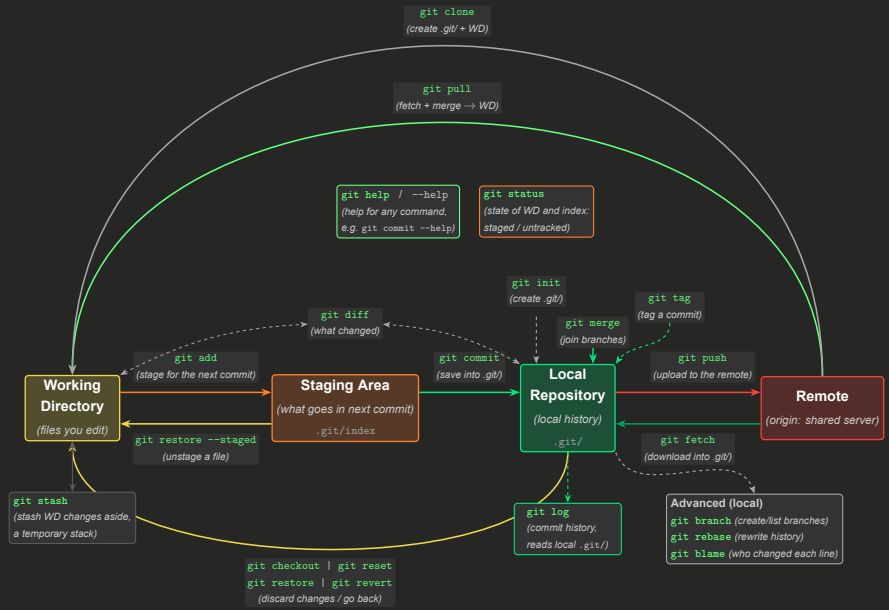
Git para operaciones (1/2)



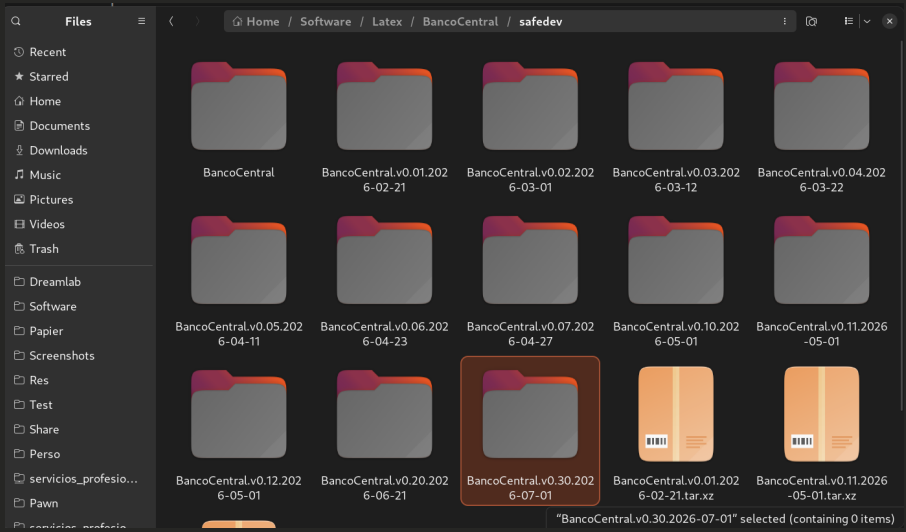
Introducción: contexto

Taller de Git, primera de dos sesiones.

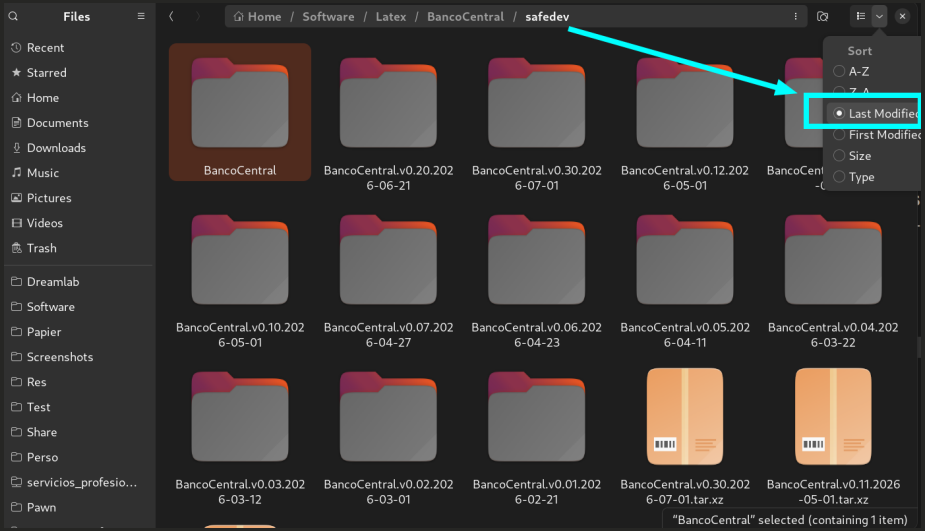
- Para quienes trabajan con archivos en formato textual.
- Hoy: guardar mi propio trabajo, buscarlo y volver atrás.
- La próxima: compartirlo, entregarlo y repararlo.

Git me sirve primero para no perder mi propio trabajo.

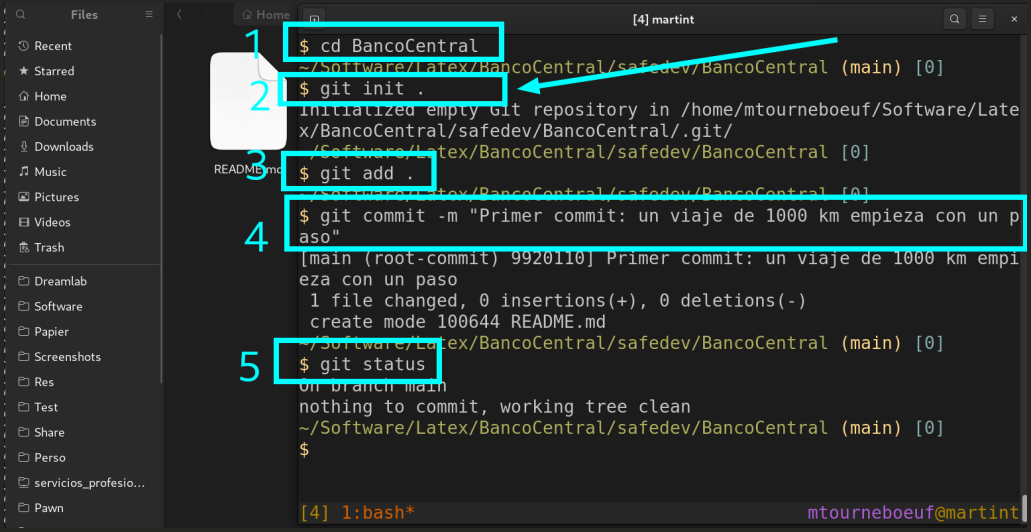
git bisect: el problema



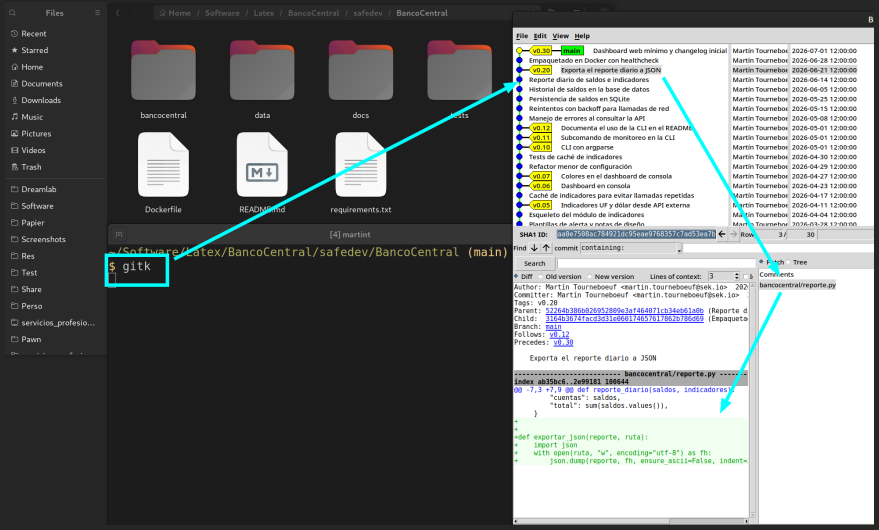
git bisect: el problema



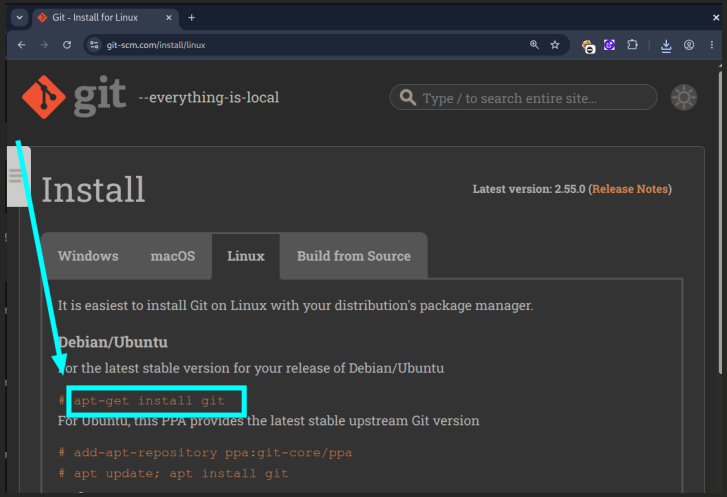
git bisect: la solución



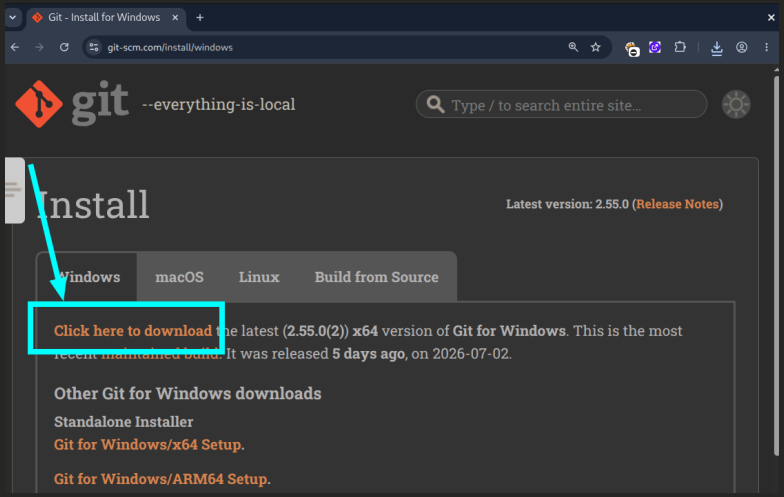
git bisect: la solución



git version: instalar en Linux



git version: instalar en Windows



git fetch: contenido

Capítulo	Sesión 1	Sesión 2	Sesión 3	Sesión 4
Iniciar	help	init	config	status
Capitalizar	diff	add	commit	show
Historiar	log	shortlog	blame	grep
Recuperar	restore	checkout	stash	clean

Capítulo 1: Iniciar

git help

```
git help
git help help
git help -a
```

```
$ git help
usage: git [-v | --version] [-h | [--exec-path[=<path>]]
           [--git-dir=<path>] [--work-tree=<path>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone     Clone a repository into a new directory
   init      Create an empty Git repository or reinitialize

work on the current change (see also: git help everyday)
   add       Add file contents to the index
   mv        Move or rename a file, a directory, or a symlink
   restore   Restore working tree files
   rm        Remove files from the working tree
```

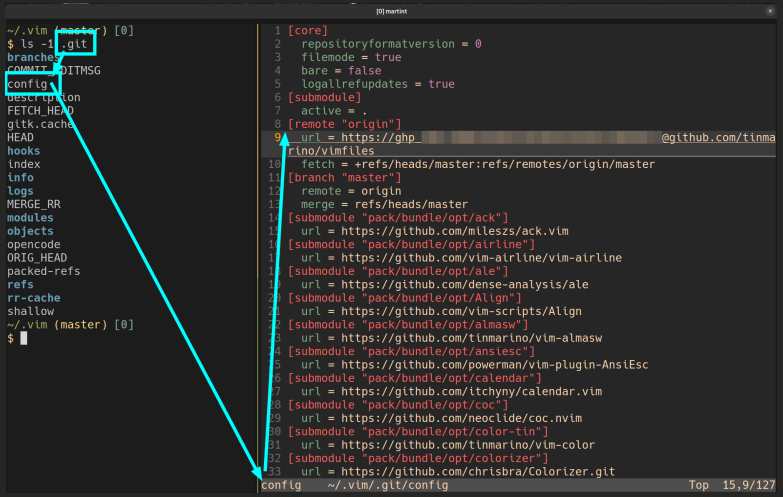
git init

Convertir una carpeta en un repositorio. Se hace una vez.

```
cd ~/Software/Python/Stock

git init .
# Initialized empty Git repository
```

git init: la carpeta oculta `.git`



Un repositorio = tu carpeta + su memoria.

git config

Configura una vez; ahorra para siempre.

```
git config --global user.name  "Tu Nombre"
git config --global user.email "tu@correo.cl"

git config --global alias.lg \
  "log --oneline --graph --all --decorate"
git lg          # tu log bonito en 2 letras
```

git config: `.gitignore`, lo que nunca debe entrar

Los datos y los secretos no son código: no se versionan.

```
# File: .gitignore, .git/ignore or  ~/.gitignore
*.csv         # datos
.env          # secretos
pdf/          # descargas
# --------

git check-ignore -v pdf/ibd010726.pdf   # ¿qué regla lo ignora?
git rm --cached data/secretos.env       # dejar de trackear
```

Un secreto commiteado es un secreto filtrado.

git status

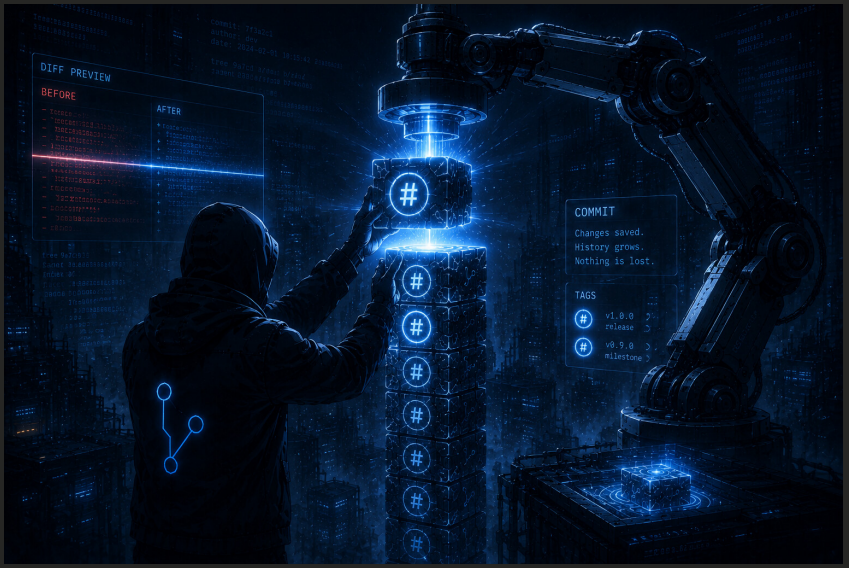
¿Qué cambió desde el último guardado?

```
git status
# On branch main
# Changes not staged for commit:
#   modified:   stock/monitor.py
# Untracked files:
#   data/transacciones_ejemplo.csv
```

- Untracked: Git aún no lo conoce.
- Modified: lo conoce y cambió.

El comando que más se escribe.

Capítulo 2: Capitalizar



git diff

Ver exactamente qué cambió, línea por línea.

```
git diff
# --- a/stock/monitor.py
# +++ b/stock/monitor.py
# -    return {c: s for c, s in ... if s < umbral}
# +    return {c: s for c, s in ... if s <= umbral}

git diff -- stock/reporte.py  # solo ese archivo
```

- Rojo (−): lo que había. Verde (+): lo nuevo.
- Mirá el diff antes de guardar.

git diff

Bonus: diff con git sin .git/.

```
git diff \
  --no-index \
  --word-diff-regex=. \
  src-le-bateau-ivre.txt \
  dst-le-bateau-ivre.txt \
```

```
diff --git a/src-le-bateau-ivre.txt b/dst-le-bateau-ivre.txt
index 0c242fa..efc226a 100644
--- a/src-le-bateau-ivre.txt
+++ b/dst-le-bateau-ivre.txt
@@ -2,23 +2,23 @@ Le bateau ivre
Arthur Rimbaud

Comme je descendais des Fleuves impassibles,
Je ne me [-s-]{+m+}entis plus guidé par les hal[-e-]{+a+}urs :
Des Peaux-Rouges criards les avaient pris pour [-c-]{+r+}ibles,
Les ayant cloués [-n-]{+d+}us aux p[-o-]{+l+}teaux de couleurs.

J'étais insoucieux de tous les équipages[-,-]{+.+}
Porteur de [-b-]{+d+}lés fl[-a-]{+t+}mands ou de cotons anglais.
Quand avec mes haleur[-s-]{+x+} ont fini ces tapages[-,-]{+.+}
Les Fleuves m'ont laissé descendre où je voulais.

Dans les [-c-]{+h+}lapotements furieux des m[-a-]{+e+}rées,
Moi, l'autre hiv[-e-]{+u+}r, plus sourd que les cerveaux d'enfants,
Je courus ! Et les Pé[-n-]{+r+}insules dém[-a-]{+e+}rrées
N'ont [-p-]{+s+}as subi tohu-bohus plus triomphants.

La tempête a béni mes éveils maritimes.
Plus léger qu'un bouchon j'ai dansé sur les flots
Qu'on appelle rouleurs éternels de victimes[-,-]{+.+}
Dix nuits, sans regretter l'oeil niais des falots !

Plus douce qu'aux enfants la chair des pommes sures,
@@ -42,27 +42,27 @@ L'Aube exaltée ainsi qu'un peuple de colombes,
Et j'ai vu quelquefois ce que l'homme a cru voir !

J'ai vu le soleil bas, taché d'horreurs mystiques,
Illuminant de [-l-]{+p+}ongs figements violets,
Pareils à des acteurs de drames très antiques
Les flots roulant au loin leurs frissons de volets !
```

git add

Elegir qué entra en el próximo guardado (staging).

```
git add stock/monitor.py   # solo este archivo
git add .                  # toda la carpeta
git add -u                 # todo lo cambiado
git add -p    # Partes del archivo. En Vim: Gdiffsplit

git status
# Changes to be committed:
#   modified:   stock/monitor.py
```

Preparar la foto antes de sacarla.

git commit

El punto de restauración, con mensaje que explica el por qué.

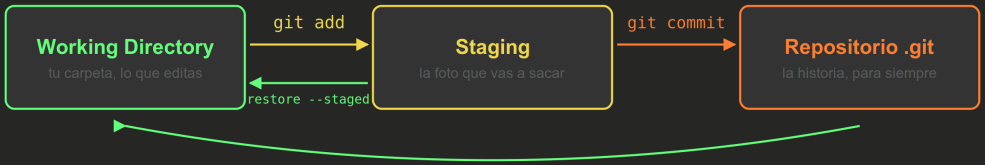
```
git commit -m "Corrige redondeo de montos en CLP"
# [main d1bb173] Corrige redondeo de montos en CLP
#  1 file changed, 3 insertions(+)

git commit --amend --no-edit
```

- Un commit = una idea coherente.
- El mensaje es para el yo del futuro.

Capitalizar el trabajo: queda para siempre.

git commit: las tres áreas



`git restore .`
volver al último commit

`git status` ¿qué cambió?

`git diff` ¿en qué cambió?

nada sale de tu computador

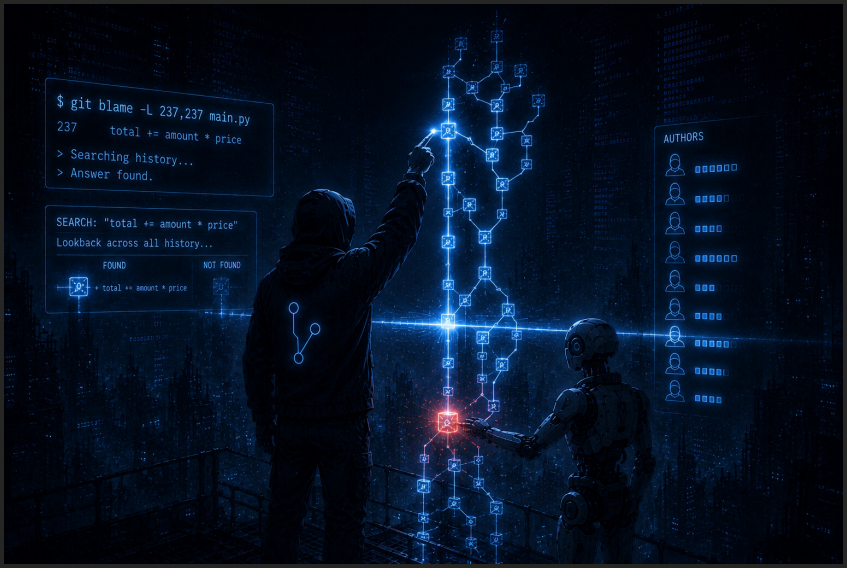
git show

Ver un commit entero, o solo qué archivos tocó.

```
git show d1bb173                # mensaje y diff
git show --stat v0.20           # solo los archivos
git show v0.20:stock/reporte.py # ese archivo, esa versión
```

Un commit es una foto: «show» la revela.

Capítulo 3: Historiar



git log

La historia: reemplaza a los folders con fecha.

```
git log
# commit 0aeb963f7bf92d72d2006697de54c4e8d7b00e91
# Author: Martín Tourneboeuf <martin.tourneboeuf@sek.io>
# Date:   Wed Jul 1 12:00:00 2026 -0400
#
#     Dashboard web mínimo y changelog inicial

git log --oneline
# aa62183 Dashboard web mínimo y changelog inicial
# 3164b36 Empaquetado en Docker con healthcheck
# ...
# 7a1bd87 Estructura inicial del proyecto y README
```

git log --graph


```
git log --oneline --graph --all --decorate
```

git log: filtrar

```
git log --since="2026-07-01"          # por fecha
git log --author="Tourneboeuf"        # por autor
git log --grep="redondeo"             # por mensaje
git log -- stock/reporte.py           # por archivo
git log --follow -- stock/screen.py   # sobrevive al git mv
```

La historia sirve cuando se puede filtrar.

git log: el pickaxe (bonus)

¿Dónde apareció, o desapareció, un texto?

```
git log -S "exportar_json"    # commits donde cambió el conteo
git log -G "timeout=[0-9]+"   # regex sobre el diff

git log -S "AKIA" -p          # ¿cuándo entró esa credencial?
```

El pickaxe encuentra lo que ya fue borrado.

git shortlog

¿Quién hizo qué, en una pantalla?

```
git shortlog -sn
#   142  Martin Tourneboeuf
#    31  Juan Perez
#     4  jenkins-ci

git shortlog -sn --since="1 month ago"
git shortlog                          # commits agrupados por autor
```

El resumen que se pega en un informe.

git grep

Buscar en el código versionado, no en el disco.

```
git grep "exportar_json"           # en la copia actual
git grep "exportar_json" v0.20     # en una versión pasada
git grep -n "password"             # con número de línea
```

git blame

¿Quién escribió esta línea, y en qué commit?

```
git blame stock/monitor.py
# d1bb173 (Martin  2026-07-01) umbral = 0.05
# 3164b36 (Juan    2026-06-12) # TODO: revisar con Riesgo

git blame -L 40,60 stock/monitor.py   # solo esas líneas
```

- No es para culpar: es para preguntarle a la persona correcta.

Capítulo 4: Recuperar

git restore

Descartar cambios no guardados. Volver al último commit.

```
git restore stock/monitor.py    # descarta ese archivo
git restore .                   # descarta lo no guardado

git restore --staged stock/reporte.py   # lo saca del índice
```

«Rompí todo y no había guardado» → un comando.

git checkout

Viajar a una versión anterior para mirar.

```
git checkout v0.05      # ver el proyecto en v0.05
git checkout main       # volver al presente
```

- Sin miedo: nada se pierde, el historial sigue ahí.

git checkout: la navaja suiza

Un solo comando hacía demasiado: cambiar de rama y tocar archivos.

```
git checkout -- stock/cli.py          # descarta cambios
git checkout v0.10 -- stock/cli.py    # de esa versión
```

- Por eso Git moderno lo separó en dos:

```
git switch  <rama>        # cambiar de rama
git restore <archivo>     # descartar cambios
```

‘checkout‘ viejo = ‘switch‘ + ‘restore‘.

git stash

Guardar el trabajo a medias sin hacer commit.

```
git stash push -m "media exportación"   # guarda con nombre
git stash list                          # ver lo guardado
git stash pop                           # recupera y lo borra
git stash apply stash@{1}               # sin borrarlo
```

«Llegó algo urgente» sin perder lo que se estaba haciendo.

git clean

Borra lo que Git no conoce. No hay vuelta atrás.

```
git clean -n     # DRY RUN: muestra qué borraría
git clean -fd    # borra archivos y carpetas untracked
git clean -fdx   # incluye los ignorados (¡cuidado!)
```

Siempre ‘-n‘ antes de ‘-f‘.

Conclusión: lo aprendido

Iniciar	Capitalizar	Historiar	Recuperar
1. help	5. diff	9. log	13. restore
2. init	6. add	10. shortlog	14. checkout
3. config	7. commit	11. blame	15. stash
4. status	8. show	12. grep	16. clean

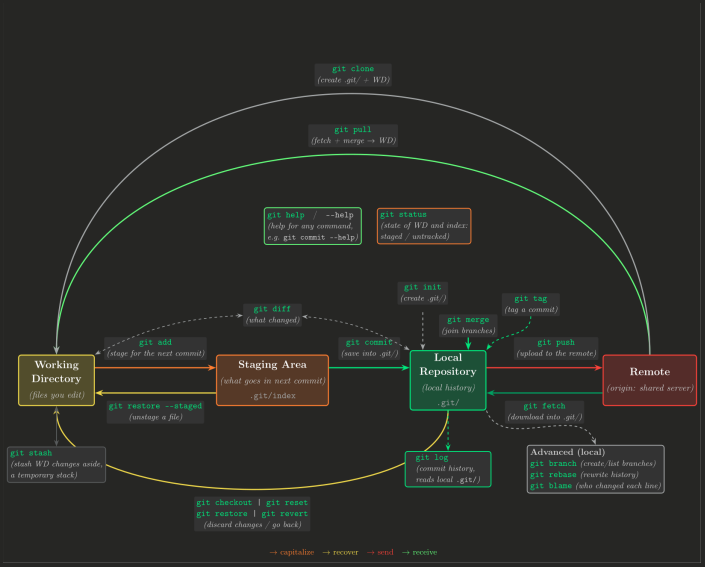
16 comandos, ninguno sale de tu computador.

git worktree: tu rutina diaria

```
git status              # ¿qué cambió?
git diff                # ¿en qué cambió?
git add -p              # elijo qué entra
git commit -m "..."     # capitalizo
```

Guardar seguido, con mensajes claros.

git worktree: las áreas de Git



git grep: ejercicios de historiar

Repo: `git clone https://github.com/tinmarino/stock && cd stock`

#	Reto	Pista
1	Un commit menciona “secreto”. ¿La flag?	`git log --grep`
4	En `pista-v2` hay una flag en un comentario.	`git grep flag- ref`
9	Un commit tiene un autor… sospechoso.	`git log --format=%an`
12	En `pista/nombres` un archivo **se llama** flag.	`git ls-tree -r`
18	Un archivo fue eliminado. ¿Qué decía?	`git log --diff-filter=D`

git grep: ejercicios de capitalizar

#	Reto	Pista
2	`pista/secreto.txt` fue borrado. Léelo.	`git show ref:file`
6	¿Quién escribió “auditoría”? Flag en ese commit.	`git blame` + `show`
13	Asunto inocente; flag en el **cuerpo**.	`git log --format=%b`
15	`pista/huella.txt` dice: “calcula mi SHA”.	`git hash-object`

git grep: ejercicios de recuperar

#	Reto	Pista
3	Flag agregada y luego borrada.	`git log -S`
14	Trabajo guardado con stash.	`git stash show -p`
16	Archivo renombrado; flag en el original.	`git log --follow`

12 flags hoy; las otras 11, la próxima clase.

git grep: flags (solucionario)

#	Flag
1	flag-01-log-grep
2	flag-02-show-file
3	flag-03-pickaxe
4	flag-04-grep-ref
6	flag-06-blame
9	flag-09-author

#	Flag
12	flag-12-ls-tree
13	flag-13-body
14	flag-14-stash
15	flag-15-hash-cc9a7591
16	flag-16-follow
18	flag-18-deleted

git notes: preguntas y referencias

¿Preguntas?

Recurso	URL
Pro Git (libro gratis)	https://git-scm.com/book/es/v2
Learn Git Branching	https://learngitbranching.js.org
Código fuente de Git	https://github.com/git/git
Wikipedia	https://en.wikipedia.org/wiki/Git

La próxima: compartir, entregar (tags) y reparar.