

**UNIVERSIDAD TECNOLÓGICA  
NACIONAL**

**FACULTAD REGIONAL RESISTENCIA**



**INGENIERÍA EN SISTEMAS  
DE INFORMACIÓN**

**Desarrollo de Aplicaciones Cliente-Servidor**

Grupo N°3:

Retamozo Agustín ([ruben7are@gmail.com](mailto:ruben7are@gmail.com))

Edgardo Gasparutti ([edgardogasp@gmail.com](mailto:edgardogasp@gmail.com))

2021

## Actividad 1:

### Informe de investigación de Sistema de Control de Versiones

- Investigar y elaborar un breve informe de sistemas de control de versiones disponibles en el mercado, tanto del tipo centralizado como descentralizado (entre 5 y 8, ejemplo git, mercurial, svn, cvs, bitkeeper, etc). Indicar los siguientes ítems:

Tipos de versionado soportados.

1. Licencia
  2. Costo (gratis / propietario)
  3. Quien lo mantiene.
  4. Plataformas soportadas (Windows, Unix, etc)
  5. Extras
1. Elaborar un cuadro comparativo que resuma los puntos antes mencionados
  2. Realizar el mismo cuadro con plataformas comerciales de sistemas de control de versiones (entre 5 y 8, por ejemplo Atlassian, Github, etc) agregando la columna sistemas de control de versiones que soporta mencionadas en el punto anterior. Además mencionar que herramientas adicionales incluyen (por ejemplo wiki, herramientas de gestión de proyectos, etc).

#### Git

GIT es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código.

#### Mercurial

Mercurial es una herramienta de gestión de control de fuente distribuida y gratuita. Le ofrece el poder de manejar proyectos de cualquier tamaño de manera eficiente mientras usa una interfaz intuitiva. Es fácil de usar y difícil de romper, lo que lo hace ideal para cualquiera que trabaje con archivos versionados.

Se escribió pensando en la independencia de la plataforma. La mayor parte de Mercurial está escrito en Python, con una pequeña parte en C portátil por razones de rendimiento.

Como resultado, las versiones binarias están disponibles en todas las plataformas principales.

## SVN

SVN es un sistema de control de versiones libre, multiplataforma y de código abierto bajo la licencia de APACHE. Maneja ficheros y directorios a través del tiempo almacenando los en un repositorio central. SVN Permite recordar todos los cambios hechos a sus ficheros y directorios y así poder recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.

## CVS

CVS (Sistema de versiones concurrentes) funciona como una interfaz para RCS, un sistema anterior que opera en archivos individuales. Se expande sobre RCS al agregar soporte para el seguimiento de cambios a nivel de repositorio y un modelo cliente-servidor.

Publicado bajo los términos de la Licencia Pública General GNU, CVSD es un software gratuito.

El software del servidor normalmente se ejecuta en Unix, mientras que los clientes CVS pueden ejecutarse en cualquier plataforma del sistema operativo principal.

## Bitkeeper

Es un SCM distribuido, rápido y listo para la empresa, que escala hasta proyectos muy grandes y hasta pequeños. Es el sistema de gestión de fuentes distribuidas original, ahora disponible como código abierto bajo la licencia Apache 2.0.

Sis de Control de Versiones	Git	Mercurial	SVN	CVS	Bitkeeper
Tipo de Versionados	Distribuido	Distribuido	Centralizado	Centralizado	Distribuido
Licencia	Software libre	Software libre	Software libre	Software libre	open-source license.

Costo	0	0	0	0	0
Mantenimiento	Supervisado por Junio Hamano y comunidad	Comunidad	Comunidad	Comunidad	Comunidad
Plataformas Soportadas	Windows, Linux, MacOS, Solaris 11 express y otros	Windows, Linux, MacOS, Solaris 11 express y otros	Windows, Mac, Debian, Ubuntu, Solaris, Fedora, etc	Windows, Linux, MacOS	AIX, FreeBSD, HP-UX, IRIX, Linux, Mac OS X, NetBSD, OpenBSD, Solaris, Windows

#### Actividad 2: Análisis y utilización de un Sistema de Control de Versiones Centralizado

- Investigar un SCV Centralizado y explicar las principales características brevemente.
- Enumerar ventajas y desventajas, y comparar con SCV Descentralizados (cuadro comparativo).
- Seleccionar un servidor que se encuentre en la nube/web gratuito para realizar un ejemplo.
- Iniciar el repositorio, clonarlo, modificarlo y generar conflictos, crear ramas y realizar merge de las mismas con el trunk principal, en un pequeño equipo por lo menos 3 miembros del grupo.
- Utilizar de ser necesario una herramienta cliente (gráfico o consola) o IDE.
- Documentar el ejemplo con capturas de commits de los miembros del equipo sobre un mismo archivo y otro ejemplo de branch y merge.

#### SVN:

SVN es un sistema de control de versiones libre, multiplataforma y de código abierto bajo la licencia de APACHE. Maneja ficheros y directorios a través del tiempo almacenando los en un repositorio central. SVN Permite recordar todos los cambios hechos a sus ficheros y directorios y así poder recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.

	Git	SVN
Arquitectura de servidor	Actúa como cliente servidor. Cada desarrollador tiene una copia local del historial de versiones completo del proyecto en su máquina individual. Los cambios de Git ocurren localmente, por lo tanto, no tiene que estar conectado todo el tiempo. Las operaciones locales son mas rápidas.	Tiene un servidor y cliente separados. Solo los archivos en los que está trabajando un desarrollador se mantienen en la máquina local, y se debe estar en línea, trabajando con el servidor. Los usuarios extraen archivos y devuelven los cambios al servidor.
Repositorios	Almacenar archivos binarios grandes en git no es realista. Los desarrolladores pasan tiempo esperando para verificar el repositorio completo en su computadora. Cada vez que se cambia y confirma un archivo grande, los repositorios de Git crecen exponencialmente.	Solo el árbol de trabajo y los últimos cambios se comprueban en las máquinas locales. Los checkouts toman menos tiempo en SVN cuando hay muchos cambios en los archivos binarios.
Branching	Las ramas de git son solo referencias a un determinado commit. Son ligeros, pero potentes. Puede crear, eliminar y cambiar una rama en cualquier momento, sin afectar los commits. Si necesita probar una nueva función o encuentra un error, puede hacer una rama, realizar los cambios, enviar el commit al repo central y luego eliminar la rama.	Las ramas se crean como directorios dentro de un repositorio. Esta estructura es el principal problema con la ramificación de SVN. Cuando la rama está lista, se une de nuevo con el tronco. Es posible que su versión no refleje las ramas de los desarrolladores, esto significa que los conflictos, los archivos faltantes y los cambios desordenados acechan a su rama. Esto lo convierte en un modelo complicado de ramificación y fusión. También requiere mucho tiempo de gestión.

Controles de acceso	De forma predeterminada, Git asume que todos los contribuyentes tienen los mismos permisos.	SVN permite especificar controles de acceso de lectura y escritura por nivel de archivo y por nivel de directorio.
Auditabilidad	La naturaleza distribuida de Git permite que cualquiera pueda cambiar cualquier parte del historial de su repositorio local. Los cambios se registran a nivel de repositorio. Aunque se desaconseja mucho impulsar un historial modificado, puede suceder. Esto causa problemas si otros desarrolladores confían en cambios particulares.	Con SVN, el historial de cambios del repositorio es bastante consistente. Para realizar cualquier cambio en el historial del repositorio, necesita acceso al servidor central. Los cambios se registran a nivel de archivo.
Requisitos de almacenamiento	Los repositorios de Git no pueden manejar archivos binarios grandes.	Los repositorios SVN pueden manejar archivos binarios grandes, además de código. El almacenamiento de archivos binarios grandes en SVN ocuparía menos espacio que en Git.
Usabilidad	Git usa la línea de comandos como interfaz de usuario principal. Pero la sintaxis en Git puede abrumar a los principiantes.	SVN utiliza la línea de comandos como interfaz de usuario principal. Es más fácil de usar por parte de los no programadores que desean versionar activos que no son de código.

En lo que respecta al rendimiento de Git frente a SVN, el modelo cliente-servidor de SVN supera el rendimiento con archivos y bases de código más grandes.

Git es mejor para ramificar. Los desarrolladores prefieren Git debido a su eficaz modelo de ramificación.

Dependiendo de sus necesidades, Git o SVN podrían ser una mejor opción. Ambos sistemas adoptan enfoques diferentes cuando se trata de permisos y acceso.

SVN es mejor para almacenar archivos binarios.

SVN a menudo se considera más fácil de aprender. Esto es especialmente cierto para usuarios no técnicos. Son capaces de ponerse al día con las operaciones habituales rápidamente.

## Ejercicios de SVN

Para la actividad usamos el servidor de asamblea.com el cual nos permite crear un repositorio svn

The screenshot shows the assembly.com SVN repository interface for the repository 'tp1.prueba1'. The breadcrumb navigation shows 'dacs2021' > 'tp1' > 'prueba1'. The repository name 'tp1.prueba1' is displayed at the top left. To the right are buttons for 'New Merge Request', 'Lock', 'Add', and 'Checkout'. Below these is a table listing the repository's structure:

Path	Last Commit	Author	Commit Message	Count	Actions
branches	2021-04-15 21:15	[Avatar]	Creando conflictos	4	[Icon]
tags	2021-04-06 05:03	[Avatar]	Automatically created readme.textile and /trunk, /...	1	[Icon]
trunk	2021-04-15 21:15	[Avatar]	Creando conflictos	4	[Icon]
readme.textile	2021-04-06 05:03	[Avatar]	Automatically created readme.textile and /trunk, /...	1	[Icon]

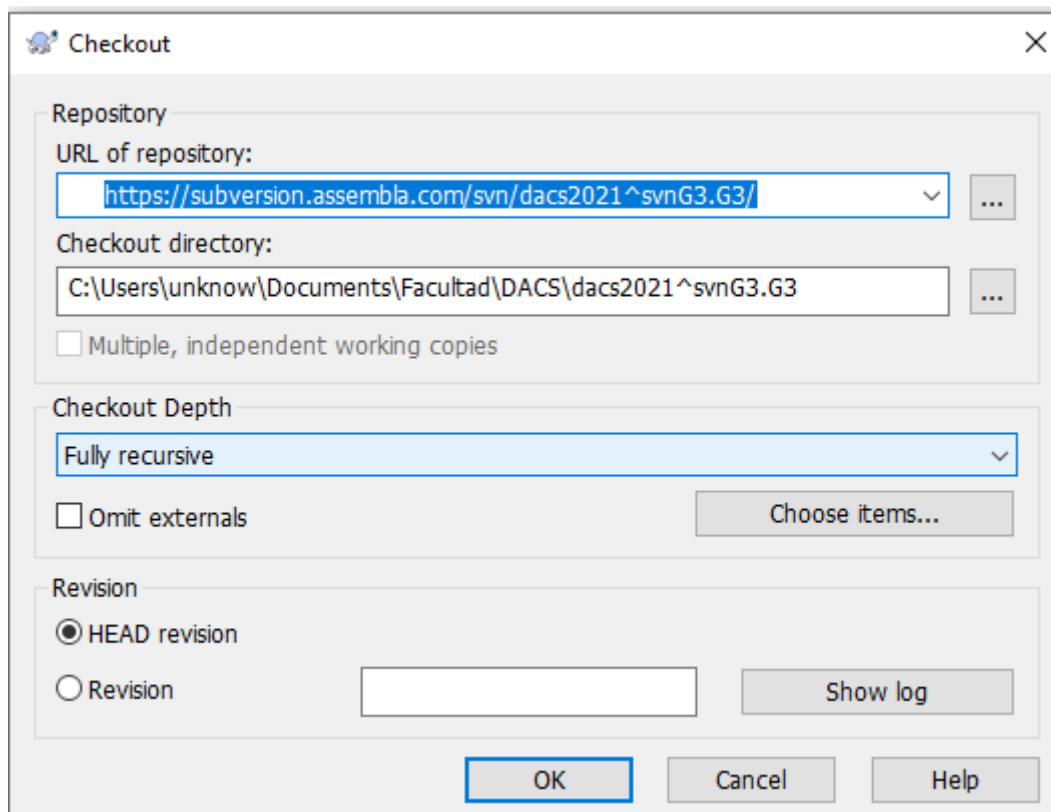
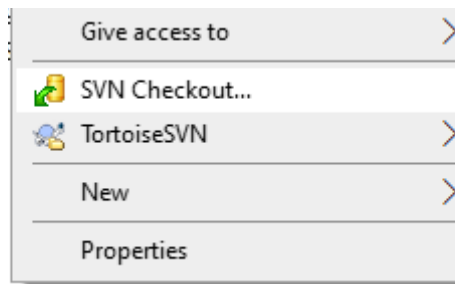
Para clonar el proyecto dentro de nuestro ordenador usamos la herramienta de TortoiseSVN, cuya IDE grafica nos facilita el trabajo.

- Creamos una carpeta donde queremos clonar el repositorio

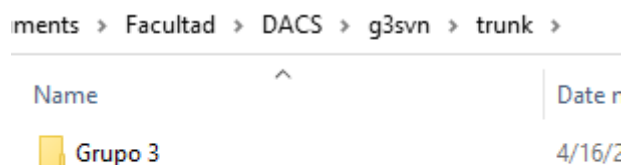
The screenshot shows the assembly.com SVN repository interface for the repository 'svnG3.G3'. The repository name 'svnG3.G3' is displayed at the top left. To the right is a button for 'New Merge Request'. Below this is a table listing the repository's structure:

Path	Last Commit	Author	Commit Message
branches	2021-04-06 05:03	[Avatar]	Automatically created readme.textile and /trunk, /...
tags	2021-04-06 05:03	[Avatar]	Automatically created readme.textile and /trunk, /...
trunk	2021-04-06 05:03	[Avatar]	Automatically created readme.textile and /trunk, /...
readme.textile	2021-04-06 05:03	[Avatar]	Automatically created readme.textile and /trunk, /...

- Usamos la opción de SVN Checkout de Tortoise y usamos la dirección URL que nos provee el servidor de Assable.

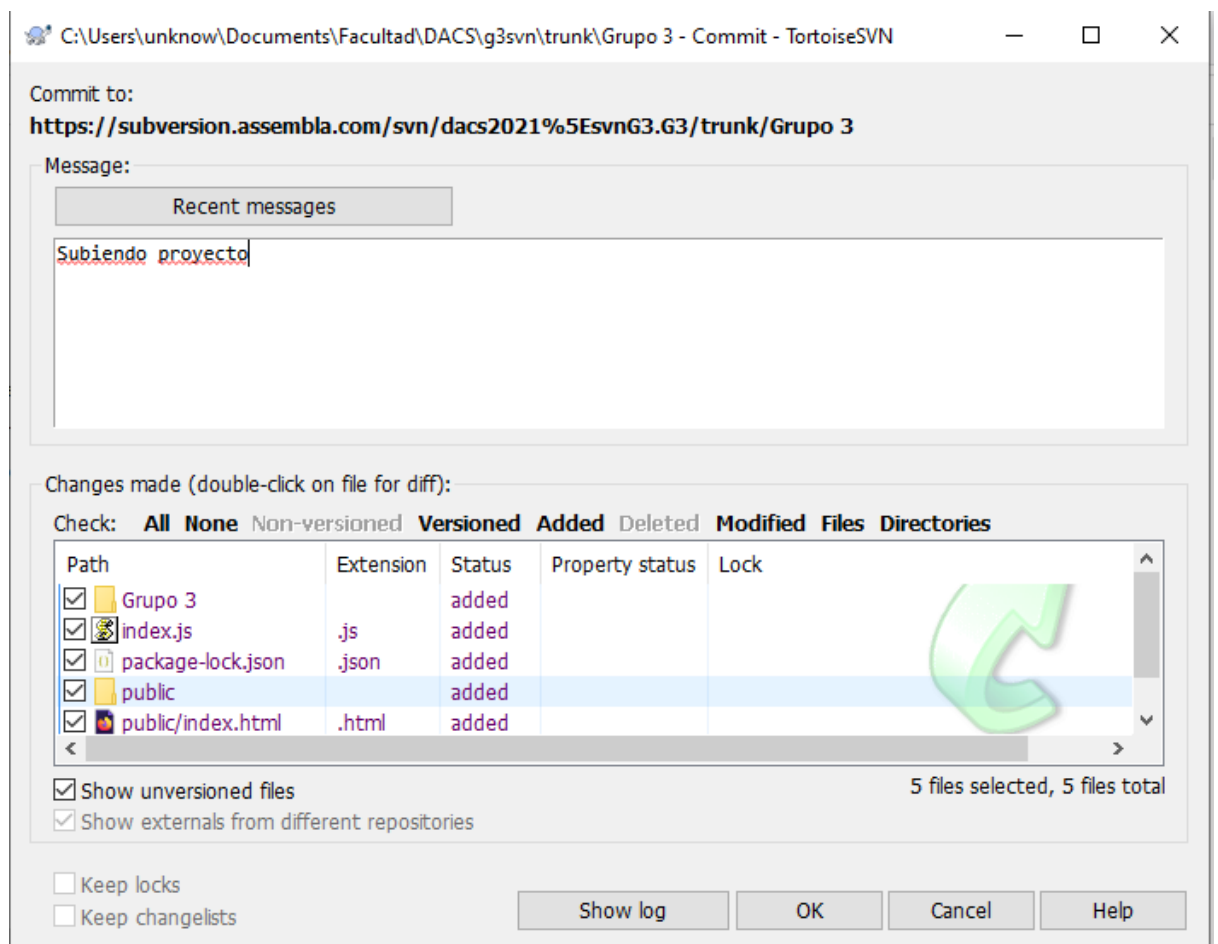
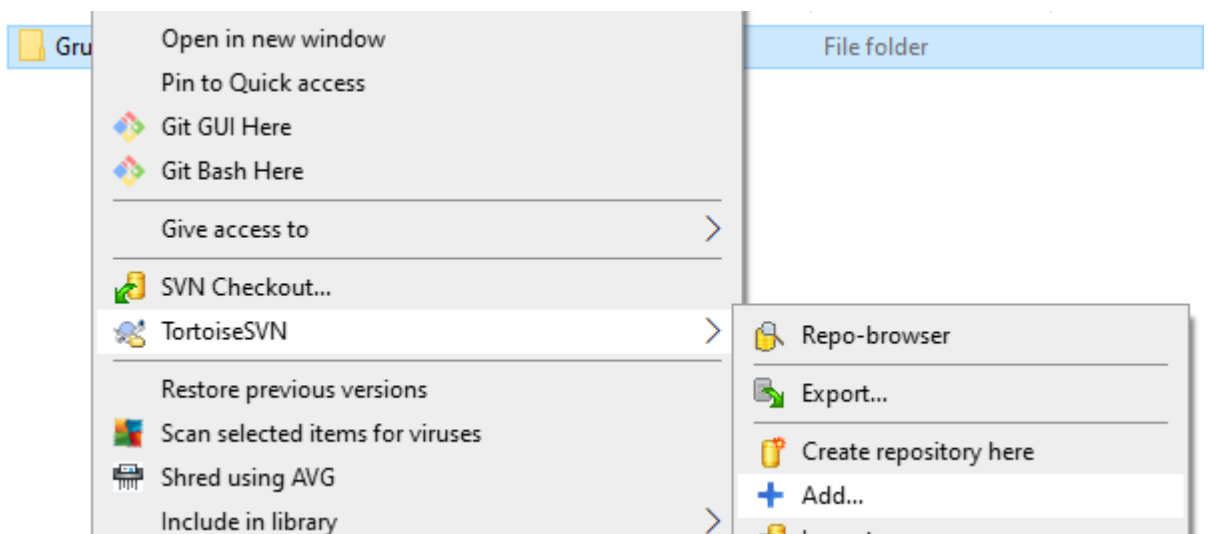


- Una vez clonado el repositorio podemos ubicar nuestro proyecto en la carpeta de trunk el cual va a que ser nuestro hilo primario de trabajo



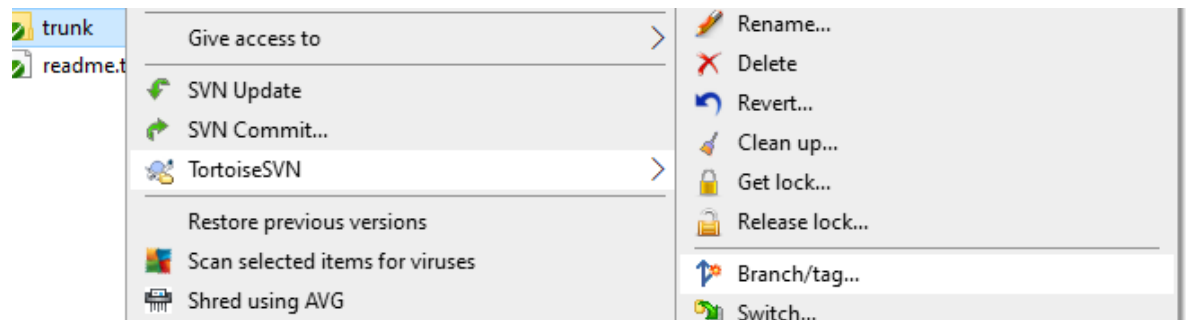
- Parar hacer cambios, los cuales vamos a agregar (add) y poder agregarlos al proyecto en la nube (commit)





como vemos ahora los cambios se ven reflejados en el servidor

- Ahora haremos un branch para trabajar en sobre el.



Repository

From WC / URL:  
https://subversion.assembla.com/svn/dacs2021%5EsvnG3.G3/trunk

To path: /branches/agustin

Destination URL:  
https://subversion.assembla.com/svn/dacs2021%5EsvnG3.G3/branches/agustin

Log message

Recent messages

Create copy in the repository from:

☐ HEAD revision in the repository

☒ Specific revision in repository

☐ Working copy

2

Show Log

Set explicit revision for these externals:

Check: **All** **None**

Path	URL	Fixed at rev
No externals found		

☐ Create intermediate folders

☐ Switch working copy to new branch/tag

OK

Cancel

Help

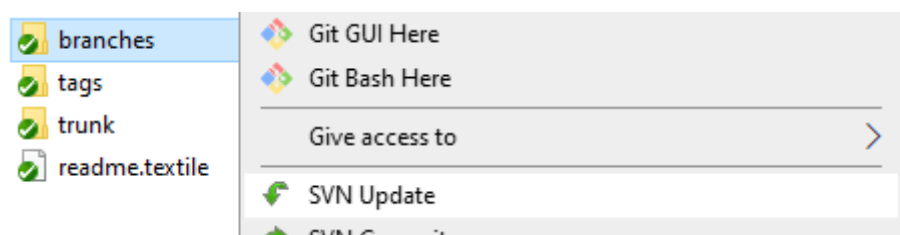


 agustin

2021-04-16 17:51



Una vez hecho esto tenemos que actualizar nuestro repositorio para que se vea reflejado el cambio localmente



para generar un conflicto usaremos el trabajo sobre un mismo archivo en diferentes branches y las intentaremos mergear al trunk

Facultad > DACS > g3svn > branches > mauro > Grupo 3 > public

Name  
 index

# HOLA, mi primer servidor web hecho por mauro

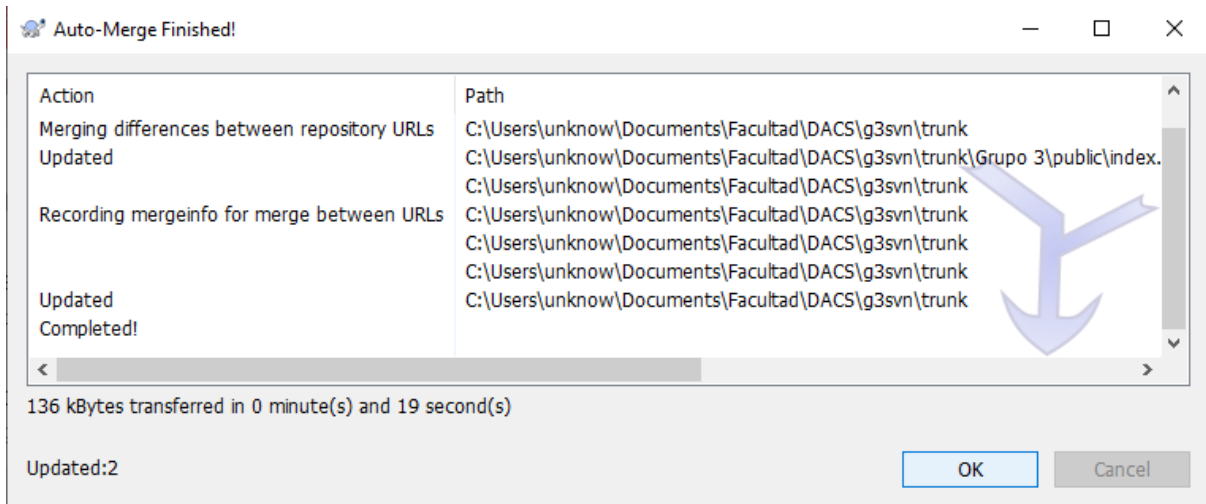
ments > Facultad > DACS > g3svn > branches > agustin > Grupo 3 > public

Name

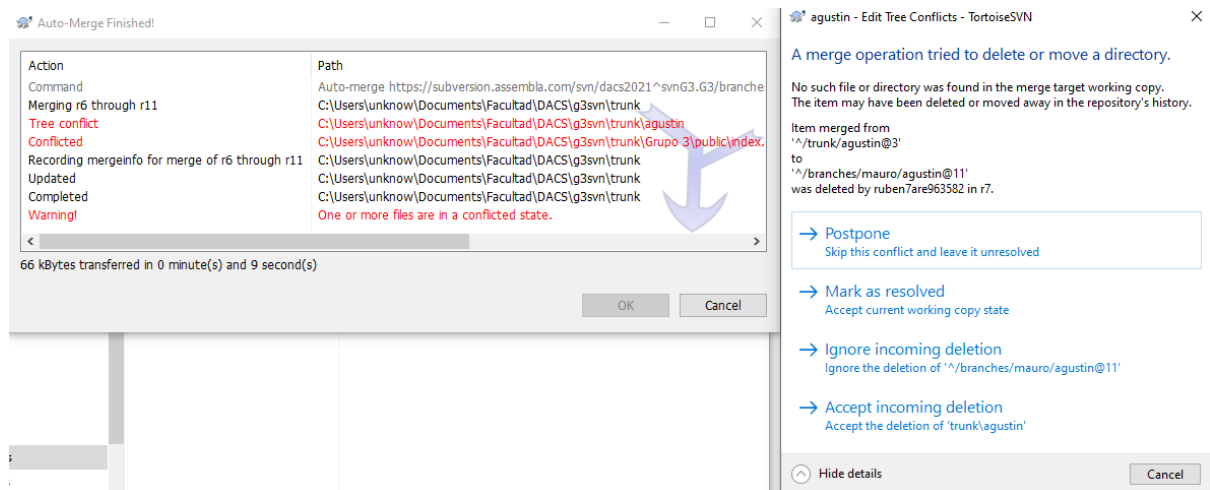
index

# HOLA, mi primer servidor web hecho por Agustin

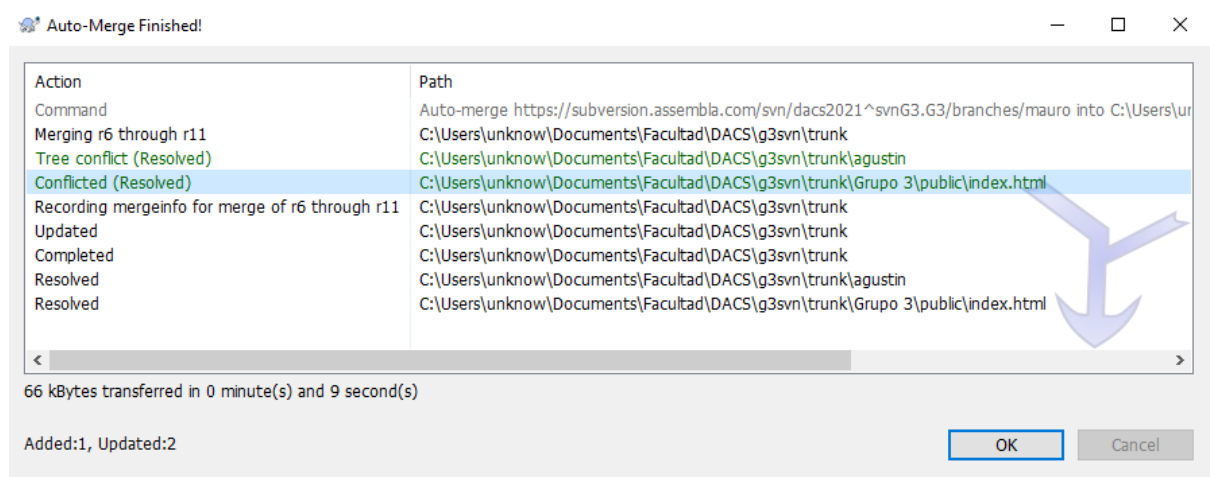
Mergemamos Agustin con trunk



y ahora intentamos mergear Mauro con trunk pero vemos por conflicto en el archivo index.html no nos deja



Solucionamos el conflicto y nos deja concluir el merge



como vemos ahora el trunk contiene los archivos con la edición final



```
1    <!DOCTYPE html>
2    <html lang="es">
3    <head>
4        <meta charset="UTF-8">
5        <meta name="viewport" content="width=device-width, initial-scale=1.0">
6        <title>Mi primer servidor web</title>
7    </head>
8    <body>
9        <h1>HOLA, mi primer servidor web hecho por mauro</h1>
10    </body>
11    </html>
```

## Actividad 3: Actividad práctica sobre Git y Github

Utilizando Git por línea de comandos o desde la Web de Github (según corresponda) realizar el siguiente ejercicio (ir evidenciando documentando los pasos ver nota al final):

1. Un miembro del equipo va a clonar el siguiente [repositorio](#) y va a crear una rama para el grupo (la misma va a tener la forma GX/principal donde X es el número de grupo).
2. En su repositorio local el usuario va a crear una carpeta de grupo (grupoX) y dentro de la misma va crear un proyecto en Node.js. Commitear los cambios en el repositorio y subir la rama al servidor remoto. Les dejo un link de ayuda:
  - a. [Link 1](#)
  - b. [Link 2](#)
3. Una vez creada la rama del grupo en el servidor uno de los miembros del grupo va hacer un fork de la rama. Clona el fork, va a insertar una función que imprime en un label una entrada de pantalla, commit.> push y pull request al repositorio del grupo.
4. Los demás miembros del grupo: Clonar el repositorio y toman la rama del grupo. A partir de la rama del grupo, crean una rama personal (gXiniciales grupo X e 2 iniciales) donde realizar una modificación en código (insertar una función que transforme el formato de un texto, que calcule una suma y la muestre en pantalla, etc) y realizar un commit y push, (Generar un conflicto y resolverlo). Ponerse de acuerdo en el grupo.
5. Realizar un pull request de la rama personal a la principal de grupo.
6. Aceptar / confirmar los pull request en la web, obtener a la funcionalidad completa del programa. Generar un tag para la versión con el nombre gX-V-1.0.0 X número de grupo (por línea de comando) y subir al repositorio remoto.
7. Realizar un cambio en el programa sobre la rama principal del grupo y subir el cambio (que introduce un error al programa).

8. Crear una rama a partir del tag creado y subir la rama al repo remoto y crear un pull request a la rama principal.
9. Aceptar / Confirmar el pull request creado en el paso anterior (corregir el error).

## 1 Clonamos y creamos la rama

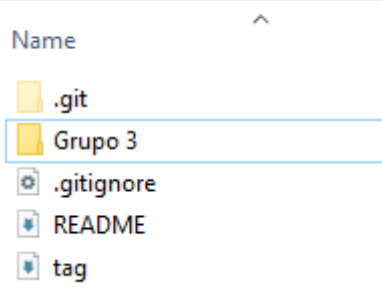
```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git2 (G3/principal)
$ git branch
* G3/principal

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git2 (G3/principal)
$ git clone https://github.com/FRRe-DACS/TP1-GIT-2021.git
Cloning into 'TP1-GIT-2021'...
remote: Enumerating objects: 632, done.
remote: Counting objects: 100% (134/134), done.
remote: Compressing objects: 100% (50/50), done.
Reremote: Total 632 (delta 93), reused 84 (delta 84), pack-reused 498
Receiving objects: 100% (632/632), 10.12 MiB | 3.33 MiB/s, done.
Resolving deltas: 100% (252/252), done.
```

Creamos un proyecto node dentro de la rama del grupo G3

Una vez creada la rama del grupo en el servidor uno de los miembros del grupo va hacer un fork de la rama. Clona el fork, va a insertar una función que imprime en un label una entrada de pantalla, commit.> push y pull request al repositorio del grupo.

nents > Facultad > DACS > git > TP1-GIT-2021 >

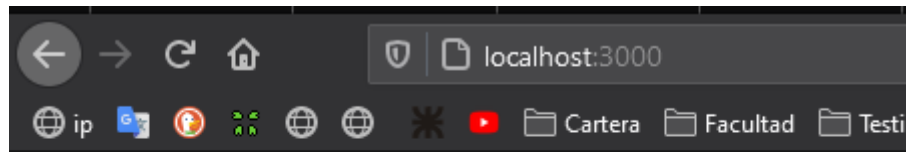


Ejecutamos el servidor

```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021/Grupo 3 (g3AR)
$ node index.js
como te llamasExample app listening at http://localhost:3000
```



Ahora vamos a la dirección en el navegador y vemos al servidor node.js corriendo



# HOLA, mi primer servidor web

Ahora clonamos el repositorio y tomamos la rama del grupo.

A partir de la rama del grupo, creamos una rama personal (gXiniciales grupo X e 2 iniciales)

```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (G3/principal)
$ git checkout -b g3AR
Switched to a new branch 'g3AR'
```

donde realizamos una modificación en código (insertar una función que transforme el formato de un texto, que calcule una suma y la muestre en pantalla, etc)

```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021/Grupo
3 (g3AR)
$ git add .

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021/Grupo
3 (g3AR)
$ git commit -m "prueba agustin"
On branch g3AR
```

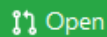
y realizar un commit y push, (Generar un conflicto y resolverlo). Ponerse de acuerdo en el grupo.

Para crear un conflicto trabajaremos en un mismo desde 2 branches disintos e interaemos  
meregear

Date modified	Type	Size
4/16/2021 4:13 PM	Firefox HTML Doc...	1 KB

# HOLA, mi primer servidor web pre conflicto

# G3 ar #52



NoSens29 wants to merge 3 commits into `G3/principal` from `g3AR`



Conversation 0



Commits 3



Checks 0



Files changed 4



NoSens29 commented now

*No description provided.*



NoSens29 added 3 commits 3 hours ago



server node andando



eliminando archivos



pre conflicto

El segunda rama intentaremos hacer un merge con un archivo diferente de la otro rama

ents > Facultad > DACS > git > TP1-GIT-2021 > Grupo 3 > public

Name	Date modified	Type	Size
index	4/16/2021 4:17 PM	Firefox HTML Doc...	1 KB

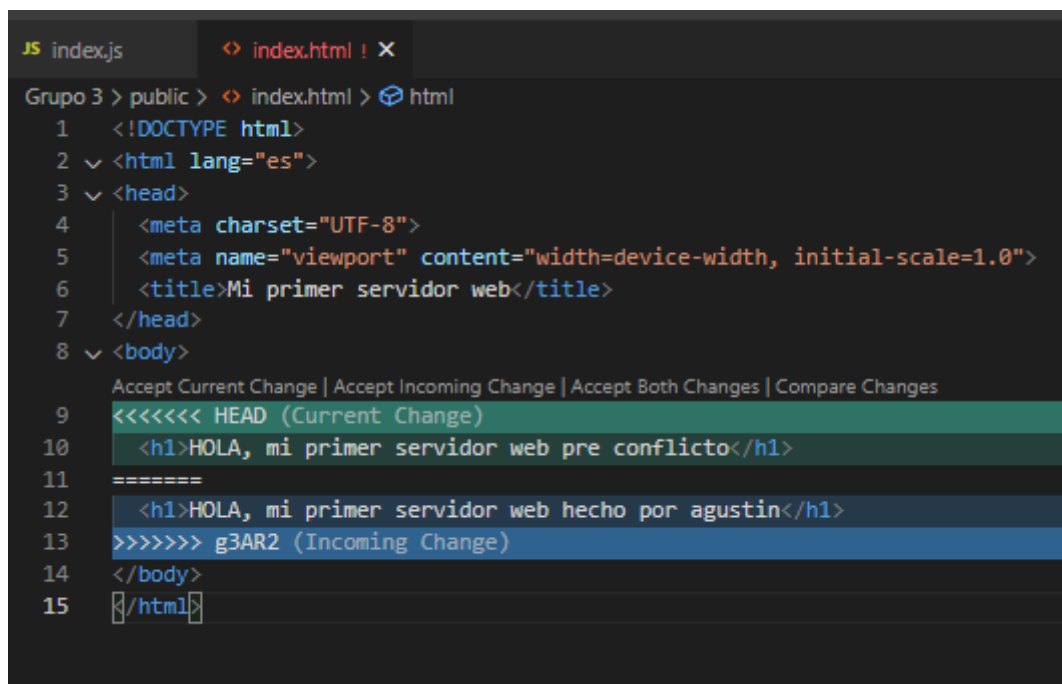
**HOLA, mi primer servidor web hecho por agustin**

```
unknow@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021/Grupo
3 (g3AR)
$ git push -u origin g3AR
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: Create a pull request for 'g3AR' on GitHub by visiting:
remote:   https://github.com/FRRe-DACS/TP1-GIT-2021/pull/new/g3AR
remote:
To https://github.com/FRRe-DACS/TP1-GIT-2021.git
 * [new branch]      g3AR -> g3AR
Branch 'g3AR' set up to track remote branch 'g3AR' from 'origin'.
```

Ahora vemos como no nos permite hacer el merge sin primero solucionar el conflicto

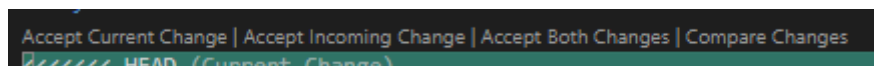
```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3AR)
$ git merge g3AR2
Auto-merging Grupo 3/public/index.html
CONFLICT (content): Merge conflict in Grupo 3/public/index.html
Automatic merge failed; fix conflicts and then commit the result.
```

La forma más práctica de arreglar el conflicto es a través de alguna herramienta IDE, en este caso usaremos Visual Studio



```
JS index.js  <> index.html ! X
Grupo 3 > public > <> index.html > html
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4    <meta charset="UTF-8">
5    <meta name="viewport" content="width=device-width, initial-scale=1.0">
6    <title>Mi primer servidor web</title>
7  </head>
8  <body>
    Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
9  <<<<<< HEAD (Current Change)
10 <h1>HOLA, mi primer servidor web pre conflicto</h1>
11 =====
12 <h1>HOLA, mi primer servidor web hecho por agustin</h1>
13 >>>>>> g3AR2 (Incoming Change)
14 </body>
15 </html>
```

Entre las opciones encontramos las opciones de aceptar el cambio actual, aceptar el cambio que viene de la otra rama, aceptar el cambio de ambas ramas o comprar los cambios



```
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
<<<<<< HEAD (Current Change)
```

Una vez solucionado podremos hacer el merge entre las ramas

```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3AR)
$ git merge g3AR2
Auto-merging Grupo 3/public/index.html
CONFLICT (content): Merge conflict in Grupo 3/public/index.html
Automatic merge failed; fix conflicts and then commit the result.

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3AR|MERGING)
$ git merge g3AR2
error: Merging is not possible because you have unmerged files.
hint: Fix them up in the work tree, and then use 'git add/rm <file>'
hint: as appropriate to mark resolution and make a commit.
fatal: Exiting because of an unresolved conflict.

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3AR|MERGING)
$ git add .

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3AR|MERGING)
$ git commit -m "conflicto solucionado"
[g3AR a50281d] conflicto solucionado

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3AR)
$ git merge g3AR2
Already up to date.
```

5 - Realizar un pull request de la rama personal a la principal de grupo.

base: G3/principal








compare: g3AR

✓ Able to merge. These branches can be automatically merged.

preparando pull g3ar a G3/principal

Write

Preview

H B I  <>     @  

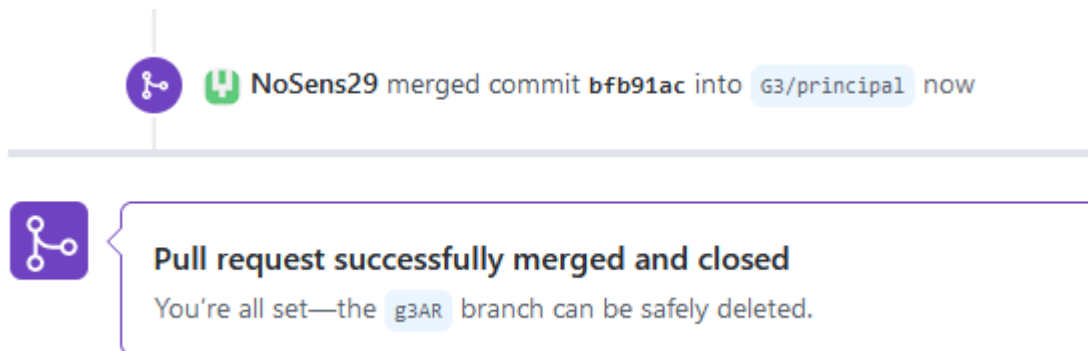
intentando no romper nada

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Add more commits by pushing to the `g3AR` branch on `FRRe-DACS/TP1-GIT-2021`.

6-Aceptar / confirmar los pull request en la web, obtener a la funcionalidad completa del programa.



Generar un tag para la versión con el nombre gX-V-1.0.0 X número de grupo (por línea de comando) y subir al repositorio remoto.

```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021/Grupo 3 (G3/principa
l)
$ git tag -a g3-v-1.0.0 -m 'grupo g3 mergenando'

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021/Grupo 3 (G3/principa
l)
$ git tag -l
g11-V-1.0.0
g2-V-1.0.0
g3-v-1.0.0
g3v0.1
g4-V-1.0.0
g7-V-1.0.0
g8-V-1.0.0
```

```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3AR)
$ git push origin g3-v-1.0.0
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 172 bytes | 43.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/FRRe-DACS/TP1-GIT-2021.git
 * [new tag]          g3-v-1.0.0 -> g3-v-1.0.0
```

g3-v-1.0.0 ...

🕒 5 hours ago 🔑 9365c8d 📦 zip 📦 tar.gz

7- Realizar un cambio en el programa sobre la rama principal del grupo y subir el cambio (que introduce un error al programa).

8 - Crear una rama a partir del tag creado y subir la rama al repo remoto y crear un pull request a la rama principal.

```
unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3/principal)
$ git checkout -b g3/tag g3-v-1.0.1
Switched to a new branch 'g3/tag'

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3/tag)
$ git >> tag.md

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3/tag)
$ git status
On branch g3/tag
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    tag.md

nothing added to commit but untracked files present (use "git add" to track)

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3/tag)
$ git add .
warning: LF will be replaced by CRLF in tag.md.
The file will have its original line endings in your working directory

unknown@DESKTOP-SBKT993 MINGW64 ~/Documents/Facultad/DACS/git/TP1-GIT-2021 (g3/tag)
$ git push -u origin
fatal: The current branch g3/tag has no upstream branch.
To push the current branch and set the remote as upstream, use

    git push --set-upstream origin g3/tag
```