UF02 - Instal·lació, configuració i recuperació de programari

INDEX

1.-Llicència en el programari

Qué és una llicència

La llicència de programari és, l'autorització que l'autor o autors, que són els que ostenten el dret intel·lectual de l'obra, concedeixen a altres per utilitzar-lo complint una sèrie de termes i condicions establertes dins de les seves clàusules. És a dir, és un conjunt de permisos que un desenvolupador li pot proporcionar a un usuari en què té la possibilitat de distribuir, utilitzar o modificar el producte sota una llicència determinada.

Les llicències de programari poden establir entre altres coses:

- La cessió de determinats drets del propietari a l'usuari final sobre una o diverses còpies del programa informàtic
- Els límits en la responsabilitat per fallades
- El termini de cessió dels drets
- L'àmbit geogràfic de validesa del contracte
- Establir determinats compromisos de l'usuari final cap al propietari. Per exemple, la no cessió del programa a tercers o la no reinstal·lació del programa en equips diferents al que es va instal·lar originalment.

Parts essencials en una llicencia de programari

- **LLicenciant**: és aquell que proveeix el programari més la llicència al llicenciatari, la qual, li permetrà a aquest últim tenir certs drets sobre el programari.
  El paper de llicenciant el pot exercir qualsevol dels següents actors:
  - **Autor (o conjunt d'autors)**: que crea el programari són qui en una primera instància posseeixen el paper de llicenciant en ser els titulars originals del programari.
  - **Titular dels drets d'explotació:** és la persona jurídica que rep una cessió dels drets d'explotació de manera exclusiva del programari des d'un tercer, transformant-lo en titular derivat.
  - **Distribuïdor**: és la persona jurídica a la qual se li atorga el dret de distribució i la possibilitat de generar subllicències del programari mitjançant la signatura d'un contracte de distribució amb el titular dels drets d'explotació i producte.
- **Garantia de titularitat**: És la garantia en la qual, s'assegura que el llicenciant té suficients drets d'explotació sobre el programari per permetre proveir una llicencia al usuari.
- **Usuari de la llicencia:** És aquella persona física o jurídica que té permís per utilitzar aquell programari complint les condicions establertes per la llicència atorgada pel llicenciant.
- **Condicions d'ús:** Plec de condicions que l'usuari de la llicència hauria de complir per tal de poder continuar utilitzant el programari del llicenciant.

- **Termini**: El termini determina la durada (en temps) per la qual els termes i condicions establerts en la llicencia continuen en vigor.
- **Preu**: El preu es determina el valor pel qual l'usuari de la llicència haurà d'abonar (si escau) al llicenciant, i amb quins terminis.

Llicència programari propietari

Als inicis, la majoria de programari desenvolupat era programari amb llicència propietaria. Aquest tipus de llicències, en general, no permeten les següents accions:

- Modificació del programari.
- Descompilació del programari.
- Copiar i/o distribuir lliurement el programari.
- Regulació de número de còpies que es poden instal·lar.
- Finalitat d'ús del programari.
- Limitació de la responsabilitat dels autors del programari en cas d'errors inesperats.

Els autors de programes sotmesos a aquest tipus de llicències en general ofereixen serveis de **suport tècnic i/o actualitzacions** durant el temps de vida del producte. Un cop finalitzada la llicencia, el suport tècnic i/o actualitzacions també deixen d'esser efectius. Tanmateix, també és possible que el mateix programari quedi inutilitzat.

Alguns exemples d'aquest tipus de llicències són les anomenades acord de llicència d'usuari final (ALUF), també anomenat en alguns països contracte de llicència d'usuari final (CLUF).
Al món anglosaxó es diu end-user license agreement (EULA).

Llicència programari lliure

El programari lliure és el programari que pot ser usat, estudiat i modificat sense restriccions, i que pot ser copiat i redistribuït bé en una versió modificada o sense modificar sense cap restricció, o bé amb unes restriccions mínimes per garantir que els futurs destinataris també tindran aquests drets.

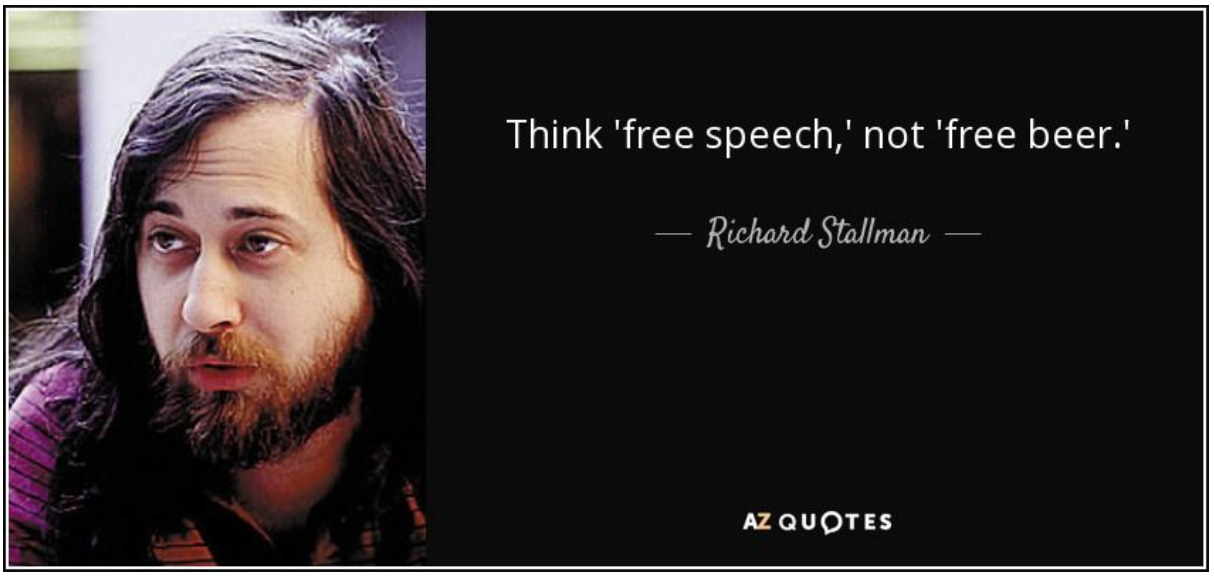
De fet, existeixen diverses definicions de programari lliure i diversos tipus de llicències per la seva distribució.

D'acord amb la definició de la *Free Software Foundation* un programari és lliure quan es garanteixen les següents llibertats:

1. Llibertat d'utilitzar el programari amb qualsevol propòsit.
2. Llibertat per estudiar com funciona el programa, modificar-lo i adaptar-lo a les teves necessitats.
3. Llibertat per distribuir còpies del programa.

4. Llibertat de millorar el programa i fer públiques aquestes millores per a tothom, de manera que tothom en surti beneficiada.

Nota: cap de les propietats especifica el cost de la mateixa



Tipologia de llicències de programari lliure més conegudes i utilitzades:

- La llicència "Creative Commons"
- La llicència (Berkeley Software Distribution)
- La llicència "Apache Software License"
- La llicència "GNU General Public License" (GPL)
- La llicència "GNU Library or Lesser General Public License" (LGPL)
- La "Mozilla Public License" (MPL)

Exercicis

1. Busca un exemple de llicència lliure pels següents programaris:
    a. Solució ofimàtica
    b. Antivirus
    c. Sistema operatiu.
    d. Client FTP
2. Realitza una comparació rapida entre dues llicències. Una llicència privativa i l'altre lliure d'un programari d'ofimàtica comú (Microsoft Office i Libre Office)
   https://www.libreoffice.org/about-us/licenses/
   https://www.microsoft.com/en-us/licensing/product-licensing/products#PT
3. Busca una llicència lliure i adequada per algun programari que haguis desenvolupat. Pots utilitzar qualsevol de les nombrades (MIT, BSD, Creative Commons...etc)

2.- Repositoris de codi

El repositoris de codi és un sistema que registra els canvis realitzats sobre un arxiu o conjunt d'arxius al llarg del temps, de manera que puguis recuperar versions específiques més endavant.
Si ets dissenyador gràfic, dissenyador web, administrador de sistemes i vols mantenir cada versió d'una imatge/disseny, un sistema de repositori de codi és una tria molt inteligent.
Et permet:

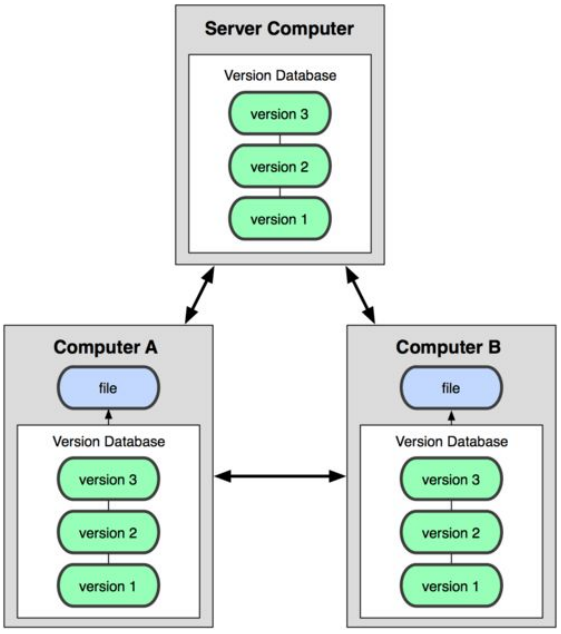
1. Revertir arxius a un estat anterior
2. Revertir el projecte sencer a un estat anterior
3. Comparar canvis al llarg del temps
4. Veure qui va modificar per última vegada alguna cosa que pot estar causant un problema
5. Qui va introduïr un error i quan

Fer servir un repositori també significa generalment que quant perds arxius, pots **recuperar-los** fàcilment.

Un mètode de control molt utilitzat (i desaconsellat) és el sistema de copy&paste. És a dir, copiem fitxers d'un directori a un altre i ja tenim un sistema de versions i també un sistema de còpies de seguretat pels fitxers de configuració. Aquest sistema no és apropiat perquè és fàcil oblidar en quin directori estàs treballant, i guardar accidentalment a l'arxiu equivocat o sobreescriure arxius que no volies.

La solució definitiva per solucionar aquest problema és el **repositori de codi**, i concretament amb els **sistemes de versions distribuïts**. Trobem varis programaris per implementar un repositori amb un sistema de versió distribuïts (**GIT**, Mercurial, Bazaar o Darcs)

Els clients d'un repositori amb un sistema de versió distribuït treballen amb un servidor central, però tenen una còpia replicada idèntica del servidor. **Entendre aquest punt és important**, perquè si el servidor principal queda innaceccible qualsevol client pot replicar la seva base de dades i restaurar el servei completament sense perdre les dades ni l'historial. En el següent gràfic es pot veure:



Breu historia del GIT

El nucli de Linux és un projecte de programari de codi obert amb un abast bastant gran. Durant la major part del desenvolupament del nucli (1991-2002), els canvis en el programari es van passar en forma de patch i arxius. El 2002, el projecte del nucli de Linux va començar a fer servir un repositori de codi propietari, BitKeeper.
A l'any 2005 l'empresa desenvolupadora de BitKeeper va canviar la seva llicencia i es va deixar d'oferir l'eina de forma gratuïta. Aquest fet, va motivar a la comunitat desenvolupadora de Linux a **crear una alternativa**. Per aquest motiu va néixer l'eina GIT, a més, aquesta eina oferia millores respecte a la solució de Bitkeeper:

1. Velocitat
2. Diseny més simple
3. Possiblitat de desenvolupament amb branques paral·leles (veurem això més endevant)
4. Sistema distribuit
5. Capacitat de manejar sistemes molt més grans.

Concepte bàsic de funcionament del GIT

1. La majoria de les operacions en GIT només necessiten fitxers i recursos locals. Això significa que per navegar al teu historial o confirmar canvis en el teu codi no necessites cap servidor intern. Per tant, pots treballar sense accés a Internet o sense necessitat d'accedir al servidor de dins l'empresa. Aquesta una diferència principal amb altres gestors de codi.
2. Tots els canvis en GIT són verificats per una suma d'integritat, un hash SHA-1.
3. Sempre que realitzes canvis dins del GIT, sempre afegeixes informació. Això és especialment interessant perquè difícilment podràs eliminar fitxers o corrompre la base de dades.

Els tres estats

Aquest és el concepte **més important,** per compendre com funciona un sistema amb GIT. Bàsicament GIT disposa de tres estats de treball en el que es poden localitzar els fitxers on treballem:

1. **Modificat (Modified)**: Els arxius que estan en aquest estat significa que s'han modificat però encara no s'han confirmat.
2. **Preparat (Staged)**: Significa quan has marcat a un fitxer modificat en la seva versió actual (no faràs més modificacions) i el pujaràs en la següent confirmació.
3. **Confirmat (Committed)**: significa que les dades estan emmagatzemades de manera segura en la teva base de dades local. Un cop els fitxers es troben en committed aquests queden guardats dins la base de dades. Si es vol fer alguna modificació/eliminació s'haura de fer un commit posterior perque aquest quedi guardat a la base de dades.

El flux normal de treball acostuma a tenir un aspecte similar al següent:

1. Modifiques una sèrie de fitxers.
2. Prepares els fitxers, afegint-los a l'àrea de preparació.
3. Confirmes els canvis. Els fitxers confirmats s'agafen de l'àrea de preparació i es guarda una instantània dels mateixos a la base de dades del GIT.

En resum, si al GIT tenim una versió concreta comfirmada d'un fitxer es considera *commited*. Si hi ha canvis en aquest fitxer des de l'ultima confirmada, però s'han afegit els canvis a l'area de *preparació*, es considera que estan *staged*. I si el fitxer ha petit canvis, però no estàn preparats es considera *modified*.

Inicis bàsics de GIT

El primer pas és insal·lar el GIT per poder realitzar operacions bàsiques.
DEBIAN

```
apt-get install git
```

RHEL

```
dnf install git-core
yum install git-core
```

WINDOWS

```
https://git-scm.com/download/win
```

Nota: en el cas de Windows recomano previament instal·lar el Notepad++.
Durant el procés d'instal·lació marcar com a editor el Notepad (és més visual)

MAC

```
https://git-scm.com/download/mac
```

Un cop instal·lat configurarem els paràmetres bàsics de configuració del GIT.

La identitat

Aquest punt és important, perquè totes les confirmacions aniran signades amb aquest usuari i aquest correu electrònic.

```
$ git config --global user.name "Pere Casas"
$ git config --global user.email ppuig@lasalle.cat
```

Comprovant la teva configuració

```
$ git config --global --list
```

Per modificar qualsevol paràmetre només has de fer-ho des del GIT, utilitzant la comanda GIT seguit de la variable i el nou valor. Per exemple d'aquesta forma:

```
$ git config --global <variable> <nou_valor>
```

Tota la informació relativa a l'ajuda del GIT la podem localitzar de la següent manera:

```
$ git help <comanda>
$ git <comanda> --help
$ man git-<comanda>
```

Fonaments del GIT

És possible obtenir un projecte GIT de dues maneres ben diferents. La primera és realitzar un clone d'algun repositori que ja estigui en funcionament i l'altre forma és crear un repositori des de zero. Nosaltres comancerem creant un repositori des de zero de la següent manera:

```
$ git init
```

Aquesta comanda crea un subdirectori (.git) amb totes les configuracions necessaries per començar.
Ara podriem crean un fitxer, afegir-lo al index i confirmant els canvis:

1. Primer a l'arrel del nostre projecte creem un fitxer qualsevol
   nano index.html (per exemple)
2. Un cop creat, afegim el index.html de la següent manera:

   ```
   $ git add index.html
   ```

3. Un cop hem afegit el arxiu al GIT, si ja hem acabat de finalitzar els canvis els podem confirmar. **IMPORTANT:** sempre que confirmem hem d'afegir un petit comentari explicant els canvis que realitzem.

   ```
   $ git commit -m "Inici del projecte"
   ```

4. Fixem-nos que un cop es realitza el commit, podem veure tot el que ha fet:

   ```
   [master (root-commit) f9128d2] inici del projecte
    1 file changed, 6 insertions(+)
    create mode 100644 index.html
   ```

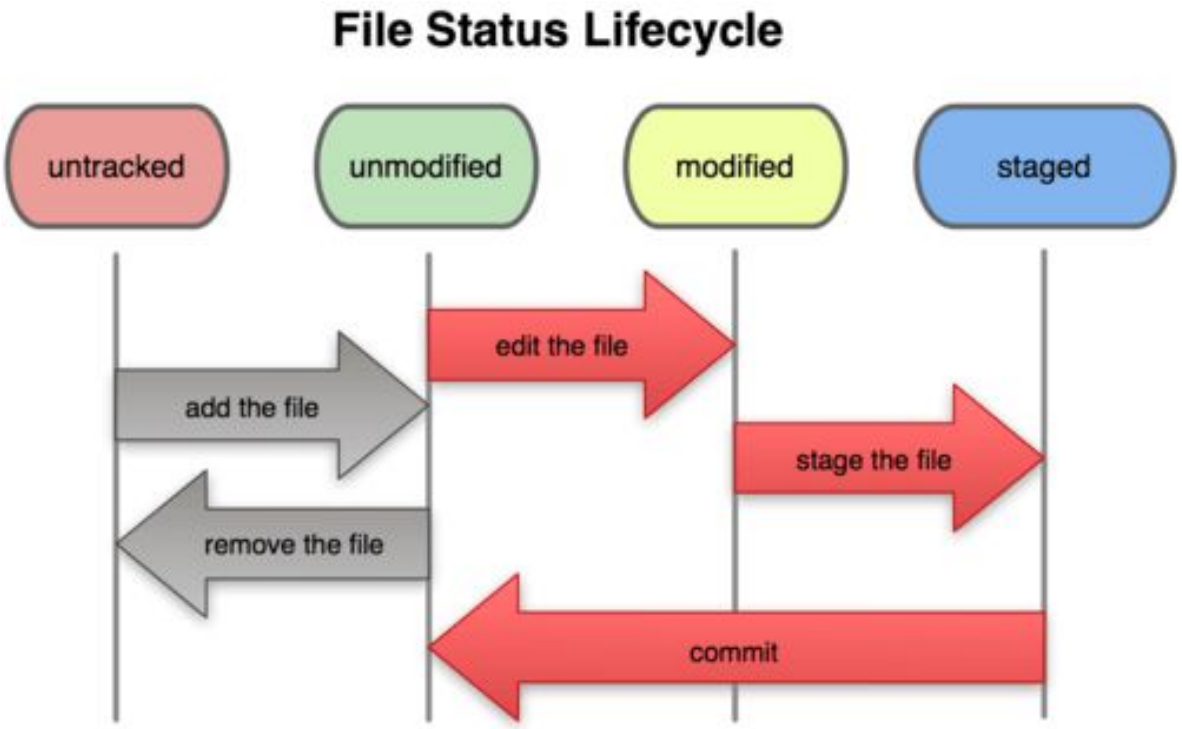
5. Més endavant entrarem més en detall, per veure cada comanda que realitza exactament.

Guardant canvis en el repositori

Tenim un repositori GIT complet, i una copia dels arxius de treball d'aquest projecte. Ara, cada cop que modifiquem fitxers, podrem guardar l'estat del nostre projecte confirmant aquests canvis.
Arrivats aquest punt és important entendre que existeixen dos tipologuies de d'estat:

- **Tracked**: són fitxer que existien quan varem fer el ultim commit (salvar el snapshot). Aquests fitxers poden estar modificats, no modificats o preparats.
- **Untracked (sense seguiment):** Són la resta de fitxers que no estaven en l'ultim commit, ni en l'area de preparació.

A mida que vas modificant fitxers GIT, els va mercant com a modificats, perque han canviat des del últim **commit.** En definitiva, en el següent gràfic hi ha un esquema del cicle de vida dels ftixers dins d'un repository GIT.



La forma més fàcil de veure el estat en que es troba el teu repository és aplicant la següent comanda:

```
[casas@casas test]$ git status
On branch master
nothing to commit, working tree clean
```

Aquesta sortida ens ensenya com actualment no hi ha canvis pendents de confirmar i conforme tenim el espai de treball net. Per altre banda, GIT **tampoc** veu cap fitxer que no estigui sota seguiment (untracked) sinó també el llistaria aqui.

Afegir nous arxius al seguiment de GIT

Per començar el seguiment de un nou fitxer, només s'ha d'utilitzar la comanda:

```
$ git add <nou_fitxer>
```

Si tornem a exectura el git status veurem que ens marca el fitxer com a pendent d'afegir. En la descripció ens marca qui ha el fitxer pendent d'afegir.

Preparant fitxers modificats

Un cop tenim els fitxers en seguiment els podem modificar. En fer-ho, quan executem la comanda

```
$ git status
```

Ens marcarà que hi ha canvis pendents en la secció “Changes not staged for commit”:

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.html
```

Per preparar els canvis, novament tornarem a aplicar la comanda :

```
$ git add <nou_fitxer>
```

Recorda que aquesta comanda és multi-usos. Un cop preparats els fitxer tornem a mirar el status i veurem com ens marca el fitxer amb un color diferent i en la secció “Changes to be committed”. Això significa que està preparat per realitzar el canvi.

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.html
```

NOTA: Un cop preparats els canvis en un fitxer el tornes a modificar t'apareixerà el mateix fitxer en les dues seccions de forma simultània. Per exemple:

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   index.html

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.html
```

En aquesta situació, si guardessim el canvis (commit) només es guarderien els canvis efectuats en el moment de executar la comanda “git add <file>”. Si volem sobrescriure els canvis preparats haurem de tornar a executar la comanda “git add <file>”

NOTA: Per simplificar el procés de preparar els fitxers podem executar la comanda:

```
$ git add .
```

Ignorant fitxers

És usual que hi haguin fitxers que no els volem tenir dins el repositori, per exemple, fitxers que només són utilitzats en desenvolupament o bé fitxers amb proves temporals. Per tal que aquests fitxers no es puguin incloure dins el repositori i el GIT els ignori es pot afegir un fitxer anomenat .gitignore.
Aquest fitxer treballa amb expressions i els seus principals patrons són els següents:

- Línies en blanc o bé amb el símbol # seran ignorades.

- Per indicar un directori s'utilitza el símbol "/"
- Per negar un patró s'utilitza el símbol "!"
- Per senyalar qualsevol objecte s'utilitza el símbol "*"

Exemple fitxer .gitignore

```
# Aquesta línia serà ignorada

# Aquest fitxer serà ignorat
test.txt

# Aquest directori serà ignorat
/dev/*

# Tots els fitxers acabats amb .tsj seràn ignorats
*.tsj
```

Veure els detalls dels canvis

Preparats vs no preparats

Si la comanda *git status* consideres que és massa imprecisa, ja que no permet veure les diferencia podem fer ús de la comanda

```
$ git diff
```

En el cas anterior que teníem el fitxer index.html en dos estats de forma simultanis (preparat i no preparat) amb aquesta comanda veuríem quins canvis s'hi han introduït o que s'ha eliminat.

```
diff --git a/index.html b/index.html
index b49fd0a..5af89ee 100644
--- a/index.html
+++ b/index.html
@@ -2,5 +2,6 @@
 <head></head>
 <body>
        <p> Això és un test! si, és un test</p>
+       <p>Que si!<p>
 </body>
 </html>
```

És important fixar-nos, que la comanda ens indica mitjançant els símbols " **+** i **-** " si s'han afegit/modificat canvis, o s'han eliminat línies.

Preparats vs guardats

Aprofitant aquesta comanda també podem visualitzar els canvis que s'han produït i preparat, i posteriorment és guardaran.

```
$ git diff --staged
```

Per exemple, en la següent captura és pot visualitzar com en els canvis que tinc preparats s'han eliminat 3 liníes respecte a la copia confirmada:

Preparats

Per finalitzar amb aquesta comanda, també podem veure tot el conjunt de canvis que ara mateix estan preparats, és a dir, a punt de confirmar:

```
$ git diff --cached
```

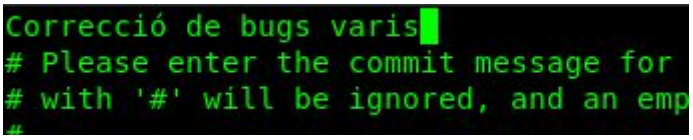
Confirmant els canvis

Un com tenim la preparació com nosaltres volem, és moment de confirmar tots els canvis.
IMPORTANT: recorda que tots els canvis pendents de preparar no es confirmaran. Si vols preparar-los els has de preparar amb la comanda *git add*
Per assegurar-ho, pots utilitzar la comanda *git status*

Per confirmar els canvis has d'executar la comanda:

```
$ git commit
```

En executar-la, s'obrirà un editor de text en la consola. En aquest editor hauríem d'escriure el missatge on descrivim els canvis que realitzem. Per exemple, correcció de bugs:



Per sortir del editor, en el cas de Linux hem de premer la tecla [ESC] + :w + :q

Un cop sortim del editor els nostres canvis han estat confirmats.

Opcionalment per estalviar-nos entrar en el editor, podem escriure directament el missatge mentres invoquem la comanda:

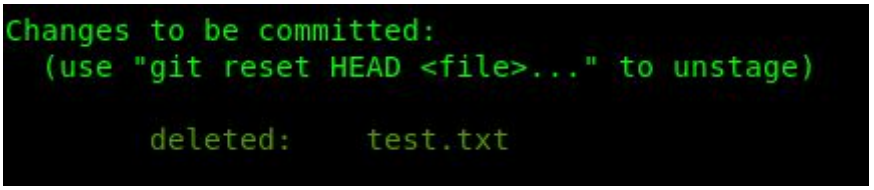
```
$ git commit -m "Correcció de bugs varis"
```

Eliminar fitxers

Per eliminar un fitxer s'ha d'eliminar del seguiment, i un cop es confirmi aquest ja desapareixerà. Per fer-ho és tant simple com executar la comanda:

```
$ git rm <arxiu>
```

Podem comprovar que passarà quan guardem consultant-ho a *git status*



En el cas de la captura, un cop realitzi el commit el fitxer test.txt desapareixarà del repositori.

Moure fitxers

Per moure fitxers (canviar de nom) és realitza amb la comanda

```
$ git mv <origen> <desti>
```

Podem comprovar que passarà quan guardem consultant-ho a *git status*

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    test.txt -> test2.txt
```

En el cas de la captura, un cop realitzi el commit el fitxer test.txt desapareixarà del repositori.

Historial de canvis

Cada cop que guardem algun canvi al GIT, nosaltres hem d'introduir un missatge descriptori. Aquest missatge és de gran ajuda, doncs ens permet visualitzar tot el historial. Per poder-lo consultar només hem d'executar la comanda:

```
$ git log
```

Per defecte la comanda ens retorna tota la informació, i en algunes ocasions ens potser complicat d'interpretar. Per minimitzar aquest problema podem utilitzar el paràmetre -p Aquesta mostra les diferencies introduïdes en cada confirmació. Si completes amb el valor 2 només és mostrem les 2 últimes entrades:

```
[casas@casas test]$ git log -p -2
commit 279403115d21da80415404156806d52ac5b26602 (HEAD)
Author: pcasas@atendata.com <pcasas@atendata.com>
Date:   Thu Dec 13 17:49:24 2018 +0100

    add test.txt

diff --git a/test.txt b/test.txt
new file mode 100644
index 0000000..4871fd5
--- /dev/null
+++ b/test.txt
@@ -0,0 +1 @@
+test.txt

commit 64462af9d3effd24d03c5f62fc91c8463bc49d25
Author: pcasas@atendata.com <pcasas@atendata.com>
Date:   Thu Dec 13 17:41:14 2018 +0100

    Correcció de bugs varis

diff --git a/index.html b/index.html
index 32340e1..b49fd0a 100644
--- a/index.html
+++ b/index.html
@@ -2,8 +2,5 @@
 <head></head>
 <body>
        <p> Això és un test! si, és un test</p>
-       <p>Això és un test2</p>
-       <p>Això és un test3</p>
-       <p>Això és un test4</p>
 </body>
 </html>
```

Per ultim, una de les formes més elegant i utlitzades es personalitzant la linea de sortida. Les dues més utilitzades (per mí) són les següents:

```
$ git log --pretty=oneline
$ git log --pretty=format:"%H :: %an :: %cr :: %s"
```

Revertint canvis

Usualment ens interessa eliminar tots els canvis que hem fet en un projecte des del últim commit, o bé ens pot interessar restaurar un fitxer en un estat anterior (commit antic).

Per resetejar completament el nostre projecte al últim commit realitzat ho farem amb la comanda següent:

```
$ git reset --hard
```

Per restaurar un estat guardat primer haurem de saber quin ID tenia aquell estat. Un cop el tenim ja podem restaurar. Ho realitzarem de la següent forma:

```
$ git log   #busquem el commit que volem revertir
$ git checkout <IdCommit>
```

Activitat 1

IMPORTANT: Guardeu el directori de treball. Més endavant s'haurà de pujar en un repositori remot.

1. Afegeix un fitxer nou, per exemple README.md i torna a executar la comanda *git status*. Quin resultat el mostra?
2. Afegeix el fitxer a preparat i posteriorment confirma els canvis.
3. Afegeix un nou fitxer anomentat index.html. Afegeix-lo a preparat.
4. Edita el fitxer index.html → Un cop modificat el git status que mostra?
5. Confirma els canvis.
6. Ara mateix en quin estat es troba el GIT?
7. Afegeix un nou fitxer anomenat test.txt. Prepara els canvis i confirma.
8. Un cop realitzat el punt anterior, canvi el nom del fitxer test.txt amb la comanda pertinent. Un cop realitzada executa git status. Que és mostra?
9. Un cop realitzat el punt anterior, elimina el fitxer test.txt amb la comanda pertinent. Un cop realitzada executa git status. Que és mostra?
10. Abans de guardar els canvis comprova quina diferència hi ha entre el status "preparat" i "no preparats"
11. Un cop finalitzat l'exercici anterior, prepara i guarda els canvis. Mostra el resultat de la comanda i de la comanda *git status*

12. Ara abans de continuar assegura que tots els fitxerss del directori estan guardats en un commit. Si tens dubte, afegeix-los tots com a preparats (git add.) i després fes commit.
13. Elimina tot el contingut de la carpeta a excepció del directori .git
14. Restaura el teu projecte mitjançant la comanda apropiada de GIT.
15. Confirma que s'han restaurat tots els fitxer.
16. Un cop finalitzat l'exercici anterior, mostra l'historial de tots els canvis que has realitzat amb tres formats gràfics.

Les branques en el GIT

Per defecte sempre treballem a la branca "master". Però es poden crear múltiples branques de treball. Això és important per poder diversificar el nostre treball i treballar paral·lelament del codi que estigui en producció i un cop estem segur que tenim una versió estable els podrem unificar. O simplement ens pot interessar per poder tenir una còpia idèntica d'algun repositori de codi.
Per visualitzar les branques executarem aquesta comanda:

```
$ git branch
```

Amb la comanda següent visualitzarem les branques locals i les remotes*
*nota: el tema del repositoris remots els veurem més endavant.

```
$ git branch -a
```

Per eliminar branques existents:

```
$ git branch -d <nom_branca>
```

Per afegir una nova branca:

```
$ git branch  <nom_branca>
```

Per canviar-nos de branca i començar a treballar amb la nova branca:

1. Primer comprovarem a quina branca treballem amb la comanda git branch . Sabrem quina és la nostra perquè estarà marcada amb el símbol *
2. Un cop sabem els noms de les branques canviarem de branca amb la següent comanda:

   ```
   $ git checkout  <nom_branca>
   ```

3. Si tornem a executar la comanda git branch veurem que el símbol ha canviat de branca.

4. A partir d'aquest moment tots els canvis que apliquem en aquesta branca no afectaran a l'altre branca.

Activitat 2

1. En un directori nou, crea un nou repositori. Crea dos fitxers bàsics (index.php i db.php)
2. En l'interior d'aquest fitxer només guarda una sola línia de text "branca màster"
3. Afegeix, prepara i fes un commit inicial.
4. Crea una branca nova, amb el nom que vulgui.
5. Canvia a la segona branca. Un cop dins, modifica la línia del fitxer (index.php) i escriu "branca secundària".
6. Elimina el fitxer db.php
7. Prepara i guarda els canvis.
8. Torna a la branca màster, i comprova si existeixen els fitxers.
9. Comprova si els canvis que has realitzat en l'exercici anterior existeixen en la branca màster.