



**Escuela Superior
de Ingeniería y Tecnología**
Universidad de La Laguna

Trabajo de Fin de Grado

GitHub Cli Extensions for the GitHub Education Community

*Herramientas para la automatización de los flujos de trabajo
del docente que usa GitHub*

Cristo García González

La Laguna, 8 de julio de 2022

D. **Casiano Rodríguez León**, con N.I.F. 42.020.072-S profesor Catedrático de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A N

Que la presente memoria titulada:

"GitHub Cli Extensions for the GitHub Education Community"

ha sido realizada bajo su dirección por D. **Cristo García González**, con N.I.F. 43489575W.

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 julio de 2022

Agradecimientos

A mis amigos graduados por aconsejarme, fortalecerme y confiar en mí.
A mis al mismo tiempo amigos y compañeros por compartir sus propias experiencias conmigo y hacer que mi vida universitaria sea una experiencia para recordar.

A los profesores Albano José González Fernández, Francisco de Sande González, Marcos Colebrook-Santamaria, Iván Castilla Rodríguez, Ignacio García Marco y Carlos Alberto Martín Galán por la excelencia y vocación que aplican en su trabajo.

A mi familia por no esperar nunca menos de mí

Y como no a mi tutor Casiano Rodríguez León por ofrecer un proyecto y unas tecnologías interesantes y por ser un profesor y usuario exigente que le importa un buen resultado

Licencia



© Esta obra está bajo una licencia de Creative Commons Reconocimiento 4.0 Internacional.

Resumen

Es un hecho consolidado el uso de GitHub como plataforma para la enseñanza, especialmente en el ámbito de la Informática. La propia empresa apoya estas iniciativas dentro de un grupo de acciones que se engloban bajo el término GitHub Education y que incluye no solo descuentos, foros, congresos, becas, etc. sino también herramientas específicas para la educación como GitHub Classroom, la cual da soporte al proceso de asignación de tareas. Recientemente, GitHub ha introducido una herramienta orientada a desarrolladores denominada GitHub CLI que permite utilizar los servicios web de GitHub desde la línea de comandos y que provee una arquitectura denominada gh-extensions que posibilita que terceros puedan añadir nuevas funcionalidades a GitHub CLI.

Este trabajo presenta una gh-extension denominada gh-edu que pretende ser la base para un ecosistema de extensiones que facilita el flujo de trabajo diario de profesores y alumnos en tareas como: la creación y manejo de organizaciones asociadas a asignaturas, la creación y manejo de asignaciones de tareas, ayudar en el descubrimiento de casos de plagio, obtener información sobre la actividad de los alumnos, facilitar la respuesta pronta a las dudas, facilitar la elaboración la retroalimentación al alumno y mejorar la eficiencia en los procesos de evaluación. Es nuestra esperanza que esta herramienta sea de utilidad para los docentes que utilizan GitHub y que en el futuro aparezcan contribuciones de la comunidad con extensiones que cubran nuevos aspectos de la labor docente.

Palabras clave: github-education, github-cli, github-classroom, graphql, rest, git, terminal, javascript, typescript, jq, go

Abstract

The use of GitHub as a platform for teaching is a consolidated fact, especially in the field of computing. The company itself supports these initiatives within a group of actions that are included under the term GitHub Education and that includes not only discounts, forums, congresses, scholarships, etc. but also specific tools for education such as GitHub Classroom, which supports the task assignment process. GitHub has recently introduced a developer-oriented tool called GitHub CLI. that allows you to use GitHub web services from the command line and that provides an architecture called gh-extensions that enables third parties to add new functionality to GitHub CLI.

This work presents a gh-extension named gh-edu that aims to be the basis for an ecosystem of extensions that facilitates the daily workflow of teachers and students in tasks such as: the creation and management of organizations associated with subjects, creating and managing task assignments, help in the discovery of cases of plagiarism, obtain information about student activity, facilitate prompt response to queries, facilitate the elaboration of feedback to the student and improve efficiency in evaluation processes. It is our hope that this tool will be useful to educators who use GitHub and that community contributions will appear in the future with extensions covering new aspects of teaching.

Keywords: *github-education, github-cli, github-classroom, graphql, rest, git, terminal, javascript, typescript, jq, go*

Índice general

1. Introducción, Antecedentes y Estado del Arte	1
1.1. Introducción	1
1.2. Antecedentes y estado actual del tema	2
2. Tecnologías	3
2.1. GitHub	3
2.1.1. GitHub CLI	3
2.1.2. Las APIs de GitHub: REST y GraphQL	4
2.2. El buscador difuso fzf	4
2.3. Los Lenguajes JavaScript y TypeScript (Node.js)	4
2.3.1. La librería commander	5
2.3.2. La librería shelljs	6
2.3.3. La librería inquirer	7
2.4. El Lenguaje Go	7
2.4.1. La librería cobra	8
2.4.2. La librería viper	8
2.5. El Lenguaje jq (JSON Query)	9
2.6. El Lenguaje L ^A T _E X y Overleaf	9
3. Modo de uso	10
3.1. El Núcleo de gh edu: Core	10
3.1.1. Subcomando set	11
3.1.2. El subcomando core get	12
3.1.3. El subcomando core clone	13
3.1.4. El subcomando install	14
3.1.5. El subcomando remove	15
3.1.6. El subcomando update	15
3.1.7. El subcomando reset	16
3.2. La Extensión view	16
3.3. La Extensión data	17
3.3.1. El subcomando log	17
3.3.2. El subcomando teams	20
3.3.3. El subcomando team-add	21
3.4. La Extensión plagiarism	22
4. Diseño	24
4.1. Consideraciones de Diseño	24

4.2. El Núcleo de gh-edu	24
4.2.1. El fichero de Configuración data.json	26
4.2.2. Integridad de los Datos	26
4.2.3. Control del Versionado del Sistema	27
4.2.4. El Manejo de la Localidad de los Datos: cache	27
4.3. La Extensión gh-edu-plagiarism	28
5. Implementación	31
5.1. El controlador	31
5.1.1. Manejo del único punto de información: data.json	32
5.2. Extensión minimalista: gh-edu-view	33
5.3. La librería shelljs y el manejo de E/S: gh-edu-data	34
5.4. Concurrencia con Go y CSP: gh-edu-plagiarism	35
6. Conclusiones y líneas futuras	37
7. Summary and Conclusions	38
8. Presupuesto	39
A. Repositorios del ecosistema	40
Glosario	41

Índice de Figuras

2.1. Tipado fuerte en el archivo de datos	6
2.2. Script GitHub release	8
3.1. Ayuda general de gh edu	10
3.2. Ayuda de gh edu set	11
3.3. Ayuda de gh edu get	12
3.4. clone usando fzf	14
3.5. Ayuda de get update	15
3.6. Ejemplo de uso de gh view con -id "alu[0-9]{10}"	17
3.7. gh edu data log. Ejemplo de archivo de entrada	18
3.8. gh edu data log. Seleccionando los datos deseados	18
3.9. gh edu data log. Enlazando los campos del fichero de entrada con el nombre y el ID	18
3.10 gh edu data log. Interfaz principal	19
3.11 gh edu data log. Resultado	19
3.12 gh edu data log. Ayuda	20
3.13 Hipotéticos equipos en una organización	20
3.14 Resultado del comando gh edu data teams	21
3.15 Ayuda para subcomando team-add	21
3.16 Ayuda de gh edu plagiarism	22
4.1. Diseño. Ejemplo de un script para ejecutar una extensión creada con lenguaje interpretado	25
4.2. Diseño. Ejemplo de contenido de una extensión creada con lenguaje compilado	25
4.3. Ejemplo de data.json. El comando data tiene su propio campo: teams	26
4.4. Resultado de MOSS sin procesar	28
4.5. Diagrama de funcionamiento de plagiarism	29
4.6. Informe final dado por plagiarsim, con enlace para comprobar el código manualmente	30
4.7. Plagiarism. Grafo resultante mostrando los niveles de similitud	30
5.1. Implementación. Código para instalar extensiones	31
5.2. Implementación. Controlador. Añadir extensiones third-party	32
5.3. Obtención del archivo data.json	32
5.4. Implementación. data.json. Comprobamos que los campos son válidos	33
5.5. Implementación. gh-edu-view. Obtención de posibles identificadores	34
5.6. Implementación. gh-edu-view. Petición para conseguir diversos campos de cada alumno	34
5.7. Paralelismo limitado con un semáforo	36
5.8. Código general de plagiarism	36

Índice de Tablas

8.1. Tabla del presupuesto 39

Capítulo 1

Introducción, Antecedentes y Estado del Arte

1.1. Introducción

Durante muchos años, profesores de programación de todo el mundo han utilizado y siguen utilizando GitHub como plataforma para la enseñanza de asignaturas relacionadas con la programación, el trabajo en equipo y el desarrollo de proyectos. La propia empresa GitHub ha apoyado estas iniciativas dentro de un grupo de acciones que se inscriben bajo el término GitHub Education. Ello incluye descuentos en plataformas y herramientas para profesores y alumnos, foros específicos de discusión, congresos, becas, y herramientas específicas. Entre estas últimas cabe señalar GitHub Classroom, la cual da soporte al proceso de asignación de tareas.

Hace solo dos años GitHub introdujo una herramienta denominada GitHub CLI (a veces referenciada como gh-cli o gh). Es una herramienta de código abierto para utilizar los servicios que provee GitHub, pero desde la línea de comandos. Es posible ver, crear, clonar, y bifurcar repositorios; crear, cerrar, editar y ver incidencias, etc. Sin embargo el conjunto de funcionalidades ofrecido es aún bastante menor que el disponible en la interfaz web.

En agosto del año 2021 los desarrolladores de GitHub Cli añadieron un mecanismo denominado gh-cli extensions (denominado gh-extensions) que facilita que terceros puedan añadir nuevas funcionalidades a gh-cli mediante un repositorio GitHub. En el contexto específico de GitHub Education, las gh-extensions abren nuevas posibilidades para proveer comandos que facilitan el flujo de trabajo diario de los profesores y alumnos y que pueden ser de uno de estos tipos:

- Creación y manejo de las organizaciones GitHub asociadas a las asignaturas
- Creación y manejo de asignaciones de tareas dentro de una asignatura
- Obtención de información sobre las actividades de los alumnos
- Facilitar la respuesta pronta a las dudas
- Facilitar la elaboración de propuestas de mejora a los trabajos realizados por los alumnos
- Mejorar la eficiencia en los procesos de evaluación

1.2. Antecedentes y estado actual del tema

Uno de los primeros intentos en integrar GitHub al mundo educativo fue [“teachers_pet”](#), una herramienta de la línea de comandos que ayudaba al profesorado a impartir sus clases intentando emular una plataforma de aprendizaje. Cada clase era una organización de GitHub y los alumnos estaban repartidos en equipos en los que estaban solo ellos mismos, de esta manera los profesores podían gestionar todos los repositorios de la organización, dar plantillas con las que los alumnos podían empezar a trabajar, comprobar el progreso, etc. Más tarde, GitHub respondería a la demanda con Github Classroom un servicio web de uso más intuitivo que dejaría obsoleto a teacher’s pet, y que a día de hoy es la más utilizada.

En lo referente al GitHub CLI ya existía con anterioridad, una herramienta antecesora similar que proveía funcionalidades similares, fue llamada hub, actualmente mantenida por Mislav Marohnić (empleado de GitHub, Inc.). Hub fue creado originalmente por Chris Wanstrath, que junto con Tom Preston, Scott Chacon y P.J. Hyett fundaron GitHub en 2008. al igual que GitHub CLI esta herramienta nos permite interactuar con Github desde la terminal, pero esta se podría considerar un producto personal y más centrado en los denominados [power users](#). El 20 de febrero de 2020 Github anuncia Github CLI, el sucesor de hub mantenido y en desarrollo por un equipo oficial de la compañía y más amigable de usar. El 20 de agosto de 2021 sale la versión 2.0 con el comando `extension`[\[1\]](#), con la cual los usuarios podrán crear sus propias extensiones que nos permiten generar por nosotros mismos y con relativa facilidad servicios y herramientas que se integren bien con las implementaciones actuales usando las [APIs](#) de GitHub REST [API](#)[\[2\]](#) y GitHub GraphQL[\[3\]](#).

Actualmente, existen cientos de extensiones dirigidas al desarrollador, pero son pocas las que están dedicadas a facilitar la gestión de las organizaciones, en especial aquellas dedicadas a la enseñanza. Este proyecto tiene como objetivo crear una extensión modular de GitHub CLI que complementa a GitHub classroom y que provee funcionalidades como:

- Creación y asignación de tareas
- Seguimiento del progreso de los miembros de la organización
- Soporte a la gestión de incidencias
- Facilitar la retroalimentación a los trabajos realizados por los alumnos
- Interoperabilidad entre subcomandos para permitir comportamientos más complejos
- Interfaz rápida y amigable

Capítulo 2

Tecnologías

La oferta de tecnologías a la hora de realizar un proyecto software es realmente extensa. A menudo la elección de estas tecnologías para formar la pila de desarrollo está vagamente justificada y el mayor argumento suele ser la familiaridad de los desarrolladores con las herramientas escogidas. Esto, de hecho, no es un argumento menor, ya que en un entorno real, adquirir los conocimientos suficientes sobre una serie de tecnologías como para crear un producto listo para la explotación necesita una gran cantidad de tiempo por parte del equipo de desarrollo y por ende también supone un importante costo económico. Sin embargo, sí que deberían conocerse los puntos débiles y fuertes de las principales opciones y debería realizarse un análisis para evitar problemas graves en el futuro. En este capítulo se describen las tecnologías empleadas para la realización de este proyecto y la justificación de la elección realizada. No se listan dependencias triviales, y que tengan un propósito inmediato (leer ficheros, crear archivos temporales, etcétera).

2.1. GitHub

GitHub es el eje central de todo este trabajo. A pesar de que no sea una plataforma enfocada en la enseñanza, es innegable el valor y productividad que proporciona a virtualmente todos los desarrolladores de software del mundo. Los profesores pueden gestionar las clases en el mismo ámbito en el que se desarrollan las prácticas o tareas, y con la herramienta desarrollada en este trabajo pretendemos reducir el cambio de contexto y redundancia que sufren los alumnos entre el Moodle de la institución (o cualquier programa de la misma índole) y GitHub.

2.1.1. GitHub CLI

Como se ha comentado anteriormente, Github CLI[4] y su comando `gh` extension nos permite crear programas destinados a la interacción con GitHub y es la base sobre el cual se construye este proyecto, permitiéndome a mí como programador y diseñador crear un código sostenible, aprovechar comandos comunes creados por el equipo de GitHub y aumentar las posibilidades de integrarme con otros programas creados por otros miembros de la comunidad.

Quiero mencionar los comandos `gh api`[5] y `gh auth login`[6]. El primero sirve para realizar peticiones HTTP (API REST y GraphQL) imprimiendo la respuesta, y el segundo autentica al usuario con un token y guarda información del usuario. Son especialmente útiles porque no tengo que pedir constantemente los credenciales y demás datos del

usuario, por lo que no solo las peticiones son más sencillas, sino que me ayuda bastante en hacer que la aplicación sea segura.

2.1.2. Las APIs de GitHub: REST y GraphQL

Las APIs que nos proporciona GitHub son completamente necesarias. Sin ellas no podríamos tener ningún tipo de comunicación con los servicios de GitHub.

Se proveen dos APIs: API REST[2] y GraphQL[3]. En este trabajo se hace uso de las dos, pero la API REST queda relegada para casos sencillos. Se ha decidido de esta forma, pues con GraphQL podemos obtener solo los datos que queremos, y podemos recibir en una única llamada información que de otra forma requeriría varias llamadas a diferentes *endpoints*.

2.2. El buscador difuso fzf

La aplicación fzf [7] del inglés *fuzzy finder* es un **buscador difuso** para la línea de comandos. Nos permite de manera interactiva filtrar de cualquier lista uno o varios elementos de una forma muy rápida. Su uso es muy minimalista y simple: el programa lee de la STDIN (estándar input) y los elementos seleccionados son escritos a la STDOUT (estándar output) Entre sus características, las que se han usado en este proyecto son:

- Previsualización del elemento seleccionado 3.10
- STDIN dinámico
- Selección múltiple 3.4
- Personalización de la interfaz 3.4 3.10

El usuario también puede hacer uso de su **sintaxis de búsqueda** para agilizar el filtrado de elementos

Uno de los problemas principales a la hora de usar fzf es que no es una librería, sino un programa independiente distribuido como un binario. Esto significa que se tiene que invocar como un comando externo, y en el caso de `shelljs`, características como el STDIN dinámico son difíciles de usar. Aparte le tenemos que pedir al usuario que tenga instalado fzf y este disponible en el PATH.

Otro punto más preocupante es que fzf utiliza STDERR para mostrar la interfaz, probablemente debido a que la STDOUT ya está siendo utilizada para devolver la selección del usuario, es decir, el resultado al presionar *ENTER*. Esto significa que los errores están siendo escritos en un descriptor de archivos que ya está siendo utilizado. Para los errores propios de fzf no hay ningún problema, pues la aplicación se cancela antes de mostrar ningún error, pero cuando se está ejecutando en conjunto con otros programas acaparar la STDERR no es buena idea y puede generar comportamiento indefinido, dependiendo de como cada emulador de terminal maneje este tipo de situaciones.

2.3. Los Lenguajes JavaScript y TypeScript (Node.js)

El lenguaje de programación escogido es JavaScript[8] (a partir de ahora JS) en su implementación de Node.js. Los motivos son variados. Por un lado, es uno de los lenguajes

con el que tengo más experiencia y destreza programando. A su vez, si tenemos en cuenta que está en el top de lenguajes más usados [en varias listas](#), es más fácil encontrar gente que esté dispuesta a contribuir al núcleo y sus principales extensiones. Así mismo, como es tan común, los usuarios no tendrían que estar instalando dependencias de más.

Dejando de lado su popularidad y centrándonos en propiedades más intrínsecas del lenguaje, podemos ver que no hay mejor lenguaje para el manejo de ficheros JSON (usado por las API de GitHub), también al ser un lenguaje interpretado y flexible me ha permitido ser muy productivo en la fase de diseño y prototipado.

Pasada la fase de diseño y prototipado, se decidió reescribir el core en TypeScript [9], con el objetivo de minimizar bugs y aprovechar una mejor integración con editores de texto compatibles con: autocompletado inteligente, saltos a referencias, refactorización, etc. Debido a que dicho código ya estaba empezando a alcanzar un tamaño considerable.

Donde más beneficio ha resultado TypeScript, ha sido en el control del archivo de datos, `data.json` (véase Figura 4.3). Al ser el punto central donde se escriben y se leen los datos, varias partes del core dependen que dicho fichero tenga una cierta estructura.

Por ejemplo,

- el comando `gh edu reset` tiene que reescribir casi todos los campos a su estado por defecto (o todos si se utiliza el flag `--force`),
- `gh edu update -c` necesita saber en qué parte estamos guardando la [caché](#).
- Si cambio el nombre o añado nuevos campos, TypeScript me avisa con errores de compilación de que los comandos dependientes también tiene que cambiar y adaptarse a la nueva estructura.

Esto también es aplicable en menor medida a todos los puntos donde se utiliza un objeto JS.

Un ejemplo de los beneficios de TypeScript se puede ver en el esquema mínimo de `data.json` (Figura 2.1)

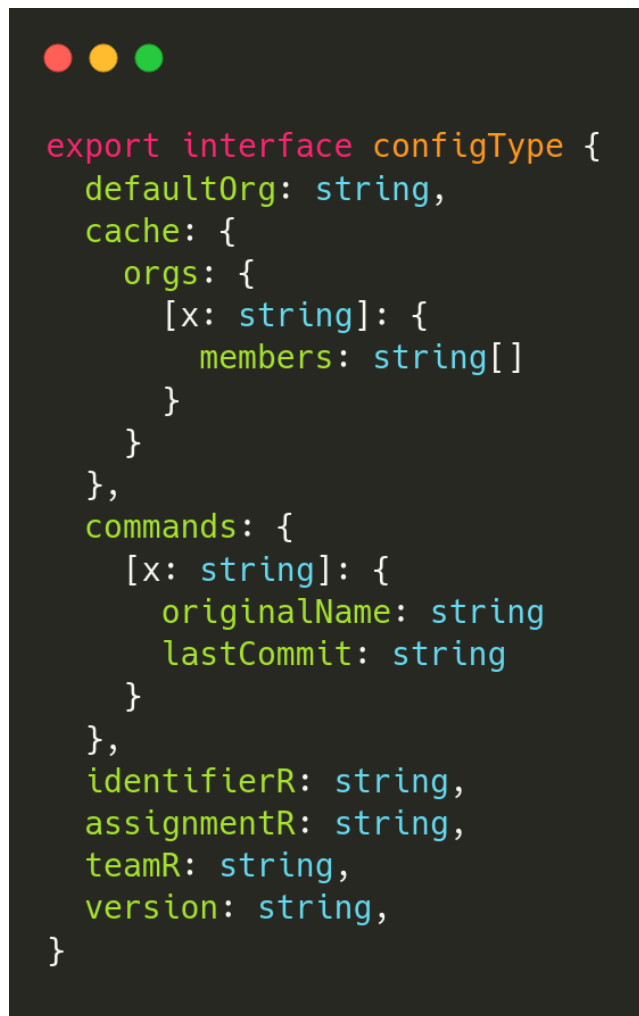
No obstante, TypeScript es un lenguaje compilado y como tal requiere de su propio compilador. Esto añadiría otra dependencia al sistema aparte de Node.js. Más preocupante aún sería pedirle a nuestro usuario que compilase el código cada vez que vaya a instalar o actualizar. Hay otras herramientas como [ts-node](#) que nos permitirían ejecutar el código directamente, pero no solo estamos añadiendo más dependencias, sino que el tiempo de arranque de la aplicación se vería enormemente afectado, incluso si se utiliza un [JIT](#) (*just-in-time compiler*). Para solucionar estos problemas se ha optado por hacer compilación anticipada o [AOT](#) (ahead-of-time) en cada `commit` que se realice con un [hook](#)[10] de Git.

En cuanto a dependencia de desarrollo, se ha usado como gestor de paquetes [npm](#)[11] por mera preferencia personal.

2.3.1. La librería `commander`

La librería `commander` es muy popular para la creación de aplicaciones [CLI](#), se encarga de parsear los argumentos y flags de la línea de comandos, de los mensajes de error relacionados con la creación de dichos argumentos e implementa un sistema de ayuda basándose en las especificaciones establecidas.

Es una parte esencial del ecosistema, a efectos prácticos genera la interfaz mínima necesaria, entre el usuario y el programa y la solución nativa `process.argv` es muy pobre y primitiva en comparación.



```

export interface configType {
  defaultOrg: string,
  cache: {
    orgs: {
      [x: string]: {
        members: string[]
      }
    }
  },
  commands: {
    [x: string]: {
      originalName: string
      lastCommit: string
    }
  },
  identifierR: string,
  assignmentR: string,
  teamR: string,
  version: string,
}

```

Figura 2.1: Tipado fuerte en el archivo de datos

Los motivos por los cuales se ha optado específicamente por commander son:

- **Popularidad.** Lo cual se traduce en un software a prueba de errores y con abundante documentación.
- **Cero dependencias.** Es bastante remarcable que un paquete del ecosistema de JS sea autosuficiente, y nos ayuda en minimizar el tamaño y mantenimiento de nuestro software, en especial en la parte del core que interesa que sea lo más convenientemente ligero.

2.3.2. La librería shelljs

La librería shelljs es el paquete que me permite llamar a comandos de la terminal desde el código fuente, independientemente del sistema operativo.

Incluye comandos propios de los sistemas Unix como cp, rm o which entre otros, y un comando exec para ejecutar cualquier otro comando no contemplado por su API.

Es bastante trivial de usar, pero su interfaz es algo pobre ante todas las posibilidades que ofrece la terminal. Alimentar la STDIN con variables del programa, cuando el comando ya ha empezado su ejecución, o mandar una señal de cierre, no es posible al momento de escribir estas líneas. Ha habido intentos de ofrecer una [solución apropiada](#), pero no ha habido avances, por lo que se ha tenido que recurrir a soluciones alternativas. Se puede

ver una explicación más detallada y el porqué este comportamiento es necesario en el capítulo **Implementación**, en la sección de `gh-edu-data` (5.3).

2.3.3. La librería `inquirer`

La librería `inquirer` me permite conseguir datos del usuario de forma interactiva (figura 3.8). Es una colección de interfaces comunes como:

- Confirmación
- Múltiple selección (checkbox)
- Listas a la `fzf`

El método principal para conseguir input del usuario de forma interactiva es `fzf`. La librería `inquirer` carece de características como la búsqueda difusa o `stdin` dinámico. La utilidad `fzf` es una herramienta diseñada para trabajar con listas. En aquellas ocasiones que se requiere una simple confirmación o que el usuario tenga que escribir manualmente, se recurre a `inquirer`.

A pesar de no ser tan potente como `fzf`, `inquirer` es una librería nativa de JS, por lo que ha sido más fácil trabajar con ella, especialmente en lo referente al manejo de errores.

2.4. El Lenguaje Go

En una de las extensiones se utiliza el lenguaje de programación Go [12], el motivo de su uso se explica mejor en su respectivo apartado (5.4).

Go es un lenguaje relativamente moderno (2009), que ha tenido una aceptación rápida en las tecnologías de la nube, como pueden ser servidores, pero también se ha utilizado mucho en el desarrollo de aplicación de terminal, de hecho, `GitHub CLI` y su predecesor `hub` han sido desarrollados usando Go.

Dentro del ámbito de este proyecto, las ventajas de Go son varias:

- **Binario enlazado estáticamente.** Por lo que los usuarios no tiene que tener instalado Go
- **Arranque inmediato.** A medida que las extensiones y el core escritos en JS/TypeScript iban creciendo en tamaño y en dependencias de desarrollo, el rendimiento de la aplicación iba decayendo. Por lo general no es especialmente notorio ni grave, pero donde más se nota es cuando la aplicación empieza la ejecución, principalmente debido a los procesos que se tienen que ejecutar al inicio, como la validación del archivo de datos o la carga de extensiones instaladas.
- **Mejor integración con la terminal.** Los paquetes `os` e `io` de la librería estándar han sido de gran ayuda. He podido utilizar todas las características de `fzf` y `jq` manteniendo un código legible y mantenible.
- **Manejo de errores.** El manejo de errores en Go es bastante explícito. Esto hace que el código sea más verboso, pero tengo la seguridad de que mi aplicación tiene todos los posibles errores cubiertos. El usuario no va a ver ningún *stack trace*, a no ser que el mensaje de error este destinado al programador.

En cuanto a desventajas, podemos hablar de la distribución:

- Go es un lenguaje compilado: a diferencia de TypeScript, no compila a código fuente, sino a código máquina.
- El propio equipo de gh proporciona una GitHub Action [13] con nombre [gh-extension-precompile](#) que compila el código y lo guarda en el apartado de “Releases” de GitHub, a la hora de instalar o actualizar, el comando `gh extension install` que se utiliza de forma interna, descarga el binario para el sistema operativo y arquitectura apropiados.
- Esto ralentiza la productividad al tener que esperar que se compile a todas las plataformas (Linux, Mac OS, Windows) antes de poder probar los resultados con el resto del ecosistema.

```
name: release
on:
  push:
    tags:
      - "v*"
permissions:
  contents: write

jobs:
  release:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: cli/gh-extension-precompile@v1
        with:
          go_version: "1.18"
```

Figura 2.2: Script GitHub release

2.4.1. La librería cobra

La librería cobra [14] es para Go lo que commander es para javascript: Una librería que permite crear aplicaciones de líneas de comandos no triviales. Generando ayuda automáticamente, autocompletado y con una integración perfecta con viper.

2.4.2. La librería viper

La librería viper [15] es una librería para leer archivos de configuración, en este caso data.json. También me permite enlazar datos que se puedan conseguir desde variables de entornos, línea de comandos o archivo de configuración en una sola variable, en ese orden de prioridad.

2.5. El Lenguaje jq (JSON Query)

Para el manejo no trivial de ficheros JSON se ha usado el DSL jq[16], un lenguaje de muy alto nivel que nos permite procesar ficheros JSON desde la terminal.

Si bien es cierto que uno de los motivos para usar JS como lenguaje de desarrollo fue la facilidad con la que podemos manejar los ficheros JSON, jq puede ser más productivo cuando se trabaja en el contexto de la terminal, como es el caso cuando se trabaja con programas como `gh api` o `fzf`, que producen o procesan información en formato JSON a través de los canales o pipelines.

Procesar JSON con jq puede simplificar la tarea de previsualizar información adicional de un *elemento* que está siendo seleccionado. Véase, por ejemplo, la extensión `gh-edu-view` (5.2).

2.6. El Lenguaje L^AT_EX y Overleaf

Para la elaboración de este documento se ha usado L^AT_EX, en específico la implementación de `overleaf`[17]. Gracias a esto hemos podido sincronizar con GitHub, mantener un historial del documento, se ha simplificado la instalación de paquetes y tenemos una copia accesible en línea que se sincroniza automáticamente

Capítulo 3

Modo de uso

En este capítulo se explica como se utiliza el ecosistema, con intención de que se pueda entender el propósito de cada extensión. En los capítulos siguientes se entrará en más detalle con respecto a la toma de decisiones en el diseño y como se ha logrado implementar todo el ecosistema

3.1. El Núcleo de gh edu: Core

El diseño de la extensión gh-edu sigue el [patrón estrategia](#) y consta de un código núcleo (core) que coordina las estrategias y extensiones. Algunas de las estrategias más básicas forman parte del código núcleo que se provee con la gh-extension gh-edu.

Todos los comandos de gh-edu se pueden ejecutar desde la línea de comandos y cuando es apropiado cuentan con un modo interactivo. La figura 3.1 muestra los subcomandos disponibles en el núcleo.

```
> gh edu -h
Usage: gh edu [options] [command]

Options:
  -h, --help            display help for command

Commands:
  clone [options]       Clone all the repos you want from one organization
  set [options] [value] Set some values in the configuration file
  get [options]         Show information from the configuration file
  install <plug-in>    Install a plug-in
  update [options]     Update your data file
  remove <plug-in...> Remove plugin[s]
  reset [options]      Set config in default state, installed commands are ignored
                       If you are calling this command because of some error in the configuration, please open a issue
  help [command]       display help for command
```

Figura 3.1: Ayuda general de gh edu

Para este apartado es relevante saber que gh-edu trabaja en torno a un archivo de datos o registro llamado `data.json`.

Este archivo de datos guarda información

- de las extensiones instaladas
- de la versión y configuración de las extensiones instaladas,
- de información proveida por las extensiones a solicitud del usuario

- del usuario y de la configuración del usuario
- del estado actual del sistema gh-edu (por ejemplo, cuál es la organización GitHub por defecto)
- de datos obtenidos desde la API de GitHub y que han sido cacheados

este fichero se guarda de forma persistente en `$HOME/.config/gh-edu/data.json`. Este directorio esta siempre bajo control de versiones y puede estar enlazado a un remoto llamado `https://github.com/github-user:/gh-edu-profile`.

Dicho fichero es único y privado para cada usuario y se obtiene de una de estas dos formas:

- Se crea uno nuevo a partir de una plantilla base, la cual tiene los elementos mínimos para dar el fichero por válido, o
- Se descarga automáticamente si el usuario tiene uno propio dentro de un repositorio en GitHub en su cuenta con el nombre `<github-user>/gh-edu-profile`.

Se entrará más en detalle sobre `data.json` en el capítulo **Diseño 4.2.1**.

3.1.1. Subcomando set

Comando importante, utilizado para establecer la configuración en el fichero de datos

```

> gh edu set -h
Usage: gh-edu set [options] [value]

Set some values in the configuration file

Options:
  -o, --org           Set organization
  -i, --identifier    Regex for the member identifier. Ex: alu[0-9]{10} for alu0101204512
  -a, --assignment    Regex for the current assignment. Ex: turingMachine-* for turingMachine-alu0101204512
  -t, --team          Regex to get information from team name. Ex: "(?<name>.*?)(?<id>.*?)(?<login>.*[^\s*])"
  -q, --quiet         Don't show any log or warning information. The result will be printed anyway
  -h, --help         display help for command

```

Figura 3.2: Ayuda de `gh edu set`

- **-o, --org**. Especifica la organización con la que estamos trabajando actualmente. En la mayoría de universidades, las organizaciones de GitHub suelen estar asociados a las asignaturas. Ejemplo de uso: `gh edu set -o gh-cli-for-education`, si pertenecemos a la organización se registrará como organización actual y se guardará como cache a los miembros pertenecientes de la organización. Si no se le pasa un valor, se activará `fzf` para poder elegir entre las organizaciones a las que el usuario pertenezca.
- **-q, --quiet**. No se muestra ninguna información por pantalla, más allá del resultado y los errores.
- **-i, --identifier**. Expresión regular para identificar los identificadores de los alumnos, en el caso de la ULL utilizamos `alu` seguido de 10 números del 0 al 9. que se corresponde con la expresión regular `alu[0-9]{10}`

```
Usage: gh-edu get [options]

Show information from the configuration file

Options:
  -m, --members          List community members
  -p, --plugins          List the installed plugins
  -o, --organization    Show the current organization
  -i, --identififier    Show the set regex for the identififier
  -a, --assignment      Show the set regex for the assignment
  -t, --team            Show the set team for the assignment
  -c, --configuration   Show the loaded configuration
  -v, --version         Get the current installed version
  -h, --help           display help for command
```

Figura 3.3: Ayuda de gh edu get

- **-t, --team.** Expresión regular para poder extraer información del nombre de los equipos. Está pensada para que se utilice con agrupamientos con nombre (“named captured group”). Por ejemplo, la expresión regular:

`(?<name>.*?).(?<id>.*?).(?<login>.*[^s*])`

Aplicada sobre el siguiente nombre de equipo:

`cristo-garcía-gonzález.alu0101204512.ggcristo`

Es capaz de generar automáticamente la siguiente información:

- **name:** cristo-garcía-gonzález
 - **id:** alu0101204512
 - **login:** ggcristo
- **-a, --assignment.** Expresión regular para marcar los repositorios sobre los que se quiere trabajar. Normalmente, se le asigna un repositorio a cada alumno por asignación/tarea. Por ejemplo: `turingMachine-.*` en el caso de que los repositorios empiecen por `turingMachine` como podría ser `turingMachine-alu0101204512`.

El uso que se les dé a estos campos depende deliberadamente de cada desarrollador y profesor.

3.1.2. El subcomando core get

El comando `get` muestra la información que se tiene actualmente

- **-m, --members.** Muestra los miembros de la organización establecida. Lee de la caché, si es posible.

- **-p, --plugins.** Lista las extensiones y comandos disponibles tanto los que vienen con el core (builtin) como los instalados por el usuario.
- **-o, --org.** Muestra la organización marcada.
- **-i, --identifier.** Muestra la expresión regular establecida para detectar los identificadores de los alumnos.
- **-a, --assignment.** Muestra la expresión regular establecida para filtrar los repositorios deseados.
- **-t, --team.** Muestra la expresión regular establecida para recolectar información de los nombres de los equipos.
- **-v, --version.** Muestra la versión instalada.
- **-c, --configuration.** Muestra por pantalla la toda la configuración y datos cargados.

3.1.3. El subcomando core clone

El comando clone solo tiene dos opciones `--org` por si queremos clonar de una organización en concreto sin tener que cambiar la organización establecida y `--quiet` para solo mostrar los mensajes de errores (y `--help`). Este comando fue planeado para clonar varios repositorios en paralelo usando la librería [concurrently](#)

```

gh-cli-for-education:Use tab to choose repos to download> gh
20/27 (2)
gh-edu
> gh-activity
gh-orgstats
gh-team-add
> gh-edu-view
gh-edu-data
> gh-org-teams
gh-org-clone
gh-utilities
gh-edu-crguezl
gh-edu-plagiarism
gh-org-browse-repo
gh-marking-scripts
gh-alias-for-teachers
.github
github2pandas_manager
github-cli-discussions
github-classroom-utilties
ocw-ull-github-education-en-el-aula-1718
ocw-ull-github-education-en-el-aula-2122

```

Figura 3.4: clone usando fzf

En la figura 3.4 se ve la interfaz de clone con fzf, arriba a la izquierda aparece el nombre de la organización y podemos seleccionar múltiples repositorios presionando *TAB* y filtrarlos de forma difusa (nótese como `.github`, el cual no es una coincidencia exacta, aparece en la lista), para aceptar se pulsa *ENTER*.

3.1.4. El subcomando `install`

Como su nombre indica `install` instala extensiones añadiendo más comandos al sistema. No acepta otro argumento más que el nombre de la extensión que queremos instalar y solamente tiene el flag `--quiet`, con la misma funcionalidad que el resto de extensiones.

El código de las extensiones tiene que estar alojado en GitHub y solo funciona con la rama por defecto.

El comando se ejecuta de la siguiente forma:

```
gh edu install <organización/nombre_extensión>
```

Ejemplo:

```
gh edu install crguezl/gh-edu-browse
```



```
Installing crguezl/gh-edu-browse ...
Plugin installed in system
Setting up configuration...
crguezl/gh-edu-browse installed as browse
```

Pero si la extensión es *first-party*, es decir, pertenece a la organización `gh-cli-for-education`, no es necesario especificar la organización. Por ende:

```
gh edu install gh-cli-for-education/view, es equivalente a gh edu install view
```

3.1.5. El subcomando `remove`

Elimina las extensiones instaladas por el comando `install`. No puede eliminar los comandos *builtin*.

El comando se ejecuta de la siguiente forma:

```
gh edu remove <nombre_extension1> <nombre_extension2> ... <nombre_extensionN>
```

3.1.6. El subcomando `update`

```
Usage: gh-edu update [options]

Everything related to updates
If no flag is specified it will update the core

Options:
  -c, --cache           Update your cache
  -p, --plugin <plugin...> Update plugins
  -r, --remote         Update remote data file
  -h, --help           display help for command
```

Figura 3.5: Ayuda de `get update`

El subcomando `update` se encarga de todo lo relacionado con las actualizaciones.

- **-c, --cache.** Actualiza todos los valores posibles de la caché. No es un comando esencial, especialmente cuando algunos comandos ya actualizan ciertas partes de la caché, pero es recomendable correrlo desde que se obtenga el fichero de datos `data.json`.
- **-p, --plugin.** Actualiza una o varias extensiones. Si no tiene un argumento, actualiza todas las extensiones instaladas.
- **-r --remote.** Actualiza el repositorio `gh-edu-profile` con los datos locales. En caso de no existir se le pregunta al usuario si quiere crear dicho repositorio.

- **Sin argumentos ni flags.** Si el usuario ejecuta el comando sin argumentos, el propio core se actualizará. Es equivalente a `gh extension upgrade gh-edu`. Fue creado para mantener la simetría con el resto de comandos.

3.1.7. El subcomando `reset`

Restaura la configuración a su estado original dejando intacto la información sobre los comandos instalados, si también se desea borrar dicha información se puede usar el flag `--force`, aunque no es recomendable, ya que generaría una incongruencia con las extensiones que están realmente instaladas en el sistema y la propia información del sistema.

Este comando existe por meros motivos de depuración y en caso de que un error en la implementación del sistema lleve al usuario a un estado anormal. No se debe de usar con regularidad.

3.2. La Extensión `view`

Es una pequeña extensión para mostrar información relevante sobre la organización, por lo que es necesario tener una organización establecida.

Dentro del contexto del TFG se ha desarrollado un único comando `members` que muestra información sobre los miembros de la organización. Se espera ampliarlo para que también muestre otro tipo de información que pueda ser relevante al profesorado y alumnado (vease [6](#)).

Esta extensión también aprovecha la expresión regular del identificador guardado en `data.json` para intentar conseguir los identificadores de los alumnos leyendo varios campos en su cuenta de GitHub

Se puede ejecutar directamente como: `gh edu view members` o añadir el flag `--id` para usar un identificador que tendrá prioridad sobre el establecido en `data`.

```

{
  login: 'crguezl',
  name: 'Casiano Rodriguez-Leon',
  bio: 'My Control Version Bio: Started with Unix RCS, then moved to CVS, then Subversion and finally git ',
  email: '',
  url: 'https://github.com/crguezl',
  avatarUrl: 'https://avatars.githubusercontent.com/u/1142554?v=4',
  possibleID: []
}
{
  login: 'algorithms-ull',
  name: null,
  bio: '',
  email: '',
  url: 'https://github.com/algorithms-ull',
  avatarUrl: 'https://avatars.githubusercontent.com/u/36507398?u=db7dc4eb94a793e038d32bb3d74c1f070917232b&v=4',
  possibleID: []
}
{
  login: 'GGCristo',
  name: 'Cristo García',
  bio: 'Computer Science (Ingeniería informática) student at Universidad de La Laguna (ULL).\r\n' +
    'KISS my terminal \\\r\n' +
    'clear && (useful || interesting)',
  email: 'alu0101204512@ull.edu.es',
  url: 'https://github.com/GGCristo',
  avatarUrl: 'https://avatars.githubusercontent.com/u/58046649?u=13818c37965d1f701bf1bfb3c65b2b579f5b3c40&v=4',
  possibleID: [ 'alu0101204512' ]
}
}

```

Figura 3.6: Ejemplo de uso de gh view con -id "alu[0-9]{10}"

3.3. La Extensión data

Ha sido creado para manejar y guardar información variada de los alumnos. Para instalarla:

```
gh edu install data
```

Tiene tres comandos principales: log, teams y team-add.

3.3.1. El subcomando log

El objetivo de log es principalmente relacionar la cuenta institucional de un alumno con su cuenta de GitHub y conseguir información extra que le pueda ser de utilidad al profesor.

Se necesita de un fichero inicial con información que el profesor tenga de los alumnos. Dicho fichero de entrada tiene que ser de tipo JSON, con un array donde cada elemento guarde información de los alumnos. Esta información tiene que contener mínimamente el nombre del alumno, aunque lo apropiado es que contenga el nombre y un identificador. Puede tener más datos (figura 3.7).

Una vez ejecutado el comando, el profesor decide que campos quiere tener de cada alumno (figura 3.8). Y cuáles de los campos entrantes corresponden al nombre, y en caso de haberlo el identificador (figura 3.9). Después, uno a uno y aprovechando la eficaz interfaz de fzf y la previsualización de los datos remotos del alumno, se filtra el alumno en cuestión (figura 3.10).

Como resultado se obtiene un nuevo fichero JSON que contiene la información dada inicialmente por el profesor, combinada con los datos proporcionados por GitHub (figura 3.11).

```
[
  {
    "name": "Cristo",
    "id": "alu0101204512",
    "score": 7.8
  },
  {
    "name": "Casiano",
    "id": "alu0101204577",
    "score": 10.0
  }
]
```

Figura 3.7: gh edu data log. Ejemplo de archivo de entrada

```
? Select the data you want to get (Press <space> to select, <a> to toggle all, <i> to invert selection, and <enter> to proceed)
> login
  name
  bio
  email
  url
  avatarUrl
```

Figura 3.8: gh edu data log. Seleccionando los datos deseados

```
? Select the data you want to get login, name, bio, email, url, avatarUrl
? Which field is the name? name
? Which field is the id?
> name
  id
  score
  [I am not using any kind of identifier]
```

Figura 3.9: gh edu data log. Enlazando los campos del fichero de entrada con el nombre y el ID

```

GGCristo
crguezl
2/3
Member: Cristo ID: alu0101204512 > c
{
  "login": "GGCristo",
  "name": "Cristo García",
  "bio": "Computer Science (Ingeniería informática) student at Universidad de La Laguna (ULL)",
  "email": "alu0101204512@ull.edu.es",
  "url": "https://github.com/GGCristo",
  "avatarUrl": "https://avatars.githubusercontent.com/u/58046649?u=13818c37965d1f701bf1bf3c65b2b579f5b3c40&v=4"
}

```

Figura 3.10: gh edu data log. Interfaz principal

```

[
  {
    name: 'Cristo García',
    id: 'alu0101204512',
    score: 7.8,
    login: 'GGCristo',
    bio: 'Computer Science (Ingeniería informática) student at Universidad de La Laguna (ULL).\r\n' +
      'KISS my terminal \\\r\n' +
      'clear && (useful || interesting)',
    email: 'alu0101204512@ull.edu.es',
    url: 'https://github.com/GGCristo',
    avatarUrl: 'https://avatars.githubusercontent.com/u/58046649?u=13818c37965d1f701bf1bf3c65b2b579f5b3c40&v=4'
  },
  {
    name: 'Casiano Rodriguez-Leon',
    id: 'alu0101204577',
    score: 10,
    login: 'crguezl',
    bio: 'My Control Version Bio: Started with Unix RCS, then moved to CVS, then Subversion and finally git ',
    email: '',
    url: 'https://github.com/crguezl',
    avatarUrl: 'https://avatars.githubusercontent.com/u/1142554?v=4'
  }
]

```

Figura 3.11: gh edu data log. Resultado

Tiene los siguientes flags para modificar su comportamiento:

```
Usage: gh-edu-data log [options] <inputFile>

Get relevant information about you students

Options:
  -o, --output <outputFile> File to write the resulting data. If not specified it will write the result to the standard output
  -c, --cache                 Cache the information in the configuration file
  -q, --quiet                 Don't show any output, except errors
  -h, --help                 display help for command
```

Figura 3.12: gh edu data log. Ayuda

Estos flags se comportan igual que en el resto de subcomandos, a excepción de **c**, **--cache**, en cuyo caso, estamos indicando de que queremos guardar el resultado en una **caché** en el fichero data.json. Un ejemplo del resultado se puede ver en la figura 4.3

3.3.2. El subcomando teams

Una de las estrategias que suelen utilizar los profesores para tener identificados a los alumnos es imponerles que creen un equipo, en el que estén únicamente ellos mismos, con un nombre que siga un patrón con información del alumno. Por ejemplo cristo-garcía-gonzalez.alu0101204512. De esta forma la tarea no se vuelve tan tediosa y propensa a errores.

teams es un comando que tiene el mismo objetivo que log, pero se aprovecha de esta estrategia, para que la obtención y búsqueda de dicha información sea inmediata.

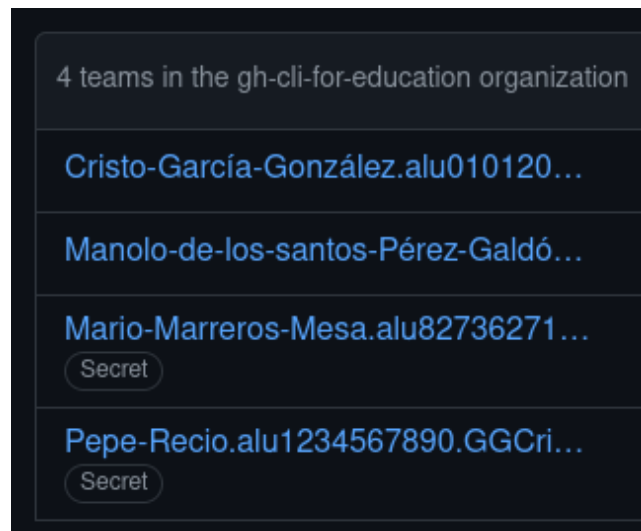


Figura 3.13: Hipotéticos equipos en una organización

```
[
  {
    url: 'https://github.com/GGCristo',
    email: 'alu0101204512@ull.edu.es',
    name: 'Cristo-García-González',
    id: 'alu0101204512',
    login: 'ggcristo'
  },
  {
    url: 'https://github.com/GGCristo',
    email: 'alu0101204512@ull.edu.es',
    name: 'Manolo-de-los-santos-Pérez-Galdós',
    id: 'alu0102340234',
    login: 'galdozz'
  },
  {
    url: 'https://github.com/GGCristo',
    email: 'alu0101204512@ull.edu.es',
    name: 'Pepe-Recio',
    id: 'alu1234567890',
    login: 'GGCristo'
  },
  {
    url: 'https://github.com/GGCristo',
    email: 'alu0101204512@ull.edu.es',
    name: 'Mario-Marreros-Mesa',
    id: 'alu8273627161',
    login: 'mmmario'
  }
]
```

Figura 3.14: Resultado del comando `gh edu data teams`

Cuenta con los mismos flags que `log` (figura 3.12)

3.3.3. El subcomando `team-add`

Este subcomando fue creado como respuesta al subcomando `teams` con la intención de también facilitar el trabajo a los alumnos, a la hora de crear los equipos en la organización.

Se aprovecha del campo `teamR` para saber que datos pedirle al alumno. Por ejemplo, si se tiene en `teamR` el siguiente valor `(?<name>.*?)\.(?<id>.*?)\.(?<age>.*[^\s*])`. El programa le preguntará al alumno por `name`, `id` y `age`. Y creará un equipo en la organización especificada por `defaultOrg`, con dichos datos y el alumno como miembro.

```
Usage: gh-edu-data team-add [options]

Create teams with certain patterns to get information later on. Empty spaces will become '-'

Options:
  -r, --regular          Don't use teamR. Use the common pattern: <name>.<id>.<login>. Login is automatic
  -s, --separator <separator> Separator to use between fields. Ignored if used with -r flag
  -h, --help            display help for command
```

Figura 3.15: Ayuda para subcomando `team-add`

- **-r, --regular.** Ignora el campo teamR y el flag **-s, --separator**. Le pregunta al alumno por su nombre e identificador y crea un equipo con dicha información, más su identificación en GitHub. El separador es '.'. Se ha considerado que este patrón es lo suficientemente conveniente, para justificar un flag exclusivo para él.
- **-s, --selector.** No se ha encontrado una manera fiable de deducir el selector a partir de teamR, por lo que el alumno tendrá que especificarlo con este flag. Si se omite, el programa le preguntará al usuario de forma interactiva.

3.4. La Extensión plagiarism

Es una extensión para detectar el porcentaje de similitud en las tareas de los alumnos. Se puede usar principalmente para ayudar al profesor a detectar plagio.

Genera un grafo mostrando los porcentajes y el número de líneas que son similares entre cada par de alumnos, y un informe con un enlace para ver las similitudes del código fuente.

Es necesario tener instalados varias dependencias:

1. **MOSS (Measure Of Software Similarity)[18] script.** El usuario tiene que solicitar un fichero en el que viene incluido la *key/id* correspondiente. Los pasos del proceso están en su [página web](#).
2. **Perl.** Para poder ejecutar el script de MOSS.
3. **mossum[19].** Para generar el grafo con los datos de MOSS.

fzf es opcional

```

gh edu plagiarism -h
gh-edu-plagiarism checks all the repositories related to an assignment and compares them to detect plagiarism

Usage:
  gh edu plagiarism [flags]

Flags:
  -a, --anonymize                Indicate if you want to randomize the names
  -c, --course string            Specify the course/organization
  -e, --exercise string         Specify the regex for the assignment/exercise
  -h, --help                    help for gh
  -l, --language https://github.com/gh-cli-for-education/gh-edu-plagiarism#compatible-languages Select the language. You can treat this flag as a boolean or pass a string
  -m, --min-lines int           Minimum lines to show links (default 1)
  -o, --output string           Save the results in the specified path
  -p, --percentage int          Minimum percentage to show links (default 1)
  -q, --quiet                   No INFO in the output only the result
  -t, --template                Indicate if there is a tutor template

```

Figura 3.16: Ayuda de gh edu plagiarism

- **-a, --anonymize**, muestra el grafo con nombres falsos aleatorios. Se podría usar en caso de que el profesor quiera mostrar el grafo a la clase.
Nota: Este comando solo se aplica al grafo, el informe se queda intacto, ya que está destinado a ser consumido únicamente por el profesor
- **-c, --course <course>**. La organización se puede indicar con este flag, y toma prioridad con respecto al archivo de configuración.
- **-e, --exercise <exercise>**. La expresión regular para la tarea se puede indicar con este flag, y toma prioridad con respecto al archivo de configuración.

- **-l, --language <language>**. Indica el lenguaje de programación utilizado.
 A pesar de que esta sea un campo obligatorio para que funcione el programa, si se omite se pedirá más tarde de forma interactiva con `fzf`
 Se aceptan los mismos lenguajes que MOSS acepta, los cuales son: `c`, `cc` (C++), `java`, `ml` (Meta Language), `pascal`, `ada`, `lisp`, `scheme`, `haskell`, `fortran`, `ascii`, `vhdl`, `perl`, `matlab`, `python`, `mips`, `prolog`, `spice`, `vb` (Visual Basic), `csharp` (C#), `modula2`, `a8086` (8086 assembly), `javascript`, `plsql` (PL/SQL), `verilog`.
- **-m, --min-lines <lines>(Por defecto: 1)**. Indica cuál es el número mínimo de líneas que deben ser similares para que el estudiante salga en el grafo.
- **-o, --output <path>**. Indica donde se quiere guardar el resultado. El informe y el grafo se guardan en el directorio temporal del sistema hasta la siguiente ejecución, utilizando este flag se puede indicar donde guardar los resultados de forma persistente.
- **-p, --percentage <percentage>(Por defecto: 1)**. Indica cuál es el porcentaje mínimo de similitud entre los pares para mostrar en el grafo.
- **-q, --quiet**. No se muestra ninguna información, aparte del informe final. Útil para redirigir el informe a otro fichero.

Capítulo 4

Diseño

En este capítulo se explica el porqué se han tomado ciertas decisiones, problemas y dudas que han ido surgiendo a lo largo del desarrollo y se muestra las especificaciones del sistema.

4.1. Consideraciones de Diseño

Cuando se utiliza GitHub en el ámbito académico no hay una sola forma de organizar las clases y queda a criterio de cada profesor. Es por esto que una aplicación *opinionated* o inflexible no puede llegar muy lejos. Como resultado, se ha decidido crear un sistema modulable y extendible a través de extensiones, con la aplicación principal manejando información que pueda resultar útil para cualquier extensión futura, pero que sea lo más genérica posible. Esta plasticidad se intenta conseguir mediante la configuración y las extensiones. Por eso se permite en un buen número de puntos de la configuración, la existencia de campos vacíos y que algunos de estos sean expresiones regulares. De esta forma, diferentes extensiones, pueden utilizar diferentes estrategias y utilizar los datos disponibles como se crea conveniente, haciendo que cada profesor pueda utilizar el programa sin que su metodología de trabajo se vea condicionada.

También tenemos que tener en cuenta que nuestro público objetivo son profesores y alumnos de ingeniería, programación, estadística y relacionados, que utilizan GitHub y que, por lo tanto, tendrán cierto grado de conocimiento con las tecnologías informáticas y la terminal. A su vez, no todas las aplicaciones se van a ver beneficiadas de tener una interfaz gráfica en el navegador y queremos complementar a GitHub Classroom, no sustituirlo, por lo que construir el ecosistema en la terminal es la mejor opción.

Las extensiones se subirán a la plataforma de GitHub con el prefijo `gh-edu-`. Se ha elegido tener un [espacio de nombre](#) o prefijo, para diferenciar las extensiones de aquellas extensiones de `gh` que no estén creadas con fines docentes o no tengan intención de aprovecharse del ecosistema.

En un intento de mantener que el ecosistema no sea más complicado de lo necesario, especialmente la interfaz, las peticiones a la API de GitHub solo funcionaran con la rama por defecto.

4.2. El Núcleo de `gh-edu`

Construir sobre `gh` extension comandos homónimos como `install` o `remove`, ha sido importante para evitar la reimplementación de un gestor de extensiones desde cero,

lo cual hubiese sido excesivo cuando solo se busca tener un mayor control sobre el estado del sistema. También gracias a esto, los creadores de extensiones pueden crear las extensiones en el lenguaje que crean convenientes. Si se trata de un lenguaje interpretado, será suficiente con tener un script escrito en bash con el mismo nombre que el del repositorio de GitHub, que se encargue de ejecutar el fichero principal. Para lenguajes compilados es necesario subir los binarios correspondientes al apartado de Releases del repositorio y el comando `gh extension install` que se ejecuta internamente se encargará de descargarlo.

```
#!/bin/bash
set -e

# Determine if an executable is in the PATH
if ! type -p node >/dev/null; then
    echo "Node not found on the system" >&2
    exit 1
fi

SCRIPT_DIR="$( cd -- "$( dirname -- "${BASH_SOURCE[0]}" )" &> /dev/null && pwd )"
# Pass arguments through to another command
node ${SCRIPT_DIR}/js/gh-edu.js "$@"
```

Figura 4.1: Diseño. Ejemplo de un script para ejecutar una extensión creada con lenguaje interpretado

```
> tree ~/.local/share/gh/extensions/gh-edu-plagiarism
/home/cristo/.local/share/gh/extensions/gh-edu-plagiarism
├── gh-edu-plagiarism
└── manifest.yml

0 directories, 2 files
```

Figura 4.2: Diseño. Ejemplo de contenido de una extensión creada con lenguaje compilado

4.2.1. El fichero de Configuración data.json

```
{
  "defaultOrg": "gh-cli-for-education",
  "cache": {
    "orgs": {
      "gh-cli-for-education": {
        "members": [
          "crguezl",
          "algorithms-ull",
          "GGCristo"
        ]
      }
    }
  },
  "commands": {
    "data": {
      "originalName": "gh-cli-for-education/gh-edu-data",
      "lastCommit": "2e5040ff",
      "teams": [
        {
          "url": "https://github.com/GGCristo",
          "email": "alu0101204512@ull.edu.es",
          "name": "Cristo-García-González",
          "id": "alu0101204512",
          "login": "ggcristo"
        },
        {
          "url": "https://github.com/GGCristo",
          "email": "alu0101204512@ull.edu.es",
          "name": "Manolo_de_los_santos-Pérez-Galdós",
          "id": "alu0102340234",
          "login": "galdozz"
        },
        {
          "url": "https://github.com/GGCristo",
          "email": "alu0101204512@ull.edu.es",
          "name": "Pepe-Recio",
          "id": "alu1234567890",
          "login": "GGCristo"
        }
      ]
    },
    "plagiarism": {
      "originalName": "gh-cli-for-education/gh-edu-plagiarism",
      "lastCommit": "7fbb40ac"
    }
  },
  "identifierR": "alu[0-9]{10,10}",
  "assignmentR": "testing-.*",
  "teamR": "(?<name>.*?)(?<id>.*?)(?<login>.*[^\s]*)"
}
```

Figura 4.3: Ejemplo de data.json. El comando data tiene su propio campo: teams

El fichero data.json es bastante importante. Se trata de un único fichero que actúa como memoria compartida para el intercambio de información entre el usuario, el core y las extensiones. Las extensiones no deben realizar operaciones de lectura, a excepción de una parte dedicada que tiene cada una para guardar sus propios datos (véase 4.3). Cabe recordar que este fichero está ubicado en `~/config/gh-edu/data.json` en sistemas Unix y siempre está bajo gestión de versiones con Git.

Tener todos los datos y configuraciones en un único fichero trae la ventaja de la portabilidad. Los usuarios pueden tener un repositorio privado con el nombre de `gh-edu-profile` y así tener siempre una copia asegurada que puede utilizarse en diferentes máquinas.

El esquema mínimo necesario ya se pudo ver en la figura 2.1

4.2.2. Integridad de los Datos

Una extensión maliciosa o mal implementada podría corromper el fichero data.json, afectando a otras extensiones. Al tratarse de un fichero, cualquier agente externo ya podía

corromperlo, pero la probabilidad de que ocurra aumenta con el número y la calidad de las extensiones. Para intentar paliar esta posibilidad, siempre que se ejecute cualquier comando, el núcleo comprueba que `data.json` es un fichero JSON válido y la validez de sus campos (véase 5.3). También en caso de error, si el error está relacionado por un campo incorrecto, como tener registrado una organización que no existe o a la cual no se pertenece, se muestra por pantalla el campo incorrecto que provocó el fallo. Junto al apoyo dado por el sistema de versiones, se espera que estas medidas sean suficientes para mejorar la robustez del sistema.

4.2.3. Control del Versionado del Sistema

Otra duda que surge de cara al futuro es como se va a controlar la evolución del controlador, de `data.json` y como las extensiones deberían reaccionar a dichos cambios. El problema surge cuando se introducen cambios en el controlador y el fichero `data.json` que son incompatibles con el pasado. En tal caso el número de mayor de la versión del controlador debería haberse incrementado. Puede ocurrir entonces que una versión de una extensión/plugin de `gh-edu` que funcionaba con una versión anterior del controlador, deje de funcionar.

El simple uso del [versionado semántico](#), no resuelve el problema debido a la posibilidad de un conflicto de dependencias en el que una extensión depende de una versión de `data.json` en específico y otra extensión depende de una diferente. Normalmente con el versionado semántico se tienen las diferentes versiones necesarias de un mismo software, pero esa solución va en contra de la especificación de tener un único fichero.

Obsérvese que el fichero `data.json` contiene el número de versión semántica del controlador actual:

```
$ jq '.version' ~/.config/gh-edu/data.json
"0.7.0"
```

y que dicho número de versión puede ser consultado por cualesquiera extensiones para determinar su compatibilidad con el núcleo/core.

En general se intentará mantener la retro-compatibilidad una vez el sistema alcance la estabilidad. Se romperá dicha compatibilidad únicamente en el caso de que sea estrictamente necesario para solucionar un error, notificando de esto a los desarrolladores de extensiones. El fichero `data.json` se mantiene en un repositorio aparte bajo control de versiones. Esta característica debería facilitar la recuperación del sistema en caso de corrupción del fichero o error.

4.2.4. El Manejo de la Localidad de los Datos: cache

Algunos comandos pueden llevar tiempo en ejecutarse, especialmente en organizaciones grandes. Es por eso que cierta información se guarda en la entrada cache del fichero `data.json` (véase figura 2.1) y solo se busca y actualiza usando las APIs cuando la caché no tenga la información necesaria o el usuario desea actualizarla, lo que hará con el comando `gh edu update -c`.

La caché es un mecanismo de optimización algo peligroso si no se trata con cuidado, es por eso que las extensiones no deberían de escribir en ella. Las extensiones pueden implementar su propia caché en su propia zona asignada. Cabe mencionar que no todos

File 1	File 2	Lines Matched
/tmp/2767945551_gh-edu-plagiarism/testing-pablo-escobar/(78%)	/tmp/2767945551_gh-edu-plagiarism/testing-leslie-lamport/(78%)	26
/tmp/2767945551_gh-edu-plagiarism/testing-pablo-escobar/(45%)	/tmp/2767945551_gh-edu-plagiarism/testing-cristo-garcia-gonzalez/(52%)	15
/tmp/2767945551_gh-edu-plagiarism/testing-leslie-lamport/(45%)	/tmp/2767945551_gh-edu-plagiarism/testing-cristo-garcia-gonzalez/(52%)	15

Any errors encountered during this query are listed below.

Figura 4.4: Resultado de MOSS sin procesar

los datos se almacenan en la caché, de momento solo los nombres de las organizaciones y los miembros pertenecientes a ellas son guardados. Se ha elegido estos datos y no otros porque muy rara vez una organización se elimina o renombra y lo mismo pasa con los alumnos o miembros de la organización.

4.3. La Extensión gh-edu-plagiarism

La extensión `plagiarism` fue creada debido a que es uno de los tipos de extensión que más solicitan los profesores en el foro GitHub Classroom Community.

Se ha diseñado para que sea intuitivo y se puede usar tanto de forma interactiva como en un script, utilizando `fzf` solo cuando los datos pasados por parámetros no son suficientes.

El servicio esencial de `plagiarism` es MOSS (Measure Of Software Similarity) [18] creado y ubicado en la universidad de Stanford. Se trata de un servicio que recibe código (en ficheros o directorios) y nos devuelve un enlace con un informe por cada combinación de pares posible.

MOSS es un servicio relativamente popular que lleva usándose desde 1994. Se eligió MOSS por encima de otras tecnologías por varios motivos:

1. **Compara cada tarea con el resto de tareas subidas.** Las alternativas comparan el código con código subido en internet. Algunos profesores no consideran plagio que ciertos fragmentos de código estén sacados de internet, incluso se llega a valorar positivamente.
2. **Utiliza un algoritmo simple.** Que el algoritmo sea relativamente simple trae muchas ventajas. Por un lado, la ejecución es rápida, algo importante si tenemos en cuenta que se tiene que realizar $\frac{n!}{(n-2)!2!}$ análisis. Por otro lado, los creadores afirman que después de años de servicio no se les ha informado de falsos positivos [20] (apartado 5.2 Plagiarism detection).

Un algoritmo simple también es suficiente para detectar técnicas comunes realizadas por los alumnos que se copian, como cambiar de nombre las variables, poner espacios en blanco o mover bloques de código de su posición original. Esta extensión fue creada para ayudar al profesor a concluir quien puede estar copiándose. No se debe confiar ciegamente en sus resultados.

Otra dependencia menos importante es `mossum` [19], un script hecho en python que tiene como input n links generados por MOSS y que genera un grafo con los porcentajes

de similitud entre las asignaciones.

La figura 4.5 muestra un diagrama de como funciona la extensión. Ha sido simplificado, ya que el código también se encarga de comprobar que se cumple con los requisitos, hace una configuración inicial y de forma paralela se comprueba en todo momento si ha ocurrido un error y se controla apropiadamente. Esto se conoce como **degradación gradual** y específicamente intenta mostrar algún resultado, incluso si otro falla, en específico si no consigue devolver el informe ni el grafo, retornará un enlace, con una lista de reportes sin procesar, los mismos que devuelve el servidor de MOSS (figura 4.4).

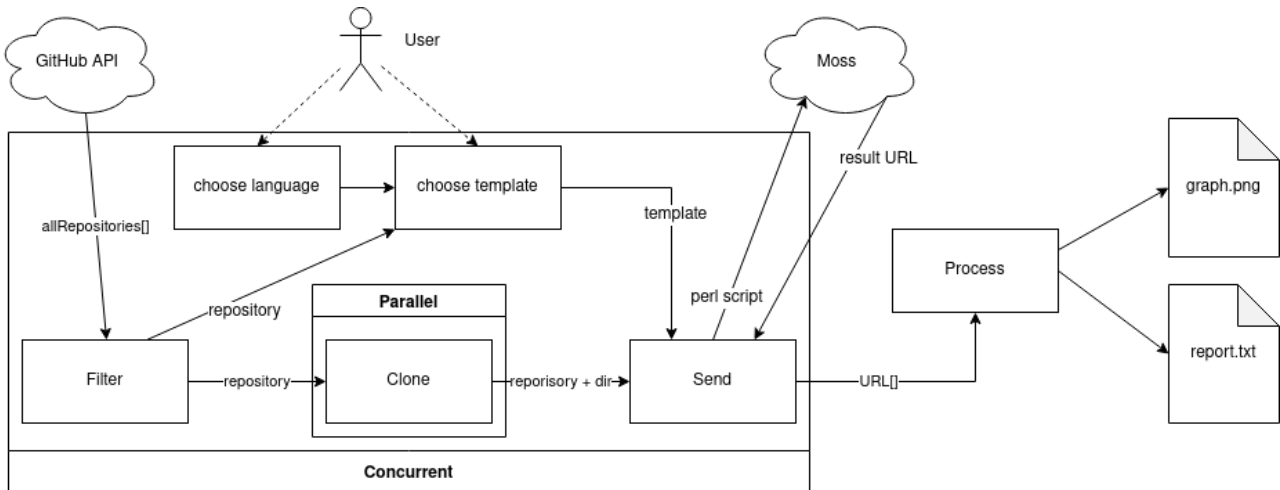


Figura 4.5: Diagrama de funcionamiento de plagiarism

Entrando en detalle, la aplicación

1. **GitHub API.** Empieza pidiendo todos los repositorios de la organización.
2. **Filter.** Después pasa por un filtrado para quedarse con los repositorios que si interesan que sería aquellos relacionados con la tarea actual de la asignatura, utilizando expresiones regulares.
3. **Clone.** Los repositorios se van clonando en paralelo en un directorio temporal del sistema.
4. **Send.** A medida que se van clonando se les adjunta la dirección absoluta y se va preparando para ser enviados al servidor de MOSS a través de su script.
5. **Process.** Una vez MOSS haya devuelto el enlace con los informes. ejecutamos el programa `mossum` para la generación del grafo y la obtención del informe separado por pares de alumnos. Cuando el programa termina se muestra por pantalla un informe de todos los pares para que el profesor pueda ver las diferencias y determinar si de verdad hubo plagio (figura 4.6) y el susodicho grafo (figura 4.7).

Desde que el programa empieza se le pide al usuario información necesaria, en el caso de que no lo haya especificado por línea de comandos, como puede ser

- el lenguaje de programación utilizado en el código o
- una plantilla que el profesor haya proporcionado a los alumnos y sirva de base (en caso de haberla)

```

Report:
Pair: testing-pablo-escobar and testing-leslie-lamport
moss_07-07-2022_114233: http://moss.stanford.edu/results/4/3989576511989/match0.html

Pair: testing-pablo-escobar and testing-cristo-garcia-gonzalez
moss_07-07-2022_114233: http://moss.stanford.edu/results/4/3989576511989/match1.html

Pair: testing-leslie-lamport and testing-cristo-garcia-gonzalez
moss_07-07-2022_114233: http://moss.stanford.edu/results/4/3989576511989/match2.html

```

Figura 4.6: Informe final dado por plagiarsim, con enlace para comprobar el código manualmente

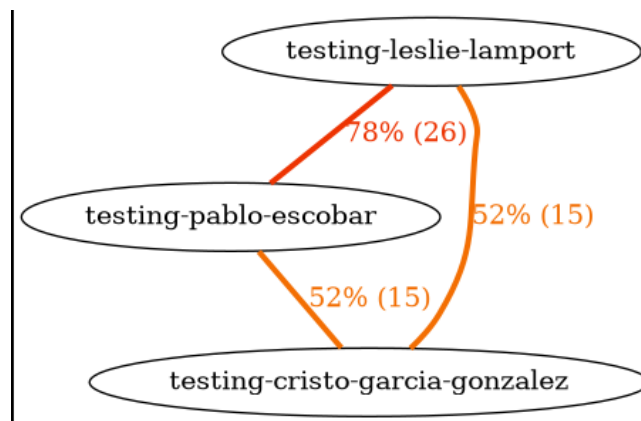


Figura 4.7: Plagiarism. Grafo resultante mostrando los niveles de similitud

Nota: Debido a la política de privacidad de MOSS los informes pueden estar disponibles por 14 días, aunque también se avisa que existe la posibilidad de que se eliminen antes para poder liberar memoria de los servidores si se alcanza cierto límite no estipulado.

La extensión plagiarism funciona bastante bien con el sistema gh-edu, pero también se ha diseñado para que sea posible su uso de forma independiente.

Capítulo 5

Implementación

El objetivo de este capítulo es demostrar las partes importantes o de interés de la implementación y como se ha logrado la realización de lo estipulado en capítulos anteriores.

5.1. El controlador

Como se ha comentado en capítulos anteriores, la mayoría de comandos se aprovecha de los comandos ya proporcionados por gh. Como es el caso de `gh edu install`. Después de determinar si la extensión es *first-party* o *third-party* (extensión independiente de la organización `gh-cli-for-education`), se obtiene la dirección del repositorio de la extensión, se invoca `gh extension install` y se añade a `data.json`.

```
utils.print(isQuiet, `Installing ${plugin} ...`);
let { stderr, code } = shell.exec("gh extension install " + url, { silent: true });
if (code !== 0) {
  process.stderr.write(stderr.replaceAll("edu-", ""));
  return;
}
utils.print(isQuiet, "Plugin installed in system");

utils.print(isQuiet, "Setting up configuration...");
const org = plugin.split('/', 1)[0];
const defaultOrg = utils.runCommand(`gh api /repos/${org}/gh-edu-${installedName} | jq .default_branch`, true);
const lastCommit = shell.exec(`gh api /repos/${plugin}/commits/${defaultOrg}`, { silent: true });
if (lastCommit.code !== 0) {
  console.error(chalk.red("Couldn't get default branch for . Skipping last commit info"));
}
config.commands = {
  [installedName]: { originalName: plugin, lastCommit: JSON.parse(lastCommit).sha?.substring(0, 8) },
  ...config.commands
}
updateJSON(config);
```

Figura 5.1: Implementación. Código para instalar extensiones

En la figura 5.1 vemos que después de instalar la extensión, hacemos una llamada API REST para saber cuál es la rama por defecto. Al tener esta información, podemos buscar cuál es el [hash](#) del último *commit*, para actualizar el campo `lastCommit`.

Nota: `utils.print()` es una función muy simple para determinar si el mensaje debería imprimirse o no, dependiendo del valor del flag `-quiet`.

Para que `commander` acepte únicamente las extensiones instaladas, se ha escrito el siguiente código (figura 5.2):

```

/** Add installed third party plugins */
for (const plugin of Object.keys((config.commands))) {
  program
    .command(plugin)
    .allowUnknownOption(true)
    .helpOption(false)
    .action( (_, commandObj) => {
      const { code } = shell.exec(`gh extension exec edu-${plugin} ${commandObj.args.join(' ')}`, { silent: false
    });
      if (code == 127) // doesn't found the binary
        console.error(`${plugin} is not installed\nPlease use gh edu remove command to remove plugins, or install it
again`);
    })
}

```

Figura 5.2: Implementación. Controlador. Añadir extensiones third-party

Todas las extensiones, que se hayan instalado en el sistema a través de gh-edu, se añadirán como comandos disponibles. También se activa la posibilidad de usar opciones desconocidas y se delega su gestión al subcomando en cuestión. Así mismo se desactiva la ayuda y en el manejo de errores solo se comprueba que el comando haya sido ejecutado, pues naturalmente de estas tareas se tiene que ocupar cada extensión.

Aquí se deja ver la madurez del programa al permitirnos utilizar comandos desconocidos, función que, al momento de escribir estas líneas, todavía no es posible con cobra.

5.1.1. Manejo del único punto de información: data.json

Lo primero que el programa hace al ser ejecutado, es comprobar la validez del fichero. La siguiente figura (5.3) muestra los estados por los que se pasa para lograrlo.

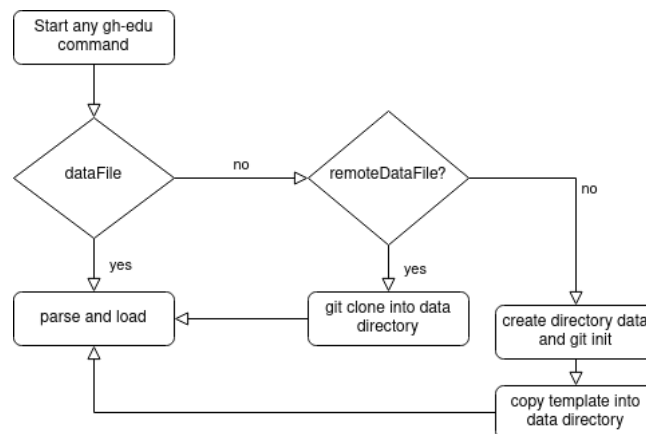


Figura 5.3: Obtención del archivo data.json

Es primer paso es comprobar si se encuentra en la localización correspondiente (~/.config/gh-edu/data.json). De no ser así, se intenta conseguirlo desde el repositorio gh-edu-profile. Como última instancia se crea un fichero nuevo, partiendo de una plantilla que contiene todos los campos necesarios.

Sabiendo que el fichero está disponible, pasamos a validar su contenido. Primero se **deserializa** para comprobar que es un fichero JSON válido. Acto seguido pasamos a realizar una comprobación simple de los campos, comprobando que estén en el fichero y que las **expresiones regulares** sean válidas.

```

function validateConfig(config: configType) {
  if (config.defaultOrg === undefined) throw "No defaultOrg field"
  if (config.cache === undefined) throw "No cache field"
  if (config.cache.orgs === undefined) throw "No orgs field in cache"
  for (const org in config.cache.orgs) {
    if (!Array.isArray(config.cache.orgs[org].members)) {
      throw "cache.orgs.<orgName> must exists and be an array"
    }
  }
  if (config.commands === undefined) throw "No commands field"
  for (const command in config.commands) {
    if (config.commands[command].originalName === undefined) throw "All commands must have an originalName"
    if (config.commands[command].lastCommit === undefined) throw "All commands must have a lastCommit field"
  }
  if (config.assignmentR === undefined) throw "No assignmentR field"
  if (!isValidRegex(config.assignmentR)) throw `assignmentR. ${config.assignmentR} is not a valid regex`
  if (config.teamR === undefined) throw "No teamR field"
  if (!isValidRegex(config.teamR)) throw `teamR. ${config.teamR} is not a valid regex`
  if (config.identifierR === undefined) throw "No identifierR field"
  if (!isValidRegex(config.identifierR)) throw `identifierR. ${config.identifierR} is not a valid regex`
  if (!config.version) {
    config.version = JSON.parse(fs.readFileSync(path.join("utils", "data.template.json"), {encoding:
    "utf-8"})).version;
    console.log(chalk.yellow(`No version in this data file\nSetting version found in template:
    v${config.version}`));
    updateJSON(config);
  }
}
}

```

Figura 5.4: Implementación. data.json. Comprobamos que los campos son válidos

Cada vez que se quiera actualizar la información, se actualiza la variable donde se cargó el fichero, se [serializa](#) y se escribe de vuelta al fichero.

5.2. Extensión minimalista: gh-edu-view

A la hora de escribir una extensión para gh-edu en JS, se procede de la misma forma que con una extensión gh. Deben crearse los ficheros gh-edu-view/gh-edu-view (script bash) y gh-edu-view/gh-edu-view.js (programa JS).

Para intentar conseguir los identificadores relacionados a los alumnos, con nada más que la información proporcionada con GitHub, no nos queda otra opción que buscar en todos los campos, donde dicha información podría estar disponible, y devolver un *array* con todos los posibles resultados.

```

function getPossibleID(data, identifierR) {
  let regex = new RegExp(identifierR);
  if (!regex.global)
    regex = new RegExp(regex.source, regex.flags + "g");
  let posiblesID = []; // [[]]
  for (const { login, name, bio, email, url } of data) {
    let memberPossibleID = [];
    let match;
    if (match = login.match(regex))
      memberPossibleID = memberPossibleID.concat(match);
    if (match = name?.match(regex))
      memberPossibleID = memberPossibleID.concat(match);
    if (match = bio?.match(regex))
      memberPossibleID = memberPossibleID.concat(match);
    if (match = email?.match(regex))
      memberPossibleID = memberPossibleID.concat(match);
    if (match = url.match(regex))
      memberPossibleID = memberPossibleID.concat(match);
    posiblesID.push(memberPossibleID);
  }
  return posiblesID;
}

```

Figura 5.5: Implementación. gh-edu-view. Obtención de posibles identificadores

Toda la información proporcionada por el comando `members` se puede conseguir con una simple petición en GraphQL (figura 5.6).

```

const query = (org) => `
  query ($endCursor: String) {
    organization(login: "${org}") {
      membersWithRole(first: 100, after: $endCursor) {
        pageInfo {
          endCursor
          hasNextPage
        }
        nodes {
          login,
          name,
          bio,
          email,
          url,
          avatarUrl
        }
      }
    }
  }
`

```

Figura 5.6: Implementación. gh-edu-view. Petición para conseguir diversos campos de cada alumno

5.3. La librería `shelljs` y el manejo de E/S: `gh-edu-data`

Uno de los defectos de `shelljs` que ya se ha comentado en el apartado de **Tecnologías** (2.3.2) es como no permite el uso de STDIN o tuberías. Algo importante en este proyecto, pero especialmente en esta extensión por el uso avanzado de `ffz` y `jq`. Estas dos tecnologías, a pesar de ser completamente independientes la una a la otra, pueden trabajar conjuntamente gracias a su diseño flexible y minimalista a través de tuberías,

haciendo posible la previsualización dinámica de los elementos en el comando `log` (figura 3.10)

Este comportamiento es requerido, ya que no es posible pasar un objeto JS o un `array` al input de un programa que todavía no ha sido ejecutado. Solo se puede usar cadenas de texto. Lo ideal es ejecutar el comando, y de forma asíncrona ir procesando la estructura de datos, enviando cadenas de texto, como se hace en `gh-edu-plagiarism` (gracias a `cmd.StdInPipe`).

Hasta que la [incidencia 424](#) no sea resuelta, se ha tenido que realizar una solución alternativa utilizando ficheros temporales. La idea consiste en formatear el objeto o `array` y escribir el resultado en un fichero, de esta forma podemos leer el fichero y transmitir esa información por medio de una tubería a otro comando en el mismo momento en el que se ejecuta el comando.

5.4. Concurrencia con Go y CSP: `gh-edu-plagiarism`

A diferencia del resto de extensiones `plagiarism` está implementado en Go. Uno de los motivos del cambio de lenguaje es demostrar como se puede desarrollar extensiones en cualquier lenguaje sin muchas dificultades.

También, como se comentó en **Tecnologías** (2.4) este programa es altamente concurrente y paralelo. Go cuenta con un modelo de concurrencia bastante particular basado en el trabajo teórico de Tony Hoare *Communicating Sequential processes* (CSP) [21].

Para entender las explicaciones de esta sección es necesario conocer dos conceptos: las gorutinas y los canales. Las gorutinas son [corrutinas](#) independientes que se ejecutan en hilos verdes (*green threads*), los cuales son hilos emulados generados en tiempo de ejecución, estos hilos se multiplexan a hilos reales del procesador a través del Go `scheduler`, que determina cuanto tiempo debe estar cada hilo verde consumiendo CPU.

Los canales son la estructura de datos que nos permite enviar información de una gorutina a otra y también sirven de elemento sincronizador, cuando se envía un elemento, la gorutina no continua hasta que el otro lado (el emisor) haya recibido la información y viceversa.

La explicación de la figura 4.5 en términos de código sería la siguiente: Cada bloque dentro del módulo `Concurrent` es una gorutina que se ejecuta de forma independiente y cada flecha pertenece a un canal, que sincroniza y a su vez transmite información de una gorutina a otra.

La gorutina `clone` va creando a su vez gorutinas a medida que más repositorios se vayan filtrando. Debido a que las gorutinas apenas consumen recursos, está bien eliminar y crear tantas como queramos. No obstante, como estamos realizando programación paralela, tenemos que tener cuidado de que no haya muchas más gorutinas corriendo que núcleos disponibles en el procesador, pues entonces conseguiríamos el efecto contrario al deseado, ralentizando toda la aplicación debido al constante cambio de contexto entre hilos. Para limitar la creación de gorutinas al número de núcleos disponibles se ha utilizado un semáforo. Un semáforo es una variable o estructura de datos que permite una cantidad arbitraria de procesos. Un semáforo binario es a efectos prácticos un `lock` o `mutex`.

```

sem := make(chan empty, runtime.NumCPU())
for repo := range reposC {
    sem <- empty{}
    go func(repo repoObj) {
        // clone repo
        <-sem
    }(repo)
}

```

Figura 5.7: Paralelismo limitado con un s emaforo

El c odigo que lo implementa (figura 5.7) ha sido simplificado para explicar el funcionamiento del paralelismo y el s emaforo.

El canal `sem` es un canal con buffer, esto quiere decir que no para el flujo (no sincroniza) hasta que el buffer se llene. El tama o de dicho buffer es `runtime.NumCPU()` que retorna el n mero de n cleos disponibles al ejecutar el programa. En cada iteraci n del bucle `repo` contiene un nuevo repositorio proveniente del m dulo `Filter` y enviamos un elemento al canal `sem`, el elemento en cuesti n es irrelevante, estamos usando el canal por sus propiedades de sincronizaci n, no para la transferencia de datos. Despu s ejecutamos una gorutina an nima que se encarga de clonar el repositorio, cuando termine escucha y descarta del canal `sem`.

De esta forma, si `sem` est  lleno significa que ya hay tantos procesos ejecut ndose como n cleos hay disponibles, por lo que el programa queda en espera hasta que quede un hueco en el s emaforo.

```

go filter(allRepos, sendToCloneC, selectTemplateC, errC)
selectLangC := make(chan string, 1)
go func() { // fzf (optional)
    selectLanguage(selectLangC, errC)
    selectTemplate(selectTemplateC, selectedTemplateC, errC)
}()
clonedC := make(chan repoObj)
go clone(sendToCloneC, clonedC, errC)
go send(clonedC, selectedTemplateC, selectLangC, errC) // send also close errC
return <-errC

```

Figura 5.8: C digo general de plagiarism

Capítulo 6

Conclusiones y líneas futuras

“Software Engineering Is Programming Integrated Over Time”

Lo que más resalto en este trabajo de fin de grado ha sido la tarea de diseñar un ecosistema modulado desde cero con unas metas marcadas y con el ideal de crear un software que no necesite de cambios en su estructura para crecer, que sea simple e intuitivo, pero útil y que permita la colaboración. Conseguir todas estas características ha requerido una amplia etapa de diseño y prototipado donde las ideas se implementaban y descartaban por igual.

También otro apartado que consumió bastante tiempo fue el manejo de errores. Al ser una aplicación que realiza muchas llamadas, lee y escribe de ficheros y le pide *input* al usuario, hay muchos puntos donde la aplicación puede fallar. Manejar estos errores y pulir la aplicación en general es trabajo que no genera un resultado inmediato, pero hace que la aplicación sea más mantenible y estable.

Por otro lado, me siento muy afortunado de haber podido trabajar con tecnologías con las que me siento cómodo como JavaScript, Go y fzf, pero han sido más las que he aprendido y dada la buena experiencia que he tenido con: jq, gh cli y GraphQL comentar que seguramente las utilice en futuros proyectos.

No obstante, me apena no haber podido cumplir con el objetivo de interoperabilidad entre comandos. El comando *view* se podría beneficiar enormemente si pudiese leer de la *standard input* para recibir información de un comando como *data*, pero preferí enfocar el tiempo en mejorar la calidad y usabilidad de todo el ecosistema.

El trabajo de fin de grado está terminado, pero el proyecto puede crecer tanto como desee. Este tipo de proyecto requiere de trabajo constante y prolongado para que tenga acogida en la comunidad. Mis intenciones son seguir contribuyendo a este proyecto y tomarlo como mi proyecto personal y mi vía de contribución al open source.

Capítulo 7

Summary and Conclusions

“Software Engineering Is Programming Integrated Over Time”

What stands out the most in this final degree project has been the task of designing a modular ecosystem from scratch with some set goals and with the ideal of creating a software that does not need changes in its structure to grow, that is simple and intuitive, but useful and that allows collaboration. Achieving all these features required an extensive design and prototyping stage, where ideas were implemented and discarded alike.

Also, another time-consuming section was error handling. Being an application that makes a lot of calls, reads and writes from files and asks for user input, there are many points where the application can fail. Handling these errors and polishing the application in general is work that does not generate an immediate result, but it does make the application more maintainable and stable. On the other hand, I feel very fortunate to have been able to work with technologies that I feel comfortable with such as JavaScript, Go and fzf, but there have been more that I have learned and given the good experience I’ve had with: jq, gh cli and GraphQL, I’m sure I’ll use them in future projects.

However, I am sorry I have not been able to meet the goal of interoperability between commands. The view command could benefit enormously if it could read from the standard input to receive information from a command such as data, but I preferred to focus my time on improving the quality and usability of the whole ecosystem.

The final degree work is finished, but the project can grow as much as I want. This type of project requires constant and prolonged work in order to be accepted by the community. My intentions are to continue contributing to this project and take it as my personal project and my way to contribute to open source.

Capítulo 8

Presupuesto

Todas las tecnologías usadas son *open source* y gratuitas y se presupone que el ordenador usado es personal, por lo que no se incluye en el presupuesto
El hipotético saldo son 15 euros la hora

Coste	Tipos	Descripción
4500	Diseño y planificación	Incluyendo el tiempo invertido en investigación de las nuevas tecnologías necesarias y el diseño y planificación de la aplicación se ha consumido un tiempo sustancial de 300 horas
3000	Desarrollo de los prototipos y aplicación final	Aproximadamente se ha dedicado unas 200 horas
495	Elaboración del informe y conclusiones	Al tratarse de un ecosistema <i>open source</i> lanzado a un público general, la documentación es importante, por lo que se ha desarrollado no solo este informe. si no documentación para usuario y desarrolladores en los respectivos repositorios. Necesitando un total de 33 horas

Tabla 8.1: Tabla del presupuesto

Esto supone un presupuesto total de, 7995 euros en un periodo de 3 meses

Apéndice A

Repositorios del ecosistema

Se adjuntan los enlaces a los repositorios de GitHub de los diferentes códigos en los que se ha trabajado.

- El código de gh-edu se encuentra en
<https://github.com/gh-cli-for-education/gh-edu>
- El código de gh-edu-view:
<https://github.com/gh-cli-for-education/gh-edu-view>
- El código de gh-edu-data:
<https://github.com/gh-cli-for-education/gh-edu-data>
- El código de gh-edu-plagiarism:
<https://github.com/gh-cli-for-education/gh-edu-plagiarism>
- Un ejemplo de extensión externa a la comunidad [gh-cli-for-education](#) puede verse en
<https://github.com/crguezl/gh-edu-browse>
- El código de la memoria:
<https://github.com/gh-cli-for-education/TFG-2122-Cristo-Garcia-Gonzalez>

Glosario

AOT En informática, la **compilación anticipada** es el acto de compilar un lenguaje de programación (a menudo) de alto nivel en un lenguaje (a menudo) de bajo nivel antes de la ejecución de un programa, normalmente en tiempo de compilación, para reducir la cantidad de trabajo necesario en tiempo de ejecución. De hecho, dado que todas las compilaciones estáticas se realizan técnicamente con antelación, esta expresión en particular se utiliza a menudo para destacar algún tipo de ventajas de rendimiento del acto de dicha precompilación. El acto de compilar Java a bytecode Java es por tanto raramente referido como AOT ya que es normalmente un requisito, no una optimización. [5](#)

API Una **interfaz de programación de aplicaciones** es una manera de que dos o más programas informáticos se comuniquen entre sí. Es un tipo de interfaz de software que ofrece un servicio a otras piezas de software. [2](#)

caché Una **cache** es un componente de hardware o software que almacena datos para que las futuras solicitudes de esos datos puedan ser atendidas más rápidamente; los datos almacenados en una caché pueden ser el resultado de un cálculo anterior o una copia de datos almacenados en otro lugar. [5](#), [20](#)

CLI Es un tipo de interfaz de usuario de computadora que permite a los usuarios dar instrucciones a algún programa informático o al sistema operativo por medio de una línea de texto simple. [5](#)

corrutina Las **corrutinas** son componentes de programas informáticos que generalizan las subrutinas para la multitarea no preferente, permitiendo suspender y reanudar la ejecución. [35](#)

degradación gradual La **tolerancia a los fallos** es la propiedad que permite a un sistema seguir funcionando correctamente en caso de fallo de uno o varios de sus componentes. A diferencia de un sistema diseñado ingenuamente, en el que incluso un pequeño fallo puede provocar una avería total. [29](#)

deserializar Extraer una estructura de datos de una serie de bytes. [32](#)

DSL Un lenguaje específico de dominio (DSL) es un lenguaje que está dirigido a resolver un tipo particular de problema. El uso de DSL es muy común en informática: CSS, expresiones regulares, make, ant, SQL, muchos bits de Rails, expectativas en JMock, el lenguaje graphviz, los archivo de configuración de strut, etc. . [9](#)

espacio de nombre En informática, un **espacio de nombres** es un conjunto de signos (nombres) que se utilizan para identificar y referirse a objetos de diversos tipos. Un

espacio de nombres garantiza que todos los objetos de un conjunto determinado tengan nombres únicos para que puedan ser fácilmente identificados. [24](#)

expresión regular Una **expresión regular** (abreviada como regex o regexp) es una secuencia de caracteres que especifica un patrón de búsqueda en un texto. Los algoritmos de búsqueda de cadenas suelen utilizar estos patrones para las operaciones de "búsqueda" o "búsqueda y sustitución" de cadenas, o para la validación de entradas. [32](#)

hash Un **hash** (que también puede llamarse un condensado o una firma) es un valor calculado a partir de un valor diferente. En la gran mayoría de los casos, este valor está representado en forma de una cadena de caracteres hexadecimales. Los hashes suelen utilizarse para verificar que un archivo no se ha visto corrompido o bien para autenticar a un usuario sin tener que almacenar su contraseña en claro. [31](#)

JIT just-in-time compiler. También conocido en español como **compilación en tiempo de ejecución**. Es una forma de ejecutar el código informático que implica la compilación durante la ejecución de un programa (en tiempo de ejecución) en lugar de antes de la ejecución. [5](#)

patrón estrategia En programación informática, el patrón de estrategia (también conocido como patrón de política) es un patrón de diseño de software de comportamiento que permite seleccionar un algoritmo en tiempo de ejecución. En lugar de implementar un único algoritmo directamente, el código recibe instrucciones en tiempo de ejecución sobre cuál de una familia de algoritmos utilizar. [10](#)

power users Un **usuario avanzado** es un usuario de ordenadores, software y otros dispositivos electrónicos, que utiliza características avanzadas del hardware informático, sistemas operativos, programas o sitios web que no son utilizados por el usuario medio. Un usuario avanzado puede no tener amplios conocimientos técnicos de los sistemas que utiliza, pero se caracteriza más bien por la competencia o el deseo de hacer el uso más intensivo de los programas o sistemas informáticos. [2](#)

serializar Es el proceso de traducir una estructura de datos o el estado de un objeto a un formato que pueda ser almacenado (por ejemplo, en un archivo o en un búfer de datos de la memoria) o transmitido (por ejemplo, a través de una red informática). [33](#)

Bibliografía

- [1] Inc. GitHub. gh extension. https://cli.github.com/manual/gh_extension, 2022. [Online; accessed 06-July-2022].
- [2] Inc. GitHub. GitHub REST API. <https://docs.github.com/en/rest>, 2022. [Online; accessed 06-July-2022].
- [3] Inc. GitHub. GitHub GraphQL. <https://docs.github.com/en/graphql>, 2022. [Online; accessed 06-July-2022].
- [4] Inc. GitHub. GitHub CLI. <https://cli.github.com/>, 2022. [Online; accessed 06-July-2022].
- [5] Inc. GitHub. gh api. https://cli.github.com/manual/gh_api, 2022. [Online; accessed 06-July-2022].
- [6] Inc. GitHub. gh auth login. https://cli.github.com/manual/gh_auth_login, 2022. [Online; accessed 06-July-2022].
- [7] Junegunn Choi. fzf. <https://github.com/junegunn/fzf>, 2022. [Online; accessed 06-July-2022].
- [8] Mozilla Foundation. JavaScript. <https://developer.mozilla.org/es/docs/Web/JavaScript>, 2022. [Online; accessed 06-July-2022].
- [9] Microsoft Corporation. TypeScript. <https://www.typescriptlang.org/docs/>, 2022. [Online; accessed 06-July-2022].
- [10] Scott Chacon y Git community. Git hook. <https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks>, 2022. [Online; accessed 07-July-2022].
- [11] Inc. npm. npm. <https://docs.npmjs.com/>, 2022. [Online; accessed 06-July-2022].
- [12] Alan AA Donovan and Brian W Kernighan. *The Go programming language*. Addison-Wesley Professional, 2015.
- [13] Chaminda Chandrasekara and Pushpa Herath. Introduction to github actions. In *Hands-on GitHub Actions*, pages 1–8. Springer, 2021.
- [14] Steve Francia. cobra. <https://cobra.dev/>, 2022. [Online; accessed 06-July-2022].
- [15] Steve Francia. viper. <https://pkg.go.dev/github.com/spf13/viper>, 2022. [Online; accessed 06-July-2022].
- [16] Stephen Dolan. jq. <https://stedolan.github.io/jq/manual/>, 2018. [Online; accessed 06-July-2022].

- [17] John Hammersley and John Lees-Miller. overleaf. <https://www.overleaf.com/learn>, 2022. [Online; accessed 06-July-2022].
- [18] Alex Aiken. MOSS. <https://theory.stanford.edu/~aiken/moss/>, 2022. [Online; accessed 07-July-2022].
- [19] Hjalti Magnússon. mossum. <https://github.com/hjalti/mossum>, 2022. [Online; accessed 07-July-2022].
- [20] Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85, 2003.
- [21] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall International Series in Computer Science. Prentice Hall, 1985.