ACTIVIDAD 13. Repositorios

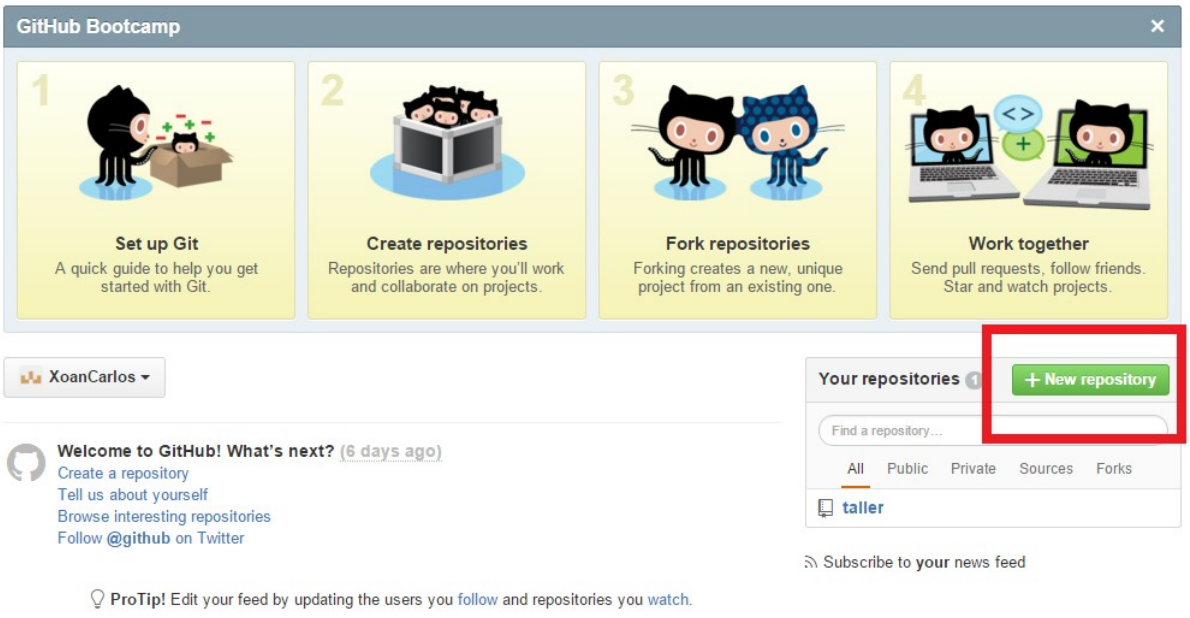
En esta actividad vamos a aprender a manejar los repositorios software de una aplicación que estemos desarrollando. Para eso usarems dos herramientas.

1. Una cuenta en Github.com
2. IDE NetBeans.

Github *(wikipedia)* es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones **Git** que permite un control sobre los cambios realizados en un proyecto software.

El IDE Netbeans permite conectar con Github y añadir en el repositorio allí creado los cambios de software modicado y comentarios realizados sobre el mismo.

Para crear un repositorio simplemente es registrarse en la web de Github y luego *Add Repository.*



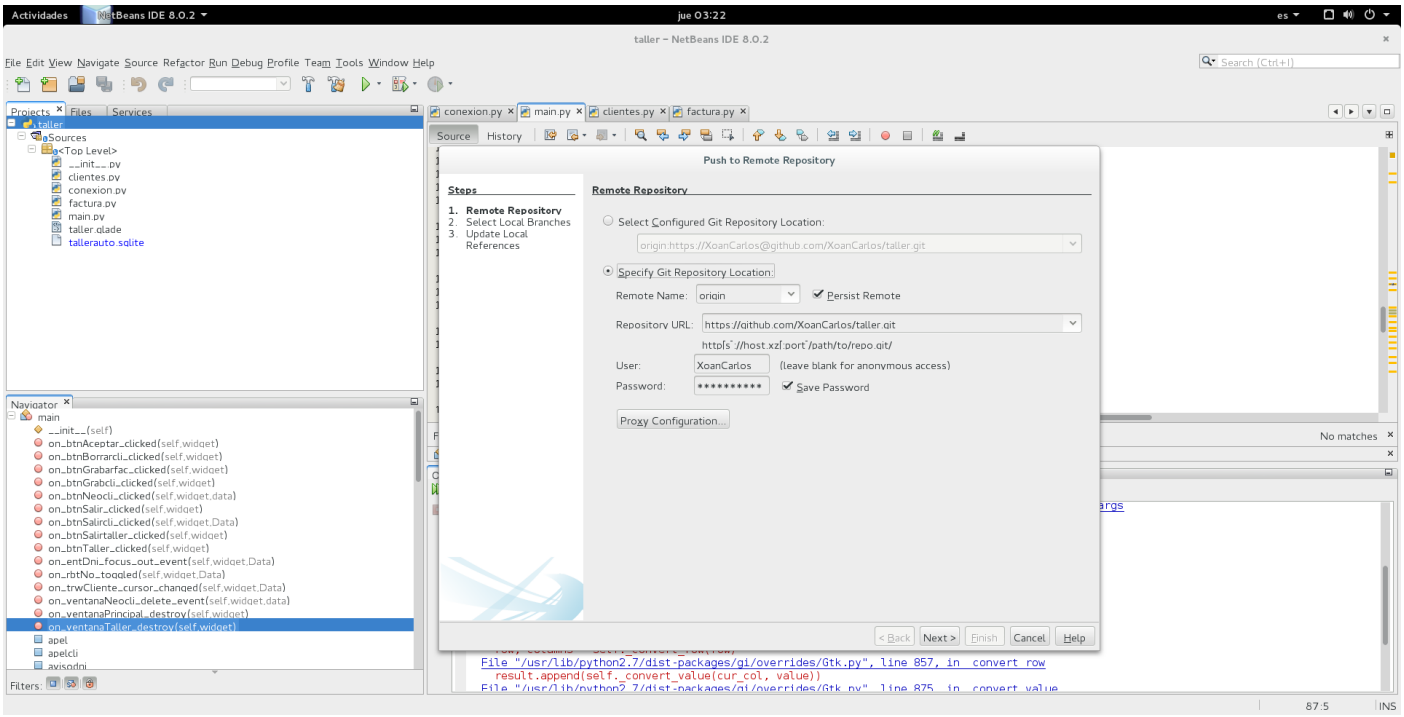
Una vez creado el repositorio tenemos que conectar el IDE con el mismo. Para eso hacemos los siguientes pasos:

1. Botón derechos sobre el proyecto creado
2. Git – > Initiaziling Versioning

Nos pedirá un primer comentario *commit* y hará el *push.*

A partir de este punto cada vez que modifiquemos el proyecto (nosotros o nuestros colaborares), solo tenemos que hacer siempre el mismo proceso

1. Commit *(suelo escribir lo que hice y lo que queda por mejorar)*
2. Push para subirlo



Algunos comandos útiles están en esta web pero insisto con lo que sepamos inicializarlo y subir los cambios es suficiente para este ejemplo:

1. Commit: Guarda una nueva versión de nuestra copia local la cual podemos etiquetar con el cambio realizar.

2. Push: Nos permite subir nuestra copia local tras un commit a nuestro repositorio.

3. Pull: Nos permite obtener cambios de los repositorios del resto de desarrolladores. Para ello basta con decir de que repositorio queremos obtener los datos. Es posible que usando el pull, si tenemos cambios que no aparecen en la versión de la que descargamos, al realizar el *merge* de ambos repositorios surgan conflictos, los cuales deberemos resolver a mano.

4. Add: Cuando creemos un fichero nuevo y queramos subirlo al repositorio, bastara con usar add sobre el archivo que queramos subir.

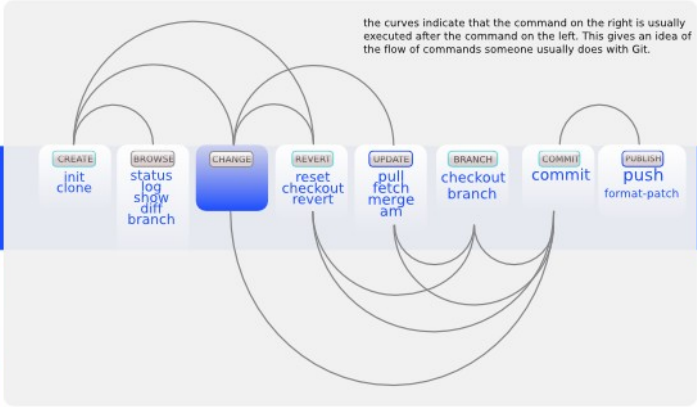
Git Cheat Sheet

http://git.or.cz/

Remember: git command --help

Global Git configuration is stored in $HOME/.gitconfig (git config --help)

Commands Sequence



Create

From existing data

```
cd ~/projects/myproject
git init
git add .
```

From existing repo

```
git clone ~/existing/repo ~/new/repo
git clone git://host.org/project.git
git clone ssh://you@host.org/proj.git
```

Show

Files changed in working directory

```
git status
```

Changes to tracked files

```
git diff
```

What changed between $ID1 and $ID2

```
git diff $id1  $id2
```

History of changes

```
git log
```

History of changes for file with diffs

```
git log -p $file $dir/ec/tory/
```

Who changed what and when in a file

```
git blame $file
```

A commit identified by $ID

```
git show $id
```

A specific file from a specific $ID

```
git show $id:$file
```

All local branches

```
git branch
```

(star '*' marks the current branch)

Cheat Sheet Notation

$id : notation used in this sheet to represent either a commit id, branch or a tag name
$file : arbitrary file name
$branch : arbitrary branch name

Concepts

Git Basics

```
master    : default development branch
origin    : default upstream repository
HEAD      : current branch
HEAD^     : parent of HEAD
HEAD~4    : the great-great grandparent of HEAD
```

Revert

Return to the last committed state

```
git reset --hard
```

⚠ you cannot undo a hard reset

Revert the last commit

```
git revert HEAD
```

Creates a new commit

Revert specific commit

```
git revert $id
```

Creates a new commit

Fix the last commit

```
git commit -a --amend
```

(after editing the broken files)

Checkout the $id version of a file

```
git checkout $id $file
```

Branch

Switch to the $id branch

```
git checkout $id
```

Merge branch1 into branch2

```
git checkout $branch2
git merge branch1
```

Create branch named $branch based on the HEAD

```
git branch $branch
```

Create branch $new_branch based on branch $other and switch to it

```
git checkout -b $new_branch $other
```

Delete branch $branch

```
git branch -d $branch
```

Update

Fetch latest changes from origin

```
git fetch
```

(but this does not merge them).

Pull latest changes from origin

```
git pull
```

(does a fetch followed by a merge)

Apply a patch that some sent you

```
git am -3 patch.mbox
```

(in case of a conflict, resolve and use `git am --resolved`)

Publish

Commit all your local changes

```
git commit -a
```

Prepare a patch for other developers

```
git format-patch origin
```

Push changes to origin

```
git push
```

Mark a version / milestone

```
git tag v1.0
```

Useful Commands

Finding regressions

```
git bisect start      (to start)
git bisect good $id   ($id is the last working version)
git bisect bad $id    ($id is a broken version)

git bisect bad/good   (to mark it as bad or good)
git bisect visualize  (to launch gitk and mark it)
git bisect reset      (once you're done)
```

Check for errors and cleanup repository

```
git fsck
git gc --prune
```

Search working directory for foo()

```
git grep "foo()"
```

Resolve Merge Conflicts

To view the merge conclicts

```
git diff          (complete conflict diff)
git diff --base  $file  (against base file)
git diff --ours  $file  (against your changes)
git diff --theirs $file  (against other changes)
```

To discard conflicting patch

```
git reset --hard
git rebase --skip
```

After resolving conflicts, merge with

```
git add $conflicting_file  (do for all resolved files)
git rebase --continue
```

Juan Carlos Pérez González