Departamento de Desarrollos

Solución propuesta al manejo de repositorios:

Versión 2.0

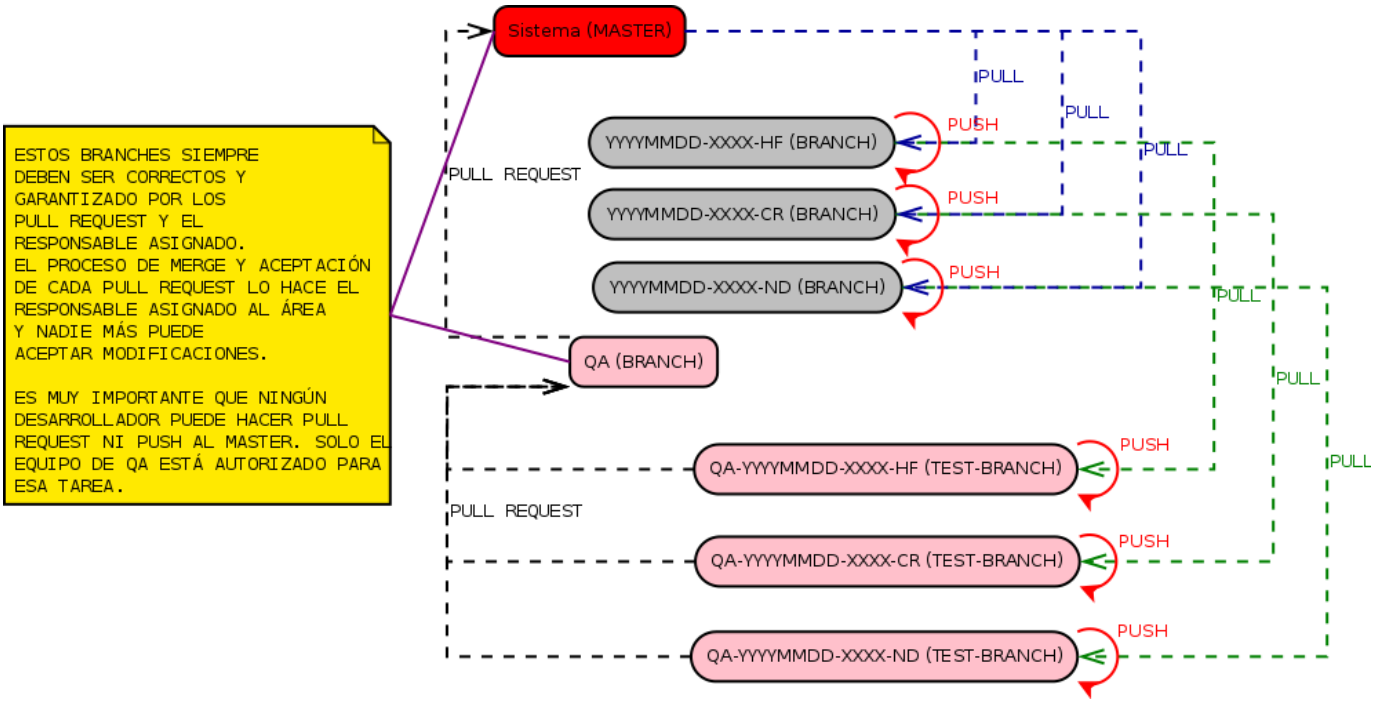
Fecha	Versión	Autor	Descripción
22/09/2017	1.0	Gabriel Ferreira	Creación del documento
26/10/2017	2.0	Gabriel Ferreira	Actualización del documento.

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

1. Descripción breve de la solución planteada

En el presente documento se expondrá una estrategia de uso de GIT en conjunto con la herramienta de gestión de desarrollos, mejor conocidas cómo herramientas ALM (Application Lifecycle Management, o Administración del Ciclo de Vida de las Aplicaciones).

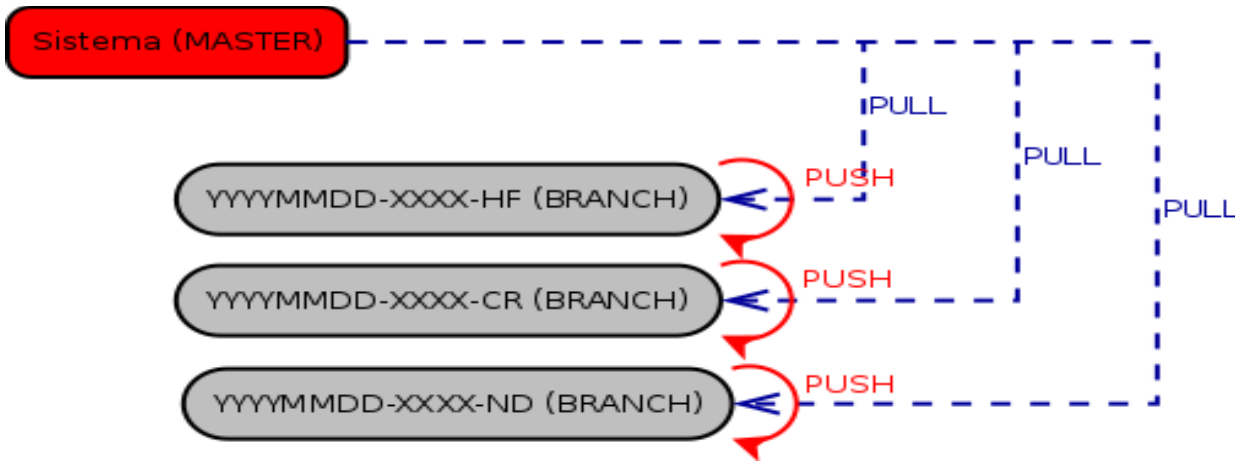
La siguiente estructura de manejo de repositorio es válida para todas las soluciones o productos que requieran un versionado seguro y continuo, en este caso usaremos cómo ejemplo Sistema:



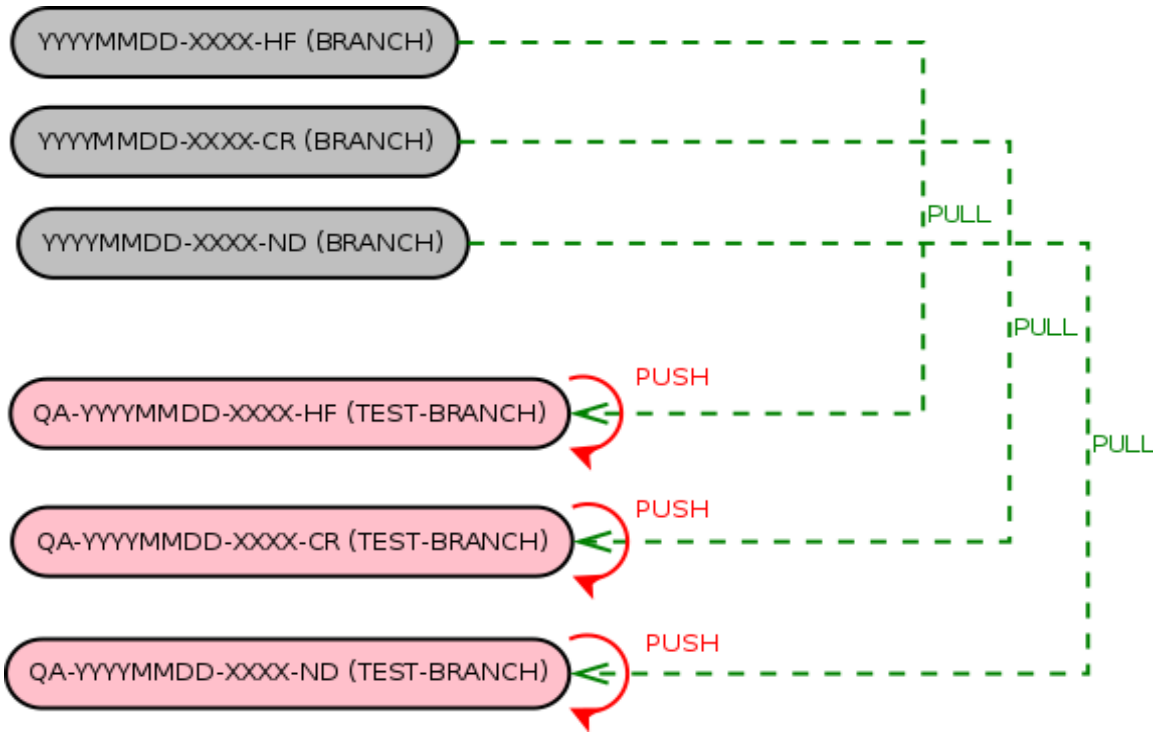
En la figura puede apreciarse cómo se dividen en dos los tipos de recursos que intervienen en el uso de repositorios:

1. primero el sector de desarrollo deben crear un branch correspondiente a su incidente o hotfix (YYYYMMDD-XXXX-HF), nuevo desarrollo (YYYYMMDD-XXXX-ND) o incidente crítico (YYYYMMDD-XXXX-CR). La convención del nombre del branch implica:
    - YYYYMMDD: la fecha del branch.
    - XXXX: es el número o código que identifica a la tarea.
    - CR: para incidentes críticos, ND: para nuevos desarrollos, HF: para demás incidentes.
2. En ese branch o rama subirán sus modificaciones mediante el uso de los comandos add, commit y push con las convenciones de uso descritas en las sección de uso de git. Es importante resaltar que nunca los desarrolladores podrán hacer pull request o push al master. También es imprescindible que el branch se mantenga actualizado desde el master regularmente, principalmente cuando un desarrollo se mantiene activo durante meses. El equipo de QA al momento de corregirlo y hacer pull de este debe además solicitar al desarrollador que lo actualice antes.

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

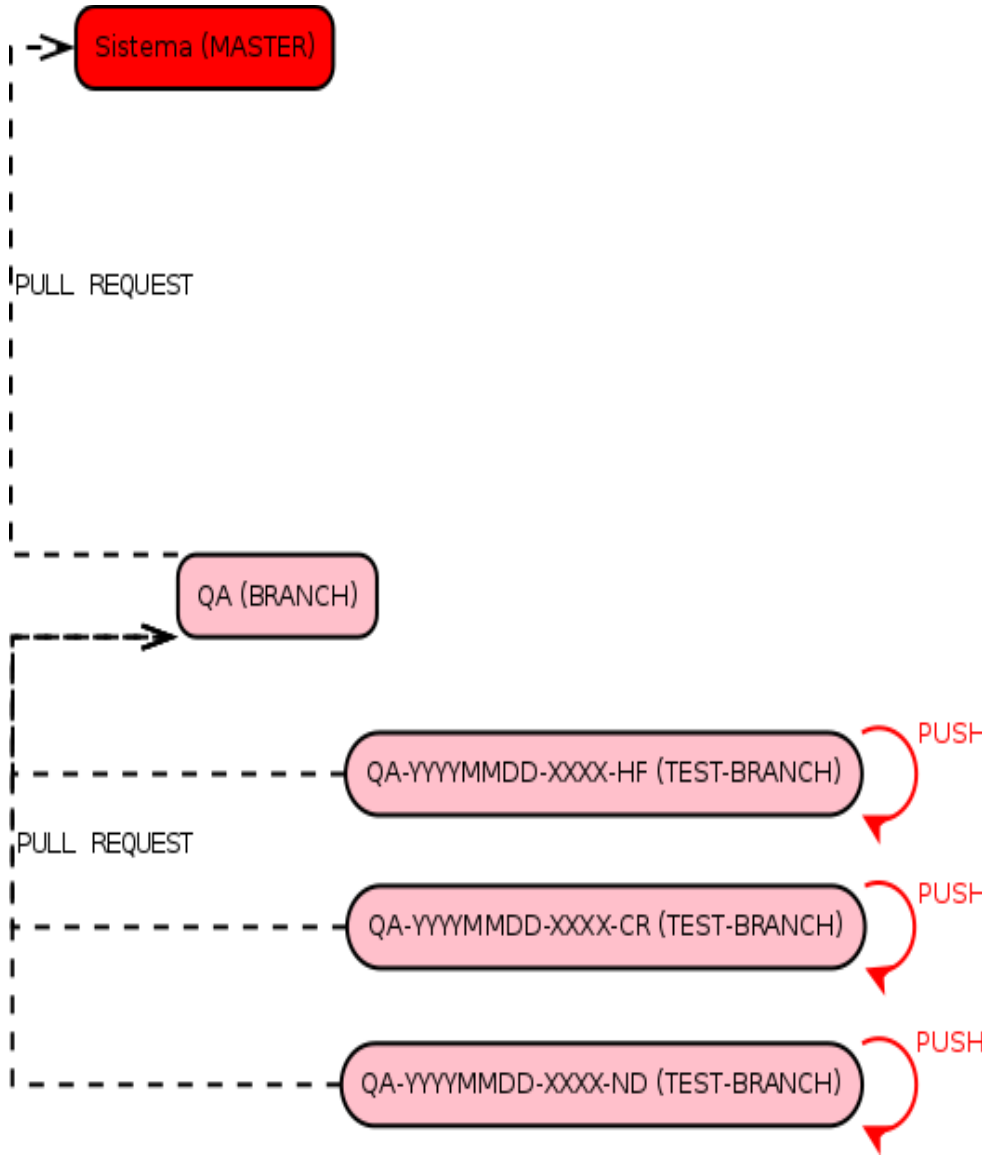


2. Segundo el sector de QA (Quality Assurance, o aseguramiento de la calidad) deben hacer pull de cada branch de desarrollo y trabajar en ese branch de manera separada para aislar y garantizar corrección. La persona responsable de QA antes de traerse un branch puede y debe solicitar al desarrollador que realice una actualización respecto al master, osea git pull origin <nombre_del_branch>, para asegurarse de que cualquier conflicto respecto al master actual se solucione en el área de desarrollo.
3. El mejor mecanismo a implementar para realizar pruebas en los desarrollos actuales para lo cual necesitamos capacitar al personal de QA en el uso de ese lenguaje para tal fin. Para finalizar el proceso deben subir sus cambios al branch y realizar la petición de cambios (PULL REQUEST) mediante la herramienta ALM si existiera.



Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

4. Por último hay una actividad de gran importancia que debe realizar los responsables en el sector de QA, es iniciar y aceptar las peticiones de cambio (PULL REQUEST) y hacer merge si todo es correcto, de lo contrario debe resolver los conflictos con la asistencia, ya sea del desarrolador o del personal de QA en cada caso. Es importante aclarar que el proceso de PULL REQUEST no depende de la herramienta de versionado GIT, sino que es una funcionalidad que algunas herramientas ALM (Application Lifecycle Management) poseen. En caso de no contar con esta funcionalidad se deberán seguir convenciones para respetar la actualización segura del servidor MASTER y/o el de producción.



Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

2. Detalle técnico (Apéndice)

A continuación se especifica técnicamente los comandos de GIT:

Para el desarrollador/a:

Clonar Sistema_Seguros_Interface:

-git clone <url de la solución>

Moverse al directorio del clone descargado:

-cd <directorio de la solución>

Crear y moverse al branch, si es ND por ejemplo:

-git checkout -b YYYYMMDD-XXXX-ND

ó

Crear y moverse al branch, si es HF por ejemplo:

-git checkout -b YYYYMMDD-XXXX-HF

Al terminar de trabajar verificar los cambios antes de agregar archivos modificados:

-git status

Agregar cambios:

-git add <archivos a agregar separados por espacio u * para todos> (cuidado con los archivos que no hay que subir si usás *)

Hacer el commit para una tarea:

-git commit -m "#numero_tarea + mensaje"

Actualizar con lo último que pueda entrar en el master (es importante hacer este paso regularmente mientras se está trabajando en el desarrollo):

-git pull origin master

Al final subir todo al propio branch (usar el nombre del branch, nunca push al master):

-git push origin YYYYMMDD-XXXX-HF

Para la o el responsable de QA:

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

Clonar Sistema_Seguros_Interface:

-git clone <url de la solución>

Moverse al directorio del clone descargado:

-cd <directorio de la solución>

Moverse al branch QA:

-git checkout QA

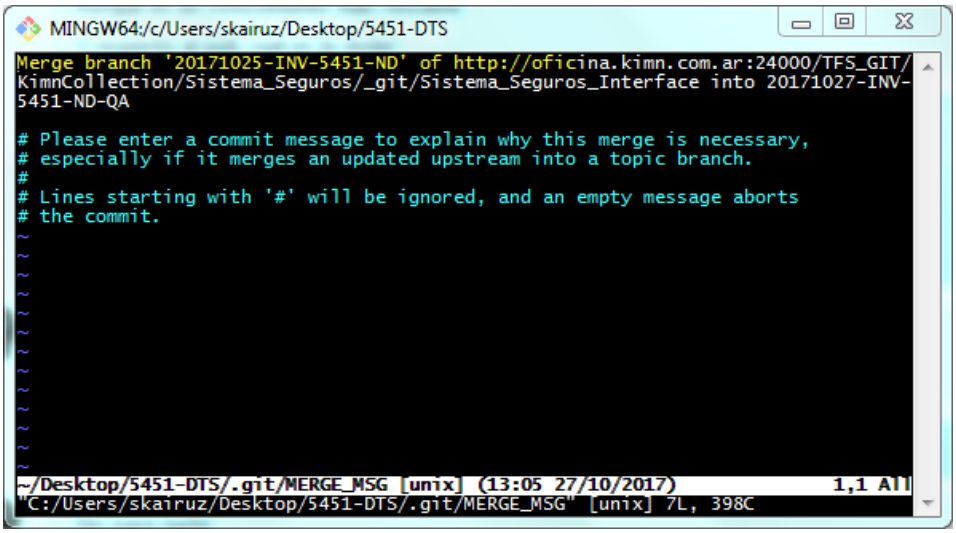
Crear y moverse al branch (usando la fecha actual):

-git checkout -b QA-YYYYMMDD-XXXX-HF

Traer los cambios desde el incidente de desarrollo:

-git pull QA-YYYYMMDD-XXXX-HF YYYYMMDD-XXXX-HF

Al hacer merge automático pueda que suceda que se abra el documento de merge, el mismo se guarda escribiendo :wq y presionando <enter>



Actualizar con lo último que pueda entrar en el master:

-git pull origin master

Al terminar de trabajar verificar los cambios antes de agregar archivos modificados:

-git status

Agregar cambios:

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

-git add <archivos a agregar separados por espacio u * para todos> (cuidado con los archivos que no hay que subir si usás *)

Hacer el commit:

-git commit -m "mensaje"

Actualizar con lo último que pueda entrar en QA:

-git pull origin QA

Al final subir todo al propio branch:

-git push origin QA-YYYYMMDD-XXXX-HF

El proceso Pull Request (en español: Solicitudes de incorporación de Cambios) queda a cargo del método propio de cada herramienta ALM, por ejemplo en Team Foundation Server sería:

-Ir a CODE -> Pull Requests -> New Pull Request, seleccionar el branch

Para aceptar un Pull Request (Esto solo lo hace la o el responsable):

-Ir a CODE -> Pull Requests, seleccionar el Pull Request y si no hay conflictos presionar "Complete Pull Request" y luego "Complete Merge"

Comandos útiles (Usar con cuidado y supervisión):

Manual de Branches:

https://www.atlassian.com/git/tutorials/using-branches

Editar el último mensaje de commit:

git commit --amend -m "YOUR-NEW-COMMIT-MESSAGE"

Contrario a git add:

git reset <file>

Deshacer commit reciente:

git reset --soft HEAD~1

git add -A .

git commit -c ORIG_HEAD

Revertir a commit especifico:

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

```
git revert --no-commit 0766c053..HEAD
git commit
```

Descartar cambios (sin stagear):

```
git stash save --keep-index
```

Descartar cambios (stageados):

```
git reset
```

Forzar push:

```
git push origin master --force
```

Cambiar origin:

```
git remote set-url origin <new-url>
```

Obtener listado de varios commits:

```
git diff --name-only HEAD~10 HEAD~5
```

Borrar un branch local:

```
git branch -d <branch_name>
```

Borrar un branch remoto:

```
git push origin --delete <branch_name>
```

Listar commits de un archivo especifico:

```
git log --follow filename
```

Listar branches remotos:

```
git branch -r
```

VOLVER PARA ATRAS HASH COMMIT:

```
git reset --hard <sha1_of_where_you_want_to_be>
```

Clonar un branch especifico:

```
git clone -b <branch> <remote_repo>
```

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

Descargar una carpeta especifica de un git:

```
git init <repo>
cd <repo>
git remote add origin <url>
git config core.sparsecheckout true
echo "Directorio/Especifico/*" >> .git/info/sparse-checkout
git pull --depth=1 origin master
```

Renombrar Branches:

1)Para renombrar una rama local:

```
git branch -m nombre-rama nombre-rama-nueva
```

2)Para renombrar una rama remota de un repositorio Git,

habrá que eliminar primero la rama remota:

```
git push origin :nombre-rama
```

3)y crear la rama renombrada en el repositorio remoto:

```
git push origin nombre-rama-nueva
```

Corregir conflictos de merge:

Copiar la lista de archivos con conflictos o recuperarla con:

```
git diff --name-only --diff-filter=U
```

- Ir archivo por archivo y corregir:

El archivo marca las partes en conflicto

<<<<<<< HEAD

Código en repositorio origin (BL o nuestro repositorio)

=======

Código en repositorio a mergear (nuestro repositorio o BL)

>>>>>>> {Hash o remote branch}

- Editar para dejar el archivo como debe ir:

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar

Ojo líneas/funciones duplicadas

Funciones que quedaron dentro de otras

- Pruebo que funcione todo bien

- Cuando terminamos:

```
git status
git add {archivo mergeado}
git commit –m "{mensaje de comit}"
```

Descartar cambios del git status

```
git checkout -- .
```

Blame (ver quien toca cuales lineas en un archivo)

```
git blame [opciones] -- archivo
```

Gabriel Nicolás González Ferreira.
Analista de Sistemas (Resolución 5817/03).
gabrielferreira@caraludme.edu.ar