This is the tutorial I used to create the slash-command slack bot for our Slack channel. It’s gotten pretty old and a few things have moved, so I did my best to correct any mistakes I saw here and clarify things that weren’t immediately apparent to me on my first time through. The original tutorial can be found here (hopefully):

<https://api.slack.com/tutorials/easy-peasy-slash-commands>

What is the difference between a bot and a slash command? Let's take a brief moment to understand the fundamental differences, and help you decide which you ought to be writing.

A [bot](https://api.slack.com/bot-users) is a program that connects to Slack just like you do: Slack effectively sees bots as users (but not as paid users, fear not!): They have access to the stream of messages that pass through Slack's servers, and they interact with Slack much like you do by reading and posting messages. Bots are great if you need a high degree of interactivity, or you want to be able to process the message stream as a gestalt.

A [slash command](https://api.slack.com/slash-commands), on the other hand, is like a command line utility for slack. It has a fixed syntax, and the program on the other end only ever sees the command itself. Likewise, it can only respond to that command in a limited way. Slash commands are great when you have a fixed set of interactions, and need to kick off a process on behalf of the user, or report structured data back to the user.

Most kinds of Slack Apps are best suited as slash commands, and so if you aren't sure, consider slash commands the default.

In this tutorial, I'm going to take you through the process of writing a slash command. If you decide that a bot is right for your idea, [have I got the answer for you](https://api.slack.com/tutorials/easy-peasy-bots-getting-started).

**Slash commands with Botkit**

Don't let the name fool you: [Botkit](http://howdy.ai/botkit/) is great for building more than just bots. [Howdy.ai](http://howdy.ai/) has done a lot (a *lot*) of the hard work for us, so why not leverage that?

First, we need to install some things. Make sure you have *node* installed on your local machine. If not, you have [several options for installing](https://nodejs.org/en/download/package-manager/) *node*.

Begin by forking and then cloning this GitHub repository: <https://github.com/DEGoodmanWilson/easy-peasy-slash-command-app>. This is just the simplest possible slash command project built around one of the Botkit example apps. It includes everything we need to not only write a slash command, but set it up for distribution via the ["Add to Slack" Button](https://api.slack.com/docs/slack-button) or even the [App Directory](https://slack.com/apps), easy-peasy.

I have a repo you would probably rather fork, as the repo mentioned above contains code that didn’t run on my local machine. Plus, this one actually echoes whatever you type into the slash command. It can be found here: <https://github.com/Immaculato/easy-peasy-slash-command-app.git>

This is a Node.js project, so of course once you have the repository on your local machine, you will need to run

npm install

to install the various dependencies.

Next, edit the package.json file to give your slash command a name, and to update the GitHub URLs to reflect the location of your fork.

**Exposing your app to the outside world**

Slash commands can't do anything without some way to invoke them. We need to be able to open a secure HTTP connection to a service running locally on your laptop. There are many tools that will let us open such a window, such as [\*ngrok](https://ngrok.com/) *and*[*Pagekite*](https://pagekite.net/)*. But I'm going to use*[*localtunnel*](http://localtunnel.me/)*. Just use \*npm* to install:

npm install -g localtunnel

and run localtunnel as such:

lt --port 8765 --subdomain awesomeslashcommand

Note that this runs the *app* on your local machine and exposes it on port 8765. We still need to register the actual *slash command* with slack, however. This is in the section that follows. Keep running this command (serving the app) when you run the slash command locally later, as the command needs to run on your actual app! (One app can host multiple slash commands, hence the 2 parts)

Once your slash command is running on local port 8765, it will be accessible at [https://myslashcommand.localtunnel.me](https://myslashcommand.localtunnel.me/), but your slash command will only be available for testing *really* once it is installed on Slack. (You might consider changing awesomeslashcommand to something more...memorable. And unique.)

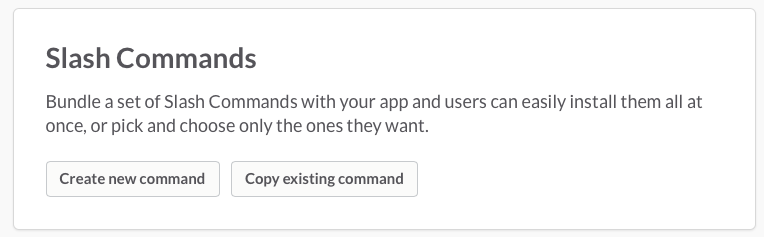
**Pass the good word to Slack**

Now we need to let Slack know that your slash command exists. Couldn't be easier. Go to <https://api.slack.com/applications/new> and fill out the form. Don't worry too much about the support URLs. But do be clear on the redirect URL. Look to the localtunnel output: The redirect URL will be

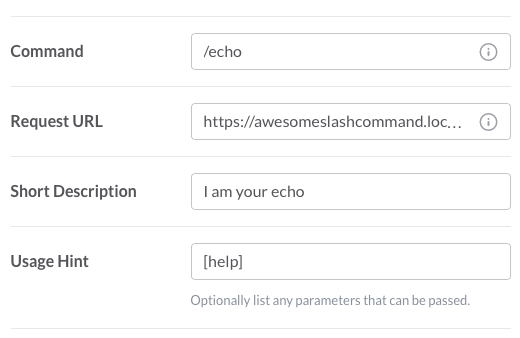
https:*//awesomeslashcommand.localtunnel.me/oauth*

This redirect URL section can be found in the *Manage Distribution* tab on the left side of the screen from your Slack app’s page (found from the link above). After clicking the tab, scroll down to “Add OAuth Redirect URLs” and add this url there.

On the next page, you will see a more detailed form for setting up your new app. Scroll down until you see the slash command section:



Click on "Create new command". You'll now be asked to fill in the details for your command. So that the command will work with the demo code, fill it out as such:



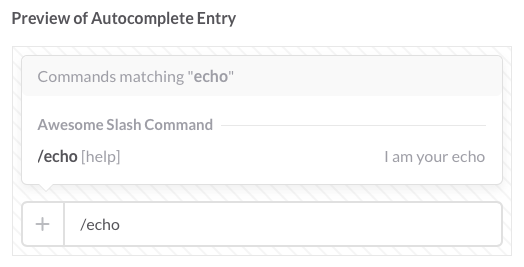
It's kind of hard to see: The Request URL should be set to

https:*//awesomeslashcommand.localtunnel.me/slack/receive*

Indeed, this is the URL you will use for *any* Botkit-based slash command — Botkit is prewired to expect *all* slash commands to arrive at this endpoint.

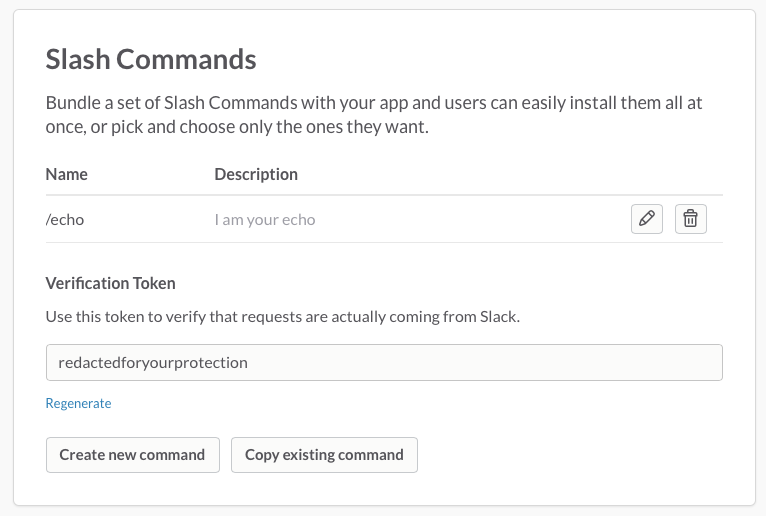
(A common gotcha at this point is having a URL that is not secure: Either it doesn't use HTTPS, or the security of the HTTPS connection can't be verified. If you are using localtunnel as advised above, this shouldn't be an issue. But once you deploy your slash command, you should be certain that your hosting service provides a valid HTTPS connection — but we can worry about this in a later tutorial.)

Slack gives us a nice little preview of what our slash command's auto-complete message will look like:



Once we've hit "Save", Slack will issue us a Verification Token. This is super important, as it is a secret handshake that Slack shares with us so we know that a given slash command was *actually* issued by Slack, and not by some imposter. Keep it safe, keep it secret — and make sure your slash command knows about it and is checking it. Slack issues one token per app, so if you have multiple slash commands, they will all use the same verification token.

This verification token can be seen at any time by going to the “Basic Info” tab on your app’s page.



**Running your slash command**

We've got Slack all set up — now to look at the code, and make a slash command worthy of the name. The template code is pretty contrived, but provides all the basic tools you'll need to make a slash command work.

**Running it locally**

Now, the time has come to run your bot. From the directory your bot is installed in, simply run

CLIENT\_ID=xxx.yyy CLIENT\_SECRET=abc VERIFICATION\_TOKEN=123 PORT=8765 npm start

**\*\*Your client ID, client secret, and verification number will come from the ‘Basic Information’ tab of your app’s page. You should run this command at the same time that you are hosting the app (See just under ‘Exposing your app to the outside world’ if you think you may not be hosting the app).**

And then visit <https://awesomeslashcommand.localtunnel.me/login> to install your bot on a team.

If you are successful, a blank page with the word ‘Success!’ will be returned from this url, meaning that your slash command and app have been installed on your team’s page, and your copy of the app is configured to expect your team.

Don't forget to restart the slash command when you make changes to the source code!