PROPEDÉUTICO DE MATEMÁTICAS Y PROGRAMACIÓN · FLACSO ECUADOR

El mapa de las herramientas

Manual de consulta — qué es cada pieza del taller, qué hace y qué no hace, cómo leer las señales de VS Code y cómo salir de los tropiezos de siempre.

Modalidad: consulta libre — no es una clase **Léelo:** después de la clase 2 o 3

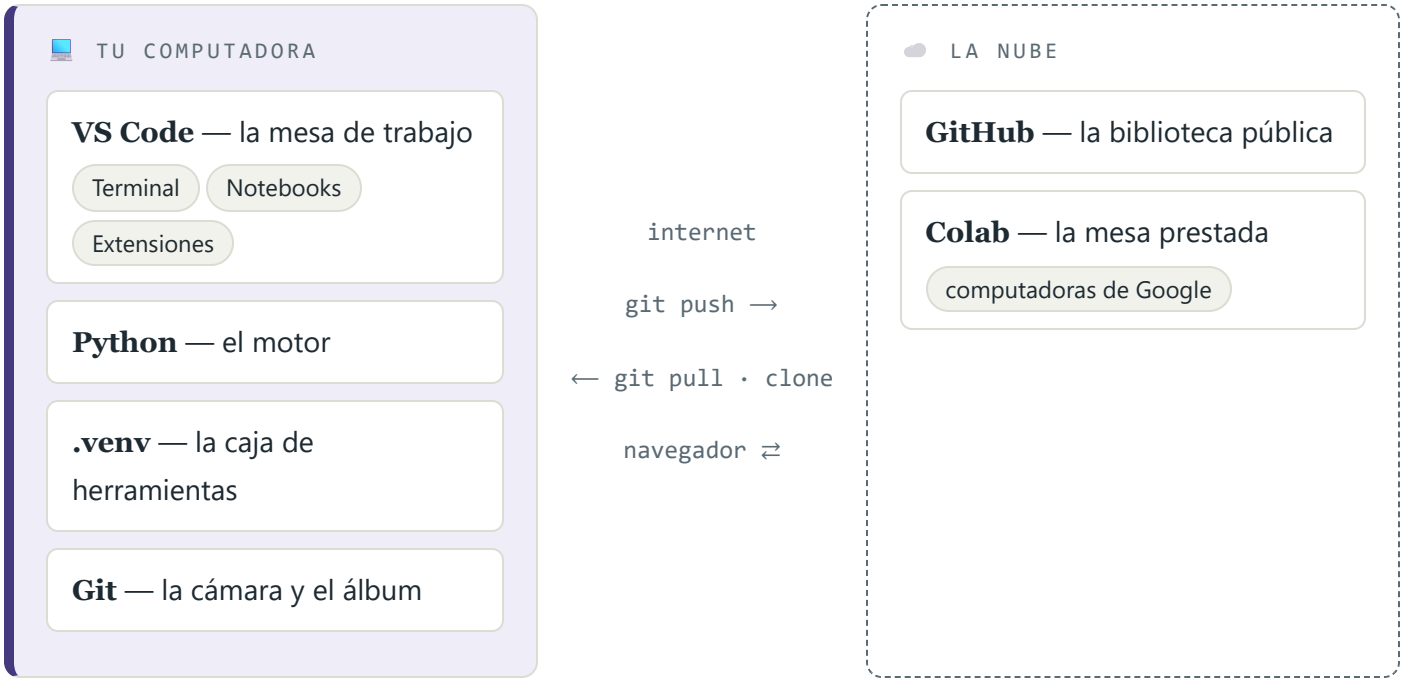
Úsalo: todo el curso — y toda la maestría

Cómo usar este manual

Este manual no es paso a paso: es un **mapa**. Los tres manuales anteriores te enseñaron a *hacer*; este te ayuda a *ubicarte*. Léelo de corrido una vez (unos 20 minutos) y después salta directo a lo que necesites:

> 💡 **¿Por qué el manual de instalación habla de «tres herramientas» y aquí hay cuatro?**
> Porque tres se instalan y la cuarta se visita: GitHub no es un programa, es un lugar. Ese es exactamente el tipo de confusión que este mapa viene a deshacer.

El mapa: lo que vive en tu computadora y lo que vive en la nube



Todo lo de la izquierda funciona sin internet. Internet hace falta solo para cruzar el puente: `git push` sube, `git pull` y `git clone` bajan — y el navegador te lleva a Colab. Guardar con **Ctrl+S** no cruza el puente; hacer commit tampoco: solo `push` publica.

Las cuatro grandes: sirve para / no sirve para

Cada herramienta tiene un oficio — y confundir los oficios es la fuente de casi todas las ideas erróneas. La columna «no sirve para» es tan importante como la otra.

VS Code — la mesa de trabajo

El editor donde escribes y ejecutas: dentro de él viven la terminal, los notebooks y las extensiones.

✓ SIRVE PARA

- Escribir y editar código y texto.
- Alojar tu terminal y tus notebooks.
- Mostrarte lo que Git ve (el semáforo de más abajo).

✗ NO SIRVE PARA

- «Saber» Python por sí solo: sin el intérprete instalado, la mesa está vacía.
- Guardar en la nube: `Ctrl+S` escribe solo en tu disco.
- Sustituir a Git: solo muestra sus señales.

Python — el motor

El lenguaje del curso — y el programa (intérprete) que entiende y ejecuta lo que escribes.

✓ SIRVE PARA

- Ejecutar tus programas y notebooks.
- Hacer los cálculos del curso.
- Trabajar sin internet.

✗ NO SIRVE PARA

- Abrirse como una app con ventana propia: trabaja detrás de VS Code y de los notebooks.
- Guardar versiones o respaldar: ese es el oficio de Git.
- Ser exclusivo de tu laptop: Colab usa este mismo motor, en la nube.

Git — la cámara y el álbum

El control de versiones: fotografía tu proyecto cuando *tú* se lo pides y guarda el álbum completo.

✓ SIRVE PARA

- Guardar versiones con `commit`.
- Recuperar cualquier fotografía del pasado.
- Conectar tu carpeta con GitHub (`push` / `pull`).

✗ NO SIRVE PARA

- Respaldar automáticamente — **no es OneDrive ni Drive** : sin `commit` y `push` no hay copia en ninguna parte.
- Ejecutar código.
- Nada que exija internet — salvo `push` , `pull` y `clone` .

GitHub — la biblioteca pública

El sitio web donde viven copias de los repositorios — el del curso y el tuyo. No se instala: se visita.

✓ SIRVE PARA

- Alojar y mostrar tus repositorios: tu portafolio.
- Distribuir los materiales del curso.
- Respaldar lo que subiste con `push` .

✗ NO SIRVE PARA

- **Ejecutar código** — es la biblioteca, no la mesa: para correr un notebook lo bajas a tu computadora o lo abres en Colab.
- Funcionar sin Git: sin Git en tu máquina, GitHub es solo una página que miras.
- Ver lo que no haya viajado con `push` .

Las piezas de cada día: terminal, notebook, entorno, Colab

La terminal — la ventanilla

La ventana (dentro de VS Code) donde le das órdenes escritas y exactas a la computadora; siempre está «parada» en una carpeta.

✓ SIRVE PARA

- Ejecutar `git`, `python`, `pip` y compañía.
- Decirte dónde estás (`pwd`).
- Acompañarte toda la maestría.

✗ NO SIRVE PARA

- Adivinar: exige la orden exacta — `git`, no `gti`.
- «Programar»: ahí se dan órdenes; el código se escribe en el editor.
- Morder: leer lo que responde es la habilidad, no el castigo.

El notebook — el cuaderno de laboratorio

Un archivo `.ipynb` que mezcla texto, código y resultados, celda por celda.

✓ SIRVE PARA

- Las clases del curso.
- Explorar datos contando la historia junto al código.
- Abrirse igual en VS Code y en Colab.

✗ NO SIRVE PARA

- Ser «de Colab» o «de VS Code»: el mismo archivo se abre en ambos.
- Correr solo: necesita un motor (kernel) — tu `.venv` o la máquina de Colab.
- Ejecutarse «en orden» por su cuenta: tú decides qué celda corre y cuándo.

El entorno virtual (.venv) — la caja de herramientas

Una carpeta del proyecto con las librerías que ese proyecto necesita — y solo esas.

✓ SIRVE PARA

- Aislar las librerías de cada proyecto.
- Reconstruirse en cualquier máquina con `requirements.txt` (la lista de compras).
- Avisarte que está activo con el `(.venv)` del prompt.

✗ NO SIRVE PARA

- Ser «otro Python» que descargaste.
- Viajar a GitHub: por eso se ve gris en VS Code — Git lo ignora a propósito; lo que viaja es `requirements.txt`.
- Activarse solo fuera de VS Code.

Colab — la mesa prestada

Notebooks en el navegador, ejecutándose en computadoras de Google: la otra vía legítima del curso.

✓ SIRVE PARA

- Trabajar desde cualquier máquina sin instalar nada.
- Traer ya instaladas las librerías del curso.
- Compartirse como un documento de Google.

✗ NO SIRVE PARA

- Ser VS Code: lo que instalas o guardas en tu laptop no existe en Colab, y viceversa.
- Guardar en tu disco: guarda en tu Google Drive.
- Publicar en GitHub: «Guardar una copia en Drive» no es `git push`.

La escalera de persistencia: ¿dónde vive tu trabajo?

Tu trabajo vive en cuatro niveles, y cada uno necesita un gesto tuyo para subir al siguiente. La mitad de los sustos del curso — «¡cambié el archivo y no pasa nada!», «¡hice commit y GitHub no muestra nada!» — son confusiones de peldaño.

`1 · escribes`

La pantalla

Lo que acabas de teclear. Existe solo en la memoria de VS Code: si algo se cierra mal, se fue. Señal: el punto ● en la pestaña.

`2 · ↑ Ctrl+S / ⌘S`

El disco

El archivo guardado en tu computadora. Ya no se pierde al cerrar — y recién ahora Git puede verlo. Señal: la × vuelve a la pestaña.

`3 · ↑ git add + commit`

El historial

La fotografía en tu álbum local. Puedes volver a ella cuando quieras — pero sigue viviendo solo en tu laptop.

`4 · ↑ git push`

La nube

Publicado en GitHub: respaldado y visible desde cualquier lugar. Recién aquí existe fuera de tu computadora.

> 💡 **Regla de bolsillo**

> Un cambio no está a salvo hasta el peldaño 4. **Ctrl+S no es commit; commit no es push.** Cuando algo «no aparece», pregúntate en qué peldaño se quedó.

El semáforo de VS Code: aprender a leer sus señales

VS Code decora los nombres de archivo con letras y colores: es Git hablándote por señas. En este curso Git se maneja **escribiendo comandos en la terminal**; esta sección no cambia eso — te enseña a *leer* el tablero, no a soltar el volante.

Señal	Dónde aparece	Qué significa	Ya la conoces de la terminal
● punto en la pestaña	Pestaña del editor, donde suele estar la ×	Cambios **sin guardar**. Git todavía no ve nada de esto.	Ninguna: primero `Ctrl+S` / `⌘S`. El hábito: guardar antes de cualquier `git status`.
`U` (verde)	Explorador y panel Source Control	*Untracked*: archivo nuevo que Git aún no vigila.	El «Untracked files» (en rojo) de `git status`. Se atiende con `git add`.
`A` (verde)	Panel Source Control (Staged Changes)	*Added*: ya está en la bandeja; saldrá en la próxima fotografía.	El «Changes to be committed» (en verde). Se atiende con `git commit -m`.
`M` (amarillo/naranja)	Explorador y Source Control	*Modified*: archivo vigilado, con cambios desde el último commit.	El «modified:» de `git status`. Se atiende con `git add`.
`D` (rojo, nombre tachado)	Source Control	*Deleted*: archivo borrado — el borrado también se fotografía.	El «deleted:» de `git status`; también pasa por `add` y `commit`.
Nombre en `gris` (sin letra)	Explorador	*Ignored*: Git lo ignora a propósito (lista `.gitignore`). Tu `.venv` se ve así — y está bien.	No aparece en `git status`, deliberadamente.

Número en el ícono de Source Control	Barra lateral izquierda	Cuántos archivos tienen cambios pendientes. Es un recordatorio, no un error.	El resumen que te daría `git status`.

> 💡 **El mismo estado, dos colores**
>
> La terminal pinta *untracked* en **rojo** (pendiente de atender) y VS Code lo pinta en **verde** (archivo nuevo). No se contradicen: cada programa eligió su paleta. Lo que no cambia es el estado: U = *untracked* = Git aún no lo vigila.

> ⚠️ **Las señales se leen; los comandos se escriben**
>
> El panel Source Control tiene botones (+, ✓ Commit, Sync) que hacen `add`, `commit` y `push` con clics. En este curso seguiremos escribiendo los comandos: entender el ciclo vale más que ahorrarse tres teclas, y la terminal funciona igual en cualquier computadora. Usa el panel como tablero de información, no como volante.

Alguna vez verás otras letras — **R** de renombrado, **C** de conflicto —: primas lejanas que no necesitas este semestre.

¿Dónde estoy? — la pregunta que resuelve la mitad de los problemas

La categoría número uno de tropiezos de principiante no es de código: es de **ubicación** — comandos correctos ejecutados en la carpeta equivocada. Tres preguntas te ubican siempre:

1. **¿Qué carpeta tiene abierta VS Code?** Míralo en la barra de título, arriba. La terminal integrada nace «parada» en esa carpeta.
2. **¿Dónde está parada la terminal?** Pregúntaselo con `pwd` (*print working directory*); la ruta antes del cursor también lo dice.
3. **¿Este repositorio es el que creo?** `git status` responde en su primera línea — o protesta con `fatal: not a git repository` si no estás en uno.

```
pwd
git status
```

> 💡 **Antes de pedir ayuda**

A un docente o a una IA, llega con estas tres respondidas: **¿dónde estoy?, ¿guardé?, ¿qué dice `git status`?** La mitad de las veces, responderlas ya resuelve el problema.

Sabiduría de laboratorio

Los informáticos llevan décadas destilando su experiencia en refranes. Casi todos son medio broma — y casi todos esconden una lección que en este curso podemos tomar en serio.

> *«Si funciona, no lo toques.»*
>
> Con Git, exactamente al revés: **si funciona, haz commit — y entonces tócalo todo.** La fotografía es tu punto de retorno: con el historial guardado, experimentar deja de ser un riesgo. El dicho nació en talleres sin control de versiones; tú ya tienes uno.

> *«Reiniciar arregla casi todo.»*
>
> Verdad legitimada. Cerrar y volver a abrir VS Code — o abrir una terminal nueva — no es superstición: la terminal nueva vuelve a leer el PATH y «se entera» de lo que acabas de instalar. Por eso los manuales insisten tanto en «abre una terminal **nueva**».

> *«Es más fácil rehacer que arreglar.»*
>
> Para principiantes, a veces es literal: si tu copia del **repositorio del curso** quedó rota y no entiendes cómo, borrar la carpeta y volver a clonar es una salida legítima — todo está en GitHub. La letra pequeña: vale para el repositorio del curso, donde no tienes cambios propios; en **tu** repositorio, lo que no haya viajado con `push` se pierde con la carpeta.

> *«No dejes que las impresoras sientan tu miedo.»*
>
> Los mensajes de error no muerden: son la computadora explicando qué le faltó, no un regaño. Léelos empezando por la **última línea**, que suele traer la clave — y a veces hasta la solución. Si no lo entiendes, cópialo completo y pregúntale a una IA con la plantilla del final.

> *«Funciona en mi máquina.»*
>
> La excusa más famosa de la informática — y la razón de ser de `requirements.txt` y de los entornos virtuales: que «tu máquina» y las demás tengan la misma caja de herramientas. Si algo corre en Colab pero no en tu laptop (o al revés), sospecha del entorno antes que del código.

> *«La computadora hace lo que le dices, no lo que quieres.»*
>
> `gti` no es `git` y `commit -n` no es `commit -m`: para la terminal, «casi correcto» es incorrecto. La buena noticia: muchas veces te lo dice con propuesta incluida (*«The most similar command is…»*). Tu parte del trato es leerla.

> *«Ante la duda, git status.»*
>
> Esta no es heredada: es la que este curso te regala. `git status` no cambia nada — es la pregunta gratis. Antes de un `add`, después de un `commit`, cuando no entiendes qué pasó: pregunta. (Su primo para saber dónde estás parado: `pwd`.)

Los errores de siempre

La mitad de los tropiezos de un principiante son de tipeo; la otra mitad, de estar parado en la carpeta equivocada. Antes de la tabla, mira lo amable que puede ser un error:

UN ERROR AMABLE, VISTO DE CERCA

```
~/Documentos/propedeutico/mi-propedeutico> git stauts
git: 'stauts' is not a git command. See 'git --help'.

The most similar command is
        status
```

Git no solo se queja: **te sugiere la palabra correcta**. Léela. La tabla siguiente reúne los tropiezos más frecuentes del curso; cuando la solución ya vive en otro manual, el enlace te lleva directo.

Síntoma (lo que ves)	Qué pasó	Qué hacer

`git: 'stauts' is not a git command`	Subcomando de Git mal tecleado.	Lee las líneas siguientes: Git casi siempre propone «The most similar command is…». Escríbelo bien y repite.
`error: unknown switch 'n'` tras un commit	Escribiste `-n` en lugar de `-m` (la m de *mensaje*).	Repite con `git commit -m "mensaje"`. Si la terminal «se puso rara», es el P5 del Manual de la clase 3.
`"gti" no se reconoce…` / `command not found`	El **nombre del programa** mal escrito — la terminal no adivina.	Compara letra por letra: `git`, `python`, `pip`. Si está bien escrito y aun así no existe, la terminal es vieja o falta instalar: terminal nueva y, si sigue, P1-P2 del Manual de instalación.
El commit responde `Please tell me who you are`	Falta tu identidad de Git en esa computadora.	La solución vive en la sección G del Manual de instalación (y en el P1 del Manual de la clase 3): dos comandos `git config` y repetir el commit.
`fatal: not a git repository`	La terminal está parada en una carpeta que no es un repositorio.	`pwd` para ver dónde estás; abre la carpeta correcta (**File → Open Folder**) y una terminal nueva.
Cambié el archivo, pero `git status` dice `nothing to commit`	Archivo sin guardar (mira el ●) o carpeta equivocada.	**Ctrl+S** / **⌘S** y repite; si persiste, `pwd` (es el P4 del Manual de la clase 3).
`python` no responde, abre la Microsoft Store o dice 2.7	El PATH — o en Mac el comando es `python3`.	P1-P3 del Manual de instalación — cada caso tiene su receta.
Instalé una librería pero el notebook dice `ModuleNotFoundError`	Se instaló fuera de la caja: el entorno no estaba activo, o el kernel del notebook no es `.venv`.	Verifica el `(.venv)` en el prompt y **Select Kernel** → `.venv` (pasos 4-6 del Manual de la clase 2).

> 💡 **¿Tu problema no está en la lista? Pídele ayuda a una IA**

Un asistente de IA (ChatGPT, Claude, Gemini o Copilot) resuelve muy bien este tipo de atascos **si le das contexto suficiente**. Copia esta plantilla, rellena los huecos [entre corchetes] y pégala en el chat:

```
Estoy siguiendo un manual del curso propedéutico de FLACSO
y me atasqué en [paso o sección del manual donde estabas].
Mi sistema operativo es [Windows o Mac].
Escribí este comando o código:
[pega aquí exactamente lo que escribiste]
Y obtuve este mensaje de error o problema:
[pega aquí el mensaje completo, copiado desde la terminal]
Por favor: primero explícame por qué ocurre el problema,
en lenguaje sencillo (no tengo experiencia en programación),
y después dame los pasos para resolverlo, uno por uno.
```

Tres consejos para que la respuesta sirva:

- **Copia el error completo como texto** — selecciónalo en la terminal y cópialo — mejor que una captura de pantalla o un fragmento suelto.
- **Lee cada comando que la IA te sugiera antes de ejecutarlo**; si no entiendes qué hace, pregúntale. Nunca ejecutes a ciegas comandos que borren archivos o carpetas.
- Si la primera respuesta no funciona, **cuéntale qué pasó al intentarlo**: la conversación de ida y vuelta es parte normal del proceso.

✅ Sabes leer el mapa cuando...

No es una tarea: es un espejo. Marca lo que ya puedas decir con seguridad:

- [ ] Puedo explicar por qué GitHub no ejecuta código — y decir dónde sí se ejecuta (mi computadora o Colab).
- [ ] Sé qué significan la **U** verde y la **M** amarilla junto a un archivo, y qué comando responde a cada una.
- [ ] Si veo un punto ● en la pestaña, sé qué hacer antes que cualquier otra cosa.
- [ ] Sé que Git solo respalda lo que fotografío con `commit` y subo con `push` — no es OneDrive.

- [ ] Ante un mensaje de error, leo la última línea antes de asustarme.
- [ ] Distingo Colab de VS Code: la misma lógica, en computadoras distintas.

Qué sigue

Este manual no se termina: se vuelve a abrir. Vuelve al semáforo cuando VS Code te muestre una letra nueva, a la tabla de errores cuando la terminal responda algo raro, y al mapa cuando dudes de dónde vive algo. Los paso a paso siguen siendo el **Manual de instalación**, el **Manual de la clase 2** y el **Manual de la clase 3** — y tu bitácora sigue esperando una entrada por clase.

Propedéutico de Matemáticas y Programación · FLACSO Ecuador — Manual de consulta: el mapa de las herramientas, versión julio 2026.
Acompaña al **Manual de instalación** (VS Code, Python y Git), al **Manual de la clase 2** (clonar el repositorio y tu primer entorno) y al **Manual de la clase 3** (tu propio repositorio).