**GIT en Kali**

Primer comprova si git està instal·lat al teu sistema amb alguna ordre senzilla:

**git --version**

Per instal·lar Git en Kali Linux, segueix aquests passos:

1. Actualitza els repositoris de paquets:

**sudo apt update**

2. Instal·la Git amb:

**sudo apt install git**

3. Verifica la instal·lació:

**git --version**

Això et mostrarà la versió de Git instal·lada. Després, pots configurar el teu nom d'usuari i correu amb les ordres següents:

**git config --global user.name "el-teu-nom"**

**git config --global user.email "el-teu-email"**

**Creació d'un projecte Git**

Crea una carpeta per al projecte:

**mkdir meuprojecte**

**cd meuprojecte**

Inicia Git en aquesta carpeta:

**git init**

Git crearà un repositori per seguir els canvis en aquesta carpeta.

**Afegir fitxers nous a Git**

Un cop creat el teu primer repositori local, estarà buit. Per tant, pots afegir un fitxer de text senzill amb un editor com `nano`:

1. Crea el fitxer:

**nano fitxer.txt**

2. Escriu-hi alguna cosa i desa'l.

3. Comprova l'estat del repositori:

**git status**

Veurem que el fitxer és "untracked" (no controlat). Perquè Git el comenci a seguir, afegeix-lo a l'escenari (staging area) o àrea de preparació:

**git add fitxer.txt**

Ara el fitxer està preparat per ser "commitejat".

*Els fitxers dins d'un repositori Git poden estar en dos estats principals:*

*1.* ***Tracked*** *(seguit): Aquests són els fitxers que Git coneix i que ja han estat afegits al repositori o que s'han modificat des de l'últim commit. Un cop es commitegen, es pot fer el seguiment dels canvis que es facin en aquests fitxers.*

*2.* ***Untracked*** *(no seguit): Aquests són els fitxers que es troben a la teva carpeta de treball, però que Git no està seguint perquè encara no han estat afegits a l'staging area. Quan crees nous fitxers en un repositori, inicialment es troben en aquest estat.*

*Exemple: Quan treballes amb un repositori buit, tots els fitxers seran untracked. Perquè Git comenci a seguir-los, has d'afegir-los a l'staging area utilitzant l'ordre:*

***git add <nom\_fitxer>***

*Aquesta acció mou els fitxers a l'estat de tracked i els prepara per al proper commit, on quedaran registrats al repositori. L'staging area és una etapa intermèdia que et permet revisar i preparar els canvis abans de guardar-los de manera permanent amb un commit.*

**Entorn d'staging de Git**

Un dels conceptes clau de Git és l'entorn d'staging i els commits. Mentre treballes en el teu projecte, pots afegir, editar i eliminar fitxers. Quan arribes a un punt important o acabes una part del treball, pots afegir els fitxers a l'entorn d'staging.

Els fitxers que es troben en l'entorn d'staging estan preparats per ser commitejats al repositori. Ara veurem com afegir fitxers a aquest entorn.

Suposem que estem treballant amb un fitxer de text anomenat `fitxer.txt`.

Afegir un fitxer a l'entorn d'staging

Un cop hem acabat de treballar amb el fitxer de text, l'afegim a l'entorn d'staging:

**git add fitxer.txt**

Comprovar l'estat:

Per veure si el fitxer s'ha afegit correctament a l'entorn d'staging, executa:

git status

Exemple de sortida:

**On branch master**

**No commits yet**

**Changes to be committed:**

**(use "git rm --cached ..." to unstage)**

**new file: fitxer.txt**

Ara el fitxer `fitxer.txt` ha estat afegit a l'entorn d'staging.

Afegir més d'un fitxer

També pots afegir més d'un fitxer a l'entorn d'staging alhora. Per exemple, crea dos fitxers addicionals: `README.md` i `config.cfg`.

- `README.md` (fitxer que descriu el repositori):

**# Exemple de repositori**

**Això és un repositori d'exemple per al tutorial de Git.**

- `config.cfg` (un fitxer de configuració senzill):

**Configuració del sistema: versió 1.0**

Per afegir tots els fitxers alhora, utilitza:

**git add --all**

L'ordre `--all` afegeix tots els canvis (nous fitxers, modificacions i eliminacions).

Comprovar l'estat de nou:

**git status**

Sortida:

**Changes to be committed:**

**(use "git rm --cached ..." to unstage)**

**new file: fitxer.txt**

**new file: README.md**

**new file: config.cfg**

**Nota:**

La comanda abreujada per git add --all és **git add -A**

Ara tots tres fitxers (`fitxer.txt`, `README.md` i `config.cfg`) estan a l'entorn d'staging i estan preparats per ser commitejats.

Resum:

- Afegir fitxers a l'staging: `git add <nom\_del\_fitxer>`

- Afegir tots els fitxers: `git add --all`

- Comprovar l'estat: `git status`

**Git Commit**

Un cop hem acabat de treballar i hem afegit els fitxers a l'entorn d'staging, estem preparats per fer el commit. Fer un commit a Git crea un punt de restauració que guarda els canvis en el repositori. Cada commit actua com un "punt de guardat", al qual pots tornar en cas que detectis un error o vulguis fer modificacions.

Quan fem un commit, és recomanable afegir un missatge clar que expliqui què s'ha canviat. Això ajuda tant a tu com a altres col·laboradors a identificar fàcilment els canvis.

Exemple de commit:

**git commit -m "Primera versió de fitxer.txt"**

Sortida:

**[master (root-commit) 221ec6e] Primera versió de fitxer.txt**

**3 files changed, 26 insertions(+)**

**create mode 100644 fitxer.txt**

**create mode 100644 README.md**

**create mode 100644 config.cfg**

Aquesta ordre fa el commit dels fitxers en l'entorn d'staging al repositori, i el missatge "Primera versió de fitxer.txt" indica què s'ha fet.

**Git Commit sense passar per l'staging**

En alguns casos, per canvis petits o ràpids, pot semblar una pèrdua de temps passar per l'entorn d'staging. Git permet fer un commit directament dels fitxers modificats utilitzant l'opció `-a`. Això només afectarà fitxers que ja estiguin seguits per Git (és a dir, fitxers que han estat afegits en commits anteriors).

Suposem que fem una petita modificació en `fitxer.txt`, afegint una línia nova:

**Hola, aquest és el meu primer fitxer en Git!**

**Aquesta és una línia nova al fitxer.**

Comprovar l'estat de manera compacta:

Abans de fer el commit, pots comprovar l'estat del repositori amb el format compacte (`--short`):

**git status --short**

Sortida:

M fitxer.txt

On:

- `M` indica que el fitxer està modificat.

- `??` són fitxers no seguits.

- `A` són fitxers afegits a l'staging.

Fer un commit directe amb l'opció `-a`:

Ara pots fer el commit directament sense passar per l'staging:

**git commit -a -m "Actualització de fitxer.txt amb una línia nova"**

Sortida:

**[master 09f4acd] Actualització de fitxer.txt amb una línia nova**

**1 file changed, 1 insertion(+)**

**Nota de precaució:**

No és recomanable saltar-se l'entorn d'staging, ja que podries acabar fent commit de canvis no desitjats. És millor utilitzar l'entorn d'staging per revisar exactament quins fitxers i canvis vols incloure en el commit.

**Treballant amb branques a Git**

Una branca a Git és una versió separada del repositori principal, que et permet treballar en canvis importants sense afectar la versió estable del projecte.

Exemple sense Git:

Sense Git, si volguessis fer canvis en un projecte gran (per exemple, redissenyar una pàgina web) sense afectar la versió actual, hauries de crear còpies de fitxers, treballar-hi, gestionar dependències, i assegurar-te que no impactes la versió "en viu". En cas d'una emergència en el projecte, hauries de desar les teves còpies, arreglar l'error i després reintegrar els teus canvis, cosa que pot ser complexa i propensa a errors.

Exemple amb Git:

Amb Git, pots crear una branca nova per al disseny, com ara `new-design`, i treballar-hi sense impactar la branca principal (`master`). En cas d'una emergència, pots crear una altra branca, com ara `error-fix`, arreglar l'error, i després fusionar la branca amb la principal. Quan acabes el disseny, fusionar la branca `new-design` amb la principal també inclourà l'arranjament de l'error.

Crear una nova branca

Per afegir noves funcionalitats al teu fitxer `fitxer.txt`, pots crear una branca nova per evitar afectar el projecte principal. Utilitza:

**git branch nova-funcio**

Això crea una branca anomenada `nova-funcio`.

Comprovar branques existents:

git branch

Sortida:

**nova-funcio**

**\* master**

L'asterisc (\*) indica que estàs a la branca `master`.

Canviar de branca

Per canviar a la branca `nova-funcio`, fes servir:

**git checkout nova-funcio**

Ara estàs treballant a la nova branca. Pots fer canvis sense afectar la branca principal.

Fer canvis i afegir fitxers

Per exemple, afegim una línia nova al fitxer `fitxer.txt`. Després de fer els canvis, el contingut podria ser:

**Hola, aquest és el meu primer fitxer en Git!**

**Aquesta és una línia nova afegida des de la branca nova-funcio.**

Comprova l'estat:

**git status**

Sortida:

**Changes not staged for commit:**

**modified: fitxer.txt**

Això indica que `fitxer.txt` està modificat però no està a l'entorn d'staging.

Afegir fitxers a l'staging:

Per afegir tots els canvis a l'staging, utilitza:

**git add --all**

Comprova l'estat de nou:

**git status**

Sortida:

**Changes to be committed:**

**modified: fitxer.txt**

Fer un commit

Un cop estiguis satisfet amb els canvis, pots fer un commit a la branca:

**git commit -m "Afegida línia nova a fitxer.txt"**

Sortida:

**[nova-funcio 0312c55] Afegida línia nova a fitxer.txt**

**1 file changed, 1 insertion(+)**

Nota:

Si vols crear una branca nova i canviar-hi directament, pots utilitzar l'ordre següent amb el paràmetre `-b`:

**git checkout -b nova-branca**

**Canviar entre branques a Git**

Ara veurem com de ràpid i fàcil és treballar amb diferents branques i com funcionen.

Actualment estem a la branca `nova-funcio`, on hem fet modificacions a `fitxer.txt`. Si llistem els fitxers del directori, veurem les modificacions que hem fet:

**ls**

**fitxer.txt README.md config.cfg**

El fitxer `fitxer.txt` ha estat modificat en aquesta branca. Si obrim el fitxer, veurem les noves línies afegides.

Ara canviem a la branca `master`:

**git checkout master**

**Switched to branch 'master'**

Llista els fitxers de nou:

**ls**

**fitxer.txt README.md config.cfg**

Encara que els fitxers són els mateixos, el contingut de `fitxer.txt` ha tornat a l'estat original, abans dels canvis de la branca `nova-funcio`.

Aquest exemple mostra com pots treballar en diferents funcionalitats o fixes sense interferir amb la branca principal. Les branques permeten treballar en diferents parts del projecte de manera segura i independent.

Creació d'una branca d'emergència

Suposem que encara no hem acabat amb els canvis a `nova-funcio`, però necessitem arreglar un error a la branca `master`. No volem fer canvis directament a `master`, ni tampoc modificar `nova-funcio`, que encara no està finalitzada. Així que crearem una nova branca per a l'arranjament d'emergència:

**git checkout -b emergency-fix**

**Switched to a new branch 'emergency-fix'**

Ara podem treballar a la branca `emergency-fix` sense afectar la resta del projecte.

Corregir l'error

Imaginem que hem trobat un error en `fitxer.txt` i l'hem de corregir. Afegim una línia nova per mostrar com funcionen les fusions:

**Hola, aquest és el meu primer fitxer en Git!**

**Aquesta línia mostra com funciona la fusió.**

Afegir i commitejar els canvis

Després de corregir l'error, comprovem l'estat del repositori:

git status

Sortida:

**On branch emergency-fix**

**Changes not staged for commit:**

**modified: fitxer.txt**

Ara afegim el fitxer a l'staging i fem el commit:

**git add fitxer.txt**

**git commit -m "Correcció d'emergència a fitxer.txt"**

Sortida:

**[emergency-fix dfa79db] Correcció d'emergència a fitxer.txt**

**1 file changed, 1 insertion(+), 1 deletion(-)**

Ara tenim la correcció d'emergència preparada per ser fusionada amb la branca `master`.

**Fusionar Branques**

Ara que tenim la correcció d'emergència a punt, procedirem a fusionar les branques `master` i `emergency-fix`.

Pas 1: Canviar a la branca `master`

Primer hem de canviar a la branca `master`:

**git checkout master**

**Switched to branch 'master'**

Pas 2: Fusionar les branques

Després, fusionem la branca actual (`master`) amb `emergency-fix`:

**git merge emergency-fix**

Sortida típica:

**Updating 09f4acd..dfa79db**

**Fast-forward**

**fitxer.txt | 2 +-**

**1 file changed, 1 insertion(+), 1 deletion(-)**

Com que `emergency-fix` prové directament de `master` i no s'han fet altres canvis a `master`, Git pot fer un "Fast-forward", actualitzant el cap de la branca `master` a la mateixa posició que `emergency-fix`.

*En Git, una fusió de tipus "Fast-forward" ocorre quan la branca que vols fusionar prové directament de la branca principal, i no s'han fet altres canvis a la branca principal mentre treballaves en l'altra. Això significa que l'historial de commits és lineal i no hi ha divergències entre les dues branques.*

*Per exemple, si crees la branca emergency-fix des de master, treballes només en emergency-fix i no fas cap canvi a master durant aquest temps, quan fusionis emergency-fix amb master, Git pot simplement "avançar ràpidament" (Fast-forward) el cap de master fins al mateix punt que emergency-fix, sense necessitat de combinar històries de commits.*

Pas 3: Eliminar la branca d'emergència

Com que ara les dues branques són idèntiques, podem eliminar `emergency-fix`:

**git branch -d emergency-fix**

**Deleted branch emergency-fix (was dfa79db).**

**Conflicte de Fusions**

Ara, tornem a treballar a la branca `nova-funcio`, on hem fet canvis a `fitxer.txt`. Suposem que afegim una nova línia al fitxer i fem un commit:

**Hola, aquest és el meu primer fitxer en Git!**

**Aquesta línia mostra com funciona la fusió.**

Després de fer el commit, podem intentar fusionar la branca `nova-funcio` amb `master`. Però què passa amb els canvis recents que hem fet a `master`?

Pas 1: Canviar a la branca `master` i fusionar

**git checkout master**

**git merge nova-funcio**

Sortida:

**Auto-merging fitxer.txt**

**CONFLICT (content): Merge conflict in fitxer.txt**

**Automatic merge failed; fix conflicts and then commit the result.**

La fusió ha fallat perquè hi ha un conflicte en el fitxer `fitxer.txt`.

Pas 2: Comprovar l'estat després del conflicte

**git status**

Sortida:

**On branch master**

**You have unmerged paths.**

**(fix conflicts and run "git commit")**

**(use "git merge --abort" to abort the merge)**

**Changes to be committed:**

**new file: fitxer.txt**

**Unmerged paths:**

**(use "git add ..." to mark resolution)**

**both modified: fitxer.txt**

Aquest missatge confirma que hi ha un conflicte en el fitxer `fitxer.txt`, però el fitxer està preparat per ser solucionat.

Pas 3: Resoldre el conflicte

Obrim el fitxer en un editor per veure el conflicte:

**Hola, aquest és el meu primer fitxer en Git!**

**<<<<<<< HEAD Aquesta línia mostra com funciona la fusió.**

**=======**

**Aquesta és una línia nova afegida des de la branca nova-funcio.**

**>>>>>>> nova-funcio**

Les línies entre `<<<<<<< HEAD` i `=======` corresponen a la versió de `master`, mentre que les línies entre `=======` i `>>>>>>> nova-funcio` són de la branca `nova-funcio`.

Resolem el conflicte editant el fitxer per combinar les dues versions:

**Hola, aquest és el meu primer fitxer en Git!**

**Aquesta línia mostra com funciona la fusió.**

**Aquesta és una línia nova afegida des de la branca nova-funcio.**

Pas 4: Afegir el fitxer resolt i finalitzar la fusió

Afegim el fitxer corregit a l'staging i fem el commit final:

**git add fitxer.txt**

**git commit -m "Fusionat amb nova-funcio després de resoldre el conflicte"**

Sortida:

**[master e0b6038] Fusionat amb nova-funcio després de resoldre el conflicte**

Pas 5: Eliminar la branca fusionada

Ara podem eliminar la branca `nova-funcio` ja que hem fusionat els canvis amb èxit:

**git branch -d nova-funcio**

**Deleted branch nova-funcio (was 1f1584e).**

**Git Revert**

L'ordre `git revert` permet desfer els canvis d'un commit anterior, però en lloc de simplement eliminar-lo, crea un nou commit que reverteix els canvis, mantenint així l'historial de commits intacte. Això és útil perquè preserva la traçabilitat i permet veure exactament quan es va fer el canvi i quan es va desfer.

Passos per utilitzar `git revert`:

Pas 1: Identificar el commit a revertir

Primer, has de trobar el commit que vols revertir. Pots utilitzar el log de Git amb l'opció `--oneline` per veure els commits de manera simplificada:

**git log --oneline**

Això mostra els primers set caràcters de cada hash de commit i el missatge del commit, facilitant la identificació del punt al qual vols tornar.

Exemple de sortida:

**52418f7 Just a regular update, definitely no accidents here...**

**9a9add8 Added .gitignore**

**81912ba Corrected spelling error**

Pas 2: Revertir el commit

Un cop localitzat el commit, pots utilitzar l'ordre `git revert` per crear un nou commit que desfa els canvis del commit anterior. Si vols revertir l'últim commit, pots fer servir:

**git revert HEAD --no-edit**

- `HEAD` fa referència al commit més recent.

- L'opció `--no-edit` aplica el missatge de revert per defecte, sense obrir l'editor de missatges.

Exemple de sortida:

**[master e56ba1f] Revert "Just a regular update, definitely no accidents here..."**

Això crea un nou commit que inverteix els canvis del commit anterior (en aquest cas, restaura el fitxer eliminat accidentalment).

Verificació del log

Després d'executar `git revert`, pots comprovar l'historial de commits amb:

**git log --oneline**

Veurem el nou commit que ha revertit el canvi, mantenint un registre clar de totes les modificacions:

e56ba1f Revert "Just a regular update, definitely no accidents here..."

52418f7 Just a regular update, definitely no accidents here...

Nota:

Si vols revertir un commit més antic (no l'últim), pots utilitzar:

**git revert HEAD~x**

On `x` és el nombre de commits enrere que vols revertir (per exemple, `HEAD~1` per revertir el penúltim commit, `HEAD~2` per revertir el tercer més recent, etc.).

**Git Reset**

L'ordre `git reset` es fa servir per tornar el repositori a un estat anterior, descartant qualsevol canvi fet després d'aquell commit. A diferència de `git revert`, que crea un nou commit per desfer els canvis, `git reset` mou el punt de referència del repositori directament al commit especificat, eliminant els commits posteriors del log local.

Passos per utilitzar `git reset`:

Pas 1: Trobar el commit anterior

El primer pas és trobar el commit al qual volem tornar. Per fer-ho, podem utilitzar `git log --oneline` per mostrar un log simplificat dels commits, mostrant els primers set caràcters del hash de cada commit i el missatge del commit:

**git log --oneline**

Exemple de sortida:

**e56ba1f (HEAD -> master) Revert "Just a regular update, definitely no accidents here..."**

**52418f7 Just a regular update, definitely no accidents here...**

**9a9add8 (origin/master) Added .gitignore**

Suposem que volem tornar al commit `9a9add8`, just abans dels canvis que volem desfer.

Pas 2: Fer un `reset` al commit

Un cop trobat el commit, podem utilitzar l'ordre `git reset` per tornar-hi. El commit hash que usem serà el de les primeres set lletres que hem trobat al log:

**git reset 9a9add8**

Això fa que el repositori torni a l'estat en què estava en el commit `9a9add8`, descartant els commits posteriors.

Verificació del log

Per comprovar que la reset s'ha aplicat correctament, podem tornar a executar el log per veure que el repositori ha tornat a l'estat anterior:

**git log --oneline**

Sortida:

**9a9add8 (HEAD -> master, origin/master) Added .gitignore**

**81912ba Corrected spelling error**

Atenció:

Modificar l'historial de commits pot ser perillós, sobretot si treballes en un repositori compartit amb altres usuaris. Fer un reset local està bé per fer canvis al teu repositori personal, però modificar l'historial d'un repositori remot pot causar problemes de sincronització amb altres col·laboradors. És per això que has de tenir cura quan fas un `git reset`, especialment en repositoris remots.

Desfer un reset

Encara que els commits ja no apareixen al log després d'un reset, Git no els esborra del tot. Si coneixes el hash d'un commit eliminat, pots tornar a fer un reset a aquest commit:

**git reset e56ba1f**

Això restaurarà el commit `e56ba1f` al log, recuperant el punt on estàvem abans del reset.

**Git `commit --amend`**

L'ordre `git commit --amend` s'utilitza per modificar l'últim commit, combinant els canvis a l'entorn d'staging amb l'últim commit per crear-ne un de nou. Aquest nou commit substitueix completament l'anterior, cosa que permet modificar tant els fitxers com el missatge del commit sense afegir un nou punt a l'historial.

Modificar un missatge de commit

Un dels usos més comuns de `--amend` és corregir errors en el missatge del commit.

Exemple:

Imagina que fas un commit amb un missatge que conté errors tipogràfics:

**git commit -m "Adding plines to reddme"**

Resultat del commit:

**[master 07c5bc5] Adding plines to reddme**

**1 file changed, 3 insertions(+), 1 deletion(-)**

Per corregir el missatge del commit, pots utilitzar:

**git commit --amend -m "Added lines to README.md"**

Sortida:

**[master eaa69ce] Added lines to README.md**

**1 file changed, 3 insertions(+), 1 deletion(-)**

Ara, el log reflecteix el commit corregit:

**git log --oneline**

**eaa69ce (HEAD -> master) Added lines to README.md**

**9a9add8 (origin/master) Added .gitignore**

Ampliar un commit amb nous fitxers

També pots afegir nous fitxers a l'últim commit. Si, després d'un commit, t'adones que t'has oblidat d'afegir algun fitxer, pots afegir-lo a l'staging i utilitzar `--amend` per incloure'l en el mateix commit.

Exemple:

1. Afegeix els fitxers que faltaven a l'staging:

**git add fitxer-que-faltava.txt**

2. Després, fes el commit modificat:

**git commit --amend**

Això actualitzarà l'últim commit, incloent-hi els nous fitxers.

Atenció:

Modificar l'historial de commits, com amb `--amend`, pot ser perillós si estàs treballant amb un repositori compartit. Evita modificar commits ja enviats a un repositori remot, especialment si altres persones hi treballen, ja que això pot provocar problemes de sincronització.

`git commit --amend` és útil per corregir errors menors o afegir canvis a l'últim commit sense crear un commit nou, mantenint un historial de commits més net i clar.