Smart Software Project

Lecture: Week 2
Embedded System & Source Control

Prof. HyungJune Lee
hyungjune.lee@ewha.ac.kr

이화여자대학교
EWHA WOMANS UNIVERSITY

Today

- Review from the last lecture
  - Arduino platform
  - Arduino programming environment

- Embedded system overview
- Processor technologies
- Collaborative programming environment
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.

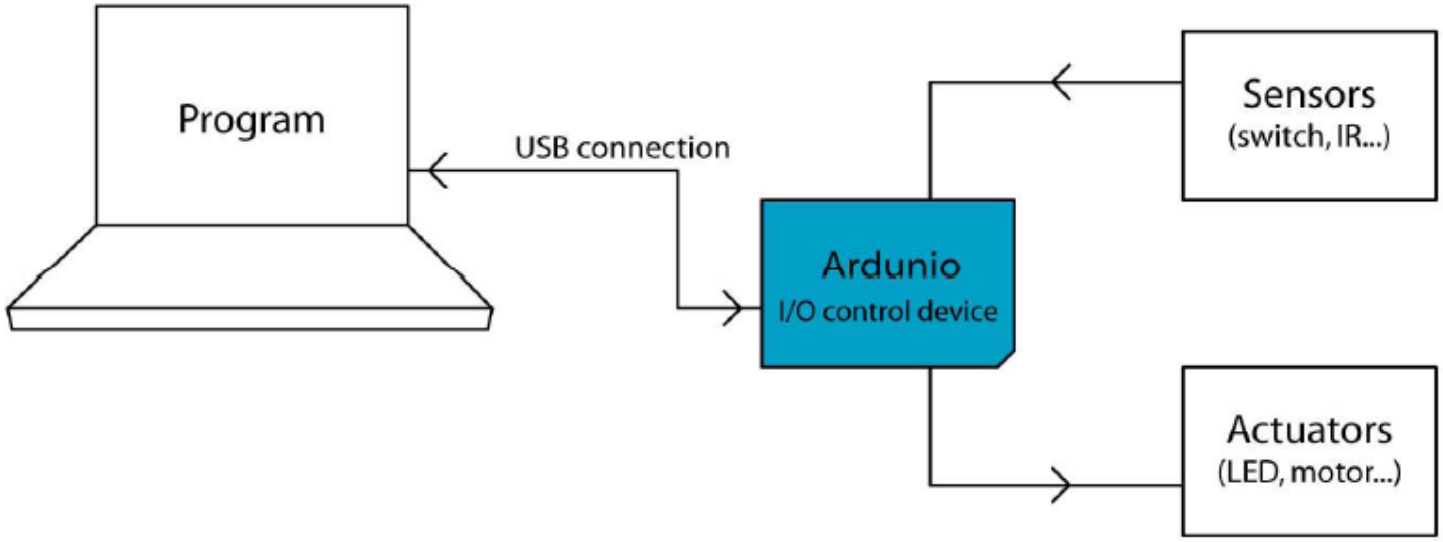
- Announcement

Class Schedule

Week	Lecture Contents	Lab Contents
Week 1	Course introduction	Arduino introduction: platform & programming environment
Week 2	Embedded system overview & source management in collaborative repository (using GitHub)	Lab 1: Arduino Mega 2560 board & SmartCAR platform
Week 3	ATmega2560 Micro-controller (MCU): architecture & I/O ports, Analog vs. Digital, Pulse Width Modulation	Lab 2: SmartCAR LED control
Week 4	Analog vs. Digital & Pulse Width Modulation	Lab 3: SmartCAR motor control (Due: HW on creating project repository using GitHub)
Week 5	ATmega2560 MCU: memory, I/O ports, UART	Lab 4: SmartCAR control via Android Bluetooth
Week 6	ATmega2560 UART control & Bluetooth communication between Arduino platform and Android device	Lab 5: SmartCAR control through your own customized Android app (Due: Project proposal)
Week 7	Midterm exam	
Week 8	ATmega2560 Timer, Interrupts & Ultrasonic sensors	Lab 6: SmartCAR ultrasonic sensing
Week 9	Infrared sensors & Buzzer	Lab 7: SmartCAR infrared sensing
Week 10	Acquiring location information from Android device & line tracing	Lab 8: Implementation of line tracer
Week 11	Gyroscope, accelerometer, and compass sensors	Lab 9: Using gyroscope, accelerometer, and compass sensors
Week 12	Project	Team meeting (for progress check)
Week 13	Project	Team meeting (for progress check)
Week 14	Course wrap-up & next steps	
Week 15	Project presentation & demo I (Due: source code, presentation slides, & poster slide)	Project presentation & demo II
Week 16	Final week (no final exam)	

Arduino Platform

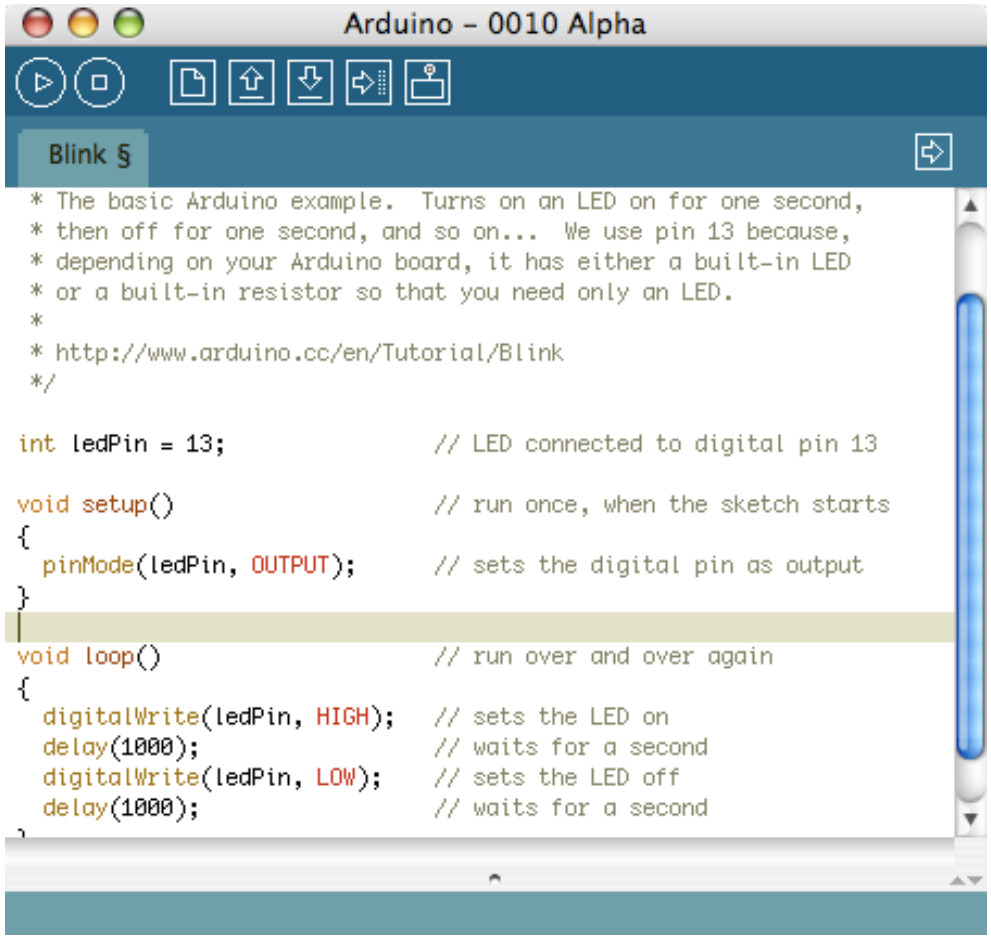
- Arduino Platform Basic Configuration



Arduino Terminology

- "sketch" – a program you write to run on an Arduino board
- "pin" – an input or output connected to something
  - Output to an LED
  - Input from temperature sensor
- "digital" – value is either HIGH or LOW
  - a.k.a. on/off, one/zero
- "analog" – value ranges, usually from 0 – 255
  - LED brightness, motor speed, etc.

Arduino Software



- Like a text editor
- View/write/edit "sketches"
- Then you program them into hardware

Standard Library

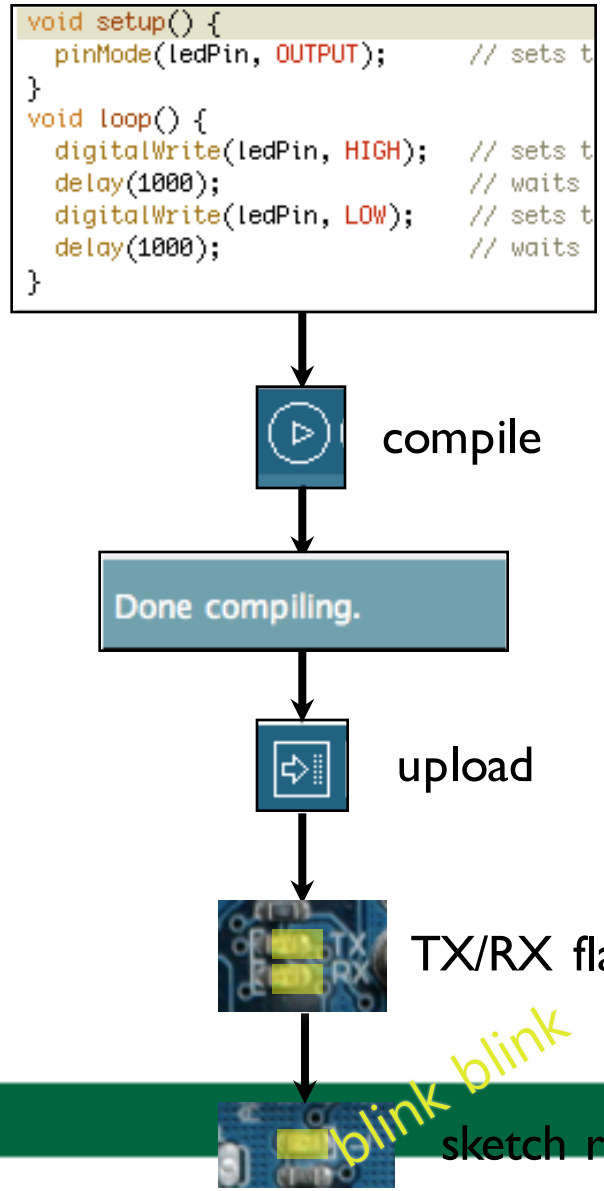
- Main Functions
  - setup()
    - Used for one-time function calls
      - Initializing pins/serial ports
    - Takes and returns no arguments
  - loop()
    - Continuous loop function (similar to while(1))
    - Takes and returns no arguments

Standard Library

- I/O Functions
  - Set pin mode as input or output
    - pinMode(pin, value) – set a pin as INPUT or OUTPUT (in value)
  - Analog I/O
    - analogRead(pin) – read an analog pin through ADC
    - analogWrite(pin, value) – write an "analog" value using PWM
  - Digital I/O
    - digitalRead(pin) – read a digital pin's state
    - digitalWrite(pin, value) – set a digital pin to HIGH or LOW
  - Timing
    - delay(time) – wait an amount of time in millisecond
    - millis() – get the number of milliseconds after beginning running the current program
  - Serial Communication
    - Begin, read, write, print, etc.

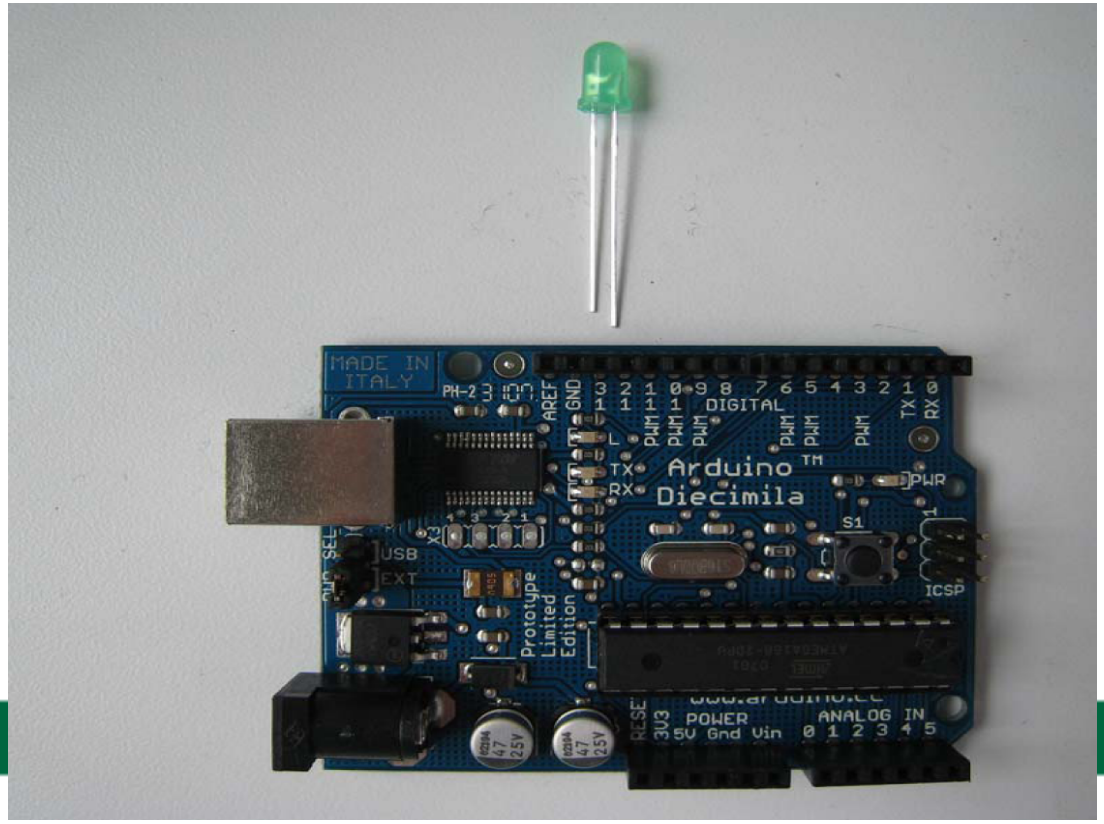
Using Arduino

- Write your sketch
- Press Compile button (to check for errors)
- Press Upload button to program Arduino board with your sketch



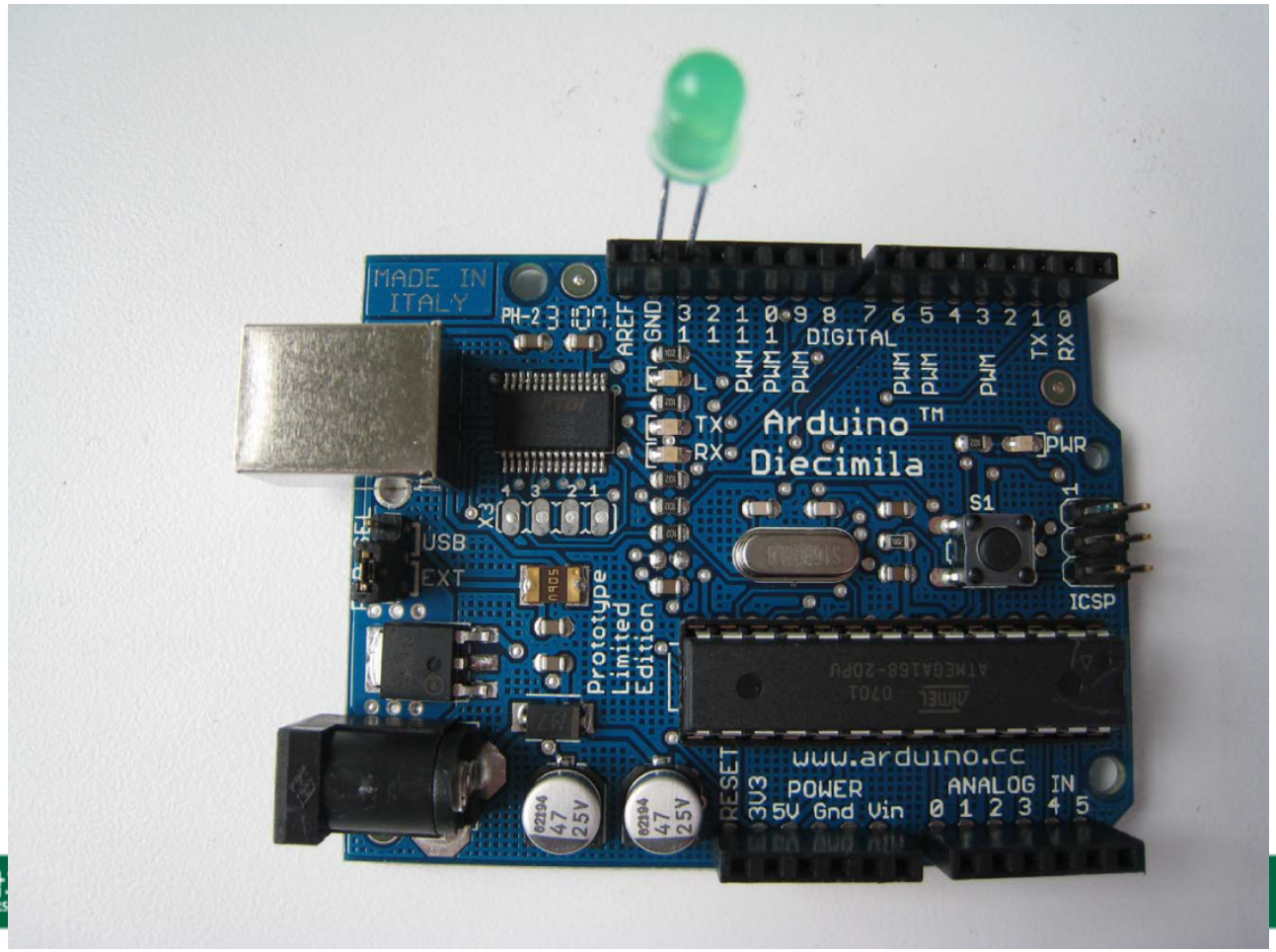
Exercise: Digital Output LED

- Will use pin 13
- On LEDs, polarity matters
  - Shorter lead is "negative" side, goes to ground



Exercise: Digital Output LED

- 1) Connect LED to Arduino Board

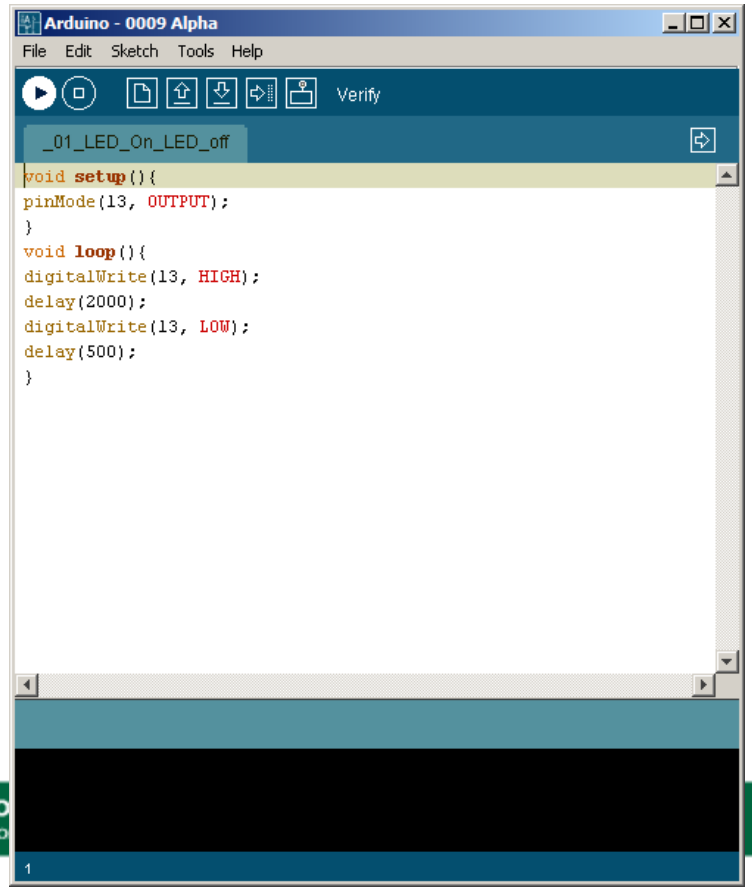


Exercise: Digital Output LED

- 2) LED should be turned ON for 1 second and OFF for 1 second, repeatedly
  - How to fill out **setup()**?
  - How to fill out **loop()**?

Exercise: Digital Output LED

- 2) LED should be turned ON for 1 second and OFF for 1 second, repeatedly




```
void setup(){
   pinMode(13, OUTPUT);
}

void loop(){
   digitalWrite(13, HIGH);
   delay(1000);
   digitalWrite(13, LOW);
   delay(1000);
}
```

Today

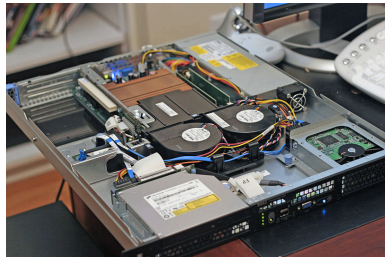
- Review from the last lecture
  - Arduino platform
  - Arduino programming environment

- **Embedded system overview**
- Processor technologies
- Collaborative programming environment
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.

- Announcement

Embedded systems overview

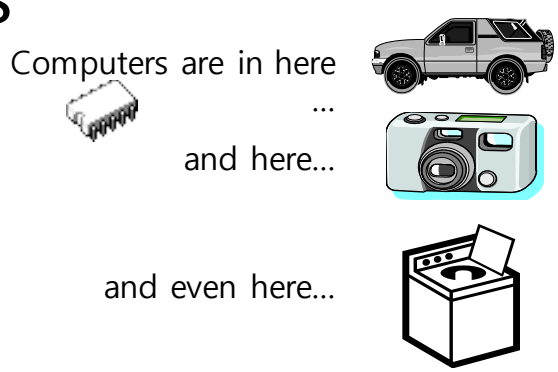
- Computing systems are everywhere
- Most of us think of “desktop” computers
  - PC’s
  - Laptops
  - Mainframes
  - Servers

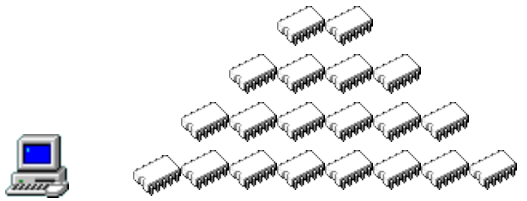


- But there’s another type of computing system
  - Far more common…

Embedded systems overview

- Embedded computing systems
  - Computing systems embedded within electronic devices
  - Hard to define. Nearly any computing system other than a desktop computer
  - Billions of units produced yearly, versus millions of desktop units
  - Perhaps 50 per household and per automobile

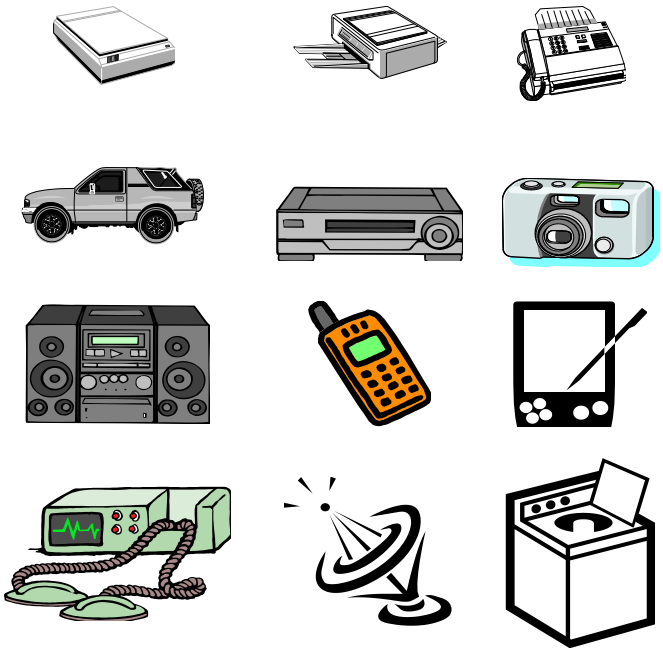




Lots more of these, though they cost a lot less each.

A "short list" of embedded systems

Anti-lock brakes
Auto-focus cameras
Automatic teller machines
Automatic toll systems
Automatic transmission
Avionic systems
Battery chargers
Camcorders
Cell phones
Cell-phone base stations
Cordless phones
Cruise control
Curbside check-in systems
Digital cameras
Disk drives
Electronic card readers
Electronic instruments
Electronic toys/games
Factory control
Fax machines
Fingerprint identifiers
Home security systems
Life-support systems
Medical testing systems
Modems
MPEG decoders
Network cards
Network switches/routers
On-board navigation
Pagers
Photocopiers
Point-of-sale systems
Portable video games
Printers
Satellite phones
Scanners
Smart ovens/dishwashers
Speech recognizers
Stereo systems
Teleconferencing systems
Televisions
Temperature controllers
Theft tracking systems
TV set-top boxes
VCR's, DVD players
Video game consoles
Video phones
Washers and dryers

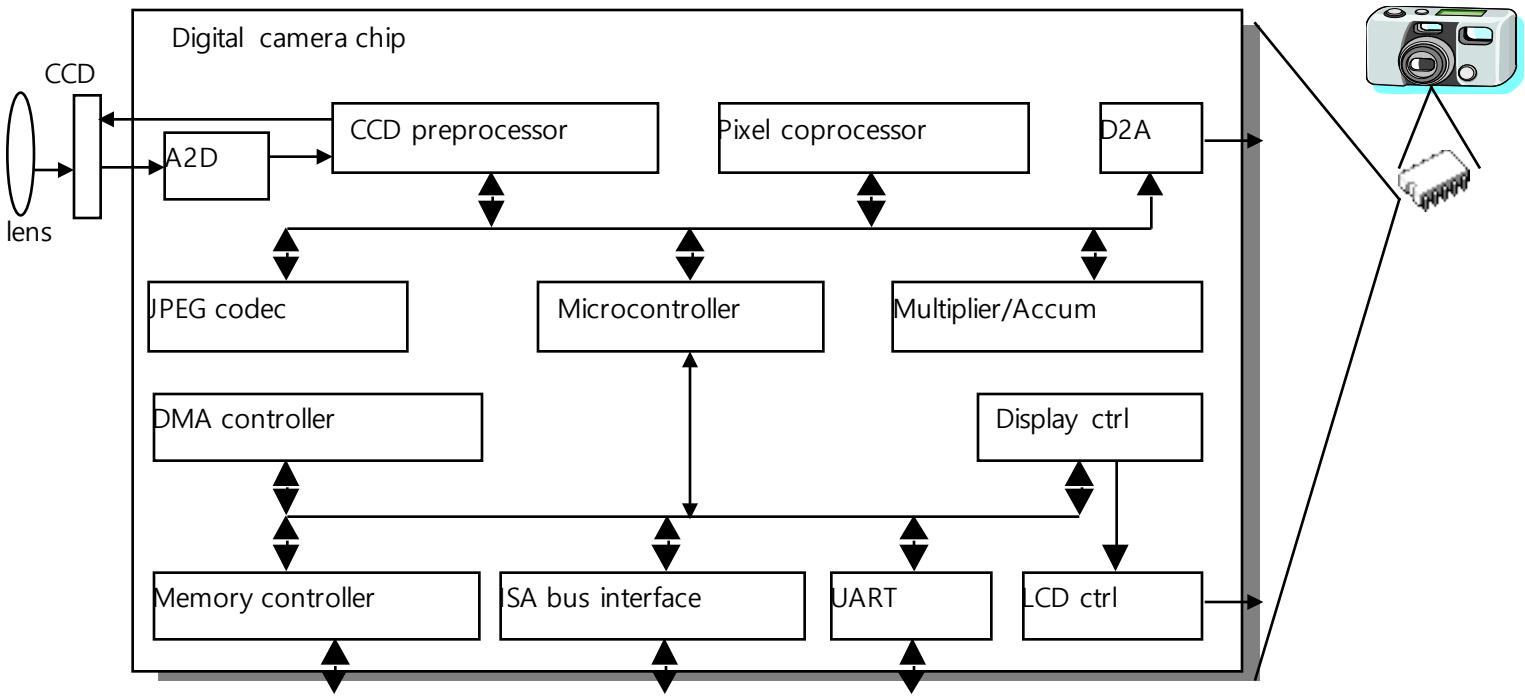


And the list goes on and on

Some common characteristics of embedded systems

- Single-functioned
  - Executes a single program, repeatedly
- Tightly-constrained
  - Low cost, low power, small, fast, etc.
- Reactive and real-time
  - Continually reacts to changes in the system's environment
  - Must compute certain results in real-time without delay

Embedded system example: digital camera



- Single-functioned -- always a digital camera
- Tightly-constrained -- Low cost, low power, small, fast
- Reactive and real-time -- only to a small extent

Today

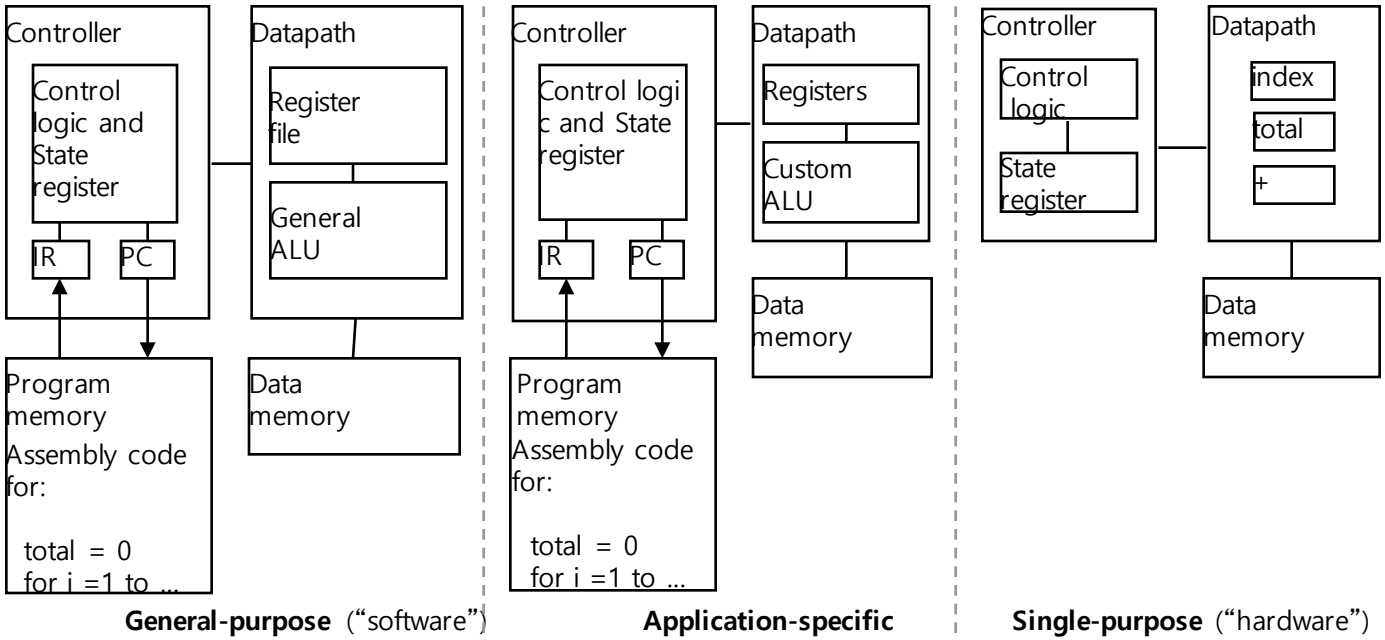
- Review from the last lecture
  - Arduino platform
  - Arduino programming environment

- Embedded system overview
- **Processor technologies**
- Collaborative programming environment
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.

- Announcement

Processor technology

- The architecture of the computation engine used to implement a system's desired functionality
- Processor does not have to be programmable
  - "Processor" *not* equal to general-purpose processor



Processor technology

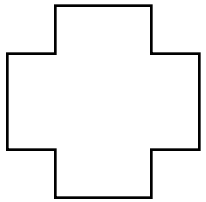
- Processors vary in their customization for the problem at hand

Desired functionality

```
total = 0
for i = 1 to N  loop
   total += M[i]
end loop
```

General-purpose processor

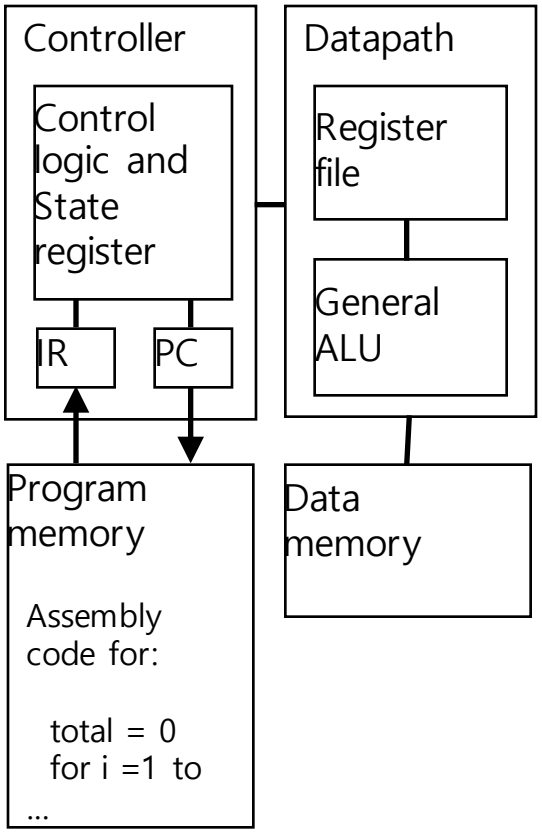
Application-specific processor



Single-purpose processor

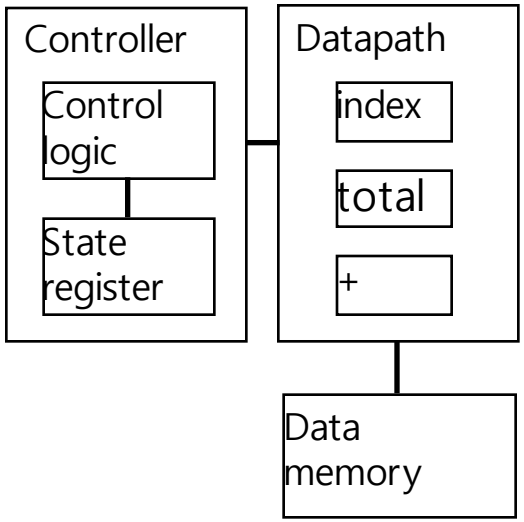
General-purpose processors

- Programmable device used in a variety of applications
  - Also known as "microprocessor"
- Features
  - Program memory
  - General datapath with large register file and general ALU
- User benefits
  - Low time-to-market and NRE costs
  - High flexibility
- "Intel Core i7" the most latest, but there are hundreds of others



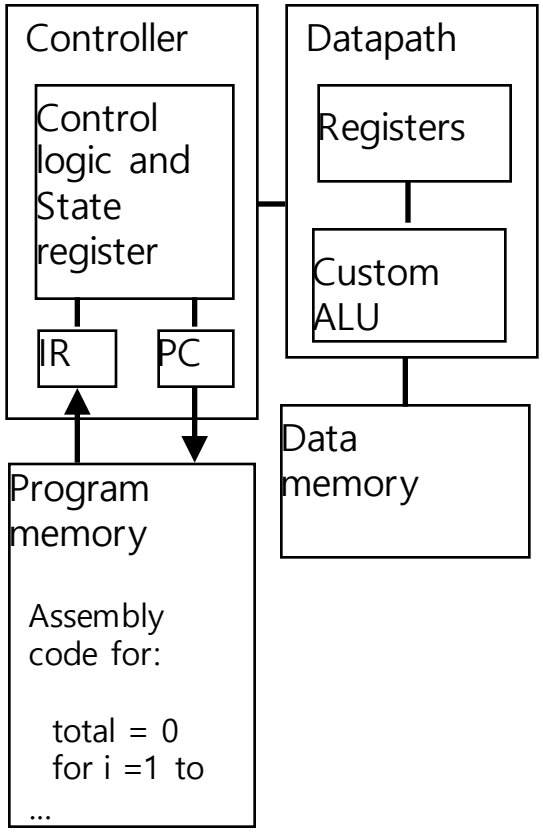
Single-purpose processors

- Digital circuit designed to execute exactly one program
  - a.k.a. coprocessor, accelerator or peripheral
- Features
  - Contains only the components needed to execute a single program
  - No program memory
- Benefits
  - Fast
  - Low power
  - Small size



Application-specific processors

- Programmable processor optimized for a particular class of applications having common characteristics
  - Compromise between general-purpose and single-purpose processors
- Features
  - Program memory
  - Optimized datapath
  - Special functional units
- Benefits
  - Some flexibility, good performance, size and power
- Example
  - Qualcomm Snapdragon AP
  - Nvidia Graphics AP



Today

- Review from the last lecture
  - Arduino platform
  - Arduino programming environment

- Embedded system overview
- Processor technologies
- **Collaborative programming environment**
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.

- Announcement

Methods for tracking versions

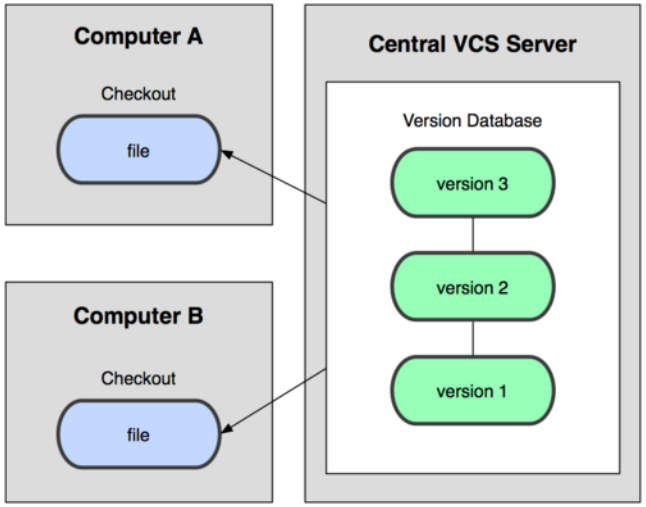
- Don't keep track
  - Good luck!
- Save numbered zip files
  - Unzip versions and diff
- Formal version control
  - Easy to see changes back in time
  - Easy to jump back and test

Why version control?

- For working by yourself:
  - Gives you a "time machine" for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- For getting an internship or job:
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work

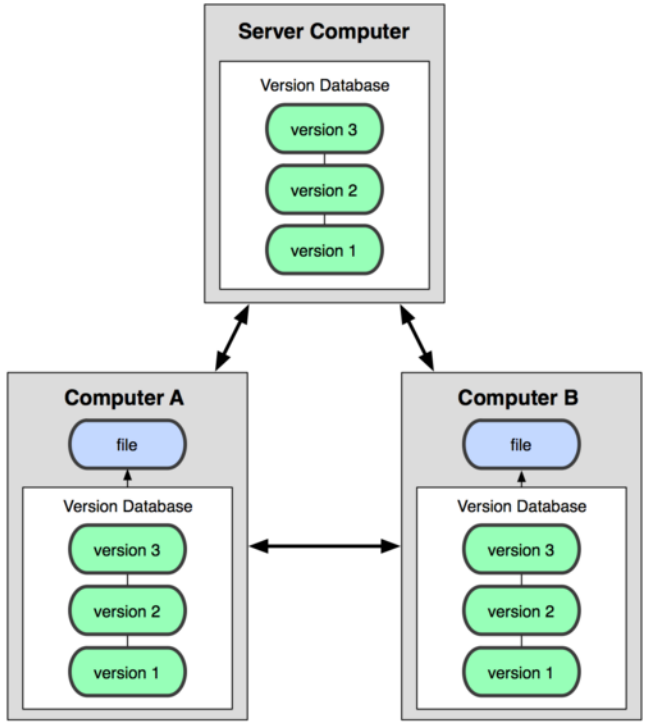
Central vs. Distributed Version Control System (VCS)

<Centralized Model>



Ex: CVS, Subversion(SVN), Perforce

<Distributed Model>



Ex: Git, Mercurial

Result: Many operations are local

What is Git?

- Formal version control system
- Developed by Linus Torvalds (creator of Linux)
  - Used to manage the source code for Linux

- Tracks any content (but mostly plain text files)
  - Source code
  - Data analysis projects
  - Manuscripts
  - Websites
  - Presentations

Why use Git?

- Git is better than CVS, SVN (and most VCSs)
  - It's fast
  - Don't need access to a server
- Distributed
- Branches
- Always in a working state
- Free

What is GitHub?

- Provide Git repositories for free
  - http://github.com
- Graphical user interface for Git
  - Exploring code and its history
  - Tracking issues
- Real open source
  - Immediate, easy access to the code
- Facilitates:
  - Learning from others
  - Seeing what people are up to
  - Contributing to others' code
- Lowers the barrier to collaboration
  - "There is a typo in your xxx.doc" vs.
    "Here is a correction that I modified for your xxx.doc"

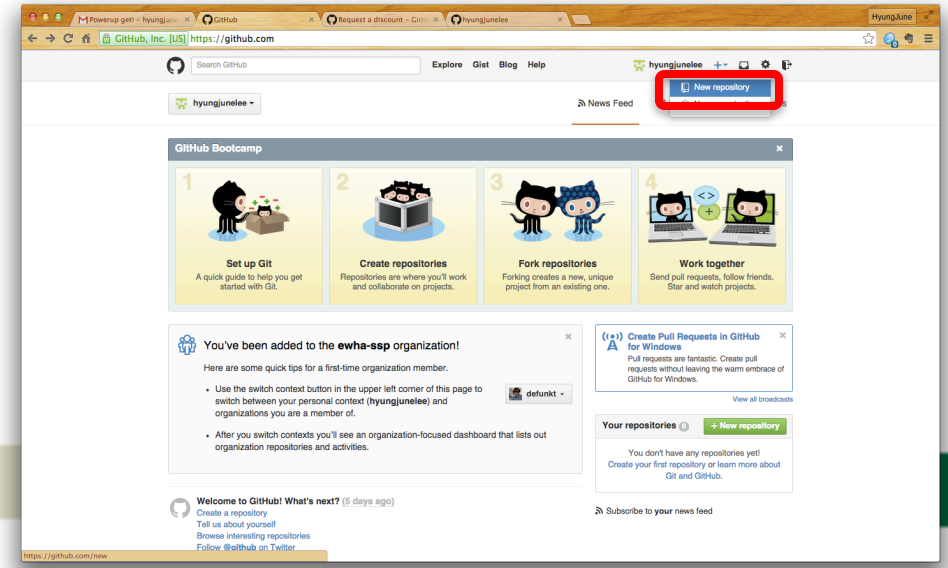
Getting started with GitHub

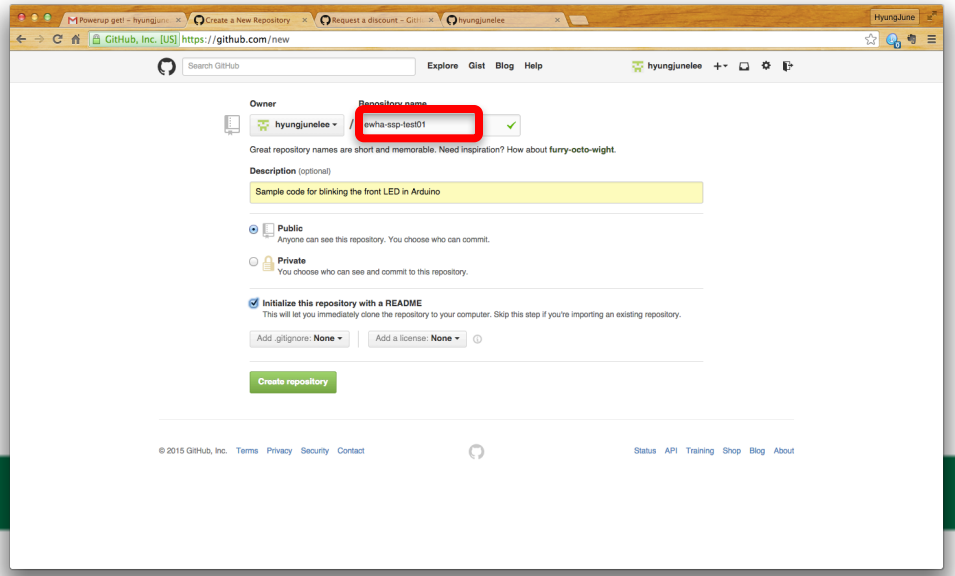
- Step 1: Get an account
  - http://github.com
- Step 2: Create a GitHub repository
  - Click the "Create a new repo" button
  - Give it a name and description
  - Click the "Create repository" button
- Step 3: Work in your computer
  - 1) Command line after installation of Git utility, or
    - Windows: http://msysgit.github.io/ (Git BASH)
    - Mac OS: no installation needed
  - 2) GitHub client for Windows or Mac
    - Windows: https://windows.github.com/
    - Mac OS: https://mac.github.com/

Q1) How to create a repository?

- Case 1: starting from nothing
  - There is no files yet
  - Just want to create a null repository
  - Will add files later on

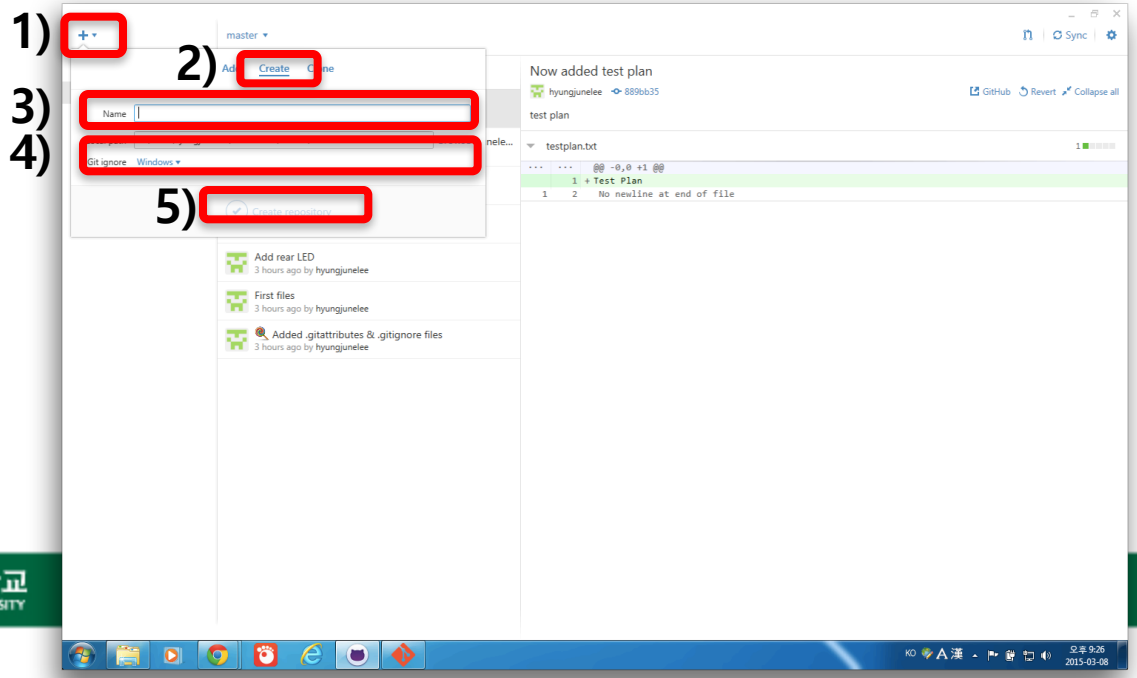
  => Create a repository directly at http://github.com
  - Step: 1) Select "new repository" ➔ 2) enter a repository name ➔3) click "create"





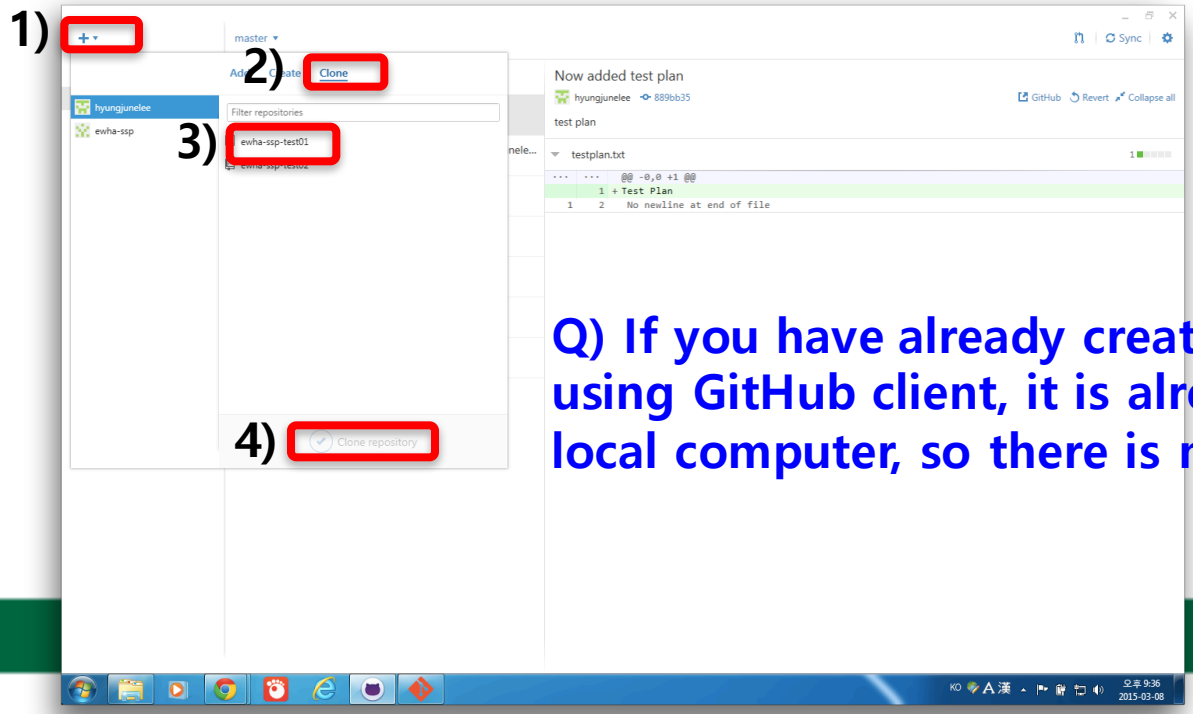
Q1) How to create a repository?

- Case 2: Some files already in your local computer
  - Want to create a repository with initial files
  - Create a repository using GitHub client
  - Step: 1) Click "+" ➔ 2) Click "Create" ➔3) enter a repository name ➔ 4) "Browse" the path for the files ➔5) "Create"



Q2) How to get a repository at your local computer from GitHub

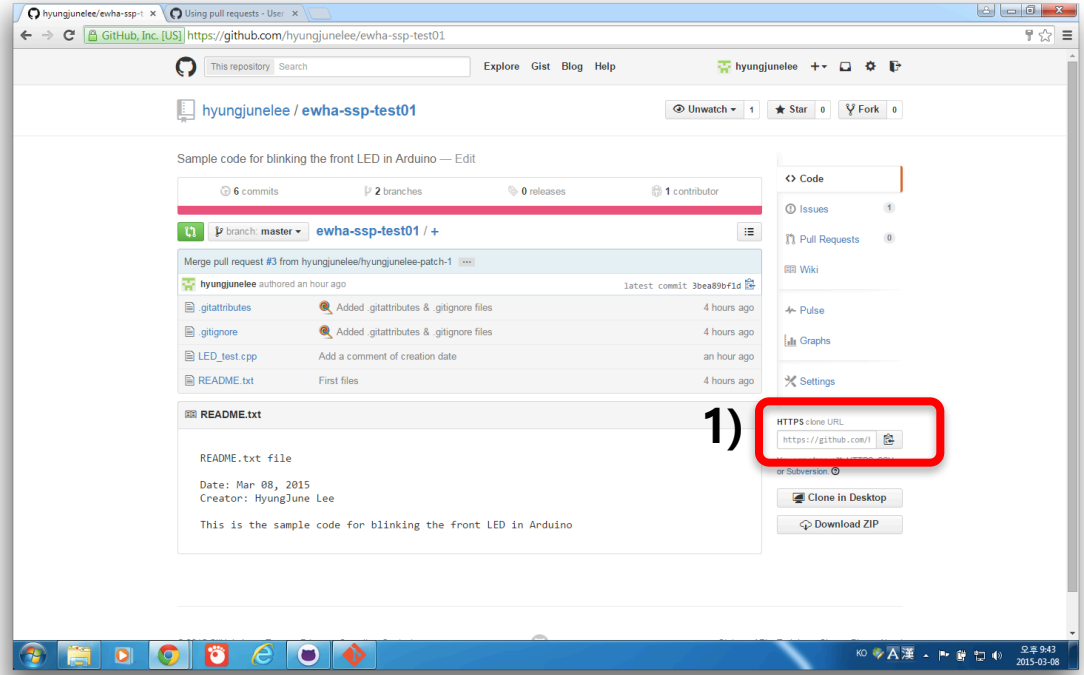
- Case 1: using GitHub client
  - Step: 1) Click "+" ➔ 2) Click "Clone" ➔3) select a repository that is already created at GitHub ➔4) Click "Clone repository" ➔5) See files



Q) If you have already created a repository using GitHub client, it is already in your local computer, so there is nothing to do!

Q2) How to get a repository at your local computer from GitHub

- Case 2: using command line
  - Step: 1) Copy your GitHub URL ➔
    2) At terminal: > git clone [GitHub URL] ➔3) See files



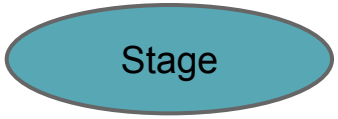
2)

```
git-repo — bash — 93×32
Hyungjunes-MacBook-Air:git-repo hyungjunelee$ git clone https://github.com/hyungjunelee/ewha-ssp-test01.git
```

Terminal Output

```
HyungJunes-MacBook-Air:git-repo hyungjunelee$ git clone https://github.com/hyungjunelee/ewha-ssp-test01.git
Cloning into 'ewha-ssp-test01'...
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 11 (delta 3), reused 11 (delta 3), pack-reused 0
Unpacking objects: 100% (11/11), done.
HyungJunes-MacBook-Air:git-repo hyungjunelee$ ls
ewha-ssp-test01
HyungJunes-MacBook-Air:git-repo hyungjunelee$ cd ewha-ssp-test01/
HyungJunes-MacBook-Air:ewha-ssp-test01 hyungjunelee$ ls
LED_test.cpp        README.txt
```

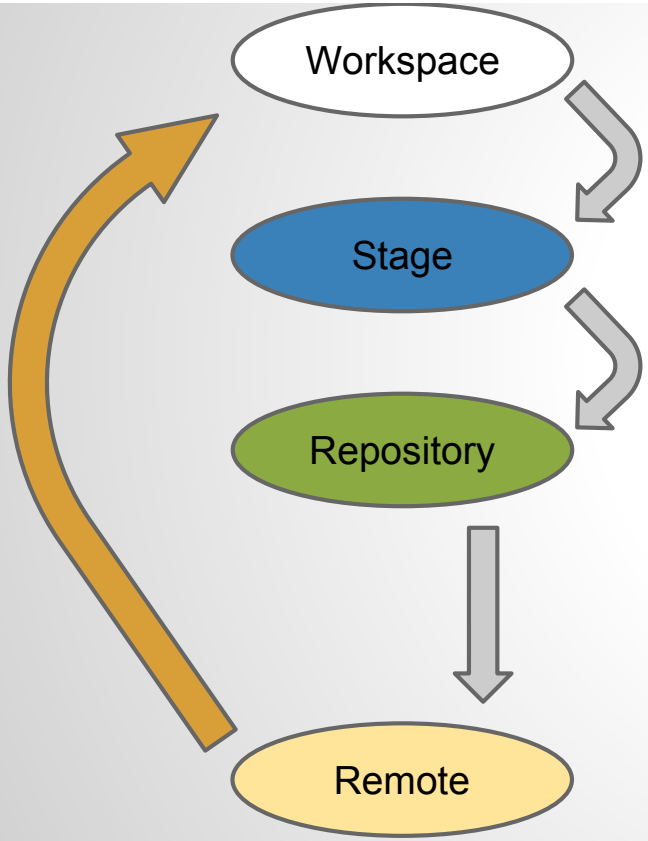
Q3) How to maintain?



Case 1: command line using Git

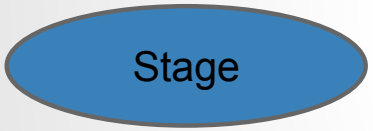
- 1) Pull changes from your collaborator
  - Before you start working
    > **git pull**
- 2) Change some files (in your "workspace")
- 3) See what you have changed
    > **git status**
    > **git diff**
    > **git log**
- 4) Indicate what changes to save (in your "stage")
- 5) Commit those changes into your local "repository"
- 6) Push the changes to your remote original "GitHub" repository
    > **git push**

git pull



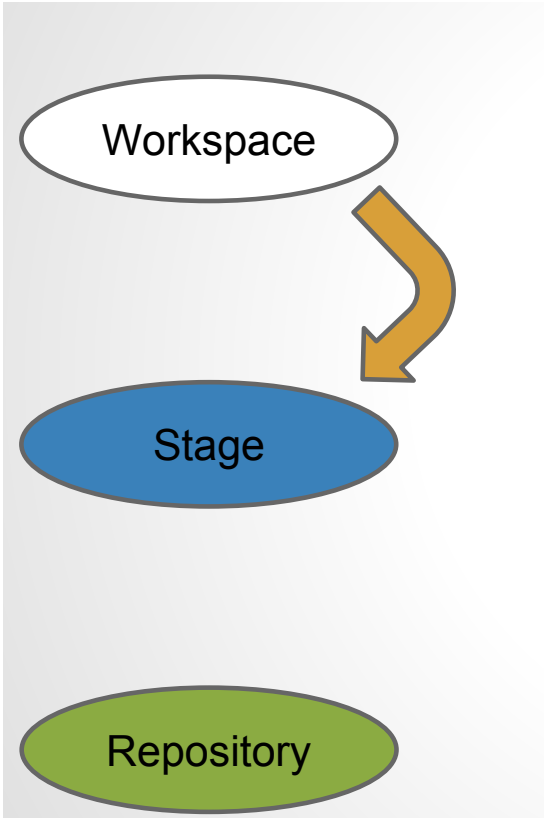

```
[~/Test]$ git pull
remote: Counting objects: 3, done
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From github.com:user/Test
   c3b8ef6..ea5dda8  master         -> origin/master
Updating c3b8ef6..ea5dda8
Fast-forward
 hello.js | 6 ++++--
 1 file changed, 4 insertions(+), 1 deletion(-)
[~/Test]$
```

git status




```
[~/Test]$ echo “console.log(‘hello world’);” > hello.js


[~/Test]$ git status
On branch master

Initial commit

Untracked files:
   (use “git add <file>...” to include in what will be committed)

        hello.js

nothing added to commit but untracked files present
[~/Test]$
```

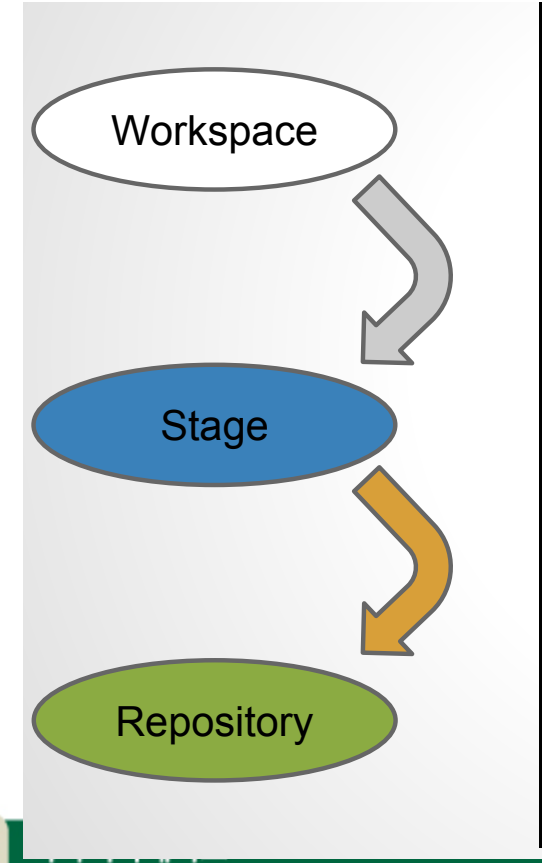
git add




```
[~/Test]$ git add .


[~/Test]$ git status
On branch master

Initial commit

Changes to be committed:
   (use “git rm --cached <file>...” to unstage)

        new file:    hello.js
[~/Test]$
```

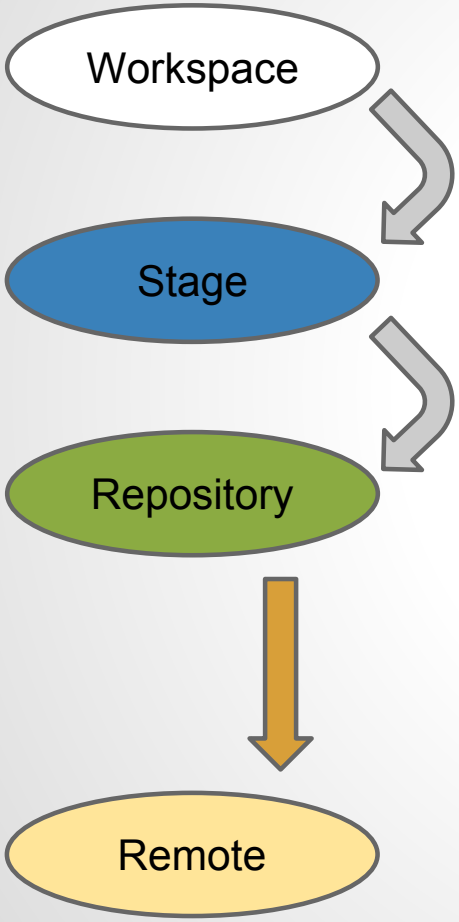
git commit

- Use –m “comment on what’s going on”




```
[~/Test]$ git commit -m “Started hello world program”
[master (root-commit) c3b8ef6] Started hello world program
 1 file changed, 1 insertion(+)
 create mode 100644 hello.js


[~/Test]$ git status
On branch master
nothing to commit, working directory clean
[~/Test]$
```

git push

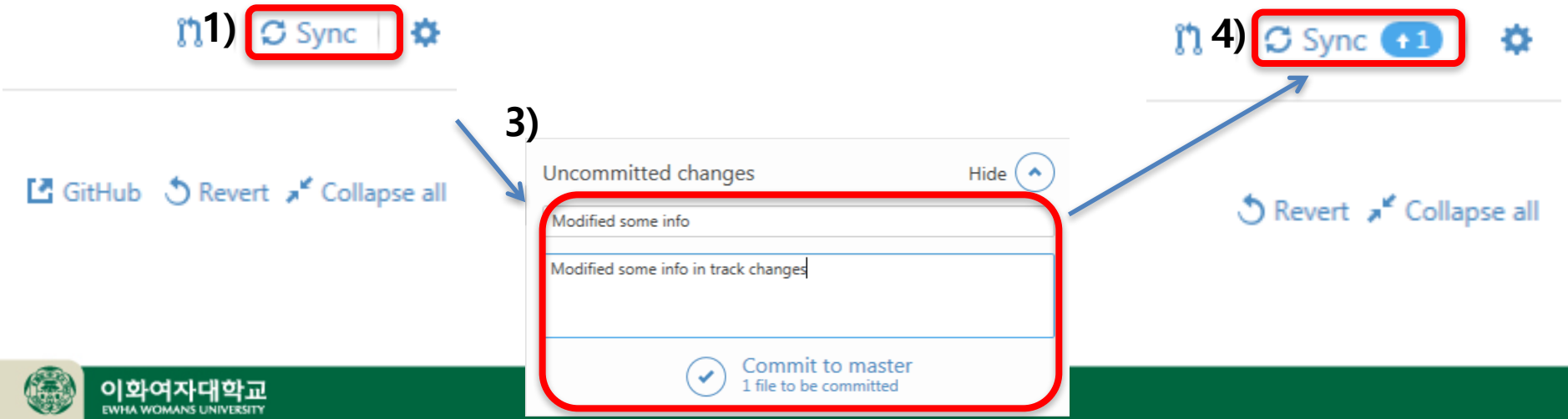



```
[~/Test]$ git push
Counting objects: 1, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (1/1), done.
Writing objects: 100% (1/1), 28 bytes | 0 bytes/s, done.
Total 1 (delta 1), reused 0 (delta 0)
To git@github.com:user/Test.git
   (root-commit)..c3b8ef6    master -> master
[~/Test]$
```

Q3) How to maintain?

Case 2: using GitHub client

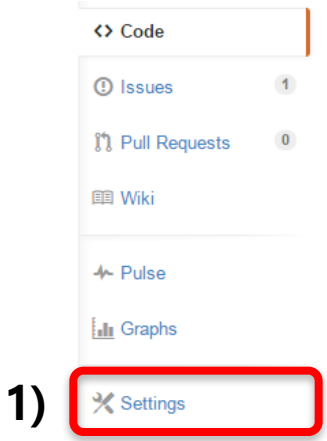
- 1) Click sync to pull (corresponding to "git pull")
  - Before you start working
- 2) Change some files (in your local computer)
- 3) Commit those changes (in your local computer)
- 4) Click sync to push (corresponding to "git push")

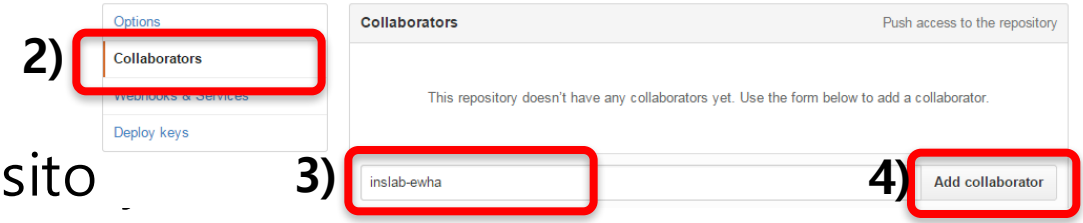


Q4) How to collaborate with other people?

Case 1: Administrator

- 1) Click "Settings" in the right panel
- 2) Click "Collaborators"
- 3) Enter a username to collaborate
- 4) Click "Add collaborator"
  - This means you give the "push" privilege to this person



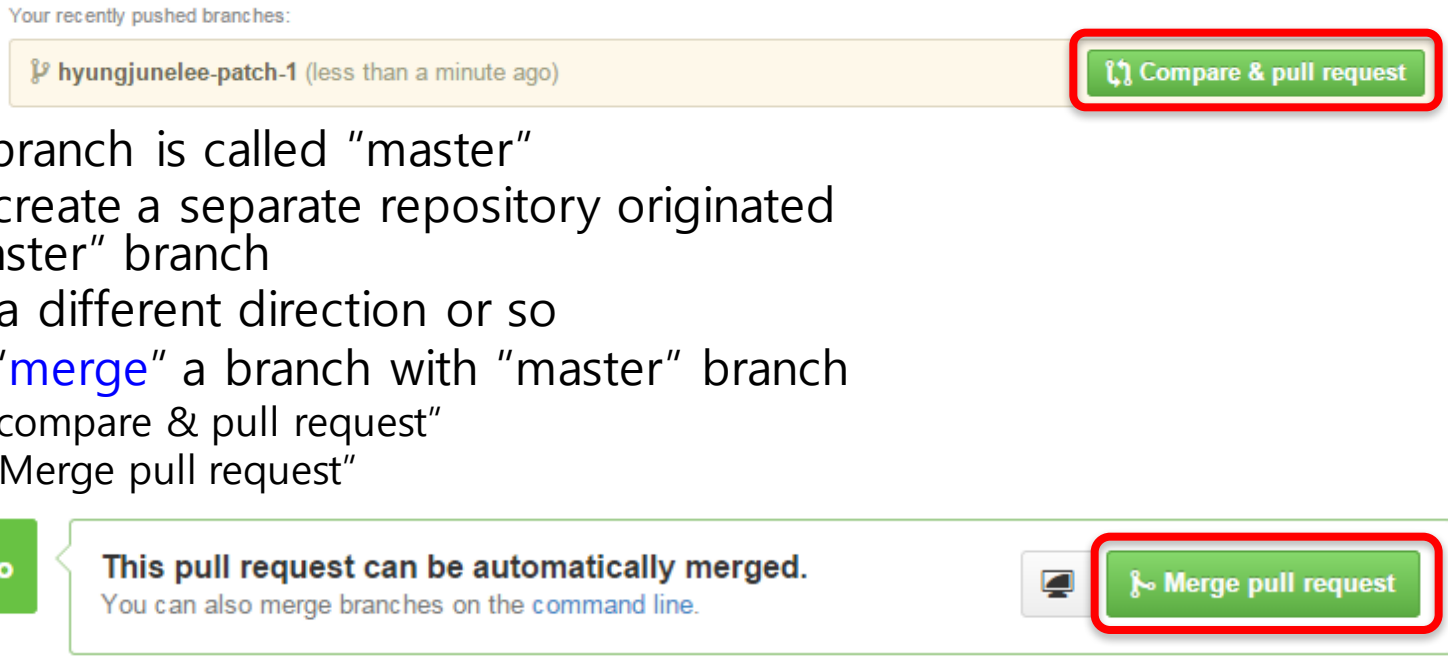


Case 2: Collaborator

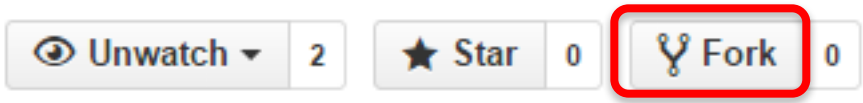
- 1) Visit the original reposito
- 2) Copy the URL
- 3) Clone it in your local computer
- 4) Work on the files
- 5) Push to the original repository

Q5) How to create a separate repository for your own purpose?

- Branch vs. Fork
- Branch
  - Original branch is called "master"
  - You can create a separate repository originated from "master" branch
  - Develop a different direction or so
  - You can "merge" a branch with "master" branch
    - Click "compare & pull request"
    - Click "Merge pull request"

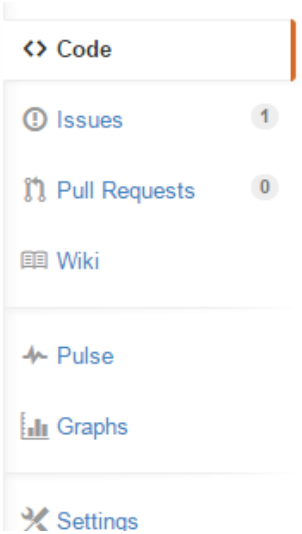


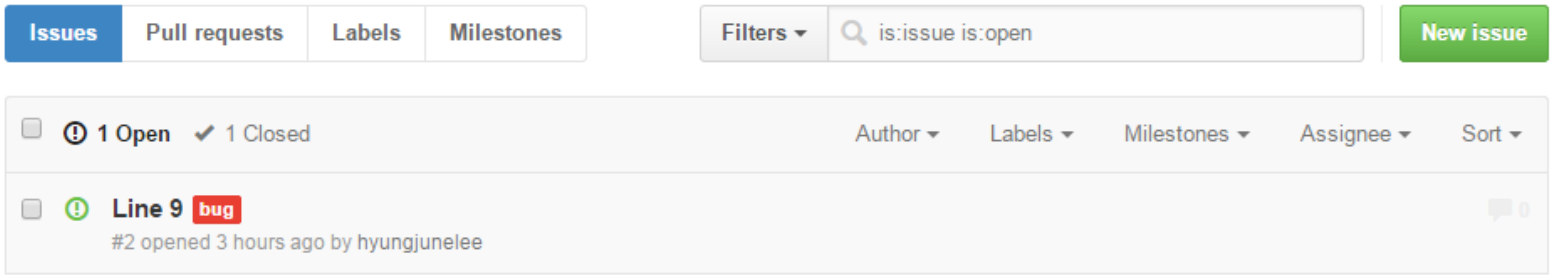
- Fork
  - You are not a collaborator, but
  - You want a personal copy of someone else's project
  - Visit a repository
  - Click "fork"



Q6) Leave some high-level comments?

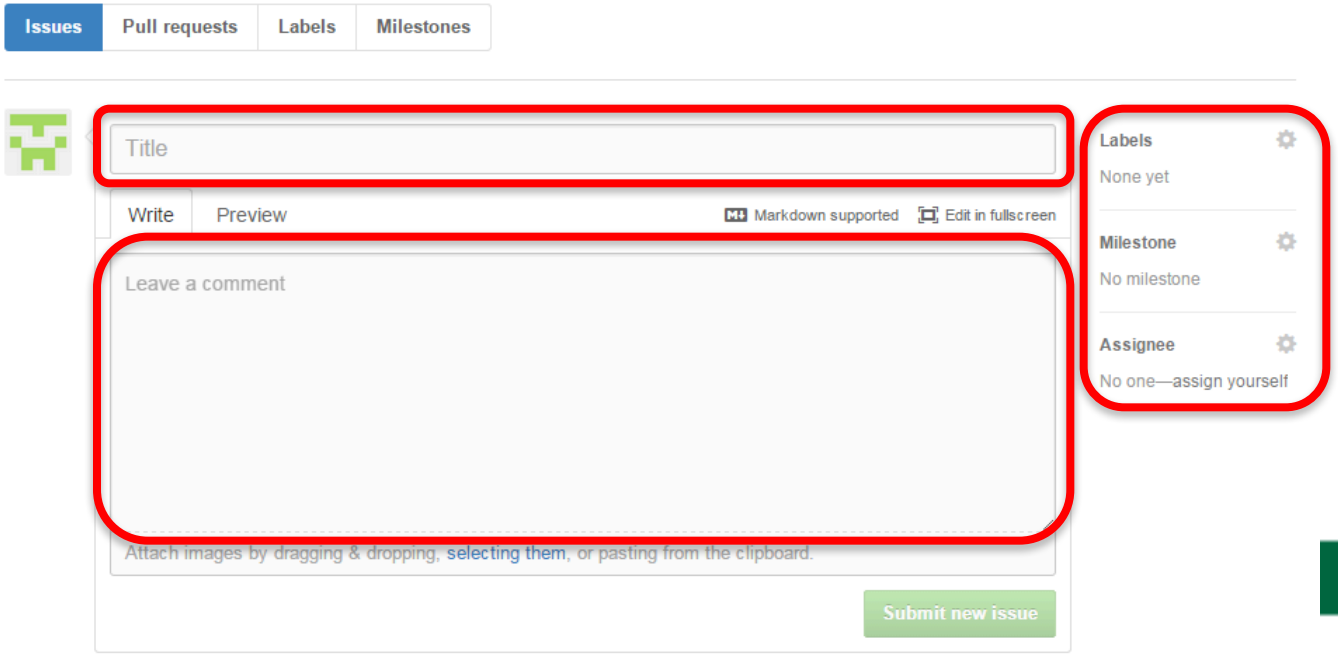
- Create an "issue"
  - Click "Issues"
  - Click "New Issue"





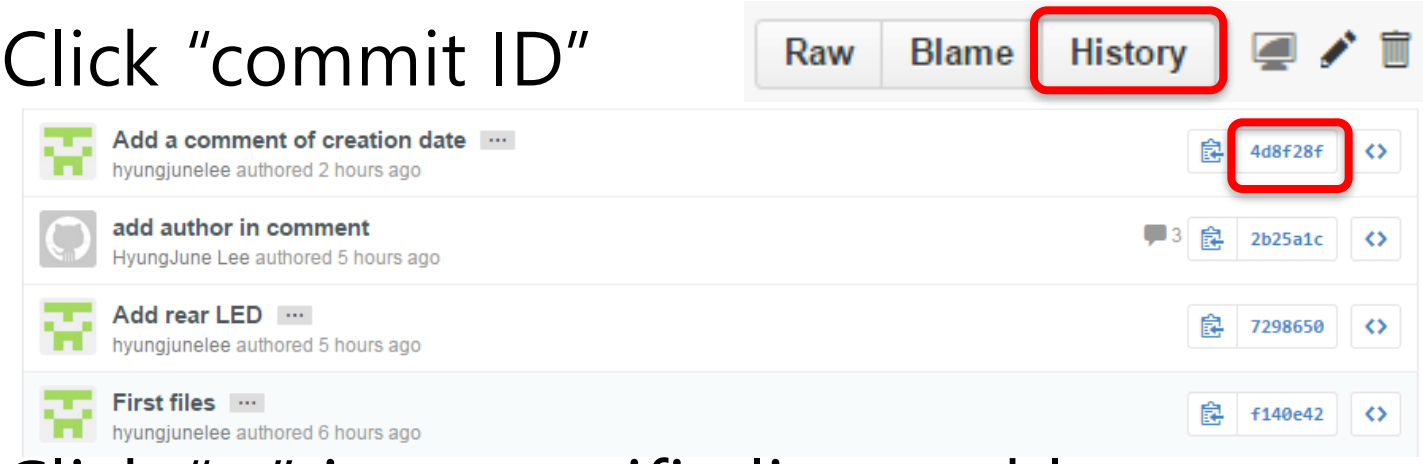
Q6) Leave some high-level comments?

- Leave comments
- Select an appropriate Label
  - Ex) bug, question, etc.
- Select Milestone
- Select Assignee: who should resolve this issue?



Q7) Leave some line-by-line comments ?

- Select a file
- Select “History”
- Click “commit ID”



- Click “+” in a specific line and leave a comment

```
2   #include "LED_test.h"
3
4   #define FRONT_LED_PIN  10
5  +#define REAR_LED_PIN   9       //Add a new code
6
7   //The setup function is called once at startup of the sketch
8   void setup()
9   {
10          pinMode(FRONT_LED_PIN, OUTPUT);
11  +       pinMode(REAR_LED_PIN, OUTPUT); //Add a new code
12  +
```

Q8) How to move to a commit or a branch using command line Git

- To a previous commit
  - > **git checkout commit-ID**
- To a certain branch
  - > **git checkout branch-name**
- To the latest revision
  - > **git checkout master**

Q9)  git vs. GitHub 

Question: Do I have to use Github to use Git?

Answer: No!

- You can use Git completely locally for your own purposes, or

- You or someone else could set up a server to share files, or

More Resources

- For a quick interactive Git tutorial
  - http://try.github.io/
- Git Cheat Sheet
  - https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf
- Many Git & GitHub Learning Materials
  - https://help.github.com/articles/good-resources-for-learning-git-and-github

Course Announcement

- I will post a thread of asking your team info
  - If you have found your partner, please write down names in comment
    Ex) John Doe & Jane Doe
  - Otherwise, please leave your comment like
    "Help for finding my partner!" or something

- For lab session, we will cover
  - Arduino Mega 2560 board
  - Programming environment setup
  - Simple example running in SmartCAR

- Next week, we will study on
  - ATmega2560 microcontroller (MCU) architecture
  - CPU, Memory, I/O ports