Recommended CS120B Workflow

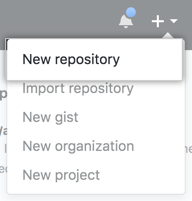
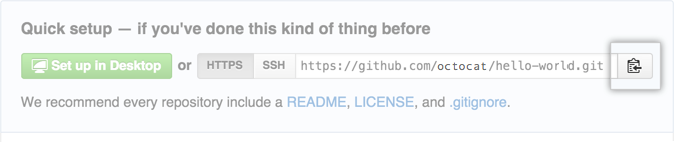
[**Initialize the project/link to GitHub**](#_pbftx5zi5nqj) **2**

[**Create a test file**](#_74d7i8gb5fde) **3**

[**Write your solution code**](#_ghozr68q5ryw) **5**

[**Submission**](#_6d6jbu3i4u1w) **6**

# Initialize the project/link to GitHub

1. Create a project for the lab (createProject.sh script location is system dependent)  
   $ path/to/script/createProject.sh  
   Project name: Lab2\_introToAVR  
   Partners name [none]:  
   Microcontroller [atmega1284]:  
   Clock Frequency [8000000]:  
   Note: the values in the square brackets [ ] are defaults.
2. Go into the directory and initialize it as a github repo ([walkthrough](https://help.github.com/en/articles/adding-an-existing-project-to-github-using-the-command-line))  
   $ cd Lab2\_introToAVR  
   $ git init  
   $ git add .  
   $ git commit -m “Initializing repository with lab files”
3. Create a new repository on Github  
   
4. At the top of your GitHub repository’s Quick Setup page, copy the remote repository’s URL  
     
   $ git remote add origin <*remote repository URL*>  
   $ git remote -v  
   origin [git@github.com](mailto:git@github.com):<user>/<reponame>.git (fetch)  
   origin [git@github.com](mailto:git@github.com):<user>/<reponame>.git (push)  
   $ git push -u origin master

# Create a test file

1. Open up the test/tests.gdb file for editing  
   $ vim test/tests.gdb  
   Note: Replace vim with your favorite editor
2. Add the test cases for the first exercise for the lab. For Lab 2 Ex 1:  
   test “PINA[1:0]: 00, PORTB0: 0”  
   setPINA 0x00  
   continue 2  
   expectPORTB 0x00  
   checkResult  
     
   test “PINA[1:0]: 01, PORTB0: 1”  
   setPINA 0x01  
   continue 2  
   expectPORTB 0x01  
   checkResult  
     
   test “PINA[1:0]: 10, PORTB0: 0”  
   setPINA 0x02  
   continue 2  
   expectPORTB 0x00  
   checkResult  
     
   test “PINA[1:0]: 11, PORTB0: 0”  
   setPINA 0x03  
   continue 2  
   expectPORTB 0x00  
   checkResult  
   Note: Lab 2 Ex 1 is easy because a truth table of all the inputs and expected outputs is provided, although there is a missing test case or two still, see if you can figure them out?
3. Add any variables you want displayed at every break point under the break command  
   commands  
   silent  
   # Add all variables you want to inspect  
   end
4. Run the tests (Note: You will need to add at least one line of code to your while(1) before you can run any tests)  
   $ make test  
   simavr -v -v -v -v -g -mmcu=atmega1284 -f 8000000 build/objects/main.elf &  
   avr-gdb -batch -x test/commands.gdb -x test/tests.gdb  
   Loaded 232 .text at address 0x0  
   Loaded 14 .data  
   Creating VCD trace file 'build/results/Lab2\_introToAVR\_trace.vcd'  
   avr\_gdb\_init listening on port 1234  
   gdb\_network\_handler connection opened  
   0x00000000 in \_\_vectors ()  
   Breakpoint 1 at 0xe0: file source/main.c, line 20.  
   Note: automatically using hardware breakpoints for read-only addresses.  
   ======================================================  
   Running all tests..."  
     
   Test 1:"PINA[1:0]: 00, PORTB0: 0"...  
   Passed.  
   Test 2:"PINA[1:0]: 01, PORTB0: 1"...  
    Expected 0x01 on PORTB  
   Failed.  
   Test 3:"PINA[1:0]: 10, PORTB0: 0"...  
   Passed.  
   Test 4:"PINA[1:0]: 11, PORTB0: 0"...  
   Passed.  
   Passed 3/4 tests.  
   ======================================================  
   Remote doesn't know how to detach  
   gdb\_network\_handler connection closed  
   signal caught, simavr terminating
5. Notice you failed one test, the one that matters. Now go write the code to fix it, but first, commit your test cases  
   $ git add test/tests.gdb  
   $ git add source/main.c  
   $ git commit -m “Add tests”

# Write your solution code

1. Open up the source/main.c file for editing  
   $ vim source/main.c  
   Note: Replace vim with your favorite editor
2. Initialize your DDRx and PORTx  
   DDRA = 0x00; PORTA = 0xFF;  
   DDRB = 0xFF; PORTB = 0x00;
3. Add any addiitonal temporary variables you may need
4. Add your solution code to the while(1)
5. Add all ports/pins of interest to the trace list in the header/simAVRHeader.h  
   const struct avr\_mmcu\_vcd\_trace\_t \_mytrace[] \_MMCU\_ = {  
    {AVR\_MCU\_VCD\_SYMBOL("PINA0"),.mask=1<<0,.what=(void\*)&PINA,},  
    {AVR\_MCU\_VCD\_SYMBOL("PINA1"),.mask=1<<1,.what=(void\*)&PINA,},  
    {AVR\_MCU\_VCD\_SYMBOL("PORTB"),.mask=1<<0,.what=(void\*)&PORTB},  
   };
6. Run the tests  
   $ make test  
   ======================================================  
   Running all tests..."  
     
   Test 1:"PINA[1:0]: 00, PORTB0: 0"...  
   passed.  
   Test 2:"PINA[1:0]: 01, PORTB0: 1"...  
   passed.  
   Test 3:"PINA[1:0]: 10, PORTB0: 0"...  
   passed.  
   Test 4:"PINA[1:0]: 11, PORTB0: 0"...  
   passed.  
   Passed 4/4 tests.  
   ======================================================
7. If your solution was correct, you should now pass all tests, commit and push  
   $ git commit -m “Completed exercise 1”  
   $ git push
8. Add copies of your main.c and tests.gdb to the turnin folder.  
   $ cp source/main.c turnin/[cslogin]\_lab[#]\_part[#].c  
   $ cp test/tests.gdb turnin/[cslogin]\_lab[#]\_part[#]\_tests.gdb
9. Continue with the rest of the lab.

# Submission

1. Create an archive of your turnin folder  
   $ tar -czvf [cslogin]\_lab[#].tgz turnin/
2. Submit on ilearn.