Control de versions

Introducció als sistemes de gestió de canvis

Josep Maria Pinyol Fontseca

27/06/2018

Endepro Software, S.L.

Continguts

Introducció

Presentació

Qui sóc?

Em dic **Josep Maria Pinyol Fontseca** i sóc un apassionat de la programació. De formació Enginyer Industrial en Electrònica, vaig començar a programar microprocesadors, però la curiositat d'entendre què hi havia darrera d'aquella programació, ara fa més de 20 anys, em va encaminar al desenvolupament de software. Des de fa 14 anys dirigeixo els projectes que desenvolupa el meu equip a Endepro Software

Introducció

Qué és un sistema de control de versions?

- Gestió de canvis d'arxius en el temps
- Revertir un arxiu a una versió anterior de forma simple
- Comparar canvis en el temps
- Visualitzar qui ha realitzat determinats canvis en un fitxer
- S'utilitza principalment a la industria informàtica

Sistemes de control de versions

Sistema de control habitural

Un dels primers sistemes de control de versions els coneixem perfectament.

Segurament molts de nosaltres hem fet això més d'una vegada:

- Oferta.docx
- Oferta-Copia.docx
- Oferta-Copia (2).docx
- Oferta-BO.docx
- Oferta-BO-2.docx

I quina és l'oferta bona?

Sistema de control habitural

Si sóm molt organitzats inclourem la data i hora en el nom del fitxer:

- Oferta-18-6-2017-17:20.docx
- Oferta-18-6-2017-17:25.docx
- Oferta-19-6-2017-8:42.docx
- Oferta-19-6-2017-9:15.docx

El mateix pasa amb projectes sencers que estan en una carpeta, fent versions de les carpetes, etc.

Per què necessitem un sistema de control de versions?

Però, per què realment necessitem un sistema de control de versions:

- Còpia i restauració: podem saltar a una versió del dia que vulguem
- Sincronització: podem compartir els fitxers i tenir la última versió
- Desfer a curt termini: podem tirar enrera dels canvis que hem anat fent durant els últims dies
- Desfer a llarg termini: si necessitem una versió de fa 1 any, cap problema
- Control de canvis: controlar tots els canvis que es fan en els fitxers
- Control dels usuaris: tenim el nom i cognoms de qui ha fet el canvi
- Branques i unions: si volem fer molts canvis en un projecte, podem aïllar el nostre codi, fer els canvis i unir-ho tot un cop estigui fet

Tipus de Sistemes de control de versions

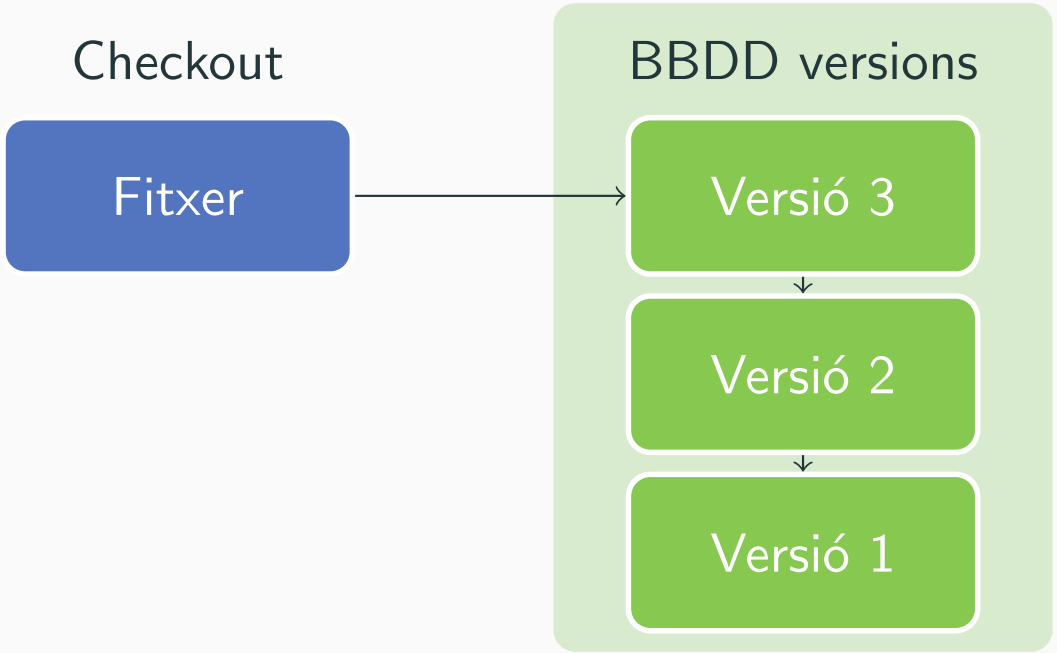
Evolució i generacions dels sistemes de control de versions:

1. Sistemes de control de versions locals
2. Sistemes de control de versions centralitzats
3. Sistemes de control de versions distribuïts

Sistemes de control de versions locals

Primera generació: local

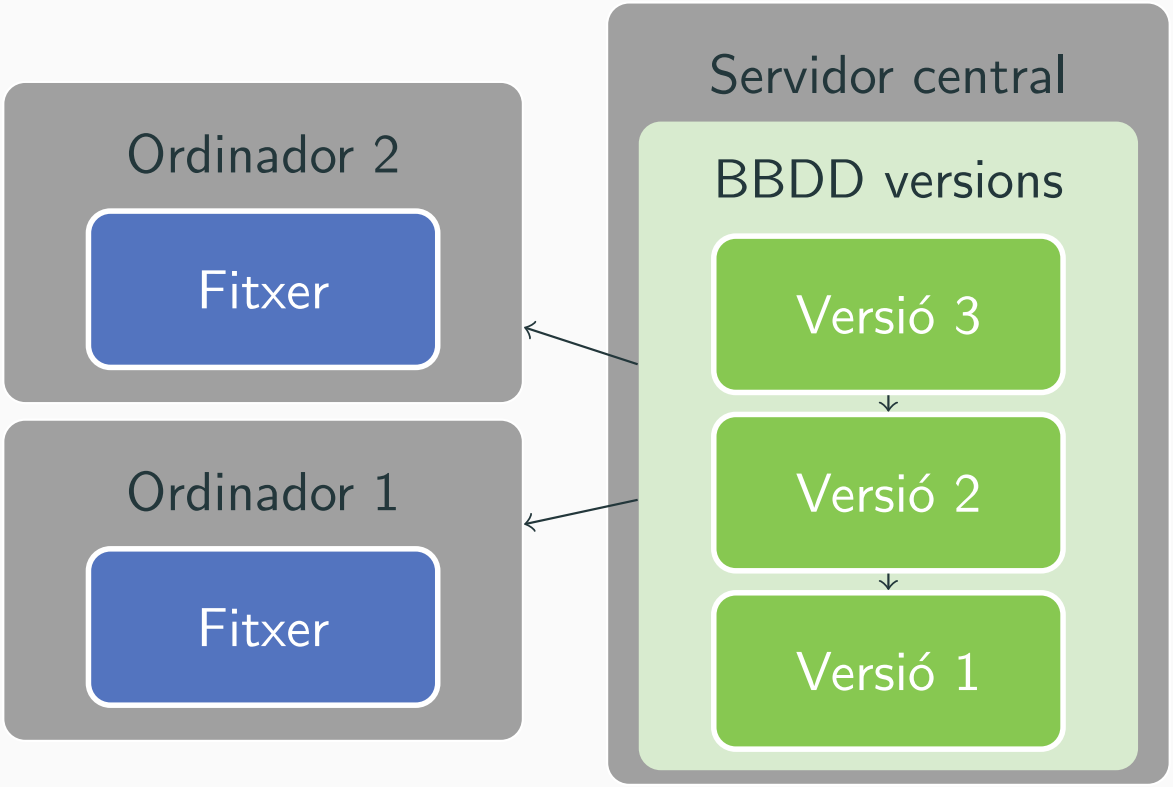
- Disponible per a sistemes UNIX des del 1972
- Disenyat per controlar els canvis realitats en el codi font o fitxers de text
- Tenim disponible el RCS per vàries plataformes



Sistemes de control de versions centralitzat

Segona generació: centralitzat

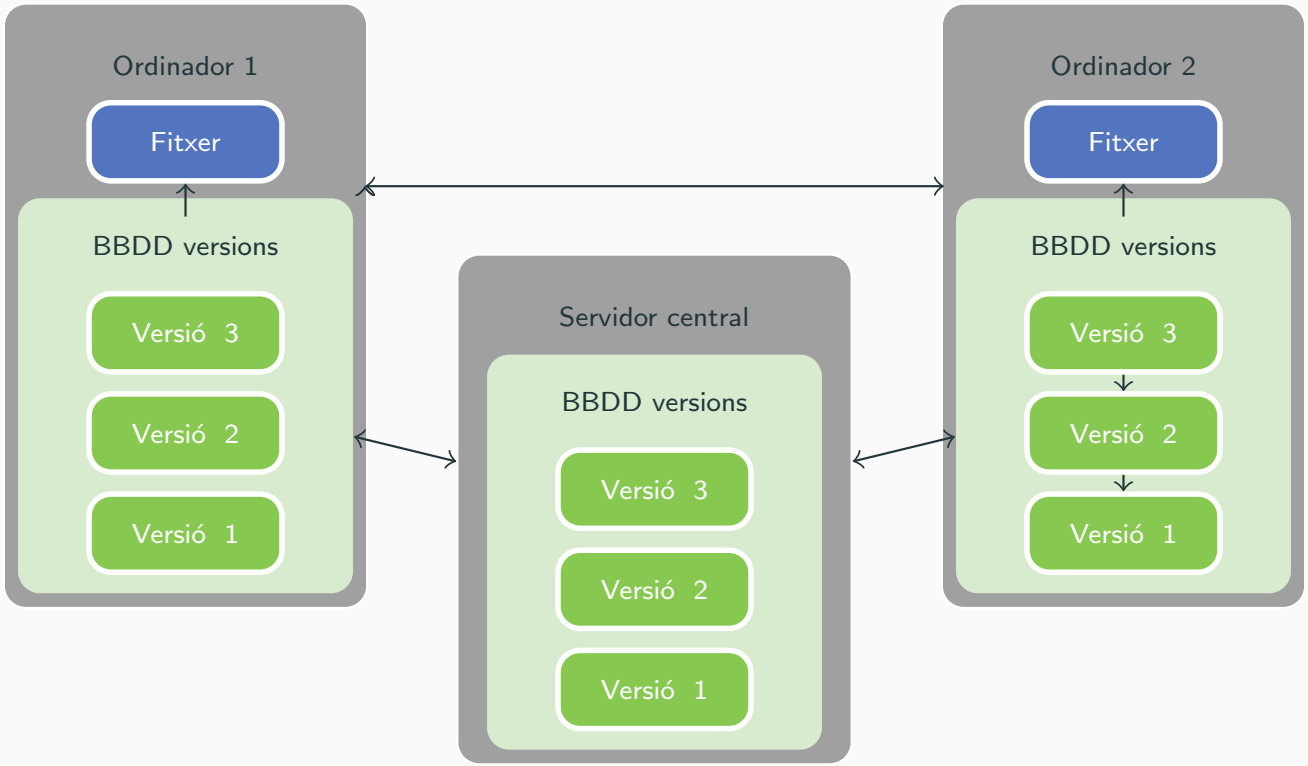
- Disponible des del 1986
- Arquitectura client servidor
- Les implementacions més utilitzades són: CVS, Subversion (SVN)



Sistemes de control distribuït

Tercera generació: distribuït

- Disponible des del 2000
- Arquitectura desentralitzada o distribuïda
- Les implementacions més utilitzades són: GIT, Mercurial (HG)



Argot

Argot general

- Repositori (repo): base de dades amb els fitxers
- Servidor: ordinador on hi ha el repositori
- Client: ordinador que es connecta al repositori
- Trunk/Main/Master: primera localització del codi en el repositori

Accions en sistemes centralitzats

- Checkout: descarregar fitxers o fitxer del repositori
- Add: afegir un fitxer/fitxers en el repositori per primer cop
- Check in: carregar fitxers al repositori
- ChangeLog/History: llista de canvis fets a un fitxer
- Update/Sync: sincronitza els fitxers locals a la última versió del repositori
- Revert: descarta els canvis locals del fitxer i agafa la última versió del repositori

Argot

Accions en sistemes distribuïts

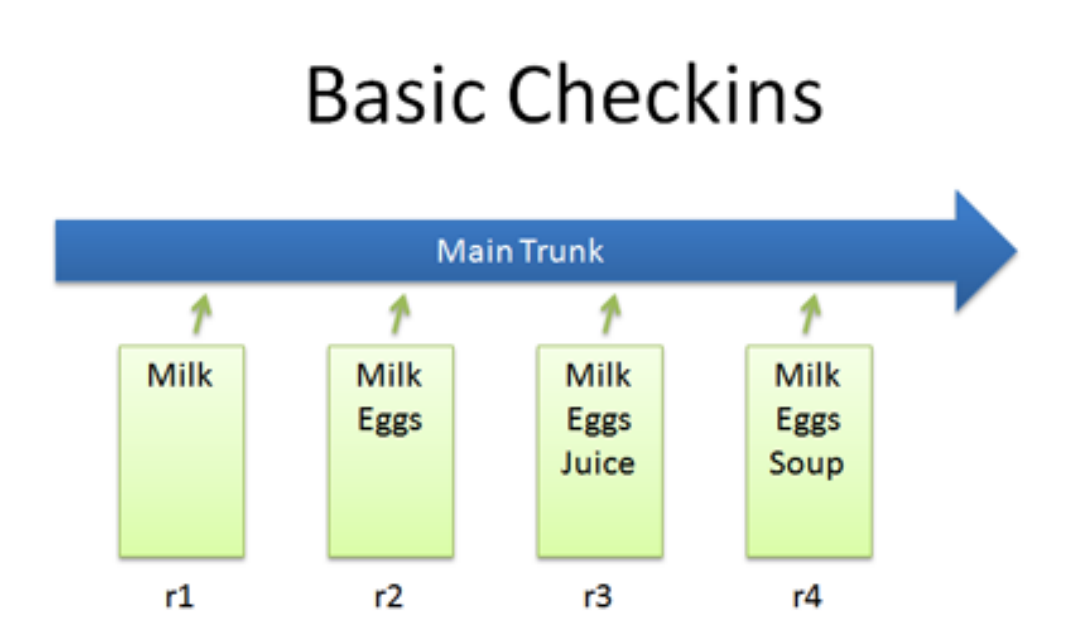
- Pull: obtenir canvis d'un altre repositori
- Push: enviar canvis a un altre repositori
- Commit: enviar canvis al repositori local

Accions avançades

- Branch: crea una copia separada per per test, correccions, etc.
- Diff: diferencia entre dos fitxers
- Merge (Path): aplica canvis d'un fitxer a un altre per tenir la última versió
- Conflict: al tenir canvis pendents que contradiuen un altre canvi (els dos canvis no es poden aplicar)
- Resolve: resolució dels conflictes

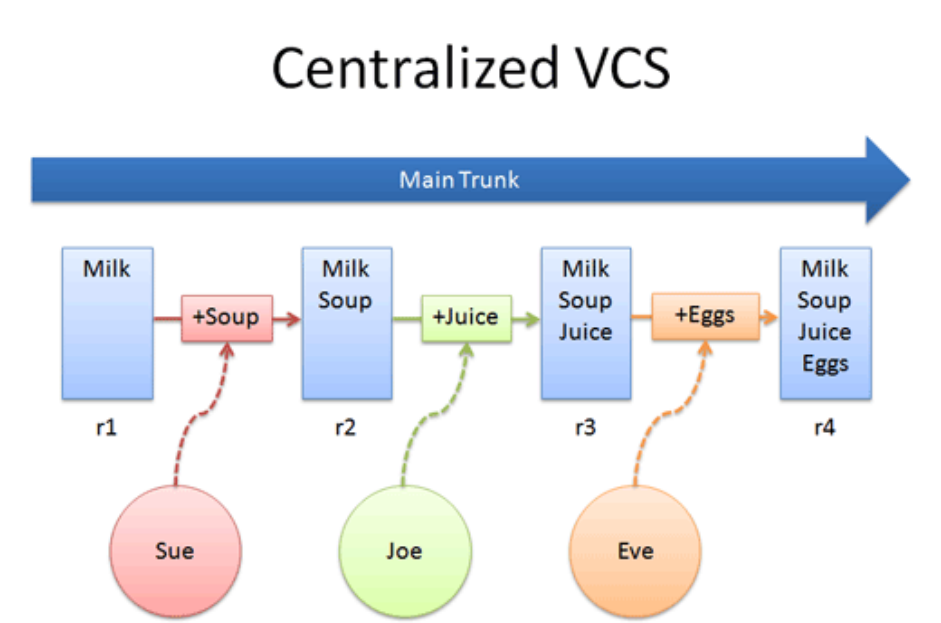
Revisions

- Cada cop que fem una nova versió tenim una nova revisió del fitxer



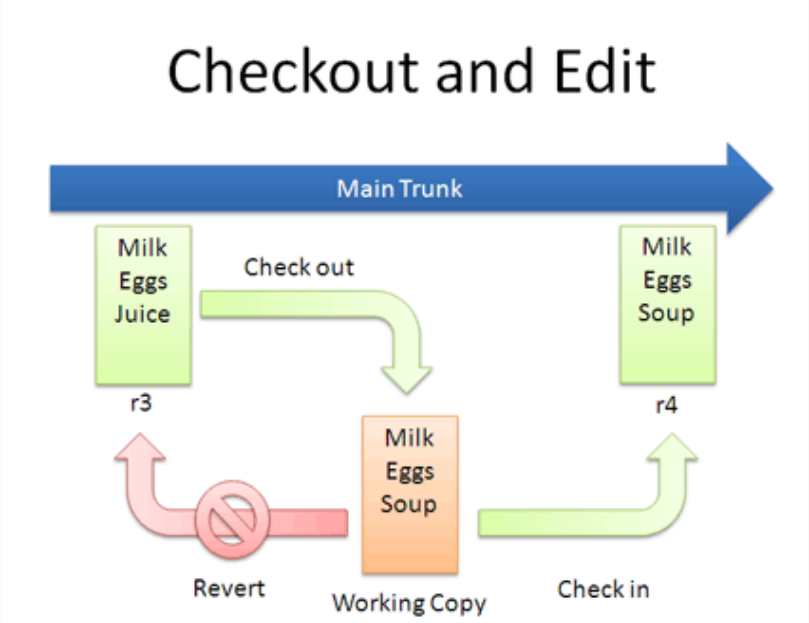
Revisions en un sistema centralitzat

- Els diferents desenvolupadors envien els canvis a la branca principal
- Teòricament s'hauria de fer una branca perque els altres ho provin



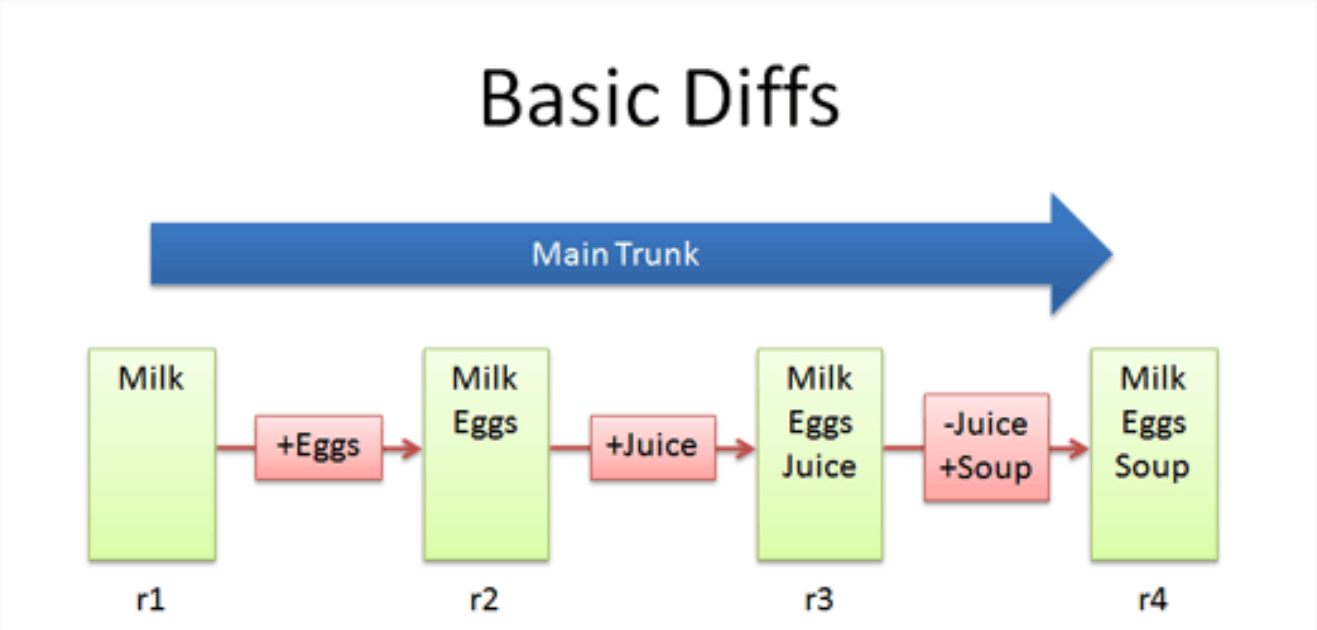
Canvis i edició

- En realitat obtenir la revisió, editem i guardem la revisió
- Podem desfer els canvis a una revisió anterior sempre que vulguem



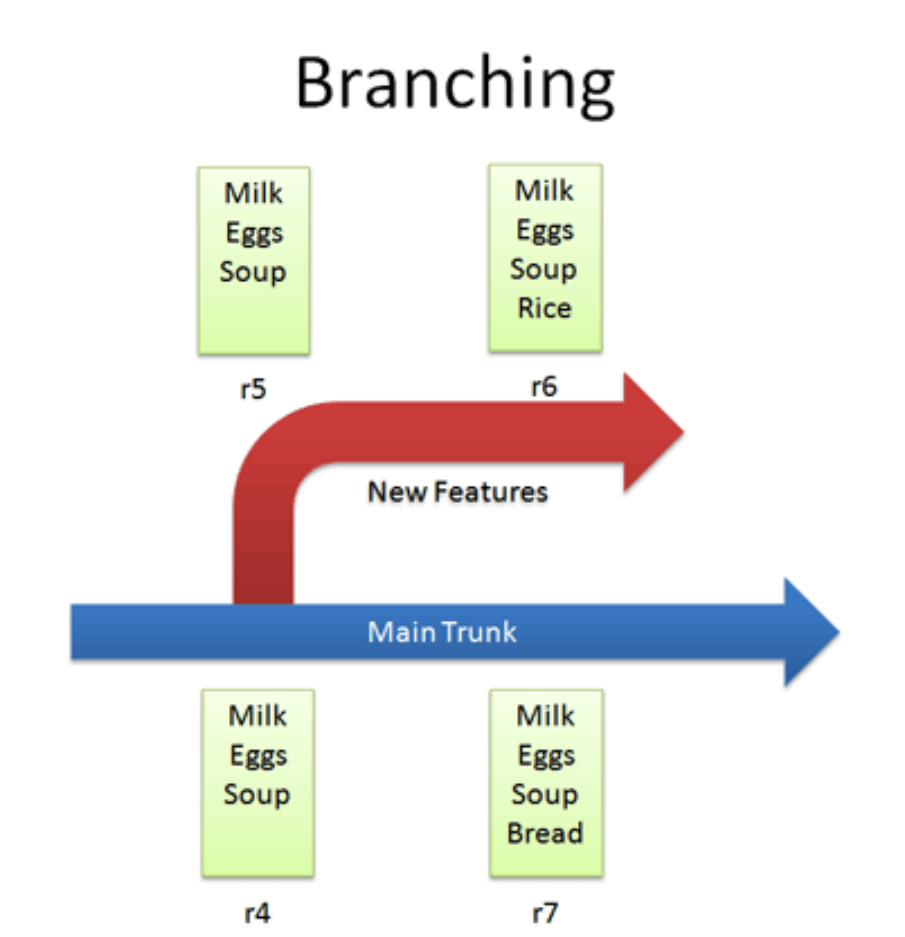
Diferències

- Es guarden les diferències entres revisions
- La majoria de sistemes de control de versions guarden aquests diferencies en comptes de tots els fitxers



Branques

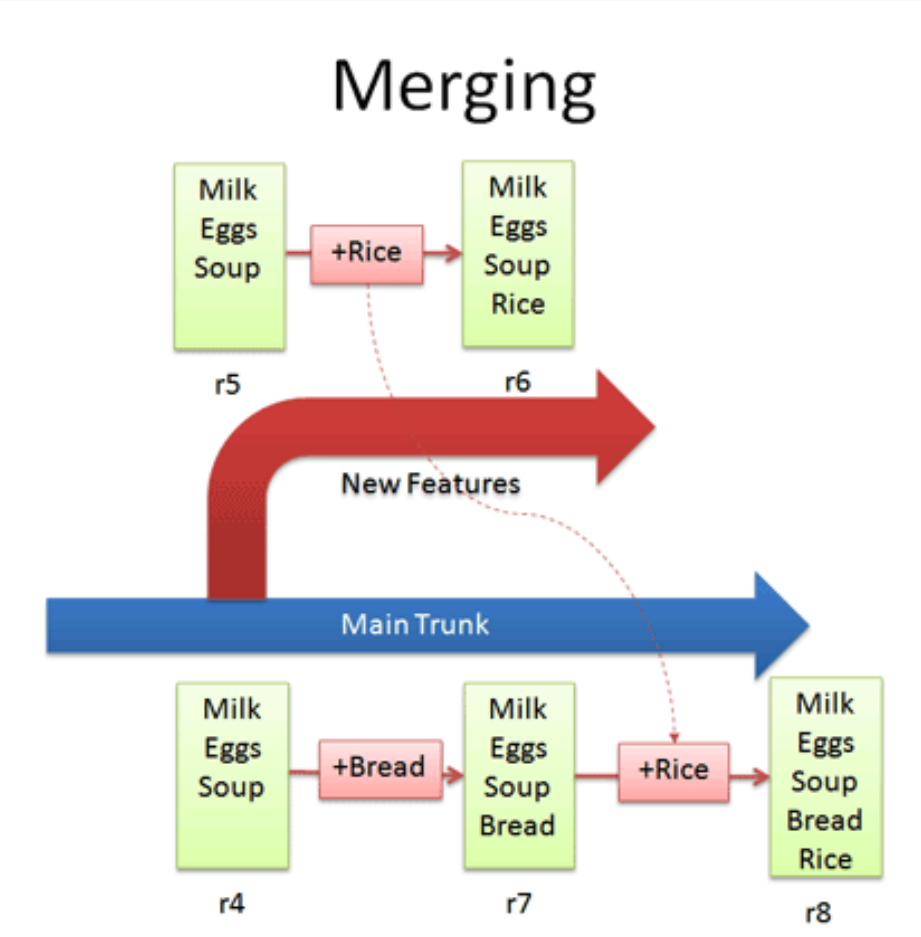
- Les branques ens permeten fer una còpia del codi en una "carpeta" diferent



Unions o Merge

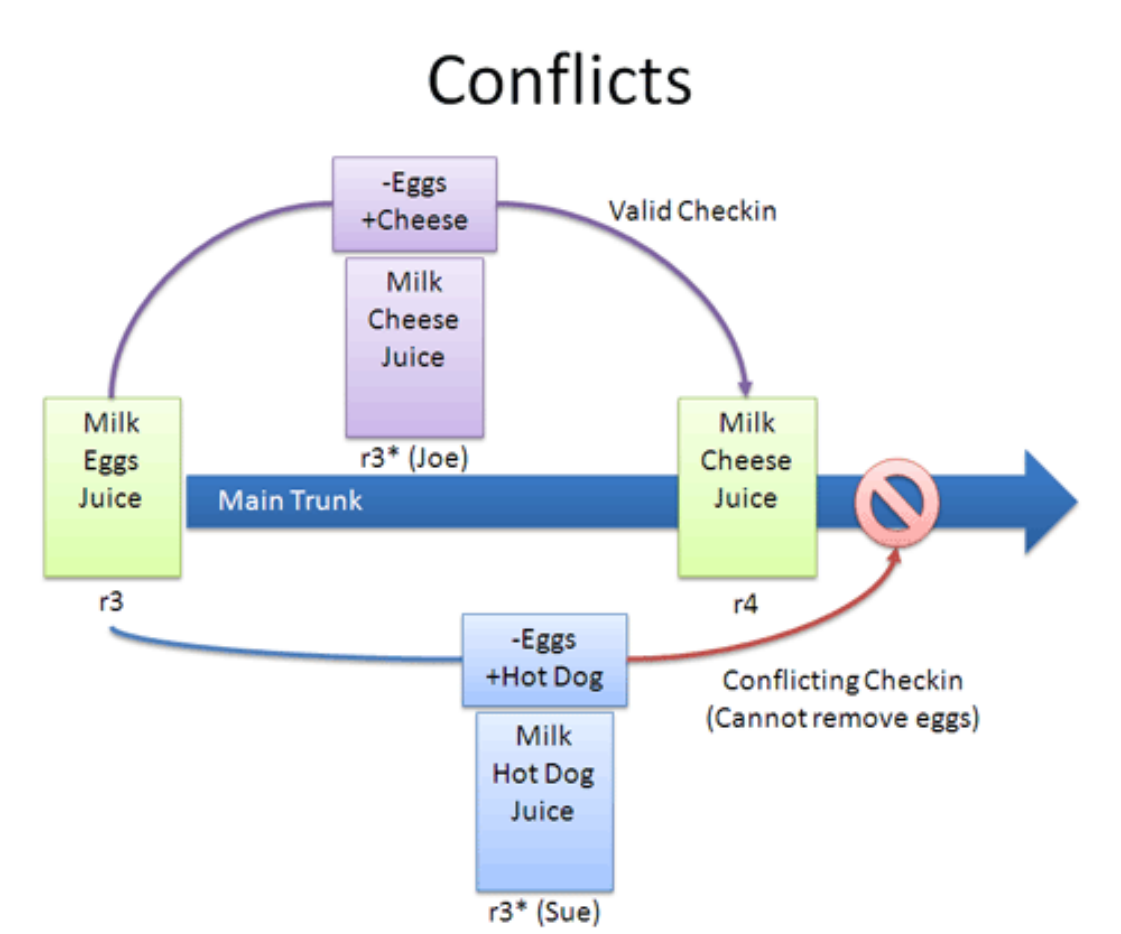
Merge

- En permet unir canvis realitzats de branques diferents



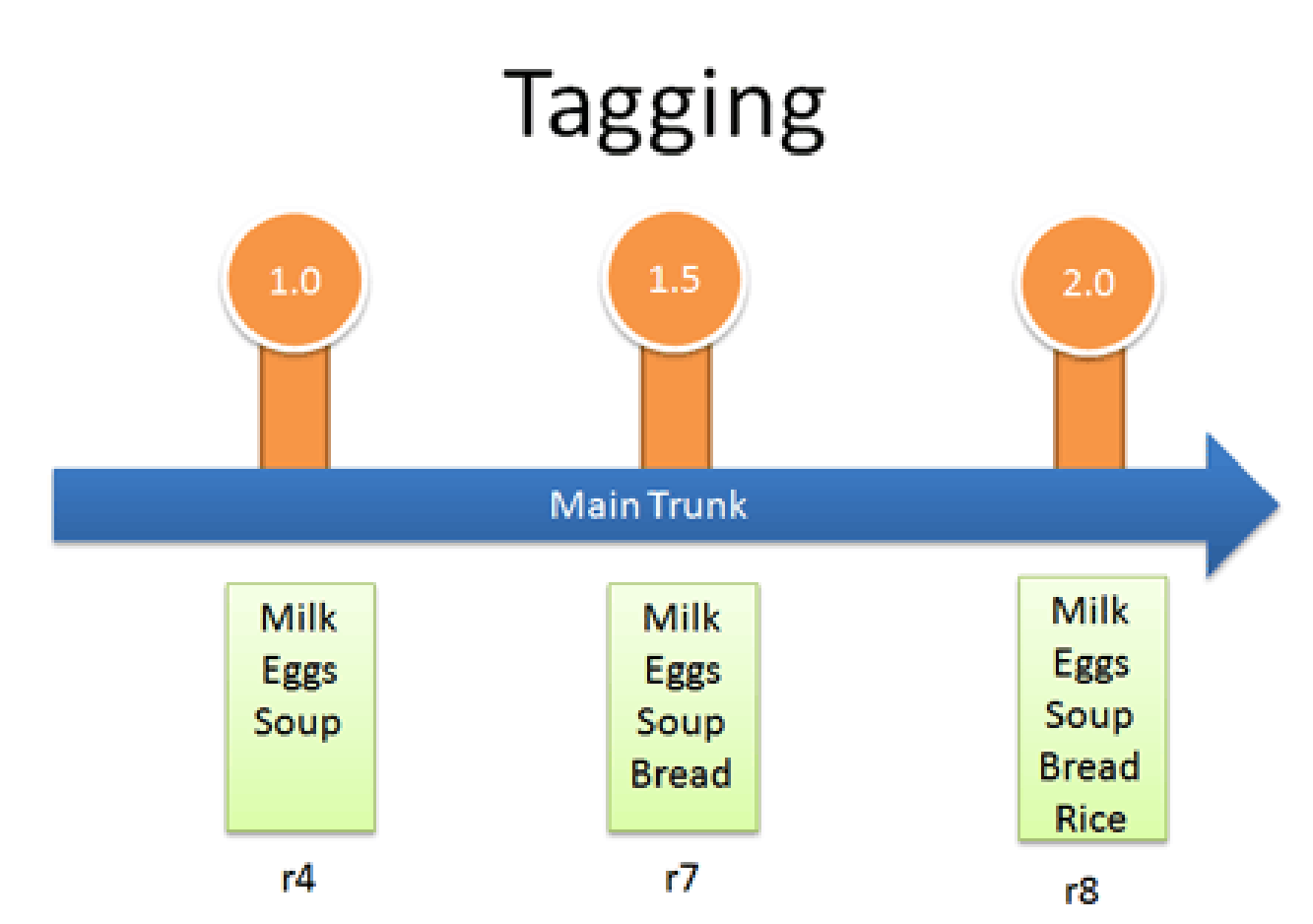
Conflictes

- A vegades les unions o merges ens donen conflictes en el codi



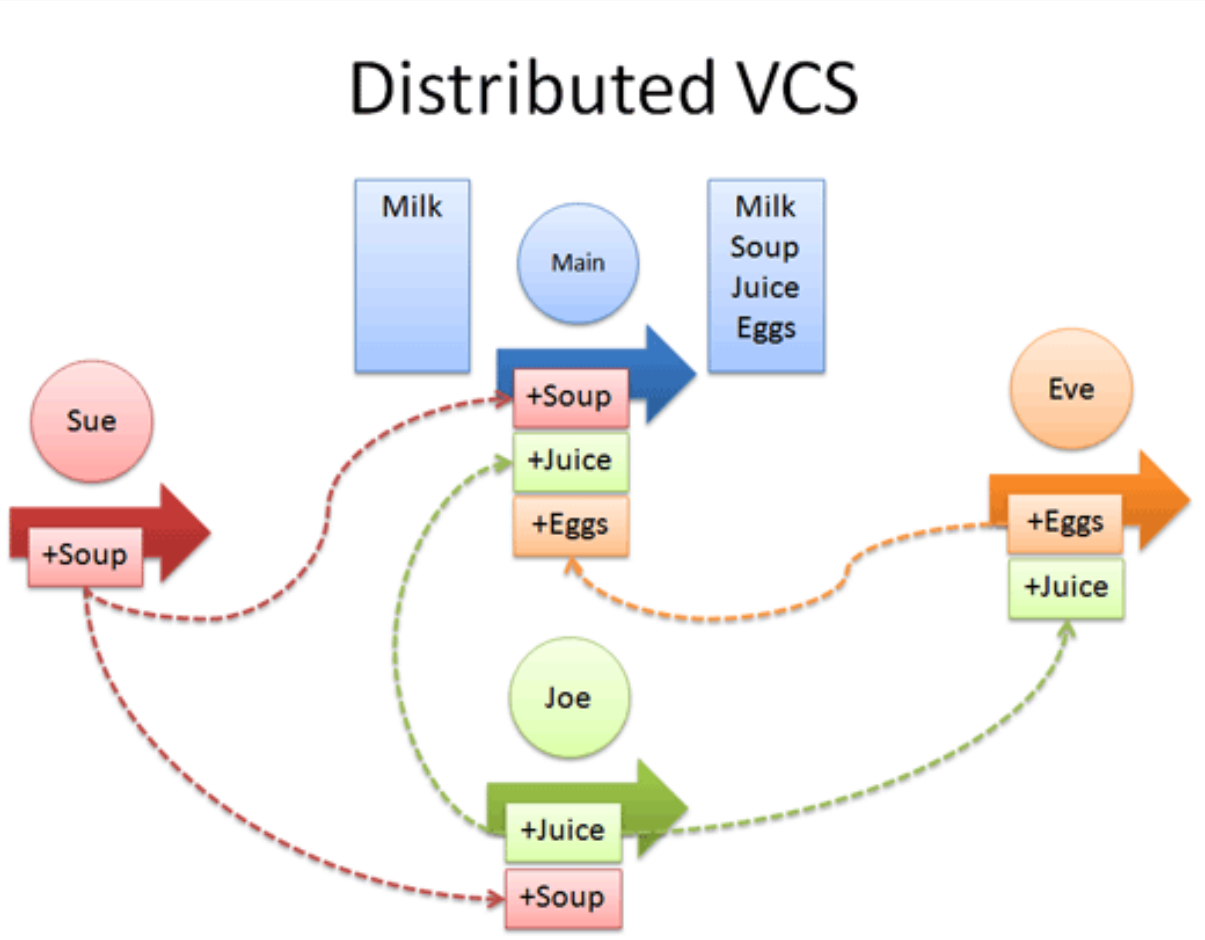
Etiquetes o Tags

- Podem marcar la revisió per tenir una etiqueta identificativa de la revisió



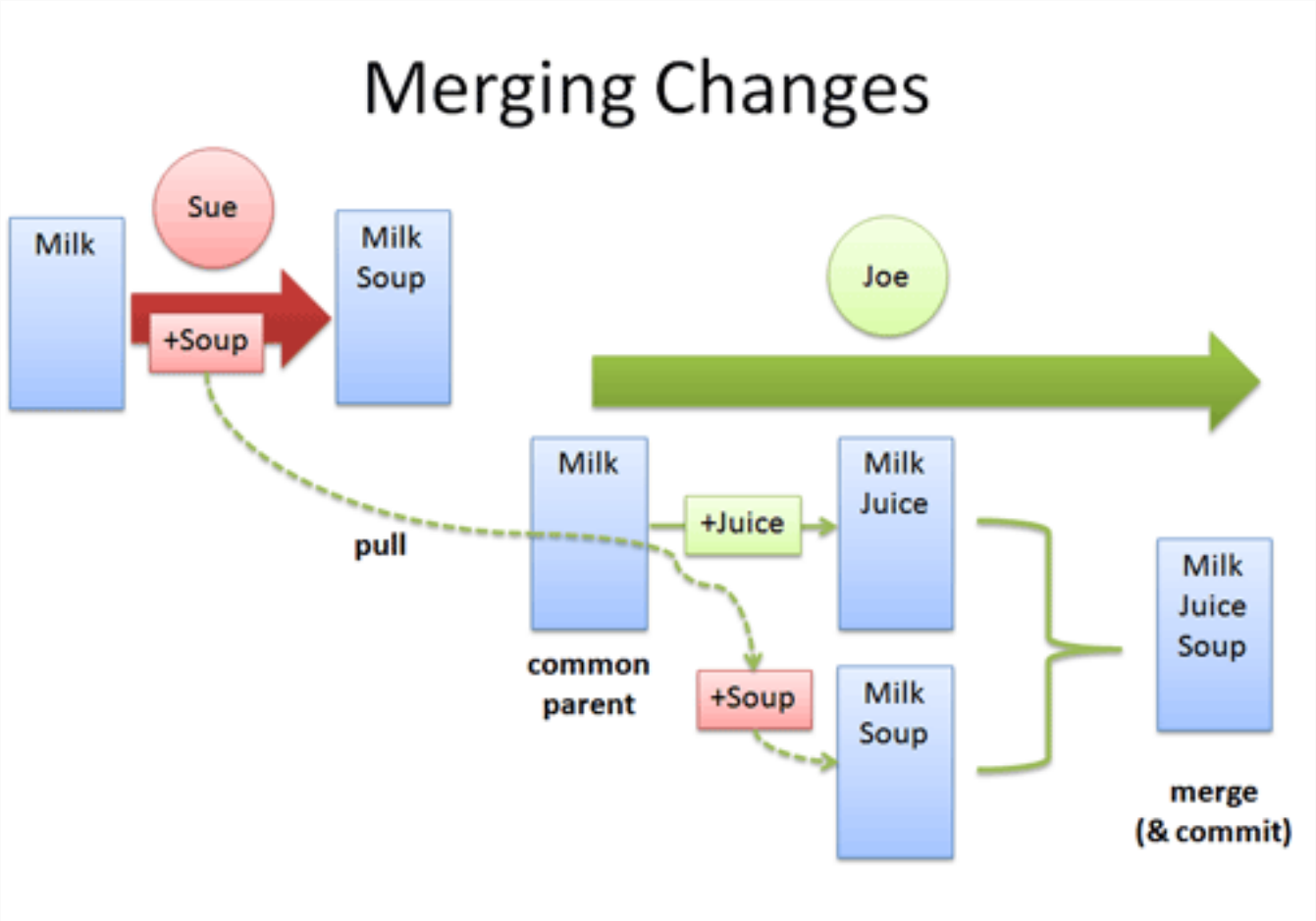
En sistemes distribuïts cada desenvolupador té el seu propi repo

- En sistemes distribuïts es fan unions entre les branques dels desenvolupadors



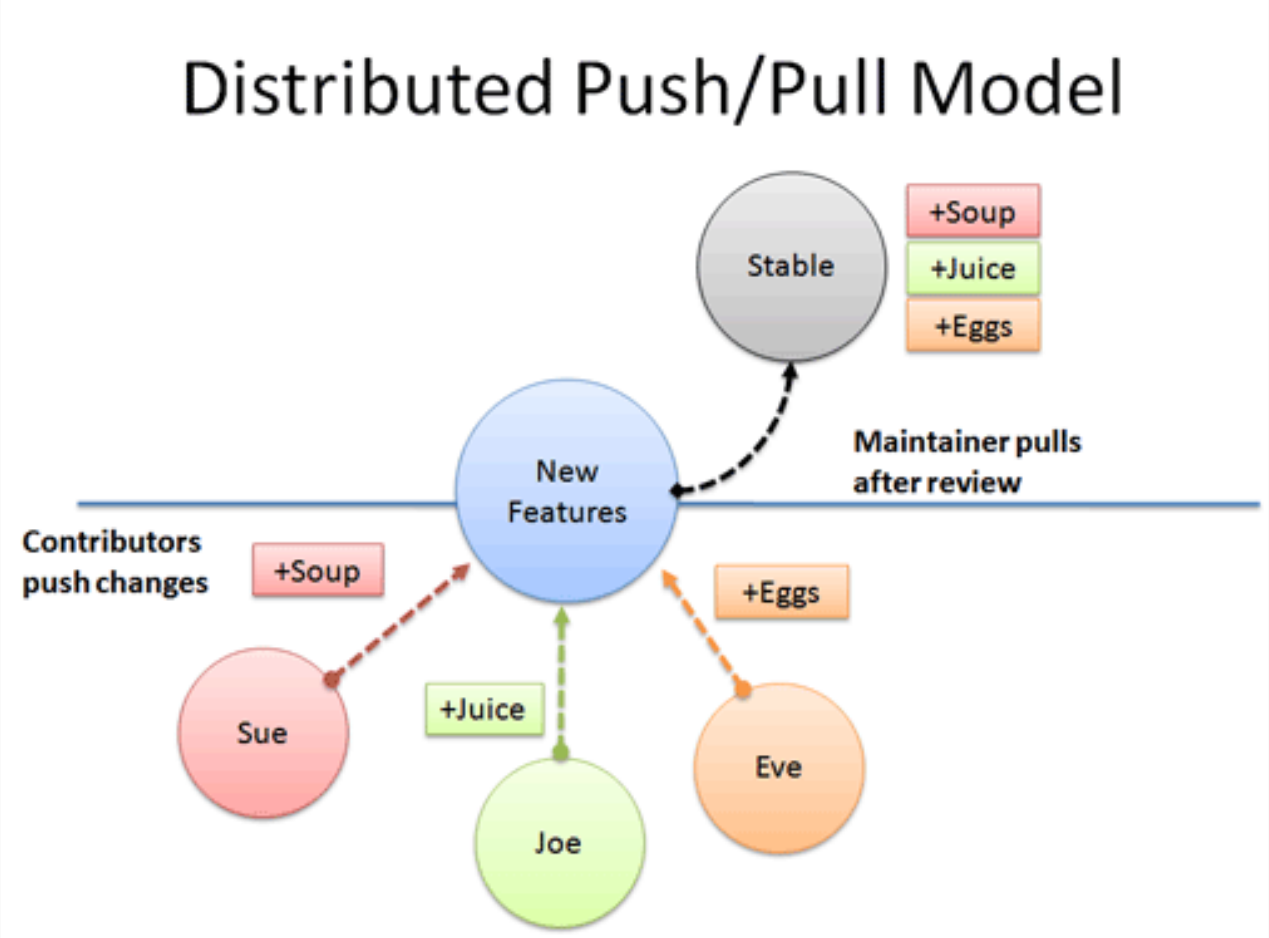
Merge en sistemes distribuïts

- En sistemes distribuïts es fan unions entre les branques dels desenvolupadors



Organització d'un projecte en sistemes distribuïts

- Els programadors comproven els canvis en una branca comuna
- El responsable revisa els canvis i ho envia a la branca stable



Avantatges del sistema Distribuït vs Centralitzat

Principals avantatges del sistema distribuït:

- Tothom té una còpia local del codi
- Es pot treballar Offline
- És ràpid, tot es fa localment i no en el servidor central
- Els canvis es gestionen millor al tenir un GUID per canvi realitzat
- Les Branques i les Unions són fàcils ja que cada desenvolupador té la seva pròpia Branca i es poden realitzar proves fàcilment
- Menys gestió, ja que no cal un servidor central en funcionament

Pràctiques

GIT - Sistema de control de versions distribuït

Utilitzarem GIT com a sistema de control de versions

- La majoria d'entorns de desenvolupament i editors ja tenen integrat el GIT
- Es pot treballar amb linia de comandes o amb programes especifics per GIT
- Per començar, descarregar i instal·lar https://git-scm.com/downloads

Inicialització d'un repositori

1. `git init`
2. Si ens hi fixem ens ha creat una carpeta **.git** en el directori de treball
3. Ja tenim el repositori apunt

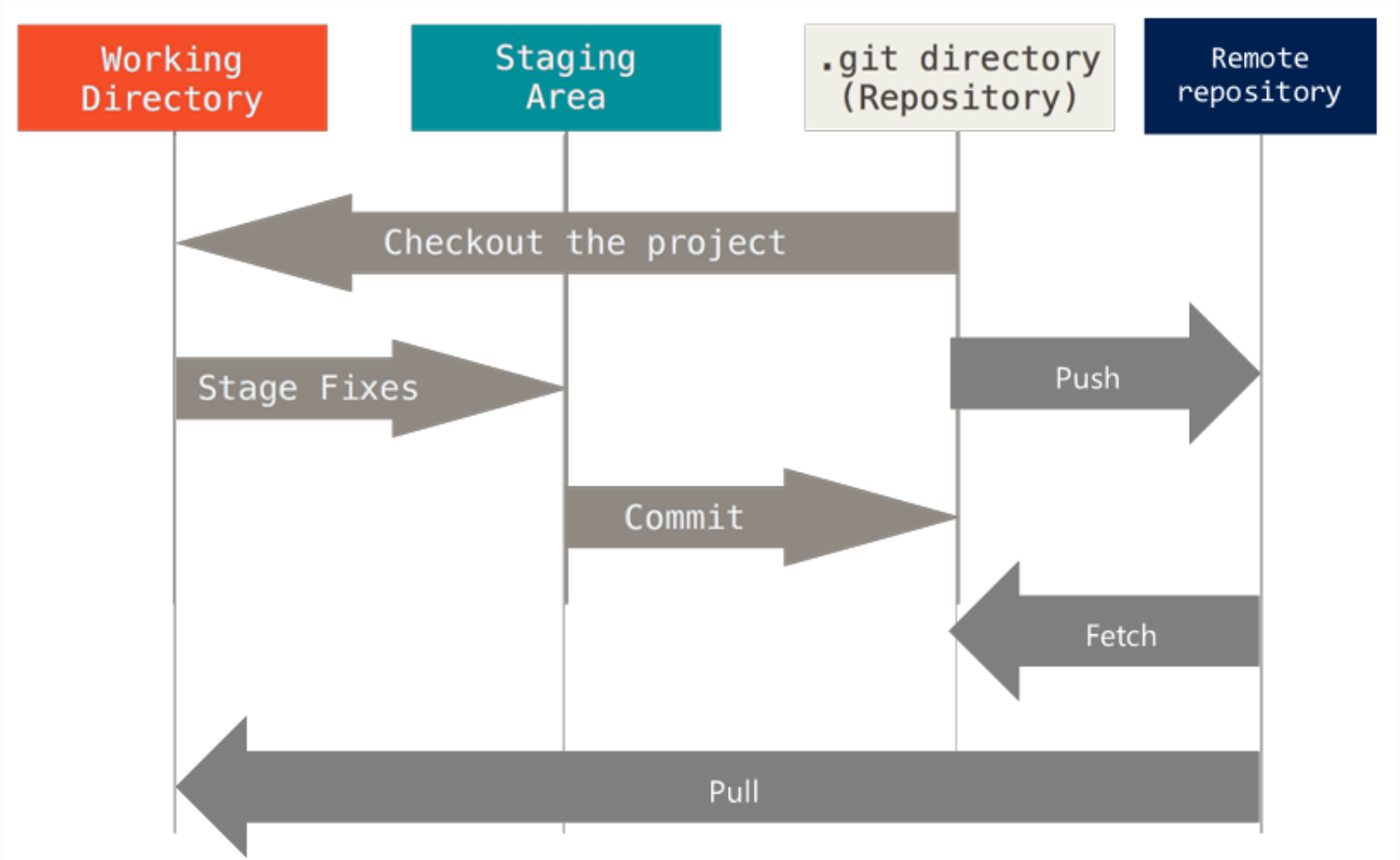
Inicialització d'un repositori - Configuració

1. Anem a configurar les dades de l'usuari que treballa amb el control de versions
2. Nom: `git config --global user.nom "Josep Maria Pinyol Fontseca"`
3. Correu: `git config --global user.email "jmpinyol@endepro.com"`

Afegim fitxers al repositori creat i ho enviem al repositori

1. `git add fitxer.txt`
2. `git commit -m "Versió inicial del fitxer"`

Fluxe de treball general



Modificant i enviant els fitxers modificats

1. Modifiquem el contingut del fitxer.txt
2. `git status` ens dona informació de l'estat actual
3. `git add fitxer.txt`
4. `git commit -m "Canvis que he fet..."`

Modificant i recuperar la versió anterior

1. Modifiquem el contingut del fitxer.txt de nou
2. `git checkout fitxer.txt`
3. Tornem a tenir la versió anterior del fitxer
4. Podem revertir un commit sencer amb un revert `git revert master`

Visualització d'estat

1. Tenim diferents instruccions per tenir informació del repositori, directori de treball, etc.
2. Per saber l'estat del directori actual: `git status`
3. Si volem un registre dels commits realitzats `git log`
4. Amb `git blame fitxer` ens dona informació d'un fitxer en concret

Modificant, stage i unstage d'un fitxer en concret

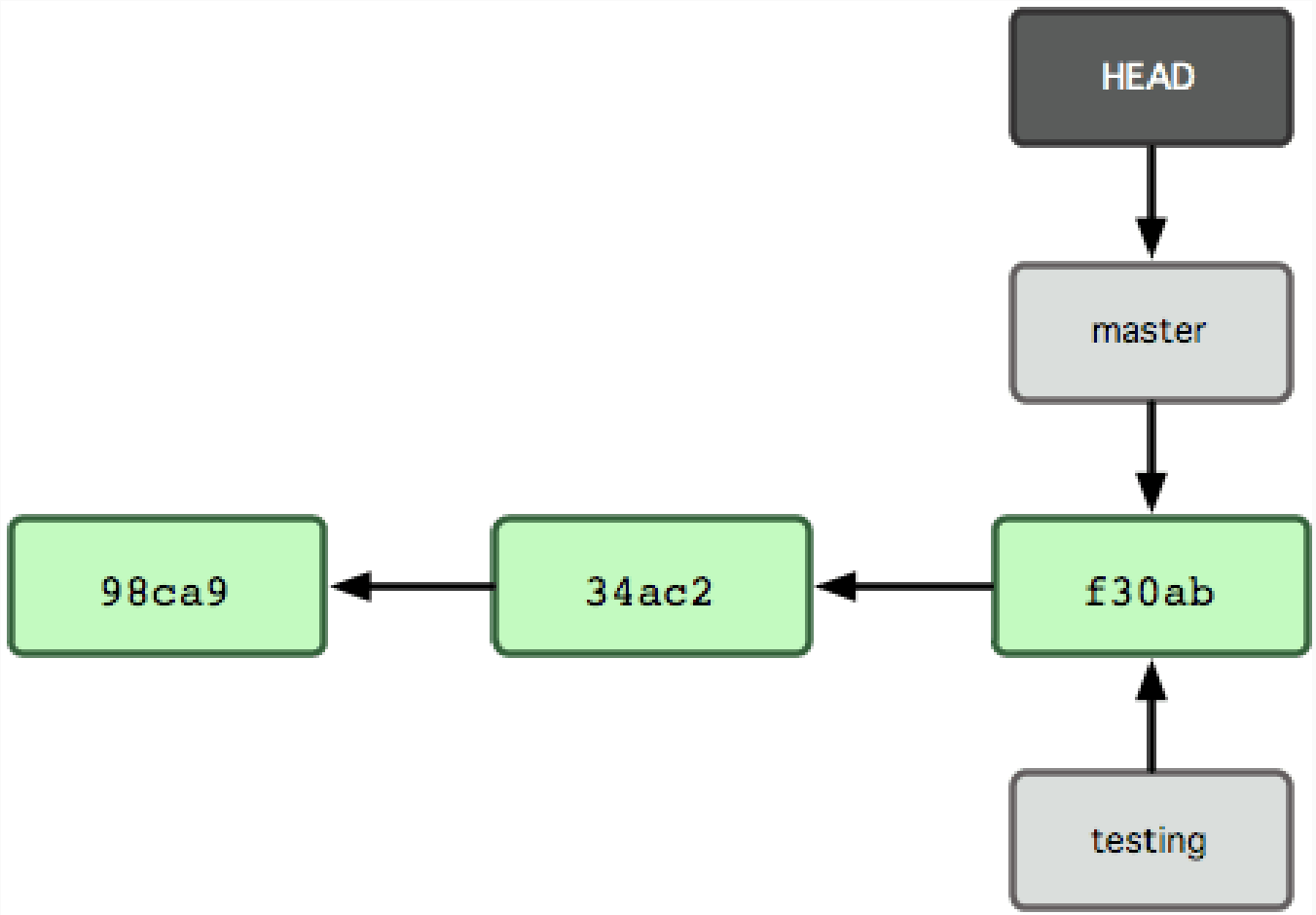
1. Modifiquem el contingut del fitxer.txt de nou
2. `git commit fitxer.txt`
3. `git status`
4. `git reset HEAD fitxer.txt`
5. El fitxer l'hem afegit per enviar al repositori però al final l'hem tret

Treballant amb etiquetes

1. `git tag -a 1.0 -m "Versió 1.0"`
2. `git tag`

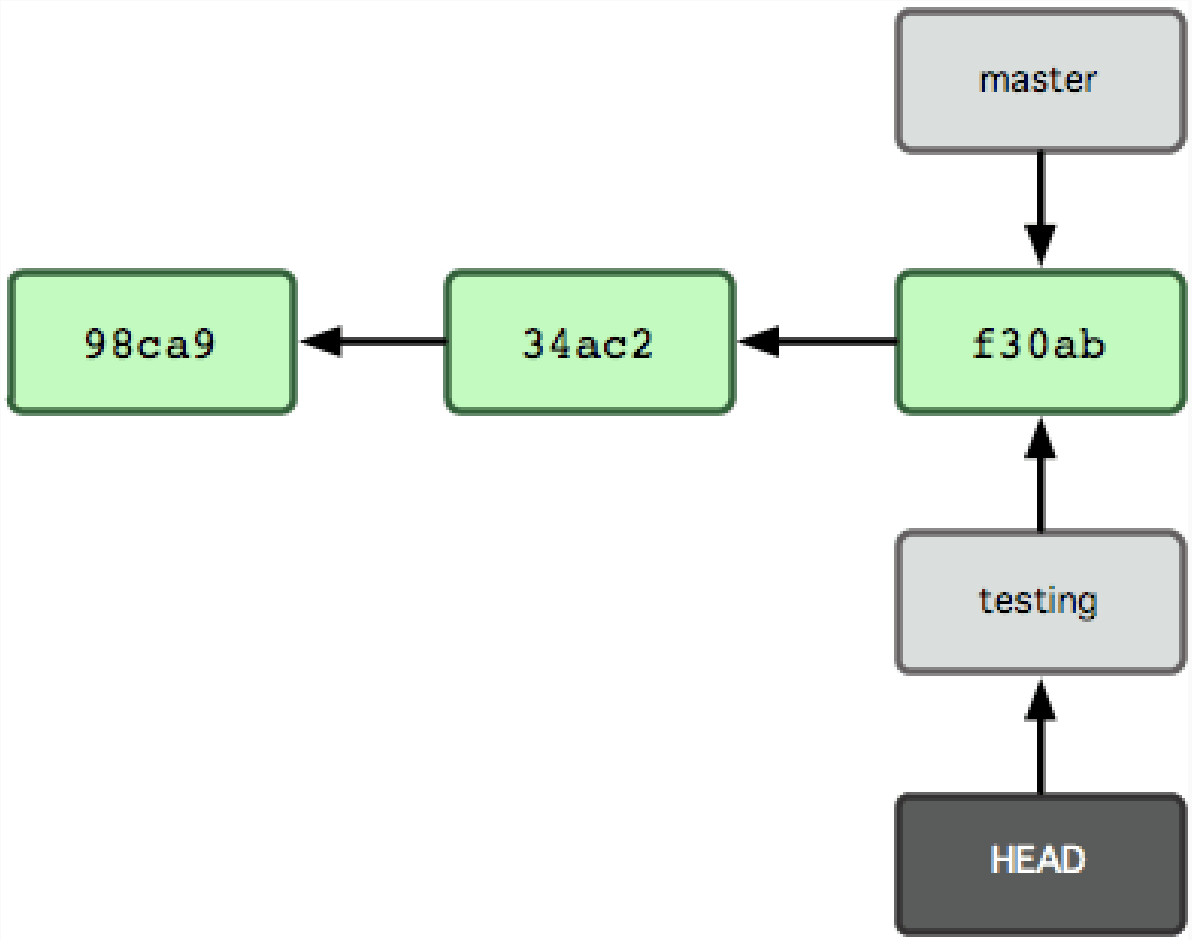
Treballant amb branques

1. `git branch testing`



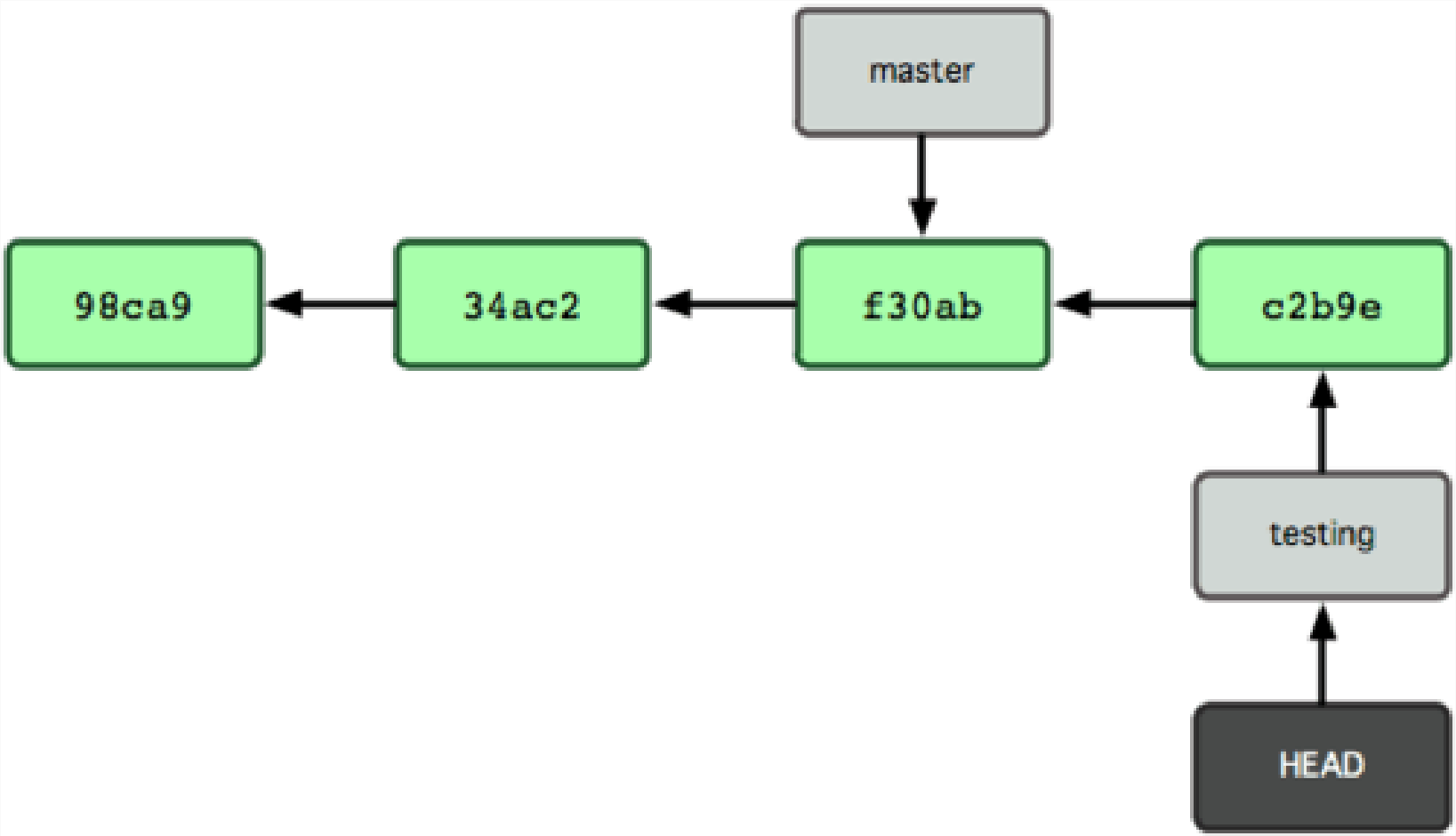
Treballant amb branques

1. `git checkout testing`



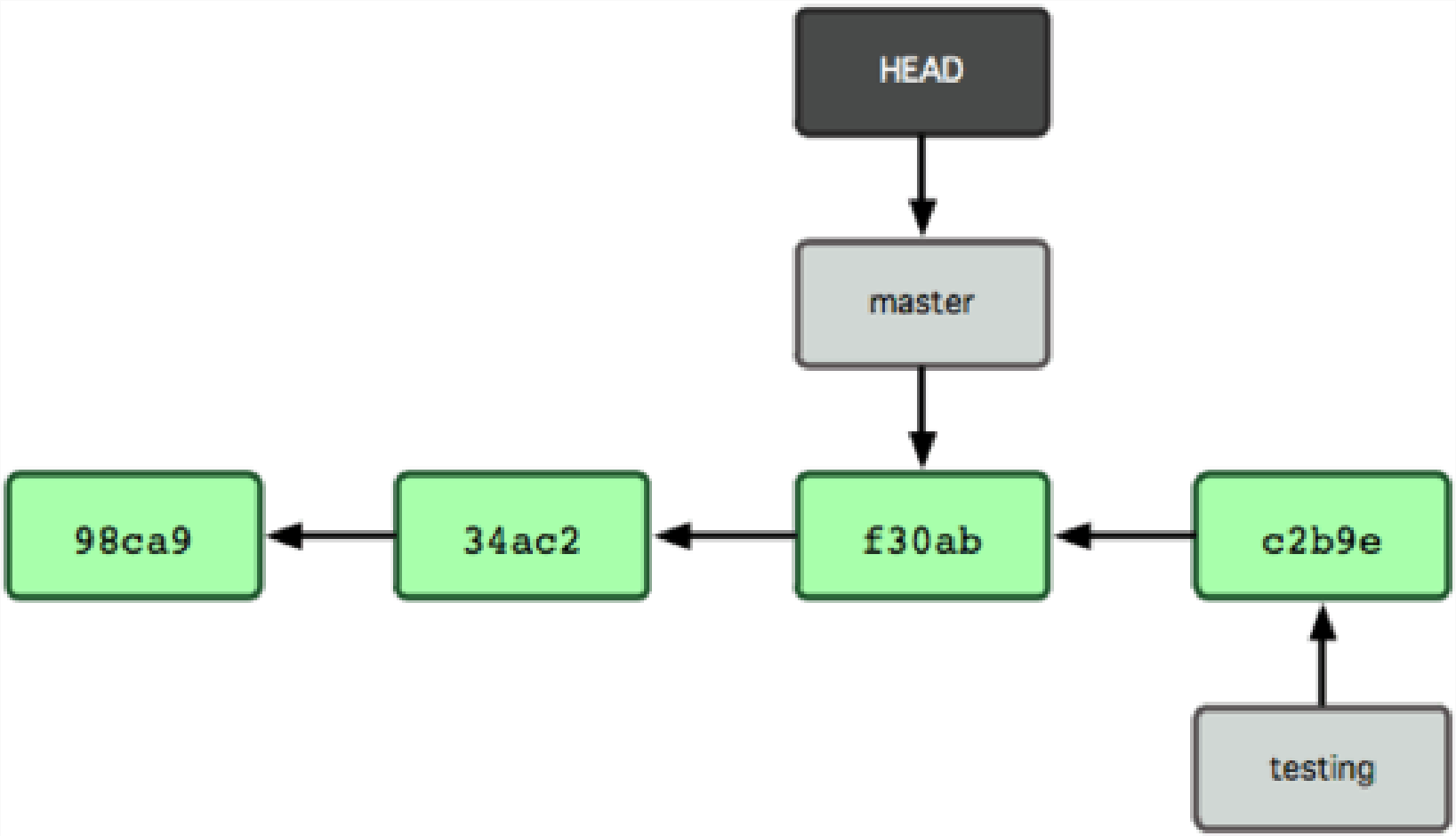
Treballant amb branques

1. Modifiquem el fitxer.txt i ho enviem al repositori



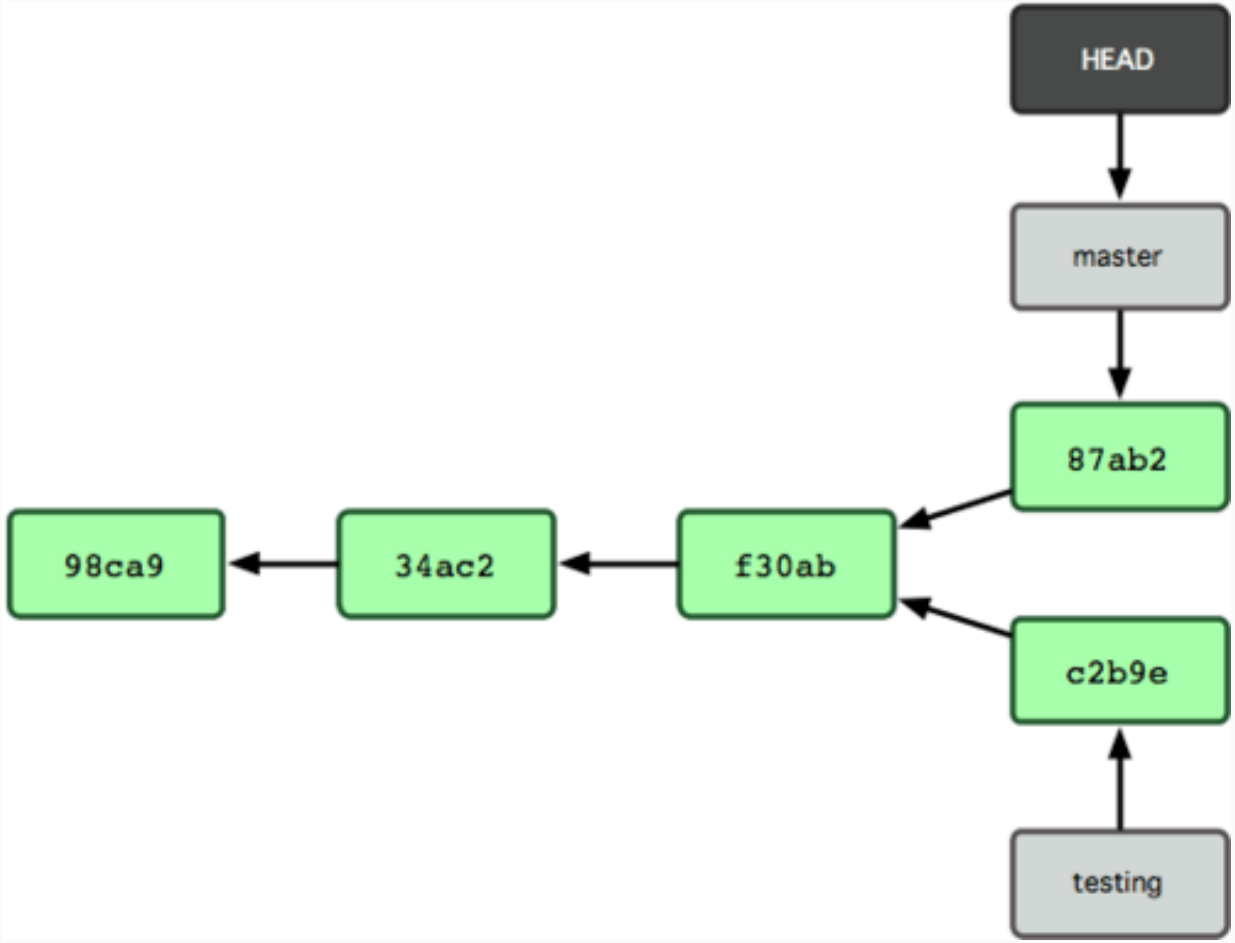
Treballant amb branques

1. Tornem a la branca master
2. `git checkout master`



Treballant amb branques

1. Modifiquem el fitxer.txt i ho enviem al repositori



Treballant amb branques - Merge

1. Entrem a la branca testing i afegim un nou fitxer
2. Si canviem de branca veurem que en una hi ha uns fitxers diferents que en l'altra
3. `git checkout master`
4. `git merge testing`

Treballant amb branques - Conflictes

1. Entrem a la branca testing, modifiquem el fitxer.txt en una mateixa linia que la master i l'enviem al repo
2. `git checkout testing`
3. `git add fitxer.txt`
4. `git commit -m "Desde de testing"`
5. Saltem a la branca master i fem un merge
6. `git checkout master`
7. `git merge testing`

Treballant amb branques

1. Anem a recuperar una versió anterior (d'un TAG)
2. `git checkout 1.0 -b branca1.0`
3. Fem els canvis pertinents
4. `git commit -m "Desde de 1.0"`
5. Saltem a la branca master i fem un merge
6. `git checkout master`
7. `git merge branca1.0`

Treballant amb branques - Detached HEAD

1. Podem anar a un commit en concret. En aquest cas, el HEAD no apuntarà a cap branca
2. `git checkout 1.0`
3. Estem en Detached HEAD state. Els commits, etc no es guardaran
4. Per tornar a l'estat normal fem un `git checkout master`
5. És millor fer una branca i treballar en aquesta. Sempre la podem esborrar un cop provat el que vulguem

Treballant amb branques - Esborrar branques

1. Moltes vegades podem fer branques per provar o corregir problemes en el projecte
2. Un cop fet i tenim el merge realitzat, podem esborrar-la amb un `git branch -d testing`

Excloure fitxers

1. Podem excloure fitxers utilitzant el fitxer `.gitignore`
2. Com a exemple, creem el fitxer test.exe
3. l'afegim al `.gitignore`
4. Si fem un `git status` no apareix
5. Afegit el `.gitignore` al repostitori
6. A la URL https://github.com/github/gitignore tenim fitxers ignore comuns per tipus de projectes

Pràctica

1. Crear el projecte "Test" de tipus consola els Visual Studio 2017 (C#) i afegir-ho a un repositori GIT

Stash

1. Podem acumular els canvis que hem fet per utilitzar més endavant
2. Fem alguns canvis el fitxer
3. Executem `git stash` i tornarem a tenir el fitxer anterior
4. Podem tenir més d'un stash a la cua. Podem veure la llista amb `git stash list`
5. Per `git stash list`
6. Per agafar un stash `git stash pop` o `git stash pop stash@2`
7. Tenir en compte que fa un merge amb l'actual

Connexió amb repositoris remots

1. Anem a descarregar un repositori remot
2. `git clone http...`
3. Ja tenim una copia del repositori remot en local
4. A partir d'aqui podem treballar localment sense necessitat de connexió

Enviar canvis al respositori central

1. Afegim algun fitxer al repositori
2. Fem un Commit
3. Enviem els canvis al respositori central: `git push`

Rebre canvis

1. Abans de rebre canvis hem de fer un **commit** del nostre repositori local
2. Fem un Pull per rebre els canvis: `git pull`
3. Si volem rebre tota la informació del repositori podem fer un fetch `git fetch`

Utilitzant un gestor GIT

Gestor GIT

Treballar amb un gestor GIT

No és fàcil treballar per linia de comandes i alguns editors o entorns tenen un gestor GIT limitat

- Utilitzar un gestor GIT que ens ajudi en el dia a dia
- Tenim moltes opcions: SmartGit, GitKraken, SourceTree, TortoiseGIT, etc.

Treballar amb un gestor GIT

Treballarem amb el SourceTree ja que utilitzem BitBucket i queda tot més integrat.

- Descarregar la última versió de https://www.sourcetreeapp.com

Gestor GIT - Pràctica

Pràctica

1. Crearem un projecte on hi participi varis usuaris
2. Primer un usuari serà el crearà l'estructura bàsica del projecte i el publicarà en un repositori
3. La resta d'usuaris farà una part del projecte

Altres comandes GIT

Altres comandes GIT

En la següent secció veurem altres comandes GIT on alguna d'elles s'han d'utilitzar amb cura.

Seran:

- log
- reset
- merge amb fast forward i sense
- cherry-picks
- rebase

Altres comandes GIT - Log

Anem a millorar una mica la visualització del log per linia de comandes

Executem el següent:

- Afegim uns quants commits on a cada commit afegit un nou fitxer i una branca b1 amb commits
- `log --all --decorate --oneline --graph`
- Veiem que la visualització és molt millor
- Crearem un alias per tenir-ho més fàcil `git config --global alias.adog "log --all --decorate --oneline --graph"`
- Si ara executem `git adog` tenim el mateix resultat

Altres comandes GIT - Reset

Ens mou la posició de la branca actual.

Provem:

- Primer crearem un nou repositori
- Afegim uns quants commits on a cada commit afegit un nou fitxer
- Per anar 3 posicions enrera `git reset HEAD~3`
- Mirem que ha passat amb els fitxers
- Altres opcions: –hard –soft
- `git reset <ID> --soft`
- `git reset <ID> --hard`

Altres comandes GIT - Fast Forward

Anteriorment haviem vist la comanda Merge per tal d'unificar branques. Per defecte s'utilitza el Fast Forward.

Provem:

- Verifiquem que tinguem alguns commits en la branca actual
- Ens guardem el ID de la master per poder tornar a enrera
- Crearem una nova branca "b1" i afegim alguns commits
- Tornem a la branca principal
- Mirem com tenim el log: `git log --graph`
- Fem un merge i mirem com tenim el log de nou
- Tornem enrere (reset) per tenir-ho com abans
- Fem un merge de nou però amb: `git merge --no-ff b1`
- Mirem el log altre cop

Altres comandes GIT - Cherry Pick

El Cherry Pick s'utilitza per fer còpies de commits a un commit nou.

Provem:

- Verifiquem que tinguem alguns commits en la branca actual i de la b1
- Copiem un commit de la branca b1 `git cherry-pick <ID>`
- Si la copia del commit ja havia estat incorporada anteriorment git ens avisa. Per exemple, `git cherry-pick master`

Altres comandes GIT - Rebase

Rebase és una alternativa al merge que ja coneixem, però deixa una història lineal en comptes de tenir un sol commit. Seria com automatitzar molts cherry-picks d'un sol cop.

Provem:

- Visualitzem el nostre històric de commits: `git adog`
- Ens guardem el ID de la master per poder tornar a enrera
- Fem un `git rebase b1`
- Mirem el log i veurem que la història és lineal a diferència del merge

Altres comandes GIT - Rebase: esborrar commits

Si volem esborrar commits consecutius (no l'últim) podem utilitzar rebase amb `--onto`

Provem:

- Visualitzem el nostre històric de commits: `git adog`
- Mirem els commits consecutius que volem esborrar
- Executem `git rebase --onto master 3 master 1 master`
- La comanda ens indica des de quin commit volem esborrar (master 3) fins el primer que volem mantenir (master 1)

Altres comandes GIT - Cherry Pick: esborrar commits

Si volem esborrar commits no consecutius farem un cherry-pick

Provem:

- Visualitzem el nostre històric de commits: `git adog`
- Mirem els commit just anterior al que volem esborrar
- Fem un checkout `git checkout <ID>`
- Creem una branca `git checkout -b arreglar`
- Afegim el commit següent al commit que volem esborrar `git cherry-pick <ID>`
- Repetim aquest pas per tots els commits que volem mantenir
- Tornem a la branca principal `git checkout master`
- Fem un Hard Reset al commit anterior al que volem esborrar `git reset --hard <ID>`
- Fem un merge `git merge arreglar`

Preguntes?

Gràcies per la vostra assistència