Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Extendiendo GitHub Cli para Gestionar Organizaciones GitHub

Una Estrategia para la Administración de Grandes Números de Proyectos Relacionados

Extending GitHub Cli to Manage GitHub Organizations

A Strategy for the Administration of Large Numbers of Related Projects

José Ramón Rodríguez Hernández

La Laguna, 8 de septiembre de 2021

D. **Casiano Rodríguez León**, con N.I.F. 42020072S profesor Catedrático de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A

Que la presente memoria titulada:

"*Extendiendo GitHub Cli para Gestionar Organizaciones GitHub*"

ha sido realizada bajo su dirección por D. **José Ramón Rodríguez Hernández**, con N.I.F. 79091897A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 8 de septiembre de 2021

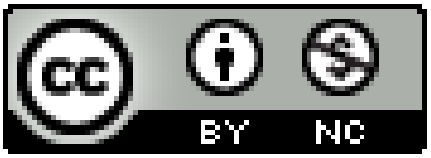
Agradecimientos

En primer lugar deseo expresar mi especial agradecimiento al tutor de este trabajo de fin de grado, Casiano Rodríguez León, por la dedicación y apoyo brindados. Gracias a su confianza, dirección y plétora de recursos es que este proyecto ha sido posible.

Asimismo, agradezco a mis compañeros de la facultad su apoyo personal y humano, en el trayecto de esta etapa maravillosa llena de obstáculos, superación y conocimiento.

Gracias a mi familia, su confianza y apoyo incondicional en cada momento.

Licencia



Resumen

*En este proyecto se realiza una extensión a GitHub Cli en **npm**. Específicamente se busca dar soporte a las organizaciones dentro de GitHub, pues a casusa de la corta vida de la herramienta tiene funciones básicas orientadas al uduario medio, pero contiene un gran potencial al facilitar mediante **alias** añadir extensiones como scripts para que funcionen armónicamente.*

Palabras clave: GitHub Cli, gh, npm.

Abstract

*In this project an extension to GitHub Cli is made in **npm**. Specifically it seeks to support organizations within GitHub, as the tool, being relatively new, only covers basic functionalities oriented to the average user, but contains great potential by facilitating through **alias** the adding of extensions as scripts so that they can work harmoniously.*

Keywords: GitHub Cli, gh, npm.

Índice general

Capítulo 1

Introducción

1.1. Objetivos

El Objetivo de este trabajo consiste en el análisis de las funcionalidades alrededor de ***GitHub Cli*****[1]**, también conocido como ***gh***, con el motivo de realizar una extensión dirigida a las organizaciones dentro de GitHub. Esto se debe a que los comandos incorporados están planteados para un usuario como individuo, probablemente a causa del poco tiempo de vida de *gh*, tomando como referencia los comandos publicados en el manual de GitHub Cli[2]. Para realizar la extensión se escogerá ***Node.Js*****[3]** ya que facilita la publicación del módulo de forma pública a través de ***npm*****[4]**, nombrada como ***gh-plugin-org*****[5]** que a efectos prácticos se mencionará como ***gh-org***.

1.2. Antecedentes y Estado Actual del Arte

Anteriormente *GitHub* había intentado desarrollar una herramienta, alrededor del año 2009, para encapsular la funcionalidad de su plataforma web dentro de la línea de comandos. Esta herramienta fue conocida como *hub*[6], y su última actualización fue lanzada el 5 de marzo de 2020, dos meses después del lanzamiento de *GitHub Cli* como una nueva versión de línea de comandos. La primera versión datada en *Github* fue la 0.4.0, en enero de 2020. Aunque no fue hasta el 17 de septiembre del mismo año que salió la versión 1.0.0, primera versión fuera de la beta. *GitHub Cli* permitió desde ese momento realizar gran parte de las funcionalidades que ofrece GitHub en la aplicación web, al mismo tiempo que cubre las necesidades básicas de un usuario medio como realizar un *pull request*, manejar un repositorio o un *issue*, aunque toda la funcionalidad referente a las organizaciones solo se encuentra disponible a través de peticiones a la API.

1.3. Metodología de Desarrollo

Antes que nada hay que hablar de los componentes de una organización dentro de *GitHub*. Los repositorios, siendo el principal recurso para todo proyecto, es un punto de almacenamiento del código fuente. Adicionalmente permite crear ramas, donde se generan diferentes puntos de desarrollo (producción, desarrollo, versiones, etc). Por otro lado, los ***"issues"*****[7]**, componen un lugar para que los usuarios sugieran cualquier tipo de cambio, como pueden ser de funcionalidad, documentación o errores, a la vez que puedan resolverse dudas sobre el proyecto. Además están los ***"teams"*****[8]**, ayudando a reflejar

una estructura dentro de la organización, al mismo tiempo que posibilitan el control de acceso y menciones más generales dentro de los *issues*, de los que cada equipo tiene su propia página. Así mismo los miembros y colaboradores, quiénes están involucrados tanto de manera externa como interna, son relevantes para realizar cualquier acción o mención dentro de la organización. Por último el ***"project board"*** **[9]**, la herramienta de administración de proyectos basada en ***Kanban*** **[10]**. Éste es un Sistema de tarjetas representativo de los elementos de trabajo para el desarrollo de un software, donde un individuo puede tomar un elemento para su elaboración y esto ser visible para toda la organización, dando de esta manera visualización al flujo de elaboración y eficacia en la realización del trabajo conjunto, además de evitar la sobrecarga del personal. Por tanto, al estar este sistema integrado en GitHub, permite trabajar con todos estos componentes mencionados en armonía, siendo así un componente de gran valor para cualquier solución y la metodología escogida para este desarrollo. Aunque existen herramientas similares como ***Pivotal Tracker*** o ***Trello*** que no trabajan implícitamente con *GitHub*.

1.4. Incidencias

A lo largo del desarrollo se han hallado diversos obstáculos, la búsqueda y manejo de las tecnologías empleadas como *Commander*, *Shelljs*, *Octokit* o *git submodules*, explicados en mayor detalle más adelante. La necesidad de realizar pruebas en diferentes equipos, con el fin de revisar los detalles de la instalación del módulo ***npm*** desarrollado. Por otro lado las peticiones a la API, se realizaron inicialmente con ***gh api***, dando resultados muy buenos en una organización pequeña, hablando de una que contenga entre uno a veinte repositorios, pero dando diferentes errores cuando la organización llega a tener más de treinta repositorios o superior a cien. Estos errores son causados por como gestiona GitHub las peticiones, con la finalidad de evitar la saturación en los servidores, cualquier petición tiene un límite establecido por defecto de treinta (sin importar el tipo de información solicitada: *issues*, repositorios, etc), aunque por parámetros se puede llegar a un límite de cien. Sabiendo esto, entra el incidente de la paginación de resultados sobre la API de GitHub[11], inicialmente se intentó abarcar desde la opción aportada por *GitHub CLi*, con *"gh api* ***–paginate****"*, siendo una vía con una curva de aprendizaje bastante alta, ya que utiliza ***GraphQL***[12] de forma nativa para filtrar la información. Debido al tiempo disponible para el desarrollo, se optó por ***Octokit*** junto con la extensión ***plugin-paginate-rest.js*** **[13]**, puesto que se evita realizar consultas complejas al utilizar una simple ruta para acceder a la información. Por otra parte la autenticación del usuario al emplear los propios comandos de *GitHub Cli*, se le pasaba el trabajo ya que requiere credenciales para utilizar la herramienta. Pero al usar *octokit* se tiene que autenticar con nuestra extensión y por tanto se necesita usar el *Personal Access Token[14]*, una clave generada por *GitHub* para demostrar la autenticidad del usuario desde un dispositivo. A causa de esto para obtener el *token* que guarda *GitHub CLi*, se debe tener en cuenta: el sistema operativo, el directorio personal y que *GitHub Cli* esté instalado en el directorio por defecto.

Capítulo 2

Tecnologías Empleadas

2.1. Control de Versiones

En todo proyecto es relevante gestionar los cambios que se van realizando a medida que avanza el desarrollo, por lo que se aconseja mantener un control de versiones, en este caso se llevó a cabo a través del versionado semántico, basado en tres números separados por un punto(X.Y.Z) que indican el mayor, menor y parche, respectivamente. Un cambio **mayor** representa que tiene incompatibilidad con versiones anteriores. El **menor** indica que solo se añade funcionalidad compatible con lo anterior y por último, el **parche** destinado a la reparación de errores manteniendo la compatibilidad.

2.2. Git

Sistema de control de versiones más utilizado, puesto que se basa en un sistema distribuido. Esto significa que cada usuario contiene una réplica del repositorio, sin depender de tener conexión estable en dónde se aloja el código fuente. Esto sumado a la implementación del desarrollo a través de las ramas, permite dar rapidez al poder trabajar con diferentes versiones simultáneamente. Así mismo, la metodología de mantener tres estados posibles para los ficheros: confirmado(*committed)*, modificado*(modified)* y preparado*(staged)*, lleva el flujo de trabajo a una mayor velocidad sin perder información. Véase ***Git*** **[15]**.

2.2.1. Git Submodules

Herramienta aportada por *git*[16], donde permite mantener un repositorio de *git* como un subdirectorio dentro de otro repositorio de *git*. Esto evita problemas de implementación y de instalación al incluir toda la información directamente.

2.3. Github

Plataforma colaborativa de proyectos destinada a guardar el código de las aplicaciones, dando como funcionalidades adicionales la descarga del proyecto, colaborar en el desarrollo, mediante el sistema de *issues* en el que cualquier usuario puede sugerir cualquier tipo de cambio. Además de que trabaja con el sistema de control de versiones, ***Git***, de forma nativa.

2.3.1. GitHub Cli

Línea de comandos de GitHub que ofrece las funcionalidades básicas para un usuario como crear repositorios, *issues* o *releases*. Además permite el manejo de la API y de alias, dejando un punto abierto para el desarrollo de cualquier tipo de *script* o *plugin* realizado por la comunidad.

2.3.2. GitHub API

Antes de explicar que es la API de GitHub se debe explicar que es una ***API (Application Programming Interfaces)***. Se define como una serie de protocolos y definiciones que habilitan la comunicación de servicios sin que el usuario necesite ver la implementación del mismo, sino como acceder al servicio y la respuesta del mismo.

Una vez explicado esto, ***GitHub API***[17] es una herramienta que nos facilita obtener información de su plataforma, como solicitar o modificar información de los repositorios, organizaciones, etc. Cabe destacar que para realizar cualquier tipo de petición, requiere tener permisos sobre dicho usuario, repositorio u organización.

2.4. JavaScript

Uno de los lenguajes más conocidos y buscados a día de hoy. Gracias a la asincronía, mejoran considerablemente la respuesta a tiempo real y la versatilidad en el diseño de cualquier software. Particularmente, una elección favorable por ***Node.js***[3] que facilita trabajar con peticiones a un servidor, aunque explicado con más detalle en el siguiente apartado.

2.5. Node.js

Uno de los frameworks de JavaScript más conocidos, especializado para el desarrollo en el *"back-end"*. Está basado en una arquitectura orientada a eventos para facilitar la escalabilidad en el desarrollo. Para esto utiliza un único hilo de forma asíncrona que admite la concurrencia de cientos de miles de peticiones, en el que cada vez que se bloquee un evento para resolverse en la E/S(operaciones de entrada y salida), no espera sino que pasa al siguiente y si cuando termina con este ha finalizado el anterior, lo continúa. Esto evita sobrecarga al no generar constantemente subprocesos para cada evento, convirtiéndose en una herramienta liviana y eficiente. Véase para más información: *"Qué es NodeJS y para qué sirve"*[18], *" Qué es NodeJS y primeros pasos"*[19] o *"What exactly is Node.js?"*[20].

2.6. NPM

Podemos diferenciar dos partes, siendo la primera, ***NPM(Node Package Manager)*****[4]** un comando ejecutable para facilitar la gestión de los paquetes de *Node.js* en nuestro equipo o proyecto, tanto para instalar, actualizar o eliminar del proyecto los módulos que deseamos. En segundo lugar, un manejador de paquetes de *Node.js* público, donde la comunidad puede crear módulos para diferentes funcionalidades y cualquiera puede utilizar en sus proyectos con la susodicha herramienta mencionada anteriormente.

Para hablar con mayor fluidez en la implementación de la extensión, se tienen que mencionar los siguientes módulos npm utilizados.

2.6.1. Commander

Con el objetivo de encapsular la funcionalidad en un comando utilizamos *commander*, puesto que facilita el manejo de los argumentos obtenidos a través de la línea de comandos y la implementación tanto de subcomandos como de opciones de ejecución(*"flags"*). Véase ***Commander*[21]**.

2.6.2. Shelljs

La ejecución de comandos nativos en terminal son necesarios para trabajar con ***git***, por tanto ***shelljs*** es una herramienta de gran valor. No solo permite ejecutar comandos, sino que puedes controlar la salida de los mismos en una variable para su gestión. Véase ***Shelljs*[22]**.

2.6.3. Octokit

Módulo oficial de ***GitHub*** que nos permite realizar la autenticación de usuario y peticiones a su API. Adicionalmente para facilitar las llamadas a la API que se realizan en base a ***GraphQL*[12]**, llamadas en el formato de consultas a bases de datos, se decidió utilizar ***plugin-paginate-rest.js*** que realiza peticiones orientadas al formato de la API, y que facilitan el manejo de la respuesta cuando se requiere paginación. Véase ***Octokit***[23] y ***plugin-paginate-rest.js***[13].

2.6.4. js-yaml

A causa de que en JavaScript y Node.Js no se pueden utilizar de forma nativa archivos en formato ***yml***, se utiliza ***js-yaml***[24].

2.6.5. untildify

Para facilitar la extracción en cualquier dispositivo y sistema operativo de las ruta de usuario, específicamente por el caracter especial ***"~"***. Véase ***untildify*[25]**.

Capítulo 3

Diseño e Implementación de la Solución

La solución desarrollada, ***gh-plugin-org***[5], es un módulo npm como extensión para ***GitHub Cli[1]*** que da soporte a las organizaciones, tanto de recopilación como de creación, a través de la API.

3.1. Diseño

Para poder hablar del diseño, se debe hacer una introducción sobre la integración de ***gh-org*** como extensión para *GitHub Cli*, teniendo en cuenta que es un módulo npm por lo que se debe instalar como tal(***"npm i -g gh-plugin-org"***). Una vez instalado, se puede realizar de dos formas la ejecución del mismo: Primero como una ejecución del módulo npm, en el que directamente podemos utilizar el gestor de paquetes con ***"npx gh-org [opciones] [comandos]"***. En segundo lugar como un alias de ***gh***, en el que se realiza a través de***"gh alias set –shell org 'gh-org $@' "***, en el que se agrega a *gh* un comando ***org***, al que cualquier argumento lo interpreta nuestro ***gh-org***.

Una vez aclarado esto el flujo de comunicación sigue unos mismos pasos de transición como se puede ver en la figura 3.1, en el que se solicita información a la API de *GitHub* y la respuesta es tratada por *gh-org*. Siendo así las peticiones pueden tomar dos caminos, primero a través de "*gh api*", donde recordar que es un comando integrado en *GitHub CLi* es relevante, ya que la gestión de las credenciales lo realiza de forma previa mediante "*gh auth*". En el segundo lugar es con *octokit*, donde se obtiene el *token* del usuario y posteriormente se llama al cliente de *octokit*, una vez validado solicita la información a la API. Finalmente la respuesta en ambos casos al adecuar la información, se muestra al usuario.

3.2. Análisis de requisitos

El punto de partida es encontrar la información asociada en *gh*. Al respecto se hallaron únicamente explicaciones sobre los *"repo"*[26] y los *"issue"*[27], por lo que se obtuvo soporte para las funcionalidades básicas de creación y clonado de repositorios. Al mismo tiempo sobre *"gh api"*[28], adquiriendo conocimiento relevante para el desarrollo de soluciones. A continuación, se realiza una comparativa de los componentes necesarios para una organización, comentados anteriormente, y las herramientas encontradas en *gh*. Dando lugar a los siguientes requisitos posibles:

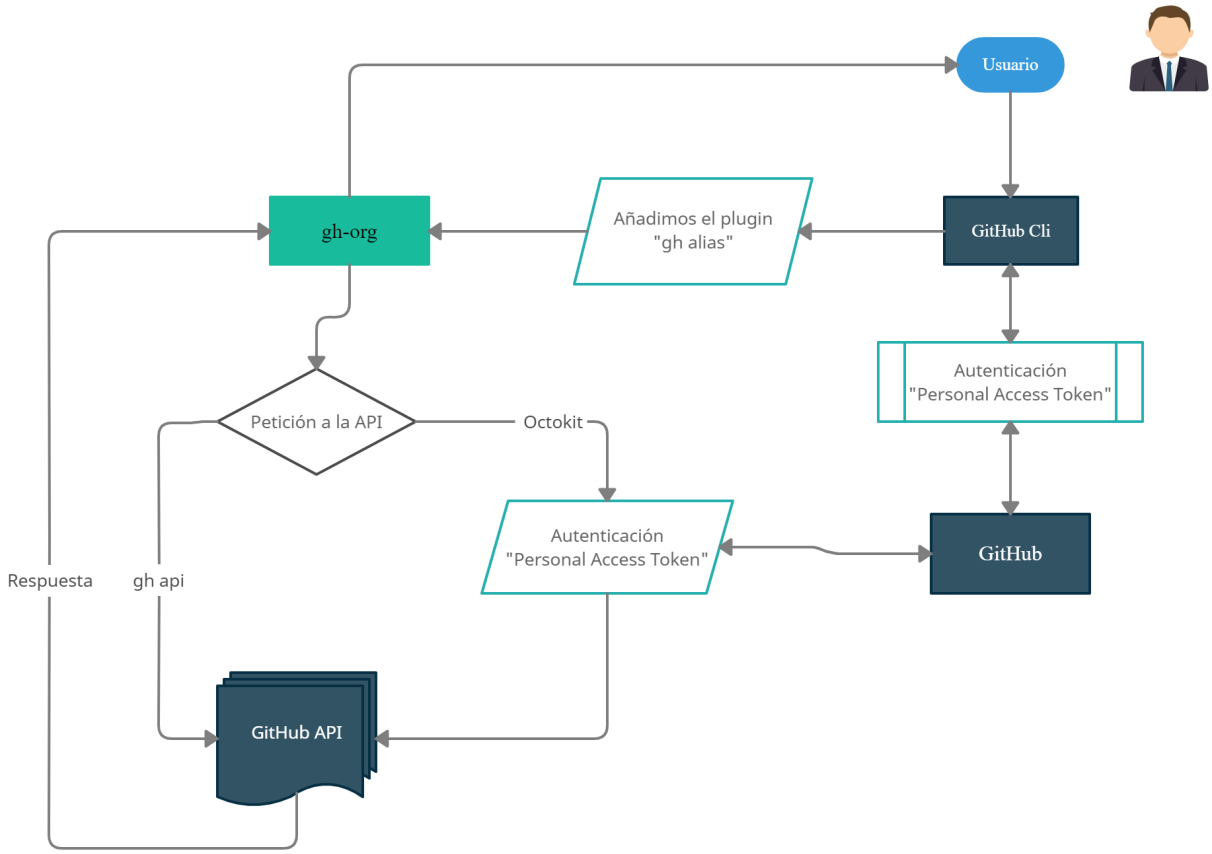


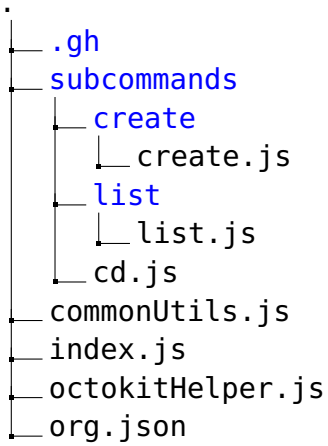
Figura 3.1: Comunicación entre la API y gh-org

1. Información a obtener de una organización:
   - Los repositorios que existen.
   - Los miembros y colaboradores externos.
   - Los equipos o la estructura de los mismos.
   - Información sobre el *“proyect board”*.
2. Creación de diferentes funcionalidades para la organización:
   - Repositorios dentro de una organización.
   - Equipos y los permisos correspondientes.
   - Invitaciones a nuevos miembros.
3. Clonado de los repositorios de una organización.

3.3. Implementación

3.3.1. Jerarquía de ficheros

Establecer una estructura bien definida previamente ayuda a mantener consistencia, escalabilidad y claridad en el desarrollo. Puesto que se va a desarrollar un comando con subcomandos del mismo, se ha decidido mantener la siguiente estructura:



Como se puede observar, la funcionalidad principal reside en la raíz del directorio y los subcomandos se guardan en un directorio diferente, distinguiendo entre subcomandos simples y complejos, explicados más adelante.

3.3.2. Crear un comando de linea de comandos en Node.js

Para comenzar con la implementación se debe hacer mención del comando ***"gh alias"*** **[29]**, específicamente la opción ***"–shell"***, puesto que permite utilizar el intérprete de la terminal y así poder crear comandos personalizados. Así pues, la adaptación del código basado en ***Node.js*** a su empleo como un comando es de vital importancia. Para ello se deben realizar unas simples modificaciones en el ***"package.json"***, para indicar una nueva clave(***"bin"***) que contendrá el nombre del comando y el archivo objetivo, en este caso añadiríamos lo siguiente:

```
{
    "bin" : {
        "gh-org" : "./index.js"
    }
}
```

A continuación, en el fichero indicado(*index.js*) se debe incorporar ***"#!/usr/bin/env node"*** al principio. Finalmente si realizamos ***"node install -g ."*** instala el código como un comando, admitiendo ejecutarlo en terminal como cualquier otro escribiendo ***gh-org***(Fuente, *"Build a Command Line Application with Node.js"* de David Neal[30]).

3.3.3. Funcionalidades del comando

Empezando con la funcionalidad y que se va a trabajar para una organización, es conveniente guardar en un fichero el nombre para realizar las acciones por defecto con dicha organización, como fin de economizar la escritura del usuario. Es por ello que el primer comando, ***cd***, toma este objetivo.

Para guardar la organización y que no se pierda una vez se cierre la terminal, se emplea un objeto ***JSON***, llamado "org.json", que guarda la organización de la siguiente manera:

```
{
  "organizationDefault": "nombreOrganizacion"
}
```

El subcomando ***list***, siguiendo los requisitos mencionados anteriormente es quien recopila todos los comandos relacionados con obtención de información dentro de las organizaciones de *GitHub*. Por ello se dispone de un subdirectorio particular, donde se almacenan todos los subcomandos que cuelgan de este. A continuación se describen uno a uno los subcomandos:

En primer lugar se encuentra***default***, una simple extracción de la organización por defecto contenida en el fichero mencionado anteriormente(*org.json*). Esto se logra a través de una de las funciones nativas que *Node.js* ofrece sobre la manipulación de archivos, como es ***fs***.

En segundo lugar, ***overview***, devuelve un resumen de los datos más relevantes de una organización como pueden ser su *url*, API o descripción. Esto se realiza mediante una petición a la API, a través de ***gh api***, puesto que se trata de una petición de un único objeto y nunca habrá problemas con el límite por petición del servidor.

Figura 3.2: Ejemplo de *overview*

En tercer lugar ***people***, muestra cada uno de los miembros de la organización tanto usuario como enlace al perfil de GitHub. Así mismo contiene un parámetro (***"–outside-collaborators"***) para indicar que se quieren mostrar los colaboradores fuera de la organización, en lugar de los miembros. Esto se lleva a cabo de la misma manera que en ***overview***, al ser una petición simple no requiere gran complejidad, más allá del entendimiento de la API de GitHub y la gestión de la respuesta.

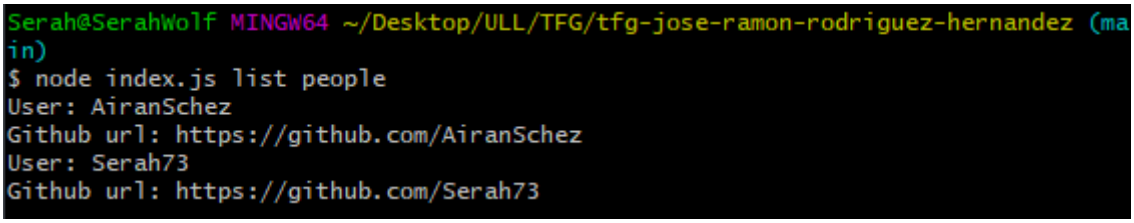


Figura 3.3: Ejemplo de *people*

A continuación ***teams***, enseña los equipos creados dentro de la organización, aunque se añaden dos opciones que se repetirán otros subcomandos. Empezando por ***"–pattern"***, acepta como argumento una expresión regular para filtrar la respuesta que queremos seleccionar. Por último, ***"–org"*** que habilita pasar por argumento otra organización, para trabajar con otras que no sean la establecida por defecto y dar mayor fluidez al usuario.

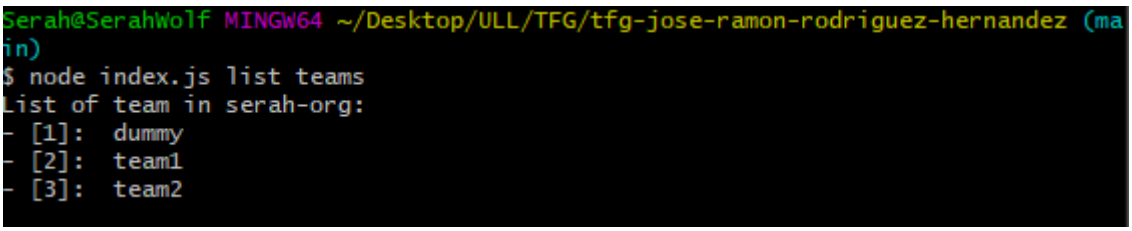


Figura 3.4: Ejemplo de *teams*

En último lugar, ***repo*** lista los repositorios de la organización, este por defecto muestra un máximo de los treinta primeros repositorios a causa del límite en la respuesta del servidor, comentado anteriormente. Esto se soluciona con la opción ***"–paginate"*** gracias a realizar la petición a través de ***octokit***, puede mostrar todos los repositorios, aunque es una llamada que puede llegar a tardar según la cantidad de información. También contiene las opciones de *"–pattern"* y *"–org"* que se explicaban en el comando *teams*.

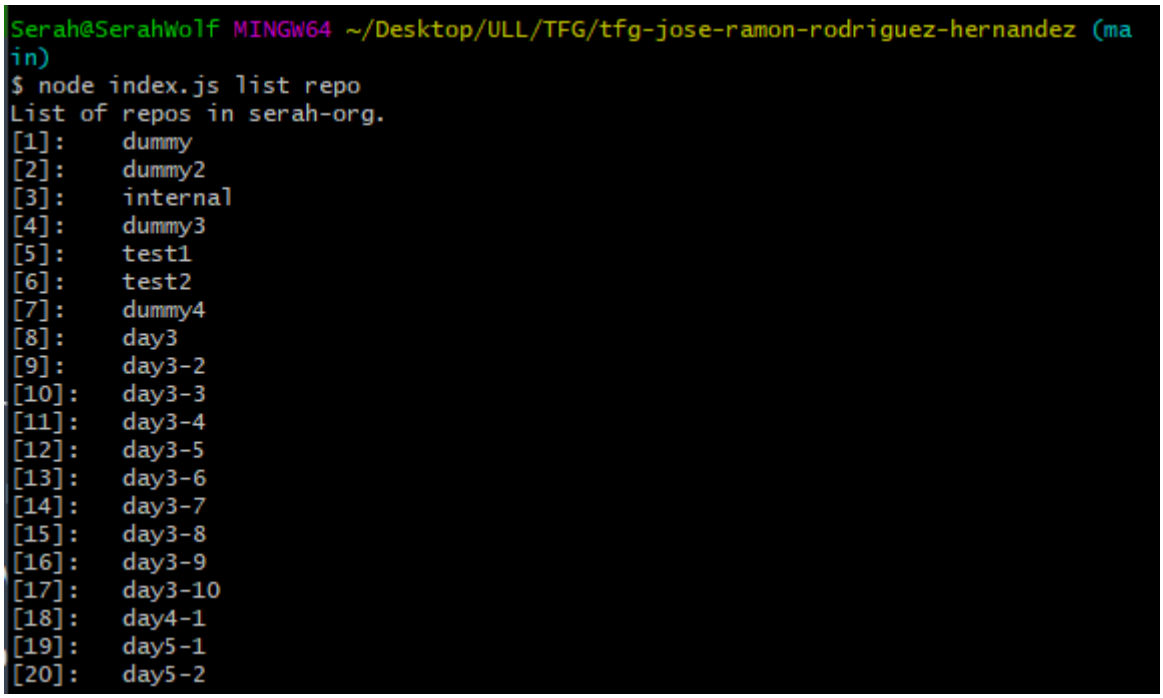


Figura 3.5: Ejemplo de *list repo*

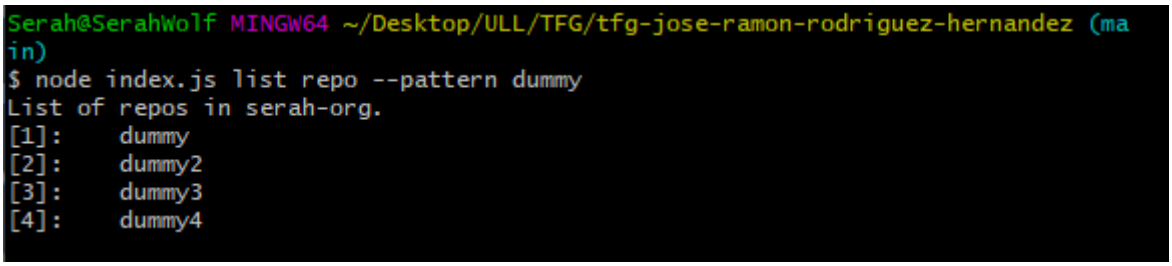


Figura 3.6: Ejemplo de *list repo –pattern*

Como se puede observar en la figura 3.6, solo se muestran cuatro repositorios(*dummy, dummy2, dummy3 y dummy4*). Aunque existen más que casen con la expresión no los encuentra por el límite en la solicitud, comentado anteriormente. En cambio si se realiza con la opción ***–paginate*** se puede observar también a *"dummy73"*. Véase la figura 3.7

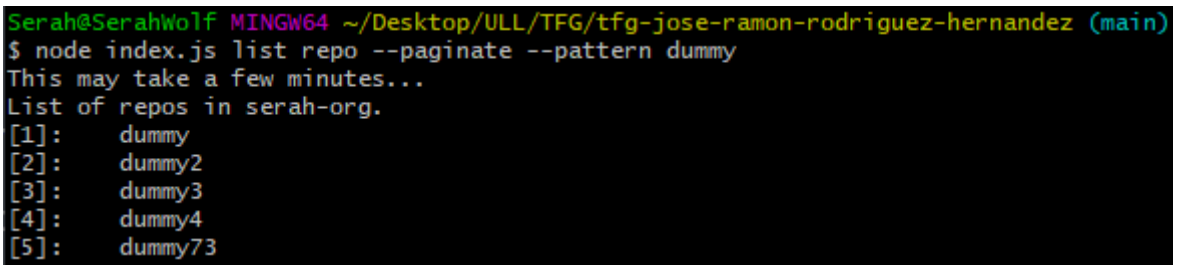


Figura 3.7: Ejemplo de *list repo –pattern –paginate*

Finalmente encontramos el subcomando ***create***, quien se encarga de lo relacionado con crear, como indica su propio nombre, diferentes recursos dentro de *GitHub*. Al igual que ***list***, dispone de un subdirectorio propio donde se encuentran los subcomandos relacionados.

En primer lugar ***repo***, crea un repositorio dentro de la organización, no se indican licencias ni un ".gitignore" por defecto, ya que de lo contrario se crea un directorio en la carpeta de instalación y puede acarrear conflictos con la comunicación de repositorios remotos. Adicionalmente, tenemos de nuevo la opción ***"–org"*** para seleccionar otra organización.

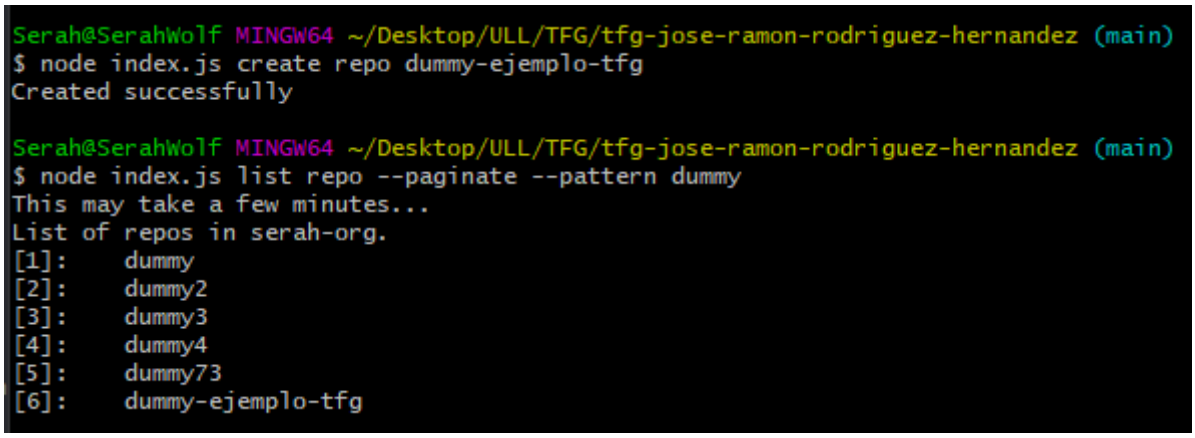


Figura 3.8: Ejemplo de *create repo*

Por último, ***superrepo*** siendo la funcionalidad con más incidencias por la cantidad de conflictos entre herramientas, permite crear un repositorio, al que se le adjudica un enlace simbólico con el mismo nombre del repositorio, debido a que cada enlace debe ser único. Este repositorio contiene todos los repositorios de organización como submódulos, mediante ***git submodule***. Esto se realiza con una petición a la API de todos los repositorios, como haría ***"gh-org list repo –paginate"***, para posteriormente agregarlos con *git submodules* y terminando con un ***push*** de toda la información al repositorio. Esto se guarda en el subdirectorio ***"/.gh/superrepo/<nombre-repositorio>"***.

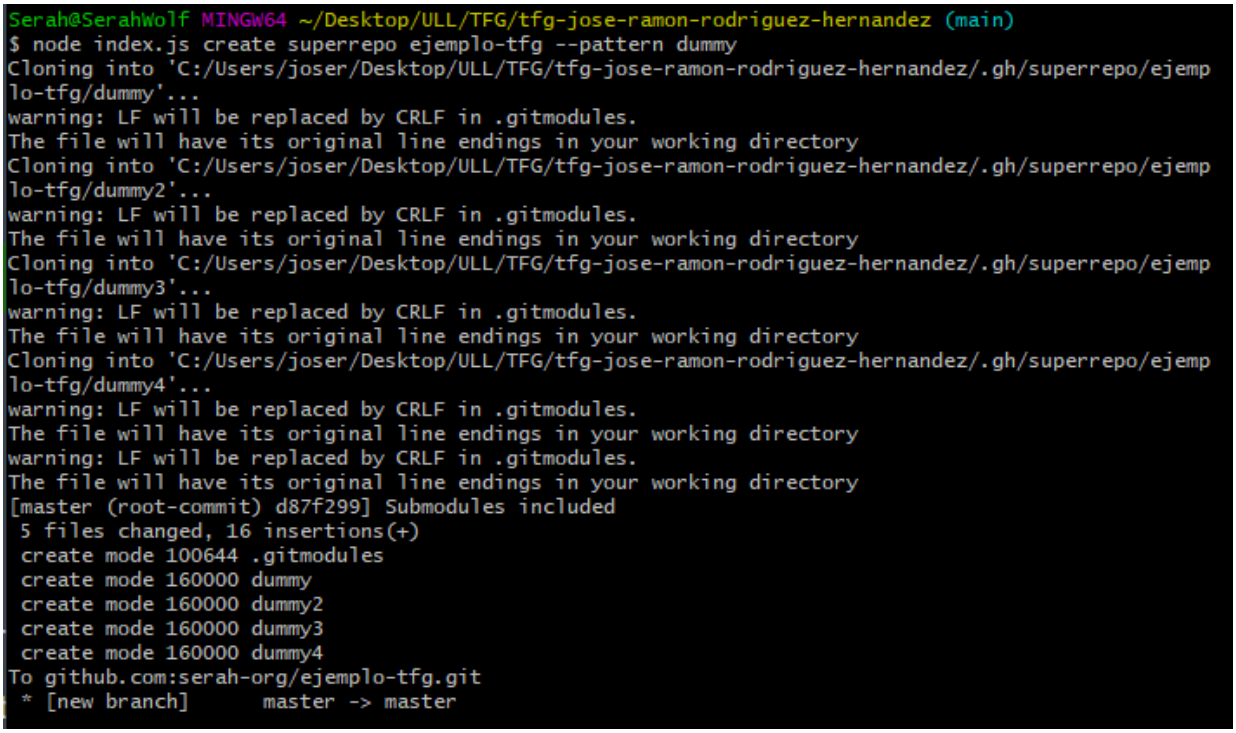


Figura 3.9: Ejemplo de *create superrepo*

Capítulo 4

Conclusiones y líneas futuras

Cabe destacar que a finales del desarrollo de nuestra extensión, *GitHub Cli* lanzó su versión “2.0” cuando se encontraban en la “1.14.0”. Esto implica cambios que pueden alterar la compatibilidad con lo anterior y da un nuevo enfoque a las líneas futuras.

Nuevas funcionalidades como el subcomando *list*, que se incorporó a todos los comandos existentes dentro de la herramienta como parte de este proyecto, llevan a pensar que en el análisis de requisitos se tomó una visión acertada de las necesidades básicas en la herramienta. Por otro lado, el nuevo comando ***gh extension*** abre una nueva vía que facilita la incorporación de módulos u otras herramientas como soporte a la misma, por lo que quedaría obsoleto el método actual pero no inservible.

La implementación actual es incompleta, no está exenta de bugs y algunos detalles hacen difícil su funcionamiento en producción.

En cuanto a la seguridad del módulo, no se han requerido métodos específicos para éste dado que no se guarda información sensible. Siendo la gestión del token la única vía problemática, está controlada por el cifrado de GitHub mediante el *“Personal Access Token”*. Así mismo, solo se han empleado módulos oficiales de GitHub para gestionar la autenticación del usuario.

Capítulo 5

Summary and Conclusions

It is worth noting that late in the development of our extension, *GitHub Cli* released their “2.0” version when they were on “1.14.0”. This implies changes that may alter backwards compatibility and gives a new approach to future lines.

New functionalities such as the subcommand *list*, which was incorporated into all existing commands within the tool as part of this project, lead to believe that the requirements analysis took an accurate view of the basic needs in the tool. On the other hand, the new command ***gh extension*** opens a new way that facilitates the incorporation of modules or other tools to support it, making the current method obsolete but not useless.

The current implementation is incomplete, it is not free of bugs and some details make it difficult to operate in production.

As for the security of the module, no specific methods have been required for it since no sensitive information is stored. Being token management the only problematic field, it is controlled by the GitHub encryption through the "*Personal Access Token*". Also, only official GitHub modules have been used to manage user authentication.

Capítulo 6

Presupuesto

Para el presupuesto del proyecto se toma un sueldo de desarrollador junior, siendo alrededor de 850 euros mensuales a media jornada. Se estima que se ha llevado a cabo el desarrollo a lo largo de dos meses computable, las tecnologías utilizadas son completamente gratuitas por ser código abierto y el coste de un equipo de trabajo, aproxima un coste total de 2500 euros para el proyecto.

Bibliografía

[1] GitHub. GitHub on the command line. https://cli.github.com.

[2] GitHub. Manual of GitHub on the command line. https://cli.github.com/manual/. github-repo: https://github.com/cli/cli/.

[3] Ryan Lienhart Dahl. Node.js, an open-source, cross-platform, back-end JavaScript runtime environment. https://nodejs.org/es/.

[4] Isaac Schlueter. npm, a package manager for the JavaScript. https://www.npmjs.com.

[5] José Ramón Rodríguez. A plugin for gh to manage GitHub organizations. https://www.npmjs.com/package/gh-plugin-org. github-repo: https://github.com/ULL-ESIT-GRADOII-TFG/tfg-jose-ramon-rodriguez-hernandez.

[6] GitHub. Hub, a command line tool that wraps git in order to extend it. https://github.com/github/hub.

[7] GitHub. Issues to track ideas, feedback, tasks, or bugs for work on GitHub. https://docs.github.com/en/issues/tracking-your-work-with-issues/about-issues.

[8] GitHub. Teams, a groups of organization members that reflect your company or group's structure. https://docs.github.com/es/organizations/organizing-members-into-teams/about-teams.

[9] GitHub. help you organize and prioritize your work. https://docs.github.com/es/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards.

[10] Taiichi Ohno. Kanban, a lean method to manage and improve work across human systems. https://en.wikipedia.org/wiki/Kanban_(development).

[11] GitHub. Explore how to use pagination to manage your responses with some examples using the Search API. https://docs.github.com/en/rest/guides/traversing-with-pagination.

[12] Facebook. A query language for your API. https://graphql.org.

[13] GitHub. Octokit plugin to paginate REST API endpoint responses. https://github.com/octokit/plugin-paginate-rest.js/.

[14] GitHub. Create a personal access token to use in place of a password with the command line or with the API. https://docs.github.com/en/github/authenticating-to-github/keeping-your-account-and-data-secure/creating-a-personal-access-token.

[15] Git. Fundamentos de Git sobre el control de versiones. https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Fundamentos-de-Git.

[16] Git. Git Submodule, a record within a host git repository that points to a specific commit in another external repository. https://git-scm.com/book/en/v2/Git-Tools-Submodules.

[17] GitHub. API de REST de GitHub. https://docs.github.com/es/rest.

[18] Jesús Lucas. Qué es NodeJS y para qué sirve. https://openwebinars.net/blog/que-es-nodejs/.

[19] Equipo Geek. Qué es NodeJS y primeros pasos. https://ifgeekthen.everis.com/es/que-es-node-js-y-primeros-pasos.

[20] Priyesh Patel. What exactly is Node.js? https://www.freecodecamp.org/news/what-exactly-is-node-js-ae36e97449f5/.

[21] TJ Holowaychuk. The complete solution for node.js command-line interfaces. https://www.npmjs.com/package/commander.

[22] Artur Adib. A portable (Windows/Linux/macOS) implementation of Unix shell commands. https://documentup.com/shelljs/shelljs. NPM: https://www.npmjs.com/package/shelljs.

[23] GitHub. Octokit.js. https://www.npmjs.com/package/octokit.

[24] Nodeca. This is an implementation of YAML, a human-friendly data serialization language. https://www.npmjs.com/package/js-yaml. GitHub: https://github.com/nodeca.

[25] Sindre Sorhus. untildify. https://www.npmjs.com/package/untildify.

[26] GitHub. Work with GitHub repositories. https://cli.github.com/manual/gh_repo.

[27] GitHub. Work with GitHub issues. https://cli.github.com/manual/gh_issue.

[28] GitHub. Make an authenticated GitHub API request. https://cli.github.com/manual/gh_api.

[29] GitHub. Make shortcuts for gh commands or to compose multiple commands. https://cli.github.com/manual/gh_alias.

[30] David Neal. Build a Command Line Application with Node.js. https://developer.okta.com/blog/2019/06/18/command-line-app-with-nodejs. GitHub: https://github.com/reverentgeek.