Universidad del Rosario

UNIVERSIDAD NACIONAL DE LA PLATA

SEDICI
REPOSITORIO INSTITUCIONAL DE LA UNLP

PREBI
prebi.unlp.edu.ar

Buenas prácticas para el desarrollo sobre DSpace

Control de cambios

Hoja de ruta

(1)

- Formas de instalación de DSpace
- Introducción a GIT
- Beneficios en el uso de GIT + GITHUB
- Repositorio dspace-crai.git

(2)

- Prácticas recomendadas para cambios
  - quickfix
  - topic-branch+merge local
  - topic-branch+pull-request
- sync con upstream
  - cherry-picks
  - actualizaciones

(3)

- Modificación de Mirage2
- Recomendaciones

Formas de instalación de DSpace

- Binary
  - Sólo para instalaciones básicas
- Source
  - Para instalaciones con cambios en core, hotfixes, etc.
  - Es la opción más conveniente
  - Permite usar versiones en desarrollo (*-SNAPSHOT)
  - 2 caminos:
    i. **ZIP**:
       - Contiene el árbol completo de archivos.
       - Permite cambiar y compilar todos los módulos

    ii. **GIT**
       - Idem anterior, pero además tiene el historial de commits de DSpace
       - Es un repositorio GIT ⇒ permite versionar cambios propios
       - Muchos beneficios!!!

Introducción a GIT y GITHUB

- Es distribuido, con copias locales. Cada copia contiene todo el repo.
- El desarrollo y versionado se hace localmente
- Se comparte de forma central
- Conceptos básicos: commits, branches, tags, remotes, etc.
- Prácticas usuales:
  - branches
    - para resolver problemas específicos, hacer pruebas complejas (migración), etc
    - mantener líneas activas de desarrollo (prox release, master, etc)
  - tags para releases
  - remotes:
    - origin referencia al repositorio propio del cual este fue clonado.
    - upstream referencia al repo del cual deriva este repositorio git

Introducción a GIT y GITHUB

Ver cheatsheet interactivo

http://ndpsoftware.com/git-cheatsheet.html#loc=remote_repo;

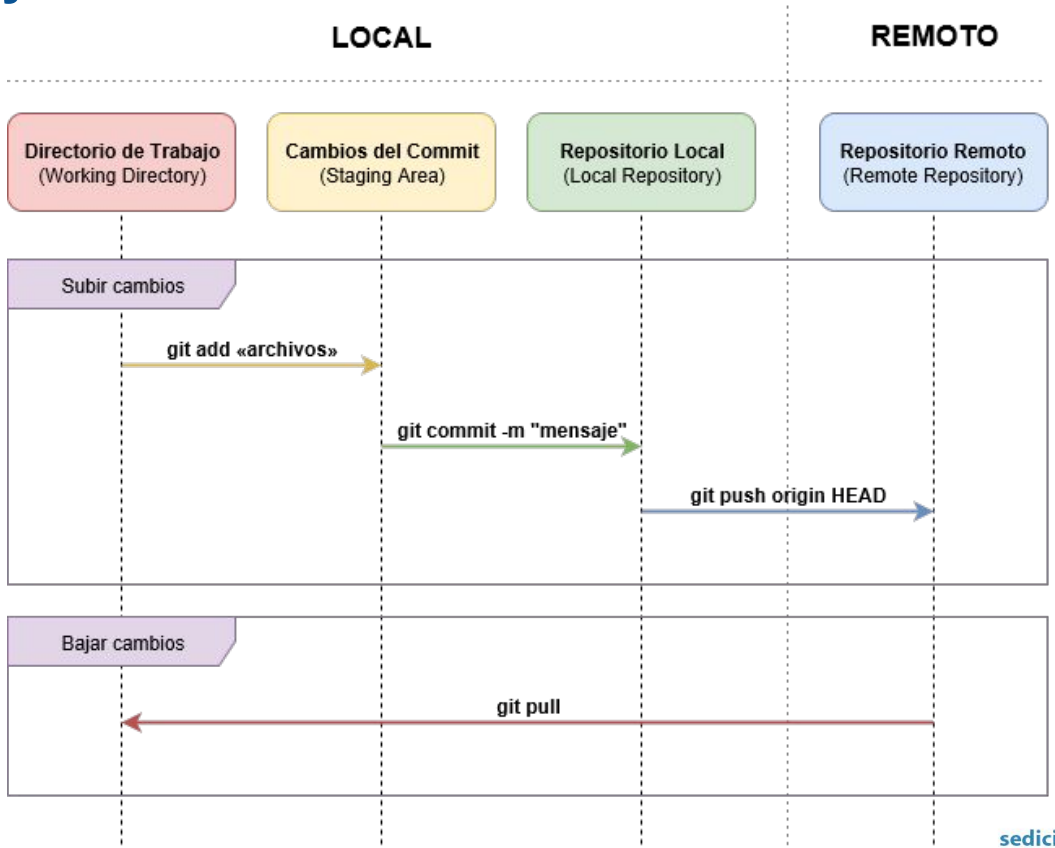


imagen de https://github.com/fedescarpa/example/wiki

Beneficios en el uso de GIT + GITHUB

- Historial de cambios navegable desde la interfaz https://github.com/sedici/dspace-crai/commits/crai-master
- Posibilidad de vincular los cambios (commits) con los sistemas internos de organización y documentación basados en tickets. Ej https://github.com/sedici/dspace-crai/commit/de2f683a9f71231c5ed246e9b8547d7b516ba9ef
- Gestión de usuarios, grupos y permisos de modificación sobre el repositorio propio https://github.com/sedici/dspace-crai/settings/collaboration
- Simplifica el desarrollo de forma colaborativa con la comunidad de desarrolladores de DSpace distribuidos en el mundo, en dos sentidos:
  - hacia el mundo: compartiendo código, proponiendo pull requests a DSpace/DSpace, etc.
  - desde el mundo: tomando código (fixes, features) de otros, actualizando versiones.
- Permite tomar cambios hechos en cualquier otro repositorio basado en DSpace utilizando cherry-picks
- muchos más ...

Repositorio dspace-crai

- Fork vs Mirror
  - Son equivalentes en lo que a git respecta
  - Los forks tiene más opciones desde la interfaz para tomar/dar código.
  - https://github.com/sedici/dspace-crai es un mirror. Para ser fork debería ser público.
- Clone

```
git clone https://github.com/sedici/dspace-crai.git dspace-src
cd dspace-src
git checkout crai-master
```

- Upstream

```
git remote add upstream https://github.com/DSpace/DSpace.git
git fetch upstream # ← trae info de commits de DSpace central
git remote add CICBA_UNLP https://github.com/CICBA/DSpace.git
git fetch CICBA_UNLP # ← trae info de commits de otro repositorio DSpace
```

Prácticas recomendadas en el uso de git

- Commits:
  - hacer “git commit” pequeños y frecuentes en repositorio local
  - usar siempre el email/username para que los cambios sean correctamente atribuidos.
  - evitar agregar archivos automáticamente con “git add .” o “git add --all” .
  - usar “git status” compulsivamente para validar los cambios del repositorio.
  - hacer “git push” cuando se completa o al final del día para tareas largas.
  - evitar “git commit” en el servidor de DSpace
- Branches:
  - evitar el desarrollo directo en la rama principal
  - crear topic branches por cada bugfix o feature:

```
git checkout crai-master
git pull origin/crai-master #sync con el repo central
git checkout -b <feature-branch>
```

- Tags
  - generar tags por cada actualización del servidor para actuar como registro del código y configuraciones efectivamente usados en un momento dado.

Alternativas para incorporar cambios en repositorios git

- Se deberá definir una política interna para incorporar cambios, algo conocido como “git workflow”.
- En esta web se explica detalladamente diferentes workflows desde.
  - centralized workflow: es el más sencillo, útil para desarrollos propios en equipos muy chicos. Es insuficiente para repositorios institucionales que usen software opensource como base.
  - Feature Branch Workflow: se organiza el trabajo por features o tareas. Ver más.
  - y muchos más.

Prácticas recomendadas para hacer cambios

- La propuesta que sigue es una variante del "Feature Branch Workflow"
- El uso de feature/topic branches ofrece muchas ventajas:
  - facilita el trabajo con largos o lentos desarrollo,
  - simplifica las pruebas(testing) manuales y automáticas,
  - permite trabajar en varios features en simultáneo, cada uno en su propia rama
- **Alternativas para hacer los cambios**
  - A) Quickfix
  - B) Topic branch + Merge local
  - C) Topic branch + Pull Request

Cambios: A) Quick fixes

- Cambios triviales con casi nula posibilidad de contener errores.
- 1 o 2 commits con muy pocas líneas modificadas.
- Queda a criterio personal si se debe testear o no el cambio.
  Ejemplo:
  - modificación de una traducción,
  - cambio de un css
  - reemplazo de una imagen
- Problema: propenso a incorporar bugs colaterales, incluso cuando se arregla otro bug.

Cambios: B) Topic Branch + Merge local sobre master

1. Cada bug/feature se desarrolla sobre un branch ad-hoc, sin afectar a la rama principal.
   - `git checkout crai-master`
   - `git pull origin/crai-master #sync con el repo central`
   - `git checkout -b <feature-branch>`
2. Luego se hace un merge con la rama principal localmente
   - `git checkout crai-master #volvemos a la rama principal`
   - `git merge --commit <feature-branch> #realizamos el merge copiando desde el topic-branch`
   - `# si hay conflictos se resuelven manualmente, luego ejecutar “git add file” para cada archivo conflictivo corregido.`
   - `git merge --continue # solo si hay conflictos resueltos`
3. Se suben los cambios
   - `git push origin crai-master`

Cambios: B) Topic Branch + Merge local sobre master (cont)

- Se debería usar sólo para desarrollos que no deben ser validados por otra persona o en equipos de un solo desarrollador.
- SIEMPRE testear los branches antes de llevar (merge) a la rama principal
- El código es revisado sólo por el autor.

Cambios: C) Topic Branch + Pull Request

- Los cambios se realizan en feature-branches que luego son compartidos y sometidos a la revisión de otros miembros del grupo de trabajo, para finalmente aceptar el cambio con o sin correcciones.
- Aplica para equipos de 2 o más desarrolladores
- ventajas:
  - excelentes resultados en lo que respecta a la calidad del código aceptado.
  - transparencia de cambios
  - favorece el trabajo en equipo
  - el merge es central

Cambios: C) Topic Branch + Pull Request

Procedimiento

1. Realizar un feature branch como en el caso anterior

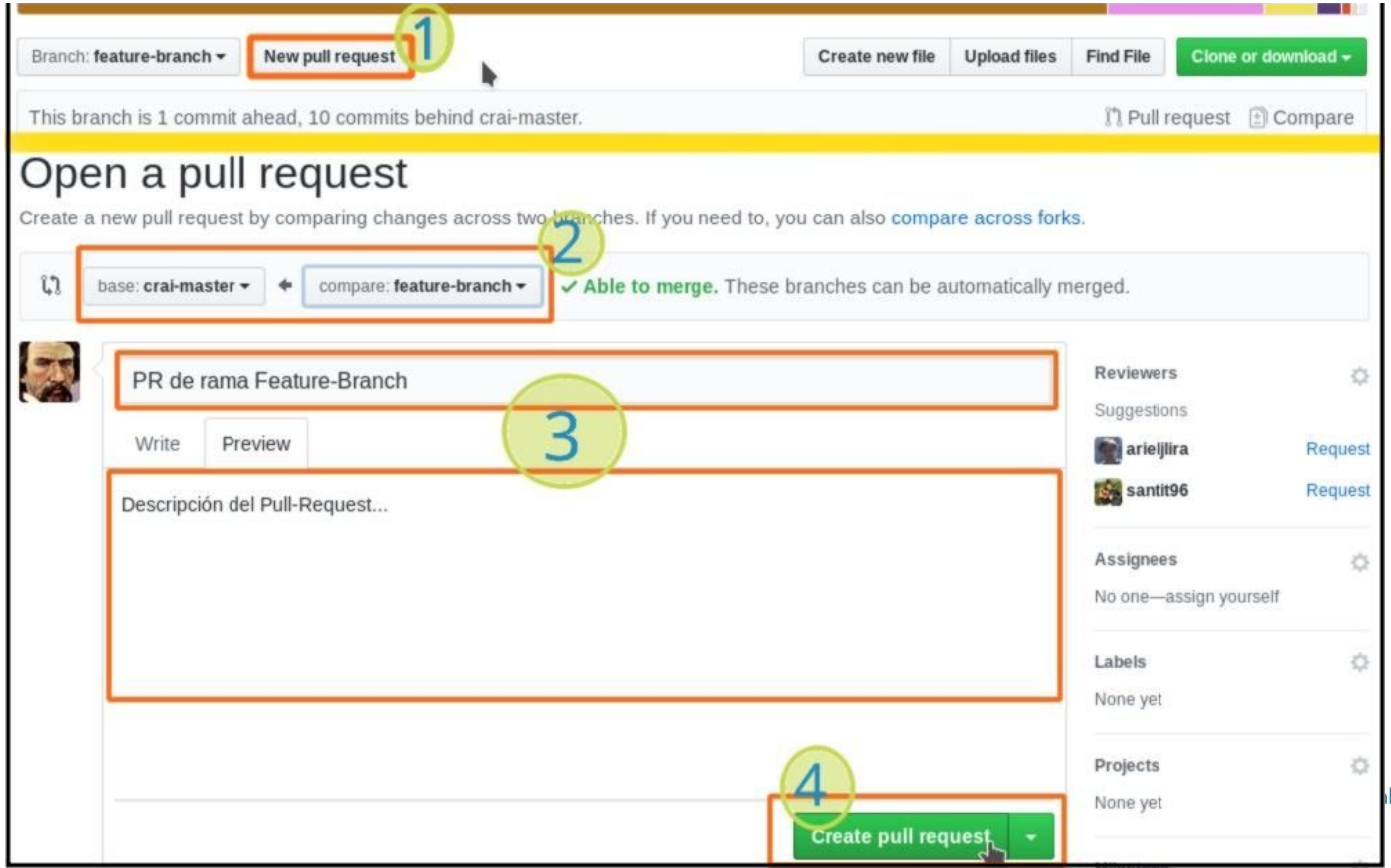
```
1.  git checkout crai-master
2.  git pull origin/crai-master #sync con el repo central
3.  git checkout -b <feature-branch>
```

2. Subirlo a origin con un *push* cuando el desarrollo esté listo

```
1.  git push origin <feature-branch>
```

3. Crear un Pull-Request en el repositorio
4. Los revisores validan el pull request tal y como está, o sugieren cambios.
5. Si es necesario se corrige localmente el branch y se sube nuevamente, es decir, se agregan commits y se hace un push del mismo branch, los cuales automáticamente serán considerados como parte del PR.
6. Se acepta el pull-request hacia la rama principal y se descarta el branch origen.

Cambios: C) (cont) *Creación del Pull Request*



Cambios: C) (cont) *Creación del Pull Request*

- En caso que al tratar de hacer el Pull Request se encuentren conflictos en entre las ramas, el botón de PR estará deshabilitado. Será necesario
  - A) resolver cada conflicto localmente

```
git checkout crai-master
git pull origin crai-master #traemos cualquier cambio de origin
git checkout <feature-branch>
git rebase crai-master  #hacemos el rebase
#Resolvemos los conflictos existentes MANUALMENTE, y hacemos 'git add' de todos los conflictos resueltos
git rebase --continue  # retomamos el rebase una vez resueltos los conflictos
git push --force origin <feature-branch>  #subimos forzadamente la feature-branch
```

  - B) resubir el branch y C) reintentar la creación del pull request.

Cambios: C) (cont) *Revisión y merge del Pull Request*

- Al crear el PR se puede solicitar la revisión de los miembros del equipo. Cualquier miembro puede revisar el código.
- Los revisores hacen revisiones y/o comentarios y pueden aceptar los cambios. Se puede comentar cada línea modificada o el PR en general.
- En caso de desfases entre el PR y crai-master, se deberá hacer un rebase del código de la forma indicada en la diapositiva anterior.
- Por default, cualquiera puede aceptar el PR haciendo clic en “Accept and Merge”
- Finalmente se puede eliminar el PR y el branch asociado.

Sincronizar con 'upstream' y actualizar DSpace

- Recuperación de commits de upstream (no es merge)
  - `git fetch upstream ← NO HACER pull`
- Cherry-pick de un bugfix
  - `git cherry-pick <bugfix-commit-id>`
  - `#TEST local`
  - `git push`
- Actualización a DSpace 6.4 (cuando se publique)
  - A) Actualización local
    - `git checkout crai-master`
    - `git checkout -b merge-con-ds64 # crea una rama para la tarea`
    - `git merge --commit upstream/dspace-6.4  # merge el tag del release`
    - `git status # Revisar cambios, resolver conflictos, etc`
    - `git merge --continue #si ocurrieron conflictos.`
    - `TEST exhasutivo`
    - `git push origin merge-con-ds64`
  - B) Luego push, pull-request, revisión, Accept & merge

Modificación de Mirage 2

Mirage2 es un tema para la interfaz XMLUI, basado en el framework Bootstrap y el uso de Sass para planillas de estilos.

Para compilarlo, se recomiendo previamente instalar localmente **node** (mediante *nvm*) y **ruby** (mediante *rvm*) e instalar dependencias necesarias de Mirage2:

- *bower* (instala dependencias JS)
- *grunt* (realiza tareas repetitivas, como minificar archivos JS)
- *sass* (extiende la sintaxis básica css3 con nuevas directivas)
- *compass* (compila y minifica archivos sass en css3, entre otras cosas)

Modificación de Mirage 2

Para modificar el tema hay que cambiar archivos en el directorio `{dspace-src}/dspace/modules/xmlui-mirage2`. SOLO AQUI

Allí se definen recursos nuevos que sobreescriben los originales de mirage, tanto css, imágenes, xsl como js.

A continuación se listan los principales archivos que controla el HTML generado:

- Vista de un ítem → `item-view.xsl`
- Vista de Comunidad → `artifactbrowser/community-view.xsl`
- Vista de Colección→ `artifactbrowser/collection-view.xsl`
- Listado resultados Discovery → `aspect/discovery/discovery.xsl`
- Home → `core/page-structure.xsl, config/news-xmlui.xml`

Modificación de Mirage 2

A diferencia de otros temas XMLUI, Mirage2 dispone de los siguientes pasos de generación del DRI en XHMTML:

- *Pre-procesamiento* del DRI (`DRI → DRI`)
- *Procesamiento normal* del DRI (`DRI → XHTML`)

Esta etapa aplica **leves transformaciones al DRI** que viene de la cadena de aspectos XMLUI, para finalmente ser procesado por *Mirage2.*

Todos los pre-procesadores están declarados en el archivo preprocess.xsl

Modificación de Mirage 2

Antes de preprocesar → http://dspace/**DRI**/handle/XX/YY

```
<referenceSet id="aspect.artifactbrowser.ItemViewer.referenceSet.collection-viewer" type="summaryView"
n="collection-viewer">
  <reference repositoryID="10336" type="DSpace Item" url="/metadata/handle/10336/14265/mets.xml">
    <referenceSet type="detailList" rend="hierarchy">
      <head>Este ítem aparece en la(s) siguiente(s) colección(ones)</head>
      <reference repositoryID="10336" type="DSpace Collection" url="/metadata/handle/10336/4813/mets.xml" />
    </referenceSet>
  </reference>
</referenceSet>
```

Después de preprocesar → http://dspace/handle/XX/YY**?XML** → preprocess/item-view.xsl

```
<referenceSet id="aspect.artifactbrowser.ItemViewer.referenceSet.collection-viewer" type="summaryView"
n="collection-viewer">
  <reference repositoryID="10336" type="DSpace Item" url="/metadata/handle/10336/14265/mets.xml">
       <referenceSet type="itemPageSummaryList" rend="hierarchy">
       <reference repositoryID="10336" type="DSpace Collection" url="/metadata/handle/10336/4813/mets.xml" />
       </referenceSet>
  </reference>
</referenceSet>
```

Recomendaciones finales

- Sacar archivos customizados de dspace-xmlui-mirage2 y pasarlos a dspace/modules/xmlui-mirage2
- Modificar el servidor y hacer una instalación source + git
- Adoptar el workflow propuesto
- Abrir el acceso al código del repositorio para permitir el uso de un fork de DSpace/DSpace
- Extraer configuraciones propias de dspace.cfg a local.cfg

Universidad del Rosario

UNIVERSIDAD NACIONAL DE LA PLATA

SEDICI
REPOSITORIO INSTITUCIONAL DE LA UNLP

PREBI
prebi.unlp.edu.ar

Muchas gracias!

Ariel Lira, Facundo Adorno
{alira, facundo}@sedici.unlp.edu.ar

Presentado el 12/06/2019 en el marco del convenio de colaboración entre la Universidad del Rosario y la UNLP