SESIÓN 5 · QA ENGINEERING

Dev Craft 🛠️

Tu suite vive solo en tu laptop — para el equipo, **no existe**.
Hoy: terminal con soltura, Git sin miedo y tu primer **Pull Request**.

`terminal` `package.json` `git` `GitHub + PR` `2 simuladores 🎮` `4 misiones 🎯`

LA VERDAD INCÓMODA

Código que no está en Git es un castillo de arena 🏝️

💀

Sin Git

Laptop muere = suite muere. "¿Qué cambié ayer?" = misterio. Dos personas editando = caos de versiones por email.

🤝

El QA Engineer real

Tu suite entra al repo del equipo **por Pull Request**, con revisión, igual que el código de los devs. Mismo flujo, mismos estándares.

🗺️ El plan de hoy

```
Acto 1 → terminal sin miedo
Acto 2 → package.json y scripts
Acto 3 → Git local 🎮  (simulador + real)
Acto 4 → GitHub + tu primer PR 🎮
Boss → tu suite del lab, publicada
```

🎮 Hoy estrenamos **terminales simuladas** dentro del deck: practicas los comandos aquí (imposible romper nada) y DESPUÉS los corres de verdad.

Terminal sin miedo: 6 comandos te bastan ⌨️

COMANDO	QUÉ HACE	MNEMONIA
`pwd`	¿DÓNDE estoy parado?	print working directory
`ls`	¿QUÉ hay aquí?	list
`cd mi-dojo-lab`	Entra a una carpeta	change directory
`cd ..`	Sube un nivel	.. = la carpeta padre
`mkdir pages`	Crea una carpeta	make directory
`code .`	Abre VS Code AQUÍ	. = esta carpeta

⚡ **Los dos superpoderes:** Tab autocompleta nombres (escribe `cd mi-d` + Tab) · flecha ↑ repite comandos anteriores. Quien los usa parece senior en una semana.

MISIÓN D1

Navega como profesional

☐ Completada

🎯 Desde una terminal NUEVA, llegar hasta tu proyecto del lab usando solo comandos, y abrirlo en VS Code.

1. Abre una terminal nueva (no la de VS Code — esa hace trampa)
2. `pwd` para ubicarte, `ls` para mirar
3. Navega con `cd` + `Tab` hasta `mi-dojo-lab`
4. Remata con:

```
$ code .
```

💡 **Pista: "code: command not found" (Mac)**

💡 **Pista: me perdí en las carpetas**

✅ **Criterio de éxito:** VS Code abierto en tu proyecto, lanzado 100% desde la terminal, sin tocar el Finder/Explorador.

package.json: la cédula del proyecto 📋

package.json

```
{
  "name": "mi-dojo-lab",
  "scripts": {
    "test": "playwright test",
    "test:ui": "playwright test --ui"
  },
  "devDependencies": {
    "@playwright/test": "^1.52.0"
  }
}
```

🪪 name

Identidad del proyecto.

📜 scripts

El **recetario de comandos** del equipo: alias cortos para comandos largos. Se corren con `npm run <nombre>`.

📦 devDependencies

La **lista de compras**: qué herramientas necesita el proyecto y en qué versión. `npm install` las trae todas.

🚫 node_modules NO viaja

Pesa cientos de MB y se **reconstruye** desde la lista con npm install. Por eso JAMÁS se sube a Git (lo ignoraremos en el Acto 3).

MISIÓN D2

Arma tu recetario de scripts

Completada

🎯 Definir los 4 scripts estándar de tu suite y usarlos con **npm run**.

1. En tu package.json, dentro de `"scripts"` :

```
"test": "playwright test",
"test:ui": "playwright test --ui",
"test:headed": "playwright test --headed",
"report": "playwright show-report"
```

2. Pruébalos:

```
$ npm run test
```

💡 **Pista: "Unexpected token" al correr npm**

🤝 El valor real: **contrato de equipo**. Cualquier persona (o un pipeline de CI) corre tu suite con `npm run test` sin saber qué hay detrás. Estándar universal del ecosistema.

✅ **Criterio de éxito:** `npm run test` y `npm run test:ui` funcionan en tu proyecto.

⭐ QUIZ 1 · El ritual del clon

Clonas el proyecto de un compañero, corres `npm run test` y explota: "Cannot find module '@playwright/test'". ¿Primer comando?

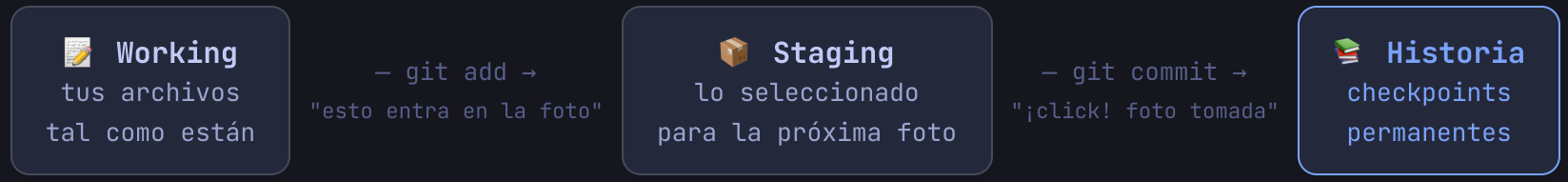
A Reinstalar Node.js completo

B `npm install` — reconstruir node_modules desde la lista de compras

C Copiarle la carpeta node_modules por USB

💡 node_modules no viaja con el repo (está ignorado) — el clon llega "sin despensa". `npm install` lee las dependencies del package.json y la reconstruye exacta. El ritual universal: **clonar → npm install → a trabajar**. La opción C funciona... como funciona pegar un billete roto con cinta: no lo hagas.

Git: checkpoints para tu código 📸



🎮

Como un videojuego

Commit = grabar partida. ¿Rompiste todo? Vuelves al último checkpoint. Nada se pierde de verdad después de un commit.

📦

¿Por qué staging?

Eliges QUÉ entra en cada foto. Cambiaste 5 archivos pero 2 son de otra tarea → fotos separadas, historia limpia.

🔍

git status = tu brújula

Te dice qué está modificado, qué está en staging y qué falta. **Cuando dudes: git status.** Siempre.

El ciclo sagrado + mensajes dignos

El ciclo (lo harás 1000 veces)

```
$ git status
```

```
$ git add .
```

```
$ git commit -m "mensaje claro"
```

`git add .` = todo lo modificado al staging · `-m` = mensaje inline.

🙈 Antes del primer commit: crea `.gitignore` con `node_modules/`, `test-results/` y `playwright-report/` — la despensa y los reportes NO van en la historia.

El mensaje es para un humano del futuro

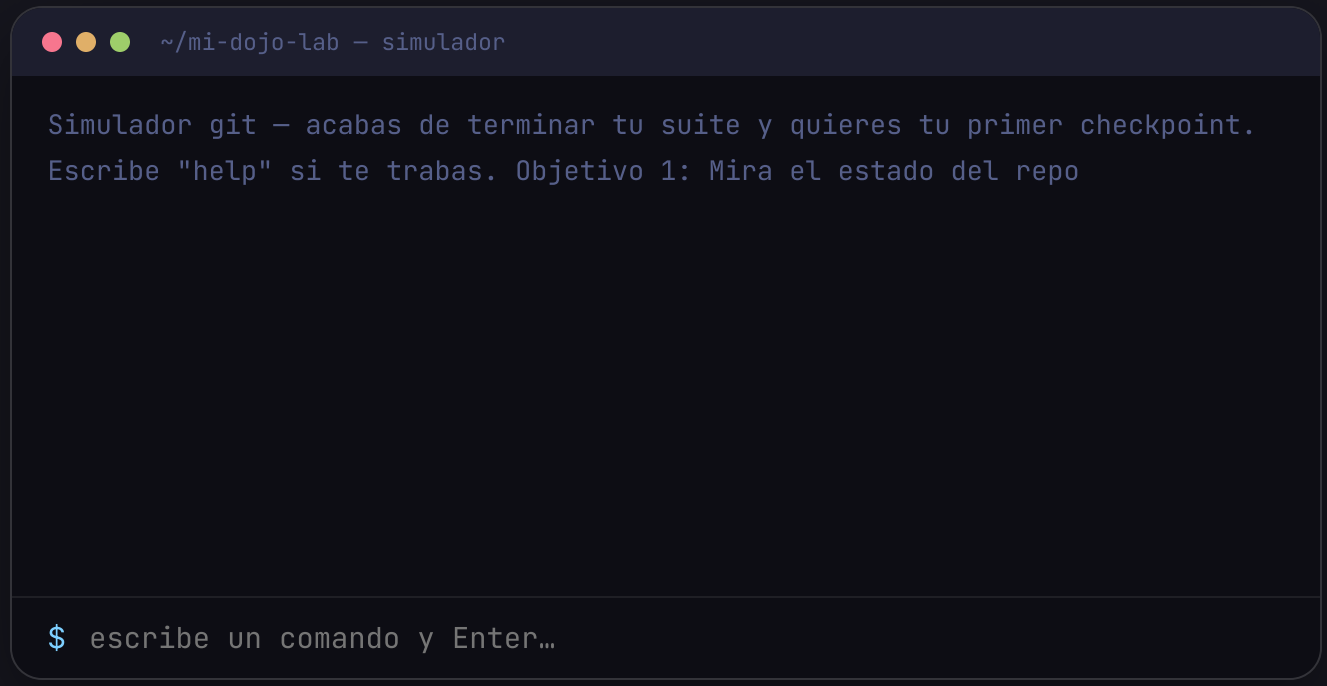
❌ INDIGNO	✅ DIGNO
"cambios"	"add login page object with semantic locators"
"fix"	"fix flaky register test with unique email"
"asdfgh"	"add API tests for auth endpoints"

Fórmula: **verbo + qué + (por qué si no es obvio)**. El humano del futuro que lee tu historia... casi siempre eres tú.

🎮 TERMINAL SIMULADA · IMPOSIBLE ROMPER NADA

Practica el ciclo aquí mismo

- [ ] 👉 Mira el estado del repo
- [ ] Pon los archivos en staging
- [ ] Crea el commit con mensaje
- [ ] Mira tu historia



MISIÓN D3

Git de verdad en tu proyecto

Completada

🎯 Poner tu proyecto del lab bajo control de versiones: init, .gitignore y primer commit con mensaje digno.

```
$ git init
```

Crea `.gitignore` (raíz del proyecto) con:

```
node_modules/
test-results/
playwright-report/
```

```
$ git add .
```

```
$ git commit -m "add dojo test suite with POM and fixtures"
```

```
$ git log --oneline
```

💡 **Pista: git pide tu identidad**

💡 **Pista: ¿el .gitignore funcionó?**

✅ **Criterio de éxito:** `git log --oneline` muestra tu commit inicial, y `git status` NO lista node_modules.

⭐ QUIZ 2 · Las 3 zonas

Modificaste 5 archivos, pero 2 pertenecen a otra tarea. ¿Cómo logras un commit limpio con solo los 3 relevantes?

A Imposible — git add . se lleva todo, es todo o nada

B `git add archivo1 archivo2 archivo3` — el staging existe exactamente para elegir qué entra en la foto

C Borrar los 2 archivos ajenos y recrearlos después del commit

💡 `git add` acepta archivos específicos — esa es la razón de ser del staging: componer CADA foto deliberadamente. `git add .` es el atajo para "todo", no la única forma. Historia limpia = un commit por intención = futuro tú feliz.

GitHub: tu historia, en la nube ☁️

Remoto = el repo "oficial" compartido. **push** sube tus commits; **pull** baja los de otros. Laptop muere → la historia vive.

Conectar tu proyecto (una vez)

Con GitHub CLI (recomendado — crea el repo Y lo conecta):

```
$ gh auth login
```

```
$ gh repo create mi-dojo-lab --private --source=. --push
```

💡 **Alternativa sin gh (web)**

Branches y PR: proponer, no imponer 🌿

🌿 **rama**
`git checkout -b test/login-suite`

→ commits →

☁️ **push**
la rama sube a GitHub

→

🔀 **Pull Request**
"revisen mi propuesta"

→ review ✓ →

🎉 **merge**
tu código entra a main

main es sagrada

Nadie empuja directo a main: es la versión estable. Tu rama = borrador seguro donde nada se rompe.

💬

El PR es conversación

Descripción de qué y por qué, código comentable línea por línea, aprobación antes del merge. Ahí ocurre la revisión real.

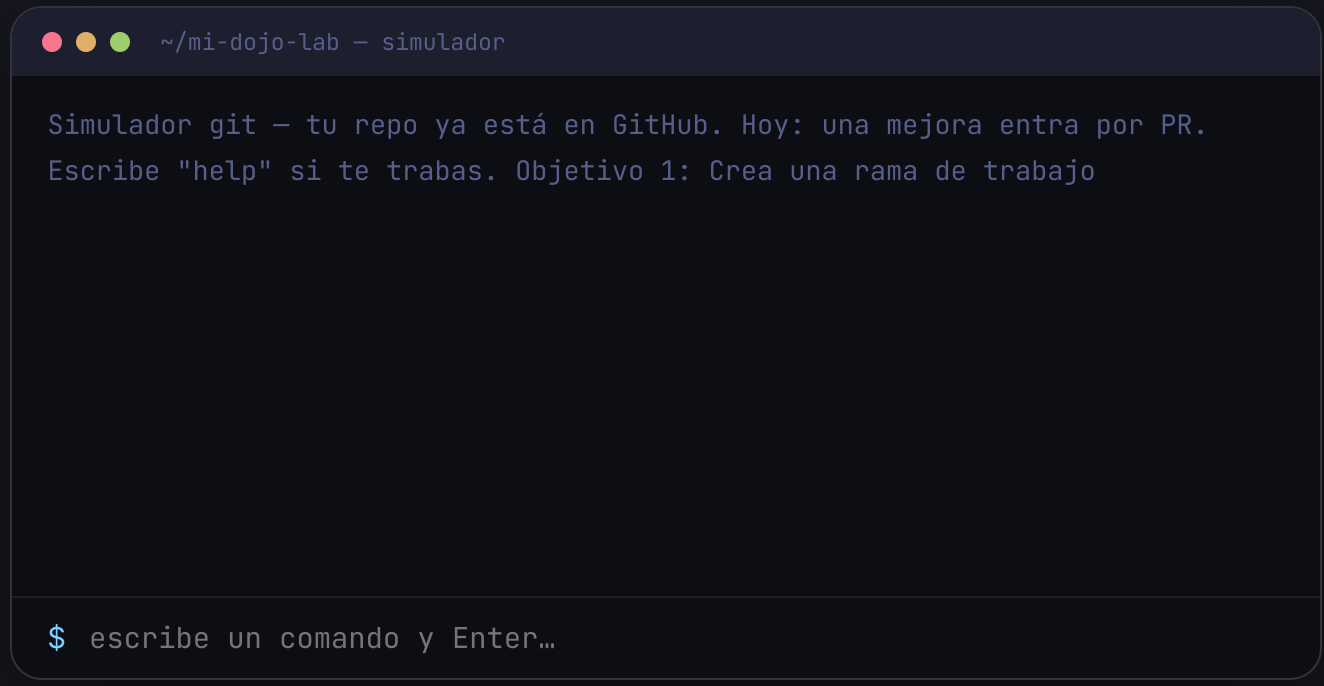
En QA

Tus suites entran por PR al repo del equipo — y los pipelines de CI corren los tests EN el PR antes de aprobar. Tu trabajo se vuelve gate de calidad.

🎮 TERMINAL SIMULADA · EL FLUJO COMPLETO

Practica: de la rama al PR

- 👉 Crea una rama de trabajo
- ⬜ Stagea tus cambios
- ⬜ Commit con mensaje digno
- ⬜ Sube la rama a GitHub
- ⬜ Abre el Pull Request



MISIÓN D4 · BOSS 🏆

Tu suite, publicada con PR

☐ Completada

🎯 Tu proyecto del lab en GitHub, con una mejora entrando por **Pull Request** — el flujo profesional completo, de verdad.

1. Repo en GitHub conectado (slide 13)
2. Crea una rama para una mejora pequeña (un test nuevo, un script, el POM de otra página):

```
$ git checkout -b test/mi-mejora
```

3. Haz el cambio → add → commit → push:

```
$ git push -u origin test/mi-mejora
```

```
$ gh pr create --fill
```

💡 **Pista: sin gh CLI**

💡 **Pista: ¿y quién me lo aprueba?**

✅ **Criterio de éxito:** URL de un PR tuyo, mergeado, en tu repo de GitHub. Guárdala — es tu primera entrega con flujo profesional.

★ QUIZ 3 · El porqué del ritual

Trabajas SOLO en tu suite. ¿Tiene sentido seguir usando ramas y PRs?

A No — los PRs solo sirven cuando hay un equipo que revise

B Sí — historial revisable, CI se engancha a los PRs, y practicas el flujo que usarás en equipo

C Solo si el proyecto es público

💡 Tres razones aun en solitario: (1) cada PR documenta un cambio con intención — historial navegable; (2) los pipelines de CI corren los tests EN el PR: tu red de seguridad automática; (3) el hábito — quien trabaja con ramas en solitario no improvisa cuando entra a un equipo. El flujo ES el entrenamiento.

TU NUEVA CAJA DE HERRAMIENTAS

El kit del oficio, completo 🧰

HERRAMIENTA	TU NUEVO SUPERPODER
⌨️ Terminal	Navegas y ejecutas sin miedo — pwd, cd, Tab, ↑
📋 package.json	Lees la cédula de cualquier proyecto JS y defines scripts de equipo
📸 Git local	Checkpoints con mensajes dignos — nada se pierde nunca más
☁️ GitHub	Tu código vive en la nube, sobrevive a tu laptop
🔀 Pull Request	Entregas cambios como profesional: propuesta → review → merge

🥋 El repo profesional al que vas usa EXACTAMENTE este flujo (con ramas `test/*` y prefijos semánticos en commits) — más un detalle: allí `bun` reemplaza a `npm`. Mismos conceptos, otro instalador.

DEBRIEF

Misiones completadas

Tarea

Un commit diario en tu proyecto esta semana — el músculo se hace con repetición. Y termina misiones pendientes.

Última parada: KATA Bridge

Tienes TODO: código, clases, Playwright pro, fixtures, git. El próximo deck cruza el puente final al framework profesional.