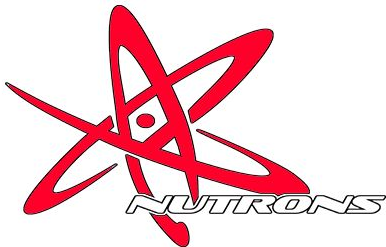
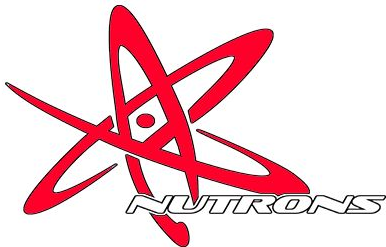
“Program your face off”



Game plan

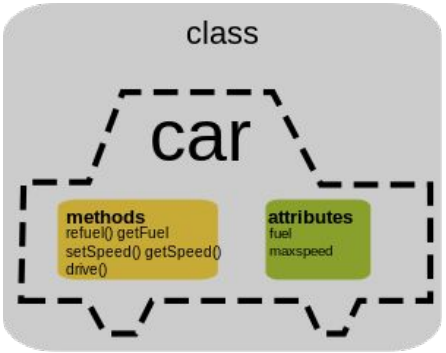
- Basics of Programming
  - Primitive types, loops, and conditionals.
  - What is an Object oriented language?
- Tips and tricks of WPIlib
- Iterative and Command Based robots
- Feedback devices(Encoders, Cameras, Sensors)
- Motion Controllers(Bang-bang, Hold Heading, PID Loops, Motion Profiling)
- Version Control and Subteam Management

Basics of Programming:



What is an object oriented language?

- Object Oriented Programming is a form of programming that relies on the idea of “objects” which contains data in its fields, and has different functions in its methods.
  - The Object instantiated would be “Car.”
  - Its fields would be max speed, fuel, tire pressure, price, etc.
  - Its methods would be functions such as driving, setting speed, retrieving speed, refueling, and getting the amount fuel left in the tank.



Field data types

- `Primitive types:`
  - Boolean - value that only reads out as true or false.
  - Char - all singular characters, usually kept between ‘ ‘, can be put together to make a string.
  - Int - an integer with a value between -2,147,483,648 and 2,146,483,648
  - Long - an integer with a larger value range than an int
  - Float - integers with decimals or decimals.
  - Double - a larger and more precise Float
  - String - an array of chars

Loops

- Just like Other Object oriented languages java contains these kinds of loops: For and While loops.
  - For loops
    - A single loop given parameters and test cases that will determine how many times it will run.
    - The test is a limit to dictate how many times the loop will run (Stopping when the test is false)
    - The parameters are an initial value and an incrementation of that same value.
  - While loops
    - Another type of loop that is responsible for repetitive actions
    - Only takes one case as an argument that will determine whether or not the loop will run again, stopping when the case is false.

For Loop Examples

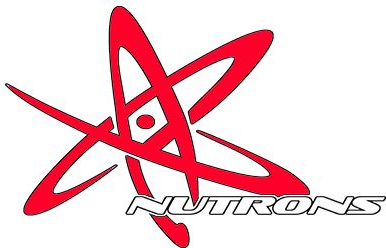
```
for(initialization ; test; update){
    Statement;
}
Now here is a real time example
for(int i = 1; i <= 125; i++){
    System.out.println(i + “Nutron”); // what does this print out
}
```


While loop Examples

```
while(Test){
    statements;
}
```

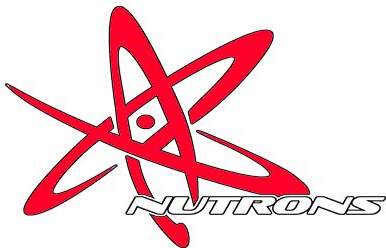
Real example:

```
while(a <= 10){
    a++;
}
```

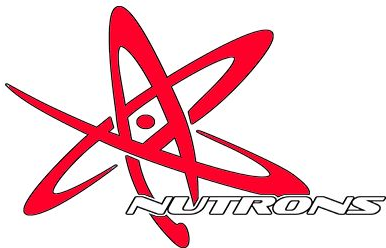
Method writing

```
access returnType methodName(parameters) {

    Function of the Method

}
```

Method writing example

```
public int multiplyByTwo(int n) {

    return n * 2;

}
```

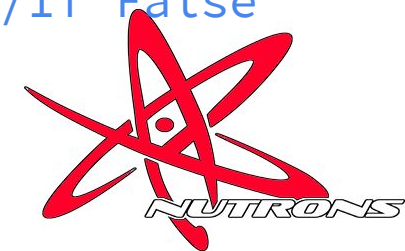



Conditionals

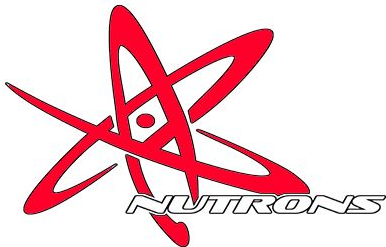
- Conditionals are boolean variables used to evaluate conditions. They use True or False values to determine what expression is evaluated.
  - If-Else statements are a form of conditionals that take in a condition and if it's true it runs a set expression that corresponds to that condition being true. If that condition is not true then it will run an expression under Else.

Conditional Example

```
Int U == 7;

if (U == 8){

    System.out.println(“This won't happen”); //If True

}

Else{

    System.out.println(“8 does not equal 7”); //If False

}
```

Different types of Robots: Iterative and Command based



Iterative based

- `Easy state transitions between two methods`
  - Init() methods
    - Called only once
  - Periodic() methods
    - Called multiple times, once for each loop of the code
    - Updates at 20 ms
    - Also parameters and functions to be continuously updated

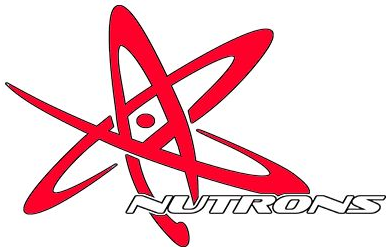
Command Based

- Uses the same base concept idea of the iterative robot
- Introduces the concept of Commands
  - Commands are a special type of object that is made once for a single action(s) then deleted.
  - Communicates directly with your subsystems.
  - Has a init() method to help setup whatever the execution is
  - Has an execute() method to run the commands desired action
  - Has a end() method to clean up and revert things back to whatever
  - Has a interupted() method to do an action just in case of interruption
- Can be used throughout the class and communicates with the robot's operator interface (OI) for teleop control.

Subsystem, Robot Map, and the Operator Interface

- Your Robot type communicates directly with your different Subsystems
- Your Subsystems communicate with your Robot Map to properly instantiate the different “pieces” of a subsystem (actuators, sensors, control)
- Your Operator Interface lays out your controllers and buttons and what subsystems they interact with to actually achieve something

Feedback Devices



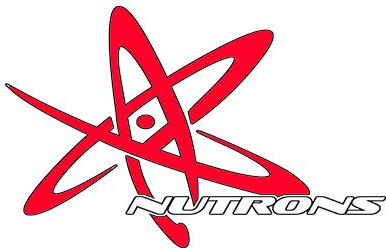
Encoders

A feedback device that allows for tracking of how much a motor has turned

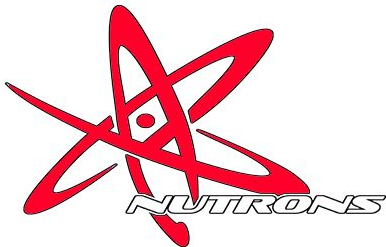
- Counts in ticks per revolution, usually a 4 digit number for preciseness
- Can be used in correspondence with wheel circumference to determine the distance travelled
- Can be used for pretty much any type of precise motor control in conjunction with different motor control (explanation for later)

Encoders example

```
public void driveInches(double inches, double wheelCircumfrence,
double ticksPerRev),  {

    if((inches / wheelCircumfrence) * ticksPerRev > encoder.pos()){

        drive(1.0);

    }

}
```

Cameras and Vision



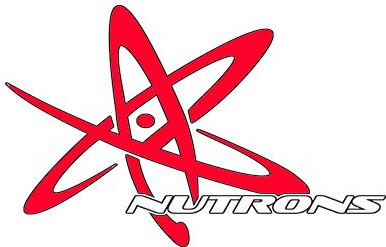
IMU’s (Gyros and Accelerometers)

Feedback sensor that allows for the ability to track the angle and position of your robot.

They measure:

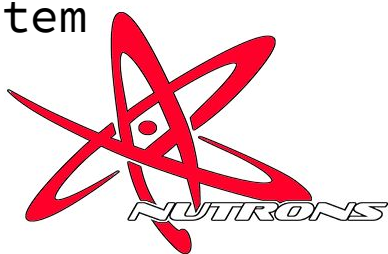
- Angle
  - Roll(**X axis**)
  - Pitch(**Y axis**)
  - Yaw(**Z axis**)
- Acceleration
- Gyro’s and Accelerometers can be used for control

Control Theory



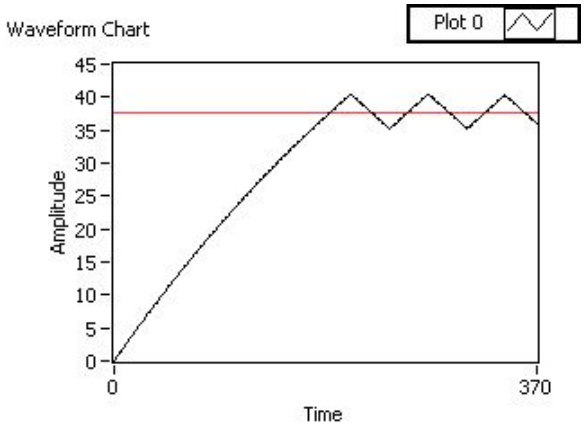
Introduction to Control Theory

- Making a physical system do what you want
- Using sensors and actuators to go to specific goal
- System - physical system with both actuators and sensors
  - Have an internal state
- State - all variables needed to describe a system's operations
  - Position, velocity, output voltage, etc
- Sensor - device used to measure the state of a system
  - Encoder, gyro, etc
- Actuator - device used to affect state of a system
  - Motors, etc



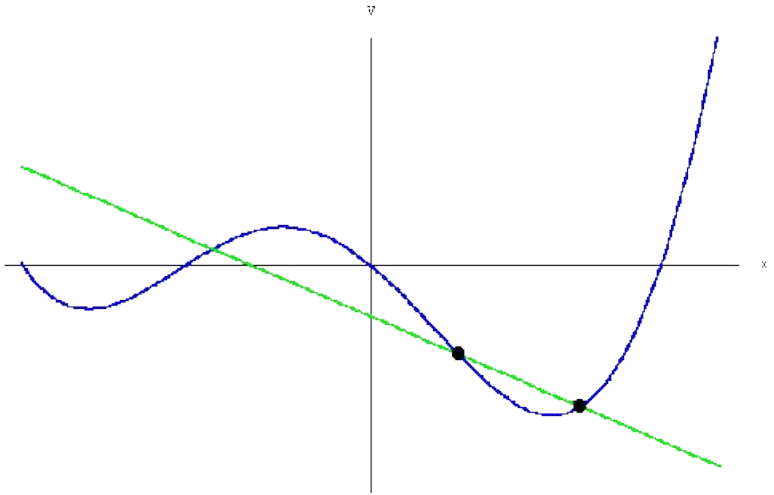
Bang Bang Controller

- Start will a goal position
- Which way do you need to go to get there?
  - Positive direction →Full forwards signal
  - Negative direction→Full backwards signal
- Results in massive oscillations (“vibrations”) around the goal
- Puts stress on certain systems
- Shooters can be effectively controlled by a bang-bang controller



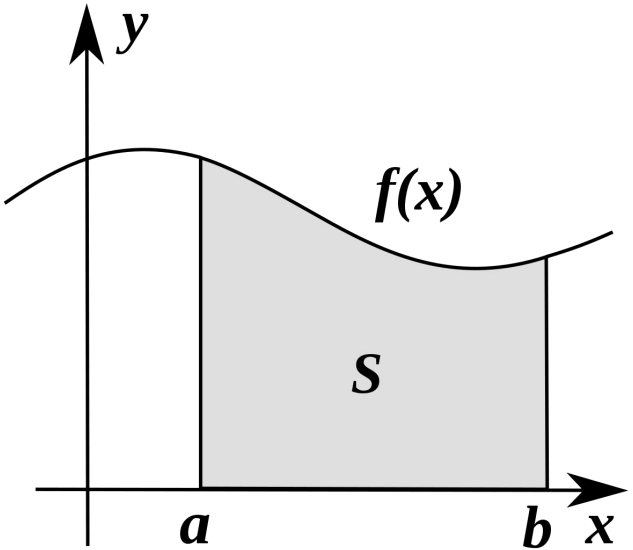
Crash Course Calculus: Derivatives

- Instantaneous rate of change
- Slope of a function
- How fast is something changing?
- Can find the velocity from the position
- Can find the acceleration from the velocity



Crash Course Calculus: Integrals

- Area underneath a function of time between two specific points
- Find velocity from acceleration
- Find position from velocity
- Be aware of your units, and make sure they are intuitive

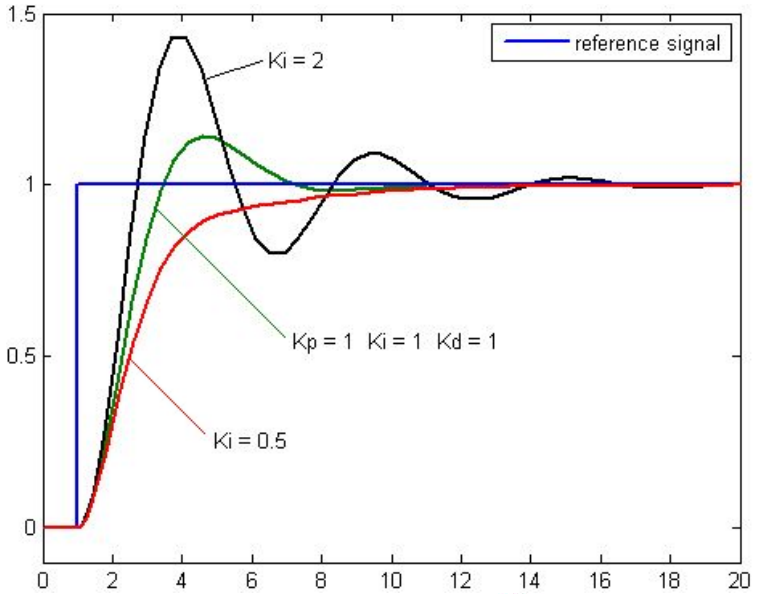


PID Loop

`error = goal - position`

`Three parts of the calculation:`

- `(P)roportional`
  - Drive towards the goal
- `(I)ntegral`
  - Builds up over time
- `(D)erivative`
  - Slows down when moving too fast



PID Function

$$u = (gain_p \times e) + (gain_i \times \int_0^t e) - (gain_d \times \frac{d_e}{d_t})$$

PID Function Continued

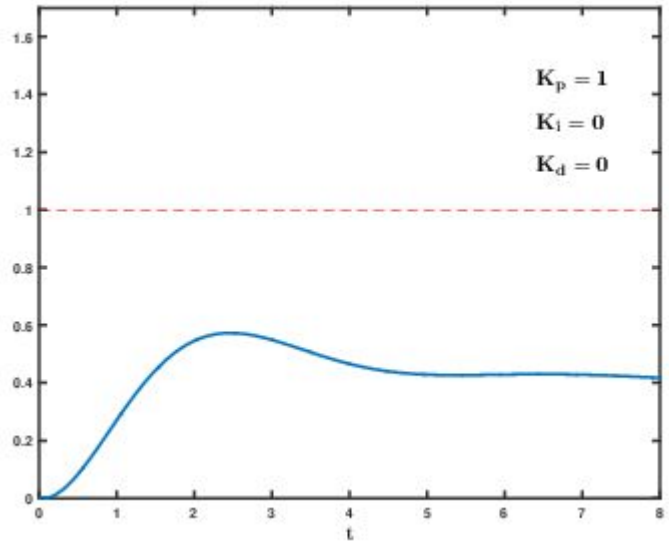
$$u = (gain_p \times e) + (gain_i \times \int_0^t e) - (gain_d \times \frac{d_e}{d_t})$$

- Using an elevator as an example, your goal is to move to a specific position
  - Output will be between -12 V and 12 V
- Think of your gains in terms of units
  - Output u is in volts, error e is in meters
  - $gain_p$ will have what units?
- Knowing units makes guessing gains much more reasonable

PID Tuning

1. Increase kP until the system starts to oscillate (move back and forth around a point)
2. If system isn't holding, increase kD until the oscillations stop
3. If there is steady state error (not reaching the goal in time), increase kI

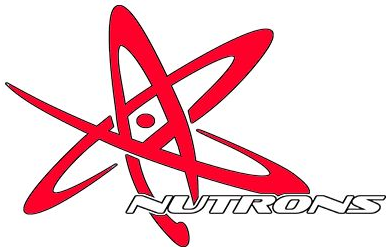
- Start slow, and be conservative. Can very easily break a system



Hold Heading

```
public void holdHeading() {
      if(getHeading() > headingSetPoint) {
            // Increase power to the left
      }
      if(getHeading() < headingSetPoint {
            // Increase power to the right
      }
}
```

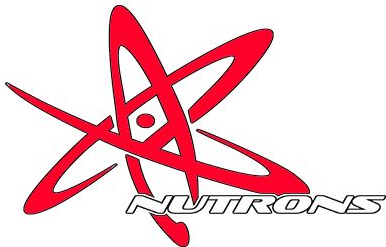

GitHub and Team Code Organization



What is GitHub?

- GitHub is a version control and code repository website where groups and organizations can create code repositories for projects, like a particular year's robot code.

Setting up a Code Base

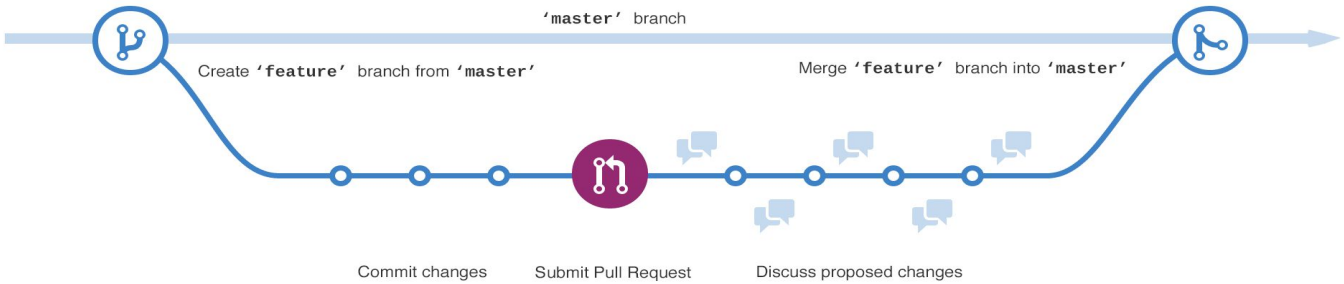


Repositories

- Repositories are code bases usually kept within organizations.
- Repositories usually consist of code for a singular project, multiple people can work on it at once under different branches.
- It's always good practice to keep the working code on the ‘master’ branch and all other code currently in work on other branches to be merged later.

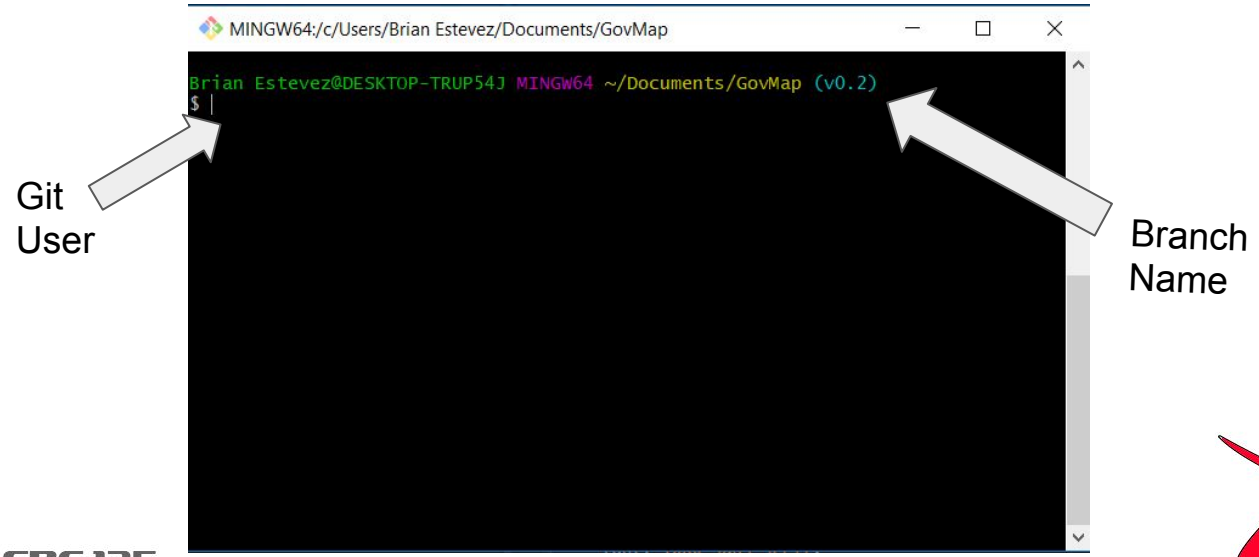
Branches and Version Control

- Branches are like mini version of your repository used to implement new features into your core code base
- The master branch is the main branch of your code base and should always be up to date
- When a feature is completed in a branch, it then merges with master



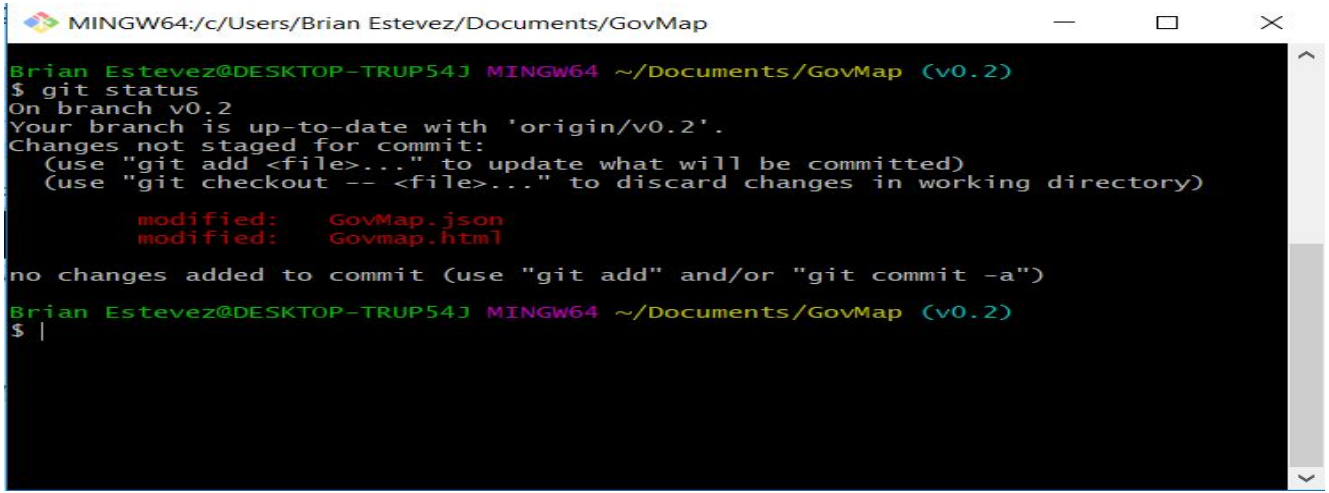
Git Bash Crash Course

- You can use the Git Bash to commit(add your code to the current repository) using git bash. Learning the bash is easy and gives you a lot more control over the process.



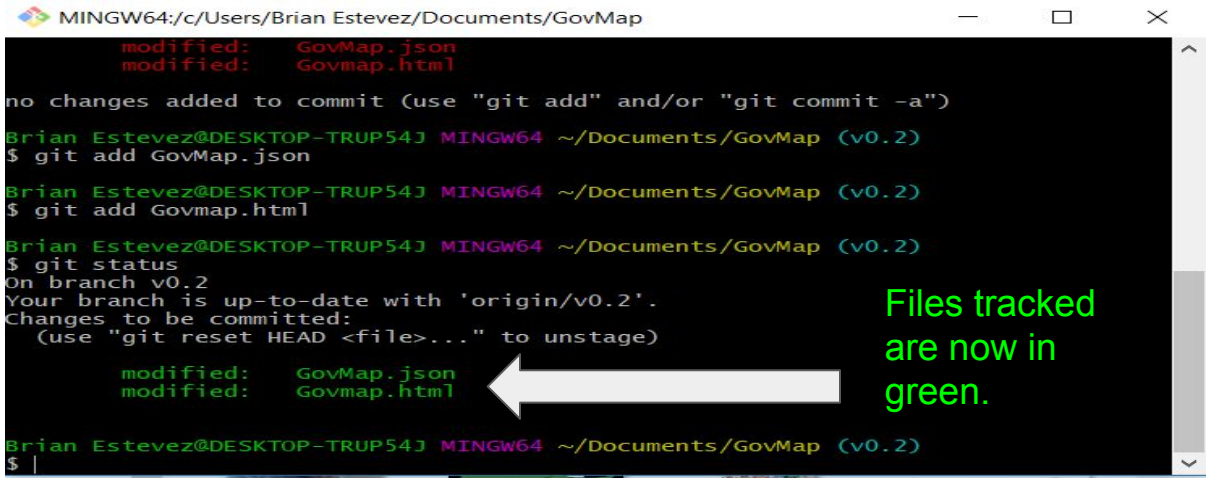
Git Bash Crash Course Cont.

- Use the command `git status` to view the changed files you have been working on locally. Changes that aren't staged to be committed(Ready to be uploaded to the repository) will be in red



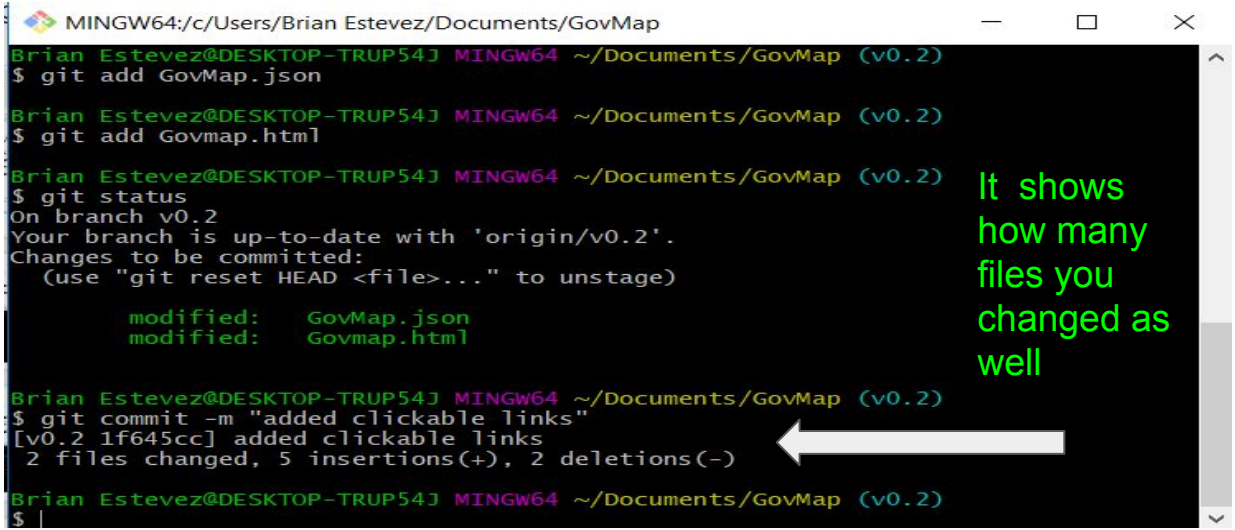
Git Bash Crash Course Cont.

- Git add is the next command you would use, type git add (file name) for whatever local files you want to be committed. Then git status again to see the files tracked to commit in green.



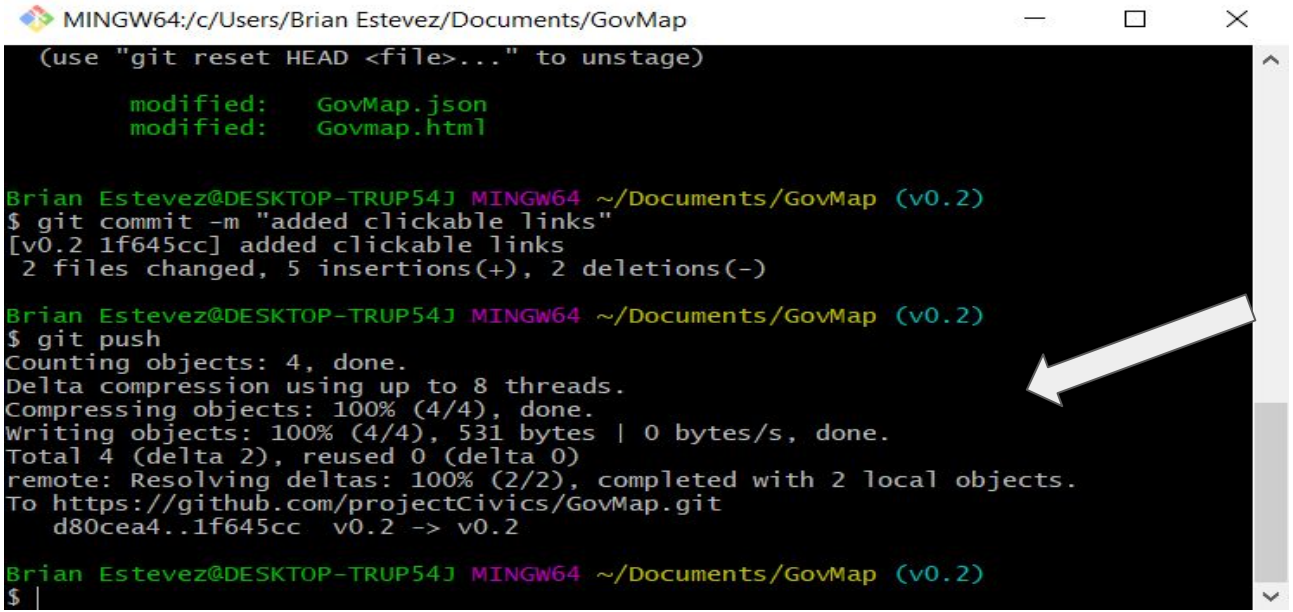
Git Bash Crash Course Cont.

- Now it's time to add your commit message. Your commit message should be what you did/edited in the code. Type git commit -m “(commit message)” and press enter.



Git Bash Crash Course Cont.

- Now type git push, and all your code would be pushed to the repository.



Now your code is pushed!

Git Bash Crash Course, Tips and Tricks

- Other github commands that are useful are git diff, git checkout -b, git merge, git mv, and git pull.
  - Git diff tells you the difference between live code and any edits you've made recently.
  - Git checkout -b [branch name] lets you make a new branch in the current repo.
  - Git merge lets you merge the code in one branch into another.
  - Git mv [original file] [new filename] lets you refactor a file name.
  - Git pull allows you to pull code from a branch from the repository.

screenstepslive