Introducing PROS 3

by
Willem Scholten
Learning Access Institute

PROS 3 Intro

- Some key resources to know about:
  - C programming tutorials:
    - https://www.cprogramming.com/tutorial/c-tutorial.html
    - https://www.studytonight.com/c/overview-of-c.php
    - https://www.youtube.com/watch?v=nXvy5900m3M&feature=youtu.be

PROS 3 Intro

- Online PROS documentation:
  - Cortex:
    - https://pros.cs.purdue.edu/cortex/index.html
  - V5
    - https://pros.cs.purdue.edu/v5/index.html

PROS 3 Intro

- **Functions.** C is a language that heavily emphasizes functions, and knowing how they work is essential to using PROS. The PROS API <../api/index.html>`_ is a set of functions, so any time that you want to interact with a sensor or motor, you're using functions.

PROS 3 Intro

- **Header Files.** The PROS template (the set of files automatically created when you start a PROS project) contains a couple of header files, and it's recommended that you make additional header files as you develop your code. Header files contain the declarations for functions and global variables (among other things), which is why the PROS API can be found in API.h. Knowing what code should go in a **header file (.h)** or a **source file (.c)** can be difficult to determine at first, but it is a very useful skill to learn.

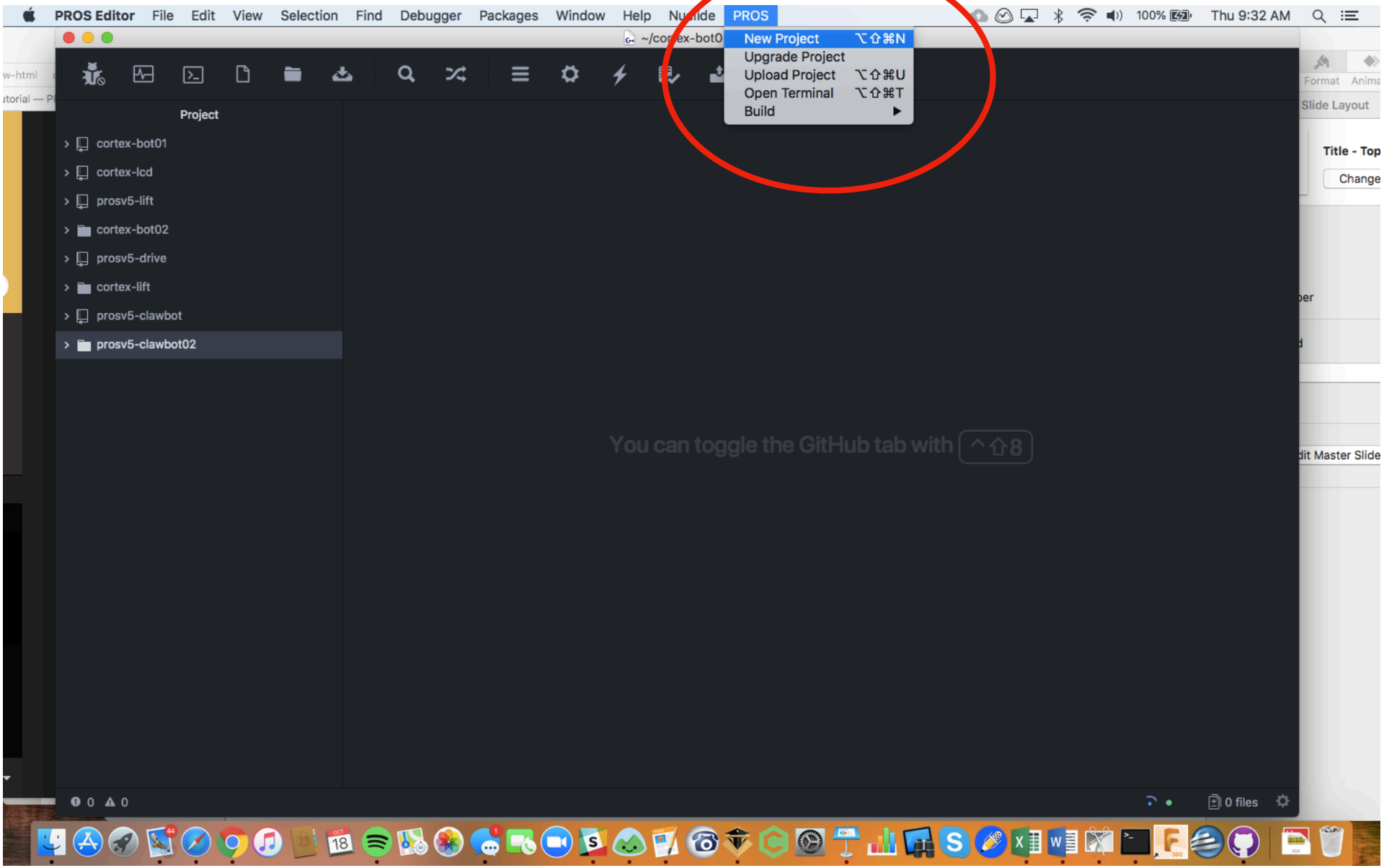
PROS 3 Intro

- **printf()**. At some point when developing PROS code, you will likely want to get some feedback on what the value of a variable is. This is not an exact replacement for a full debugging utility by any means, but is the standard method for troubleshooting issues in most languages and can be used for viewing sensor values or your own variables' values. The **output** from these **printf()** statements can be **viewed** in the terminal by running **pros terminal.**

PROS 3 Intro

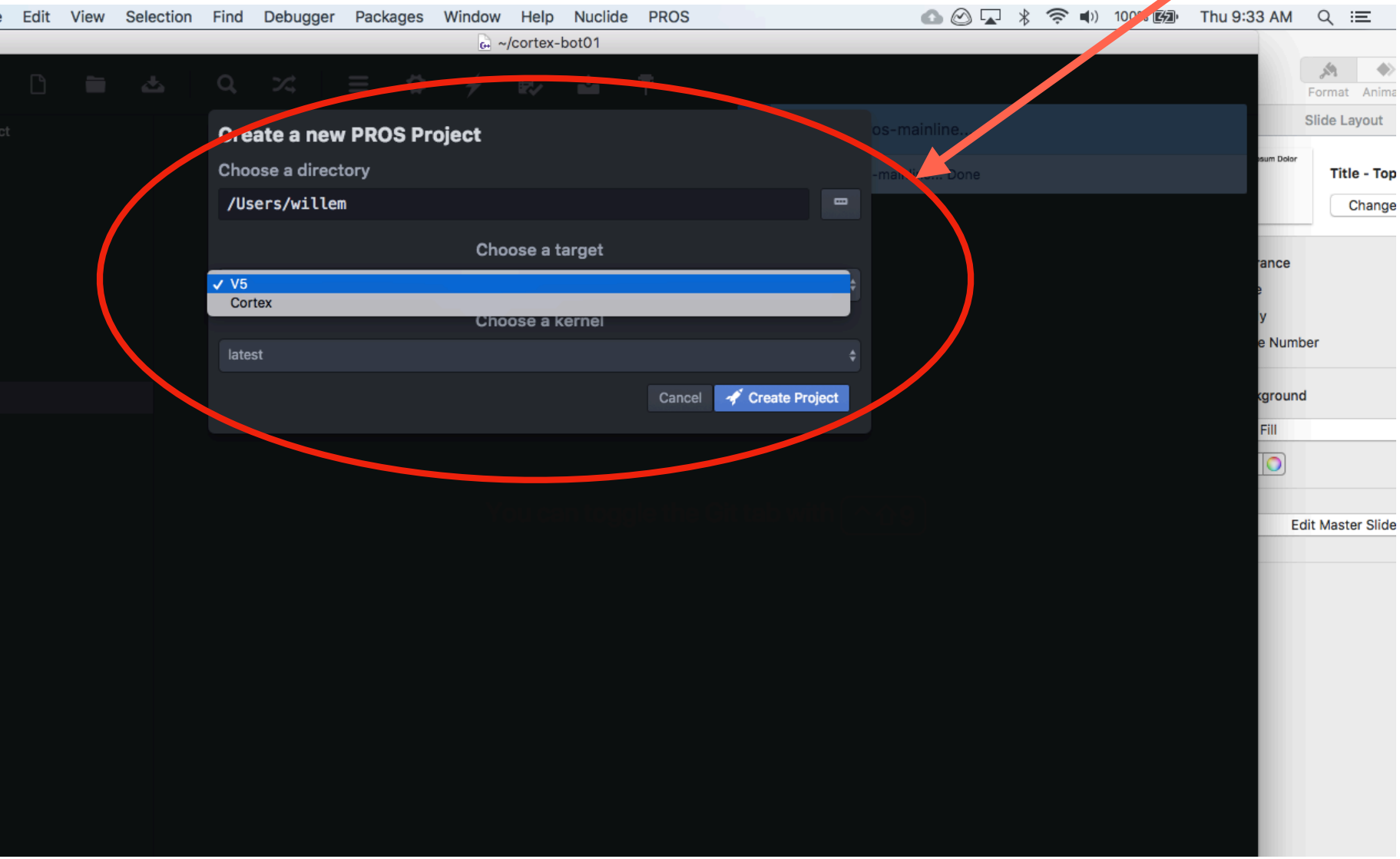
```
project
|   project.pros          (used by PROS CLI to know about kernel version and other meta data)
|   Makefile              (instructs make how to compile project)
|   common.mk             (helper file for Makefile)
|___src                   (Source files should go here)
|   |   auto.c            (source fiel for autonomous functions)
|   |   init.c            (source file for initialization)
|   |   opcontrol.c       (source fiel for operator control)
|   |   Makefile          (instructs make how to compile your source files)
|
|___include               (Header files should go here)
|   |   API.h             (lets source files know the PROS API functions)
|   |   main.h            (includes API.h and anything else your projects should know project wide)
|
|___firmware              (NEVER need to be in here)
|   |   cortex.ld         (instructs linker how to construct binary fields for cortex)
|   |   libpros.a         (Pre-compiled PROS library)
|   |   STM32F10x.ld      (instructs linker how to construct binary fields for cortex)
|   |   uniflash.jar      (Legacy flashing uti)
```

PROS 3 Intro


PROS Editor File Edit View Selection Find Debugger Packages Window Help Nuclide PROS
100% Thu 9:32 AM
New Project ⌥⇧⌘N
Upgrade Project
Upload Project ⌥⇧⌘U
Open Terminal ⌥⇧⌘T
Build
Project
cortex-bot01
cortex-lcd
prosv5-lift
cortex-bot02
prosv5-drive
cortex-lift
prosv5-clawbot
prosv5-clawbot02
You can toggle the GitHub tab with ^⇧8
0 files

PROS 3 Intro

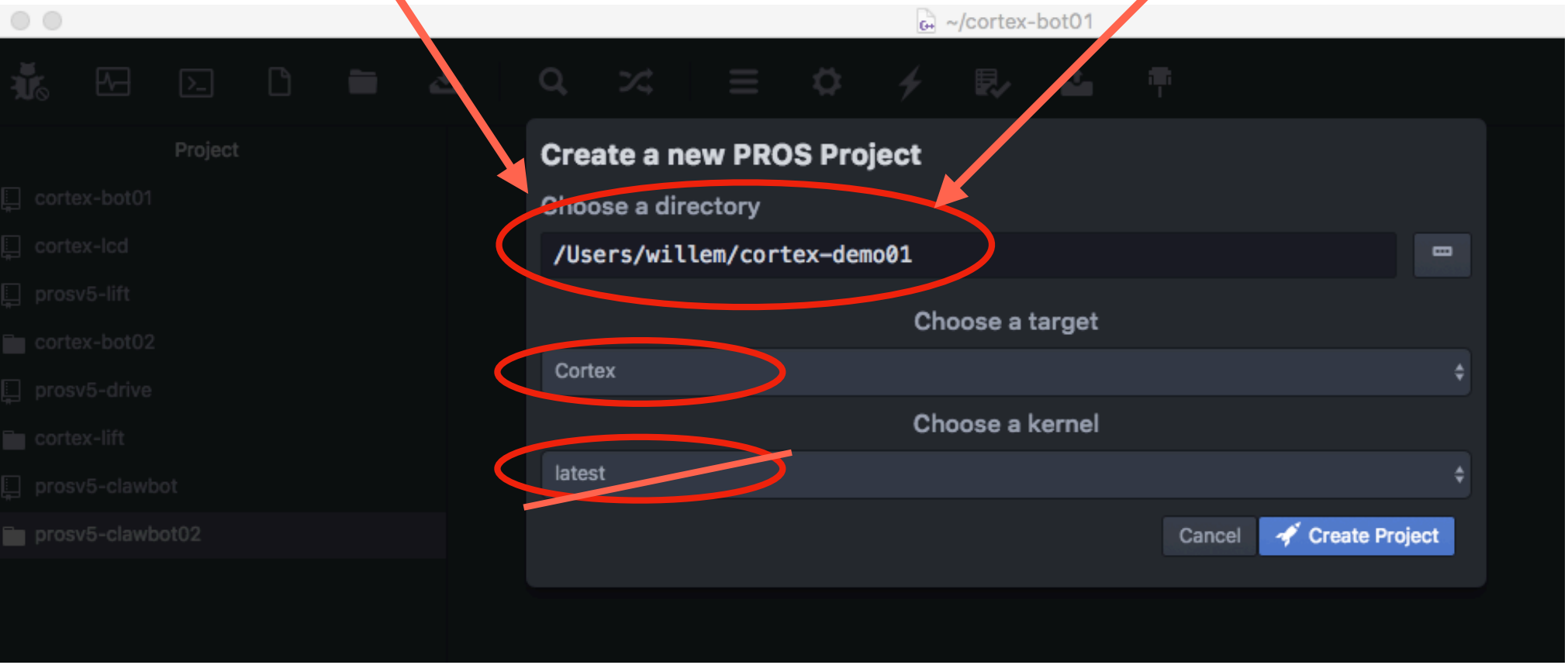
Pick the right compile target - Cortex or V5



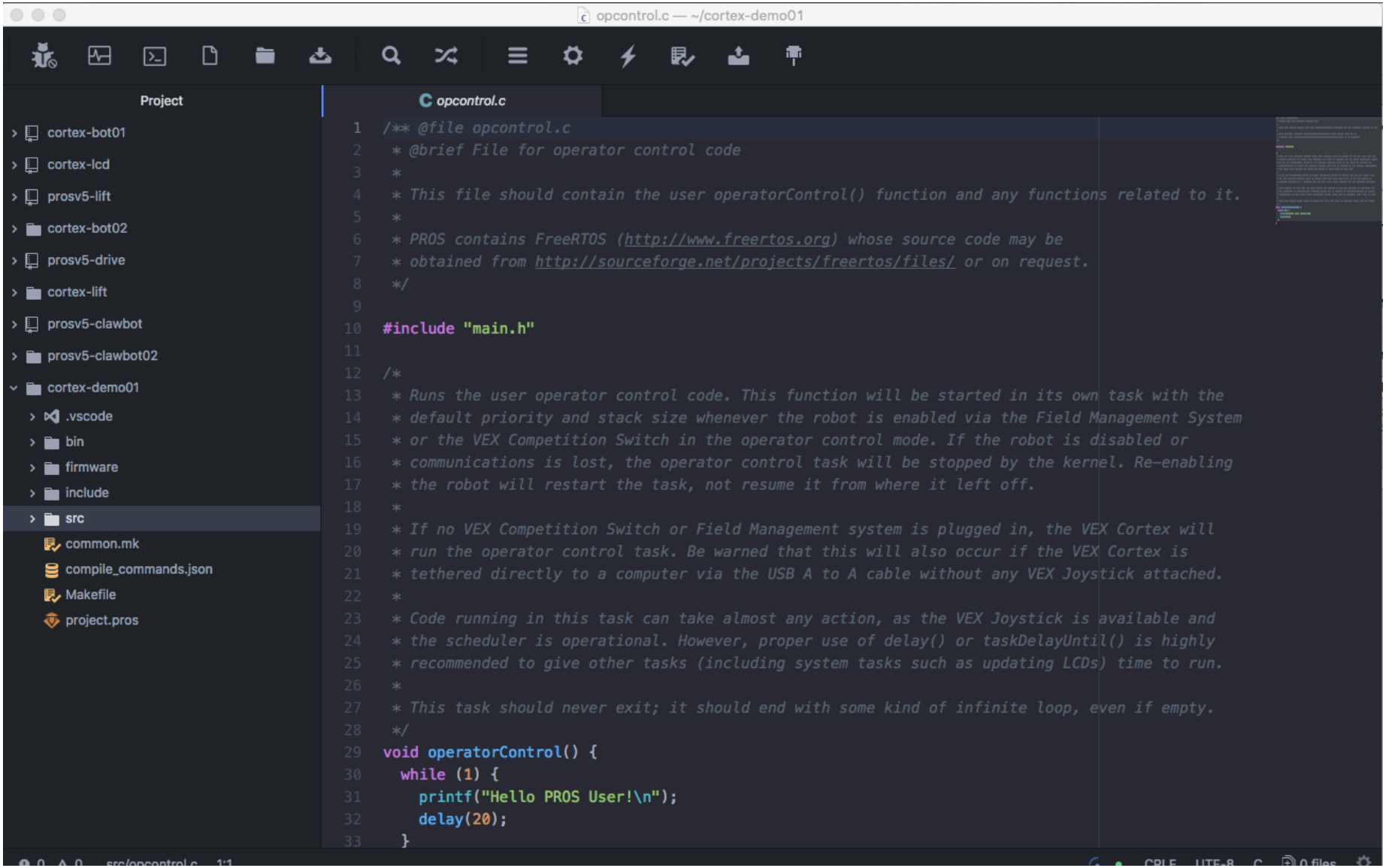
PROS 3 Intro

Pick the right compile target - Cortex

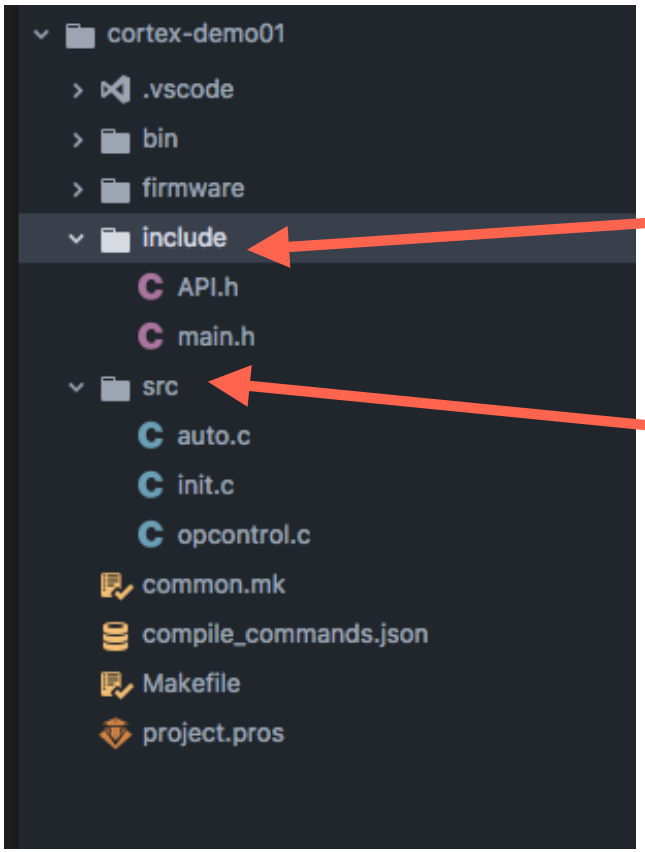
Give your project a name



PROS 3 Intro


opcontrol.c — ~/cortex-demo01
Project
cortex-bot01
cortex-lcd
prosv5-lift
cortex-bot02
prosv5-drive
cortex-lift
prosv5-clawbot
prosv5-clawbot02
cortex-demo01
.vscode
bin
firmware
include
src
common.mk
compile_commands.json
Makefile
project.pros
opcontrol.c
/** @file opcontrol.c
* @brief File for operator control code
*
* This file should contain the user operatorControl() function and any functions related to it.
*
* PROS contains FreeRTOS (http://www.freertos.org) whose source code may be
* obtained from http://sourceforge.net/projects/freertos/files/ or on request.
*/
#include "main.h"
/*
* Runs the user operator control code. This function will be started in its own task with the
* default priority and stack size whenever the robot is enabled via the Field Management System
* or the VEX Competition Switch in the operator control mode. If the robot is disabled or
* communications is lost, the operator control task will be stopped by the kernel. Re-enabling
* the robot will restart the task, not resume it from where it left off.
*
* If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will
* run the operator control task. Be warned that this will also occur if the VEX Cortex is
* tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.
*
* Code running in this task can take almost any action, as the VEX Joystick is available and
* the scheduler is operational. However, proper use of delay() or taskDelayUntil() is highly
* recommended to give other tasks (including system tasks such as updating LCDs) time to run.
*
* This task should never exit; it should end with some kind of infinite loop, even if empty.
*/
void operatorControl() {
while (1) {
printf("Hello PROS User!\n");
delay(20);
}

PROS 3 Intro



Include (header .h) files for your project

Source (.c) files for your project

PROS 3 Intro

- By convention, the **opcontrol()**, **autonomous()**, and **initialize functions** are separated into separate files (**opcontrol.c, auto.c, and init.c**). They could be all in the same file, but it can be helpful to organize your functions into multiple files to keep things from becoming messy.

PROS 3 Intro

Core opcontrol.c functions:

```
int joystickGetAnalog ( unsigned char joystick,    // the joystick slot to check (1 for master,
                                                   // 2 for partner)
                        unsigned char axis         // One of the joystick channels on a VEX
                                                   // Joystick: 1, 2, 3, 4, ACCEL_X, or ACCEL_Y
                      );
```

Gets the value of a control axis on the VEX joystick. *Returns the value of -127 to 127, or 0 if no joystick is connected to the requested slot(S)

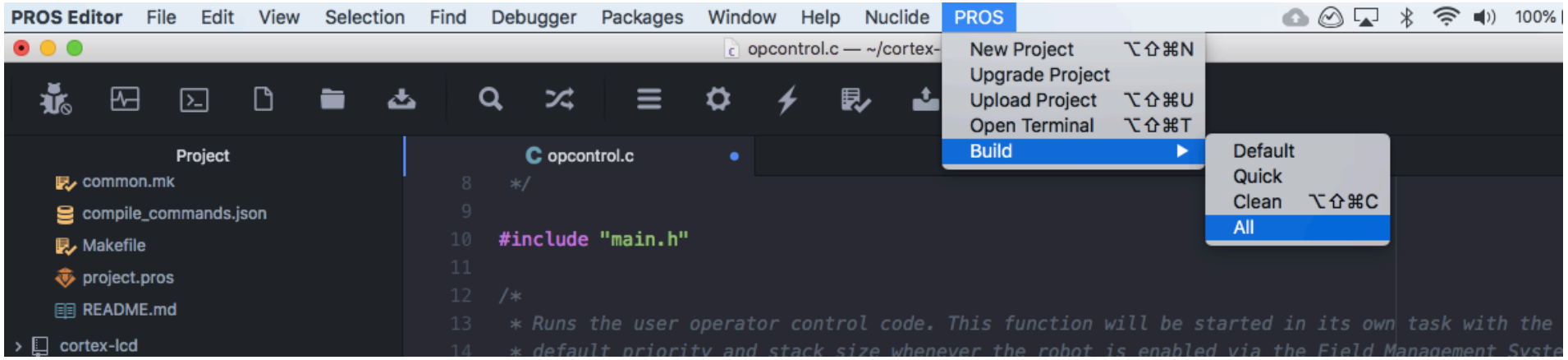
```
void motorSet ( unsigned char channel,          // motor channel to set from 1-10
                int speed                       // new signed speed from -127, to 0, to 127
              );
```

PROS 3 Intro

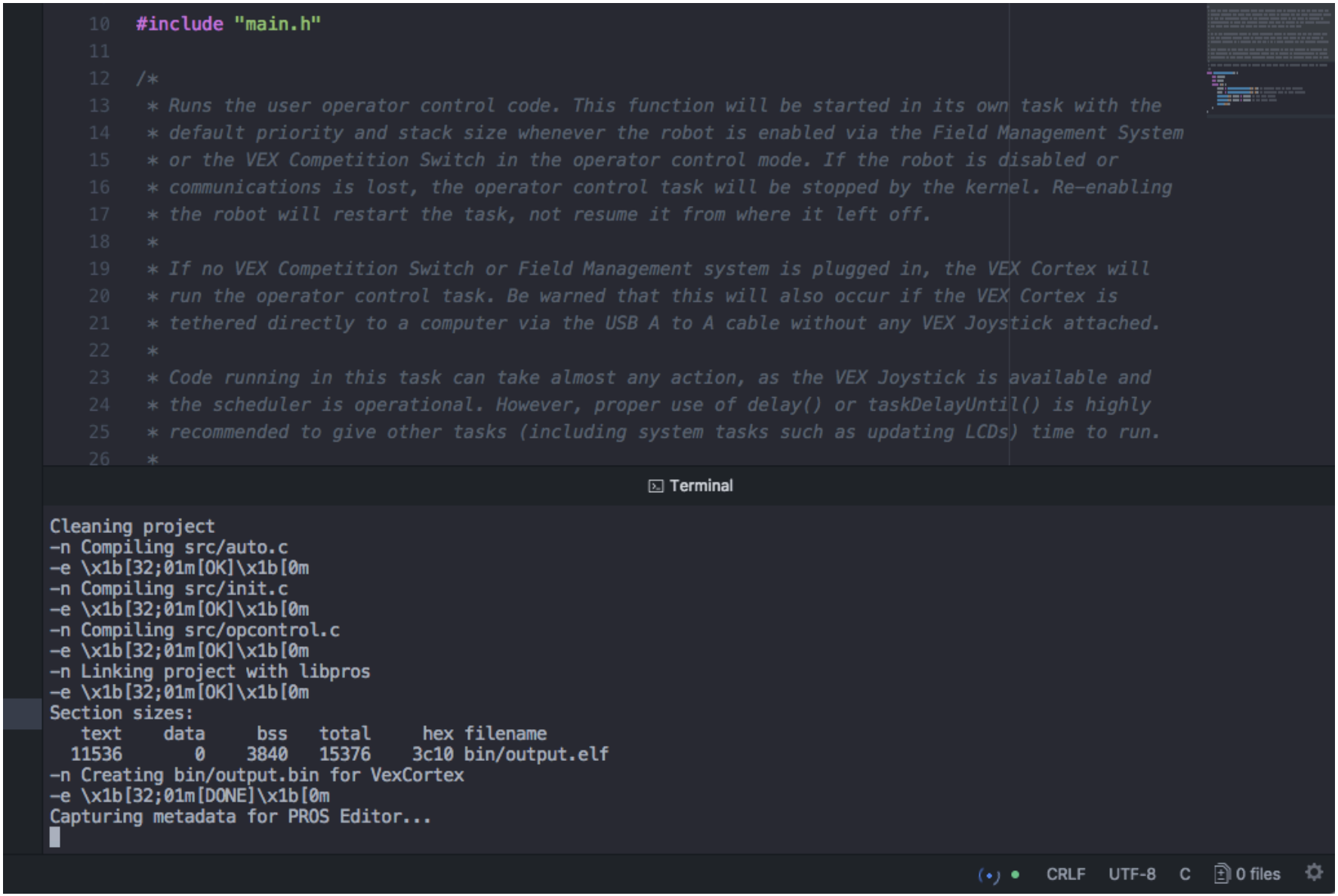
Arcade Control Sample

```
void operatorControl() {
        int power;
        int turn;
        while (1) {
            power = joystickGetAnalog(1, 2); // vertical axis on left joystick
            turn  = joystickGetAnalog(1, 1); // horizontal axis on left joystick
            motorSet(2, power + turn); // set left wheels
            motorSet(3, power - turn); // set right wheels
            delay(20);
        }
}
```

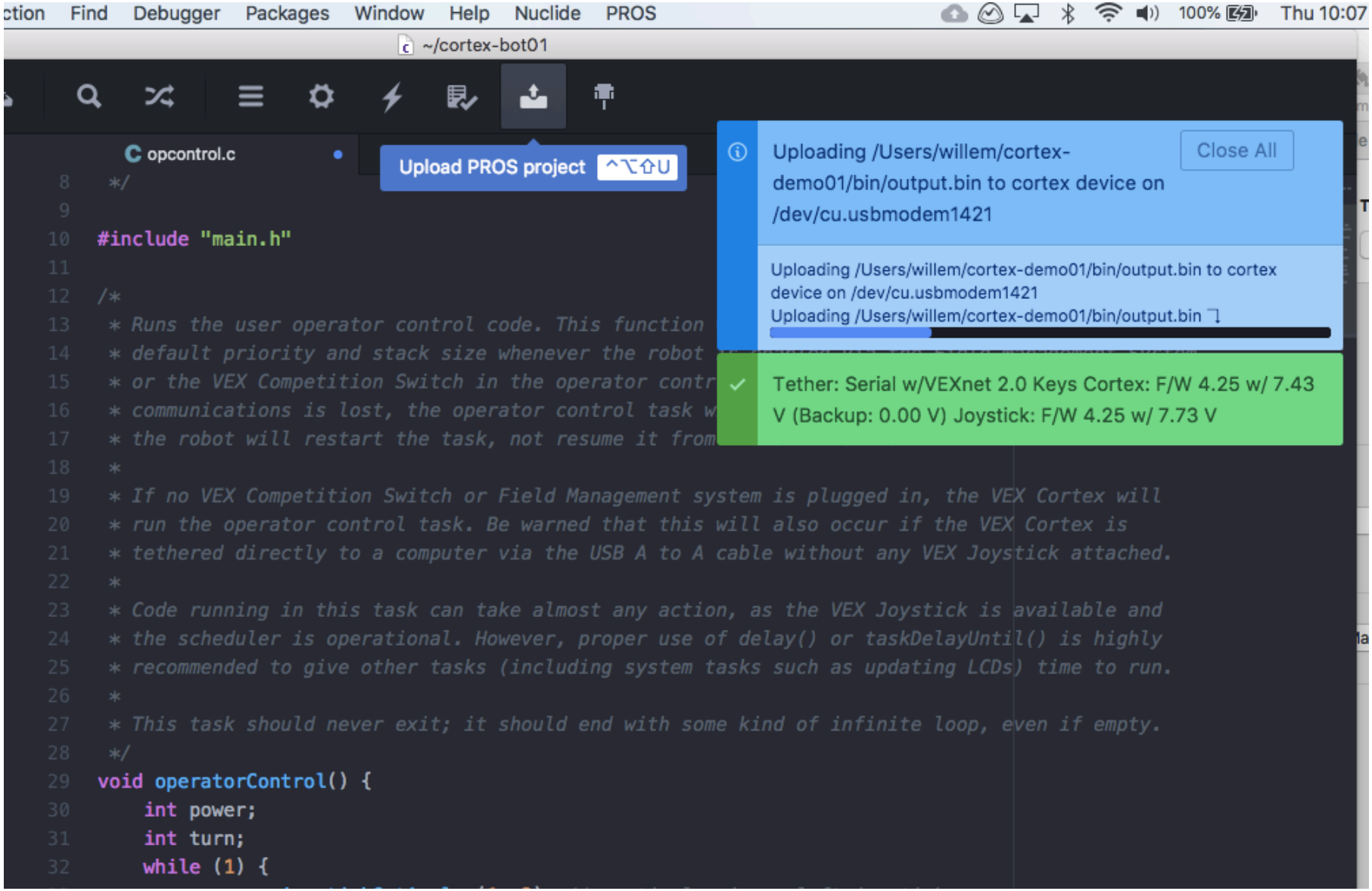
PROS 3 Intro


PROS Editor
File
Edit
View
Selection
Find
Debugger
Packages
Window
Help
Nuclide
PROS
100%
opcontrol.c — ~/cortex-
New Project ⌥⇧⌘N
Upgrade Project
Upload Project ⌥⇧⌘U
Open Terminal ⌥⇧⌘T
Build
Default
Quick
Clean ⌥⇧⌘C
All
Project
common.mk
compile_commands.json
Makefile
project.pros
README.md
cortex-lcd
opcontrol.c
*/
#include "main.h"
/*
* Runs the user operator control code. This function will be started in its own task with the

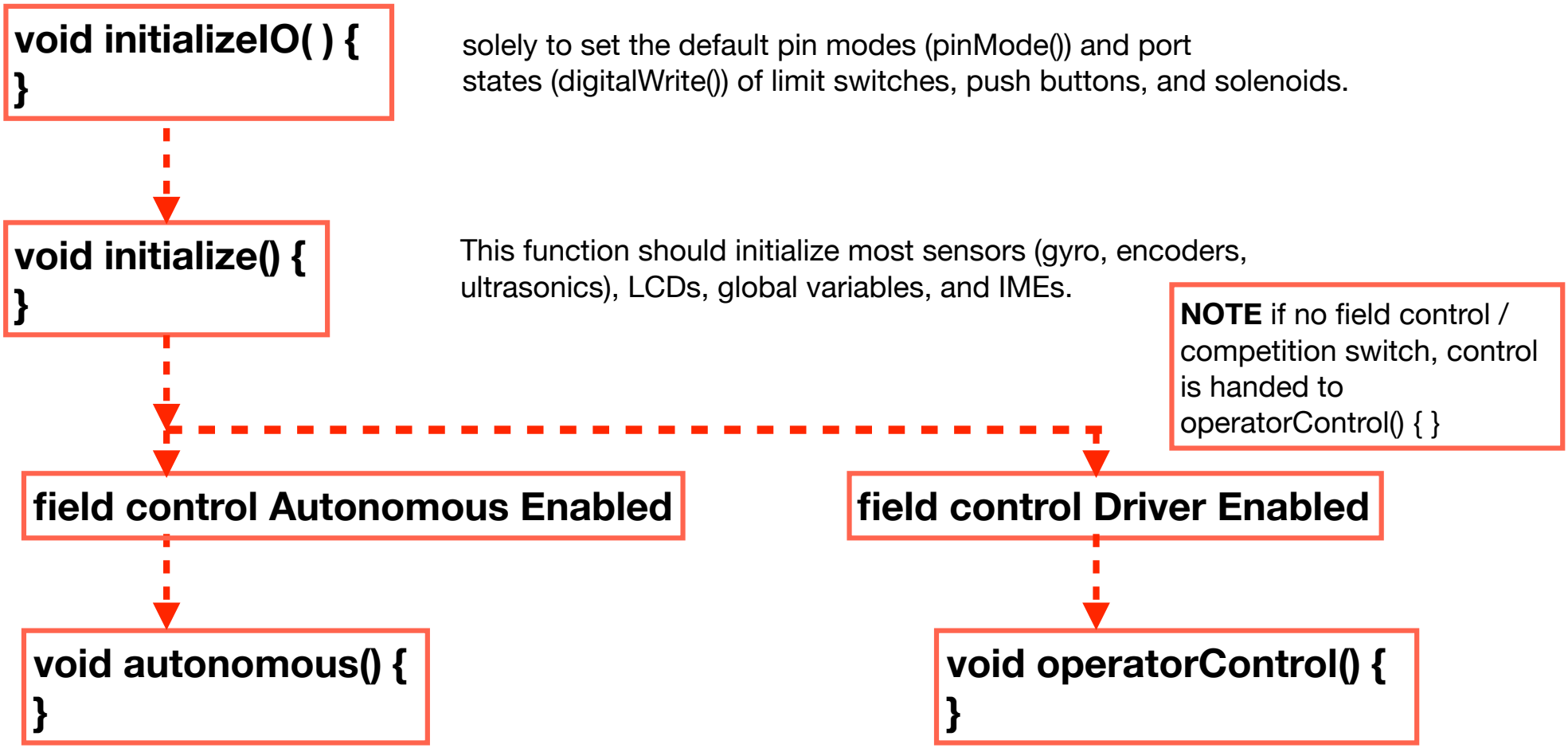
PROS 3 Intro


10 #include "main.h"
11
12 /*
13 * Runs the user operator control code. This function will be started in its own task with the
14 * default priority and stack size whenever the robot is enabled via the Field Management System
15 * or the VEX Competition Switch in the operator control mode. If the robot is disabled or
16 * communications is lost, the operator control task will be stopped by the kernel. Re-enabling
17 * the robot will restart the task, not resume it from where it left off.
18 *
19 * If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will
20 * run the operator control task. Be warned that this will also occur if the VEX Cortex is
21 * tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.
22 *
23 * Code running in this task can take almost any action, as the VEX Joystick is available and
24 * the scheduler is operational. However, proper use of delay() or taskDelayUntil() is highly
25 * recommended to give other tasks (including system tasks such as updating LCDs) time to run.
26 *
Terminal
Cleaning project
-n Compiling src/auto.c
-e \x1b[32;01m[OK]\x1b[0m
-n Compiling src/init.c
-e \x1b[32;01m[OK]\x1b[0m
-n Compiling src/opcontrol.c
-e \x1b[32;01m[OK]\x1b[0m
-n Linking project with libpros
-e \x1b[32;01m[OK]\x1b[0m
Section sizes:
text data bss total hex filename
11536 0 3840 15376 3c10 bin/output.elf
-n Creating bin/output.bin for VexCortex
-e \x1b[32;01m[DONE]\x1b[0m
Capturing metadata for PROS Editor...
CRLF
UTF-8
C
0 files

PROS 3 Intro


ction Find Debugger Packages Window Help Nuclide PROS
100% Thu 10:07
~/cortex-bot01
opcontrol.c
Upload PROS project ^⌥⇧U
Uploading /Users/willem/cortex-demo01/bin/output.bin to cortex device on /dev/cu.usbmodem1421
Close All
Uploading /Users/willem/cortex-demo01/bin/output.bin to cortex device on /dev/cu.usbmodem1421
Uploading /Users/willem/cortex-demo01/bin/output.bin
Tether: Serial w/VEXnet 2.0 Keys Cortex: F/W 4.25 w/ 7.43 V (Backup: 0.00 V) Joystick: F/W 4.25 w/ 7.73 V
*/
#include "main.h"
/*
* Runs the user operator control code. This function
* default priority and stack size whenever the robot
* or the VEX Competition Switch in the operator contr
* communications is lost, the operator control task w
* the robot will restart the task, not resume it from
*
* If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will
* run the operator control task. Be warned that this will also occur if the VEX Cortex is
* tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.
*
* Code running in this task can take almost any action, as the VEX Joystick is available and
* the scheduler is operational. However, proper use of delay() or taskDelayUntil() is highly
* recommended to give other tasks (including system tasks such as updating LCDs) time to run.
*
* This task should never exit; it should end with some kind of infinite loop, even if empty.
*/
void operatorControl() {
int power;
int turn;
while (1) {

PROS 3 Intro


void initializeIO( ) {
}
solely to set the default pin modes (pinMode()) and port states (digitalWrite()) of limit switches, push buttons, and solenoids.
void initialize() {
}
This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global variables, and IMEs.
NOTE if no field control / competition switch, control is handed to operatorControl() { }
field control Autonomous Enabled
field control Driver Enabled
void autonomous() {
}
void operatorControl() {
}

PROS 3 Intro

Autonomous Sample

```
void autonomous() {
  motorSet(2, 100);   // set right wheels
  motorSet(3, -100);  // set left wheels - reversed !

  delay(500);         // drive for 500ms forward

  motorSet(2, 0);     // stop both wheels
  motorSet(3, 0);
}
```

PROS 3 Intro

- PROS 3 kernel upgrade of projects:

(Mostly applies to PROS for V5 - cortex kernel is stable)

  - In terminal:

    - change to the project directory: cd prosv5-clawbot01

    - run: prosv5 c u

```
Willems-MacBook-Air:prosv5-clawbot02 willem$ prosv5 c u
Upgrading kernel
Applying kernel@3.1.2 [####################################] 100%
Finished applying kernel@3.1.2 to /Users/willem/github/sprobotics/prosv5-clawbot02
Upgrading okapilib
Applying okapilib@3.3.5 [####################################] 100%
Finished applying okapilib@3.3.5 to /Users/willem/github/sprobotics/prosv5-clawbot02
Willems-MacBook-Air:prosv5-clawbot02 willem$
```

GitHub and GIT

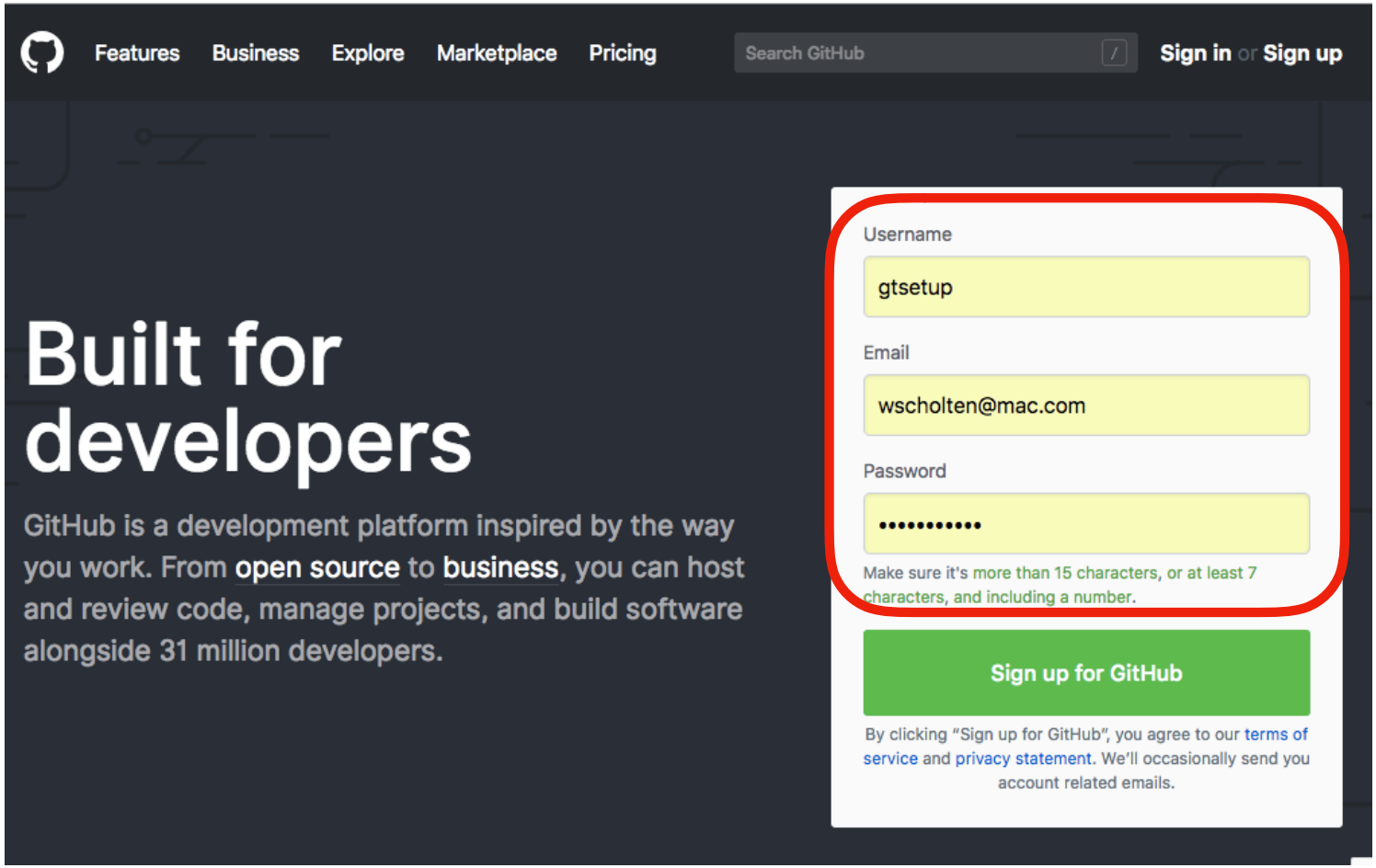
by
Willem Scholten
Learning Access Institute

gitHUB Access

- Sample code repositories for learning the Cortex (and later the V5) can be cloned from the following URL:
  - https://github.com/sprobotics

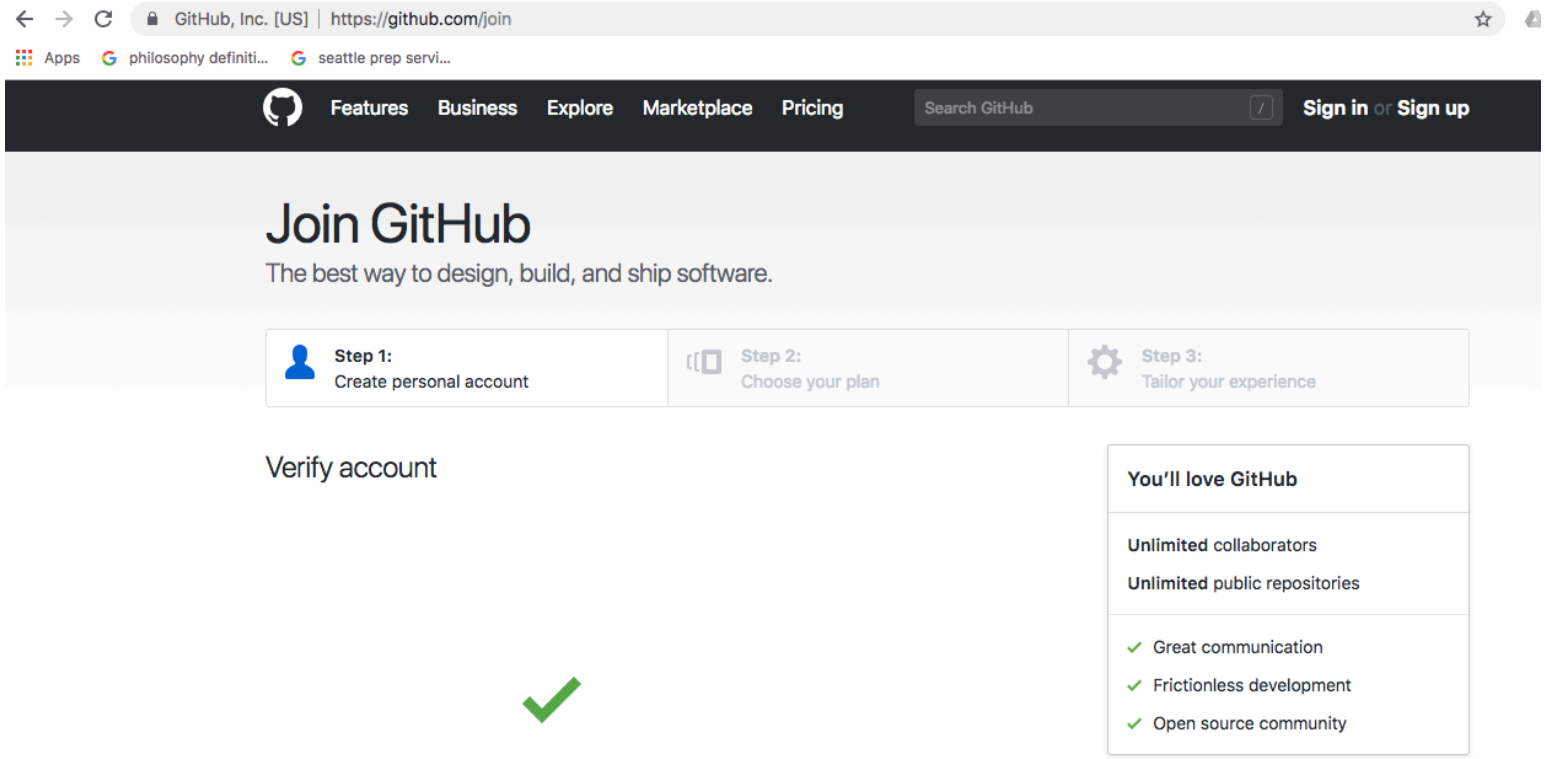
gitHUB Access

Goto: https://github.com

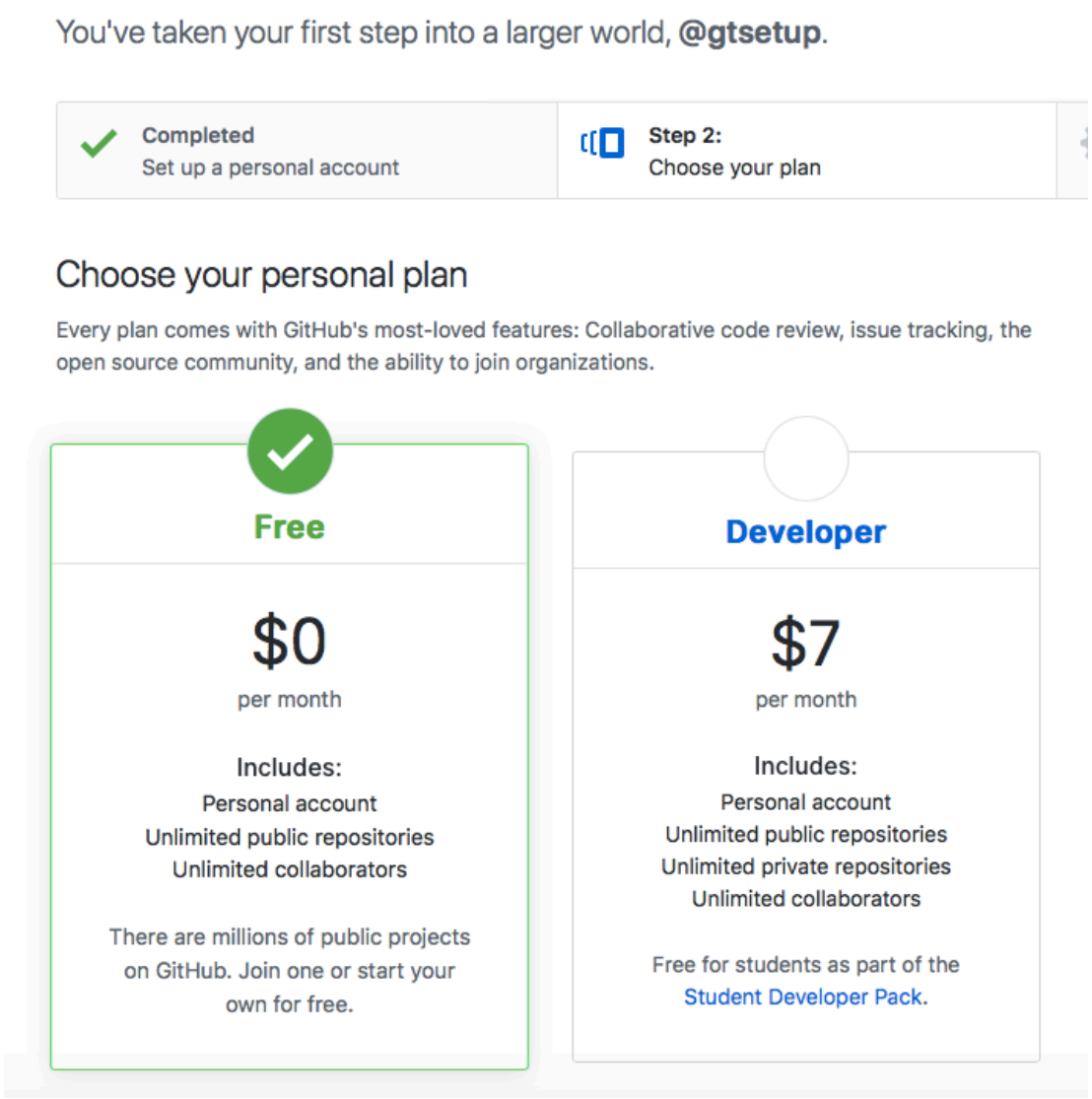


Pick username, add your email and pick a password

gitHUB Access

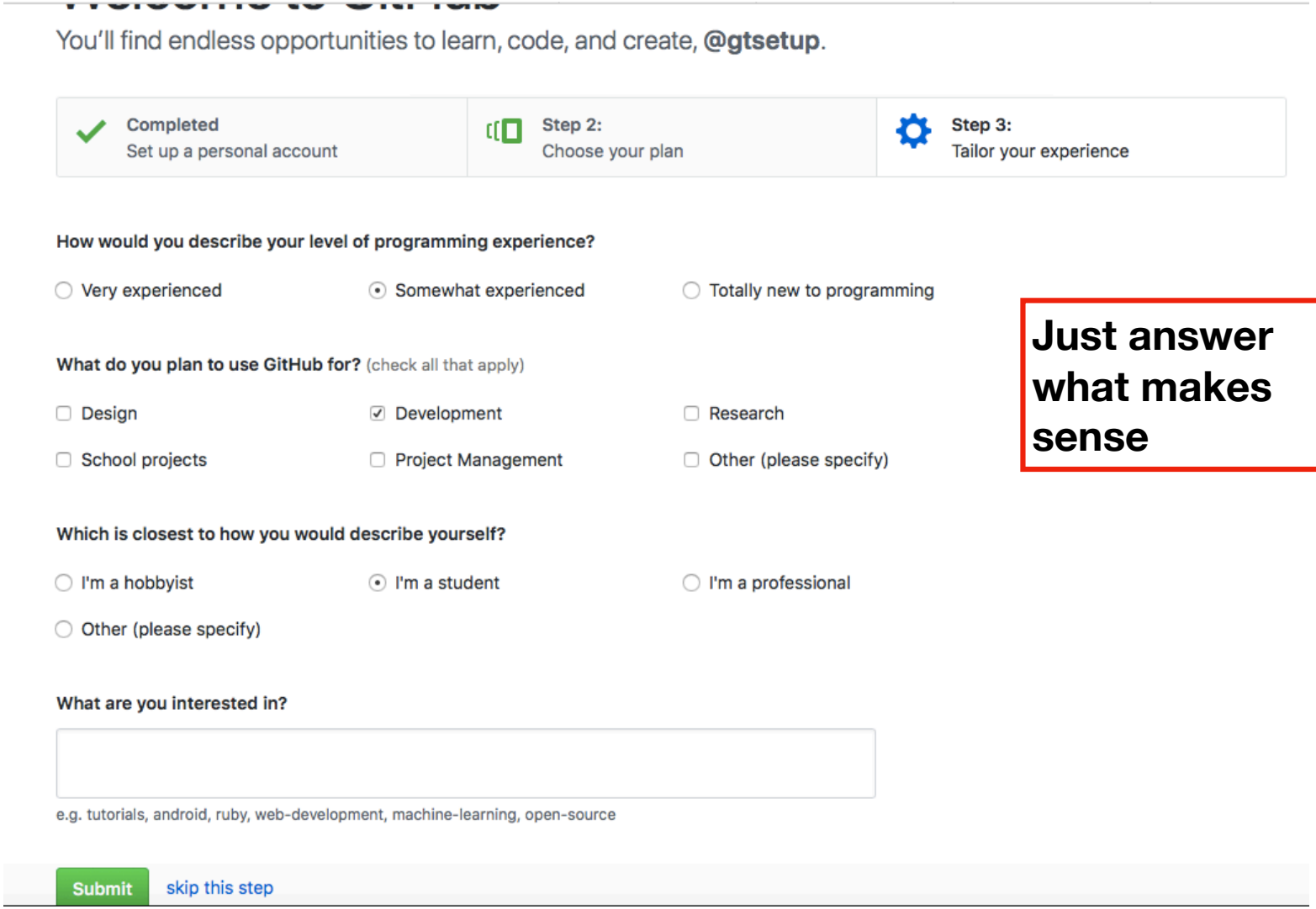


gitHUB Access



Pick the free plan

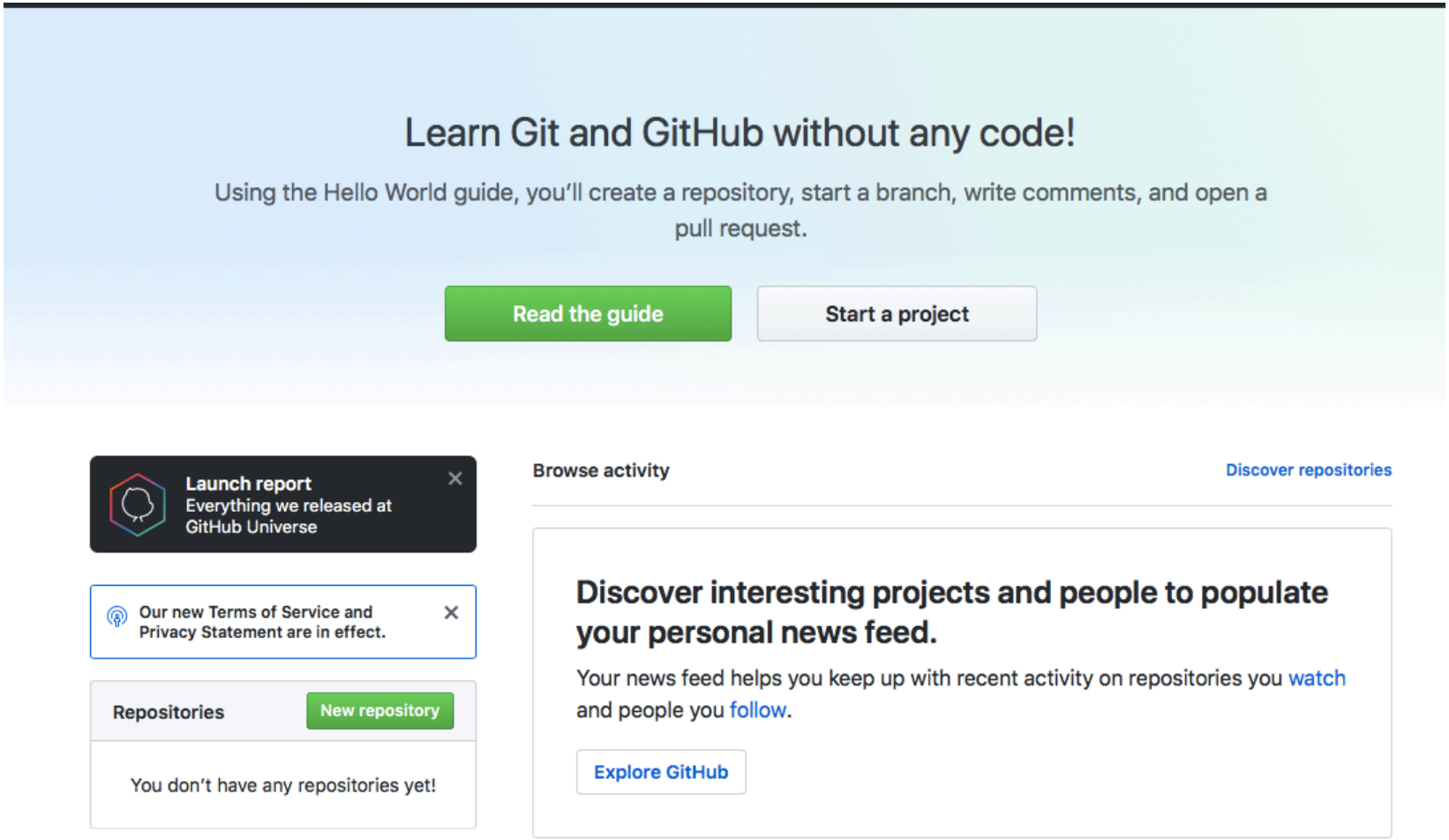
gitHUB Access



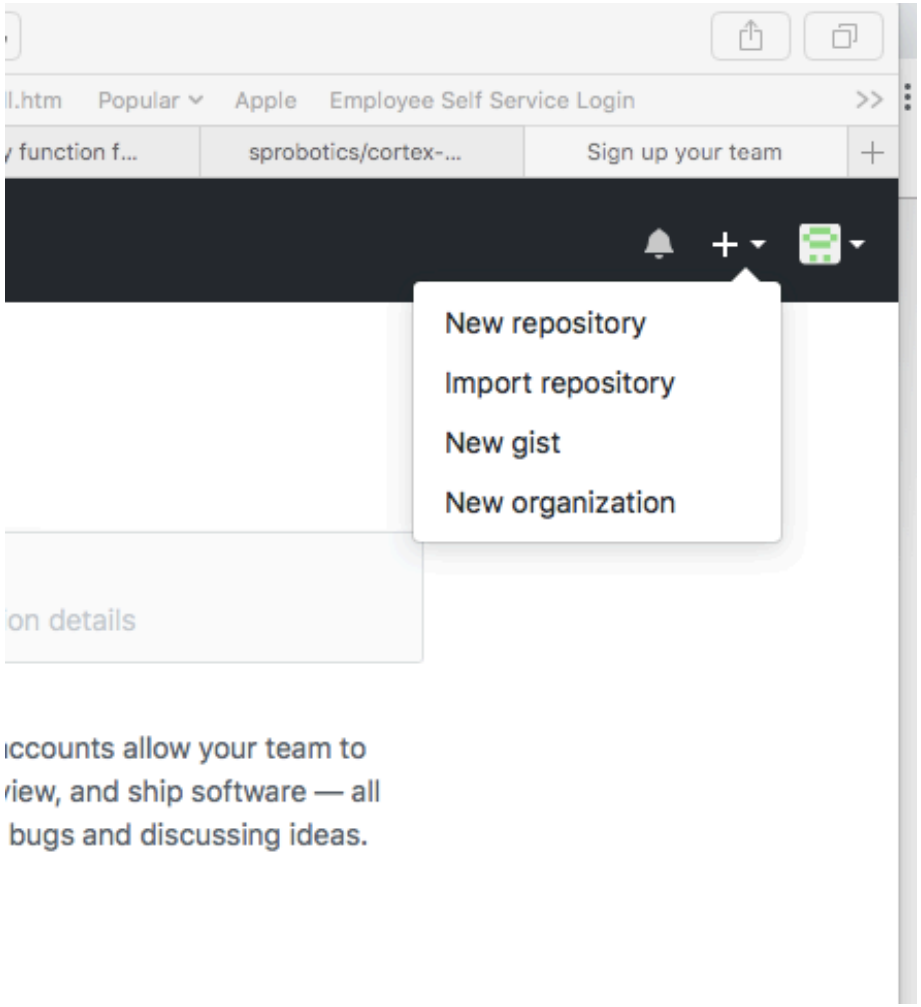
Just answer what makes sense

gitHUB Access

Create Repositories as needed or via Window gitHUB client



gitHUB Access



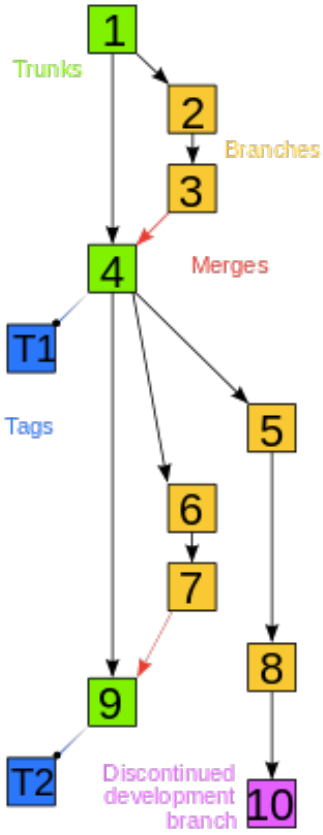
You can create an organization - being your team, and then add others to the repository to submit code changes.

gitHUB Access

- **Git** — provides source code control
- **gitHUB** — host the repositories, including documentation for your team and beyond

gitHUB Access

- **git** - source code control is a version **control** system designed to track changes in **source code** and other text files during the development of a piece of software. This allows the user to retrieve any of the previous versions of the original **source code** and the changes which are stored.



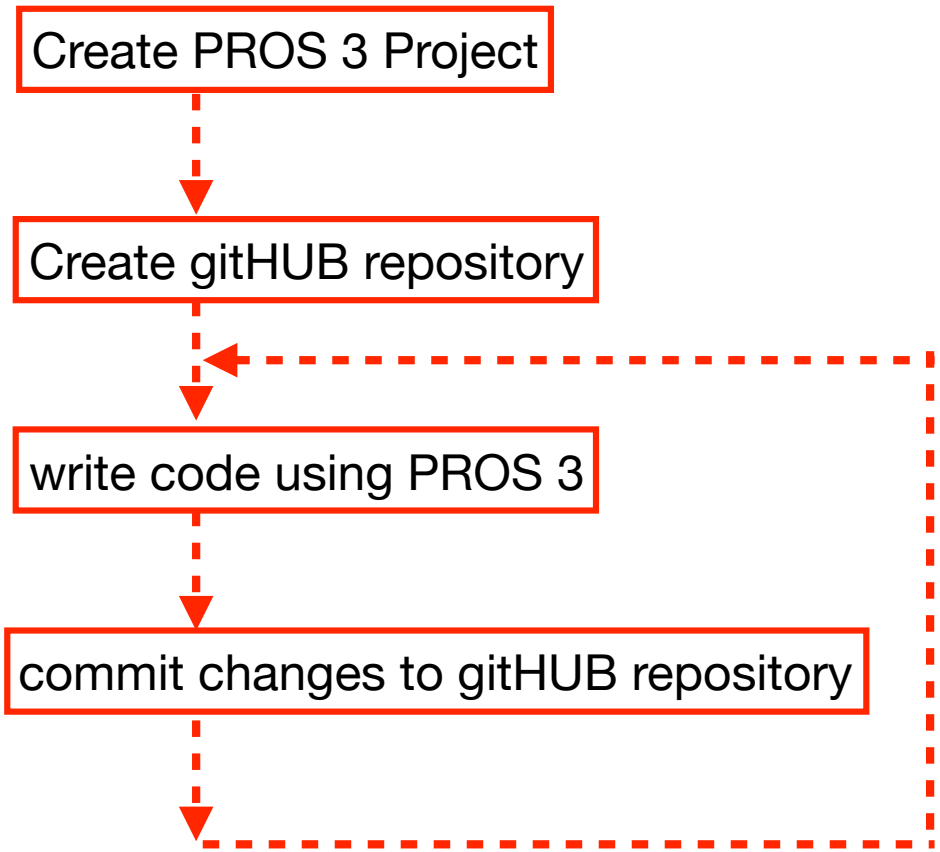
gitHUB Access

- A **repository,** or Git project, encompasses the entire collection of files and folders associated with a project, along with each file's revision history.

- The **file history** appears as snapshots in time called **commits**, and the commits exist as a linked-list relationship, and can be organized into **multiple lines of development** called **branches**.

gitHUB Access

Simple workflow of code development using gitHUB



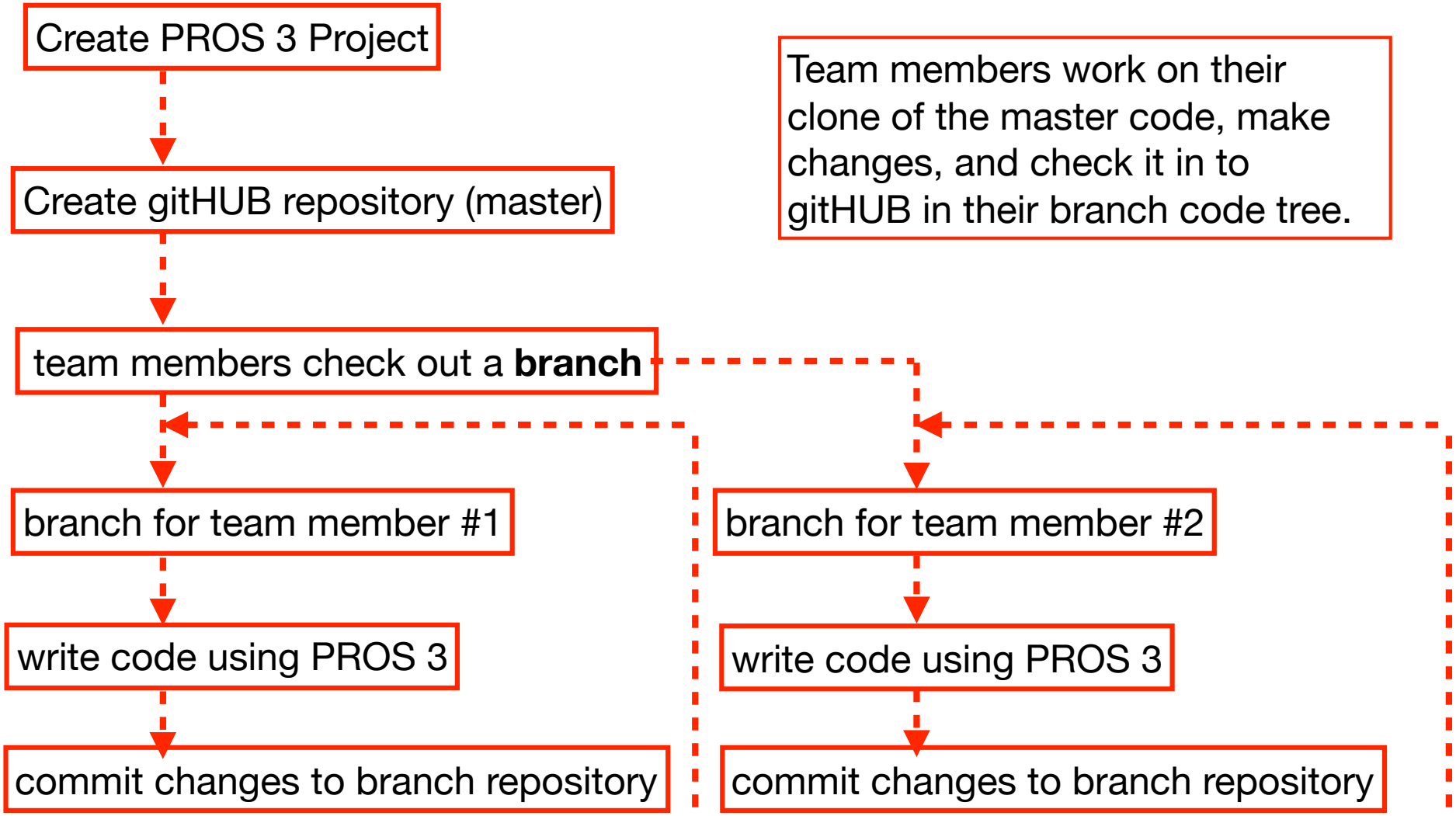
This method works well when it is a single person working on the code, it allows you to track your changes, publish simple releases (v1.0, 1.01 etc) to track your code progress.

gitHUB Access

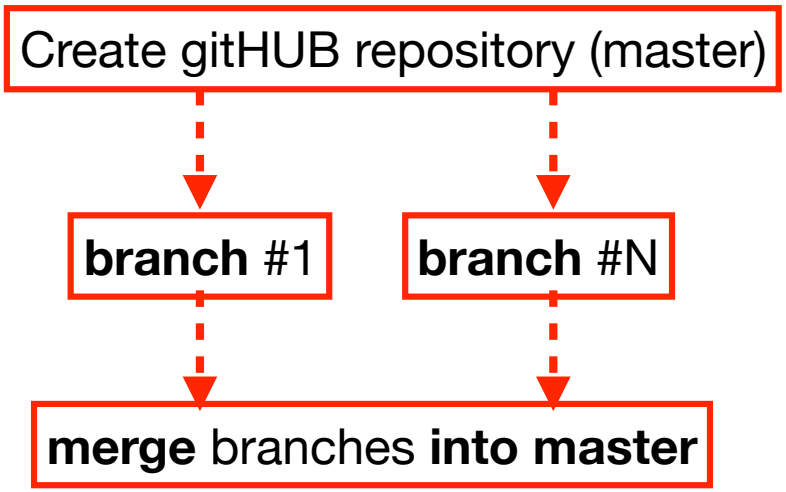
- Using **gitHUB** with multiple developers working on the same code base - this is where gitHUB / GIT's strength comes in, allowing each developer to work on the code base independently - **branches** - and then **merging** all the code together to a new agreed upon **master** version to then be **released**.

- gitHUB helps with code conflict resolutions - two or more developers submitting conflicting changes which need to be resolved.

gitHUB Access



gitHUB Access



During the merge into the master, if their are conflicts between branches, they must be resolved first prior to the merge being able to succeed.

The new master after merge will represent all agreed upon code merges.

gitHUB Access

- Once branches are **merged** into the master, one of two things can happen:
  - team members check out a **new branch** based on the newly created master
  - a **release** is created

gitHUB Access

- When to create a **release**:
  - When there is **solid** agreed upon **code base** which can be handed over **to testing**
  - Code should always be **released** for deployment to a **competition day robot**, so that any observations and new code designs can be implemented on a well defined **check point** during the development cycle.
  - Release are **solid checkpoints** you can **roll-back** to

gitHUB Access

- A **release**:
  - A **release** has a **Major** number and **Minor** Number, for example V1.0 - indicating first full release based on the **specification**.
  - Code fixed or enhanced based still on the **same specifications**, become **minor release** increments, for example V1.1, V1.2 or V1.0.1, V1.0.2
  - Code which is written as a **subsequent release** based on **new specification** should increase the Major number, for example: V2.0

gitHUB Access

- Learning more:
  - https://lab.github.com/courses
  - https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf
  -