GAÑANADA Nº 010

El commit que nunca debió abandonar el ordenador de su creador

· · ·

CATEGORÍA	GIT Y OTRAS TRAGEDIAS
GRAVEDAD	■■■■■ V — PATRIMONIO HISTÓRICO DE LA GAÑANERÍA
FECHA DEL HALLAZGO	18 DE NOVIEMBRE DE 2026
ETIQUETAS	#git #secretos #credenciales #historial #datos-personales

§ 1

Descripción del hallazgo

El espécimen, conservado en el historial del repositorio interno, presenta la siguiente ficha:

```
$ git show --stat 3f9c2e1
commit 3f9c2e1 (HEAD -> main, origin/main)
Date:   Fri Nov 13 18:52:07 2026 +0100

    cambios

 .env                                |    12 +
 config/produccion.yml               |     4 +-
 dump_produccion_FINAL_v2_bueno.sql  | Bin 0 -> 3412885504 bytes
 node_modules/...                    | (48 213 archivos)
 src/pagos/PasarelaService.ts        |     1 +
```

Un solo commit con cinco problemas, un mensaje que no describe ninguno y una línea de código que, según su autor, «era lo único que quería subir».

El archivo `.env` contenía las claves de acceso a la cuenta de nube de producción, con permisos de administración. El volcado contenía la base de datos de clientes completa: nombres, direcciones, teléfonos e historial de pedidos.

§ 2

Contexto del descubrimiento

La secuencia, reconstruida a partir de los *reflogs* y de los registros del servidor, fue la siguiente:

1. A las 18:49, el autor ejecutó `git add .` para incluir su cambio de una línea.
2. A las 18:52, `git commit -m "cambios"` y `git push`. El servidor rechazó el envío porque dos compañeros habían subido trabajo esa tarde.
3. A las 18:53, `git push --force`. El envío se aceptó. El trabajo de los compañeros desapareció de la rama principal.
4. A las 18:58, consciente de que «había subido algo raro», el autor ejecutó `git rm .env dump_produccion_FINAL_v2_bueno.sql`, hizo un commit titulado «quitado lo sensible» y se fue a casa.

El segundo commit eliminó los archivos de la versión actual, pero no del historial: cualquier `git checkout 3f9c2e1` los devuelve intactos. Para entonces, el repositorio ya se había sincronizado en el servidor de integración continua, en su caché de artefactos y en los portátiles de tres personas que hicieron `git pull` antes de cenar.

El hallazgo se produjo el lunes, cuando la integración continua empezó a fallar por falta de espacio en disco y alguien preguntó por qué el repositorio pesaba 3,5 GB.

§ 3

Consecuencias observadas

- **Credenciales comprometidas.** Desde el momento del push, las claves deben darse por expuestas, aunque el repositorio sea privado. Hubo que rotarlas todas y revisar los registros de acceso de la cuenta de nube.
- **Datos personales replicados** en servidores, cachés y equipos de trabajo, con la obligación de evaluar el incidente como una posible violación de seguridad de datos personales.
- **Trabajo ajeno destruido** por el push forzado. Se recuperó gracias al *reflog* local de uno de los compañeros afectados, que aún no había actualizado su copia.
- **Reescritura del historial**, que obligó a coordinar a todo el equipo para limpiar el repositorio con `git filter-repo` y volver a clonarlo, y a invalidar las cachés de la integración continua.

§ 4

Recomendación de la academia

Git es muy bueno recordando. Esa es precisamente su función, y conviene tenerlo presente antes de pedirle que recuerde algo.

Antes de cada commit, revisa lo que vas a subir: `git status` y `git diff --staged`. Añade archivos por nombre, no con `git add .` a ciegas, y escribe mensajes que describan el cambio. Si el mensaje honesto sería «cinco cosas no relacionadas», son cinco commits.

Protege el repositorio desde el primer día:

```
# .gitignore mínimo de supervivencia
.env
.env.*
*.sql
*.dump
node_modules/
```

- **Detección de secretos** antes del commit (por ejemplo, Gitleaks como *hook* de pre-commit) y en el servidor, rechazando los envíos que contengan credenciales.
- **Rama principal protegida**: sin push forzado, con integración mediante *pull request*. En tus propias ramas, usa `git push --force-with-lease`, que se niega a sobrescribir trabajo que no has visto.
- **Ningún dato personal real en el repositorio.** Para pruebas, datos sintéticos o anonimizados.

Si el secreto ya ha salido, el orden importa: primero rota las credenciales —asume que están comprometidas—; después limpia el historial con `git filter-repo`, coordinando al equipo; y por último revisa qué accesos se produjeron mientras estuvieron expuestas. Borrar el archivo en un commit nuevo no es una solución: es una nota a pie de página que señala dónde está el tesoro.



No trabajes así, no seas un gañán.

REFERENCIAS

1 GitHub Docs — «Removing sensitive data from a repository»

2 git-filter-repo

3 Gitleaks — detección de secretos en repositorios

Separata del volumen I de la Bitácora del Gañán, publicada por AXXA.
https://axxa.embeddedsource.es/bitacora-del-ganan/010-el-commit-que-nunca-debio-abandonar-el-ordenador