Week 09
Apache2 and Flask
Making a Professional Webserver

Melvyn Ian Drag

December 22, 2019

Abstract

Last class we took a look at the toy 'SimpleHTTPServer' in Python3. Tonight we'll use a real website backend - Flask - and we'll server content on a real server - Apache2. As before, we'll make **GET** and **POST** requests using the command line tool cURL.

1 Meta Data

Still, we'll try and make tonight as 'stand-alone' as possible. If you missed last week, you ought to read the notes to catch up on what curl is and how to use it.

2 What's a Webserver?

The term is overloaded. A server is a computer, connected to a network, that other computers can access. There may be more ways to define it, but that's what I think of. But for your midterm I had you install some postgresql server stuff. In that case I was referring to the database server software. And tonight I'm going to refer to Apache2 as a 'webserver' - it's actually software that runs *on* a webserver, which is the machine. It's just that the terms are overloaded.

3 What's Flask?

Flask is what is called a **Back End Web Framework**. It is sort of like a **Library**. Libraries are bits of code that you can use in a language to solve a particular problem. For example, if you know Java you've probably used the *java.util* library for things like *ArrayList*s. In Python last week we used a library called *SimpleHTTPServer* to serve up some webpages for us on the address *localhost:8000*. A **Back End Web Framework** is like a library, but the purpose of this code is to respond to HTTP requests against a web server. If that sounds confusing, don't worry! Just continue through these notes and by the end you'll have a notion of what I'm trying to say here.

4 What's a REST API

Too many buzzwords out there. REST is an acronym. You can read about it more if you want. For the rest of the lecture just understand that you usually pass JSON data back and forth over a REST API. We'll talk about JSON later.

4.1 Job interview prep

Interviewer: *I see you know a bit about Linux webserver maintenance. Do you know what a REST API is?*
You: Honestly, not too much because I'm not a webdeveloper. But I do know that REST APIs are very popular now, and they typically involve the exchange of JSON data. JSON data is when the data is structured like a Python dicitonary, or a Java Hashmap - its a bunch of key-value pairs in curly braces.

If you can give the above answer, you'll know enough for now.

5   What we'll do today

Today we'll bring a real website online. Last week we made a trivial website with Python's SimpleHTTPServer, but today we're going to use a real, professional-grade server (Apache2) and a real backend web framework (Flask). There are other servers like NGINX and there are other backend webframeworks like Django, Play, Spring, Ruby on Rails, etc., I've just chosen this pair because it's the easiest to code of the few things I know. Spring on NGINX would be considerably harder to configure.
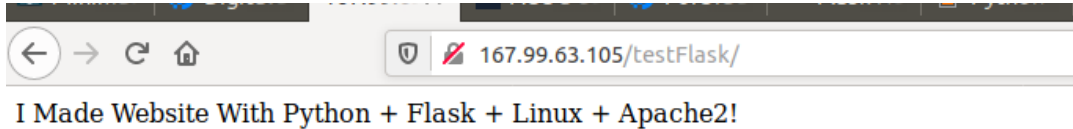

Figure 1: A website running on my server at IP address http://167.99.63.105/testFlask/ . This is the first website we will build together today. It will run on Linux using Python, Flask and Apache2

6   Setting up server

So let's begin building our Flask + Apache2 website. Get a debian 10 server on digital ocean. Then run these commands:

```
1   root@machine$ apt update
2   root@machine$ apt install apache2
3   root@machine$ apt-get install libapache2-mod-wsgi-py3 python3-dev python3-pip
4   root@machine$ pip3 install flask flask-restful
5   root@machine$ apt install curl
6   root@machine$ service apache2 status
7   # should report that apache2 is running
8   root@machine$ curl localhost
9   # lots of data
10  root@machine$ curl localhost > curlResponse.html
11  # dump the data to a file so it's easier to look through
12  root@machine$ less curlResponse.html # look through, verify you have the default apache page. That means
        the server is running.
13  root@machine$ curl ipinfo.io/io # get your server ip address
```

Open a browser on a laptop or computer. Go to the server ip address, make sure the server is running. You should see the same default apache page you saw with curl localhost. But now you are looking at it through a browser on a different machine (you ran curl on the webserver itself).

So we've installed all the software we need! That was easy. Now we can build a little website and then connect the website code to the apache server we just installed and activated.
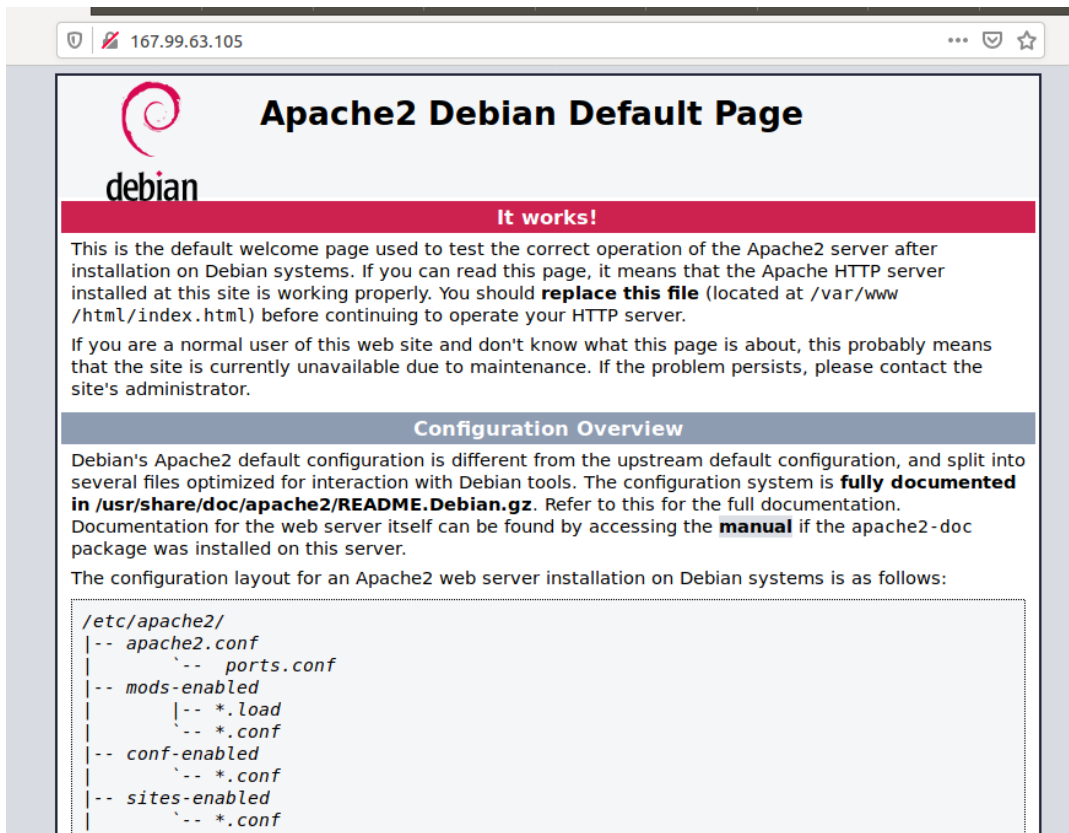
Figure 2: default webpage provided by a fresh Apache2 install

Warning! Make sure you have succeeded in this first task. If you haven't, you won't be able to do the rest of the exercises. If you can see the default Apache webpage, then continue on.

7 Creating a Flask application

The first thing we need to do is create a non root user for managing this stuff. We'll give the user sudo permission for the few root things we need to do to bring the website online. I've created user 'webdeveloper'. You should do the same so you can copy and paste the code I've provided. If you create a different username, you'll need to make the minor modifications to make everyting match up.

```
1  root@machine$ adduser webdeveloper
2  root@machine$ usermod -a -G sudo webdeveloper
3  root@machine$ sudo su - webdeveloper
4  root@machine$ mkdir /home/webdeveloper/ExampleFlask
```

Inside the ExampleFlask directory, put the files described below. Note that these files are available in the Example1/ExampleFlask directory in the class Git repo.

7.1 __init__.py

This is an empty file

7.2 my_flask_app.py

```
1  from flask import Flask
2  app = Flask(__name__)
```

3

```
3   @app.route("/")
4   def hello():
5       return "I Made Website With Python + Flask + Linux + Apache2!"
6   if __name__ == "__main__":
7       app.run()
```

7.3   my_flask_app.wsgi

```
1   #!/usr/bin/python3
2
3   import logging
4   import sys
5   logging.basicConfig(stream=sys.stderr)
6   sys.path.insert(0, '/home/webdeveloper/ExampleFlask')
7
8   from my_flask_app import app as application
9   application.secret_key = 'anything you wish'
```

7.4   Verify your configuration

By this point you should have these three files on your machine. I don't expect you to know Python, so I've provided these files online in the class repo under the Example1/ExampleFlask directory. You can copy and paste, wget, etc. to get the files, whatever you like. In the end, you need to see this on your computer:

```
1   webdeveloper@machine$ cd /home/webdeveloper/ExampleFlask
2   webdeveloper@machine$ ls
3   __init__.py my_flask_app.py my_flask_app.wsgi
```

8   Connect Flask to Apache2

Now we need to wire up our Flask application to the Apache webserver software. You will need to know your machine's ip address. I showed you a website you can curl that will tell you your ipaddress

```
1   user@machine$ curl ipinfo.io/ip
2   # returns your external ip address
```

Then you need to create this file, and put the proper ip address. Note that you'll need to run vim with sudo as this is a privileged file.

8.1   /etc/apache2/sites-available/ExampleFlask.conf

At this point, you need to create the file */etc/apache2/sites-available/ExampleFlask.conf*. The contents of the file are shown below. Make sure you change the ip address in the file to the ipaddress of your machine.

```
1   <VirtualHost *:80>
2       # Add machine's IP address (use curl ipinfo.io/ip)
3       ServerName 167.99.63.105
4       # Give an alias to to start your website url with
5       WSGIScriptAlias /testFlask /home/webdeveloper/ExampleFlask/my_flask_app.wsgi
6       <Directory /home/webdeveloper/ExampleFlask>
7               Options FollowSymLinks
8               AllowOverride None
9               Require all granted
10      </Directory>
11      ErrorLog ${APACHE_LOG_DIR}/error.log
12      LogLevel warn
13      CustomLog ${APACHE_LOG_DIR}/access.log combined
14  </VirtualHost>
```

8.2 Turn on and Test the Website

Run these commands as the user 'webdeveloper'

```
1  webdeveloper@machine$ sudo a2ensite ExampleFlask
2  webdeveloper@machine$ sudo a2enmod wsgi
3  webdeveloper@machine$ sudo service apache2 restart
```

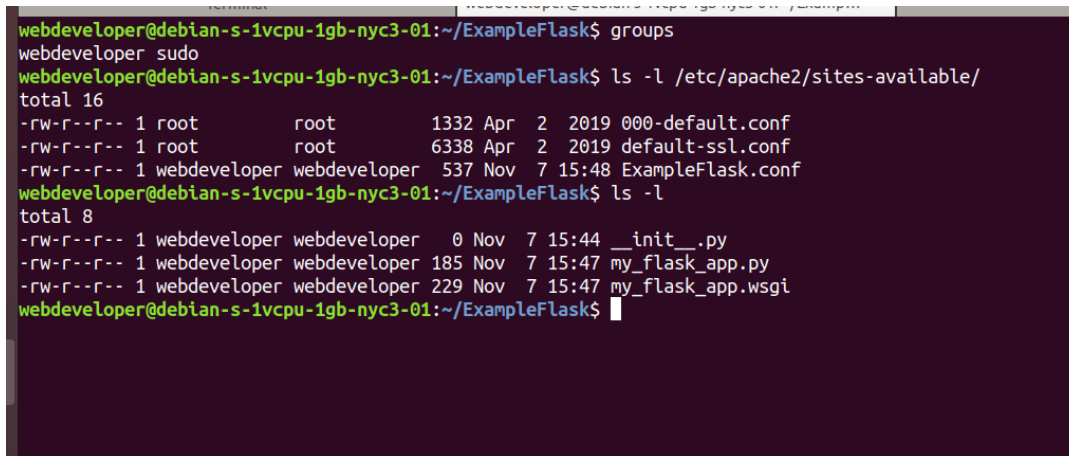Then, open a browser on your laptop or the NJCU PC in front of you and go to

$$my.ip.addr.ess/testFlask$$

You should see a message saying you've made your first website with Linux, flask, apache and python. You can share the link to show off your website to your friends and family! You could now also go to godaddy or namecheap, buy a domain name, and wire that up to your server so people can go to website.com instead of a scary looking ip address.

9 Additional Considerations for Example 1

So we have a simple flask app running on apache now. As you've probably seen by now, Linux servers are very particular about groups and permissions. I'm not sure about all of the ins and outs of the permissions for Flask and Apache. I'm showing the permisisons and groups I've set up on my machine. If you do something else I'm not sure it will work, and I don't know why it owuld be broken. We could experiment with permissions if you have something else.

I'm also not sure about all the details of the ExampleFlask.conf file. There are many options to it. I've always set mine up by scouring the internet for information and banging in what ever details I found. I hope to one day understand this file in depth, but for now all I know is how to make it work.

Figure 3: My permissions and groups on a successful install

Wait for class to all be caught up and have their servers running

10 About the code for Example 1

This isn't a Python class as we said, but I'll just mention a few things about the Python code so you get some sense of how it's working. You will see the line

```
1  @app.route("/")
2  def hello():
3    return "some string here"
```

This says that when we GET the location "/" in our domain name, the webserver will respond with the return value.

We have to go to "ipaddress/testFlask" to see this data because in our config file we have the line

```
1  ...
2  WSGIScriptAlias /testFlask /home/webdeveloper/ExampleFlask/my_flask_app.wsgi
3  ...
```

and this registers the base end point for our website as /testFlask. If there's time we can change this. Right now apache is serving the default apache website on /, so we had to put our website at a different location on our server. Well need to delete some config files and change this config file so that we can see our website when we go to "/" instead of having to go to "/testFlask".

The contents of the .wsgi file is boilerplate stuff that won't matter to you unless you want to become a serious python developer. Also, the purpose of the .wsgi file is interesting, but probably won't matter to you. Python applications can't run natively on webservers. So, about 20 years ago, some developers go together and wrote some middleware that allowds Python applications to communicate with webserver software like Apache2. The details are complicated - you can read about it if you're curious.

https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface

I think php can talk right to the webserver, but I'm not sure. I've never written a line of php in my life. All I know is that PYhton needs a thing called wsgi (which we installed at the beginning of this lecture with libapache-mod-wsgi-py3). You'll note that there are different version for python3 and python2.

11   Example2: Adding Endpoints and Serving HTML

To do this

1. Change the ExampleFlask.conf file to the one below.

2. Create the /home/webdeveloper/Example2 directory add the files from the class repo

3. Restart apache2

11.1   New /etc/apache2/sites-available/ExampleFlask.conf

Note that this file now serves two webpages! You can host multiple websites on one webserver.

```
1   <VirtualHost *:80>
2       # Add machine's IP address (use curl ipinfo.io/ip)
3       ServerName 167.99.63.105
4
5       # Some error logging stuff
6       ErrorLog ${APACHE_LOG_DIR}/error.log
7       LogLevel warn
8       CustomLog ${APACHE_LOG_DIR}/access.log combined
9
10      # Give an alias to to start your website url with
11      WSGIDaemonProcess site1
12      WSGIScriptAlias /testFlask /home/webdeveloper/ExampleFlask/my_flask_app.wsgi
13      <Directory /home/webdeveloper/ExampleFlask>
14              WSGIApplicationGroup site1
15              WSGIProcessGroup site1
16              Options FollowSymLinks
17              AllowOverride None
18              Require all granted
19      </Directory>
20
21      # Set up the other website
22      WSGIDaemonProcess site2
23      WSGIScriptAlias /somewhereElse /home/webdeveloper/ExampleFlask2/my_flask_app.wsgi
24      <Directory /home/webdeveloper/ExampleFlask2>
25              WSGIApplicationGroup site2
26              WSGIProcessGroup site2
27              Options FollowSymLinks
28              AllowOverride None
```

```
29          Require all granted
30      </Directory>
31  </VirtualHost>
```

11.2 my_flask_app.py

```python
1   from flask import Flask
2   app = Flask(__name__)
3
4   @app.route("/")
5   def hello():
6       return "I Made Website With Python + Flask + Linux + Apache2!"
7
8   @app.route("/returnsHTML")
9   def secondEndPoint():
10      return "<html><body><h1>A Header!</h1><p>Here is some data in a paragraph!</p></body></html>"
11
12
13  if __name__ == "__main__":
14      app.run()
```

11.3 my_flask_app.wsgi

```python
1   #!/usr/bin/python3
2
3   import logging
4   import sys
5   logging.basicConfig(stream=sys.stderr)
6   sys.path.insert(0, '/home/webdeveloper/ExampleFlask2')
7
8   from my_flask_app import app as application
9   application.secret_key = 'anything you wish'
```

11.4 __init__.py

Blank as before. To be honest I haven't given it thought if this is required or not. Just put the file. If you know python you can remove it and then meditate on why it works / doesn't work if you remove this file.

11.5 Results

Here are some images showing my new site in action. The /testFlask endpoint still works, but now so do /somewhereElse and /somewhereElse/returnsHTML.

Wait for class to all be caught up and have their servers running again

12 Example3: A Rest API

Only do this section if the class is caught up and it's before 9 PM

So far we've returned plain text and text formatted as HTML from our website. A popular thing to do now is to pass around JSON data. JSON is used all over. I showed it to you last week just to wet your feet, but you'll see it everywhere. You'll use it for web programming, internally in Python code, passing data between applications running on a laptop, for exchanging data files with scientists - everywhere. I avoided JSON for a while because I thought it was just a stupid buzz word that the nerds used to show off, but it truly is an ubiquitous data format. Now you've seen it. You'll see it again, mark my words.

So we'll just see how to return json from a Flask application.

Figure 4: somewhereElse now works

Just as we did in example 2, set up your server with Example3 now. Get the Example3 code from github, copy the new ExampleFlask.conf file to the proper location in /etc, create the /home/webdeveloper/Example3 directory, and put the proper code in it.

12.1   The Flask Application: my_flask_app.py

You can skim the code to see what it does. It has some code for handling the GET and POST requests. A GET request will echo back to you the entire contents of the TODOS data it's maintaining. A POST request will add to the TODOS data set. You can verify after POSTing that your data was added by sending a GET request.

```
1  """
2  Stolen from the flask-restful documentation here:
3  https://flask-restful.readthedocs.io/en/latest/quickstart.html#a-minimal-api
4
5  Super simple RESTful API that handles GET and POST requests.
6  """
7
8  from flask import Flask
9  from flask_restful import reqparse, abort, Api, Resource
10
11 app = Flask(__name__)
12 api = Api(app)
13
14 TODOS = {
15     'todo1': {'task': 'build an API'},
16     'todo2': {'task': '?????'},
17     'todo3': {'task': 'profit!'},
18 }
19
20 parser = reqparse.RequestParser()
21 parser.add_argument('task')
22
23 # TodoList
24 # shows a list of all todos, and lets you POST to add new tasks
25 class TodoList(Resource):
26     def get(self):
27         return TODOS
28
29     def post(self):
30         args = parser.parse_args()
```

Figure 5: We can now return HTML from our Flask website too!

```
31          todo_id = int(max(TODOS.keys()).lstrip('todo')) + 1
32          todo_id = 'todo%i' % todo_id
33          TODOS[todo_id] = {'task': args['task']}
34          return TODOS[todo_id], 201
35
36   ##
37   ## Actually setup the Api resource routing here
38   ##
39   api.add_resource(TodoList, '/todos')
40
41   if __name__ == '__main__':
42       app.run()
```

12.2   Accessing Our REST API

Then you can get JSON data in the browser as shown below:

Then, on your laptop, make a GET request with cURL

And, you should also be able to make the following POST request:

```
1   curl -X POST \
2     -H "Content-Type: application/json" \
3     --data '{"task":"Learn Linux in CS407"}' \
4     http://167.99.63.105/myRESTService/todos
```

And then look, after POSTing data, when you GET data you will see the data that you POSTed!
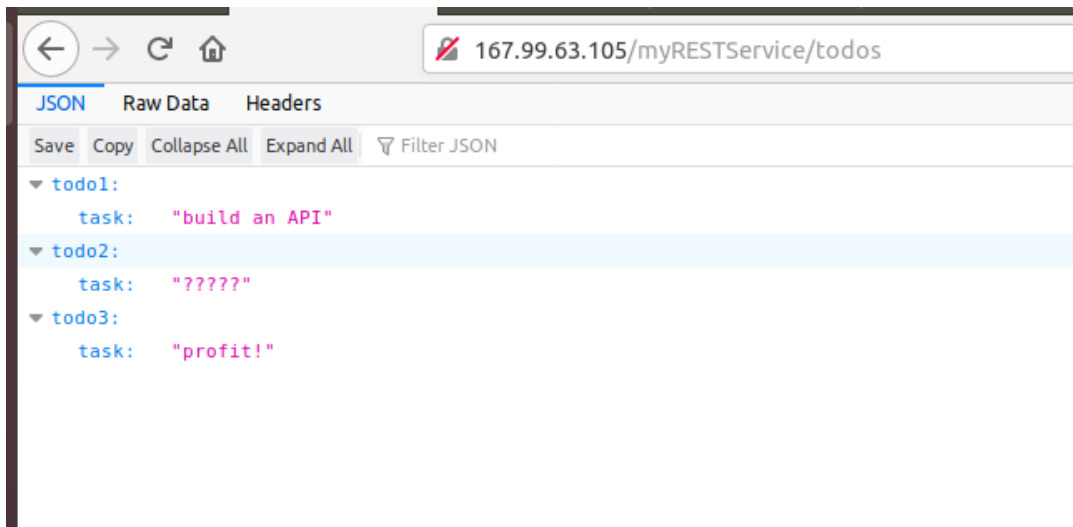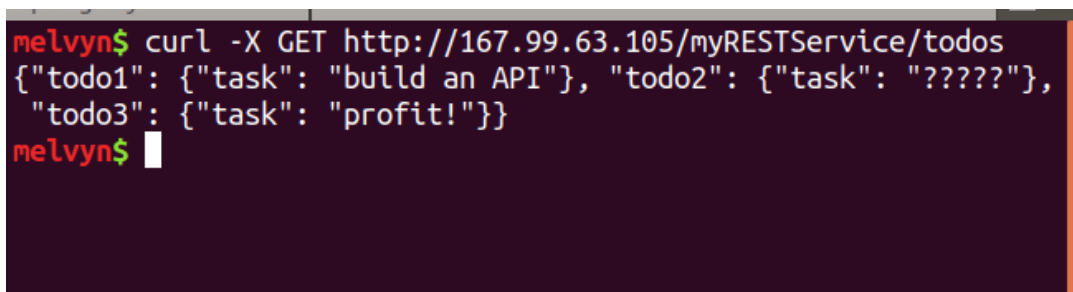
Figure 6: See some json in the browser!

Figure 7: Make a GET request with cURL!

13   Flask and Bigger Websites

Typically your website code will return HTML. I've only shown you how to return HTML strings in example 2, but typically you create HTML templates and populate the template with data. Too complicated to explain, there is no time. Google 'Flask and HTML templates' if you want to know more.

NOTE if class ends early, we can do this together, but I suspect we'll be plenty busy tonight.

14   Databases

For your exam you set up a postgres server with some data in it. I've just shown you how to set up a Flask web application on an Apache server. And you have a good knowledge now about some linux server maintenance things.

In the TODOList REST application we stored all of our data in a dictionary - in the real world this is not how you do it. Can you guess what you do for real websites? You use a DATABASE! You are now poised to go pick up a book on Flask and learn how to wire a database and a website backend together!

Of course we won't do that in this class, but now you have the prerequisite information to go and follow a tutorial on building websites with Flask.

15   sed

sed is one of the more popular command line tools. So far we know grep, vim, cat, mv, ls, etc.. sed is another one of the "big ones".

sed stands for "stream editor" and is a simple little programming language (or tool, depending how you want to view it) that is used for processing text files line by line.

Figure 8: POST data to your website with cURL!

15.1 What to do with sed #1

Sed can be used to replace all occurences of a string in a file. See sedExamples/01.

```
1  user@machine$ cat oldFile
2  user@machine$ sed 's/hello//' oldFile
3  # will replace first occurences of 'hello' with nothing\
4  # note that a line with two occurences of 'hello' will only have the first occurence replaced
5  user@machine$ cat oldFile
6  # the file is not changed. sed did not overwrite the original file
7  user@machine$ sed 's/hello/HELLO/' oldFile
8  # see first occurences of hello were replaced with HELLO
```

these changes were done in place. If you want to create a new file you can do it like this

```
1  user@machine$ sed 's/hello/HELLO' <oldFile >newFile
2  user@machine$ sed 's/hello/HELLO' oldFile >newFile
3  user@machine$ sed 's/hello/HELLO' 0<oldFile >newFile
4  # all of the above are equivalent
```

or you can change the input file in place

```
1  sed -i 's/HELLO/hewwo/' newFile
```

the -i means 'in place'.

15.2 What to do with sed #2

We saw sed can do search and replace first occurence per line. It can also replace all occurrences in a file.

```
1  user@machine$ sed 's/hello/HELLO/g' oldFile
2  #note all occurences of hello now say HELLO
```

by adding the g after the search/replace pattern you can replace all occurences.

15.3 What to do with sed #3

Sed can do regular expressions. If you want to capitalize any 'h' at the start of a line, you can do this:

```
1  user@machine$ sed 's/\(^[h]\)//' oldFile
2  ello world
```

11

```
3  ello world world
4  world hello hello world
```

15.4   What to do with sed #4

You can also extract patterns. This is a weird thing that you can't always do with things like search/replace in microsoft word/excel/whatever other tools you are familiar with.

the same command as before remembers and stores the search pattern in a variable called \1. You can do the same thing with \2, \3, etc for subsequent search patterns. **The search patterns are the regexs enclosed in \(and \).** In the example above, my regex was looking for a line that starts with h. If you scroll down a bit youll see a regex for a line that starts with an ASCII letter.

```
1  sed 's/\(^[h]\)/\1\1\1/' oldFile
2  hhhello world
3  hhhello world world
4  world hello hello world
```

You may not be mentally prepared to appreciate what I've just shown you? Maybe one day you'll be trying to do a complicated search and replace task and then you'll remember sed!

Check out this interesting example too!

```
1  user@machine$ sed 's/\(^[a-zA-Z]\)/\1\1\1/' oldFile
2  hhhello world
3  hhhello world world
4  wwworld hello hello world
```
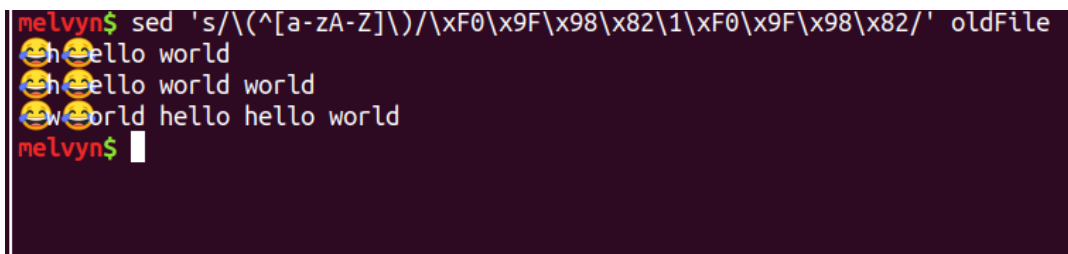
If I want to wrap the first letter of every line with smileys, for example, I can do this:

```
1  user@machine$ # emojis!
2  user@machine$ sed 's/\(^[a-zA-Z]\)/\xF0\x9F\x98\x82\1\xF0\x9F\x98\x82/' oldFile
3  # look what comes out! Depending on your laptop, the output will be different from the output I have as
       can be seen in the figure below
```


Figure 9: How the heck did he do that? If you already know, I'm very impressed

But this may or may not work on your computer, we'll talk about this in depth if there is time this semester.

15.5   It's not Halloween, I don't want to scare you

So far today I've showed you what to type to turn on an Apache webserver and we copy and pasted some Python Flask code. Then I showed you a whole new programming language called 'sed'. Well, sed isn't quite a language. It is and it isn't, depends on what you want to define as a programming language. You've seen it can be used for search and replace. And you've seen that it can use regexes.

Questoin for class: What is REGEX short for?What are some tools you already know that use regular expressions?

Here is a site that will teach you a ton about sed: http://www.grymoire.com/Unix/Sed.html

in fact, you will need to read this website for your homework. Theres so much to know, I've only shown you the tip of the iceberg. You can go now and learn as much as you care to, I think you understand enough to hit the ground running. There is also a super famous book out there about sed & a related language called awk:

You could spend 20-30 bucks here for it: https://www.amazon.com/sed-awk-Dale-Dougherty/dp/1565922255

Or you can take it for free from this class repo! Inthe reading materials directory you will find the book sedandawk.pdf

There is alot to know to be a power Linux user and you already know alot, that's great, good for you!

16 Recap of what you know

1. A bit about the Bash programming language

2. grep & regular expressions

3. What a user is on Linux

4. What a group is on Linux

5. What is git?

6. You're comfortable using a command line interface now

7. There are a few different languages on the command line - we've used bash and dash

8. What permissions are and how to modify them

9. What is a root user

10. What is a process

11. What is a job

12. How to send signals to processes (kill, CTRL+C, CTRL+Z)

13. How to make code ignore or block signals

14. How to install software on Linux

15. A cool trick for using setuid / setgid to make a non-root user do some root stuff

16. basics of relational dbs and how to configure one on Linux

17. What is cron

18. curl

19. some vocabulary like API, REST, regex, SQL, DB

20. A bit about python programming

21. What is a website backend? Set up a website backend with Apache2 + Flask

22. What is sed?

17 Coming Up In This Class

We have a few more important things to cover

1. set up a git server

2. add a gitlab front end to the git server

3. Linux and Text encodings

4. xxd, everyone's favorite binary packet analyzer

5. The AWK programming language

6. The hows and whys of formatting harddrives, usb sticks, solid state drives, etc.

7. Encryption with gpg + pgp. How it relates to ssh and other pub/priv key schemes.

18   References

This lecture was based on what I read on these websites:

`https://www.codementor.io/abhishake/minimal-apache-configuration-for-deploying-a-flask-app-ubuntu-18-04-phu50a7ft`

There were some bugs in the tutorials above that I sorted out to make sure our lecture had a good flow.