

Lab 1: Setting up a productive workflow

Starter Files

Download [lab01.zip](#). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the [OK](#) autograder.

Submission

By the end of this lab, you should have submitted the lab with `python3 ok --submit`. You may submit more than once before the deadline; only the final submission will be graded.

Table of Contents

- [Introduction](#)
- [Getting Started](#)
 - [Opening a terminal](#)
 - [Logging into your class account](#)
 - [Registering your account](#)
 - [Changing your password](#)
 - [Logging out](#)
- [Installing Python](#)
- [Organizing your files](#)
 - [Directories](#)
 - [Making New Directories](#)
 - [Moving to other directories](#)
 - [Downloading the assignment](#)
 - [Moving files](#)
- [Installing a text editor](#)
- [A first look at lab01.py](#)
- [Running doctests](#)

Introduction

[Lab 0](#) walked you through how to use the school computers for CS 61A. This lab explains how to use your own computer to complete assignments!

Getting Started

Opening a terminal

The terminal is a program that allows you to interact with your computer by entering commands. No matter what operating system you use (Windows, MacOS, Linux), the terminal will be an essential tool for CS 61A.

If you're on a Mac or are using a form of Linux (such as Ubuntu), you already have a program called `Terminal` on your computer. Open that up and you should be good to go.

For Windows users, we recommend downloading a terminal called [GitBash](#).

Logging into your class account

At the start of your first lab section, you should have received a class account form. At the top left of the form should be your username and your temporary password:

Login: cs61a-aaa
Password: ...

Most of the work in this class can be done without logging into your account. However, you will periodically login to your class account to check your grades.

Let's login in now. Open up your terminal and type in the following command:

```
ssh cs61a-??@cory.eecs.berkeley.edu
```

where `??` is replaced with your two (or three) letter login.

If you're interested, here's an explanation of what the command does:

1. `ssh` is a secure shell (i.e. terminal) that connects to remote

servers

2. cs61a-?? is the username on the remote server
3. cory.eecs.berkeley.edu is the name of the remote server. Again, here is a list of [servers](#) that belong to Berkeley's CS department

You can also watch this [video](#) for help.

Once you press enter, the following message will appear:

```
The authenticity of host 'cory.eecs.berkeley.edu' can't be established.  
RSA key fingerprint is ...  
Are you sure you want to continue connecting (yes/no)?
```

Type in yes and press enter. Next, you will be prompted for your password. If you haven't changed your password yet, use the temporary password on your class account form.

When you type your password, nothing will show up! This is a security feature, not a bug. Continue typing and press enter to login.

Once you are logged in, you'll see something like this:

```
hw0 — ssh — 79x37
Last login: Tue Aug 26 14:58:45 2014 from airbears2-10-14
Sun Microsystems Inc. SunOS 5.10 Generic January 2005

-----
                W E L C O M E T O N O V A
        brought to you by Instructional Support Group
-----

- Nova is an 8 core (1.4 Ghz) UltraSparc T2 running Solaris 10.
- Intel binaries will not run here.
- See the green bulletin boards by 199 Cory, 271/330 Soda for general info.
- For more info on labs please see http://inst.eecs.berkeley.edu
- To find scheduled office hours for staff please finger inst@inst
- Send questions/problems to inst@EECS.Berkeley.EDU

- Any processes left running for over 24 hours will be killed.

- Ongoing COMPUTER HELP SESSIONS: see http://www.CSUA.Berkeley.EDU
-----

Date                Notices
-----

5/1      Nova will be decomissioned and retired from service on June 12th 2014.
        Please note that we will gradually be phasing out all of our solaris
        systems between June and August 2014. Thank You

(type "more /etc/motd" to repeat this message)

'cs61a' is using 17680/26214 MB (67%) of its disk quota on /home/ff.
(Type 'more /share/b/pub/disk.quotas' for more information.)

nova [501] ~ #
```

Registering your account

The first time you login to your class account, your terminal will ask you some registration questions about the following:

- Last name
- First name
- Student ID
- Email (please use your berkeley.edu email!)
- Code name (we don't use this information, you can enter anything you want)

If your terminal doesn't prompt you for this information the first time you login, you can type `register` to begin the process. You don't need to do this again if you've already registered before.

If you find errors (e.g. you typed your last name as "ssh update"), fix them immediately by running the command:

re-register

Changing your password

The temporary password is not the easiest thing to remember. While still logged in, you should change your password by typing in

ssh update

You will be prompted for the temporary password again. After that, following the instructions for changing your password.

Logging out

Once you've registered your account and changed your password, you can log out by pressing `Ctrl-D`.

Installing Python

You can get Python [here](#). Download one of the installers (for example, "Windows x86-64 MSI installer" or "Mac OS X 64-bit installer"). If your computer is a 64-bit machine, you should download the 64-bit installer. (In general, if your computer is new within the past 2 years, it is likely 64-bit. Ask a TA or lab assistant if you're not sure.)

MacOS users can refer to this [video](#) for additional help on setting up Python.

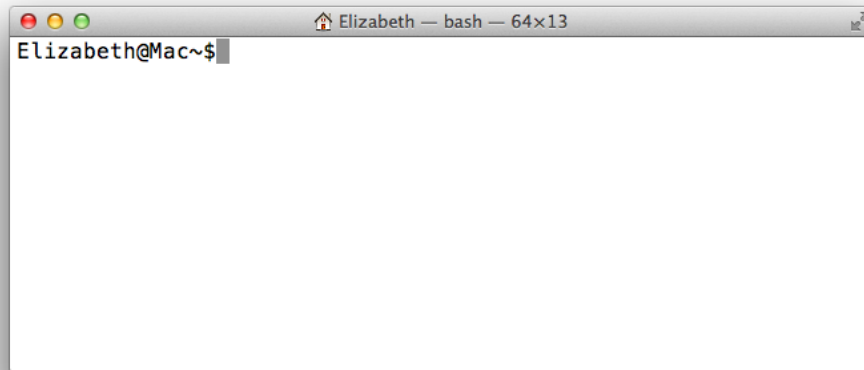
If you're having trouble opening the installer, you can right-click the icon and select "Open".

Windows users can refer to this [video](#) for additional help on setting up Python (up to 1:09 into the video).

You will also need to configure your `PATH` environment variable; the same [video](#) describes how to do this from 5:00 to 5:54.

Organizing your files

First, open up a terminal, if you haven't already. Since we're learning how to work on your personal computer, make sure you are not logged in to your class account.



Right now, I'm in my home directory. The home directory is represented by the `~` symbol. When you first open your terminal, you will start in the home directory.

Don't worry if your terminal window doesn't look exactly the same; the important part is that the text on the left hand side is relatively the same (with a different name) and you should definitely see a `~` (tilde).

Directories

Throughout this lab, we will be using terminal commands (like in [lab 0](#)) to manage our files. The first command we'll use is `ls` (the letter `l` and the letter `s`). Try typing it in the terminal:

```
ls
```

The `ls` command lists all the files and folders in the current directory. A directory is another name for a folder (such as the `Documents` folder). Since we're in the home directory right now, you should see the contents of your home directory.

Making New Directories

Our next command is called `mkdir`, which makes new directories. Let's make a directory

called `cs61a` to store all of the assignments for this class:

```
mkdir cs61a
```

A folder called `cs61a` will appear in our home directory.

Moving to other directories

To move into another directory, we use the `cd` command. Try typing the following command into your terminal:

```
cd cs61a
```

The `cd` command will change directories — in other words, it moves you into the specified folder. In the example above, we chose to move into the `cs61a` directory.

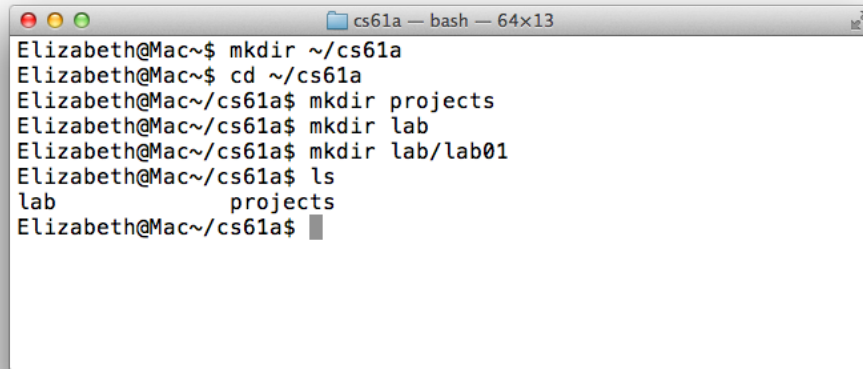
If we want to go back to our home directory, there are a few ways to do so:

- Type `cd ..` (two dots). The `..` means "the parent directory". In this case, the parent directory of `lab0` happens to be our home directory, so we can use `cd ..` to go up one directory.
- Type `cd ~` (the tilde). Remember that `~` means home directory, so this command tells your terminal to change to the home directory, no matter where you currently are.
- Type `cd` (that is, the `cd` command with no arguments). In UNIX, typing just `cd` is a shortcut for typing `cd ~`.

At this point, let's create some more directories. Make sure you are in the `~/cs61a` directory, using the necessary `cd` commands. Then create `projects`, `lab`, and `lab01` folders inside of our `lab` folder:

```
cd ~/cs61a
mkdir projects
mkdir lab
mkdir lab/lab01
```

Now if we list the contents of the directory (using `ls`), we'll see two folders, `projects` and `lab`.

A terminal window titled 'cs61a — bash — 64x13' showing a series of commands and their outputs. The user 'Elizabeth' is at a Mac. The commands are: 'mkdir ~/cs61a', 'cd ~/cs61a', 'mkdir projects', 'mkdir lab', 'mkdir lab/lab01', and 'ls'. The 'ls' command shows 'lab' and 'projects' as subdirectories.

```
Elizabeth@Mac~$ mkdir ~/cs61a
Elizabeth@Mac~$ cd ~/cs61a
Elizabeth@Mac~/cs61a$ mkdir projects
Elizabeth@Mac~/cs61a$ mkdir lab
Elizabeth@Mac~/cs61a$ mkdir lab/lab01
Elizabeth@Mac~/cs61a$ ls
lab      projects
Elizabeth@Mac~/cs61a$
```

Downloading the assignment

Next, download the template file for this lab, [lab01.py](#). Once you've done that, let's find our downloaded file. On most computers, `lab01.py` is probably located in a directory called `Downloads` in your home directory. Let's use the `ls` command to check:

```
ls ~/Downloads
```

If you don't see `lab01.py`, ask a TA or lab assistant for help.

Moving files

Let's move our starter file into our new lab directory. From the terminal use the following command:

```
mv ~/Downloads/lab01.py ~/cs61a/lab/lab01
```

The `mv` command will move the file located at `~/Downloads/lab01.py` to the directory `~/cs61a/lab/lab01`.

Now, go back into the `lab01` folder that we made earlier.

```
cd ~/cs61a/lab/lab01
```

We're ready to start editing a file. Don't worry if this seems complicated — it will get much easier over time. Just keep practicing! You can also take a look at [Lab 0](#) for a more detailed explanation of terminal commands.

Installing a text editor

The Python interpreter that you installed earlier allows you to *run* Python code. You will also need a text editor, which will help you *write* Python code.

A text editor is similar to Microsoft Word — a program that allows you to write in one or more languages. You will be using a text editor to create, modify, and save files.

There are many editors out there, each with its own set of features. You can choose your own editor, but for the purpose of this class, your text editor must

- Have an easy way to open and edit new files
- Work well with Python files and plain text in general
- Have line numbers

In addition, almost all useful editors have

- Syntax highlighting
- Keyboard shortcuts

Here are the text editors that we recommend. We've written short guides for each one to get you started:

- [Sublime Text](#) (a popular choice among students)
- [Emacs](#)
- [Vim](#)

Each guide has a section on installing the text editor to your personal computer. For the sake of this lab, you can just jump to there, but come back to finish up the lab!

Note: Please, please, *please* do not use Microsoft Word to edit programs. Word is designed to edit natural languages like English — trouble will ensue if you try to write Python with Word!

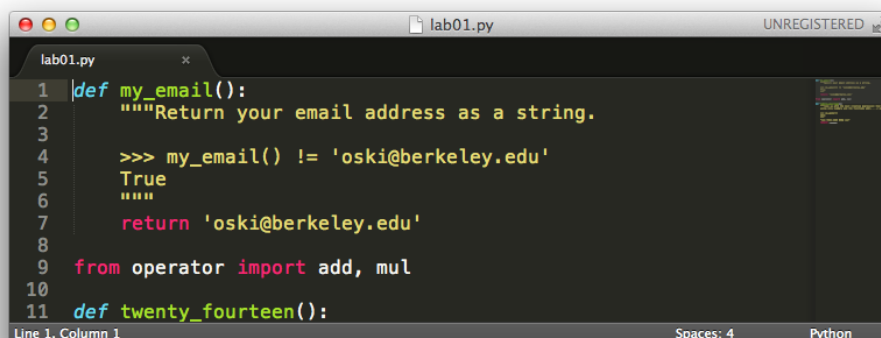
Note: Python usually comes with a text editor called IDLE. We do not recommend it because you won't gain as much experience using the command line to run your Python programs. This is very important once you start working on the projects for this class so we highly recommend that you use the command line to run your files from day 1!

A first look at lab01.py

Notice the various blocks of yellow text that are wrapped within three quotation marks `"""`. That text is called a docstring, which is a description of what the function is supposed to do.

Within the docstring, there are lines that begin with `>>>`. These lines are called doctests. Doctests are a great way to explain what the function does by showing actual Python code: "if we input this Python code (the lines that say `>>>`), what should the expected output be (the lines underneath the `>>>`)?"

In particular, let's look at the `my_email()` function:

A screenshot of a code editor window titled 'lab01.py'. The code defines a function `my_email()` with a docstring. The docstring contains a doctest: `>>> my_email() != 'oski@berkeley.edu'` followed by `True`. The function returns `'oski@berkeley.edu'`. Below the function, there is an import statement `from operator import add, mul` and the start of another function `def twenty_fourteen():`. The status bar at the bottom indicates 'Line 1, Column 1', 'Spaces: 4', and 'Python'.

The doctest checks that you changed the return value (return ...) from the default `'oski@berkeley.edu'`.

Note: You should never change the doctests in your assignments! The only part of your assignments that you'll need to edit is the code.

How do we use these tests? Glad you asked!

Running doctests

To run our doctests, switch over to our terminal and type in the following command:

```
python3 -m doctest lab01.py
```

If you are using Windows and the `python3` command doesn't work, try using just `python` or `py`. If neither of those work, take another look at the video in the section on [installing Python](#) to make sure you are setting up your PATH correctly. Ask a TA or lab assistant for help if you get stuck!

This will print out a lot of output that shows which tests we are currently failing. You should see something like this:

```
lab01 — bash — 70x32
Elizabeth@Mac~/cs61a/lab/lab01$ python3 -m doctest lab01.py
*****
File "/Users/Elizabeth/cs61a/lab/lab01/lab01.py", line 4, in lab01.my_email
Failed example:
    my_email() != 'oski@berkeley.edu'
Expected:
    True
Got:
    False
*****
File "/Users/Elizabeth/cs61a/lab/lab01/lab01.py", line 15, in lab01.twenty_fourteen
Failed example:
    twenty_fourteen()
Exception raised:
Traceback (most recent call last):
  File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/doctest.py", line 1324, in __run
    compileflags, 1), test.globs)
  File "<doctest lab01.twenty_fourteen[0]>", line 1, in <module>
    twenty_fourteen()
  File "/Users/Elizabeth/cs61a/lab/lab01/lab01.py", line 18, in twenty_fourteen
    return
NameError: name '_____' is not defined
*****
2 items had failures:
  1 of 1 in lab01.my_email
  1 of 1 in lab01.twenty_fourteen
***Test Failed*** 2 failures.
Elizabeth@Mac~/cs61a/lab/lab01$
```

Oh man, we had 2 failures! :(Before we begin fixing our bugs, let's see what this output says. In particular, look at the `my_email` test that we failed:

```
lab01 — bash — 75x9
*****
File "/Users/Elizabeth/cs61a/lab/lab01/lab01.py", line 4, in lab01.my_email
Failed example:
    my_email() != 'oski@berkeley.edu'
Expected:
    True
Got:
    False
*****
```

Notice it says we had an error in `lab01.py` on line 4, in the function `my_email`. Now we know exactly where to look for the bug! (This is why line numbers are a must for text editors.)

After we find line 4, let's try to understand what the test is saying: the function `my_email`, when called with zero inputs, should not return the string `'oski@berkeley.edu'`. The problem is, ours is returning that string! Change that to your `berkeley.edu` email:

```
def my_email():  
    """Return your last name as a string.  
  
    >>> my_email() != 'oski@berkeley.edu'  
    True  
    """  
    return 'jane.doe@berkeley.edu'
```

Once you've changed the return value of the function `my_email` (make sure that you're returning a string, which has quotes around it), try running the doctests again from your terminal:

```
python3 -m doctest lab01.py
```

The test for `my_email` should pass. One down, one to go!

For the second function, `twenty_fifteen`, replace the blank line with an expression that evaluates to the number 2015, using only integers and the functions `add` and `mul`. There are many simple ways to do this — try to come up with the most creative solution! Make sure your doctests pass once you're done.

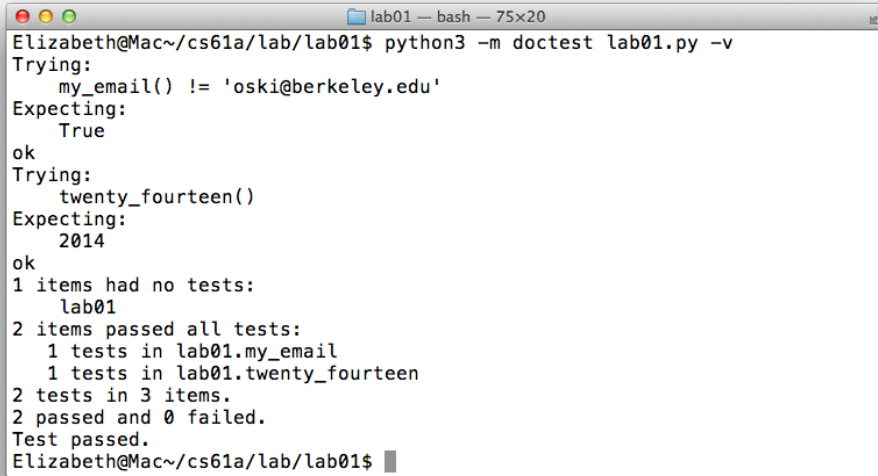
You might run your doctests and see that there is no output. This is actually good — it means all the tests passed. By default, doctests only print output when you have failures in your tests.

However, if you want to double-check that you are passing all of the tests, you can add a command line flag `-v` to see *all* the output, including the tests that passed:

```
python3 -m doctest -v lab01.py
```

This is called "verbose" mode.

Once all of your tests pass, the verbose output of the doctests (`python3 -m doctest -v lab01.py`) should look something like this:

A screenshot of a terminal window titled 'lab01 -- bash -- 75x20'. The prompt is 'Elizabeth@Mac~/cs61a/lab/lab01\$'. The command entered is 'python3 -m doctest lab01.py -v'. The output shows two tests: 'my_email()' and 'twenty_fourteen()'. Both tests pass. The final output is 'Test passed.' and the prompt returns.

```
Elizabeth@Mac~/cs61a/lab/lab01$ python3 -m doctest lab01.py -v
Trying:
    my_email() != 'oski@berkeley.edu'
Expecting:
    True
ok
Trying:
    twenty_fourteen()
Expecting:
    2014
ok
1 items had no tests:
    lab01
2 items passed all tests:
   1 tests in lab01.my_email
   1 tests in lab01.twenty_fourteen
2 tests in 3 items.
2 passed and 0 failed.
Test passed.
Elizabeth@Mac~/cs61a/lab/lab01$
```

For more help throughout the semester on fixing bugs, take a look at [Albert's debugging guide](#).