Marten - Make Command Line Example

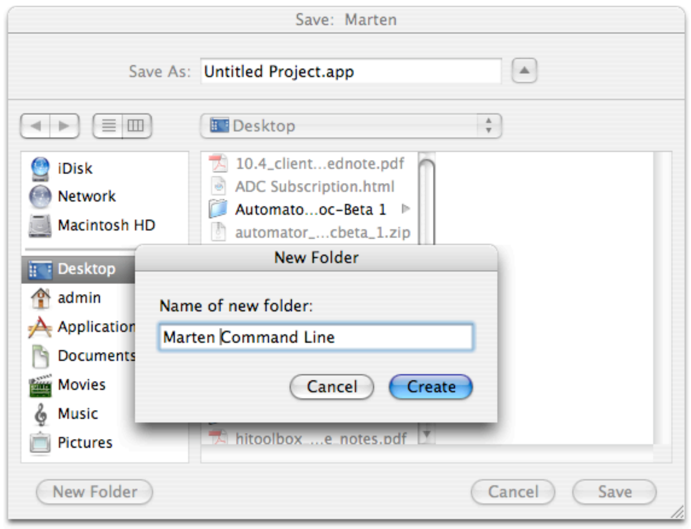
This document will show you how to create a command line tool using the Marten development environment.

Important - Read Me

Command line tools cannot be created by the Marten Update command and must instead be created by using the Marten Build command; they must be created as C applications. In this example, the Export functionality of the Marten IDE is used to emit the appropriate C language source code files for the example command line project. These files must then be compiled and linked by a third party C language development environment known as GCC (the GNU Compiler Collection) which will be spawned as a process by the Marten IDE during the build phase. Because these files will be linked against Marten frameworks, these frameworks must have been installed in the appropriate location for users to complete this example. In addition, users should have the GCC environment installed on their system. This will be the case if they have the Apple Xcode IDE installed.

Creating the Marten project

To begin, launch the Marten IDE and command click in the Projects window to create a new project. You will be prompted to save the project application. Create a new folder (in this example, on the Desktop) and name it "Marten Command Line".


Save: Marten

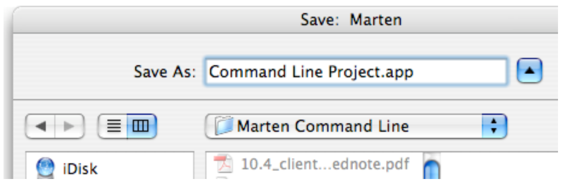
Save As: Untitled Project.app

Desktop

New Folder

Name of new folder: Marten Command Line

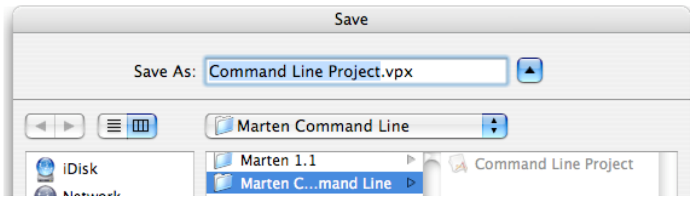
Then supply the name of the project application as "Command Line Project.app" and save it to the disk.


Save: Marten

Save As: Command Line Project.app

Marten Command Line

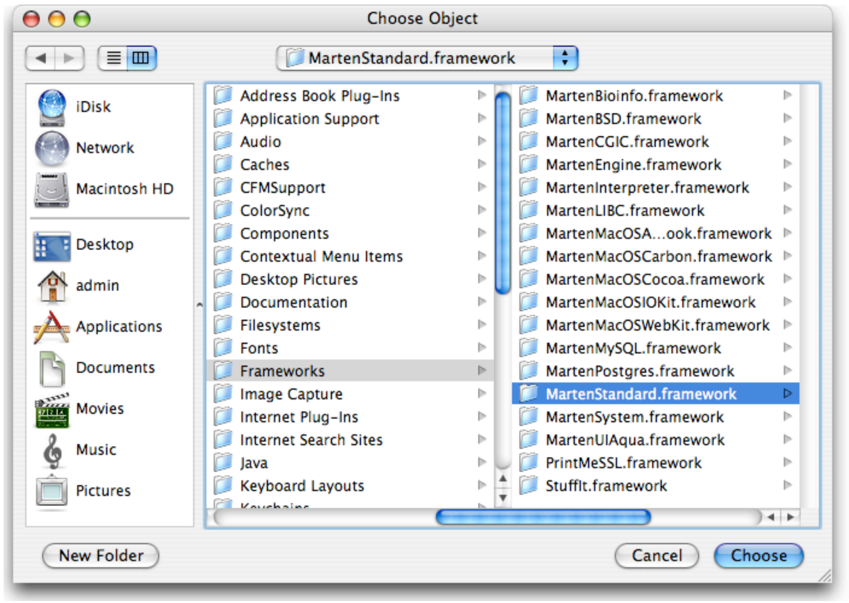
Name the project "Command Line Project" and save it to the disk. You will be prompted for a name and location. ALWAYS accept the suggested name. Naming of projects and sections in Marten is always done in the Projects and Sections windows, NOT when saving projects and sections to the disk.


Save

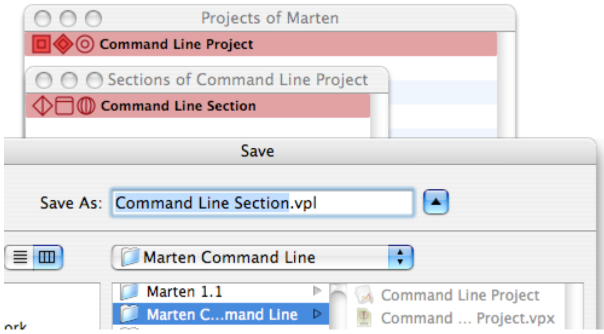
Save As: Command Line Project.vpx

Marten Command Line

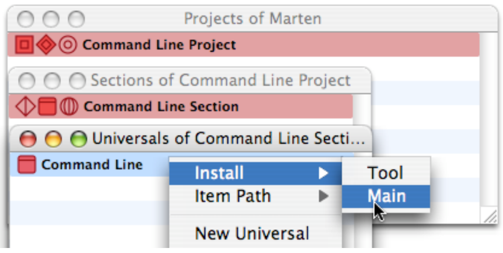
For this example, you will need the extensions provided by the Standard library. Select the "Add To Project" command under the File menu and choose the MartenStandard.framework file from the Frameworks folder where you installed the Marten libraries.



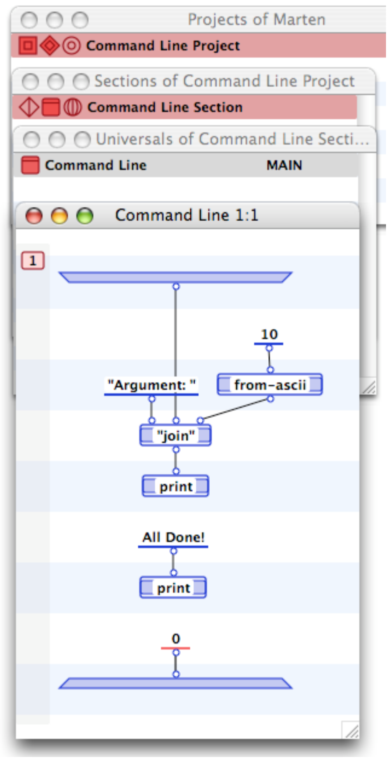
Double-click the Section icon of the Command Line Project item to open up a Sections window. Create a new section, name it "Command Line Section", and then save it to the disk. Because this section has never been saved before, you will be prompted for a name and location. As with the project, accept the suggested name and save the section to the project folder.



Open up the Universals window of the section, create a new universal called "Command Line", and install the "Main" type. This establishes the universal "Command Line" as the MAIN method of the project. It will automatically be created with one case containing one root on the input bar and one terminal on the output bar. A MAIN method must accept a list of strings as the input and supply an integer for the output.

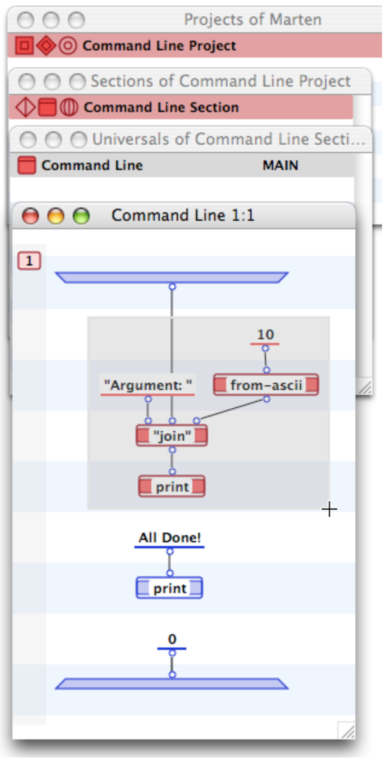


This example creates a command line application that simply parrots the command line arguments and then prints "All Done!". Supply this functionality by creating the code illustrated below.

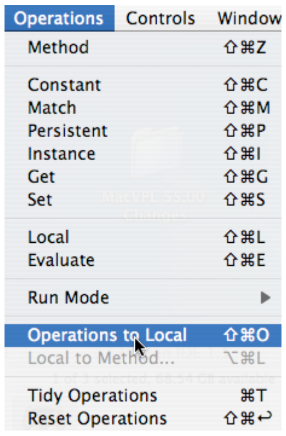


The arguments will be printed to the terminal with the prefix "Argument: ". In addition, we want each argument to be printed to a new line, so we append a Unix line feed (ASCII 10) to the end of each argument string.

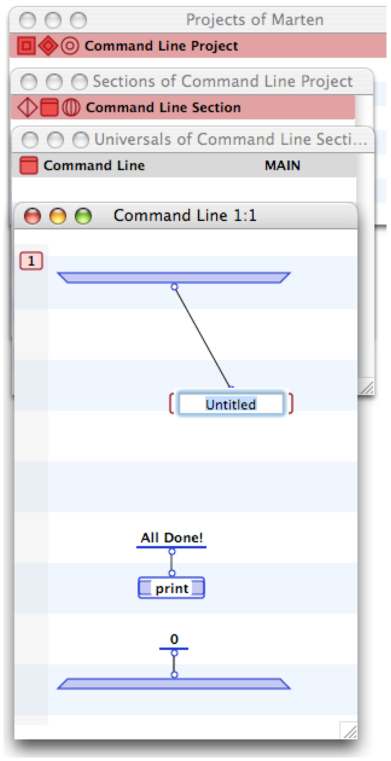
As stated before, a MAIN method has one input that contains a list of strings, one for each command line argument passed by the system to the application. As it stands, your code can only handle a single input string. You will modify it to handle a list of strings by using the "Operations to Local" code refactoring command. Select the operations to be incorporated into the local by depressing the mouse and dragging out a marquee rectangle containing the appropriate operations.



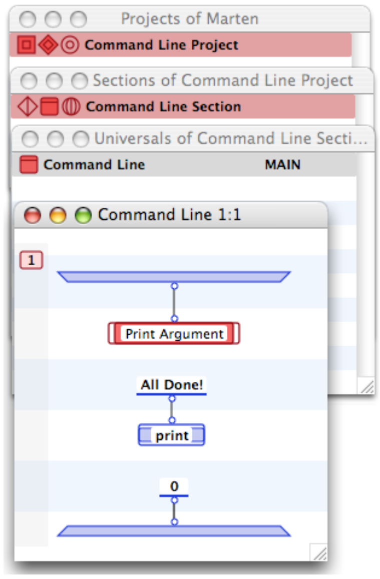
With those operations (and just those operations) selected, select the "Operations to Local" command from under the Operations menu.



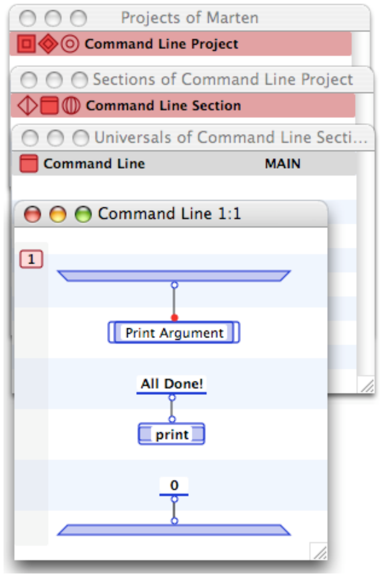
The operations will disappear and be replaced by an untitled local.



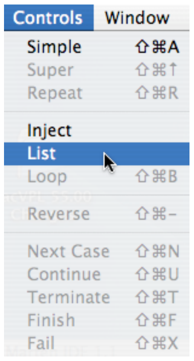
Rename the local "Print Argument" and tidy the window up.



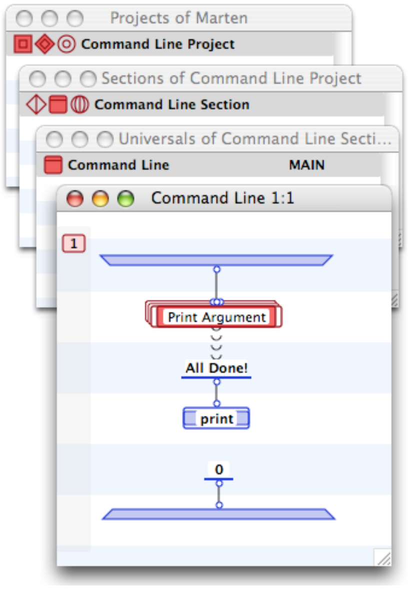
As it stands, this local does not handle a list input of strings. You must modify the terminal of the local to be a list terminal. Select the local input terminal.



Then choose the List command from under the Controls menu.



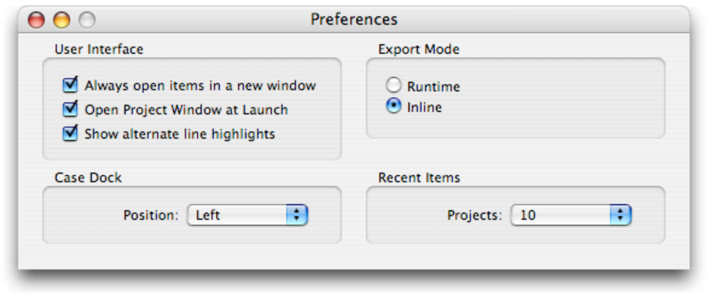
The terminal of the local will now change to a list terminal and the local will receive a Repeat control annotation (depicted by an "oval" to the left of the operation). To ensure that the argument printing is performed before the "All Done!", create a synchro from the local to the "All Done!" constant operation (select the local, hold the Option key down, and then click on the constant). The final code should look like:



As required by a MAIN method, this method accepts a list of strings and supplies an integer.

Exporting the project code

Because the Marten Update command can only generate an application bundle, it cannot be used to create a command line tool which is a single executable file. Therefore to create such a tool, you must export the code as C source code files and then compile and link them to generate a command line executable. In this example, you will export the code in the "Inline" C export format. Check that this export mode is the current one by opening up the Marten Preferences window. The Export Mode should be the same as this illustration:


Preferences

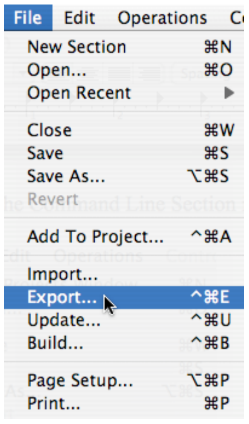
User Interface
- [✓] Always open items in a new window
- [✓] Open Project Window at Launch
- [✓] Show alternate line highlights

Export Mode
- Runtime [✓ Inline]

Case Dock
Position: Left

Recent Items
Projects: 10

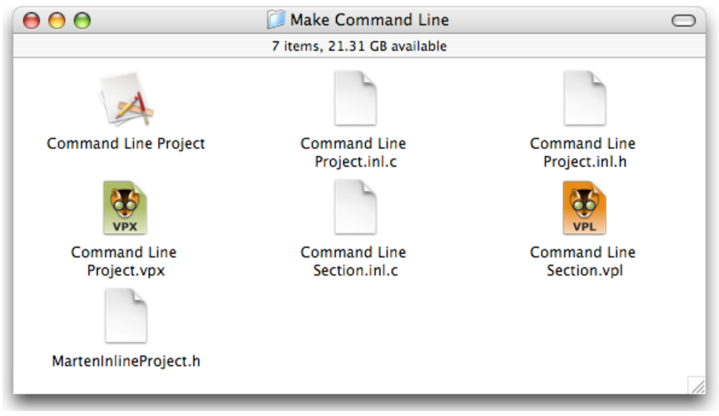
Select the Command Line Section item in the Sections window and then choose the File menu Export command.



A C source code file representing the graphical Prograph code of Marten will be created in the same folder as your section file.

Do the same for the Command Line Project file. When using the Inline format option, it is a good idea to ALWAYS export the project file when any changes are made, otherwise you may end up with undefined symbols when compiling.

The project folder will now contain the files:


Make Command Line

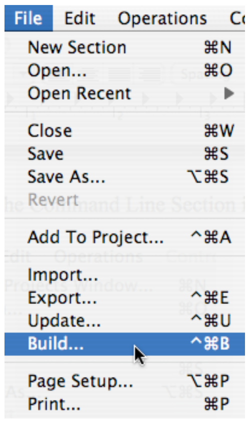
7 items, 21.31 GB available

- Command Line Project
- Command Line Project.inl.c
- Command Line Project.inl.h
- Command Line Project.vpx
- Command Line Section.inl.c
- Command Line Section.vpl
- MartenInlineProject.h

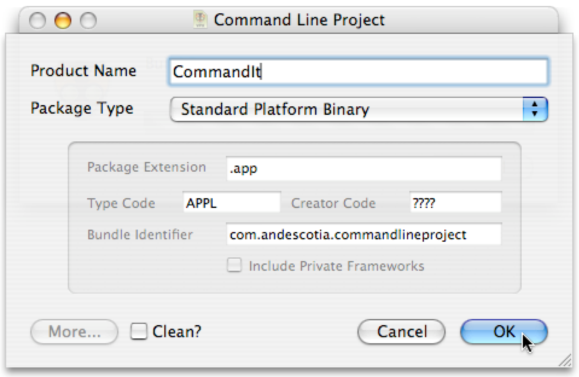
You are finished with Marten coding, so save the project and section. Next you will use Marten to build the command line tool.

Building the tool with Marten

You will build the application using Marten. Marten will spawn a GCC process that will compile and link your exported C files into a command line executable. To start this process, select the File menu Build command.

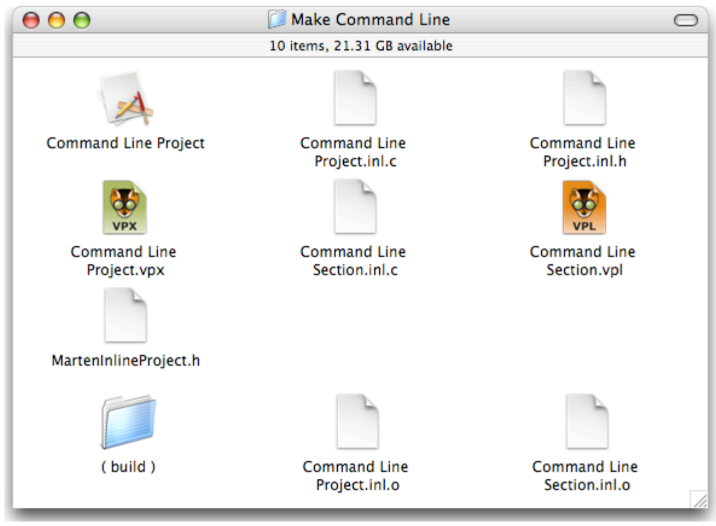


A settings dialog will open. Select the "Standard Platform Binary" package type and name your executable "CommandIt".


Command Line Project

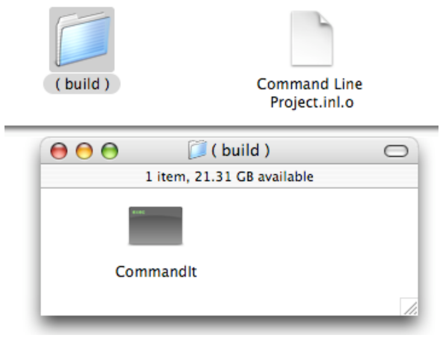
Product Name: CommandIt
Package Type: Standard Platform Binary
Package Extension: .app
Type Code: APPL
Creator Code: ????
Bundle Identifier: com.andescotia.commandlineproject
Include Private Frameworks: ☐
Clean?: ☐

Marten will compile and link an executable. After the dust settles your project folder should contain the following files:

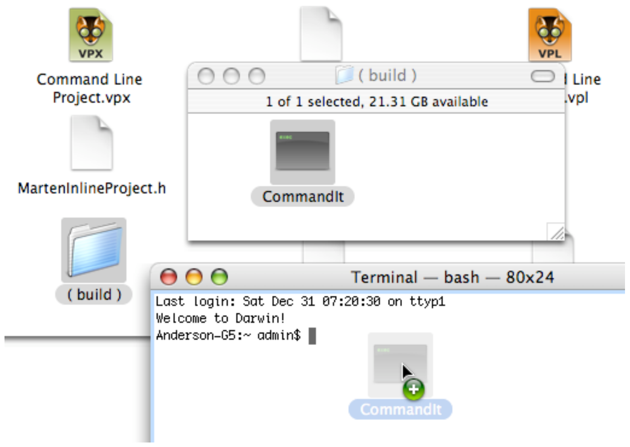

Make Command Line
10 items, 21.31 GB available

Command Line Project
Command Line Project.inl.c
Command Line Project.inl.h
Command Line Project.vpx
Command Line Section.inl.c
Command Line Section.vpl
MartenInlineProject.h
(build)
Command Line Project.inl.o
Command Line Section.inl.o

The actual CommandIt executable will be in the "(build)" subfolder:

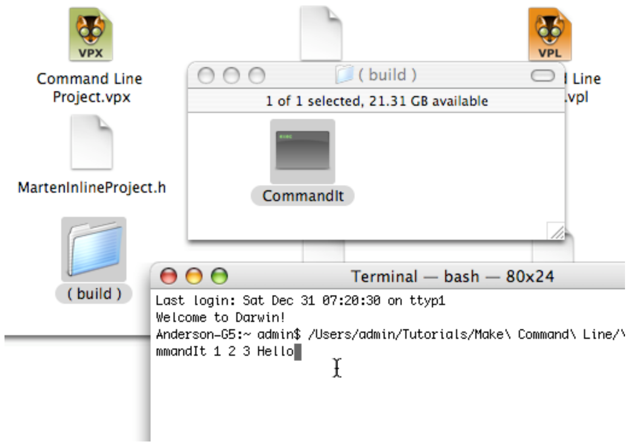


To try out the command line tool from Terminal, launch the Terminal application and open up a Terminal window. Drag the CommandLine executable and drop it on the Terminal window.



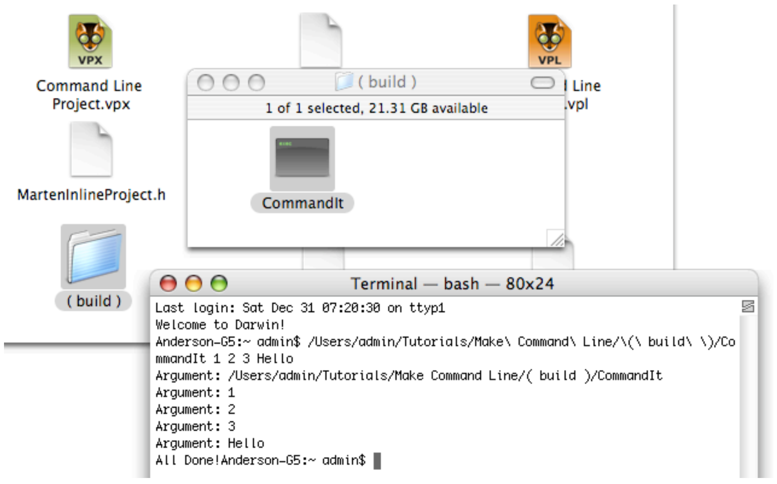
```
Last login: Sat Dec 31 07:20:30 on ttyp1
Welcome to Darwin!
Anderson-G5:~ admin$ 
```

The CommandIt executable path will appear in the Terminal window. Add a couple of additional command line arguments.



```
Last login: Sat Dec 31 07:20:30 on ttyp1
Welcome to Darwin!
Anderson-G5:~ admin$ /Users/admin/Tutorials/Make\ Command\ Line/\
mmandIt 1 2 3 Hello
```

Then hit the Return key. Your command line tool will run and should print out the arguments to the Terminal:



```
Last login: Sat Dec 31 07:20:30 on ttyp1
Welcome to Darwin!
Anderson-G5:~ admin$ /Users/admin/Tutorials/Make\ Command\ Line/\(\ build\ \)/Co
mmandIt 1 2 3 Hello
Argument: /Users/admin/Tutorials/Make Command Line/( build )/CommandIt
Argument: 1
Argument: 2
Argument: 3
Argument: Hello
All Done!Anderson-G5:~ admin$ 
```

It's that simple to create a command line tool in Marten!