CONTROL DE VERSIONES GIT / GITHUB

¿Por qué un Data Scientist debe aprender Git y GitHub?

- **Control de versiones**

  Gestión del historial de cambio en los códigos.
- **Automatización y reproducibilidad**

  Permite reproducir experimentos y regresar a versiones anteriores.
- **Colaboración eficiente**

  Facilita que múltiples personas trabajen sobre un mismo proyecto.
- **Perfil profesional sólido**

  Es una habilidad muy valorada en el mercado laboral.

CONTROL DE VERSIONES GIT / GITHUB

¿En qué se diferencian?

- **Git** es una herramienta que **se instala en local** y permite gestionar las versiones del código **en mi equipo.**

- Trabaja sobre la terminal, mediante comandos, teniendo un control preciso sobre los cambios

- **GitHub** es una **plataforma online** basada en Git que aloja repositorios y facilita la colaboración entre equipos además de la integración con otras herramientas de desarrollo.

PYTHON DATA SCIENCE MASTERY

CONTROL DE VERSIONES GIT / GITHUB

DS4B

Algunos de los comandos más usados en nuestro trabajo

Setup

GIT INIT

Inicia un nuevo repositorio GIT en la carpeta actual, esto creará el área de ensayo "Staging"

GIT CLONE

Copia un proyecto existente en un repositorio público (incluyendo su historial de cambios)

Inspección

GIT STATUS

Muestra el estado del repositorio, incluyendo archivos modificados y pendientes de confirmación

GIT DIFF

Muestra las diferencias entre el estado actual del código y la última versión guardada

Staging

GIT ADD

Agrega los archivos modificados al "Staging" quedando a la espera de ser confirmados

GIT RESET

Revierte cambios en el área de ensayo "Staging" o en commits anteriores

Ramas

GIT MERGE

Fusiona los cambios de una rama en la rama actual, combinando sus historiales

GIT BRANCH

Crea, lista, renombra o elimina ramas en el repositorio local

Commit

GIT COMMIT

Guarda los cambios registrados en el "Staging" dentro del repositorio local

Sincronizar

GIT PULL

Descarga y fusiona los cambios del repositorio remoto en la rama actual

GIT PUSH

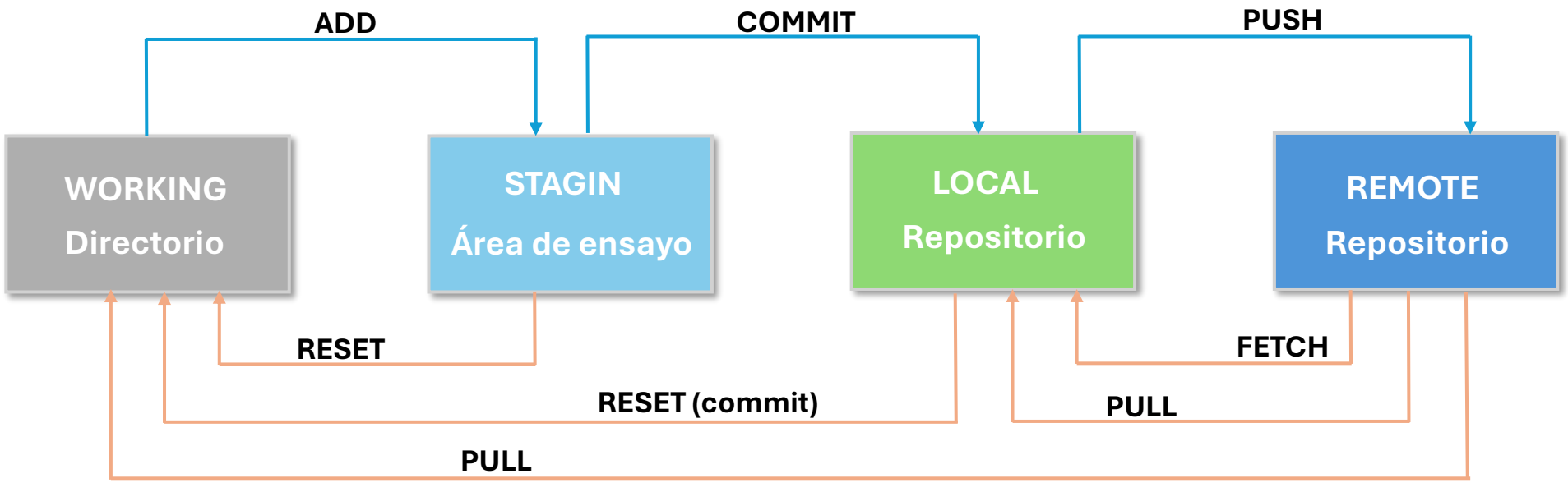
Envía los commit a un repositorio remoto. (origin es el nombre por defecto del repositorio remoto)

GIT FETCH

Descarga cambios del repositorio remoto, pero sin aplicarlos automáticamente

CONTROL DE VERSIONES GIT / GITHUB

Flujo de trabajo con los comandos

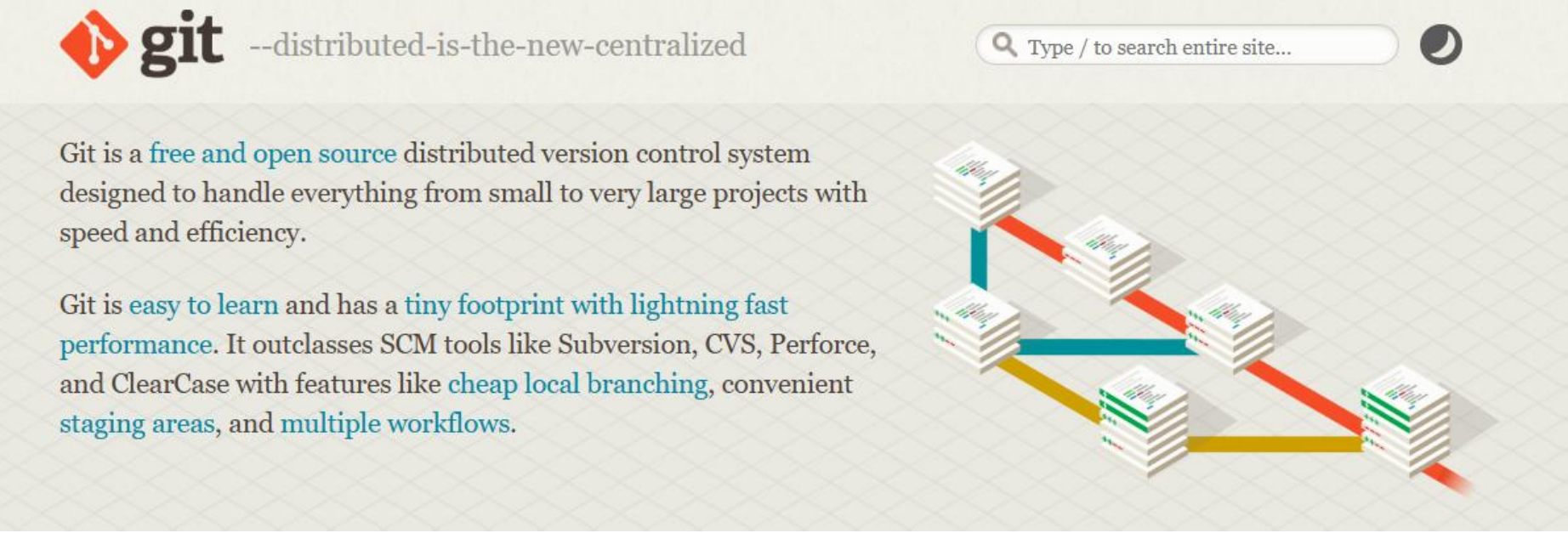


CONTROL DE VERSIONES GIT / GITHUB

Instalación y configuración inicial

- Puedes acceder a la página de descargas mediante el buscador o a través de su página web: ***https://git-scm.com/***

- El proceso es guiado y sencillo, puedes dejar todas las opciones por defecto hasta finalizar.



CONTROL DE VERSIONES GIT / GITHUB

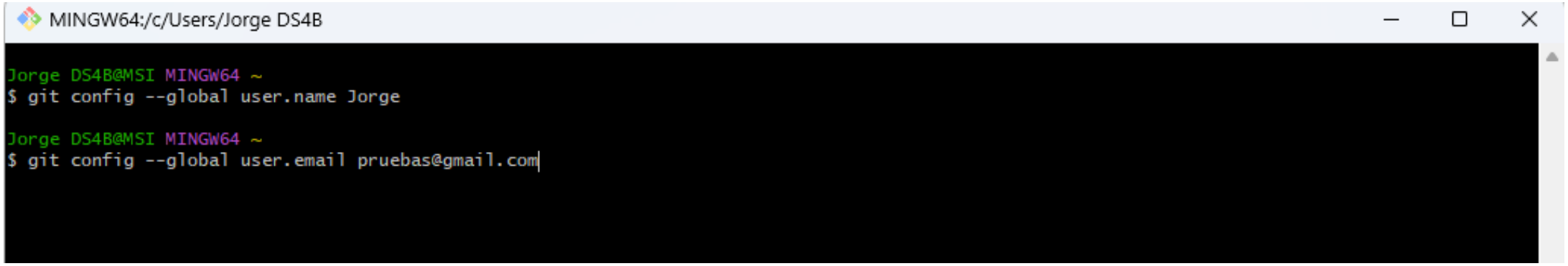
Instalación y configuración inicial

- Git incorpora su propia terminal **Git Bash**, similar a la usada en **macOS/Linux**, pero podemos usar cualquiera de las disponibles en el equipo.

- Dependiendo de la terminal que usemos, debemos aplicar los comandos correspondientes.

- Debemos configurar nuestra identidad una sola vez en la terminal, permitirá identificar nuestros commits claramente, útil principalmente cuando trabajamos en equipo.

git config --global user.name "Tu Nombre"
git config --global user.email tu@email.com

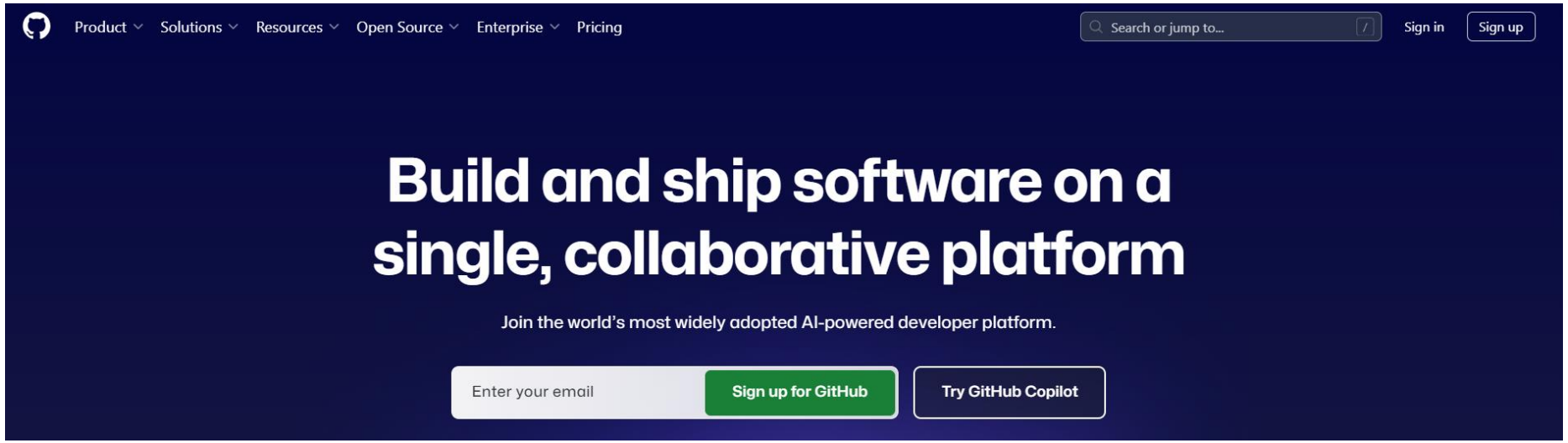


CONTROL DE VERSIONES GIT / GITHUB

Pasos para crear nuestro repositorio

- Puedes acceder a la página mediante el buscador o a través de su página web: ***https://github.com/***

- Pide los datos básicos para crear la cuenta y una confirmación mediante código enviado al mail.



VINCULACIÓN DE GIT CON GITHUB

¿Por qué vincularlos?

- GitHub no es solo una web, es un servidor remoto donde alojar los repositorios y colaborar.

- El trabajo habitual es en nuestro equipo, de forma local, pero tiene sus pros y contras, por ello sincronizamos con GitHub para subir (push) o bajar (clone) código.

Protocolos de autenticación

- **HTTPS:** Es un formato sencillo, pero menos seguro, cada push/pull pide credenciales o token.

- **SSH:** Es un protocolo mas seguro, cifra la comunicación entre nuestro equipo y un servidor. **Requiere generar y gestionar claves**

Enlace a la documentación oficial de GitHub:
https://docs.github.com/es/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

VINCULACIÓN DE GIT CON GITHUB

Claves SSH: pública y privada

- **Clave privada** (id_ed25519): tú la guardas siempre en tu máquina (nunca se comparte).
- **Clave pública** (id_ed25519.pub): la subes a GitHub.

Generar las claves SSH

Comando para Git Bash:

Avanzado

ssh-keygen -t ed25519 -C "tu@email.com" *-f ~/.ssh/id_ed25519_alias*

Comando para generar claves	Algoritmo que encripta	Parámetro modificable	Permite renombrar claves generadas	Alias para reconocer claves

Una vez escribes el comando, te pedirá una contraseña “passphrase”, no se verá nada por pantalla y el proceso se hace dos veces.

VINCULACIÓN DE GIT CON GITHUB

Clave pública (.pub)

- Se reconoce porque es la que empieza con ***ssh-ed25519 AAAA ...*** y acaba con el comentario ***tu@email.com***. Es la que debemos copiar y pegar en GitHub

Cómo ver y copiar la clave completa

Comando Git Bash:

cat ~/.ssh/id_ed25519.pub

Una vez visualizada la clave, tenemos que seleccionar la parte que nos interesa, copiarla y pegarla en GitHub

Foto perfil -> Settings -> SSH and GPG Keys

VINCULACIÓN DE GIT CON GITHUB

Para hacer el push debemos tener previamente el repositorio creado

- Una vez creado, GitHub nos proporciona directamente los comandos que debemos ejecutar en local y así completar la tarea.

git remote add origin git@github.com:TuUsuario/mi-repo.git
git branch -M main
git push -u origin main

Avanzado

Antes de escribir los comandos en Git Bash

- Haremos un paso extra, que es añadir la clave privada al "agente SSH", este proceso **gestiona nuestras claves y sus alias** y le indica al sistema que podemos usarlas para conectarlas a servidores remotos como GitHub

  - ***eval $(ssh-agent -s)*** # Inicializamos el agente

  - ***ssh-add ~/.ssh/id_ed25519_alias*** # Añadimos la clave privada al agente sin el (.pub)

VINCULACIÓN DE GIT CON GITHUB

Evitar subir archivos pesados o confidenciales

- Si trabajamos con Datasets muy voluminosos, o con información sensible, podemos evitar subirlos al repositorio.

```
echo "*.csv" >> .gitignore
git add .gitignore
git commit -m "Ignorar archivos sensibles"
```

Ignorar archivos temporales de Jupyter

- Internamente Jupyter genera una carpeta “checkpoints” de archivos temporales, podemos evitar también subirla al repositorio.

```
echo "*.ipynb_checkpoints/" >> .gitignore
```