# Building a CLI with a GitHub App

**In this article**



Follow this tutorial to write a CLI in Ruby that generates a user access token for a GitHub App via the device flow.

## Introduction

This tutorial demonstrates how to build a command line interface (CLI) backed by a GitHub App, and how to use the device flow to generate a user access token for the app.

The CLI will have three commands:

- `help`: Outputs the usage instructions.
- `login`: Generates a user access token that the app can use to make API requests on behalf of the user.
- `whoami`: Returns information about the logged in user.

This tutorial uses Ruby, but you can write a CLI and use the device flow to generate a user access token with any programming language.

### About device flow and user access tokens

The CLI will use the device flow to authenticate a user and generate a user access token. Then, the CLI can use the user access token to make API requests on behalf of the authenticated user.

Your app should use a user access token if you want to attribute the app's actions to a user. For more information, see "Authenticating with a GitHub App on behalf of a user."

There are two ways to generate a user access token for a GitHub App: web application flow and device flow. You should use the device flow to generate a user access token if your app is headless or does not have access to a web interface. For example, CLI tools, simple Raspberry Pis, and desktop applications should use the device flow. If your app has access to a web interface, you should use web application flow instead. For more information, see "Generating a user access token for a GitHub App" and "Building a "Login with GitHub" button with a GitHub App."

## Prerequisites

This tutorial assumes that you have already registered a GitHub App. For more

information about registering a GitHub App, see "Registering a GitHub App."

Before following this tutorial, you must enable device flow for your app. For more information about enabling device flow for your app, see "Modifying a GitHub App registration."

This tutorial assumes that you have a basic understanding of Ruby. For more information, see Ruby.

## Get the client ID

You will need your app's client ID in order to generate a user access token via the device flow.

1. In the upper-right corner of any page on GitHub, click your profile photo.
2. Navigate to your account settings.
   - For a GitHub App owned by a personal account, click **Settings**.
   - For a GitHub App owned by an organization:
     a. Click **Your organizations**.
     b. To the right of the organization, click **Settings**.
3. In the left sidebar, click <> **Developer settings**.
4. In the left sidebar, click **GitHub Apps**.
5. Next to the GitHub App that you want to work with, click **Edit**.
6. On the app's settings page, find the client ID for your app. You will use it later in this tutorial. Note that the client ID is different from the app ID.

## Write the CLI

These steps lead you through building a CLI and using device flow to get a user access token. To skip ahead to the final code, see "Full code example."

### Setup

1. Create a Ruby file to hold the code that will generate a user access token. This tutorial will name the file `app_cli.rb`.
2. In your terminal, from the directory where `app_cli.rb` is stored, run the following command to make `app_cli.rb` executable:

```
Text
chmod +x app_cli.rb
```

3. Add this line to the top of `app_cli.rb` to indicate that the Ruby interpreter should be used to run the script:

```
Ruby
```

Ruby

```
#!/usr/bin/env ruby
```

4. Add these dependencies to the top of `app_cli.rb`, following `#!/usr/bin/env ruby`:

Ruby

```ruby
require "net/http"
require "json"
require "uri"
require "fileutils"
```

These are all part of the Ruby standard library, so you don't need to install any gems.

5. Add the following `main` function that will serve as an entry point. The function includes a `case` statement to take different actions depending on which command is specified. You will expand this `case` statement later.

Ruby

```ruby
def main
  case ARGV[0]
  when "help"
    puts "`help` is not yet defined"
  when "login"
    puts "`login` is not yet defined"
  when "whoami"
    puts "`whoami` is not yet defined"
  else
    puts "Unknown command `#{ARGV[0]}`"
  end
end
```

6. At the bottom of the file, add the following line to call the entry point function. This function call should remain at the bottom of your file as you add more functions to this file later in the tutorial.

Ruby

```ruby
main
```

7. Optionally, check your progress:

`app_cli.rb` now looks like this:

Ruby

```ruby
#!/usr/bin/env ruby

require "net/http"
require "json"
require "uri"
require "fileutils"

def main
  case ARGV[0]
```

```ruby
  when "help"
    puts "`help` is not yet defined"
  when "login"
    puts "`login` is not yet defined"
  when "whoami"
    puts "`whoami` is not yet defined"
  else
    puts "Unknown command `#{ARGV[0]}`"
  end
end

main
```

In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb help`. You should see this output:

```
`help` is not yet defined
```

You can also test your script without a command or with an unhandled command. For example, `./app_cli.rb create-issue` should output:

```
Unknown command `create-issue`
```

## Add a `help` command

1. Add the following `help` function to `app_cli.rb`. Currently, the `help` function prints a line to tell users that this CLI takes one command, "help". You will expand this `help` function later.

```ruby
def help
  puts "usage: app_cli <help>"
end
```

2. Update the `main` function to call the `help` function when the `help` command is given:

```ruby
def main
  case ARGV[0]
  when "help"
    help
  when "login"
    puts "`login` is not yet defined"
  when "whoami"
    puts "`whoami` is not yet defined"
  else
    puts "Unknown command #{ARGV[0]}"
  end
end
```

3. Optionally, check your progress:

   `app_cli.rb` now looks like this. The order of the functions don't matter as long as the `main` function call is at the end of the file.

Ruby

```ruby
#!/usr/bin/env ruby

require "net/http"
require "json"
require "uri"
require "fileutils"

def help
  puts "usage: app_cli <help>"
end

def main
  case ARGV[0]
  when "help"
    help
  when "login"
    puts "`login` is not yet defined"
  when "whoami"
    puts "`whoami` is not yet defined"
  else
    puts "Unknown command #{ARGV[0]}"
  end
end

main
```

In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb help`. You should see this output:

```
usage: app_cli <help>
```

## Add a `login` command

The `login` command will run the device flow to get a user access token. For more information, see "Generating a user access token for a GitHub App."

1. Near the top of your file, after the `require` statements, add the `CLIENT_ID` of your GitHub App as a constant in `app_cli.rb`. For more information about finding your app's client ID, see "Get the client ID." Replace `YOUR_CLIENT_ID` with the client ID of your app:

Ruby

```ruby
CLIENT_ID="YOUR_CLIENT_ID"
```

2. Add the following `parse_response` function to `app_cli.rb`. This function parses a response from the GitHub REST API. When the response status is `200 OK` or `201 Created`, the function returns the parsed response body. Otherwise, the function prints the response and body an exits the program.

Ruby

```ruby
def parse_response(response)
  case response
  when Net::HTTPOK, Net::HTTPCreated
    JSON.parse(response.body)
```

```ruby
  else
    puts response
    puts response.body
    exit 1
  end
end
```

3. Add the following `request_device_code` function to `app_cli.rb`. This function makes a `POST` request to `http(s)://HOSTNAME/login/device/code` and returns the response.

Ruby

```ruby
def request_device_code
  uri = URI("http(s)://HOSTNAME/login/device/code")
  parameters = URI.encode_www_form("client_id" => CLIENT_ID)
  headers = {"Accept" => "application/json"}

  response = Net::HTTP.post(uri, parameters, headers)
  parse_response(response)
end
```

4. Add the following `request_token` function to `app_cli.rb`. This function makes a `POST` request to `http(s)://HOSTNAME/login/oauth/access_token` and returns the response.

Ruby

```ruby
def request_token(device_code)
  uri = URI("http(s)://HOSTNAME/login/oauth/access_token")
  parameters = URI.encode_www_form({
    "client_id" => CLIENT_ID,
    "device_code" => device_code,
    "grant_type" => "urn:ietf:params:oauth:grant-type:device_code"
  })
  headers = {"Accept" => "application/json"}
  response = Net::HTTP.post(uri, parameters, headers)
  parse_response(response)
end
```

5. Add the following `poll_for_token` function to `app_cli.rb`. This function polls `http(s)://HOSTNAME/login/oauth/access_token` at the specified interval until GitHub responds with an `access_token` parameter instead of an `error` parameter. Then, it writes the user access token to a file and restricts the permissions on the file.

Ruby

```ruby
def poll_for_token(device_code, interval)

  loop do
    response = request_token(device_code)
    error, access_token = response.values_at("error", "access_token")

    if error
      case error
      when "authorization_pending"
        # The user has not yet entered the code.
        # Wait, then poll again.
        sleep interval
        next
      when "slow_down"
```

```
        # The app polled too fast.
        # Wait for the interval plus 5 seconds, then poll again.
        sleep interval + 5
        next
      when "expired_token"
        # The `device_code` expired, and the process needs to restart.
        puts "The device code has expired. Please run `login` again."
        exit 1
      when "access_denied"
        # The user cancelled the process. Stop polling.
        puts "Login cancelled by user."
        exit 1
      else
        puts response
        exit 1
      end
    end

    File.write("./.token", access_token)

    # Set the file permissions so that only the file owner can read or
modify the file
    FileUtils.chmod(0600, "./.token")

    break
  end
end
```

6. Add the following `login` function.

   This function:

   a. Calls the `request_device_code` function and gets the `verification_uri`, `user_code`, `device_code`, and `interval` parameters from the response.

   b. Prompts users to enter the `user_code` from the previous step.

   c. Calls the `poll_for_token` to poll GitHub for an access token.

   d. Lets the user know that authentication was successful.

Ruby

```
def login
  verification_uri, user_code, device_code, interval =
request_device_code.values_at("verification_uri", "user_code",
"device_code", "interval")

  puts "Please visit: #{verification_uri}"
  puts "and enter code: #{user_code}"

  poll_for_token(device_code, interval)

  puts "Successfully authenticated!"
end
```

7. Update the `main` function to call the `login` function when the `login` command is given:

Ruby

```
def main
```

```ruby
  case ARGV[0]
  when "help"
    help
  when "login"
    login
  when "whoami"
    puts "`whoami` is not yet defined"
  else
    puts "Unknown command #{ARGV[0]}"
  end
end
```

8. Update the `help` function to include the `login` command:

```ruby
def help
  puts "usage: app_cli <login | help>"
end
```

9. Optionally, check your progress:

`app_cli.rb` now looks something like this, where `YOUR_CLIENT_ID` is the client ID of your app. The order of the functions don't matter as long as the `main` function call is at the end of the file.

```ruby
#!/usr/bin/env ruby

require "net/http"
require "json"
require "uri"
require "fileutils"

CLIENT_ID="YOUR_CLIENT_ID"

def help
  puts "usage: app_cli <login | help>"
end

def main
  case ARGV[0]
  when "help"
    help
  when "login"
    login
  when "whoami"
    puts "`whoami` is not yet defined"
  else
    puts "Unknown command #{ARGV[0]}"
  end
end

def parse_response(response)
  case response
  when Net::HTTPOK, Net::HTTPCreated
    JSON.parse(response.body)
  else
    puts response
    puts response.body
    exit 1
  end
end
```

```
def request_device_code
  uri = URI("http(s)://HOSTNAME/login/device/code")
  parameters = URI.encode_www_form("client_id" => CLIENT_ID)
  headers = {"Accept" => "application/json"}

  response = Net::HTTP.post(uri, parameters, headers)
  parse_response(response)
end

def request_token(device_code)
  uri = URI("http(s)://HOSTNAME/login/oauth/access_token")
  parameters = URI.encode_www_form({
    "client_id" => CLIENT_ID,
    "device_code" => device_code,
    "grant_type" => "urn:ietf:params:oauth:grant-type:device_code"
  })
  headers = {"Accept" => "application/json"}
  response = Net::HTTP.post(uri, parameters, headers)
  parse_response(response)
end

def poll_for_token(device_code, interval)

  loop do
    response = request_token(device_code)
    error, access_token = response.values_at("error", "access_token")

    if error
      case error
      when "authorization_pending"
        # The user has not yet entered the code.
        # Wait, then poll again.
        sleep interval
        next
      when "slow_down"
        # The app polled too fast.
        # Wait for the interval plus 5 seconds, then poll again.
        sleep interval + 5
        next
      when "expired_token"
        # The `device_code` expired, and the process needs to restart.
        puts "The device code has expired. Please run `login` again."
        exit 1
      when "access_denied"
        # The user cancelled the process. Stop polling.
        puts "Login cancelled by user."
        exit 1
      else
        puts response
        exit 1
      end
    end

    File.write("./.token", access_token)

    # Set the file permissions so that only the file owner can read or
modify the file
    FileUtils.chmod(0600, "./.token")

    break
  end
end

def login
  verification_uri, user_code, device_code, interval =
request_device_code.values_at("verification_uri", "user_code",
"device_code", "interval")

  puts "Please visit: #{verification_uri}"
  puts "and enter code: #{user_code}"
```

```
  poll_for_token(device_code, interval)

  puts "Successfully authenticated!"
end

main
```

a. In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb login`. You should see output that looks like this. The code will differ every time:

   ```
   Please visit: http(s)://HOSTNAME/login/device
   and enter code: CA86-8D94
   ```

b. Navigate to http(s)://HOSTNAME/login/device in your browser and enter the code from the previous step, then click **Continue**.

c. GitHub should display a page that prompts you to authorize your app. Click the "Authorize" button.

d. Your terminal should now say "Successfully authenticated!".

## Add a `whoami` command

Now that your app can generate a user access token, you can make API requests on behalf of the user. Add a `whoami` command to get the username of the authenticated user.

1. Add the following `whoami` function to `app_cli.rb`. This function gets information about the user with the `/user` REST API endpoint. It outputs the username that corresponds to the user access token. If the `.token` file was not found, it prompts the user to run the `login` function.

   Ruby

   ```ruby
   def whoami
     uri = URI("http(s)://HOSTNAME/api/v3/user")

     begin
       token = File.read("./.token").strip
     rescue Errno::ENOENT => e
       puts "You are not authorized. Run the `login` command."
       exit 1
     end

     response = Net::HTTP.start(uri.host, uri.port, use_ssl: true) do |http|
       body = {"access_token" => token}.to_json
       headers = {"Accept" => "application/vnd.github+json", "Authorization" =>
   "Bearer #{token}"}

       http.send_request("GET", uri.path, body, headers)
     end

     parsed_response = parse_response(response)
     puts "You are #{parsed_response["login"]}"
   end
   ```

2. Update the `parse_response` function to handle the case where the token has expired or been revoked. Now, if you get a `401 Unauthorized` response, the CLI will prompt the user to run the `login` command.

```ruby
def parse_response(response)
  case response
  when Net::HTTPOK, Net::HTTPCreated
    JSON.parse(response.body)
  when Net::HTTPUnauthorized
    puts "You are not authorized. Run the `login` command."
    exit 1
  else
    puts response
    puts response.body
    exit 1
  end
end
```

3. Update the `main` function to call the `whoami` function when the `whoami` command is given:

```ruby
def main
  case ARGV[0]
  when "help"
    help
  when "login"
    login
  when "whoami"
    whoami
  else
    puts "Unknown command #{ARGV[0]}"
  end
end
```

4. Update the `help` function to include the `whoami` command:

```ruby
def help
  puts "usage: app_cli <login | whoami | help>"
end
```

5. Check your code against the full code example in the next section. You can test your code by following the steps outlined in the "Testing" section below the full code example.

## Full code example

This is the full code example that was outlined in the previous section. Replace `YOUR_CLIENT_ID` with the client ID of your app.

```ruby
```

```ruby
#!/usr/bin/env ruby

require "net/http"
require "json"
require "uri"
require "fileutils"

CLIENT_ID="YOUR_CLIENT_ID"

def help
  puts "usage: app_cli <login | whoami | help>"
end

def main
  case ARGV[0]
  when "help"
    help
  when "login"
    login
  when "whoami"
    whoami
  else
    puts "Unknown command #{ARGV[0]}"
  end
end

def parse_response(response)
  case response
  when Net::HTTPOK, Net::HTTPCreated
    JSON.parse(response.body)
  when Net::HTTPUnauthorized
    puts "You are not authorized. Run the `login` command."
    exit 1
  else
    puts response
    puts response.body
    exit 1
  end
end

def request_device_code
  uri = URI("http(s)://HOSTNAME/login/device/code")
  parameters = URI.encode_www_form("client_id" => CLIENT_ID)
  headers = {"Accept" => "application/json"}

  response = Net::HTTP.post(uri, parameters, headers)
  parse_response(response)
end

def request_token(device_code)
  uri = URI("http(s)://HOSTNAME/login/oauth/access_token")
  parameters = URI.encode_www_form({
    "client_id" => CLIENT_ID,
    "device_code" => device_code,
    "grant_type" => "urn:ietf:params:oauth:grant-type:device_code"
  })
  headers = {"Accept" => "application/json"}
  response = Net::HTTP.post(uri, parameters, headers)
  parse_response(response)
end

def poll_for_token(device_code, interval)

  loop do
    response = request_token(device_code)
    error, access_token = response.values_at("error", "access_token")

    if error
      case error
      when "authorization_pending"
```

```
        # The user has not yet entered the code.
        # Wait, then poll again.
        sleep interval
        next
      when "slow_down"
        # The app polled too fast.
        # Wait for the interval plus 5 seconds, then poll again.
        sleep interval + 5
        next
      when "expired_token"
        # The `device_code` expired, and the process needs to restart.
        puts "The device code has expired. Please run `login` again."
        exit 1
      when "access_denied"
        # The user cancelled the process. Stop polling.
        puts "Login cancelled by user."
        exit 1
      else
        puts response
        exit 1
      end
    end

    File.write("./.token", access_token)

    # Set the file permissions so that only the file owner can read or modify the
file
    FileUtils.chmod(0600, "./.token")

    break
  end
end

def login
  verification_uri, user_code, device_code, interval =
request_device_code.values_at("verification_uri", "user_code", "device_code",
"interval")

  puts "Please visit: #{verification_uri}"
  puts "and enter code: #{user_code}"

  poll_for_token(device_code, interval)

  puts "Successfully authenticated!"
end

def whoami
  uri = URI("http(s)://HOSTNAME/api/v3/user")

  begin
    token = File.read("./.token").strip
  rescue Errno::ENOENT => e
    puts "You are not authorized. Run the `login` command."
    exit 1
  end

  response = Net::HTTP.start(uri.host, uri.port, use_ssl: true) do |http|
    body = {"access_token" => token}.to_json
    headers = {"Accept" => "application/vnd.github+json", "Authorization" =>
"Bearer #{token}"}

    http.send_request("GET", uri.path, body, headers)
  end

  parsed_response = parse_response(response)
  puts "You are #{parsed_response["login"]}"
end

main
```

## Testing

This tutorial assumes that your app code is stored in a file named `app_cli.rb`.

1. In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb help`. You should see output that looks like this.

   ```
   usage: app_cli <login | whoami | help>
   ```

2. In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb login`. You should see output that looks like this. The code will differ every time:

   ```
   Please visit: http(s)://HOSTNAME/login/device
   and enter code: CA86-8D94
   ```

3. Navigate to http(s)://HOSTNAME/login/device in your browser and enter the code from the previous step, then click **Continue**.
4. GitHub should display a page that prompts you to authorize your app. Click the "Authorize" button.
5. Your terminal should now say "Successfully authenticated!".
6. In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb whoami`. You should see output that looks like this, where `octocat` is your username.

   ```
   You are octocat
   ```

7. Open the `.token` file in your editor, and modify the token. Now, the token is invalid.
8. In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb whoami`. You should see output that looks like this:

   ```
   You are not authorized. Run the `login` command.
   ```

9. Delete the `.token` file.
10. In your terminal, from the directory where `app_cli.rb` is stored, run `./app_cli.rb whoami`. You should see output that looks like this:

    ```
    You are not authorized. Run the `login` command.
    ```

## Next steps

### Adjust the code to meet your app's needs

This tutorial demonstrated how to write a CLI that uses the device flow to generate a user access token. You can expand this CLI to accept additional commands. For example, you can add a `create-issue` command that opens an issue. Remember to update your app's permissions if your app needs additional permissions for the API requests that you

want to make. For more information, see "Choosing permissions for a GitHub App."

## Securely store tokens

This tutorial generates a user access token and saves it in a local file. You should never commit this file or publicize the token.

Depending on your device, you may choose different way to store the token. You should check the best practices for storing tokens on your device.

For more information, see "Best practices for creating a GitHub App."

## Follow best practices

You should aim to follow best practices with your GitHub App. For more information, see "Best practices for creating a GitHub App."