situational Leadership

This section on situational leadership is adapted from the training trip leaders receive from Stanford Outdoor Education (SOE).

Leadership Styles

Different styles of leadership are appropriate at different phases of group development and in different situations. In general, for a given problem, leaders progress through these styles sequentially as their groups mature. The more critical or difficult a problem, the more slowly a later leadership style will become appropriate. To take an example from Stanford's Outdoor Education program, trip leaders adopt a more relaxed leadership style (delegating) quite early on for trivial decisions like what music to listen to. On the other hand, leaders take much more control (directing) in emergencies regardless of the group's maturity or experience.

- **Directing**: A directive leader is more authoritative. They direct members on what to do next, and they provide near-constant oversight and guidance. While this leadership may seem over-bearing, it's actually critical when groups are new or inexperienced. These kinds of groups are often comforted by knowing (a) that someone is in charge and (b) exactly what they are supposed to do. Setting clear expectations and a good example are critical here because this early stage is when the tone and culture of the team are set.
- **Coaching**: As teams become more comfortable in their work, a leader can step back a bit and let members learn by doing. Teams are generally not yet able to work on their own, but they are comfortable enough to try. Here, leaders should support that desire to try and learn. Members will still need lots of guidance, but they also now need to see the big picture–the why–to motivate their work. Leadership here should focus on being transparent in how decisions are made, answering questions, reinforcing good practices, and pointing out ways to improve. Note that the big difference between directing and coaching is not in ability but rather in comfort. Members probably aren't that much more capable than in the directing phase, but they're comfortable enough to play a more active role by trying and learning.
- **Supporting**: This is where a group's capabilities grow. Members are beginning to have the skills to work independently, but they haven't mastered them (not that any of us have!), and they still need help with tricky parts. After setting the initial direction, a leader here can often step back and provide targetted guidance and help as needed. Now a leader is more focussed on encouraging and participating alongside members rather than being apart from the other members. A leader here will still need to step in sometimes to help though; it's just that that intervention isn't constant anymore.
- **Delegating**: This is when a group is essentially autonomous and a leader can pretty much step back and let members do their own thing. This can be immensely empowering for the group, but it can also end badly if the group runs into problems or disagreements without a leader to step in. Here a leader is more of an equal member

of the group, but they are still watchful for potential problems that may need them to step in and assert more control.

This section on leadership styles was adapted from page 5 of the Outdoor Leadership section of the SOE 2017-2018 Field Guide.

Self-Awareness

The example you set as a leader is one of your most effective leadership and teaching tools. Especially with new groups, group members tend to immitate the example set by their leader. Therefore, make sure you are engaged, courteous, and respectful whenever interacting with group members. That can make the difference between someone feeling included and feeling like they don't belong.

Learning who you are as a leader comes largely with practice, but here are some suggestions:

- Be mindful of your own emotional state. If you're frustrated, it's likely your team will pick up on it. This is especially important when resolving conflicts, as you can't diffuse tension if you're not calm yourself.
- Keep a close watch on what you find challenging as a leader. It might be that you have a hard time giving directions, accepting criticism, or responding to questions you don't have an answer to. Notice them and focus on improving them.
- Especially with new groups, don't be afraid to be "in charge" and directive. It can be difficult to give instructions to a bunch of your peers, but your team needs you to lead.
- At the same time though, be attentative to how your team is feeling. You also don't want to be demanding or establish a poor relationship with your members. Leadership requires mutual trust.
- There's a balance to be struck between giving groups instructions (e.g. "go learn this framework using this website") and letting them learn on their own ("here's a problem, see if you can figure it out with Google"). The former is important for setting a good technical foundation, but the latter is way more fun if a team is ready for it. There'll probably be some trial and error here as you figure out what each of your members is ready for.

14.1.3 Conflict Resolution

Conflicts will eventually arise in any group, and they will sometimes become disruptive. Leaders need to step in to resolve those disruptive conflicts, and it may be appropriate to also resolve conflicts that have the potential to become disruptive.

The EAR Method

When reading this method, you might feel like it's an over-the-top response to the kinds of simple disagreements we encounter everyday. However, consider what you do when disagreeing with someone. You probably listen to their position, explain what you want, and try and reach some common understanding. The EAR method is just a formalization of that technique. Why bother with the formality? As a leader, the pressure to "lead" and fix the problem can destroy the calm and empathy dispute resolution requires. Following a formalization, even loosely, can help you cut through the pressures and clear your judgement.

Use the mnemonic "EAR" to remember these steps:

- **Listen In**: Take a moment to evaluate your own mental and emotional state. Are you calm? You'll need to be composed and level-headed to resolve the situation. Stepping in when you're angry or upset could inflame tensions even more.
- **Listen Out**: Evaluate the situation before stepping in. What is the context within which the conflict has arisen? Has the team been frustrated working through a difficult task? Is everyone stressed about their classes? You won't be able to figure out the cause of the conflict yet, but the "environmental" factors you notice can help you

understand later on. (This is unlikely to be an issue for CtC, but in other contexts, like SOE, you might also need to consider whether the conflict poses safety issues and therefore necessitates an immediate resolution.)

- **Express**: Let each party to the conflict express their view of the situation. It is important for everyone to know that their views and position are being heard, so everyone needs to actively listen. This is also a time to diffuse the emotion of the situation, so parties should be asked (and reminded if necessary) to speak only for themselves and avoid insults. "I" statements can be helpful here (e.g. "You're being lazy" becomes "I feel like I'm doing more work than everyone else, and I don't feel that's fair"). Everyone, including you, should be *listening* here, not judging.
- **Address**: You summarize the situation, including everyone's concerns, out-loud to the group. Try to be as fair and neutral as possible here, as the point is to make sure everyone understands what the problem to be addressed is. It also serves to make the conflict a problem to solve, so the group should also discuss potential solutions. The point here is to figure out a solution that satisfies everyone's needs, not to pick a winner and put the blame on a loser.
- **Resolve**: The leader now suggests or adopts a plan to resolve the conflict. It may require a compromise where each party has to give something up, but it should be neutral and rational. The group then discusses and tweaks the plan so that everyone feels as good about it as possible. It is important for everyone in the group to accept the plan.

Thank you to Sam Spinner for teaching me this method during an SOE training trip. It is adapted from the curriculum used by the Boy Scouts of America. See this link for more information.

14.1.4 Building Community

This section will focus on ideas for promoting a few selected values that you might want to promote in your team.

Collaboration

Consider the challenges to collaboration for a new team. Your members probably don't really know each other, and at least some of them are probably newcommers to some of the technologies you will be using. To build a collaborative team environment, you will probably need to address both of these barriers.

Helping Strangers Collaborate

At its core, this is just a question of how to help strangers get to know each other. (Let's assume you're not trying to promote a more distributed open source model where collaborators don't actually need to know each other.) Doing introductions and talking with your members will help. Your example as a leader can be very powerful here, for example by addressing your members by name and chatting with them. You'll probably have to do introductions multiple times, as your members are unlikely to memorize names right away. Another strategy is to get your team to work in pairs, perhaps on their first tasks. This gives them a chance to get to know one other member in more depth, and it has the added benefit of making the task less intimidating. (Also, it's harder to show up without having done anything when you have a partner relying on you!)

Helping Newcommers Collaborate

A common motif in new teams is your team having pretty much no idea what you're saying but none of them wanting to seem ignorant by asking. However, you need them to understand what the basics for them to be able to start working on anything. One solution to this conundrum is to directly teach the basics and do so at a *very* elementary level. Remember that you probably have some members with next to no experience. Be wary of using jargon like "API", "frontend", "token", "framework", etc. One technique is to, as a first "assignment," have your team pick various technologies you'll be working on to learn about. (Make sure everything important gets covered either here or by you!) Then, they can all present their findings at the next meeting to help bring everyone up to speed. If all goes well, most of your meeting will be answering questions as your team learns.

Support

Your team will probably be much more effective if members support each other and feel supported. This has many dimensions, from the emotional to the technical. Here, we'll focus on building a culture of technical support. Overt encouragement can help here, for example by encouraging people to send questions they have or problems they run into to the whole group (e.g. via Slack). You can also set an example of being available to help and quickly responding to member questions and problems. This might mean searching the documentation for someone's question, or it might mean meeting up with someone to debug their code. As time goes on, hopefully other members will start taking over and helping each other.

14.2 Licensing and Attribution

This work was initially created for Stanford Code the Change.

CHAPTER

FIFTEEN

LICENSE AND ATTRIBUTION

These guides were created by the amazing members of Stanford Code the Change. The authors of each guide hold its copyright, and any licensing terms are specified in the guides themselves.

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- modindex
- search