🥋 Kata Git: Ejercicios Progresivos

📘 Descripción General

Esta kata está diseñada para practicar **Git paso a paso**, con ejercicios que se construyen sobre los anteriores. A medida que avances, vas mejorando tu "forma" en el uso de Git, enfrentando pequeños desafíos cada vez más realistas.

🧱 Requisitos

- Git instalado
- Editor de código (VS Code, Sublime, etc.)
- Terminal de comandos
- Un entorno limpio para pruebas (carpeta local vacía)

🧭 Estructura de la Kata

Cada etapa tiene:

- 🎯 **Objetivo**
- 📋 **Instrucciones**
- 🤔 **Preguntas de reflexión**
- 💡 **Tips de comandos**

🥋 Kata 1: Primeros pasos

🎯 **Objetivo:** Iniciar un repositorio local y hacer el primer commit

📋 **Instrucciones:**

1. Crear una carpeta llamada `kata-git`.
2. Iniciar un repositorio con `git init`.
3. Crear un archivo `hola.py` que diga `print("Hola mundo")`.
4. Confirmar el estado con `git status`.
5. Agregar y commitear el archivo.

🤔 **Pregunta:** ¿Qué archivos aparecen como "sin seguimiento"? ¿Cómo cambia el estado luego del commit?

💡 **Comandos clave:**
`git init` — `git status` — `git add` — `git commit`

🥋 Kata 2: Segundo commit y exploración

🎯 **Objetivo:** Modificar un archivo, revisar cambios y hacer otro commit.

📋 **Instrucciones:**

1. Editar `hola.py` para que pida nombre al usuario.
2. Usar `git diff` antes de hacer `add`.
3. Commit de los cambios.

🤔 **Pregunta:** ¿Qué muestra `git diff` antes y después de `add`?

💡 **Comandos clave:**
`git diff` — `git add` — `git commit`

🥋 Kata 3: Volver al pasado

🎯 **Objetivo:** Desplazarse a un commit anterior.

📋 **Instrucciones:**

1. Listar el historial con `git log`.
2. Hacer `checkout` a un commit anterior.
3. Volver a la rama principal con `git checkout main`.

🤔 **Pregunta:** ¿Qué pasa con los archivos si estás en un commit antiguo?

💡 **Comandos clave:**
`git log` — `git checkout <hash>`

🥋 Kata 4: Branches para probar ideas

🎯 **Objetivo:** Crear y usar ramas.

📋 **Instrucciones:**

1. Crear una rama `saludo-ingles`.
2. Cambiar el mensaje a `print("Hello world")`.
3. Hacer commit.
4. Volver a `main`.

🧐 **Pregunta:** ¿Dónde se encuentran tus cambios?

💡 **Comandos clave:**
`git branch` — `git checkout -b` — `git commit`

🥋 Kata 5: Comparar ramas

🎯 **Objetivo:** Analizar diferencias entre ramas.

📋 **Instrucciones:**

1. Desde `main`, comparar con la rama `saludo-ingles`.
2. Usar `git diff saludo-ingles`.

🧐 **Pregunta:** ¿Qué pasa si hacés `git diff main saludo-ingles` y al revés?

💡 **Comandos clave:**
`git diff rama1 rama2` — `git log --graph --oneline --all`

🥋 Kata 6: Fusionar cambios

🎯 **Objetivo:** Realizar un merge sin conflictos.

📋 **Instrucciones:**

1. Volver a `main`.
2. Hacer `merge` de `saludo-ingles`.
3. Confirmar el resultado con `git log`.

🧐 **Pregunta:** ¿Qué pasó con los commits de la rama fusionada?

💡 **Comandos clave:**
`git merge` — `git log`

🥋 Kata 7: Manejo de conflictos

🎯 **Objetivo:** Resolver un conflicto manualmente.

📋 **Instrucciones:**

1. Crear una rama `saludo-frances` y cambiar el saludo a `Bonjour`.

2. Volver a `main`, cambiar saludo a `Hallo` y commitear.
3. Intentar fusionar `saludo-frances`.

🤔 **Pregunta:** ¿Cómo se muestra el conflicto en el archivo? ¿Qué aprendiste del proceso?

💡 **Comandos clave:**
`git merge` — editar archivo — `git add` — `git commit`

🥋 Kata 8: Deshacer cambios

🎯 **Objetivo:** Probar formas de revertir commits.

📋 **Instrucciones:**

1. Hacer un commit incorrecto (por ejemplo, borrar una línea sin querer).
2. Usar `git reset` con distintas opciones: `--soft`, `--mixed`, `--hard`.

🤔 **Pregunta:** ¿Cuál de las opciones mantiene tus archivos intactos?

💡 **Comandos clave:**
`git reset` — `git reflog`

🔍 Introducción a `.gitignore`

Cuando trabajamos con Git, **no todos los archivos del proyecto deberían ser versionados**. Algunos archivos son **temporales, personales o generados automáticamente**, y **no tienen sentido compartirlos ni subirlos al repositorio**.

Por ejemplo:

- Archivos de configuración local (`.env`, `config.txt`)
- Archivos generados por el sistema (`.DS_Store`, `Thumbs.db`)
- Archivos temporales (`*.log`, `*.tmp`)
- Directorios de compilación o entornos (`node_modules/`, `build/`, `venv/`)

👉 **`.gitignore` es un archivo especial que le dice a Git qué archivos debe ignorar.**

🥋 Kata 9: Ignorar archivos innecesarios

🎯 **Objetivo:** Usar `.gitignore` para evitar que Git rastree archivos irrelevantes o sensibles.

📋 Instrucciones detalladas:

1. En la raíz del proyecto (`kata-git`), creá un archivo de texto llamado `config.txt`.
2. Escribí dentro de `config.txt` algo como:

   ```
   clave_secreta=12345
   ```

3. Verificá con `git status` que el archivo aparece como **sin seguimiento**.
4. Ahora creá un archivo llamado `.gitignore` (ojo con el punto al inicio).
5. Dentro de `.gitignore`, escribí:

   ```
   config.txt
   ```

6. Guardá y volvé a ejecutar `git status`.

🚧 ¿Qué debería pasar?

- Si **no habías agregado** `config.txt` antes con `git add`, ahora Git **ya no lo muestra** en `git status`. ¡Está ignorado!
- Si **ya lo habías agregado antes**, Git **lo sigue rastreando**. Para que empiece a ignorarlo de verdad, tenés que ejecutar:

```
git rm --cached config.txt
```

Y después, si querés, hacer un commit explicando el cambio:

```
git commit -m "Se eliminó config.txt del control de versiones"
```

🧪 Prueba extra (opcional):

1. Creá un archivo `registro.log` y agregálo al `.gitignore` con esta línea:

   ```
   *.log
   ```

2. Probá agregar más archivos `.log` y confirmá que Git los ignora automáticamente.

🧐 Preguntas para reflexionar:

- ¿Por qué no conviene subir archivos como `config.txt` a GitHub?
- ¿Qué podría pasar si alguien sube su `.env` con contraseñas?
- ¿Podrías tener distintos `.gitignore` para distintos lenguajes o entornos?

💡 Comandos clave:

- `git status`
- `git add`
- `git rm --cached archivo`
- Archivos `.gitignore`

🥋 Kata 10: Publicar el repo (opcional)

🎯 **Objetivo:** Subir el proyecto a un repositorio remoto.

📋 **Instrucciones:**

1. Crear un repositorio vacío en GitHub.
2. Enlazar con `git remote add origin <url>`.
3. Subir los cambios.

🤔 **Pregunta:** ¿Qué significa el flag `-u` en `git push -u`?

💡 **Comandos clave:**
`git remote` — `git push -u origin main`