# GitHub

<https://github.com>

GitHub is the popular hosting service for development projects, it utilizes the Git revision control package. Users are offered free and paid plans, the idea being that free repositories will be utilized more for open source projects. Currently GitHub is the most popular code repository for open source projects.

As a web development and/or software application development student you should have a GitHub account. It’s a great place to store your code and show it off to the world, even potential employers.

Today, I will be doing a intro to GitHub and show you how to install, configure, and create a local git repository. After, I will show you how to create a repository on Github and push your local repository files to your remote repository. During the process, you will learn how to add files to a commit list, make commits with comments, and how to fork a repository.

### Installing Git:

The first thing we will need to do is install Git. Windows will be our platform and when selecting the download, please select the installer for windows.

1. Go to http://git-scm.com/download/win
2. A download window should appear, click save. If a window does not appear click the “click here to download manually link”
3. After the download has finished, navigate to your download location and double click the ‘Git-1.8.4-preview20130916.exe’ installer.
4. Click ‘Next’ until you arrive at the Select Components screen. You will need to click On the Desktop under Additonal Icons, this will create a desktop icon that we will use later. After you have done that, press ‘Next’, leaving all the defaults. When the install has finished, uncheck ‘View ReleaseNotes.rtf’, then click ‘Finish’
5. Git is now installed, and you should see a Git Bash Icon on your desktop. If you do not see one you can access Git Bash from the start menu by clicking start and typing ‘Git Bash’
6. Open Git BASH, you will see a black screen similar to the windows command prompt or a linux terminal. When using Git Bash, you will need to use Linux/Unix Bash commands. If you are unfamiliar with these commands you can google ‘linux commands’ or use the graphic interface for Git.
7. We will be working in the desktop directory, so you will need to type ‘**cd Desktop/**’, the line should end with ‘~/Desktop’ now. You can check that you are in your desktop directory by typing ‘pwd’ if you are unsure.
8. We will need to make a directory to hold the files we will be commiting to your repository. To do this we can type the command ‘mkdir’ followed by the directory name, for this presentation we will be using FlaskPres as the name of our directory. Type ‘**mkdir FlaskPres**’ now to create this directory. To confirm this directory was create, you can type ‘**ls**’ to list the contents of the directory. You should see a FlaskPres entry, the directory was successfully created.
9. After creating the directory we will need to navigate to it, to do this we need to change directories by typing the ‘cd’ command followed by the directory we want to move into. We want to move into the FlaskPres directory so we will need to type ‘**cd FlaskPres/**’, after entering the command you will notice the line change and will say ‘~/Desktop/FlaskPres’
10. Now we are at the point where we start issuing Git specific commands, first we need to set our name, so that when we make commits, people will know we did it. To do this we will issue the

**git config --global user.name ‘Your Name Here”** command, where you will place your name instead of ‘Your Name Here’

1. Next we need to set your email, the command is similar to the one we just typed.

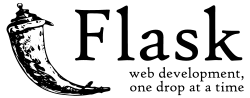
**git config --global user.email “Your\_email@here.com”.** Now we have our email set, so others will be able to see our email when we make commits.

1. The next thing is we need to initialize our directory be a git repository, to do this, issue the command ‘**git init**’. You should see **Initialized empty git repository in c:/Users/cisco/Desktop/FlaskPres/.git/**
2. We will need to create a GitHub account if you have not done so already, to do this you will need to go to <https://github.com> and fill out the form you see, then click ‘Sign up for Github’, You will need to go to your email and click the link in the confirmation email. After that you will be able to access your Github account and create a repository.
3. After activating your account, go back to <https://github.com> and click ‘Sign In’ on the top right, fill in your credentials. Click ‘Login’ when you have filled the fields out.
4. Upon successful login, you will see your Github dashboard, you can explore the options available to you later, but now we will be creating a repository.
5. In the top right, one icon to the right of your username, you will see an icon that looks like a book. You will need to click this, and you will be brought to the screen to create your repository.
6. Under repository name, you will put the name of your repository. We will be using the name ‘**FlaskPres’**, and you may fill out the description field if you wish, but it is not necessary. We will keep it a public repository. All we need to do is click ‘**Create repository**’
7. You will be given instructions on the next screen, this are very informative and will help you in the future, but for now we will disregard them.
8. Now we will return to our Git Bash prompt, the first thing you should do is create a README.md document. This document is where you can describe what your code will do and help others understand what it does. To make this document you can do it through the Windows Explorer interface or type ‘**touch README.md’** in the Git Bash prompt. This will create an empty file called ‘README.md’. We will need to put text in it next
9. To open README.md and place text in it, we can type ‘**notepad README.md’**, this will bring up notepad and allow us to enter text. For this presentation we will just write “**This is a repository that will hold the code for a Python Flask website.”** After you have typed this, click File > Save and close notepad.
10. Now that we have our README.md done, we can add it to the commit list, commit it to our local repository, then push it to our remote repository, GitHub.
11. To add our README.md to our commit list, we will need to issue the command **‘git add README.md’**
12. After adding it to our commit list, we need to actually commit it an add a comment to let others know what the commit was about. To do this we can type’ **git commit -m ‘added README.md**’ You will see something like [master (root-commit) 33c938e] added README.md. This means we successfully made a commit to our local repository.
13. We will next add our remote repository, so that we can branch to GitHub. We will need to enter the command ‘**git remote add origin https://github.com/<username>/FlaskPres.git**’, where <username> is your GitHub username.
14. This is the last part, we will be pushing our commits to Github, all we need to type is ‘**git push origin master**’. You will be asked for your GitHub username and password, enter those and you will see information that says a new branch was created on GitHub. We can see these now by going to **https://github.com/<username>/FlaskPres.git**’, where <username> is your username.
15. Congratulations, you have create a local git repository, added files, made commits, added a remote repository and made a new branch on your remote repository.

### Forking a Repository

You have used Git and GitHub to create your own repositories, but now you may want to use other people’s code and contribute to it, or maybe use it for one of your own projects. We can do this by forking their repository. We will be forking my FlaskPres repository, so we will be ready for the Flask part of my presentation.

1. On GitHub, you will need to go to https://github.com/zero1three/Flask-Pres. Here you will see the files for the presentation. You have multiple options , you can download a zip file of the files, clone it with git, and fork it. We will be forking it, so on the top right below your username you will see a Fork button, click it. This will fork it to your GitHub account.
2. Now you will need to get the files to your local desktop. We can do this through Git Bash prompt by entering the command ‘**git clone https://github.com/<username>/Flask-Pres.git ‘** where <username> is your github username.
3. After it is cloned, it will have a default remote called ‘origin’, that points to your Github, not the original repository. We still want to keep track of the original repository, in case there are updates. We will first need to move into the repository directory by typing ‘**cd Flask-Pres’**
4. Next we need to add the other remote, we need to enter the command ‘**git remote add upstream https://github.com/zero1three/Flask-Pres.git’**
5. Now that we added the original respository, we will need to pull in changes that aren’t in our local repository by typing ‘**git fetch upstream**’
6. Now that you have your remote forked repository, you can start working on it, add files and make commits. After you can type **‘git push origin master’** to push commits to your GitHub.
7. If later the original repository has updates you can pull them down by typing **‘git fetch upstream’**
8. You can also do a **‘git pull upstream master**’, but beware that it will merge the commits without telling you. Fetching and Merging allows you to decide.



# Flask (A Python Micro Web Framework)

‘Flask is a microframework for Python based on Werkzeug, Jinja2 and good intentions’ (Flask)

Flask is a really great web framework for Python, its fun to work with and getting a simple static site up and running takes only a couple minutes. There are hundreds of extensions you can use with Flask, things like login management, object relational mapping, and many more. The current latest release is 0.10, this will be the version we will be using.

Python is the language behind Flask, and my language of choice. Python can be used for anything from system administration, games, to web development and more. It runs on Windows, Linux/Unix and Mac, and other operating systems. I got interested in python my senior year in high school when my friend told me he was playing around with it, so I gave it a try. I didn’t do much then, but it’s really what I like to use now if I can. I hope this part of the presentation gets you acquainted with python syntax and flask as a web framework.

Python’s latest version is 3.3.2, but there are many syntax changes and differences, so many developers still stick with 2.7.5 until they are worked out. Python 2.7.5 should already be installed on the lab machines.

## Setting up a virtualenv

Virtualenv is a module for python that allows you to create isolated python environments. The advantages of this these environments is that you can create separate installation directories, with different libraries. Maybe you are developing an application that uses a certain version of a library and changing or updating this library may make your application not work anymore. A virtualenv environment can separate that application from others and use it’s own libraries and files.

We will be setting up a virtualenv in our **‘Flask-Pres’** directory, but first we need to install a couple utilities to make installing virtualenv and others libraries we will be using much easier.

If you go to <http://www.yzguy.com/python/ez_setup.py>, you can download a python file to run an installation to install easy\_install. We will be using easy\_install to install pip, which we will use to install flask and our virtualenv along with a few other libraries.

Download ez\_setup.py to the desktop, then shift + right click the desktop and click “**Open a command window here”** this will open the command prompt already in the desktop directory.

We will need to check that python is installed and that it is in the PATH environment variable so we can run it from the command prompt. Type **“python”**, if you see “Python 2.7.5” you are in the python shell. This means python is not installed and/or it is not in the PATH variable.

If you get a “’python’ is not recognized as an internal or external command…. Message we will need to do some troubleshooting. First we will open My Computer and double click on the C:\ Drive, if you see a Python27 directory, that means python is installed. If you do not you will need to visit <http://www.python.org/download/> and download the Python 2.7.5 Windows x86-64 installer and run it. Then repeat the previous steps to see if it is in the PATH variable.

If it is not recognized as a command we will need to edit our PATH variable and add it so that we can run it from the command prompt. To do this we need to enter the control panel, and click “system”, where we need to click “advanced system settings” on the left, and click the “**Environment Variables”** button, which will open a window. We will need to scropt in the System Variables list till we find PATH or Path, click it in the list, then click **Edit.** It’s very important you don’t delete anything in this new window, we just need to add to it. “The PATH environment variable that specifies a set of directories where executable programs are located” (Wikipedia). We need to add the path to Python.exe and it’s scripts. Go to the end of the “**variable value**” textbox. If there is not a “;”(semi colon) terminating the line we will need to add one to separate the paths. We next need to add C:\Python27\; C:\Python27\Scripts; to the end.

After adding the new paths, we can hit **OK**, close System properties, and the control panel. We will need to close out of our already opened command prompt by hitting the X in the top right or typing **exit**.

We will **shift + right click** and click **“Open a command window here**” again to open the command prompt, now we will type **”python”** and we should be brought to the python shell. Yay, now Python works.

Now we can run our ez\_setup.py script and have easy\_install ready. Go ahead and type **“python ez\_setup.py”.** This will run the script and you will see the output, it’s just installing easy\_install and when it’s done we will be able to type again. We need to check that easy\_install is installed and working, we can do this by typing easy\_install. You will get a “**erro: no urls, filenames,….**” Message. That means easy\_install is working.

The next step is to install pip, which is a package manager for python. Easy\_install is also a package manager, but it is not as full featured as pip. We will now enter the command **“easy\_install pip”,** we will see some more output, and when it’s done we will type pip. We should get a bunch of output telling us the options that are available to use with pip. Good, now pip is installed and we can install virtualenv.

We want to install virtualenv system wide, so we will issue the command **“pip install virtualenv”**, we will see output and it will tell use virtualenv is installed successfully.

Great, now we can make isolated python environments.

We should still be in our desktop directory, we need to move into our Flask-Pres directory and create our virtualenv.

Type **‘cd Flask-Pres/’**, and you will move into that directory. Now we need to make our virtualenv environment, we will be calling it **‘venv’**, to do this we type **‘virtualenv venv**’, we’ll see output and that it is copying python executable and other things into the virtual environment.

When it’s done we will have our virtual environment, but we need to activate it. To do this on windows we need to type the name of the virtualenv. Ours is ‘venv’ so we will type **venv\Scripts\activate.** You should see a (venv) before you prompt now, that means we have activated our virtualenv and now are “in it”. We can now install our libraries.

We will need the following libraries: **flask, flask-wtf, flask-sqlalchemy, flask-login**

To Install them we need to issue these commands:

**Pip install flask**

**Pip install flask-wtf**

**Pip install flask-sqlalchemy**

**Pip install flask-login**

Each will take a little time to complete, and when they are done you will see the library name has installed successfully.

Great! Now we have our virtualenv, our libraries. We can start our journey with Python and Flask Web Framework!

I will be going over the syntax and everything during the presentation so I will just post the final script here so you will have it for reference in case you miss something or need help.

Here is also a link to download the whole project (minus the virtualenv) from my website:

<http://www.yzguy.com/python/flask-pres.zip>

'Imports for Flask, Flask-WTF, Flask-Login, Flask-SQLAlchemy

from flask import Flask, render\_template, url\_for, redirect, flash, session, request

from flask.ext.wtf import Form

from wtforms import TextField, PasswordField, SubmitField

from wtforms.validators import Required

from flask.ext.login import LoginManager, login\_user, logout\_user, current\_user, login\_required

from flask.ext.sqlalchemy import SQLAlchemy

'Create instance of Flask Class

'Secret Key for Sessions

'Config variable for database URI (Relative Pathname)

app = Flask(\_\_name\_\_)

app.secret\_key = 'flask presentation'

app.config['SQLALCHEMY\_DATABASE\_URI'] = 'sqlite:///pres.db'

'Create LoginManager object

'Initialize it

login\_manager = LoginManager()

login\_manager.init\_app(app)

'Make SQLAlchemy Object

'Initialize it

db = SQLAlchemy()

db.init\_app(app)

db.app = app

'user\_load callback, used to reload the user object from

'the user ID stored in the session

@login\_manager.user\_loader

def load\_user(userid):

return db.session.query(User).get(userid)

'Class for LoginForm, Flask-WTF uses classes for forms

class LoginForm(Form):

username = TextField('username', validators=[Required()])

password = PasswordField('password', validators=[Required()])

submit = SubmitField('submit')

'When the user form is created it has function get the user object from the db

def get\_user(self):

return db.session.query(User).filter\_by(username=self.username.data).first()

'Class for RegisterForm

class RegisterForm(Form):

firstname = TextField('firstname', validators=[Required()])

lastname = TextField('lastname', validators=[Required()])

email = TextField('email', validators=[Required()])

username = TextField('username', validators=[Required()])

password = PasswordField('password', validators=[Required()])

submit = SubmitField('submit')

'Class for the User Model

class User(db.Model):

id = db.Column(db.Integer, primary\_key = True)

firstname = db.Column(db.String(64))

lastname = db.Column(db.String(64))

email = db.Column(db.String(120), unique = True)

username = db.Column(db.String(64), unique = True)

password = db.Column(db.String())

def \_\_init\_\_(self, firstname, lastname, email, username, password):

self.firstname = firstname

self.lastname = lastname

self.email = email

self.username = username

self.password = password

def \_\_repr\_\_(self):

return '<User %r>' % self.username

'Required method for your User Model when using Flask-Login

def is\_authenticated(self):

return True

'Required method for your User Model when using Flask-Login

def is\_active(self):

return True

'Required method for your User Model when using Flask-Login

def is\_anonymous(self):

return False

'Required method for your User Model when using Flask-Login

def get\_id(self):

return unicode(self.id)

'Declare a route of / and /index, if a user goes to

'these directories Flask will run the function for that route

@app.route('/')

@app.route('/index')

def index():

'returns a rendered template back to the user

return render\_template('index.html',

title = 'Home')

'Another route, this time with GET and POST methods, this is our

'register form, it posts back to itself. If the form is validate,

'it creates a new User object and adds it to the database and

'redirects the user to the index page

@app.route('/register', methods=['GET', 'POST'])

def register():

form = RegisterForm()

if request.method == 'POST':

if form.validate\_on\_submit():

newuser = User(form.firstname.data, form.lastname.data,

form.email.data, form.username.data, form.password.data)

db.session.add(newuser)

db.session.commit()

session['email'] = newuser.email

return redirect( url\_for('index') )

elif request.method == 'GET':

return render\_template('register.html',

title = 'Register', form = form)

'Route for Login, another form. If its valid and the user exists in the

'database, then the user is logged in with login\_user(user) which is

'a part of Flask-Login. If its incorrect it sends them back to login and

'tells them its wrong

@app.route('/login', methods=['GET', 'POST'])

def login():

form = LoginForm()

if request.method == 'POST':

if form.validate\_on\_submit():

user = form.get\_user()

if (user != None):

login\_user(user)

return redirect( url\_for('home') )

else:

return render\_template('login.html',

title = 'Login', incorrect = True, form = form )

else:

return render\_template('login.html',

title = 'Login', form = form)

elif request.method == 'GET':

return render\_template('login.html',

title = 'Login', form = form)

'Route for logout, just calls logout\_user() function, which is a part

'of Flask-Login, just gets rid of the session and everything that was

'a part of that user logging in

@app.route('/logout')

def logout():

logout\_user()

return redirect( url\_for('index') )

'Home route, also @login\_required. With Flask-WTF you can declare routes

'to only be seen if a user is logged in, this is what the @login\_required

'does. Other than that, this page just displays the information for the

'current logged in user after they log in

@app.route('/home')

@login\_required

def home():

u = current\_user

return render\_template('home.html',

title = 'Home',

firstname = u.firstname,

lastname = u.lastname,

username = u.username,

email = u.email)

'Give ability to run as module or standalone

'Run server with debugging on

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug = True)