Tutorial: Subir Cambios Locales a GitHub

Guía Paso a Paso con Explicación Detallada

🎯 OBJETIVO

Aprender a subir cualquier cambio que hagas en tu PC local al repositorio remoto en GitHub de manera correcta y segura.

📋 FLUJO BÁSICO (Los 4 Pasos Fundamentales)

```
Cambios Locales → Staging Area → Commit Local → GitHub
      ↓              ↓              ↓           ↓
   git add .    git commit -m    git push   ✅ GitHub
```

🔧 PARTE 1: COMANDOS BÁSICOS EXPLICADOS

1. `git add .` - Preparar cambios

¿Qué hace?

- Toma TODOS los archivos modificados/nuevos/eliminados
- Los coloca en el "área de staging" (zona de preparación)
- Es como decir "estos cambios están listos para guardarse"

Variaciones:

```bash
# Agregar todos los archivos
git add .

# Agregar un archivo específico
git add archivo.html

# Agregar una carpeta específica
git add carpeta/

# Agregar solo archivos JavaScript
git add *.js

# Agregar archivos por tipo
git add *.html *.css *.js
```

2. `git commit -m "mensaje"` - Guardar cambios

¿Qué hace?

- Crea una "fotografía" permanente de todos los cambios en staging
- Le asigna un ID único (hash)
- Guarda el mensaje descriptivo
- Los cambios quedan guardados LOCALMENTE (aún no en GitHub)

Mensajes de commit efectivos:

```bash
# ✅ Buenos ejemplos
git commit -m "Agregar página de contacto"
git commit -m "Corregir error en calculadora de precio"
git commit -m "Actualizar estilos del header"
git commit -m "Eliminar archivos temporales"

# ❌ Malos ejemplos
git commit -m "cambios"
git commit -m "fix"
git commit -m "update"
```

3. `git push` - Subir a GitHub

¿Qué hace?

- Envía todos los commits locales a GitHub
- Sincroniza tu repositorio local con el remoto
- Hace que otros puedan ver tus cambios

Variaciones:

```bash
# Push normal (después del primer push)
git push

# Push con tracking (primera vez)
git push -u origin main

# Push forzado (CUIDADO)
git push -f origin main
```

🎯 PARTE 2: TU FLUJO DIARIO ESPECÍFICO

Escenario: Modificaste archivos en tu proyecto

Paso 1: Ir a tu carpeta del proyecto

```
bash

cd ~/Documentos/repositoy-node.js
```

Paso 2: Ver qué cambió

```
bash

git status
```

Interpretando la salida de `git status`:

- **Rojo = archivos modificados** pero no preparados
- **Verde = archivos preparados** para commit
- **"Untracked files" = archivos nuevos** que Git no está siguiendo

Ejemplo de salida:

```
Changes not staged for commit:
  modified:   U13/cotizador-hogar/index.html   ← Archivo modificado
  modified:   U13/cotizador-hogar/css/styles.css

Untracked files:
  U14/nueva-leccion.html                       ← Archivo nuevo
```

Paso 3: Agregar cambios al staging

```
bash

git add .
```

Verificar qué se agregó:

```
bash

git status
```

Ahora deberías ver en verde:

```
Changes to be committed:
  modified:   U13/cotizador-hogar/index.html
  modified:   U13/cotizador-hogar/css/styles.css
  new file:   U14/nueva-leccion.html
```

Paso 4: Hacer commit

```bash
git commit -m "Actualizar cotizador y agregar lección 14"
```

Paso 5: Subir a GitHub

```bash
git push
```

📊 PARTE 3: DIFERENTES ESCENARIOS

Escenario A: Agregaste nuevos archivos

```bash
# Ejemplo: creaste U15/nueva-funcionalidad.html
ls U15/  # Verificar que el archivo existe

git status  # Verás "Untracked files"
git add .  # Agregar todos los nuevos archivos
git commit -m "Agregar funcionalidad de U15"
git push
```

Escenario B: Modificaste archivos existentes

```bash
# Ejemplo: editaste styles.css
git status  # Verás "Changes not staged for commit"
git add .
git commit -m "Mejorar estilos responsive del cotizador"
git push
```

Escenario C: Eliminaste archivos

```bash
# Ejemplo: borraste archivo obsoleto
rm U13/archivo-viejo.html

git status  # Verás "Changes not staged for commit: deleted"
git add .  # Esto también registra las eliminaciones
git commit -m "Eliminar archivo obsoleto de U13"
git push
```

Escenario D: Agregaste una carpeta completa

```bash
# Ejemplo: creaste carpeta U16 con varios archivos
git status  # Verás la carpeta como "Untracked files"
git add U16/  # Agregar solo esa carpeta
# o
git add .    # Agregar todo

git commit -m "Agregar lección completa U16 - APIs"
git push
```

🔍 PARTE 4: COMANDOS DE VERIFICACIÓN

Antes de hacer cambios:

```bash
# ¿Dónde estoy?
pwd

# ¿Qué archivos tengo?
ls -la

# ¿Cuál es el estado del repositorio?
git status
```

Durante el proceso:

```bash
# Ver qué archivos están en staging
git status

# Ver diferencias específicas
git diff              # Cambios no preparados
git diff --cached     # Cambios preparados para commit

# Ver últimos commits
git log --oneline -5     # Últimos 5 commits
```

Después de subir:

```bash
# Verificar que todo se subió
git status  # Debería decir "nothing to commit, working tree clean"

# Ver el historial
git log --oneline

# Verificar conexión remota
git remote -v
```

⚠️ PARTE 5: ERRORES COMUNES Y SOLUCIONES

Error: "Authentication failed"

```bash
# Síntoma: Falla al hacer git push
# Solución: Verificar credenciales
Username: gustavomanfre
Password: [TU PERSONAL ACCESS TOKEN]
```

Error: "Nothing to commit"

```bash
# Síntoma: git commit no hace nada
# Causa: No agregaste archivos con git add
# Solución:
git add .
git commit -m "mensaje"
```

Error: "Updates were rejected"

```bash
# Síntoma: git push rechazado
# Causa: GitHub tiene cambios que no tienes localmente
# Solución:
git pull origin main  # Descargar cambios primero
git push              # Luego subir
```

Error: "Untracked files"

```bash
# Síntoma: git status muestra archivos en rojo
# No es error: Son archivos nuevos
# Solución:
git add .  # Para incluirlos
# o crear .gitignore para excluirlos
```

🎯 PARTE 6: FLUJOS ESPECÍFICOS PARA DIFERENTES TIPOS DE TRABAJO

Flujo para sesión de trabajo corta:

```bash
# Al empezar
cd ~/Documentos/repositoy-node.js
git status

# Trabajar (crear/modificar archivos)
# ...

# Al terminar
git add .
git commit -m "Avances de la sesión: [describir qué hiciste]"
git push
```

Flujo para proyecto grande:

```bash
# Commits frecuentes (recomendado)
git add archivo-completado.html
git commit -m "Completar ejercicio 1 de U15"

git add archivo-2.html
git commit -m "Completar ejercicio 2 de U15"

# Al final del día
git push  # Sube todos los commits juntos
```

Flujo para experimentación:

```bash
# Crear rama para experimentos (opcional avanzado)
git branch experimento
git checkout experimento

# Experimentar...
git add .
git commit -m "Probar nueva funcionalidad"

# Si funciona, volver a main y mergear
git checkout main
git merge experimento
git push
```

📝 PARTE 7: COMANDOS DE REFERENCIA RÁPIDA

Flujo básico diario:

```bash
cd ~/Documentos/repositoy-node.js  # Ir al proyecto
git status                 # Ver estado
git add .                  # Preparar cambios
git commit -m "descripción"      # Guardar cambios
git push                   # Subir a GitHub
```

Comandos informativos:

```bash
git status                 # ¿Qué está pasando?
git log --oneline          # ¿Qué he hecho?
git diff                   # ¿Qué cambió?
git remote -v              # ¿A dónde se conecta?
```

Comandos de corrección:

```bash
git add archivo-olvidado.html    # Agregar archivo específico
git commit --amend -m "nuevo mensaje"  # Cambiar último commit
git reset --soft HEAD~1        # Deshacer último commit (mantener cambios)
```

🔄 PARTE 8: SINCRONIZACIÓN CON EL TRABAJO

En tu PC del trabajo:

Primera vez:

```bash
cd ~/Documentos
git clone https://github.com/gustavomanfre/repository-node.js.git
cd repository-node.js
```

Cada día:

```bash
cd ~/Documentos/repository-node.js
git pull  # Descargar cambios de casa
```

Si haces cambios en el trabajo:

```bash
git add .
git commit -m "Cambios hechos en el trabajo"
git push
```

De vuelta en casa:

```bash
cd ~/Documentos/repositoy-node.js
git pull  # Descargar cambios del trabajo
```

✅ PARTE 9: LISTA DE VERIFICACIÓN

Antes de cada sesión:

- [ ] Estoy en la carpeta correcta (`pwd`)
- [ ] Tengo la última versión (`git pull` si trabajo en múltiples PCs)
- [ ] Veo el estado limpio (`git status`)

Durante el trabajo:

- [ ] Guardo archivos frecuentemente (`Ctrl+S`)
- [ ] Hago commits pequeños y frecuentes
- [ ] Uso mensajes descriptivos en commits

Al terminar cada sesión:

- [ ] `git status` para ver todos los cambios
- [ ] `git add .` para preparar todo
- [ ] `git commit -m "mensaje claro"`
- [ ] `git push` para subir a GitHub
- [ ] `git status` para verificar que está limpio

🎉 EJEMPLOS PRÁCTICOS CON TU PROYECTO

Ejemplo 1: Agregaste una nueva unidad

```bash
# Creaste carpeta U16 con archivos
cd ~/Documentos/repositoy-node.js
git status  # Verás "Untracked files: U16/"
git add .
git commit -m "Agregar Unidad 16: Introducción a APIs REST"
git push
```

Ejemplo 2: Actualizaste el cotizador

```bash
# Modificaste archivos en U13/cotizador-hogar/
git status  # Verás archivos modificados en rojo
git add U13/cotizador-hogar/
git commit -m "Mejorar validación de formulario en cotizador"
git push
```

Ejemplo 3: Agregaste documentación

```bash
# Creaste archivos README o documentos
git add "*.md" "*.pdf"  # Solo documentación
git commit -m "Actualizar documentación del proyecto"
git push
```

🚨 REGLAS DE ORO

1. **Siempre `git status` antes de cualquier cosa**
2. **Commits frecuentes con mensajes claros**
3. **`git push` al final de cada sesión de trabajo**
4. **`git pull` al inicio si trabajas en múltiples PCs**
5. **No uses `git push -f` a menos que sepas exactamente por qué**

📚 COMANDOS PARA MEMORIZAR

Los 4 comandos esenciales:

```bash
git status    # Ver qué pasa
git add .     # Preparar cambios
git commit -m "mensaje"  # Guardar cambios
git push      # Subir a GitHub
```

¡Con estos 4 comandos puedes manejar el 90% de tu trabajo diario!