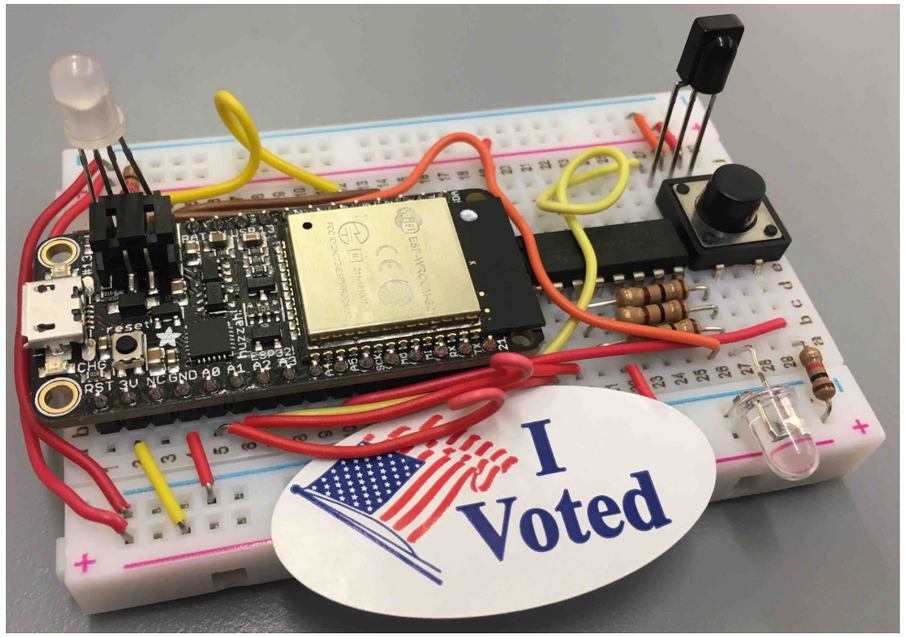


BU-EC444

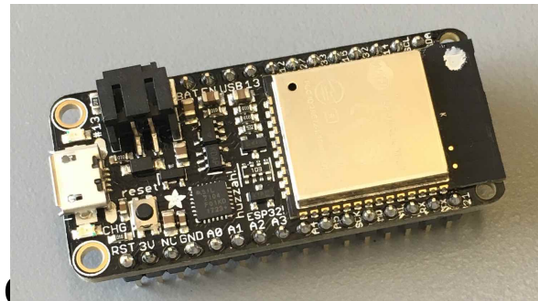
Fall 2024
Prof. Little

Quest 0: Installs and Review of Hardware

Installs – link to ESP-IDF Programming Guide

Navigate to the skill cluster for Quest 0 – Install ESP32 IDF and Toolchain

1. Pick your OS
2. Clone the ESP-IDF libraries to your laptop
3. Setup the tools
4. Note: different options for each OS
   - OSX: python, Cmake (compiling), ninja (build system), ho[illegible] (for building apps to OSX)
5. Proceed to demo builds – you are ready to flash the ESP32
6. Progress through the skills pending for Quest 0



Software Installs

- Installing the ESP Integrated Development Framework (IDF) and Toolchain on your personal laptops
- Setting up GitHub – private repos in BU-EC-444 Organization
  - Individual
  - Team (soon)
- Checking in on each install – need to be complete

Toolchain – Lots of parts but not so bad to install

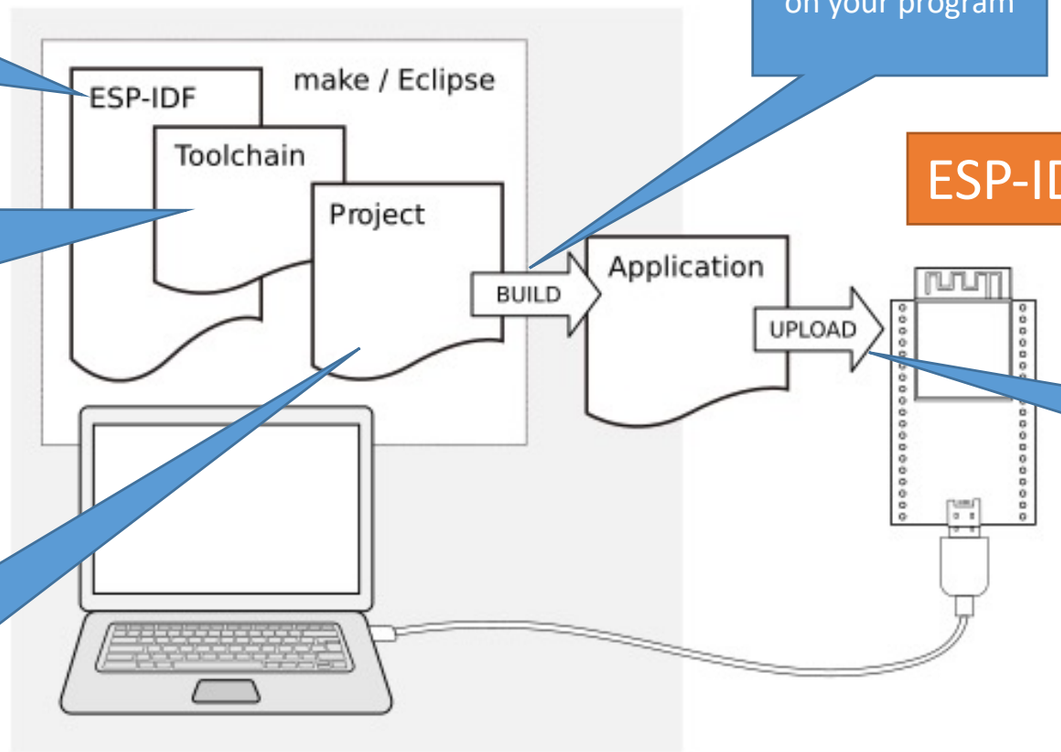
- **Toolchain** to compile code for ESP32
- **Build tools** - CMake and Ninja to build a full **Application** for ESP32
- **ESP-IDF** that essentially contains API (software libraries and source code) for ESP32 and scripts to operate the **Toolchain**
- **Text editor** to write programs (**Projects**) in C, e.g., Eclipse

C libraries and sample code here

Compiling tools

Use your editor to create a program

Execute compile on your program



Flash your micro (requires USB driver)

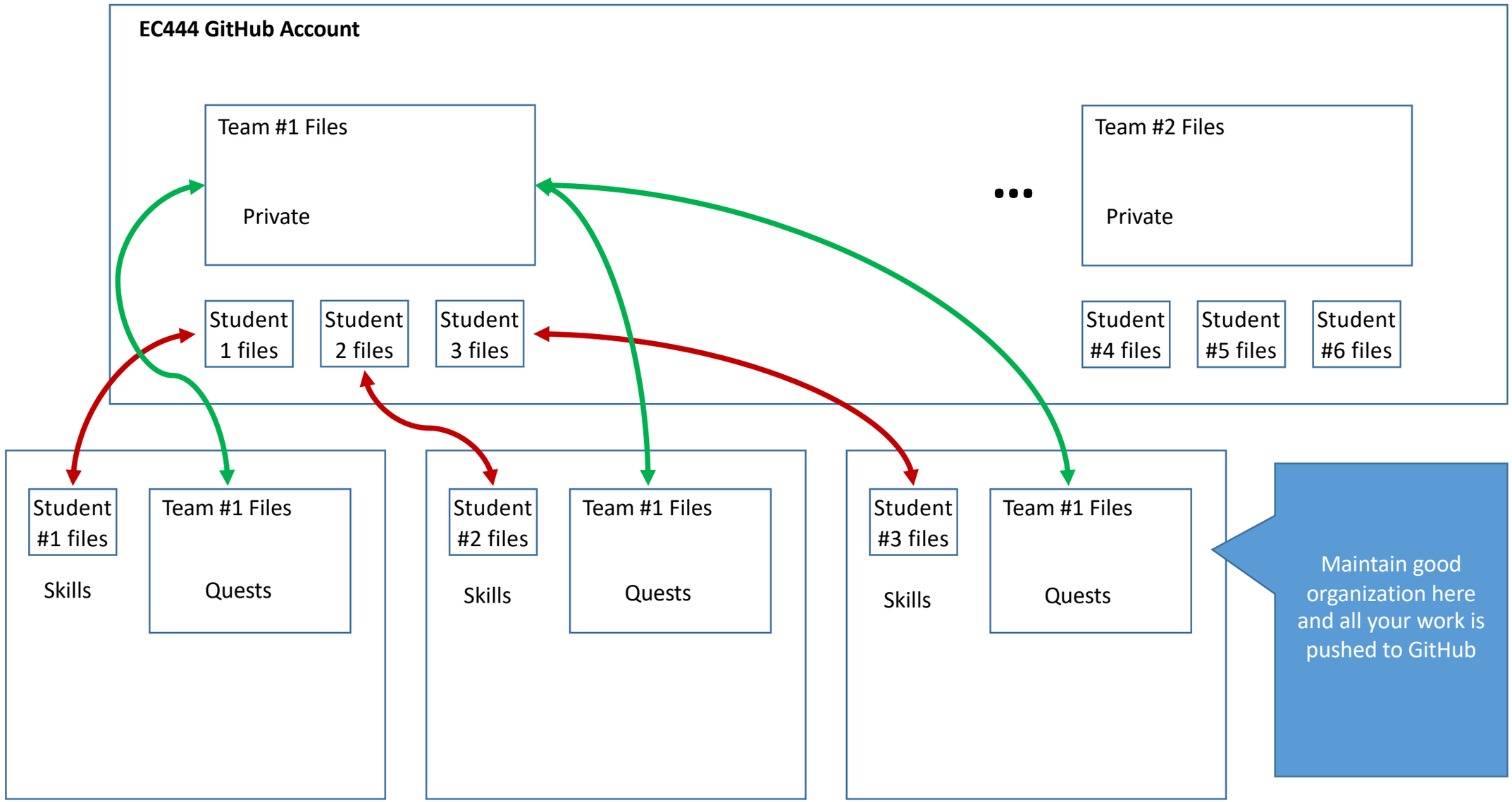
ESP-IDF == IoT Development Framework

Development of applications for ESP32

Notes on GitHub accounts

- Sign up for account if you don't have one already
- Convert to a student account – allows unlimited private repos
- Share your GitHub ID on Piazza so that we can add you to the BU-EC444 Organization
- More details found in GitHub setup skill
- We need to add you to BU-EC444 Organization so that your files are shared with the instructors (very important)
- Similarly, for sharing video on Google Drive, you need to be logged-in using your BU credentials

Reporting – using GitHub (supports markdown)

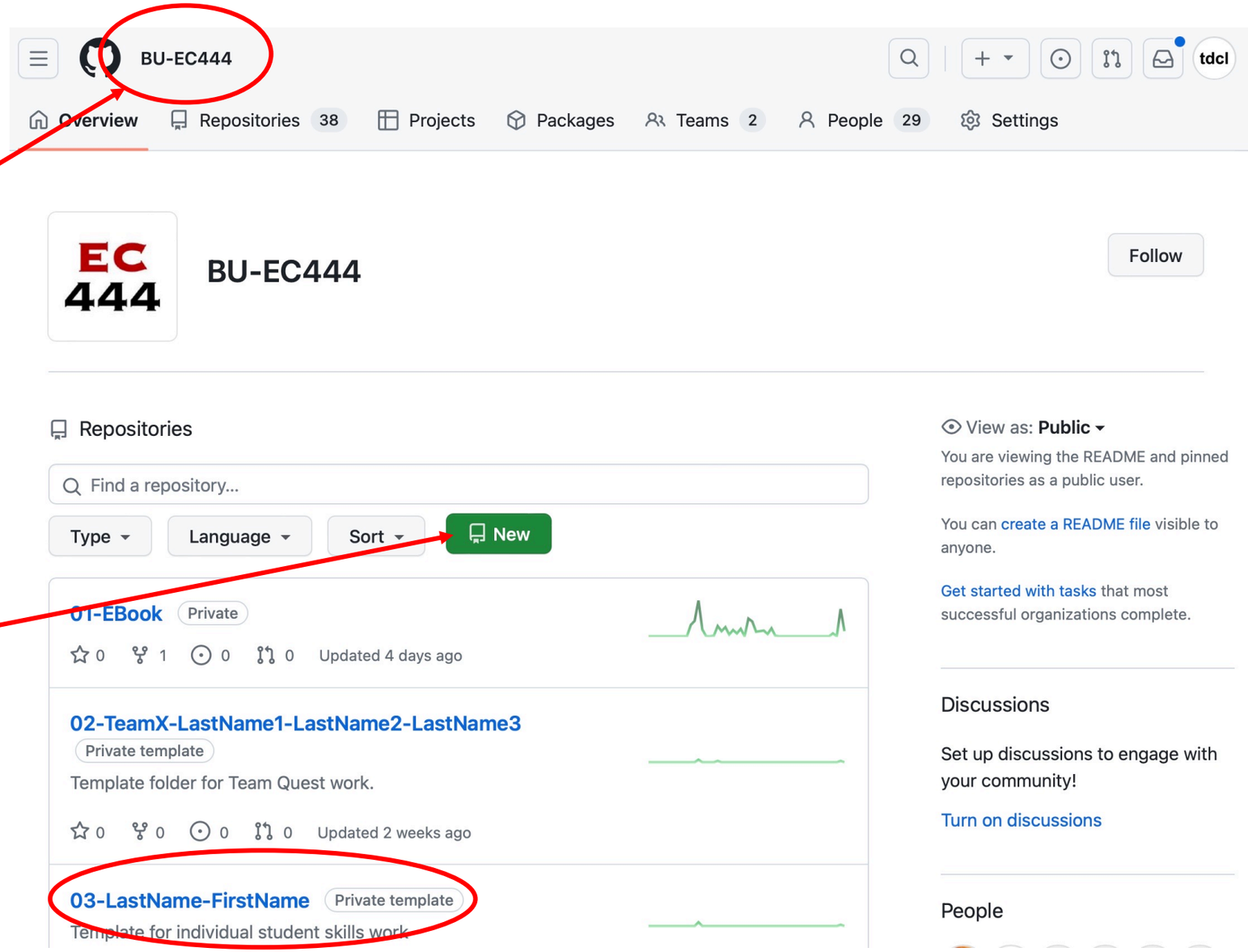


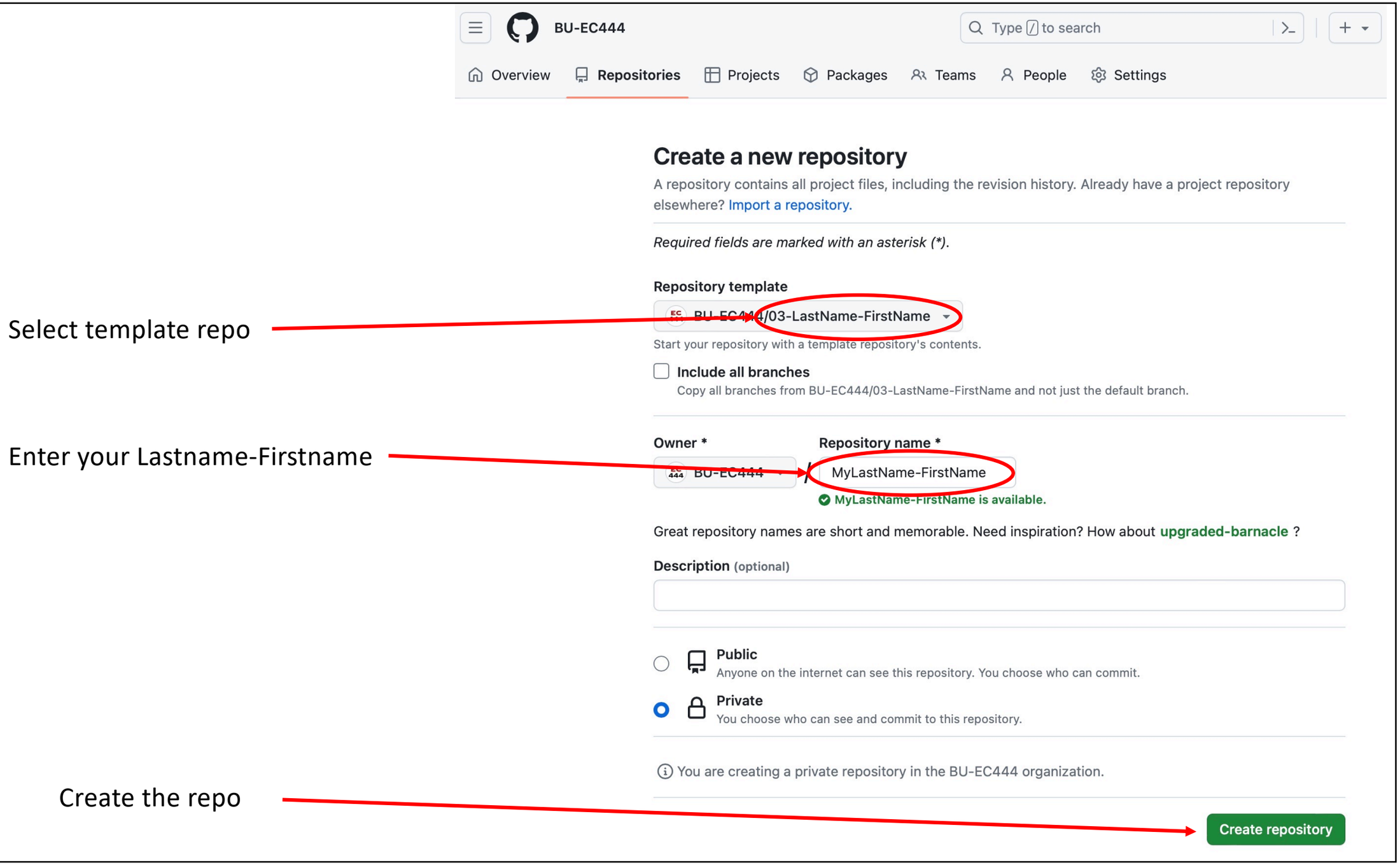
Creating GitHub repository based on template repo

Start on this page/repo

Click to create new repo

Template repo that we will use

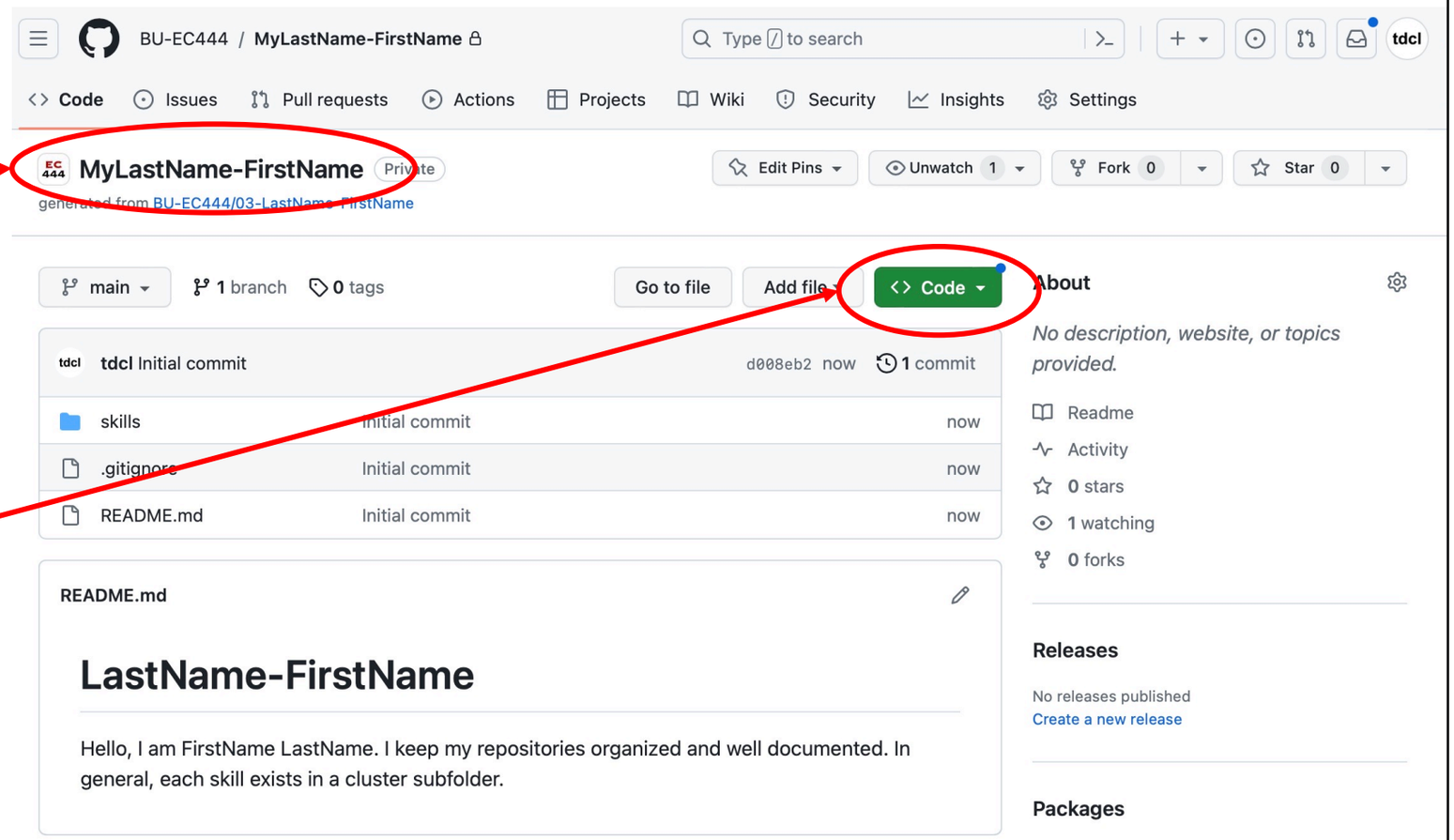



BU-EC444
Type / to search
Overview
Repositories
Projects
Packages
Teams
People
Settings
Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.
Required fields are marked with an asterisk (*).
Repository template
BU-EC444/03-LastName-FirstName
Start your repository with a template repository's contents.
Include all branches
Copy all branches from BU-EC444/03-LastName-FirstName and not just the default branch.
Owner *
Repository name *
BU-EC444
/
MyLastName-FirstName
MyLastName-FirstName is available.
Great repository names are short and memorable. Need inspiration? How about upgraded-barnacle ?
Description (optional)
Public
Anyone on the internet can see this repository. You choose who can commit.
Private
You choose who can see and commit to this repository.
You are creating a private repository in the BU-EC444 organization.
Create repository
Select template repo
Enter your Lastname-Firstname
Create the repo

Repo is now on GitHub

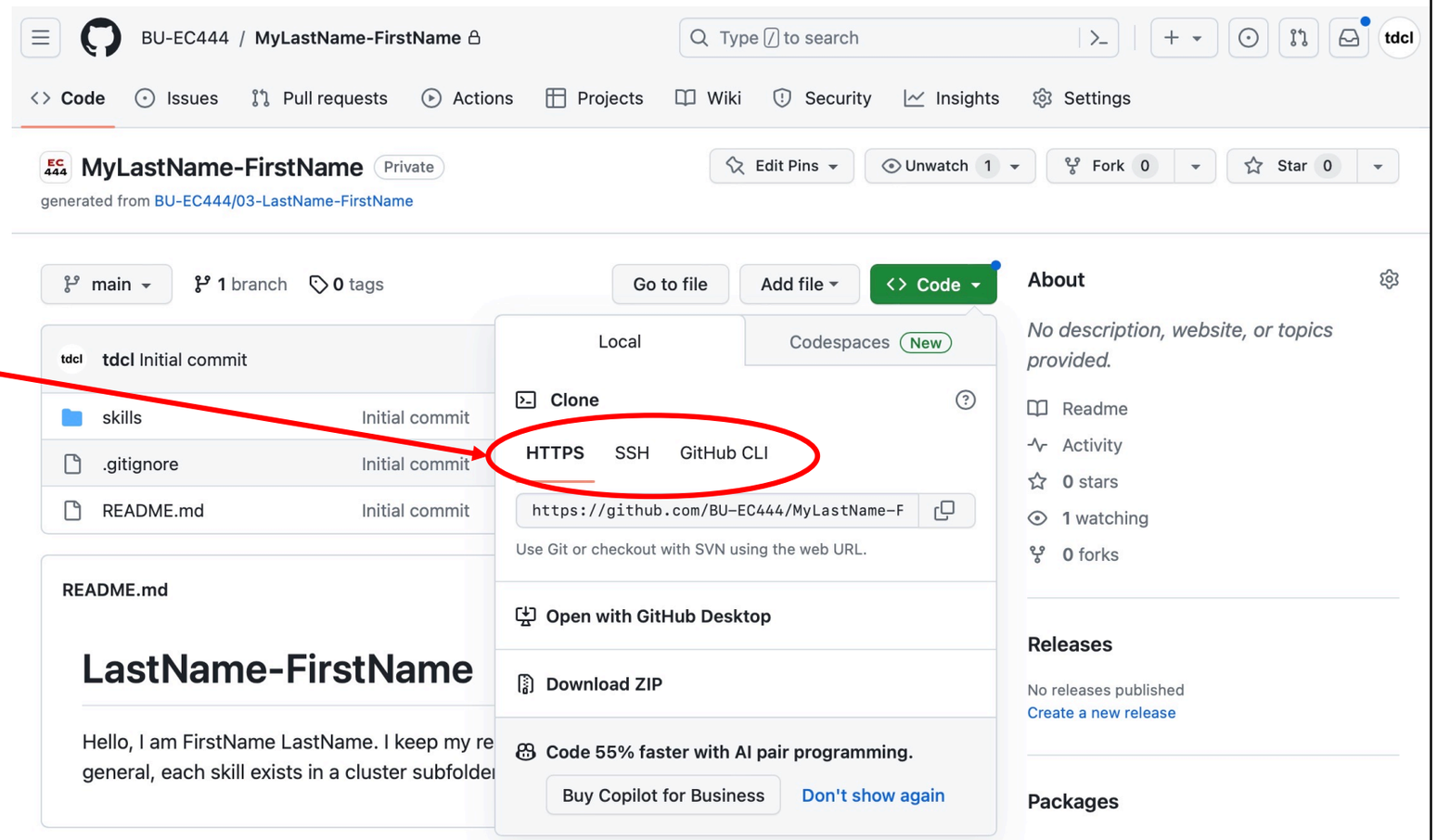
(but not on your laptop)

Press this button for clone options

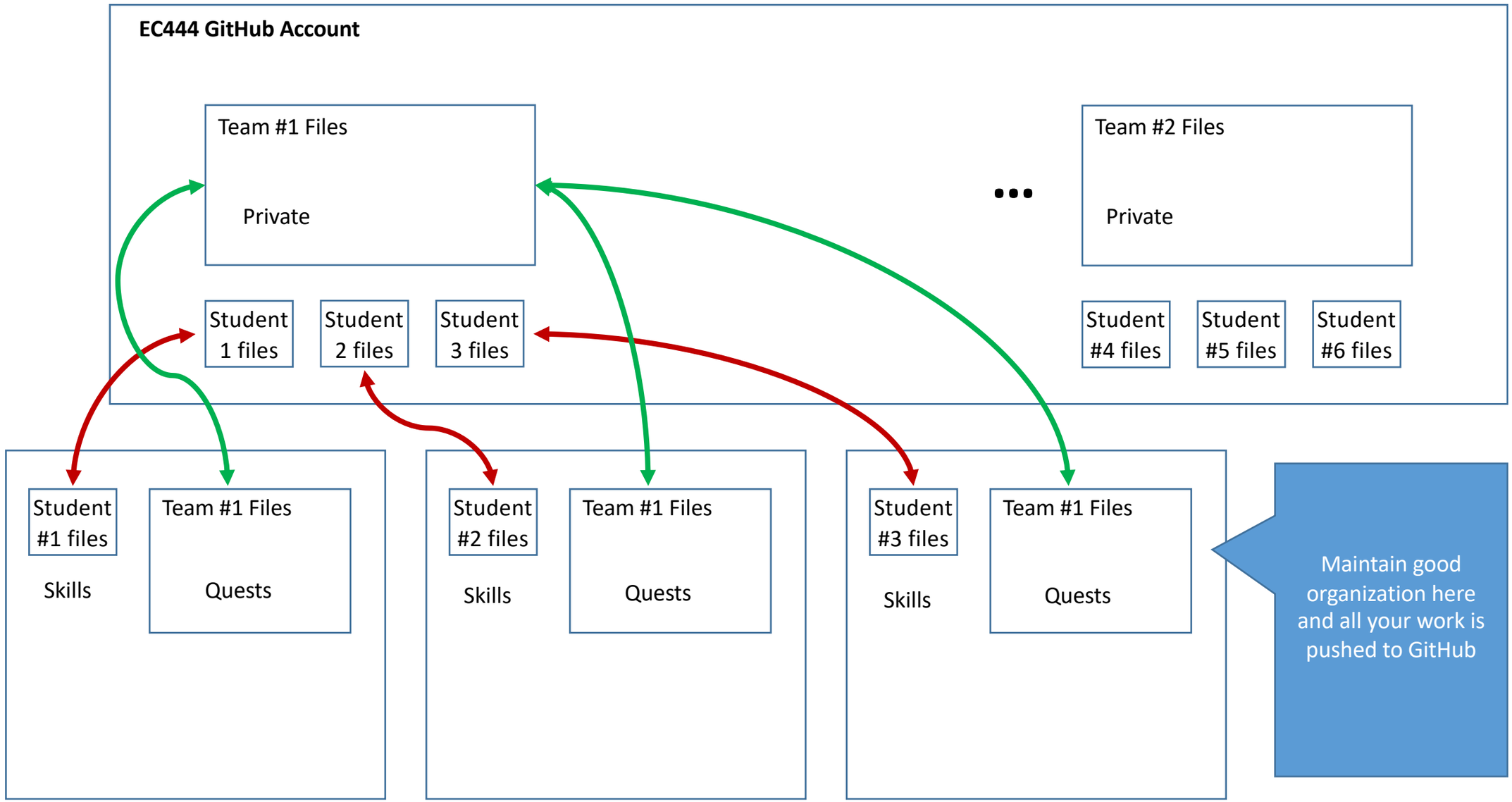


Use one of these options to download to your laptop

Then use GitHub Desktop (download) or git command line to push or pull changes from laptop to GitHub



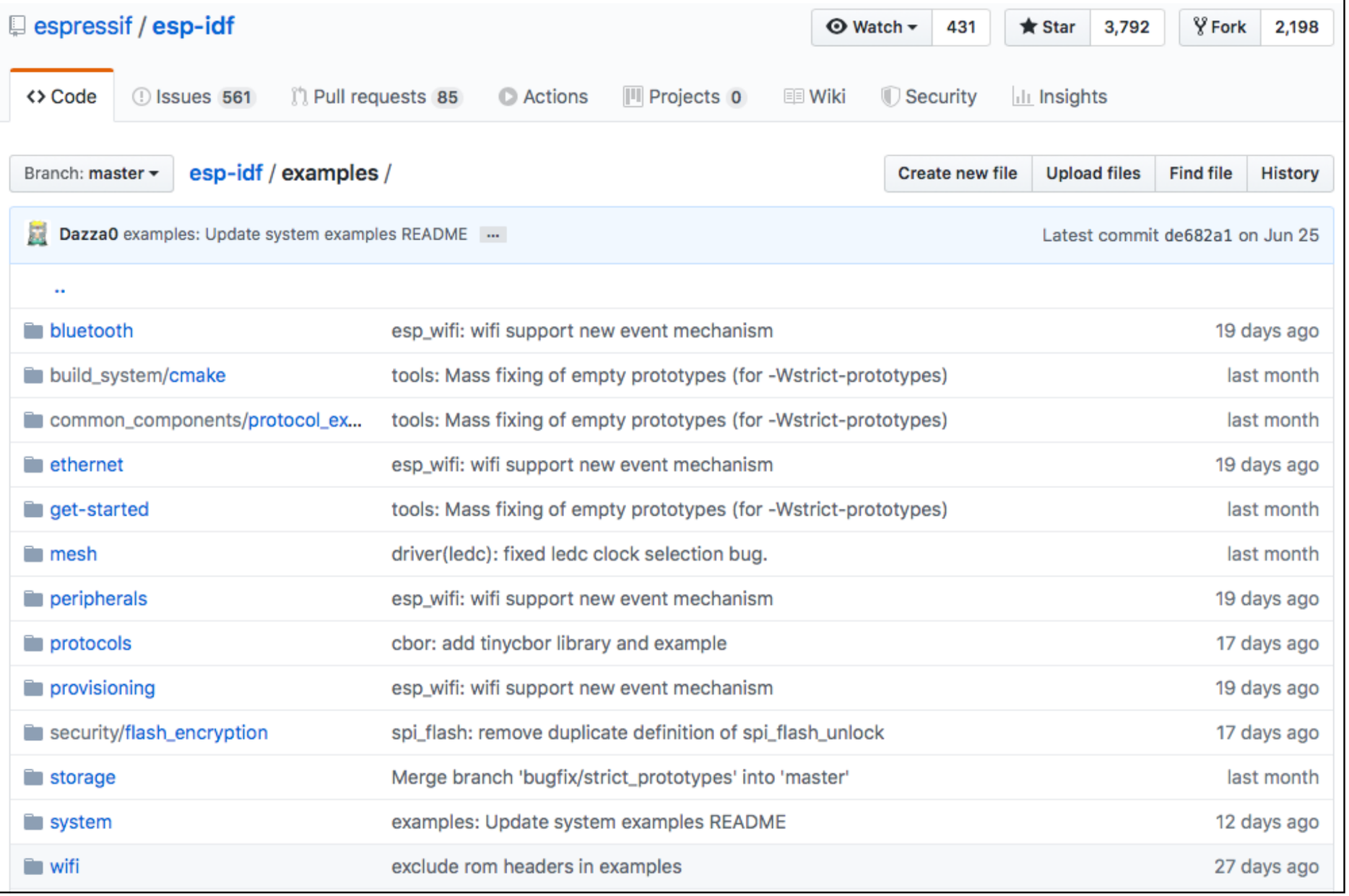
Reporting – using GitHub (supports markdown)



Review of Resources Installed

Review of ESP32 resources installed

Linked from "Utilities" And on your local repo



espressif / esp-idf

Watch 431 | Star 3,792 | Fork 2,198

Code | Issues 561 | Pull requests 85 | Actions | Projects 0 | Wiki | Security | Insights

Branch: master ▾ esp-idf / examples / Create new file | Upload files | Find file | History

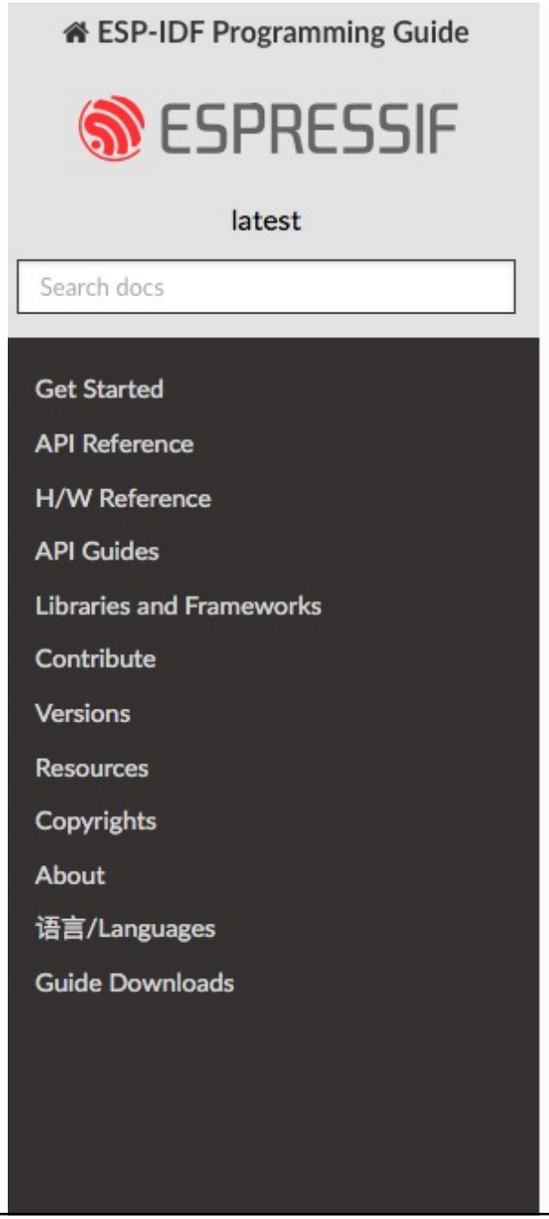
Dazza0 examples: Update system examples README ... Latest commit de682a1 on Jun 25

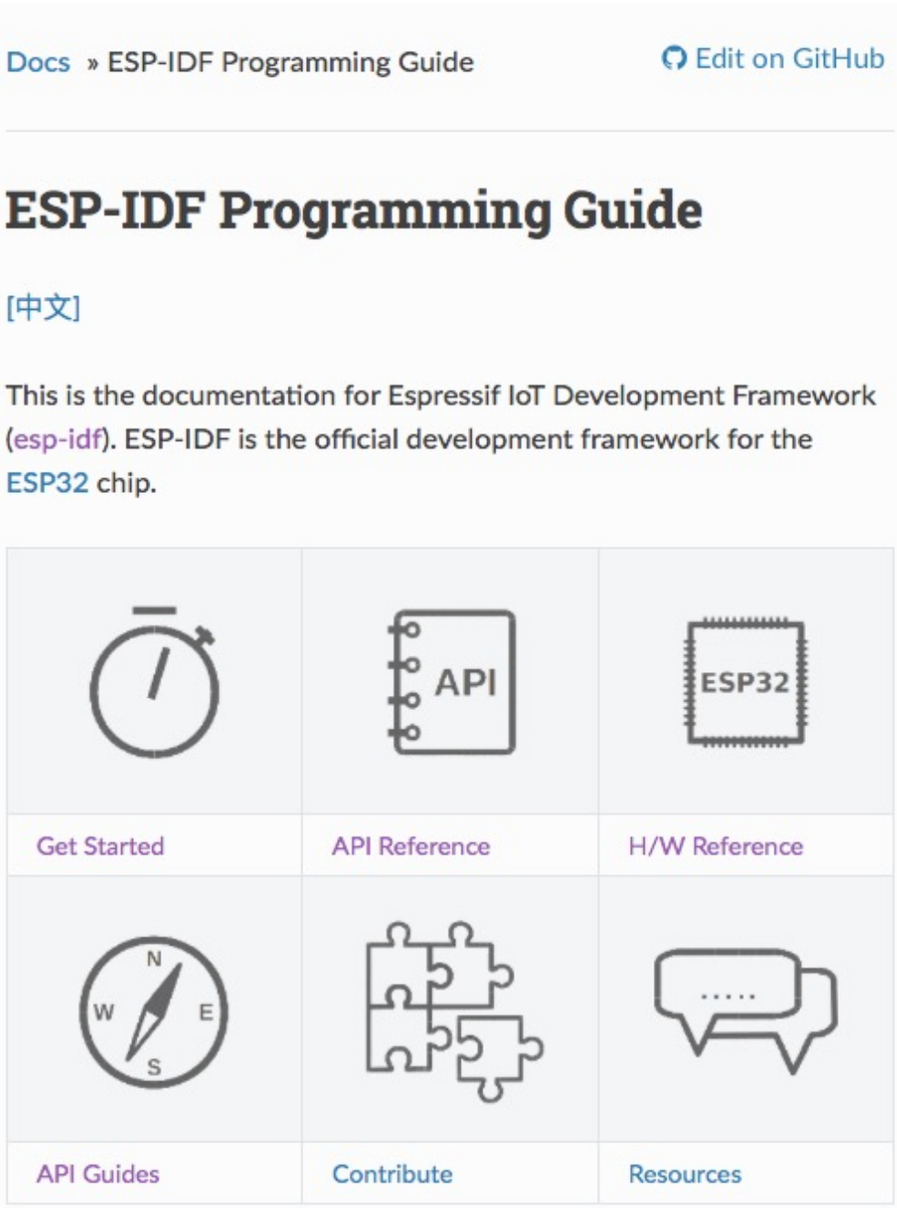
..		
bluetooth	esp_wifi: wifi support new event mechanism	19 days ago
build_system/cmake	tools: Mass fixing of empty prototypes (for -Wstrict-prototypes)	last month
common_components/protocol_ex...	tools: Mass fixing of empty prototypes (for -Wstrict-prototypes)	last month
ethernet	esp_wifi: wifi support new event mechanism	19 days ago
get-started	tools: Mass fixing of empty prototypes (for -Wstrict-prototypes)	last month
mesh	driver(ledc): fixed ledc clock selection bug.	last month
peripherals	esp_wifi: wifi support new event mechanism	19 days ago
protocols	cbor: add tinycbor library and example	17 days ago
provisioning	esp_wifi: wifi support new event mechanism	19 days ago
security/flash_encryption	spi_flash: remove duplicate definition of spi_flash_unlock	17 days ago
storage	Merge branch 'bugfix/strict_prototypes' into 'master'	last month
system	examples: Update system examples README	12 days ago
wifi	exclude rom headers in examples	27 days ago

Review of ESP32 resources installed

Linked from “Utilities”

Go-to reference Guide





Build notes – in a project directory

- Starting new project – **copy from another project and modify** (need all the pieces)
- Connect your ESP32 to the USB
- Figure out the serial port (various, usually `/dev/cu.SLAB_USBtoUART` on OS X, `COMX` on Windows, and `/dev/tty` on Linux)
- `idf.py set-target esp32` clears the build directory and regenerates `sdkconfig`
- `idf.py menuconfig` sets up your build environment including port speed
- `idf.py build` does the build
- `idf.py -p PORT flash` builds and flashes
- `idf.py -p PORT monitor` monitors serial output
- `idf.py -p PORT flash monitor` does all three
- See also:
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html#using-the-build-system
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/kconfig.html?highlight=menuconfig
  - https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/tools/idf-monitor.html
  - Can use other tools (putty, etc.)
  - Make sure the port parameters are what you expect (ESP32 and terminal program need same parameters)

Build notes – in a project directory

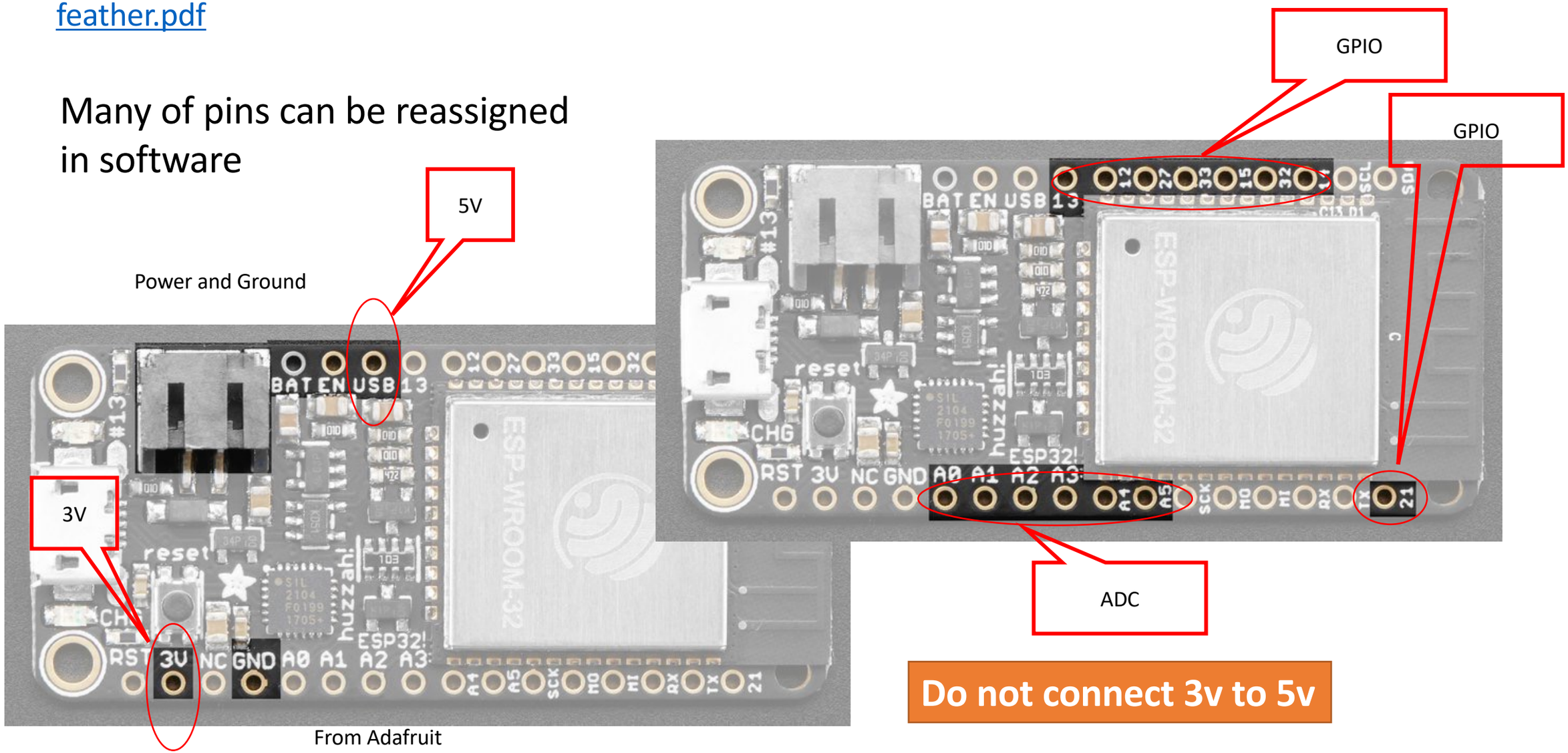
- Items that are shown in `idf.py menuconfig` are setup in `kconfig`
- We will not typically use `kconfig` but instead use `#define` in the program header to set operating values

Review of ESP32 Huzzah Board

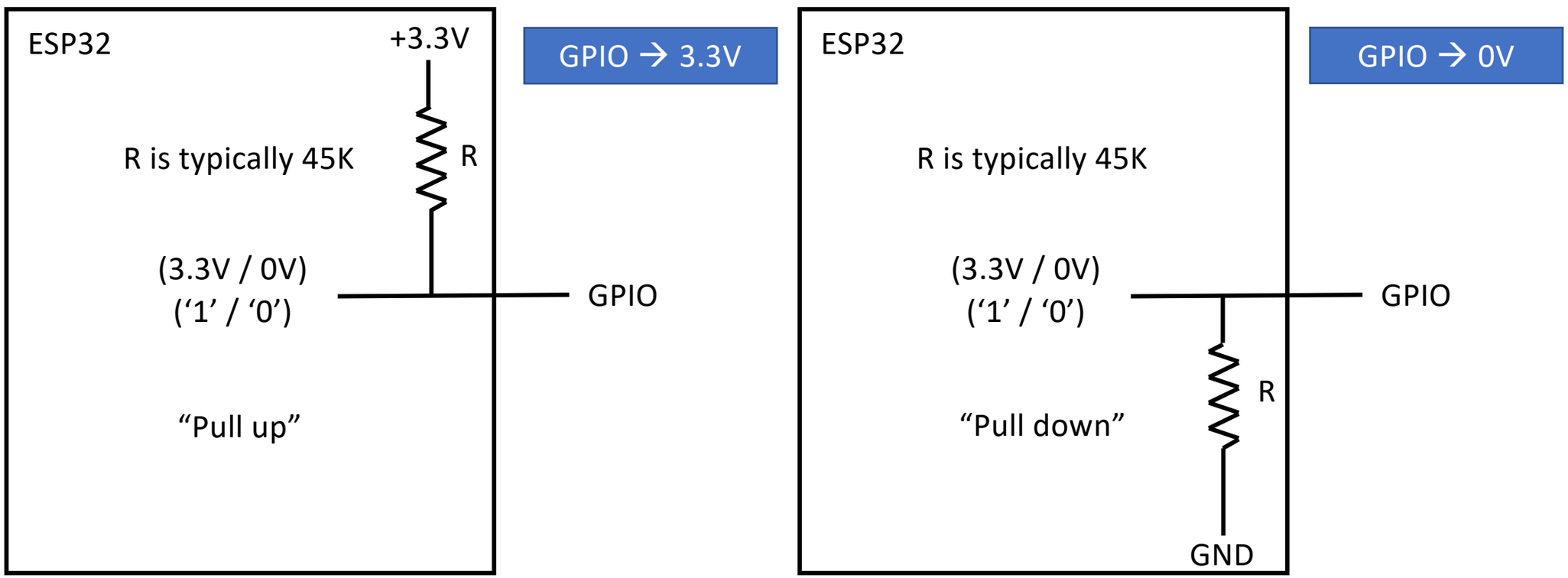
Review of the ESP32 Huzzah board

https://cdn-learn.adafruit.com/downloads/pdf/adafruit-huzzah32-esp32-feather.pdf

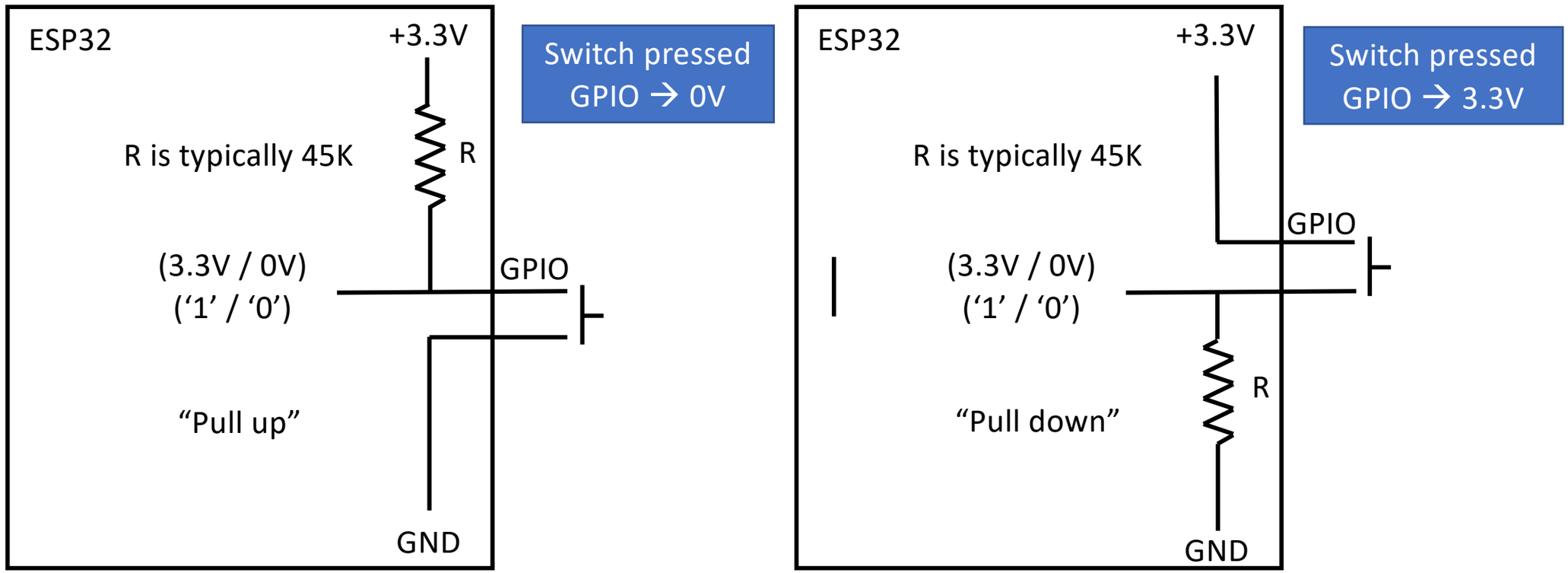
Many of pins can be reassigned in software



ESP32 ‘pull up’ and “pull down” resistors



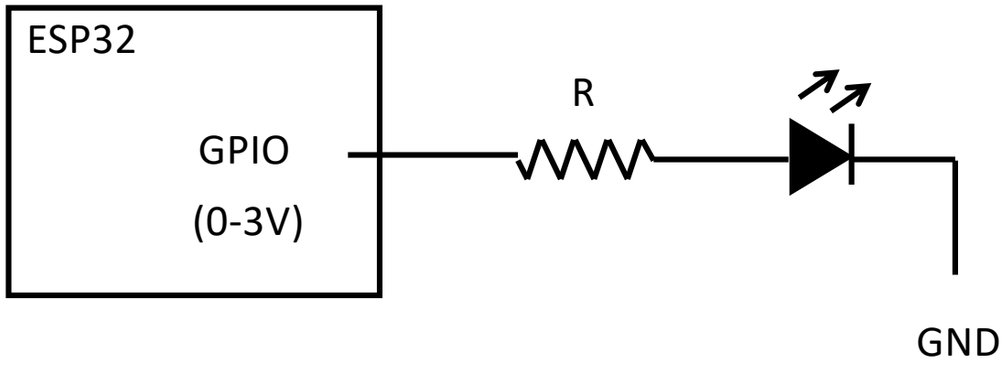
ESP32 ‘pull up’ and “pull down” resistors



Selection of ‘pull up,” “pull-down,” and “high impedance” are **software selectable**

GPIO driving an LED

Direct from GPIO as output



GPIO ‘1’: current flows through R to LED
GPIO ‘0’: current does not flow through R

R set to limit current through LED
Via Ohm’s law

Maximum current draw through GPIO pin is 40mA

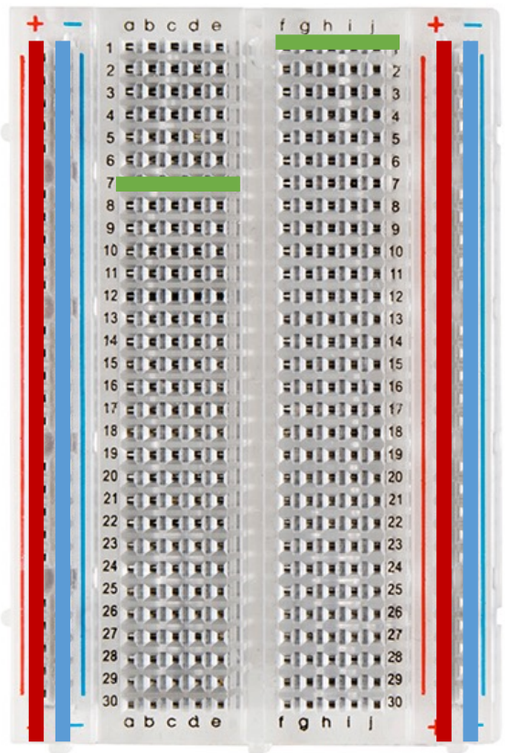
Maximum current sourced from ESP32 is ~250mA of the total of 500mA from the USB cable

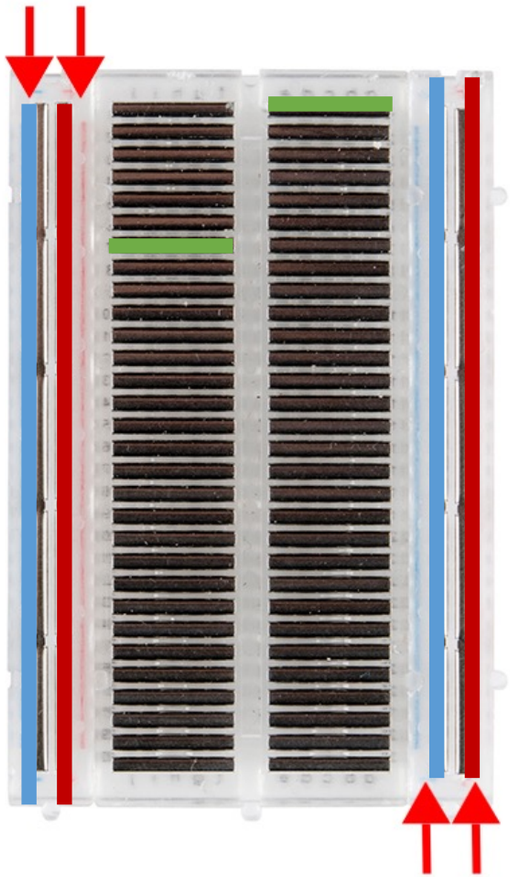
Overloading your computer’s USB current limit can shut down your port (Macs require reboot)

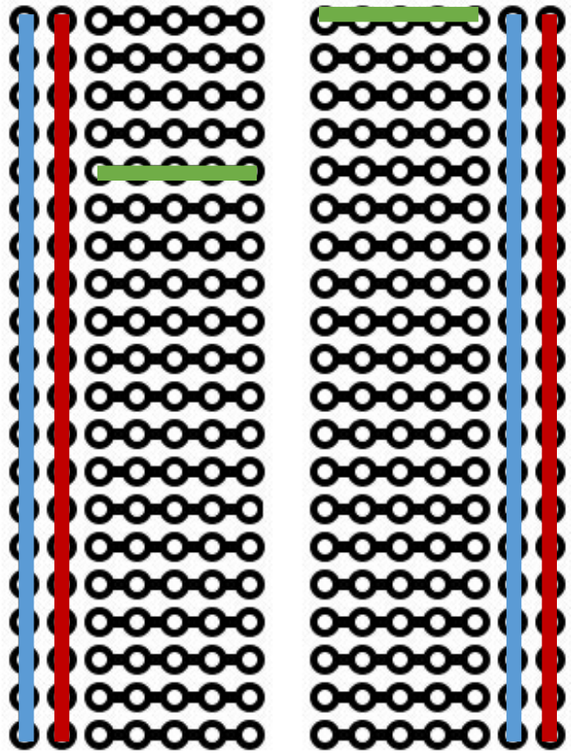
Greater than 40mA or for different voltage levels (e.g., motors)
→ use external driver circuit (e.g., H-bridge, included in kit)

Breadboards

Anatomy of a Breadboard

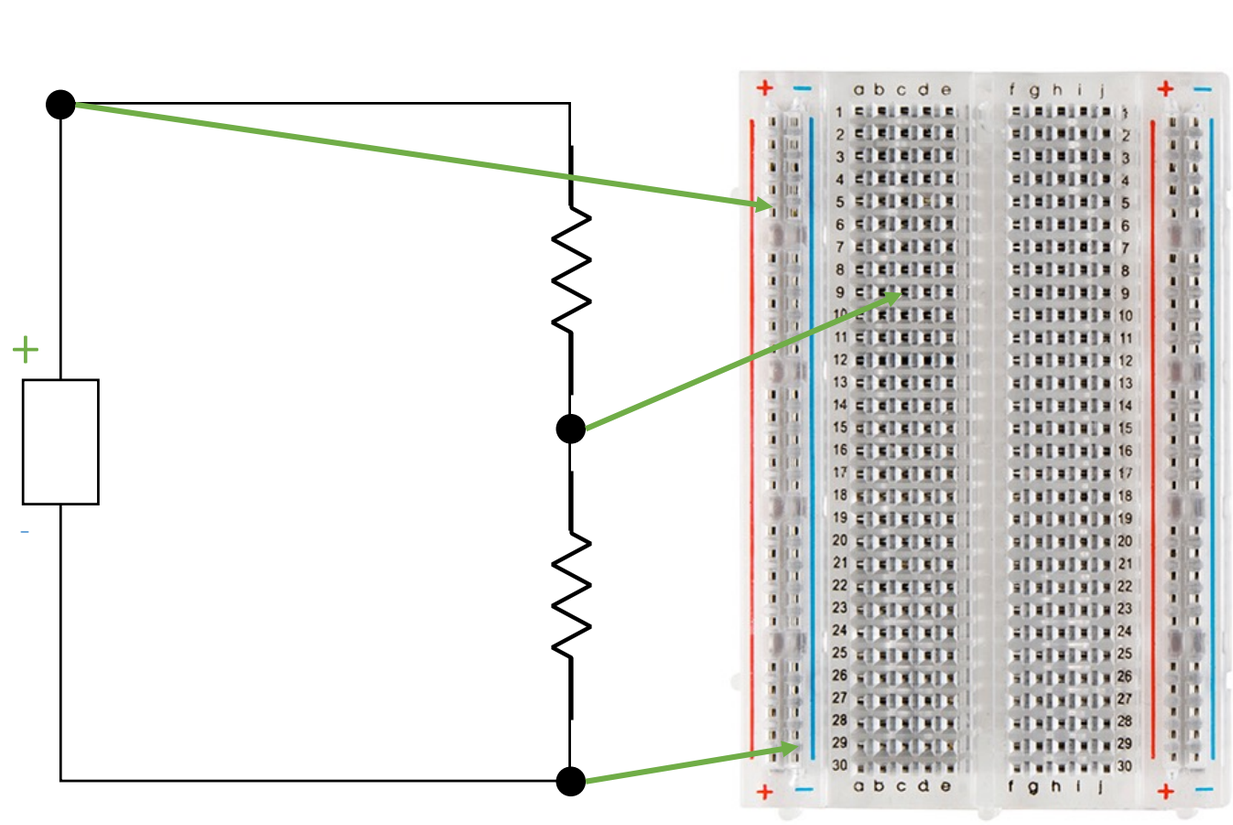


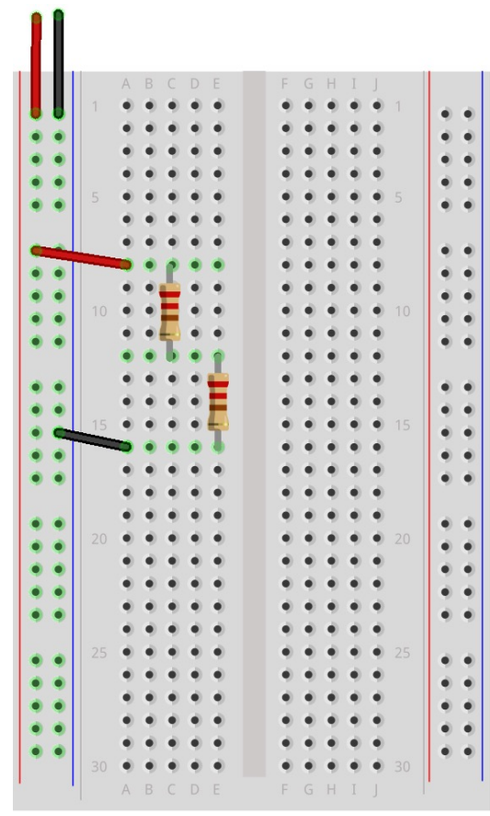




Schematics and Breadboards

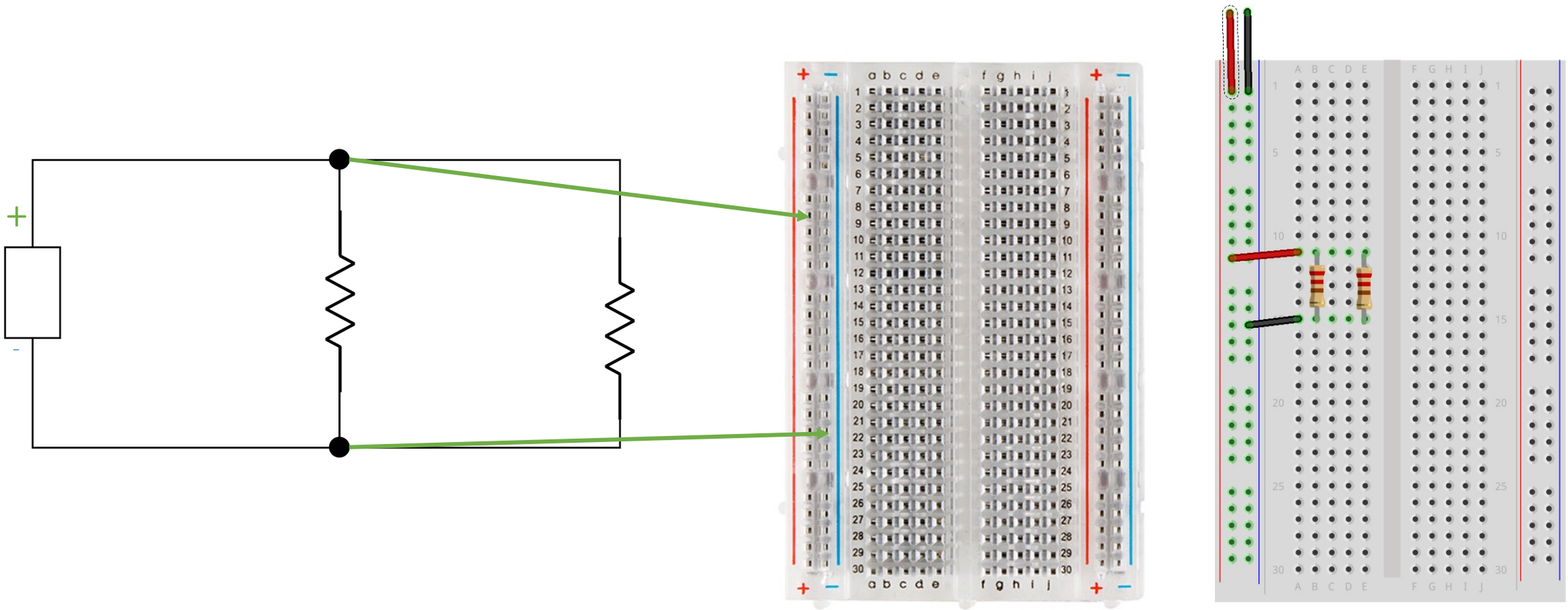
- *Connect nodes of a schematic to a connected row of the breadboard*





Schematics and Breadboards

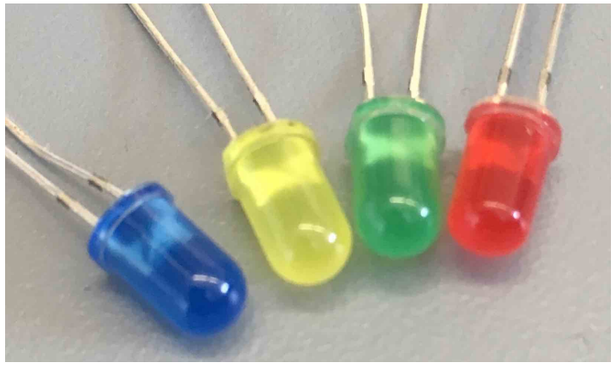
- *Connect nodes of a schematic to a connected row of the breadboard*



GPIO Control of LEDs:
Binary Counting Exercise

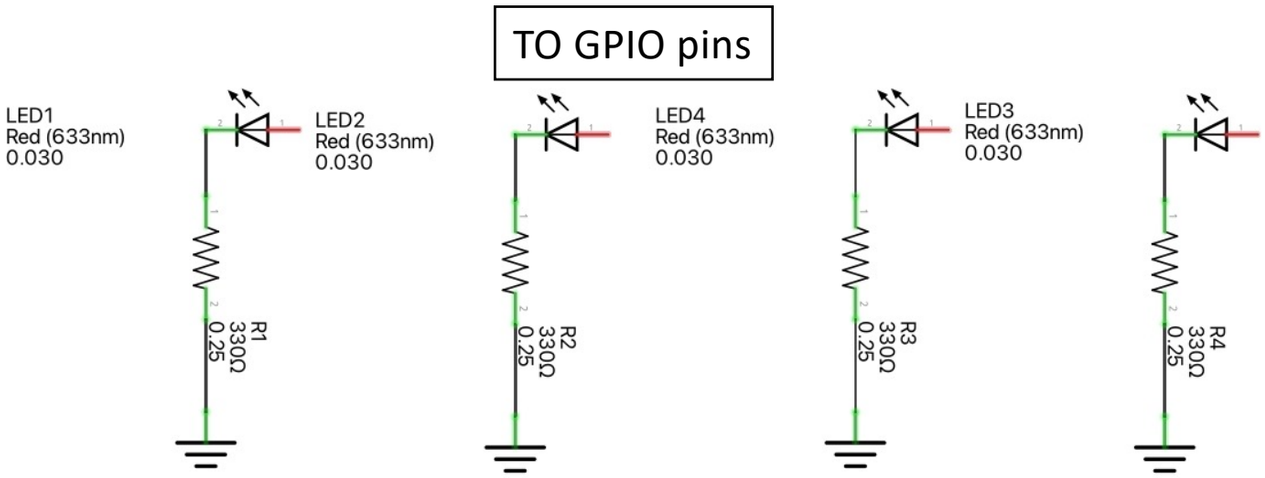
Skills: GPIO on LEDs

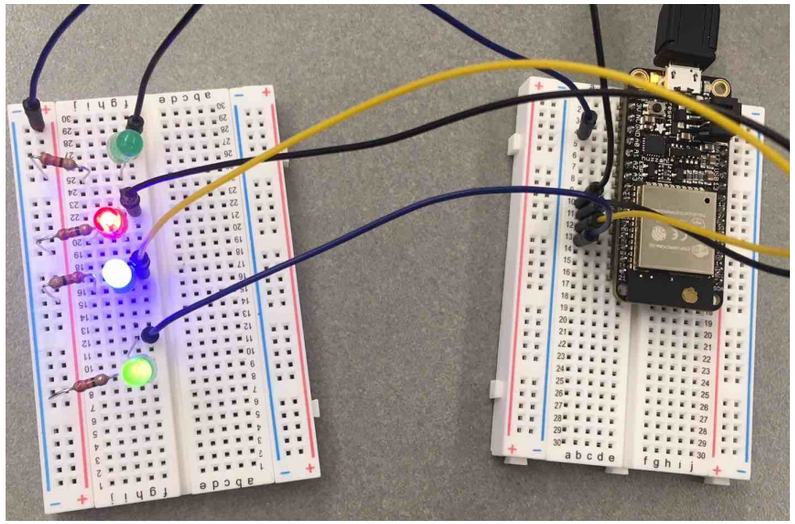
Goal is to toggle GPIO signals on and off to control a set of LEDs



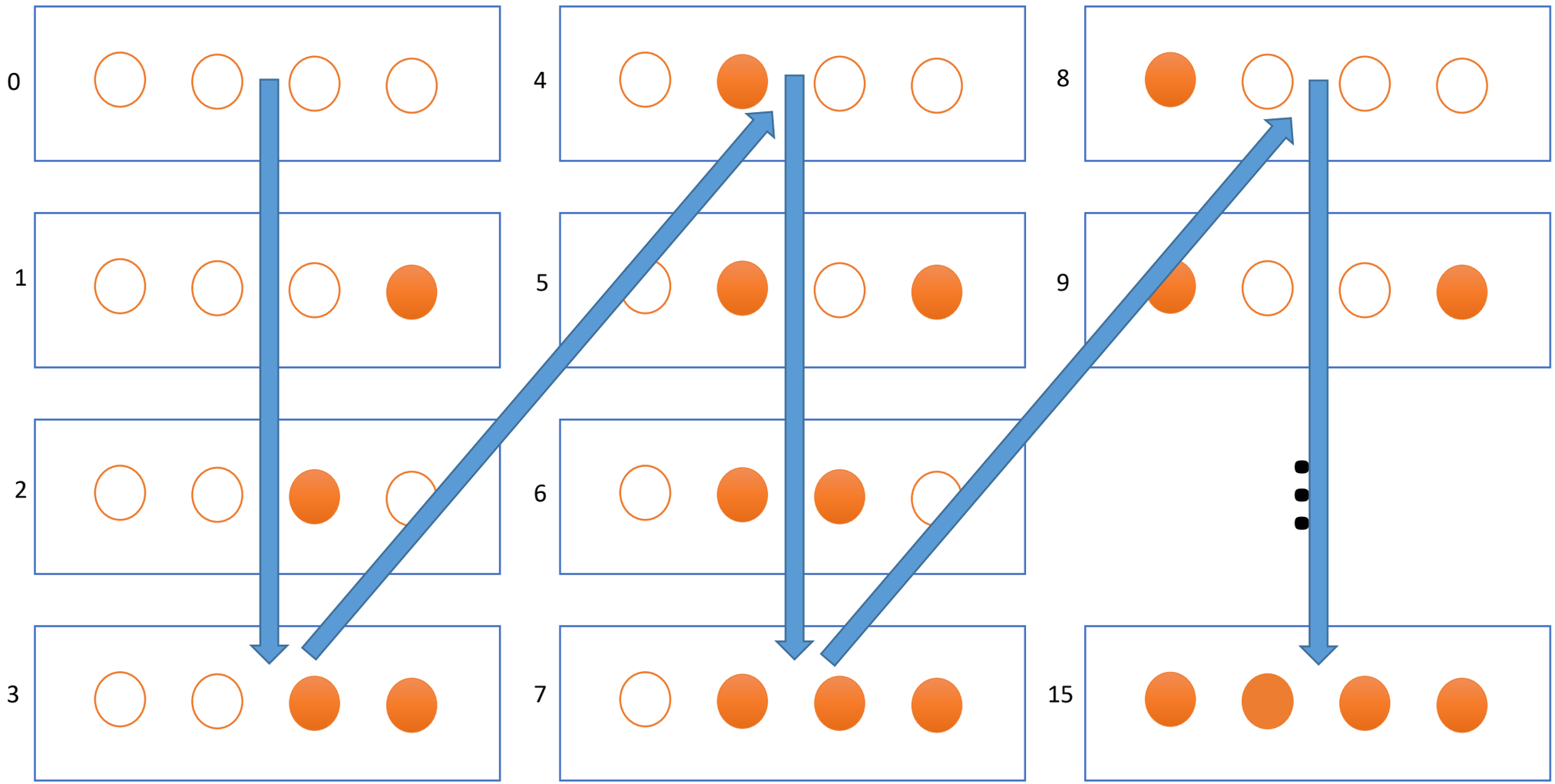
- Wire up 4 LEDs on your breadboard
  - Pick 4 open GPIO pins
  - Use resistors so that I <= max current mA (I=V/R), V is your source voltage (use 330 ohm) at 3.3V
  - Plug the ESP32 into a USB power source

- Drive a blink-type program on the set of 4 LEDs per the posted skill assignment





GPIO: Count from 0 to 15 in binary – then wrap to 0



GPIO: Blink example is a pretty good starting place

```c
#define BLINK_GPIO CONFIG_BLINK_GPIO

void app_main(void)
{
    gpio_pad_select_gpio(BLINK_GPIO);
    /* Set the GPIO as an output */
    gpio_set_direction(BLINK_GPIO, GPIO_MODE_OUTPUT);

    while(1) {
        printf("Turning off the LED\n");
        gpio_set_level(BLINK_GPIO, 0);              /* Blink off (output low) */
        vTaskDelay(1000 / portTICK_PERIOD_MS);      /* Wait (block this task) 1 s */
        printf("Turning on the LED\n");
        gpio_set_level(BLINK_GPIO, 1);              /* Blink on (output high) */
        vTaskDelay(1000 / portTICK_PERIOD_MS);      /* Wait again */
    }
}
```

This program uses delay waiting (within a single task) to achieve timing