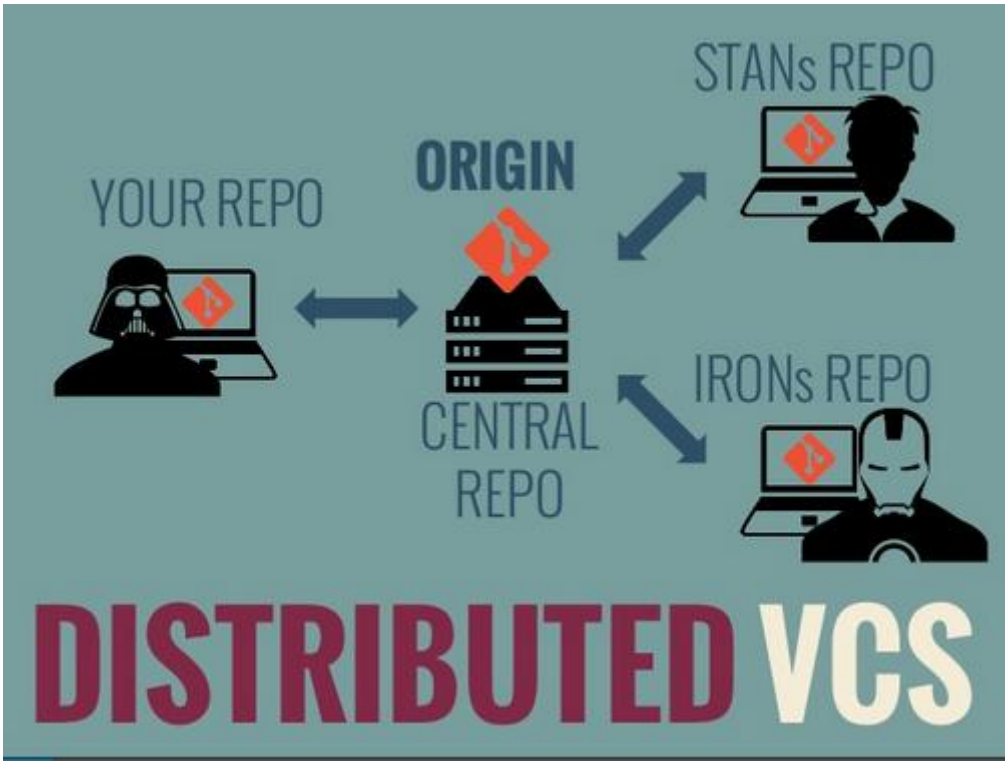
Guia ràpida - GIT - <Command line >

Recordeu, GIT és un gestor de versions (CVS) distribuït. Cada usuari té un repositori de codi amb tota la història.



Treballant només amb REPOSITORI LOCAL

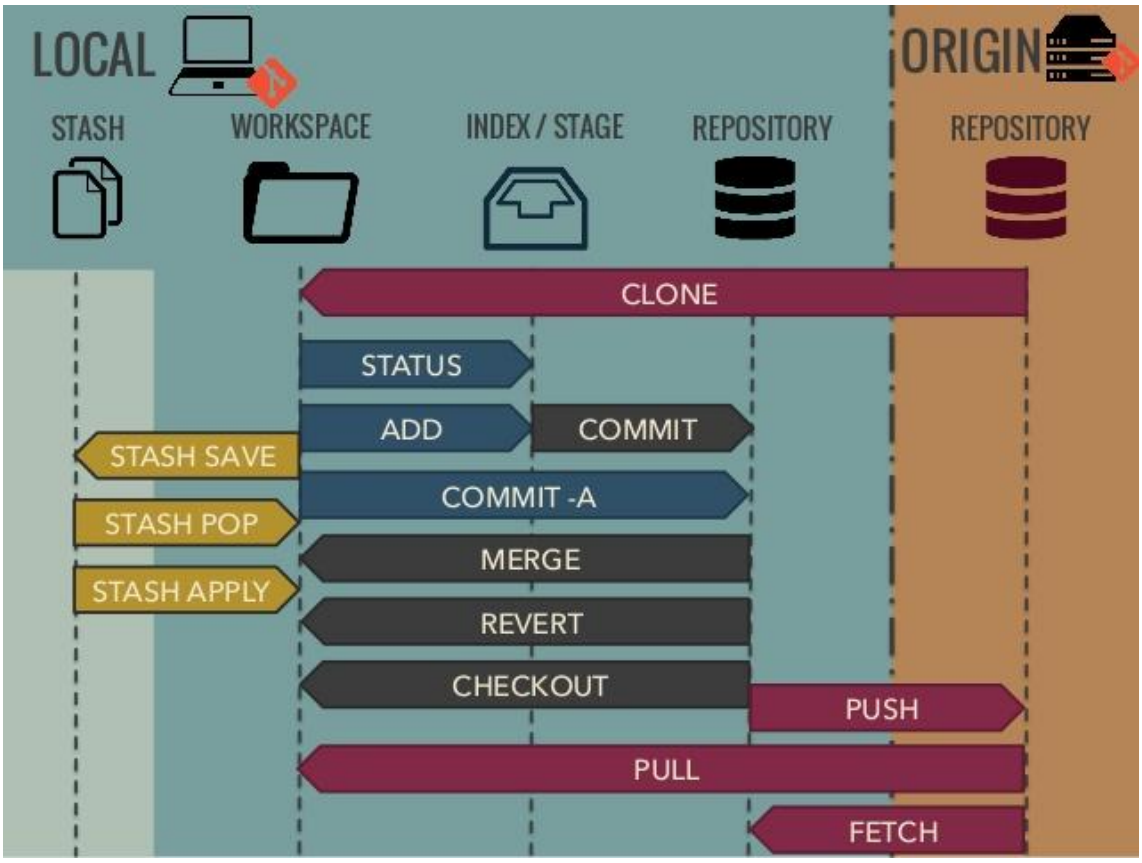
1. Introducció a la gestió de versions
2. Sistemes de control de versions locals, centralitzats i distribuits
3. Setup (GIT Windows)
4. Definició de credencials a GIT

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

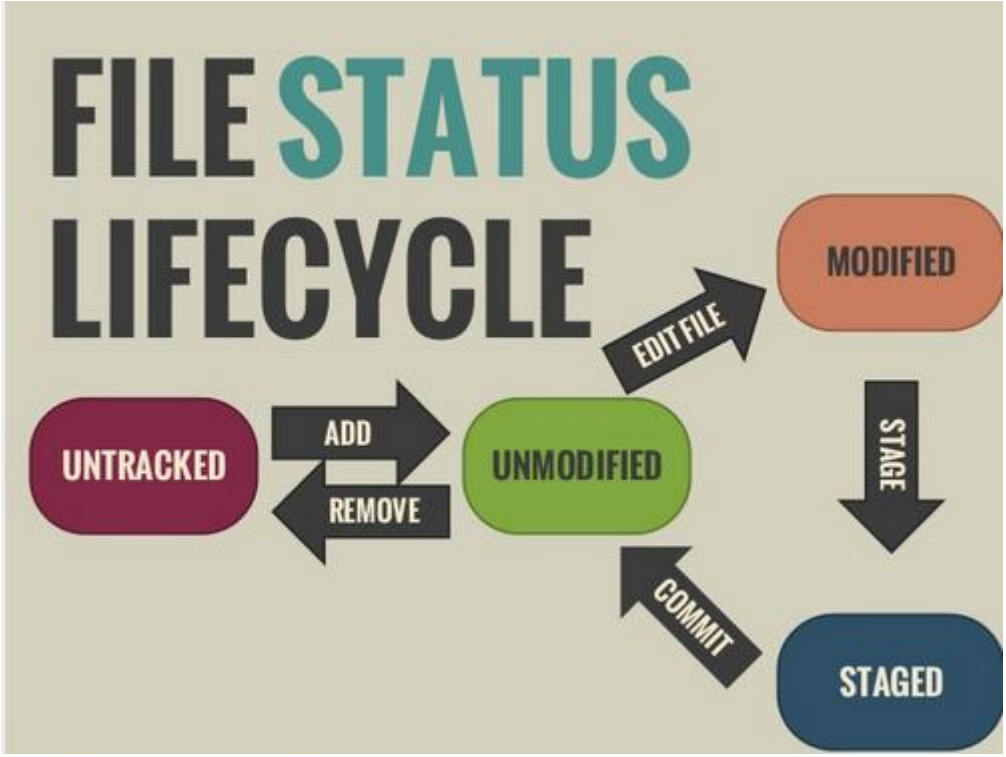
5. Inicialització del repositori local:

```
git init
```

6. Utilització de GIT local



Stash és la zona de còpies ràpides quan està la feina a mitges i no es pot fer un commit, però desitgem fer una operació amb el repositori local.	• arxius unstaged (modificats o esborrats) • arxius untracked (nous)	Stage area és on s'han de posar els arxius per poder aceptar els canvis	Commited accepta els canvis.	



7. Add i commit
   7.1. Passar a Staged els arxius nous, modificats i esborrats :

```
git add . --all
```

   7.2. Commit :

```
git commit –m ‘Missatge’
```

   7.3. Add+commit (**només** arxius modificats, no arxius nous) :

```
git commit –a –m ‘Missatge’
```

8. Llistar tots els canvis respecte la última versió confirmada (últim commit):

```
git diff
```

9. Canvi de nom d’un arxiu

```
git mv [nom_vell] [nom_nou]git
```

10. **SI L'HEM CAGAT**  . Operacions per arreglar destrosses:

10.1. Desfer els canvis "staged" (desfer git add):

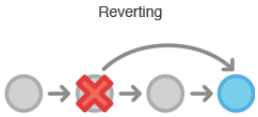
```
git reset nom_arxiu
```

10.2. Descartar tots els canvis respecte la última versió confirmada (últim commit):

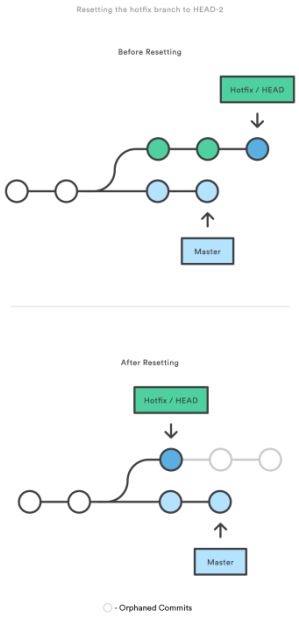
```
// a partir de la carpeta actual, recursivament
git checkout -- .
// de tot l'arbre
git reset --hard HEAD
```

10.3. Desfer l'últim commit (desfà TOTS els canvis introduïts pel commit, i fa un altre commit amb la versió "restaurada")

```
git revert HEAD
```



10.4. Destruir tot fins una versió anterior




```
git reset --hard <version_id>
```

11. Ignorant arxius: Crear un arxiu `.gitignore` a l'arrel del projecte. Es mostra un exemple a continuació dels seus continguts. Useu una plantilla existent https://www.gitignore.io/

```
# Created by https://www.gitignore.io/api/java
### Java ###
*.class
# BlueJ files
*.ctxt
# Mobile Tools for Java (J2ME)
.mtj.tmp/
# Package Files #
*.jar
*.war
*.ear
# virtual machine crash logs, see
http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*
# End of https://www.gitignore.io/api/java
```

12. Git log, (és el *Git Show History* des de Netbeans):

```
git log
```

12.1. Per veure jerarquía:

```
git log --graph --all
```

13. Definició d'Àlies i millora gràfica del log:

Dins de C:\Program Data\Git\config, editeu

`~/.gitconfig`

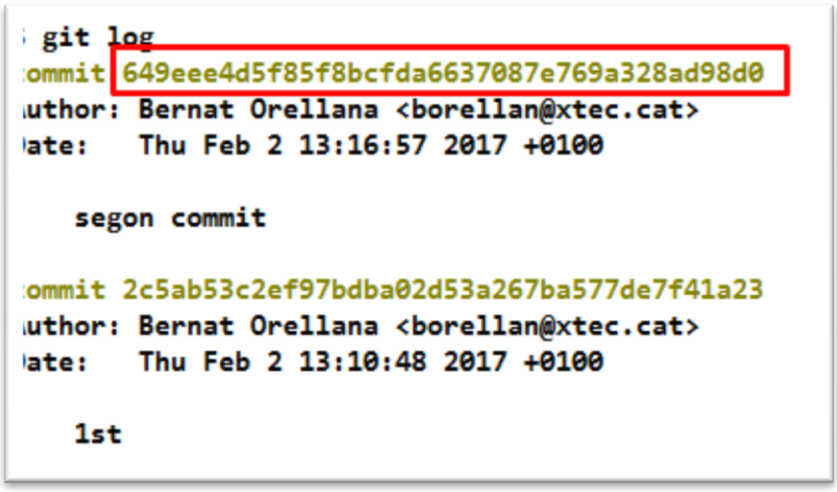
Afegiu les línies següents:

```
[alias]
lg1 = log --graph --abbrev-commit --decorate --date=relative --
format=format:'%C(bold blue)%h%C(reset) - %C(bold green)(%ar)%C(reset)
%C(white)%s%C(reset) %C(dim white)- %an%C(reset)%C(bold red)%d%C(reset)' -
-all
lg2 = log --graph --abbrev-commit --decorate --format=format:'%C(bold
blue)%h%C(reset) - %C(bold cyan)%aD%C(reset) %C(bold
green)(%ar)%C(reset)%C(bold red)%d%C(reset)%n''
```

```
%C(white)%s%C(reset) %C(dim white)- %an%C(reset)' --all
lg = !"git lg1"
```

14. Concepte de ID de commit (hash)

Cada commit s'identifica únicament amb un codi Hash:



15. Git checkout : permet tornar l'estat de tot el projecte complet a una versió indicada.
    15.1. A un commit concret (usant HASH):

```
git checkout {inici del hash del commit}
```

    15.2. A l'última versió de la branca (master) :

```
git checkout master
```

16. El paper de HEAD : HEAD representa la posició actual dins de l'arbre de versions.

HEAD és un apuntador a la versió on actualment estem treballant. Actua com un apuntador a un identificador de commit concret. Quan feu git lg1 veureu la marca a la dreta, sobre el commit on actualment estigueu:

```
$ git lg1
* 9511ca7 - (6 seconds ago) Primers canvis - Bernat Orellana (HEAD, master)
* 4469b4e - (8 minutes ago) Pujada inicial - Bernat Orellana
```

17. El problema dels Detached HEAD: Si feu checkout a una versió anterior usant directament un id de commit, estare en mode "Detached HEAD". Això significa que si feu canvis i els confirmeu estareu creant una "branca detached" de la branca principal.

```
$ git lg1
* 9511ca7 - (6 seconds ago) Primers canvis - Bernat Orellana (HEAD, master)
* 4469b4e - (8 minutes ago) Pujada inicial - Bernat Orellana

$ git checkout  4469b4e
Note: checking out '4469b4e'.

You are in 'detached HEAD' state.

$ git lg1
* bc5b4cd - (11 seconds ago) temerari detached head - Bernat Orellana (HEAD)
| * 9511ca7 - (5 minutes ago) Primers canvis - Bernat Orellana (master)
|/
* 4469b4e - (12 minutes ago) Pujada inicial - Bernat Orellana
```

Si tornem a la branca master (**`git checkout master`**), les braques detached **s'oculten**. Les podem fer visibles usant la comanda **`git reflog:`**

```
$ git reflog
9511ca7 HEAD@{0}: checkout: moving from bc5b4cdc7de2bae13b963c7e9c6e1ae0d96b14f9 to master
bc5b4cd HEAD@{1}: commit: temerari detached head
4469b4e HEAD@{2}: checkout: moving from master to 4469b4e
9511ca7 HEAD@{3}: commit: Primers canvis
4469b4e HEAD@{4}: commit (initial): Pujada inicial
```

També podeu usar

```
git log --graph --decorate $(git rev-list -g --all)
```

18. Marques de versió (TAGS!)

18.1. Llistar tags:

```
git tag
```

18.2. Crear un tag:

```
 git tag {tagname}
```

18.3. Crear un tag anotat (per versions importants): (

```
git tag –a {tagname} –m {missatge de versió )
```

18.4. Compartir un tag a un repo remot:

```
git push origin {nom_del_tag}
```

18.5. Veure dades d'un tag:

```
git show {tagname}
```

18.6. Moure's a la versió:

```
git checkout tag
```

IMPORTANT: després d'aquesta comanda estem en Detached HEAD: no podem fer commits si hem fet un checkout a un tag o a un commit concret !!

19. Stash

19.1. Guardar a l'stash

```
git stash
```

19.2. Recuperar l'últim l'stash

```
20. git stash apply
```

20.1. Llistar stash

```
git stash list
```

```
$ git stash list
stash@{0}: WIP on master: b4b9b9e Aquest commit és una putada enorme
stash@{1}: WIP on master: b4b9b9e Aquest commit és una putada enorme
stash@{2}: WIP on master: b4b9b9e Aquest commit és una putada enorme
stash@{3}: WIP on master: b4b9b9e Aquest commit és una putada enorme
```

20.2. Recuperar l'stash per identificador (en vermell)

```
git stash apply stash@{0}
```

21. Branques

21.1. Concepte de branca i organització interna de GIT

21.2. Creació de branques:

```
git branch {branchname}
```

21.3. Moure's a una branca :

```
git checkout {branchname}
```

21.4. Merge de branques

```
git checkout {branca_destí_del_merge}
git merge {branca_origen_del_merge}
```

21.5. Conflictes de merge i la seva resolució.

Treballant amb REPOSITORI REMOT

1. Llista de repositoris remots:

```
git remote
```

2. Clonar un repositori remot (per defecte s'anomenarà *origin*):

```
git clone {url}  {opcional:nom del directori}

# si no indiquem carpeta, crea una carpeta amb el nom de l'arxiu *.git. Per
# exemple, la línia següent crea una carpeta GitTesting
git clone https://github.com/infomila/GitTesting.git

#   la línia següent crea una carpeta GitTestingPaco
git clone https://github.com/infomila/GitTesting.git  GitTestingPaco
```

Si volem canviar el nom del repositori remot podem indicar-ho amb "-o"

```
git clone –o otherorigin https://github.com/infomila/GitTesting.git
```

3. Pujar els canvis locals al repositori remot:

```
git push {repo remot}
```

```
git push
```

4. Sincronitzar el nostre repositori amb el repositori remot (sense barrejar canvis ¡)

```
git fetch {repo remot}
git fetch origin
```

git fetch descarrega els canvis del repositori remot que no teníem al local, però no barreja la nostra versió actual amb la última versió del servidor. Per tant, mai dona conflictes.

Si volem a part d'actualitzar les dades de versions, barrejar la nostra versió amb la del servidor cal usar addicional la comanda *merge*:

```
git merge origin/master
```

Fer un fetch seguit d'un merge és tant freqüent, que s'ha condensat en una única comanda **git pull**:

```
git fetch origin
git merge origin/master
#   la línia següent equival a les dues anteriors
git pull origin
```

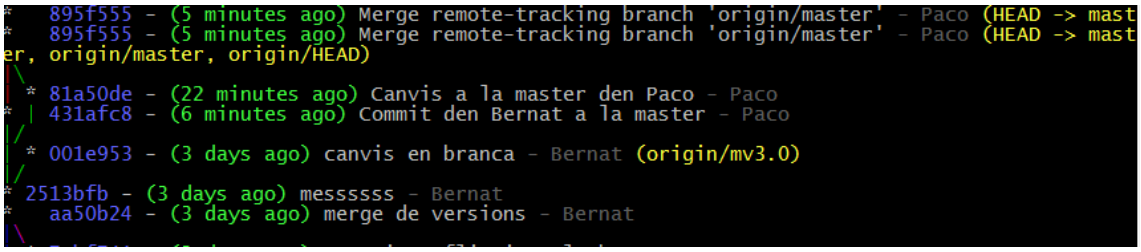
5. Les branques són locals per defecte, i per tant no són compartides als repositoris remots. Si volem fer-ho , cal fer un push explicit:

```
git push {origin} {nom_branca}
```

6. Llista de branques remotes

```
git branch -r
```

7. Seguir una branca remota:




```
git checkout -b refactoredLocal origin/refactored
```

```
$ git checkout --track -b mv3.0 origin/mv3.0
Branch mv3.0 set up to track remote branch mv3.0 from origin.
Switched to a new branch 'mv3.0'
```