Cosas de Git

Git, esa herramienta para desarrolladores que tantos dolores de cabeza nos da y tanto amor la tenemos

- Como eliminar de nuestro repositorio git un fichero o directorio
- Como forzar un git pull sobre escribiendo los ficheros locales
- Como hacer debug a git para corregir o conocer problemas de conexión
- Busquedas y análisis de cambios en git
- Salvaguarda de main en Producción antes de Deploy

Como eliminar de nuestro repositorio git un fichero o directorio

Introducción

Un error muy común es que algunas veces, con las prisas olvidamos añadir a nuestro `.gitignore` algo que no está en el template con el que trabajamos.

No es correcto dejarnos ciertos ficheros o directorios en el repositorio.

Como eliminar un fichero o directorio en el git (repositorio)

- Primero un backup de nuestro repositorio local, por favor. Siempre un backup.
- Añade el fichero o directorio al fichero `.gitignore`
- En tu `branch` comprueba que el fichero está fuera del master

```
git checkout master -- .gitignore
git add .
git commit -m ‘Update .gitignore’
git pull
```

- Remueve el fichero o directorio del árbol de git

```
git rm --cached -r nombre_de_fichero_o_directorio
git add .
git commit -m ‘Remove nombre_de_fichero_o_directorio’
git push
```

Esta operación sólo elimina del git (repositorio) el fichero, pero no de la historia, ni de las ramas.

Como eliminar un fichero o doriectori en el git (repositorio) en todas las ramas y en la historia

Alguna vez he visto un repositorio que por error contenía datos sensibles, en alguna parte de la historia.

Como norma general, se está usando un método complejo y que es propenso a errores utilizando `git filter-branch` pese a que recibimos un aviso a navegantes.

```
 git filter-branch --tree-filter "rm -f myssh.sh"  --prune-empty HEAD
WARNING: git-filter-branch has a glut of gotchas generating mangled history
         rewrites.  Hit Ctrl-C before proceeding to abort, then use an
         alternative filtering tool such as 'git filter-repo'
         (https://github.com/newren/git-filter-repo/) instead.  See the
         filter-branch manual page for more details; to squelch this warning,
         set FILTER_BRANCH_SQUELCH_WARNING=1.
```

Ni se os ocurra. Es el mejor camino para tener que restaurar la copia y luego forzar un push. Todo ello, peligros para la estabilidad de tu repositorio.

Para hacerlo usaremos `git-filter-repo` , un paquete de **python3** que nos hará la vida más fácil con estas cosas, y que es recomendado por el propio git.

Instalación de git-filter-repo

[Repositorio de git-filter-repo]

```
python3 -m pip install --user git-filter-repo
Collecting git-filter-repo
  Downloading git_filter_repo-2.29.0-py2.py3-none-any.whl (97 kB)
     |████████████████████████████████| 97 kB 967 kB/s
Installing collected packages: git-filter-repo
Successfully installed git-filter-repo-2.29.0
```

Añadimos el comando a nuestro `.bashrc` o `.zshrc` su path. En mi caso la instalación se efectuó en `:$HOME/.local/bin`

```
export PATH=$HOME/bin:$HOME/.local/bin:/usr/local/bin:$PATH
```

Hacemos un reload de nuestro RC o salimos de la sesión para aplicar los cambios.

```
reload
```

Limpieza con git-filter-repo

Ahora, es hora de trabajar con el comando para limpiar nuestro repositorio.

```
❯ git-filter-repo  --path myssh.sh --invert-paths --force
Parsed 108 commits
New history written in 0.06 seconds; now repacking/cleaning...
Repacking your repo and cleaning out old unneeded objects
HEAD está ahora en cce4227 Sin especificar
Enumerando objetos: 405, listo.
Contando objetos: 100% (405/405), listo.
Compresión delta usando hasta 12 hilos
Comprimiendo objetos: 100% (232/232), listo.
Escribiendo objetos: 100% (405/405), listo.
Total 405 (delta 232), reusado 299 (delta 157)
Completely finished after 0.18 seconds.
```

Como veis en la salida, el programa se ha encargado de viajar por la historia de los commits, y realizar la limpieza y cambiar la historia.

Ahora actualizaremos el remoto. Como veremos, los datos de nuestro .git/config han sufrido una actualización (que bueno tener backups si no tenemos o conocemos bien ciertas cosas)

```
git add .
git commit -m 'Update gitignore and clean'
[master 7d77a36] Update gitignore and clean
 1 file changed, 1 insertion(+), 1 deletion(-)
git push
fatal: No se ha configurado un destino para el push.
Puedes o especificar una URL desde la línea de comandos o configurar un repositorio remoto
usando

    git remote add <nombre> <url>

y luego haciendo push al nombre del remoto

    git push <nombre>
```

oops... no pasa nada. Se trata de reconstruir la parte que se ha eliminado de la configuración relativa al repositorio remoto.

```
[remote "origin"]
        url = git@gitlab.castris.com:root/nombre_del_repo.git
        fetch = +refs/heads/*:refs/remotes/origin/*
[branch "master"]
        remote = origin
```

Podemos hacerlo editando `.git/config` o vía comando

```
git remote add  master git@gitlab.castris.com:root/utilidades.git
git remote add  origin git@gitlab.castris.com:root/utilidades.git
git push -f --set-upstream origin master
Enumerando objetos: 405, listo.
Contando objetos: 100% (405/405), listo.
Compresión delta usando hasta 12 hilos
Comprimiendo objetos: 100% (157/157), listo.
Escribiendo objetos: 100% (405/405), 401.98 KiB | 100.50 MiB/s, listo.
Total 405 (delta 232), reusado 405 (delta 232)
remote: Resolving deltas: 100% (232/232), done.
To gitlab.castris.com:root/utilidades.git
 + 1ba0d77...cd97f4c master -> master (forced update)
```

Ahora ya podemos estar tranquilos. Hemos borrado los datos sensitivos de nuestro repositorio.

Agradecimientos y enlaces interesantes

- How to remove the .idea folder from git
- Remove folder and its contents from git/GitHub's history
- How do you install git-filter-repo?
- Remove file from git repository history

Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como está, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.

Como forzar un git pull sobre escribiendo los ficheros locales

Introducción

Algunas veces con git, puede pasar que hay mínimos cambios en local que no son importantes, y queremos hacer un **git pull** para sobre escribir, pero no nos deja (por seguridad)

Cómo forzar un git pull para sobreescribir los ficheros locales

Es importante entender que cualquier cambio en los ficheros locales, se perderá. Todo cambio, con o sin la opción --hard de los commits locales que no hayan sido subidos, se perderán

Opción rápida

```
git pull --rebase
```

Opción opinada

Primero, traer todas las ramas del origen

```
git fetch --all
```

Después hacer un backup de la rama actual en local

```
git branch backup-master
```

Aquí hay dos opciones:

```
git reset --hard origin/<branch_principal>
```

O hacerlo en otra rama

```
git reset --hard origin/<branch_name>
```

Si hay ficherio untracked

```
git clean -fd
```

Explicación

`git fetch` descarga actualizado del remoto sin intentar un `merge` o un `rebase`

Después `git reset` reiniciará la rama master que tú quieres tomar.

Mantener los commits locales pendientes

Es una buena idea muchas veces mantener los cambios locales creando una rama desde el master antes de hacer el reset.

```
git checkout <branch_principal>
git branch new-branch-to-save-current-commits
git fetch --all
git reset --hard origin/<branch_principal>
```

Después de esto, todas las confirmaciones (commits) se mantendrán en la nueva rama, `new-branch-to-save-current-commits`

> Como no, en este tipo de acciones es absolutamente necesario, hacer un backup antes de nada.

Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como está, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.

Como hacer debug a git para corregir o conocer problemas de conexión

Mode debug en comando git en red

Alguna vez, se te puede quedar un comando git remoto (pull, push, etc) colgado, como si no funcionara ñla conexión remota o hubiera problemas ssh.

Solución

GIT_TRACE_CURL

```
$ GIT_TRACE_CURL=true  git clone  https://........
...
17:42:56.835334 http.c:717              <= Recv data:  3
39;....L....qE...5.jl..~..=..................S..6.>Z\.!.
17:42:56.835348 http.c:717              <= Recv data:
......Z%...;0019.h.@.l.'....E........0006..003f.Total 157 (d
17:42:56.835355 http.c:717              <= Recv data: elta 32), reused 88 (delta 28), pack-
reused 570006..0000
remote: Total 157 (delta 32), reused 88 (delta 28), pack-reused 57
Receiving objects: 100% (157/157), 1.33 MiB | 3.02 MiB/s, done.
Resolving deltas: 100% (48/48), done.
17:42:56.840003 http.c:729              == Info: Connection #0 to host gitlab.xxxx.xxx left
intact
```

Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como esta, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.

Busquedas y análisis de cambios en git

Introuducción

A veces tenemos que nadar buscando cosas en Git.

Historial de cambios de un fichero

```
git log --follow -p path/fichero | cat
```

Historial resumido

```
git log --follow --pretty=format:"%h %ad | %s [%an]" --date=short path/fichero| cat


42920a19b 2025-01-07 | Boolena issues elasticksearch [Abdelkarim Mateos Sanchez]
056398efc 2024-10-22 | Migrations tools ELK [Abdelkarim Mateos Sanchez]
62f149c7f 2024-10-14 | Change mapping [Abdelkarim Mateos]
1bdff9d90 2024-09-24 | Add mappings [Abdelkarim Mateos Sanchez]
061764aca 2024-08-07 | Corrections for failed test [Abdelkarim Mateos Sanchez]
74eaf192f 2024-08-07 | Corrections for failed test [Marco Antonio Mateos Sanchez]
3a6aaca47 2024-06-12 | Change mapping to 2024061201 - Typo in mapping [abkrim]
3c1ea3ec5 2024-06-12 | Change mapping to 2024061201 - Typo in mapping [abkrim]
12b369a27 2024-06-12 | Change mapping to 2024061201 [abkrim]
638910eba 2024-04-17 | MApping modems correct objetcs [abkrim]
1863780bc 2024-04-16 | Error in mappings 3 [abkrim]
8f3f45061 2024-04-16 | Error in mappings 2 [abkrim]
55b6043fc 2024-04-02 | Elastic error in mapping [abkrim]
b95710953 2024-03-29 | New mappings [abkrim]
f9fc8280a 2022-11-29 | Remapping modems because worng data offsets [abkrim]
```

Cambios especificos en un revisión

```
git show 42920a19b | cat
```

Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como está, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.

Salvaguarda de main en Producción antes de Deploy

Contexto

Procedimiento para crear una copia de seguridad del branch `main` actual en producción antes de realizar un deploy, permitiendo rollback inmediato si es necesario.

Variables de entorno

```
# Definir al inicio de la sesión
PROYECTO_DIR="/ruta/del/proyecto/sitelight"
FECHA_VERSION=$(date +%Y%m%d)
BACKUP_BRANCH="main_${FECHA_VERSION}"
```

Prerequisitos

- Acceso SSH al servidor de producción
- Permisos de escritura en el repositorio remoto (para push del backup)
- Git configurado correctamente

Proceso paso a paso

1. Conexión y preparación

```
# Conectar al servidor
ssh usuario@servidor-produccion

# Ir al directorio del proyecto
cd "$PROYECTO_DIR"
```

```
# Verificar que estamos en un repositorio git
git rev-parse --git-dir &>/dev/null || { echo "ERROR: No es un repositorio git"; exit 1; }
```

2. Verificación del estado actual

```
# Ver estado del working directory
git status

# Verificar rama actual
CURRENT_BRANCH=$(git branch --show-current)
echo "Rama actual: $CURRENT_BRANCH"

# Verificar si hay cambios sin commitear (CRÍTICO)
if [[ -n $(git status -s) ]]; then
    echo "ADVERTENCIA: Hay cambios sin commitear"
    git status -s
    echo "¿Continuar? (s/n)"
    read -r response
    [[ ! "$response" =~ ^[Ss]$ ]] && exit 1
fi
```

3. Actualizar main local

```
# Fetch sin merge
git fetch origin

# Cambiar a main (si no estamos ya)
git checkout main

# Verificar que estamos en sync con origin
LOCAL=$(git rev-parse main)
REMOTE=$(git rev-parse origin/main)

if [ "$LOCAL" != "$REMOTE" ]; then
    echo "main local y remoto difieren"
    echo "Local:  $LOCAL"
    echo "Remote: $REMOTE"
    echo "Actualizando..."
```

```
    git pull origin main
fi
```

4. Crear backup con fecha

```
# Definir nombre del backup con fecha de hoy
FECHA_VERSION=$(date +%Y%m%d)
BACKUP_BRANCH="main_${FECHA_VERSION}"

# Verificar que no existe ya
if git show-ref --verify --quiet "refs/heads/$BACKUP_BRANCH"; then
    echo "ADVERTENCIA: La rama $BACKUP_BRANCH ya existe"
    echo "Opciones:"
    echo "  1) Usar nombre con timestamp: main_${FECHA_VERSION}_$(date +%H%M)"
    echo "  2) Eliminar la existente y recrear"
    echo "  3) Cancelar"
    read -r option
    case $option in
        1) BACKUP_BRANCH="main_${FECHA_VERSION}_$(date +%H%M)" ;;
        2) git branch -D "$BACKUP_BRANCH" ;;
        *) exit 1 ;;
    esac
fi

# Crear la rama de backup (sin checkout)
git branch "$BACKUP_BRANCH"

# Verificar creación
git show-ref --verify "refs/heads/$BACKUP_BRANCH" || { echo "ERROR: No se pudo crear
$BACKUP_BRANCH"; exit 1; }
```

5. Pushear backup al remoto

```
# Push de la rama de backup
git push origin "$BACKUP_BRANCH"

# Verificar que está en remoto
git ls-remote --heads origin "$BACKUP_BRANCH" || { echo "ERROR: No se pudo pushear"; exit 1; }
```

```
echo "✓ Backup creado: $BACKUP_BRANCH"
```

6. Verificación del backup

```
# Ver último commit del backup
echo "Último commit en $BACKUP_BRANCH:"
git log --oneline -1 "$BACKUP_BRANCH"

# Ver todas las ramas main*
echo "Ramas de backup disponibles:"
git branch -a | grep "main_"

# Comparar backup con main actual (deben ser idénticos)
DIFF=$(git diff main "$BACKUP_BRANCH")
if [ -n "$DIFF" ]; then
    echo "ADVERTENCIA: Hay diferencias entre main y $BACKUP_BRANCH"
    echo "$DIFF"
else
    echo "✓ main y $BACKUP_BRANCH son idénticos"
fi
```

7. Deploy del nuevo main

```
# Ahora es seguro actualizar main
git pull origin main

# Verificar nuevo estado
git log --oneline -5
```

Procedimiento de Rollback

Opción A: Reset hard (reescribe historia)

```
# ADVERTENCIA: Esto reescribe la historia de main
git checkout main
```

```
git reset --hard "$BACKUP_BRANCH"

# Forzar push (requiere permisos)
git push origin main --force

# Verificar
git log --oneline -1
```

Opción B: Revert (preserva historia)

```
# Identificar commits problemáticos
git log --oneline main.."$BACKUP_BRANCH"

# Revertir commits específicos
git revert <hash-commit-problemático>

# Push normal
git push origin main
```

Opción C: Merge del backup

```
# Si hubo cambios en main que queremos deshacer
git checkout main
git merge -s ours "$BACKUP_BRANCH" -m "Rollback to $BACKUP_BRANCH"
git push origin main
```

Script automatizado

```
#!/bin/bash
# backup_main_produccion.sh

set -e  # Exit on error

PROYECTO_DIR="${PROYECTO_DIR:-/ruta/del/proyecto/sitelight}"
FECHA_VERSION=$(date +%Y%m%d)
BACKUP_BRANCH="main_${FECHA_VERSION}"
```

```
echo "=== Backup de main en Producción ==="
echo "Proyecto: $PROYECTO_DIR"
echo "Backup: $BACKUP_BRANCH"
echo

cd "$PROYECTO_DIR"

# Verificar repositorio
git rev-parse --git-dir &>/dev/null || { echo "ERROR: No es un repositorio git"; exit 1; }

# Verificar cambios sin commitear
if [[ -n $(git status -s) ]]; then
    echo "ERROR: Hay cambios sin commitear"
    git status -s
    exit 1
fi

# Actualizar main
echo "Actualizando main..."
git fetch origin
git checkout main
git pull origin main

# Verificar si backup existe
if git show-ref --verify --quiet "refs/heads/$BACKUP_BRANCH"; then
    BACKUP_BRANCH="main_${FECHA_VERSION}_$(date +%H%M%S)"
    echo "AVISO: Usando $BACKUP_BRANCH (ya existía versión diaria)"
fi

# Crear backup
echo "Creando backup: $BACKUP_BRANCH"
git branch "$BACKUP_BRANCH"

# Push backup
echo "Pusheando a remoto..."
git push origin "$BACKUP_BRANCH"

# Verificación
```

```
echo
echo "✓ Backup completado exitosamente"
echo "  Branch: $BACKUP_BRANCH"
echo "  Commit: $(git log --oneline -1 "$BACKUP_BRANCH")"
echo
echo "Ahora puedes hacer deploy con seguridad:"
echo "  git pull origin main"
echo
echo "Para rollback si es necesario:"
echo "  git reset --hard $BACKUP_BRANCH"
```

Verificaciones post-deploy

```
# Comprobar que la aplicación arrancó correctamente
systemctl status nombre-servicio

# Ver logs recientes
journalctl -u nombre-servicio -n 50 --no-pager

# Verificar conectividad/endpoints
curl -I https://dominio.com/health

# Comparar versión actual vs backup
git log --oneline "$BACKUP_BRANCH"..main
```

Limpieza de backups antiguos

```
# Listar backups de más de 30 días
git for-each-ref --format='%(refname:short) %(committerdate:iso8601)' refs/heads/main_* | \
    awk -v cutoff="$(date -d '30 days ago' +%Y-%m-%d)" '$2 < cutoff {print $1}'

# Eliminar backups locales antiguos
git branch -D main_20241001 main_20241002  # etc

# Eliminar del remoto
git push origin --delete main_20241001 main_20241002
```

Troubleshooting

"cannot push: rejected"

```
# Verificar permisos en remoto
git remote -v
git ls-remote --heads origin

# Si es necesario, forzar push del backup (solo backup, nunca main)
git push origin "$BACKUP_BRANCH" --force
```

"already exists"

```
# Ver cuándo se creó
git log --oneline -1 main_$(date +%Y%m%d)

# Opciones:
# 1. Usar timestamp completo
BACKUP_BRANCH="main_$(date +%Y%m%d_%H%M%S)"

# 2. Eliminar la existente
git branch -D main_$(date +%Y%m%d)
git push origin --delete main_$(date +%Y%m%d)
```

"divergent branches"

```
# Ver diferencias
git log --oneline --graph --all --decorate | head -20

# Forzar sincronización (solo en casos excepcionales)
git reset --hard origin/main
```

Notas importantes

- **Naming convention**: `main_YYYYMMDD` o `main_YYYYMMDD_HHMM` si hay múltiples backups en un día

- **Retención**: Mantener backups de al menos 30 días
- **Remoto**: Siempre pushear el backup al remoto (protección ante pérdida del servidor)
- **Verificación**: Confirmar que backup y main son idénticos antes del deploy
- **Rollback**: `reset --hard` es más rápido pero reescribe historia; `revert` preserva historia

---

Última actualización: 2025-11-07
Validado en: Git 2.x
Entorno: Producción sitelight

Aviso

Esta documentación y su contenido, no implica que funcione en tu caso o determinados casos. También implica que tienes conocimientos sobre lo que trata, y que en cualquier caso tienes copias de seguridad. El contenido el contenido se entrega, tal y como está, sin que ello implique ningún obligación ni responsabilidad por parte de Castris

Si necesitas soporte profesional puedes contratar con Castris soporte profesional.