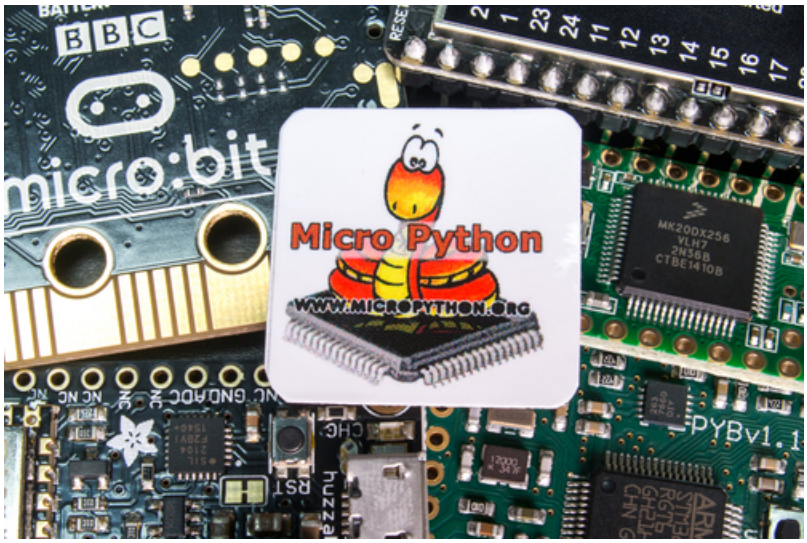
adafruit learning system

MicroPython Basics: Load Files & Run Code

Created by Tony DiCola



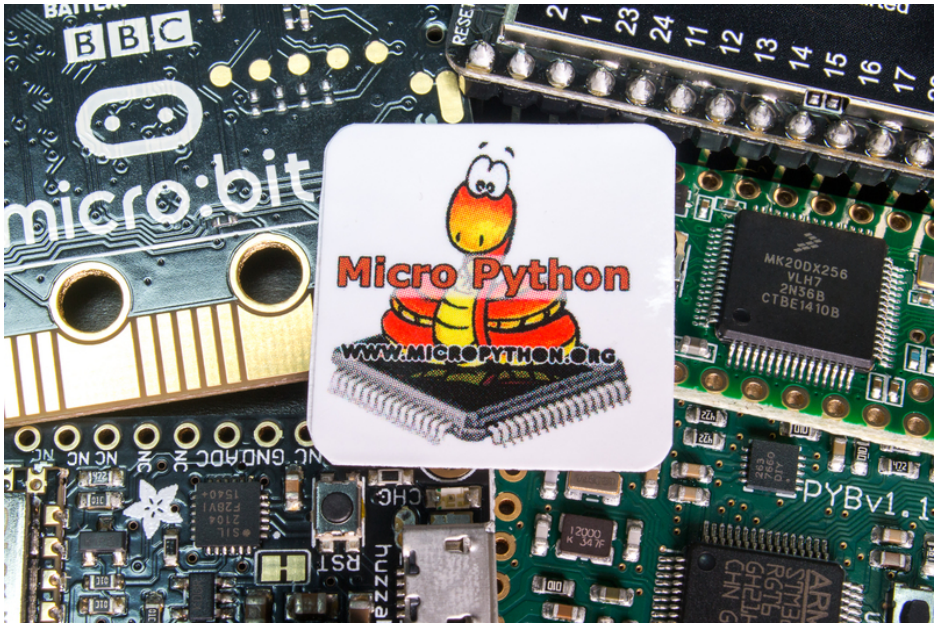
Last updated on 2018-12-10 10:33:43 PM UTC

Guide Contents

Guide Contents 2
Overview 3
Install ampy 5
Upgrade Ampy 6
Source Install 6
Disable ESP8266 Debug Output 9
Run Code 10
File Operations 13
Copy Files to Board 13
Copy Directories to Board 13
Read Files From Board 14
Create Directories 14
List Directories 15
Remove Files & Directories 15
Boot Scripts 17

Overview

Note this guide was written for MicroPython.org firmware and not Adafruit CircuitPython firmware.



This guide explores how to load files and run code on a MicroPython board. In the earlier introductions to MicroPython you manually typed all the code you wanted to run into the board's serial REPL. This process is great for learning and experimenting, but not great for developing and running complex programs because you have to type in the program every time you want it to run. However MicroPython has an internal filesystem which can store code that's run whenever the board powers up, just like an Arduino runs an Arduino sketch. Using a simple tool you can learn how to load code and other files into MicroPython's filesystem and enable an 'Arduino-like' workflow for developing code on your computer that runs on a MicroPython board.

Before you get started be sure to check your board's documentation (https://adafru.it/pXc) for more details on its filesystem. Some MicroPython boards like the pyboard have a microSD card which can store large amounts of data in its filesystem. Other boards like the ESP8266 reserve just a small part of their internal flash memory for the filesystem. Each board is slightly different in how it creates and uses its filesystem so check your board's documentation for more details.

For this guide we'll use the **Adafruit MicroPython tool (ampy) (https://adafru.it/r1F)** to load files and run code on a MicroPython board. If you're curious ampy is not the only tool for manipulating files and more on a MicroPython board, there are several other tools such as:

- **ESP8266 web REPL** - For ESP8266-based boards the web REPL provides a basic web interface for uploading files to the board. This is handy for dropping a file on a board, but it requires being connected to the web REPL which might not always be convenient.
- **rshell (https://adafru.it/q2a)** - rshell is a remote MicroPython shell tool which allows you to access the files and more from a MicroPython board connected over its serial/USB connection. Check out the rshell forum post (https://adafru.it/q2b) for more details on its usage.
- **mpfshell (https://adafru.it/q2c)** - mpfshell is similar to rshell and provides file and REPL access in a MicroPython-specific shell. However mpfshell is made specifically to support ESP8266-based boards and the WiPy board. Check out the mpfshell forum post (https://adafru.it/pFe) for more details on its usage.

This guide uses `ampy` because it is a simple cross-platform command line tool that provides just enough functionality to access MicroPython's filesystem without being too complex. Feel free to explore other tools and options once you learn about MicroPython's filesystem.

Also be aware `ampy` does not support talking to boards without a serial/USB REPL connection. In particular the WiPy board requires accessing the REPL over telnet and won't currently work with ampy. Consider using the mpfshell tool mentioned above, or even PyCom's editors and tools.

Before continuing make sure you have a MicroPython board and can access its serial REPL. (https://adafru.it/pNC) If you're new to MicroPython start by reading these guides that explain what it is and how to get started:

- MicroPython Basics: What is MicroPython? (https://adafru.it/pMb)
- MicroPython Basics: How to Load MicroPython on a Board (https://adafru.it/pNB)
- MicroPython Basics: Blink a LED (https://adafru.it/q2d)

Install ampy

To install the Adafruit MicroPython tool (https://adafru.it/r1F) (ampy) you'll first need to make sure you have Python (https://adafru.it/cFQ) installed on your computer. The tool will work with either Python 2.7.x or 3.x so you can use whichever version you prefer.

For Linux and Mac OSX you probably already have a version of python installed--trying running the python or pip command to see that it's available. If you don't see python installed consult your package manager or a tool like Homebrew (https://adafru.it/df3) to easily install it.

For Windows you'll need to install Python and be sure to check the box during installation to add python to your system path.

Once Python is avaialble on your system you can easily install ampy from the Python package index. If you're using **Python 2.7.x** open a terminal and run this command:

Note: If you have both Python 2.7.x and Python 3.x on your Windows computer, make sure you are running the Python 3.x version of pip. Having both versions of Python in your PATH statement is not enough. If the install of ampy is successful, you'll see it in your C:\Python37Path\Scripts\ folder, or replacing Python37Path with your local install path.

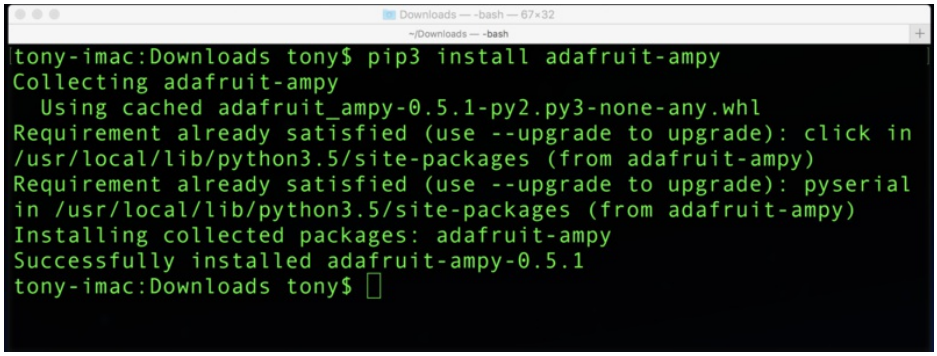
```
pip install adafruit-ampy
```

Note on some Linux and Mac OSX systems you might need to install as root with sudo:

```
sudo pip3 install adafruit-ampy
```

Or if you'd like to use **Python 3.x** run the **pip3** command instead (using sudo if necessary):

```
pip3 install adafruit-ampy
```



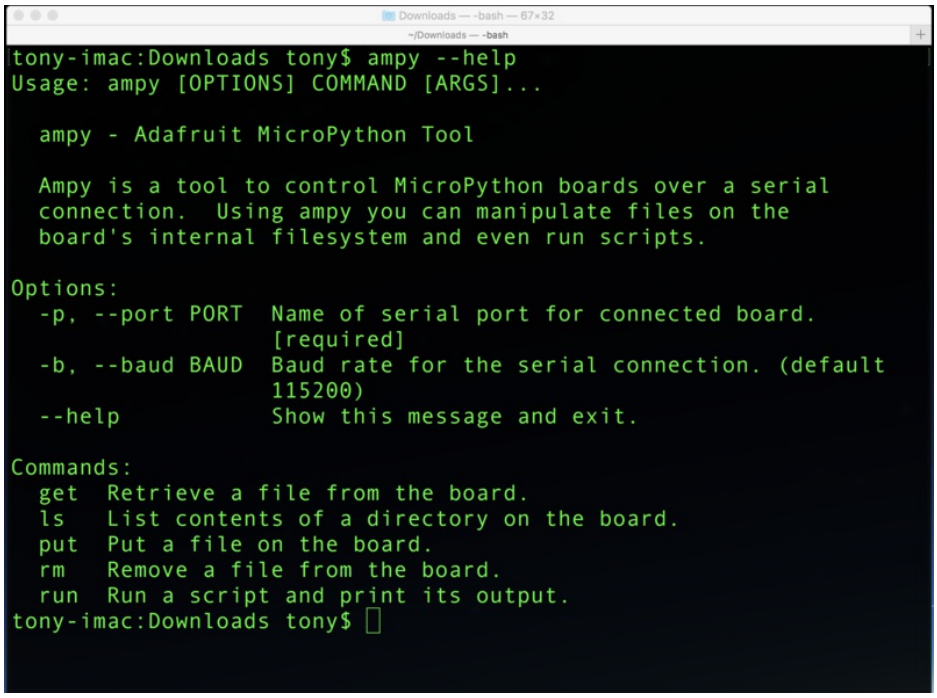
Finally in some rare cases like Mac OSX with Homebrew and multiple Python versions installed you might need to use the pip2 command to explicitly install in Python 2.7.x:

```
pip2 install adafruit-ampy
```

Make sure the pip command finishes without an error. If you see an error then go back and check you have python installed and are running it as root with sudo if necessary.

To check that ampy installed successfully run the following command to print its usage:

```
ampy --help
```



You should see usage information for the tool printed, like what commands it has and options for using them. If you see an error instead go back and carefully check the pip install command above succeeded, and that python is in your system path.

Upgrade Ampy

If you installed ampy with pip you can run a small command to check for an updated version and install it. Just add the **--upgrade** option to the install commands above, for example to upgrade ampy with Python 3 you can run:

```
pip3 install adafruit-ampy --upgrade
```

Make sure to add **--upgrade** to the end of the pip install command you use to install ampy. If you forget the upgrade parameter pip won't install the latest version!

Source Install

If you'd like to install ampy from its source on GitHub (https://adafru.it/r1F) you can do so easily with a few commands. If you followed the above steps to install from the Python package index this isn't necessary, but if you'd like the current code or are perhaps modifying it then you'll want to install from source.

First download the source (https://adafru.it/q2e) or use the git tool to clone it from GitHub:

```
git clone https://github.com/adafruit/ampy.git
```

Then in a terminal navigate to the directory with the cloned or extracted source and run the following command to

install with **Python 2.7.x**:

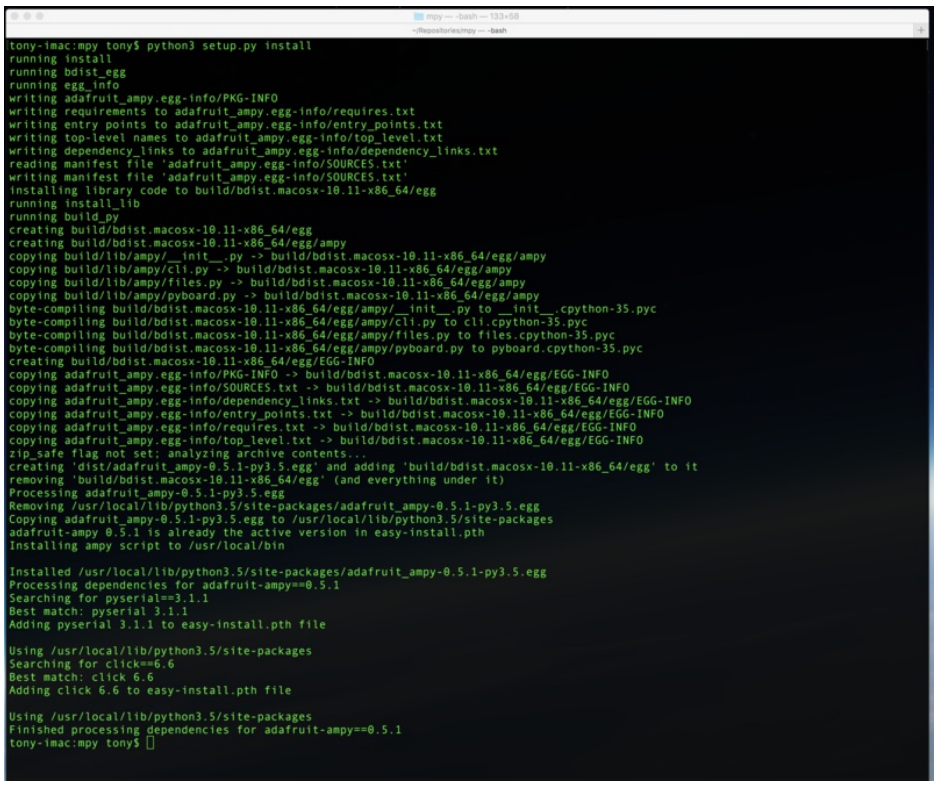
```
python setup.py install
```

Note on some Linux and Mac OSX machines you might need to run as root with sudo:

```
sudo python setup.py install
```

Or to install for **Python 3.x** use the **python3** command (using sudo when necessary too):

```
python3 setup.py install
```



Carefully inspect the output of the command to make sure it finished without an error or exception. You should see something like 'Finished processing dependencies for adafruit-ampy...' as the last line. Once installed in this way the ampy tool should be available in your path just like if installed from the Python package index.

One final way to install ampy from source is in **develop** mode, this way the cloned / downloaded code will actually be the code Python runs instead of copying and installing it into an internal Python module cache. This is handy if you're working on the code and want to see your changes immediately updated. To install in develop mode just run the setup.py command above but change **install** to **develop**.

Also note on Python 2.7.x if you plan to run the unit tests in the source code you will need the mock module installed:

```
pip install mock
```

Disable ESP8266 Debug Output

> These instructions are only for ESP8266 board users!

For ESP8266-based boards before using a tool like ampy you might need to disable debug output on the board. If you're using an official release build (https://adafru.it/pMd) of MicroPython (i.e. one that ends in a simple version like **1.8.3** instead of a more complex daily build like **1.8.3-38-gf2a21a2**) debug output is **already disabled** and you don't need to do anything extra. However if you're using a daily build or custom build from source you'll need to disable debug output that can confuse tools like ampy.

To disable debug output connect to the board's serial REPL and run the following commands:

```
import esp
esp.osdebug(None)
```

The `esp.osdebug` function should run and return no output. After running the command debug output will not be printed to the serial terminal and you can use tools like ampy.

It is **highly recommended** to add the above two lines to the board's **boot.py** so debug output is disabled permanently. If you don't make this change you'll need to manually disable debug output every time you reset the board! You can learn more about the **boot.py** file on the Boot Scripts page of this guide (https://adafru.it/q2f).

Run Code

> Before using ampy with the ESP8266 be sure you've disabled debug output if necessary: https://learn.adafruit.com/micropython-basics-load-files-and-run-code/disable-esp8266-debug-output

Using `ampy` you can take Python code written on your computer and run it on a connected MicroPython board. This gives you a simple workflow for exploring MicroPython. Write code on your computer in your favorite text editor, then use ampy's **run** command to run it on a board!

To use the **run** command just specify a path to a Python file on your computer. Ampy will send the file to the board, wait for it to finish running, and print any output from the program.

For example create a file **test.py** on your computer and save inside it the following Python code:

```
print('Hello world! I can count to 10:')
for i in range(1,11):
    print(i)
```

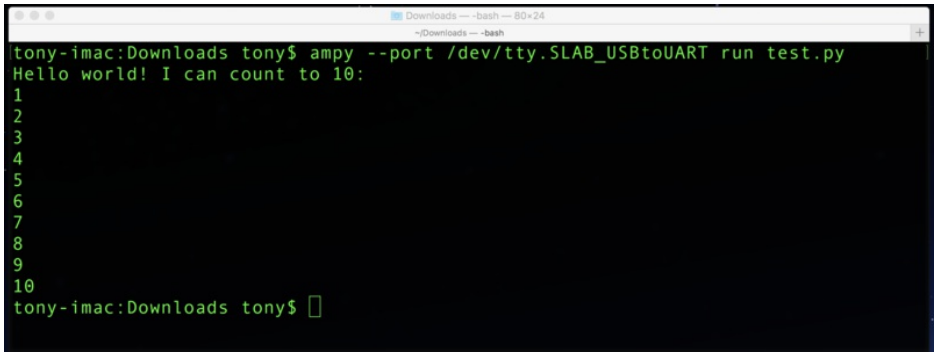
In a terminal in the same directory as **test.py** run the following ampy command to execute the script on a connected MicroPython board:

```
ampy --port /serial/port run test.py
```

Where **/serial/port** is the path or name of the serial port connected to the MicroPython board.

> If you don't want to constantly specify the --port option you can set the AMPY_PORT environment variable in your terminal session and ampy will use it as the board's serial port.

You should see the output of the code after it was run on the board:



If you receive an error that ampy failed to receive the expected response be sure you disabled debug output as mentioned at the top of the page! Also double check the board is connected to your computer and you are specifying the correct serial port for the board. Be sure the file test.py is in the same directory as you're running the ampy command too.

Be aware the run command is not a shell or tool that allows you to send input from your computer to the board! If you

need to send input you'll want to connect to the board and use its serial REPL (https://adafru.it/pMf).

By default the run command will wait for the script to finish running on the board before printing its output. In some cases you don't want this behavior--for example if your script has a main or infinite loop that never returns you don't want ampy to sit around waiting forever for it to finish. In this case add the **--no-output** option to the run command. This flag tells ampy not to wait for any output and instead just start running the script and return.

For example modify the **test.py** script so that it counts numbers forever in an infinite loop:

```
import time
print('Hello world! I can count:')
i = 1
while True:
    print(i)
    i += 1
    time.sleep(1.0)  # Delay for 1 second.
```

Then run it with the **--no-output** option and notice it immediately returns:

```
ampy --port /serial/port run --no-output test.py
```

However open the board's serial REPL and watch it count numbers every second!

Remember the program is still running, ampy just didn't wait for it to stop!

The **--no-output** option is great for writing scripts that are like an Arduino sketch. In Arduino you have an explicit **setup** and **loop** function which you fill in with code that runs once (in **setup**) and code that runs forever (in **loop**). MicroPython doesn't have exactly the same concept, but you can create it yourself in your own Python scripts!

In fact look at the **test.py** above and notice all the code before the **while True** loop is like the **setup** function from an Arduino sketch, it's executed just once at the start of the program. Then the code inside the **while True** loop is like the **loop** function from Arduino, this code runs repeatedly as fast as possible. To make it a little more clear here's the **test.py** with comments that show where the setup code goes and where the loop code goes:

```
##############################################################################
# Setup code goes below, this is called once at the start of the program: #
##############################################################################
import time
print('Hello world! I can count:')
i = 1

while True:
    ######################################################################
    # Loop code goes inside the loop here, this is called repeatedly: #
    ######################################################################
    print(i)
    i += 1
    time.sleep(1.0)  # Delay for 1 second.
```

If you're coming to MicroPython with a background in Arduino, consider writing your MicroPython scripts in a similar style as the above. Put your setup code first and then a main loop that runs forever. Just be sure you add the **--no-output** option when running with ampy so it knows not to wait for the script to finish!

File Operations

In addition to running code `ampy` you can also manipulate files on a MicroPython board's filesystem. You can copy files from your computer to the board, read files from the board back to your computer, and even create and manage directories on the board.

Think of the filesystem on a MicroPython board like the filesystem on your computer. Just like on your computer your board can have a complex hierarchy of directories with files and other subdirectories inside them. MicroPython's filesystem is similar to Unix filesystems that separate parts of the path with forward slashes ('/') between parent directories. For example a file **/foo/bar.txt** on a MicroPython board exists in a folder **foo** under the root of the board.

Copy Files to Board

The **put** command can copy files from your computer to a MicroPython board. This is great for copying over Python source code and other files you create on your computer.

For example to copy a file called **test.py** from your computer to the root of a MicroPython board's filesystem under **/test.py** run the following command:

```
ampy --port /serial/port put test.py
```

Where **/serial/port** is the path or name of the serial port connected to the MicroPython board. Make sure **test.py** is in the same directory as you're running the command too. If the file isn't there then specify the full path to it on your computer.

You can also put the file on the board in a path other than the root. Just specify as another argument the full path and filename to use on the board. For example to copy a **test.py** from your computer to a file **/foo/bar.py** on the board run (note the parent foo directory must already exist!):

```
ampy --port /serial/port put test.py /foo/bar.py
```

The put command will always overwrite files on the board without warning!

Copy Directories to Board

In addition to copying files the **put** command can also copy an entire directory and all of its child files and folders to the board. This is perfect for copying a MicroPython module or other directory to the board. For example if you have a folder called **adafruit_driver** that contains files and subfolder with driver code, you can copy it to your board with a command like:

```
ampy --port /serial/port put adafruit_driver
```

This command will copy the **adafruit_driver** folder (which should be in the same directory as the terminal you're running the command from) to the board's root. If the folder already exists the contents will be copied over and replaced without warning!

You can also change the path that the folder is copied into, for example to copy **adafruit_driver** to the path **/foo/adafruit_driver_2** on the board you can run:

```
ampy --port /serial/port put adafruit_driver /foo/adafruit_driver_2
```

The put command will always overwrite files on the board without warning!

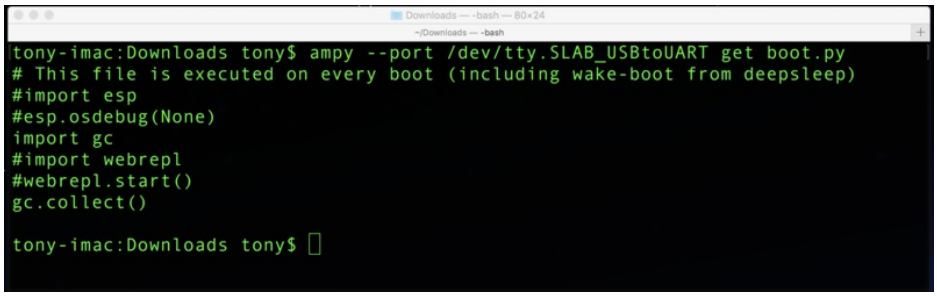
Be sure you've updated ampy to the latest version (https://adafru.it/tZA) as earlier versions did not support directory copying with put!

Read Files From Board

The **get** command can read and copy files from a MicroPython board to your computer.

For example to print the contents of **/boot.py** from a board run the following command:

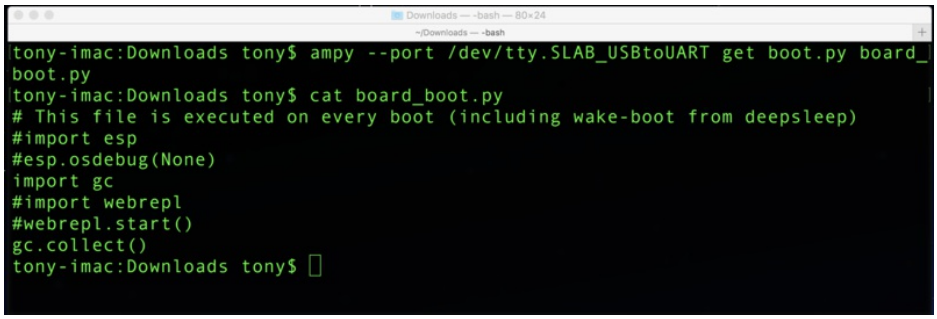
```
ampy --port /serial/port get boot.py
```



This will print out the contents of **boot.py** from the board's root directory.

You can instead copy the contents of **boot.py** into a file on your computer by specifying the path to the file to save as a second argument. For example to copy **/boot.py** from a board to a file **board_boot.py** on your computer you can run:

```
ampy --port /serial/port get boot.py board_boot.py
```



The get command will always overwrite files on the computer without warning!

Create Directories

You can create hierarchies of folders on the MicroPython board's filesystem with the **mkdir** command.

For example to create a **/foo** folder under the root of a board run the following command:

```
ampy --port /serial/port mkdir foo
```

You can create directories inside directories too, for example to create a folder **bar** inside the **foo** folder above you can run:

```
ampy --port /serial/port mkdir /foo/bar
```

Make sure the parent foo directory exists before trying to create the bar subdirectory inside of it! The mkdir command won't create parent directories that don't exist.

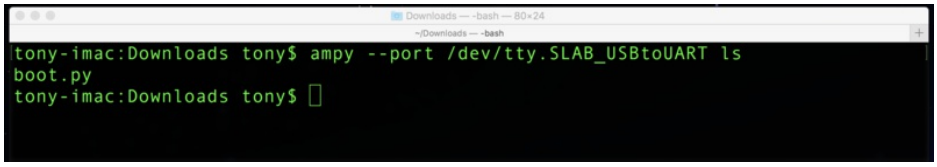
List Directories

You can list the file and folder contents of a directory with the **ls** command.

If you don't specify any argument to the ls command then the contents of the MicroPython board's root will be listed. However if you'd like to list the contents of a different directory just specify its path on the board as an argument.

For example to list the root contents of a board run:

```
ampy --port /serial/port ls
```



Or to list the contents of a subfolder **foo** run:

```
ampy --port /serial/port ls /foo
```

Remove Files & Directories

The **rm** command can remove a file or directory from a MicroPython board's filesystem. To use the command just specify as an argument the path to the file or directory on the board to delete. Note that directories must be empty before they can be deleted!

For example to delete a file **test.py** in the root of a board run the following commmand:

```
ampy --port /serial/port rm test.py
```

Or to delete a folder **/foo/bar**, assuming it's empty, run the following command:

```
ampy --port /serial/port rm /foo/bar
```

In addition ampy now has a **rmdir** command that will remove a directory and all of its child files and folders (even if they

aren't empty). For example to delete the folder /foo/bar on the board filesystem regardless of it containing child files and folders run the following command:

```
ampy --port /serial/port rmdir /foo/bar
```

The rm and rmdir commands will delete files and folders without warning or asking to confirm!

Be sure you've updated ampy to the latest version (https://adafru.it/tZA) as earlier versions did not support the rmdir command.

Boot Scripts

There are two important files that MicroPython looks for in the root of its filesystem. These files contain MicroPython code that will be executed whenever the board is powered up or reset (i.e. it 'boots'). These files are:

- **/boot.py** - This file is run **first** on power up/reset and should contain low-level code that sets up the board to finish booting. You typically don't need to modify boot.py unless you're customizing or modifying MicroPython itself. However it's interesting to look at the contents of the file to see what happens when the board boots up. Remember you can use the ampy **get** command to read this and any other file!
- **/main.py** - If this file exists it's run **after** boot.py and should contain any main script that you want to run when the board is powered up or reset.

The **main.py** script is what you can use to have your own code run whenever a MicroPython board powers up. Just like how an Arduino sketch runs whenever the Arduino board has power, writing a **main.py** to a MicroPython board will run that code whenever the MicroPython board has power.

You can create and edit the **main.py** on a board using the file operations in ampy (https://adafru.it/q2A). For example create a **test.py** on your computer and put the following Python code inside it:

```
# Setup code goes below, this is called once at the start of the program: #
import time
print('Hello world! I can count:')
i = 1

while True:
    # Loop code goes inside the loop here, this is called repeatedly: #
    print(i)
    i += 1
    time.sleep(1.0)  # Delay for 1 second.
```

Then copy the file to **/main.py** on a connected MicroPython board with `ampy`'s put command:

```
ampy --port /serial/port put test.py /main.py
```

Reset the board or unplug it and plug it back in, then connect to the serial REPL and notice the board is counting numbers! The **main.py** code started as soon as the board finished booting.

Putting all the pieces of this guide together you can see a simple workflow for MicroPython that's similar to Arduino & Arduino sketches:

- Write Python code on your computer using your favorite text editor. Structure the code so it puts setup code at the top and loop code inside a main loop.
- Use the ampy **run** command with the **--no-output** option to run the script on the MicroPython board.
- Edit and run the script as much as you need for it to work the way you expect.
- When you want the code to automatically run on boot use the ampy **put** command to save the script as a **/main.py** file on the board.

That's all there is to loading files & running code on MicroPython boards!