SIGAC

HANDBOOK-DEV

TABLA DE CONTENIDO

Push

Recomendaciones

Código Fuente

DinaeWebMaven

DinaeWebMaven es un proyecto Maven de tipo multimodulo.

Estructura de carpetas del proyecto

La estructura de directorios del proyecto es la siguiente:

1. DinaeWebMaven:

   Este proyecto se encarga de agrupar los componentes que hacen parte del proyecto tales como: código fuente, scripts, diagramas, documentación, librerías.

   1.1. DinaeWebMaven-ear

   Proyecto que se encarga de empaquetar el código fuente de los proyectos y se usa como archivo desplegable.

   1.2. DinaeWebMaven-ejb

   Proyecto java tipo EJB Module, la lógica de la aplicación debe ir en este proyecto. Tres de las capas van acá: Entidades, Acceso a datos, Servicios.

   1.3. DinaeWebMaven-web

   Proyecto java tipo Web Application, es el cliente que consume los servicios expuestos por el módulo EJB, solo se debe manejar lógica de cliente. Las otras dos capas se encuentran en este proyecto: Controladores e Interfaz de usuario

   1.4. db

   Carpeta que agrupa los archivos que tienen que ver con la base de datos, como scripts, diagramas, backups, etc...

   1.5. docs

   Carpeta que agrupa los documentos que pueden ser de utilidad al momento de trabajar en el proyecto, tales como: tutoriales, estándares, manuales, fichas técnicas, etc...

   1.6. libs

   Carpeta que agrupa las librerías que son necesarias al momento de configurar el ambiente de desarrollo como los drivers de conexión a bases de datos que por lo general deben ser incluidos en los servidores de

aplicaciones, o para el desarrollo del proyecto, como es el caso de librerías que deban ser instaladas manualmente en el repositorio de maven.

DinaeWebMaven-ejb [API]

Entidades [POJO]

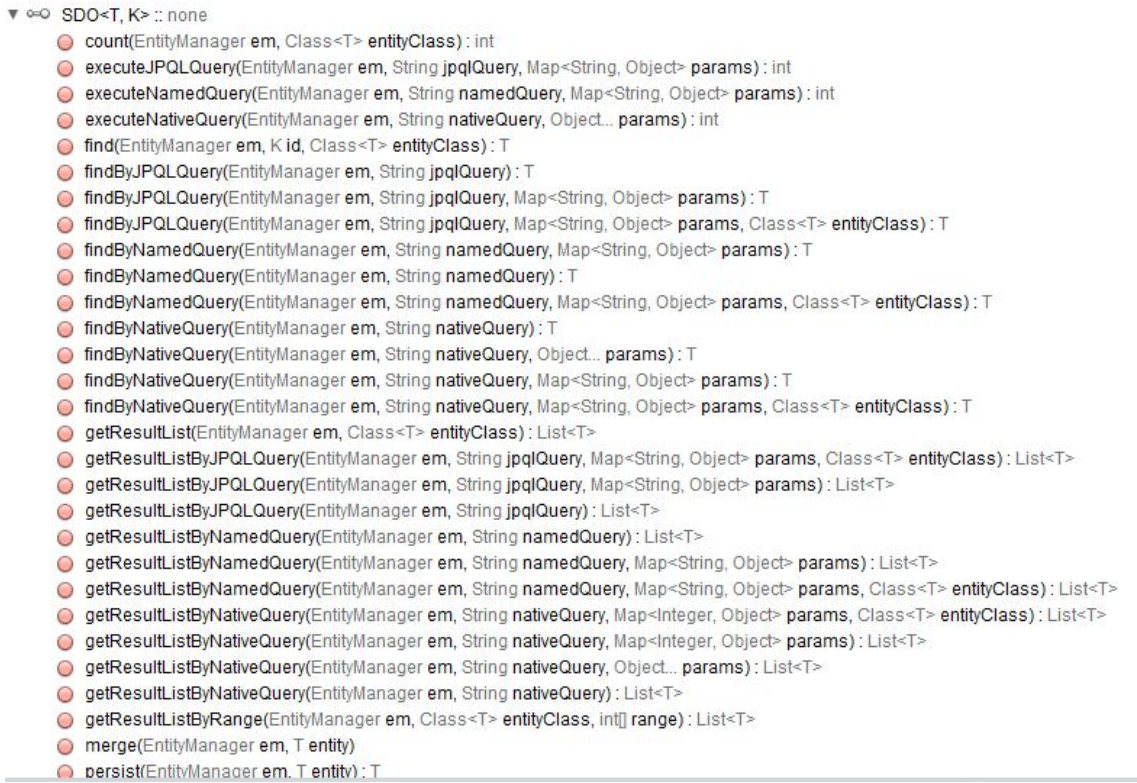
Cosas a tener en cuenta:

- debe implementar Serializable
- los NamedQueries deben ir en constantes públicas, para que puedan ser usadas desde la capa de servicios o SDO.
- si la llave primaria depende de una secuencia, se deben agregar las anotaciones al atributo.
- validar contra la base de datos, que los atributos de la clase correspondan con las columnas de la tabla así como sus tipos de datos, tamaños y obligatoriedad.
- en lo posible tratar de mantener los pojos, lo más sencillo posible.

```
@Entity
@Table(name = "SIEDU_DOMINIO")
@XmlRootElement
@NamedQueries({
  @NamedQuery(name = Dominio.FIND_ALL, query = "SELECT s FROM Dominio s"),
  @NamedQuery(name = Dominio.FIND_BY_ID, query = "SELECT s FROM Dominio s WHERE s.idDominio = :idDominio"),
  @NamedQuery(name = Dominio.FIND_BY_DESCRIPCION, query = "SELECT s FROM Dominio s WHERE s.descripcionDominio = :descripcionDominio")})
@NamedQuery(name = Dominio.FIND_BY_TIPO, query = "SELECT s FROM Dominio s WHERE s.vigente='SI' and s.idTipoDominio.idTipoDominio = :tipoDom")
public class Dominio implements Serializable {

  private static final long serialVersionUID = 1L;
  public static final String FIND_ALL = "Dominio.findAll";
  public static final String FIND_BY_ID = "Dominio.findByIdDominio";
  public static final String FIND_BY_DESCRIPCION = "Dominio.findByDescripcionDominio";
  public static final String FIND_BY_TIPO = "Dominio.VigenteByTipoDominio";
  @Id
  @Basic(optional = false)
  @NotNull
  @Column(name = "ID_DOMINIO")
  @SequenceGenerator(name = "DOMINIO_SEQ_GEN", sequenceName = "SIEDU_DOMINIO_SEQ", allocationSize = 1)
  @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "DOMINIO_SEQ_GEN")
  private Short idDominio;
  @Basic(optional = false)
  @NotNull
  @Size(min = 1, max = 100)
  @Column(name = "DESCRIPCION_DOMINIO")
  private String descripcionDominio;
  @Column(name = "VIGENTE")
  private String vigente;
  @JoinColumn(name = "ID_TIPO_DOMINIO", referencedColumnName = "ID_TIPO_DOMINIO")
  @ManyToOne(optional = false)
  private DominioTipo idTipoDominio;
```

Capa de acceso a datos [SDO]

Para acceder a los datos o manipularlos se cuenta con un API que ofrece métodos que permiten ejecutar sentencias tipo JPQL, NamedQueries, NativeQueries. Este API es genérico, si es necesario se puede extender para casos particulares.



Servicios [EJB]

Esta es la capa de lógica del negocio. Se compone de una Interface y su Implementación. Las reglas de negocio se deben implementar en esta capa, todo lo relacionado con la persistencia de datos se gestiona mediante la capa SDO.

Interface

```java
import co.gov.policia.dinae.ssl.comun.excepciones.ServiceException;
import co.gov.policia.dinae.modelo.Dominio;
import java.util.List;
import java.util.Map;

/**
 * description
 *
 * @author: Diego Poveda <diego.poveda@softstudio.co>
 * @version: 1.0
 * @since: 1.0
 */
public interface DominioService {

  List<Dominio> findAll() throws ServiceException;

  List<Dominio> findByFilter(Map<String, Object> params) throws ServiceException;

  Dominio findById(Long id) throws ServiceException;

  Dominio create(Dominio record) throws ServiceException;

  void update(Dominio record) throws ServiceException;

  void delete(Dominio record) throws ServiceException;

  void delete(Long id) throws ServiceException;
}
```

Implementación

```java
@Stateless
public class DominioJPAService implements DominioService {

  private static final Logger LOG = LoggerFactory.getLogger(DominioJPAService.class);
  @PersistenceContext(unitName = "DinaeWeb-PU")
  private EntityManager em;
  @Inject
  @GenericSDOQualifier
  private SDO sdo;

  @PostConstruct
  public void init() {
    LOG.trace("method: init()");
    if (sdo == null) {
      sdo = new GenericSDO();
    }
  }

  @Override
  public List<Dominio> findAll() throws ServiceException {
    LOG.trace("method: findAll()");
    List<Dominio> list;
    try {
      list = sdo.getResultList(em, Dominio.class);
      em.clear();
    } catch (Exception ex) {
      LOG.error("Error en <<findAll>> ->> mensaje ->> {}", ex.getMessage());
      throw new ServiceException(ex);
    }
    return list;
  }
```

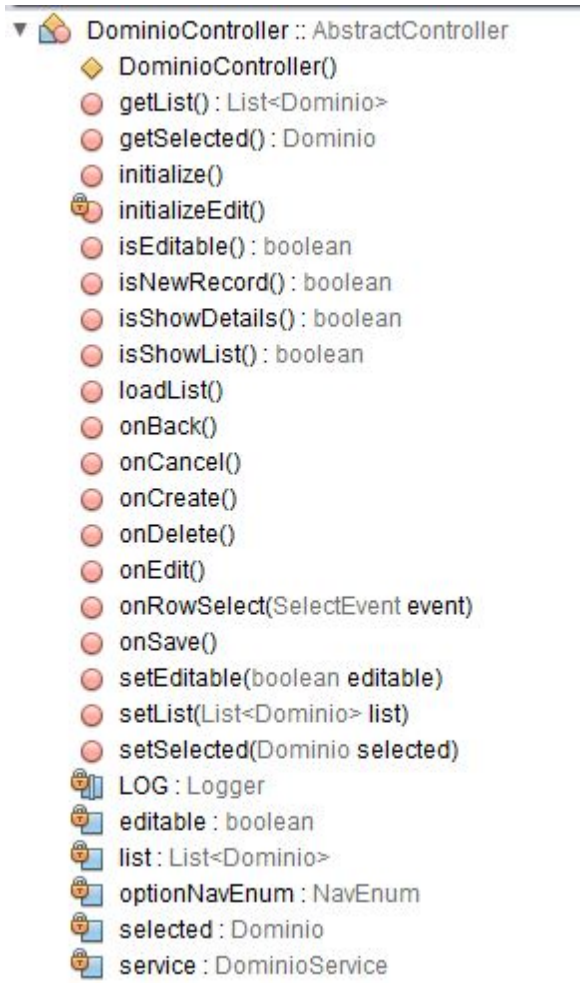
Los métodos siempre deben lanzar una excepción ServiceException. Esta excepción es de tipo ApplicationException y en caso de presentarse causa rollback.

```java
@ApplicationException(rollback = true)
public class ServiceException extends Exception {

  public ServiceException() {
  }

  public ServiceException(String message) {
    super(message);
  }

  public ServiceException(Throwable cause) {
    super(cause);
  }

  public ServiceException(String message, Throwable cause) {
    super(message, cause);
  }
}
```

DinaeWebMaven-web [Cliente JSF]

Controladores [JSF - Backing Bean]

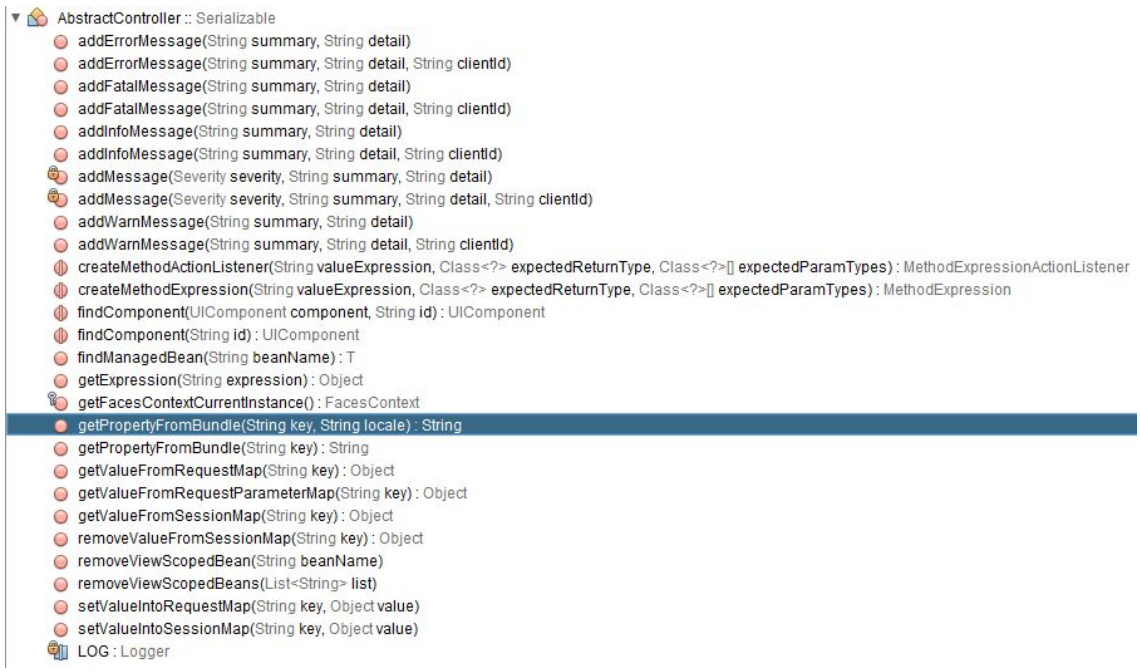
Es la clase encargada de recibir las peticiones del usuario y direccionarlas, así mismo se encarga de entregar las respuestas al usuario.



Los controladores deben extender de la clase AbstractController.

```java
@Named
@SessionScoped
public class DominioController extends AbstractController {

  private static final Logger LOG = LoggerFactory.getLogger(DominioController.class);

  @EJB
  private DominioService service;
  private Dominio selected;
  private List<Dominio> list;
  private boolean editable;
  private NavEnum optionNavEnum;

  public DominioController() {
    LOG.trace("method: constructor()");
  }

  @PostConstruct
  public void initialize() {
    LOG.trace("method: initialize()");
    optionNavEnum = NavEnum.LIST;
    setSelected(null);
    setEditable(false);
    loadList();
  }

  /**
   *
   */
  public void loadList() {
    LOG.trace("method: loadList()");
```

Esto le garantiza acceder a una variedad de métodos:



AppController

AppController es un controlador con alcance de aplicación, en este controlador se deben poner los métodos que son comunes a más de una funcionalidad. Ejemplo:

- obtener una lista de dominios en particular
- obtener la lista de Unidades de la Policía
- obtener la lista de Escuelas de la Policía
- etc...

Convertidores

Existe un convertidor genérico abstracto AbstractXMLConverter<T>

del cual deben extender los convertidores para un tipo de objeto en particular

```
@FacesConverter("dominioConverter")
public class DominioConverter extends AbstractXMLConverter {

  @Override
  public Class getClazz() {
    return Dominio.class;
  }
}
```

Validadores

Interface de Usuario [JSF - xhtml]

Se está usando primefaces 5.3 como librería de componentes de interfaz de usuario [http://www.primefaces.org/showcase/].

Para los desarrollos nuevos, la página se esta estructurando usando Responsive Desing de primefaces [http://www.primefaces.org/showcase/ui/misc/responsive.xhtml]

Links de consulta.

Responsive Desing: http://www.w3schools.com/html/html_responsive.asp

Material Desing [Especificación]: https://www.google.com/design/spec/material-design/introduction.html

Material Desing Lite [Implementación]: https://www.getmdl.io/

```
<!DOCTYPE html>
<ui:composition
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:ui="http://java.sun.com/jsf/facelets"
  xmlns:h="http://java.sun.com/jsf/html"
  xmlns:f="http://java.sun.com/jsf/core"
  xmlns:p="http://primefaces.org/ui"
  template="/resources/default/template/main.xhtml">
  <ui:define
    name="content">
    <p:panel
      id="main"
      styleClass="ui-fluid">
      <h:form
        id="frmList"
        prependId="false"
        rendered="#{dominioController.showList}">
        <h:panelGroup>
          <h:outputText
            value="#{bundle['title.dominios']}"
            styleClass="toolbar-title"/>
          <p:commandButton
            id="btnCreate"
            title="#{bundle['commons.btn.new']}"
            icon="ui-icon-document"
            actionListener="#{dominioController.onCreate()}"
            update=":main"/>
          <p:commandButton
            title="#{bundle['commons.btn.refresh']}"
            icon="ui-icon-refresh"
            actionListener="#{dominioController.loadList()}"
            disabled="#{dominioController.editable}"
            update="dtList"/>
```

Como se observa en la imagen anterior, los textos que se muestran al usuario se encuentran internacionalizados, es decir que se usa un archivo de propiedades por cada idioma que se desee manejar en la aplicación.

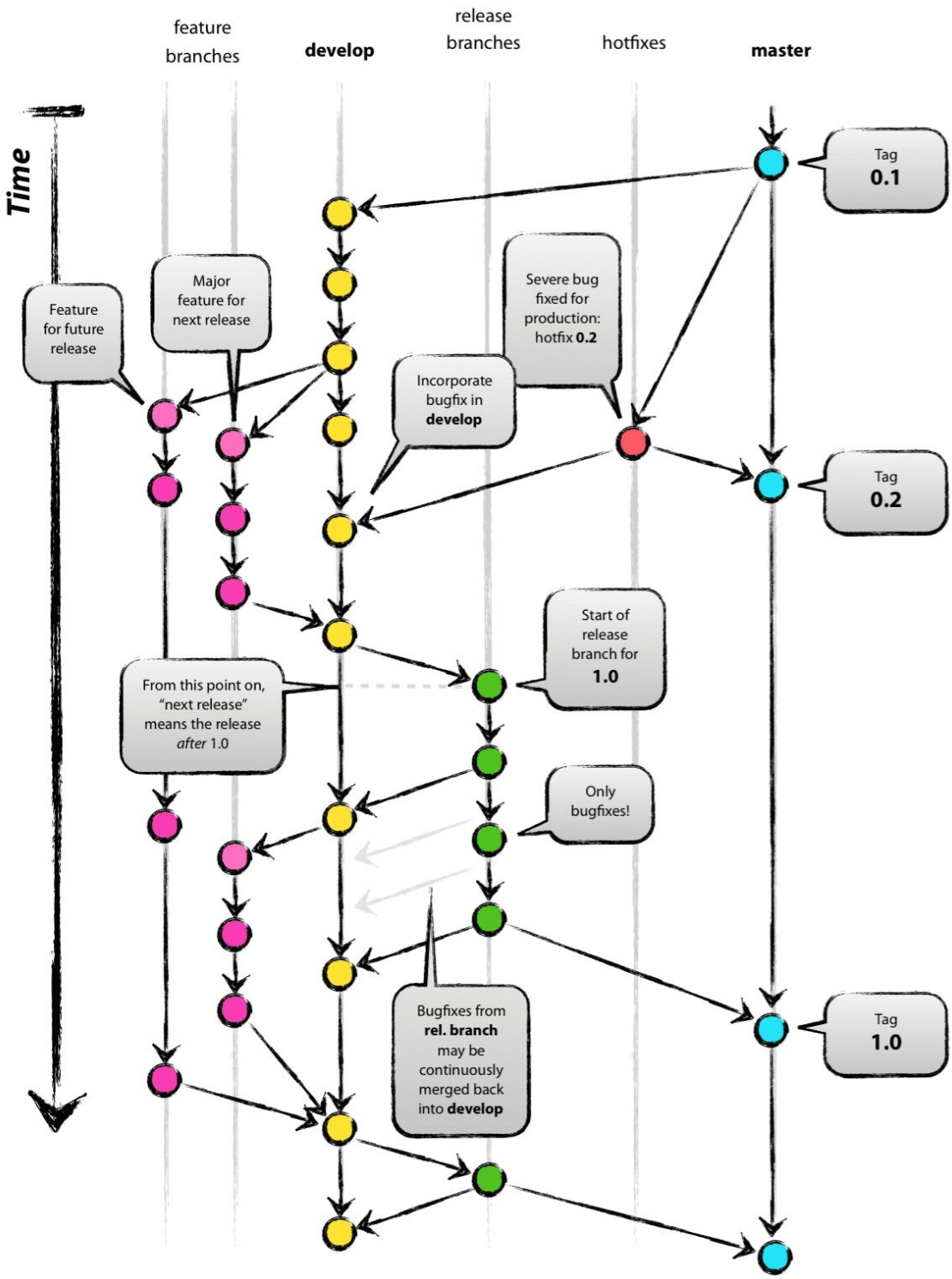
```
#############################################################################################
# patterns
#############################################################################################
pattern.default.date=yyyy-MM-dd
pattern.default.datetime=yyyy-MM-dd HH:mm:ss
pattern.default.thousand=##,###,###,###
#############################################################################################
# mask
#############################################################################################
mask.default.date=__/__/____
#############################################################################################
# commons
#############################################################################################
commons.btn.refresh=Refrescar
commons.btn.new=Crear
commons.btn.add=Agregar
commons.btn.edit=Editar
commons.btn.remove=Eliminar
commons.btn.print=Imprimir
commons.btn.save=Guardar
commons.btn.cancel=Cancelar
commons.btn.ok=Ok
commons.btn.back=Volver
commons.btn.login=Ingresar
commons.btn.logout=Salir
commons.btn.download=Descargar archivo
commons.btn.upload=Subir archivo
commons.dt.emptymessage=No se encontraron registros
commons.cfmdlg.header=Alerta - Confirmar operación
commons.cfmdlg.remove.message=Desea eliminar el registro?
commons.cfmdlg.btn.yes=Si
commons.cfmdlg.btn.no=No
commons.msg.success=Exitoso
```

Repositorio

Como sistema de control de versiones se va a usar GIT. la guia de referencia y todo lo relacionado, se puede consultar en el siguiente link [https://git-scm.com/].

Modelo de ramificación

Se van a trabajar tes niveles de ramas:



[http://nvie.com/posts/a-successful-git-branching-model/]

master

es la rama principal, es la base para el desarrollo, esta rama siempre se debe actualizar con la versión liberada estable.

tags

https://confluence.atlassian.com/bitbucket/use-repo-tags-321860179.html

develop

esta rama se desprende de la master, y es donde se van liberando para pruebas los desarrollos que van concluyendo.

feature/topic

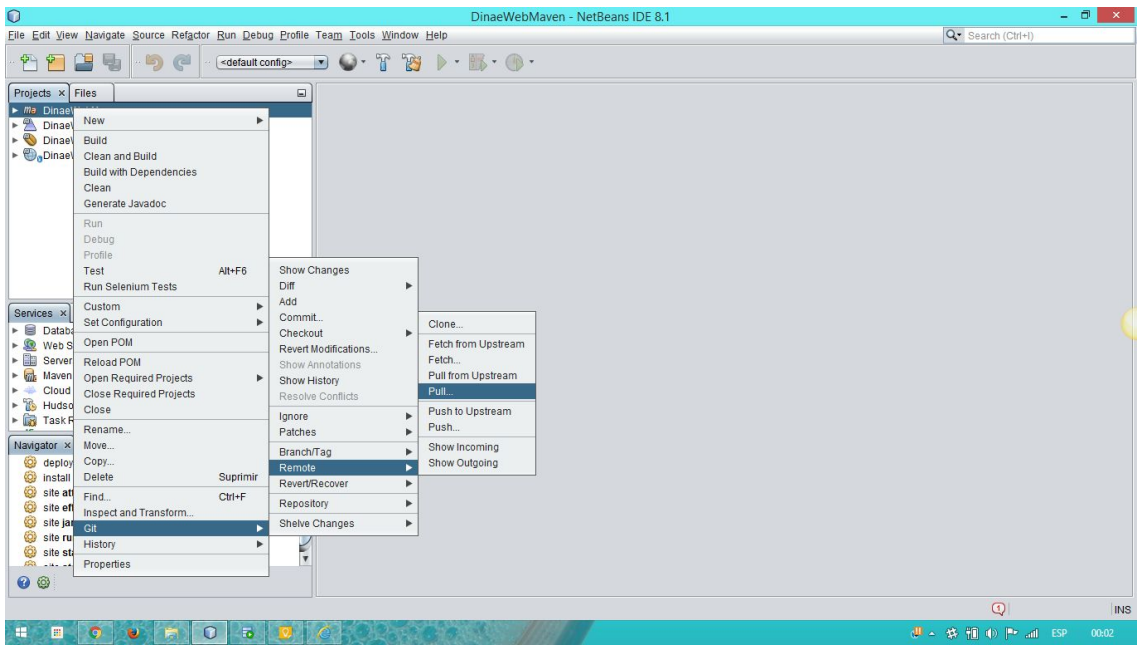
esta no es una rama en particular, se debe crear una rama que se desprende de develop por cada uno de los casos de uso o funcionalidades a desarrollar, la idea es, que hasta que no se haya concluido todo el desarrollo de esa funcionalidad, no se integre con la versión estable.

Desarrollo

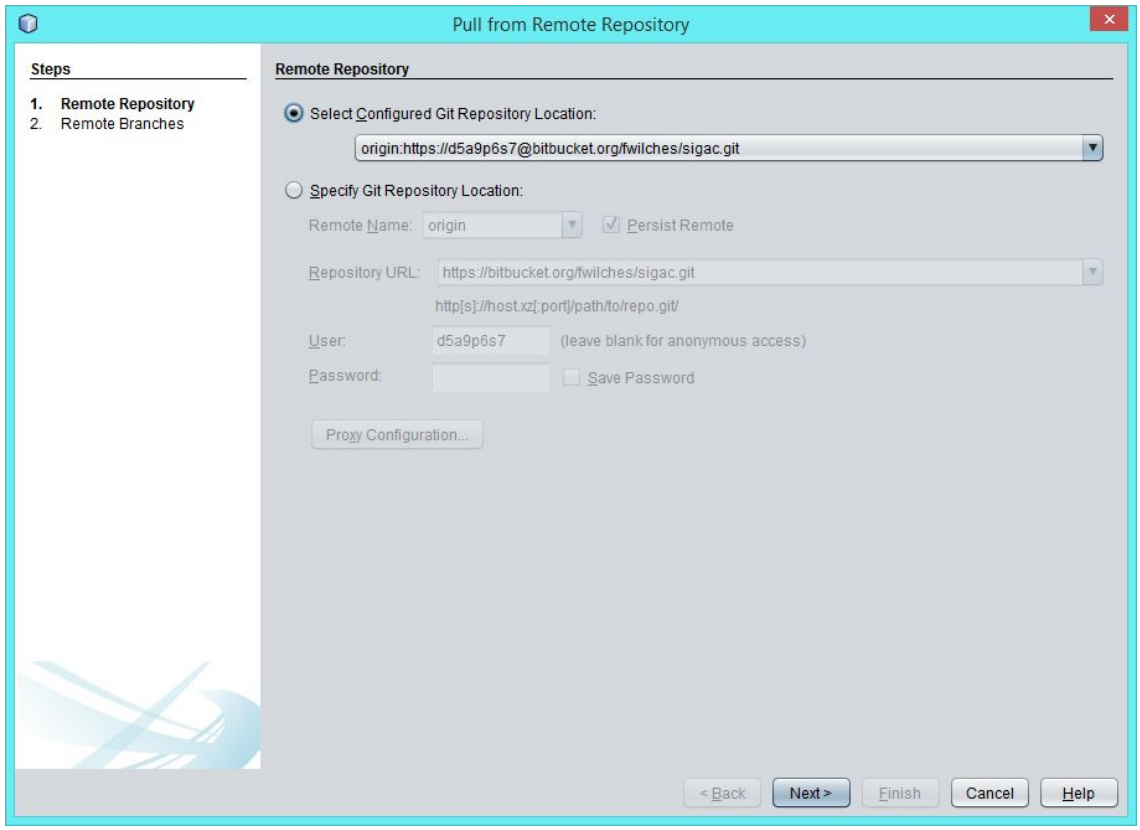
La idea es que el programador siempre se encuentre trabajando en una rama de nivel *topic.*La rama *develop* solo se usa para los merge y la *master* no la deben tocar para nada.

Pull

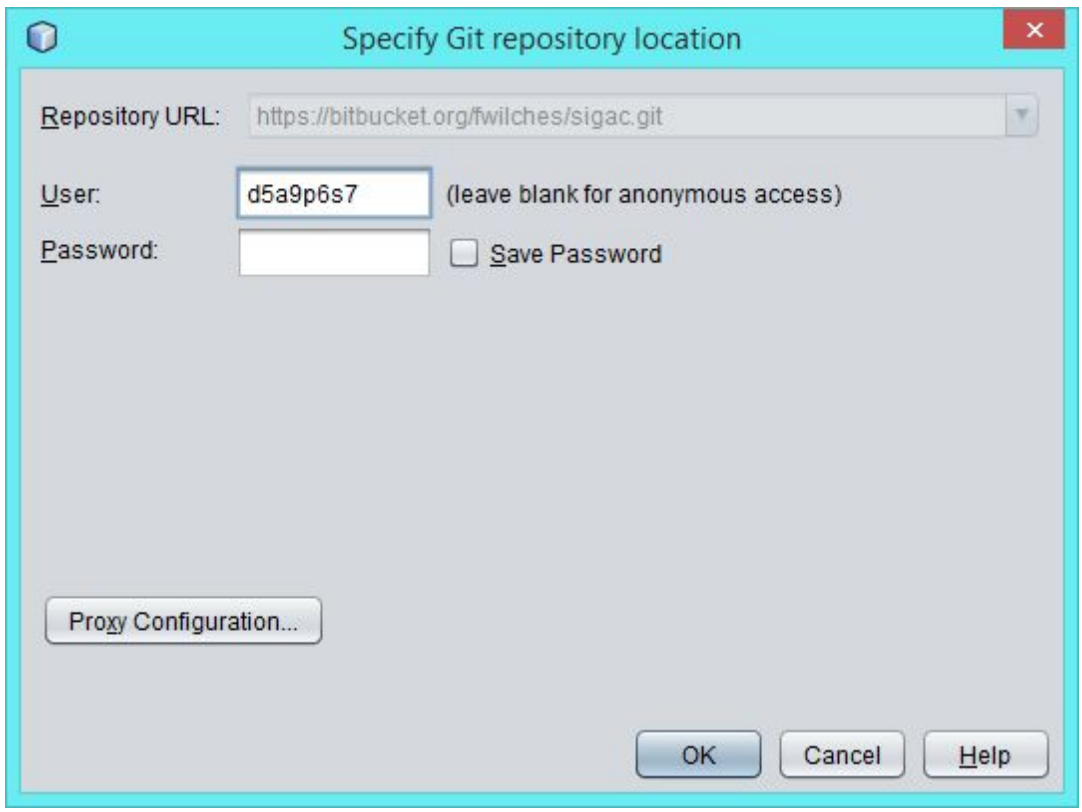
Pull



Next

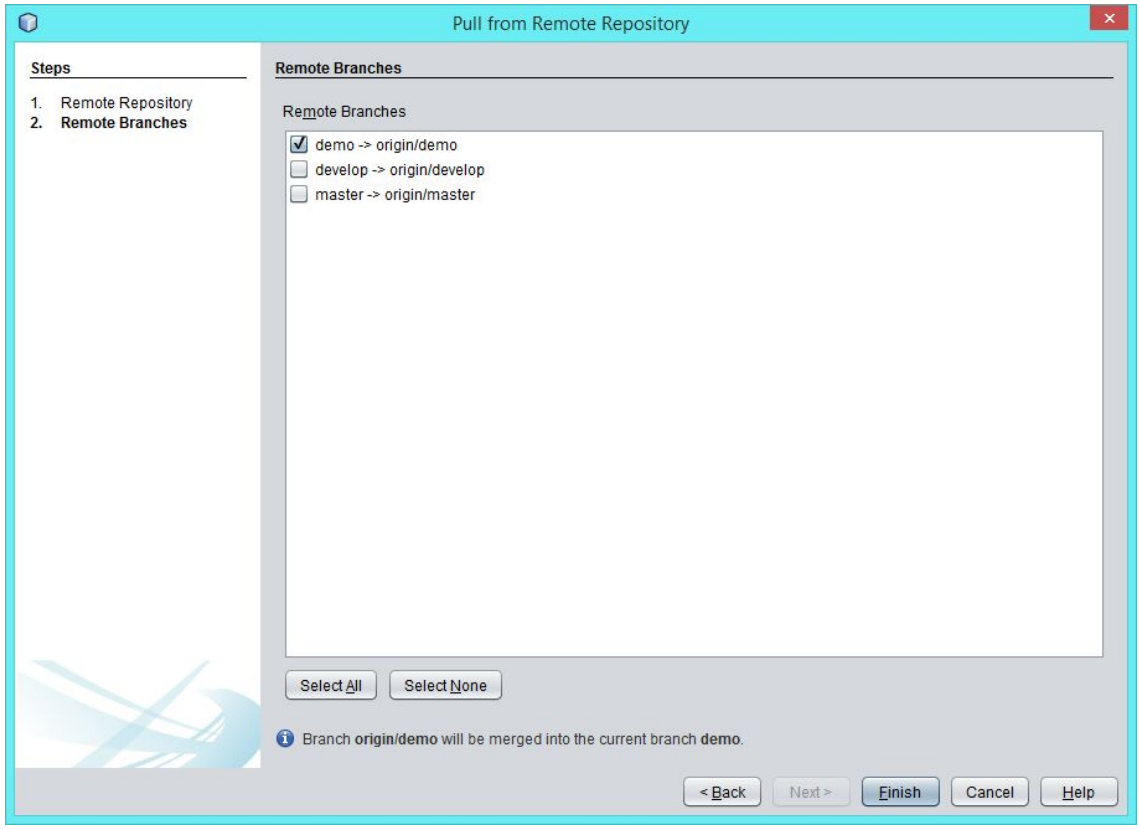


OK



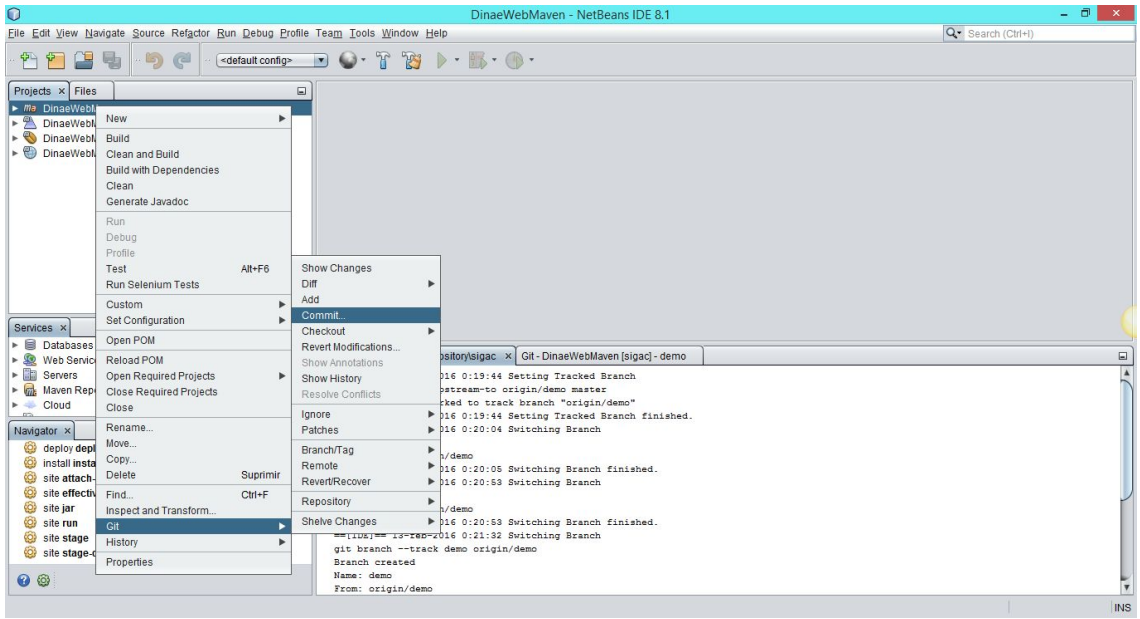
Seleccionar la rama que se va a actualizar.

Finish.



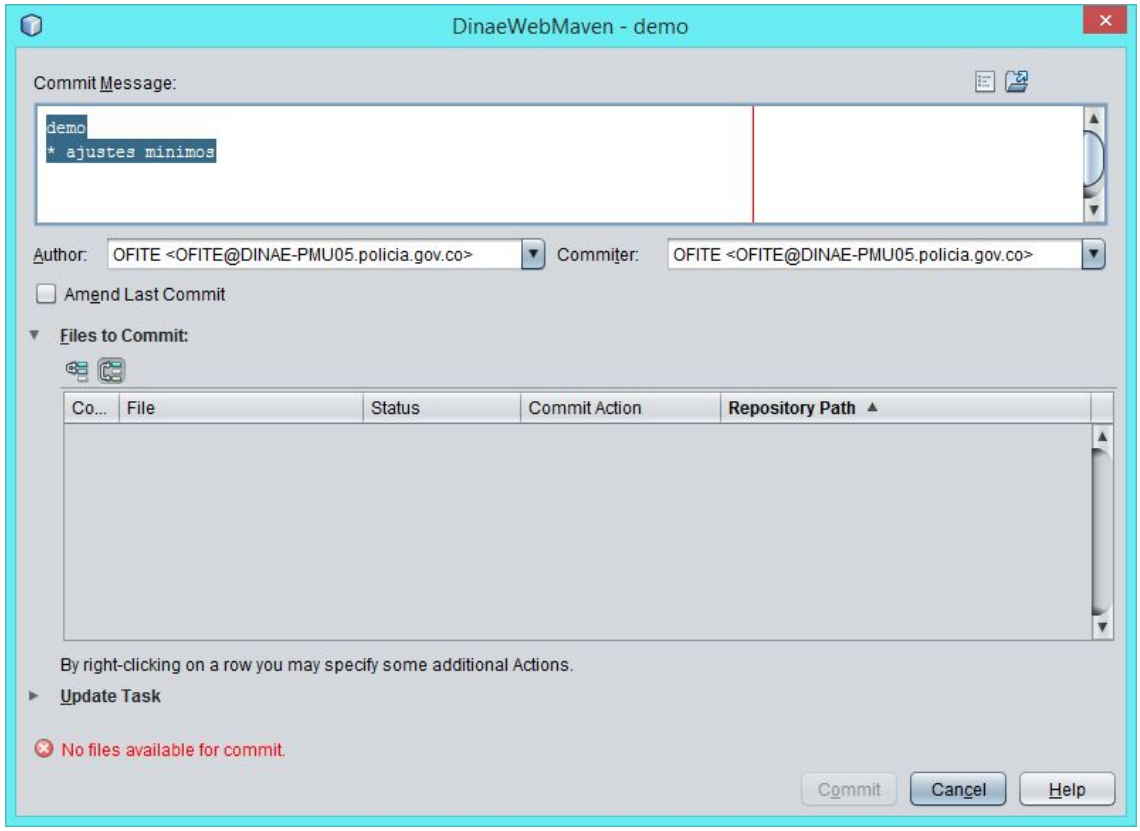
Commit

Commit



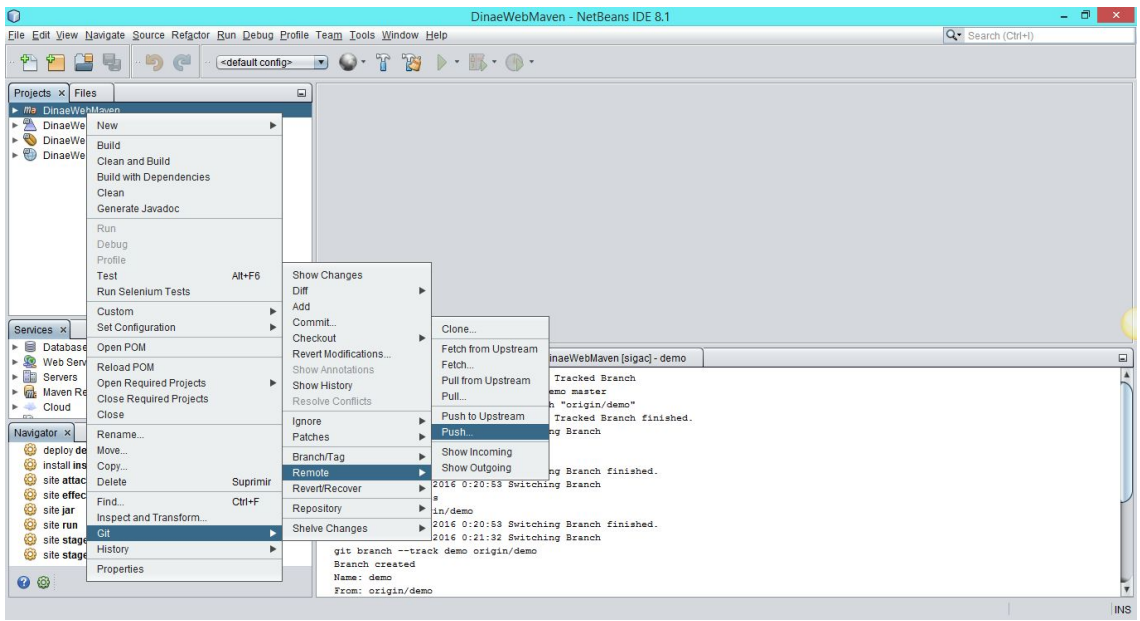
Diligenciar el mensaje del commit, en la tabla inferior se muestran los archivos que han sido modificados.

Commit.

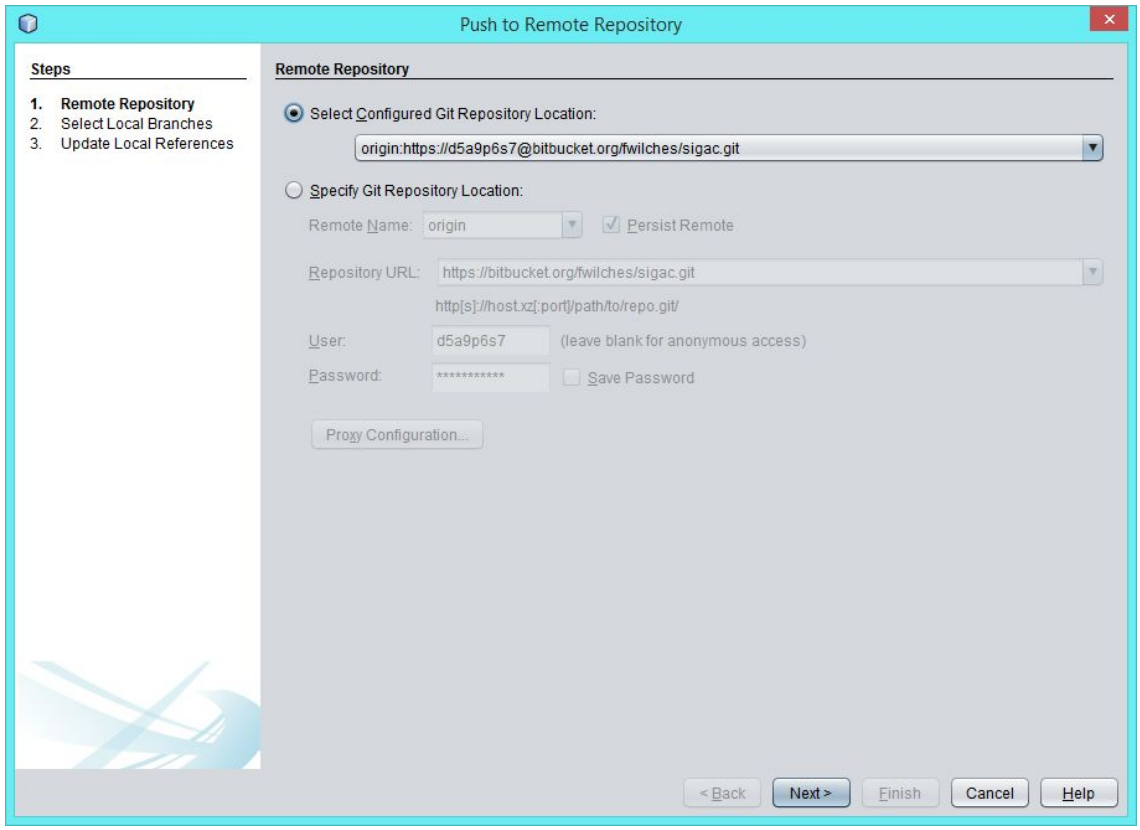


Push

Push

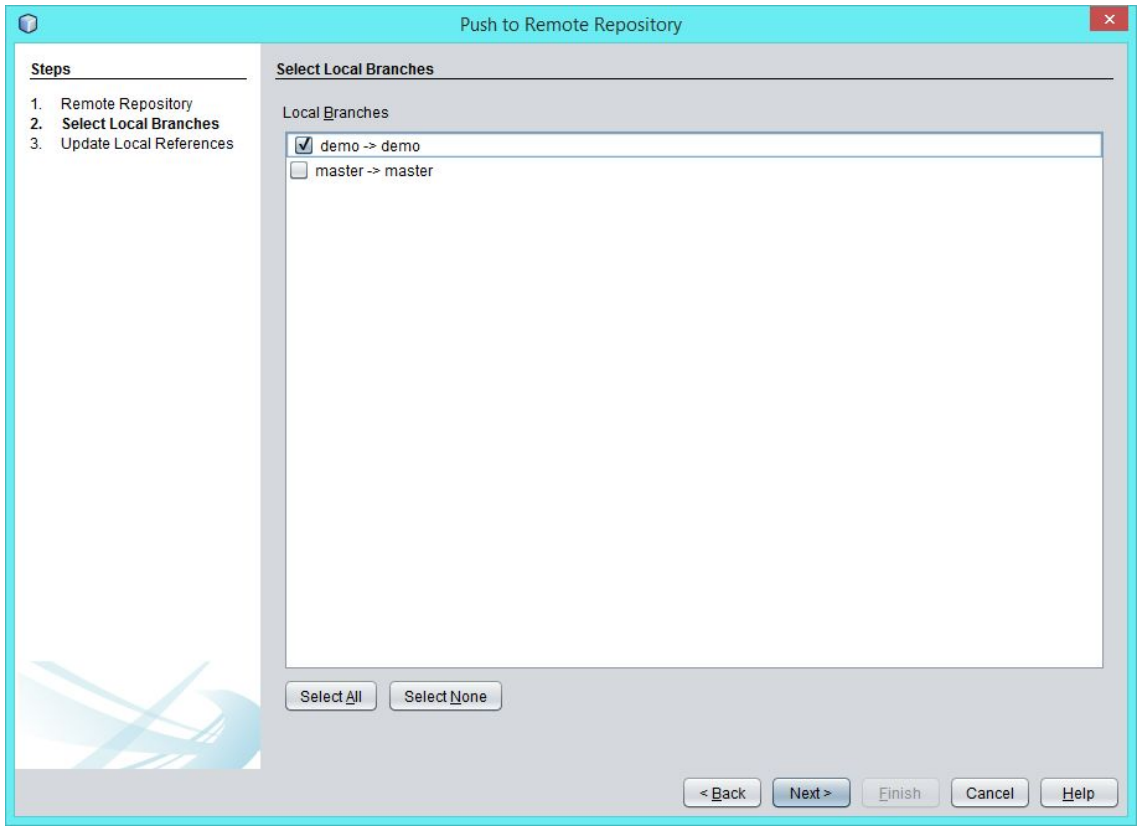


Next

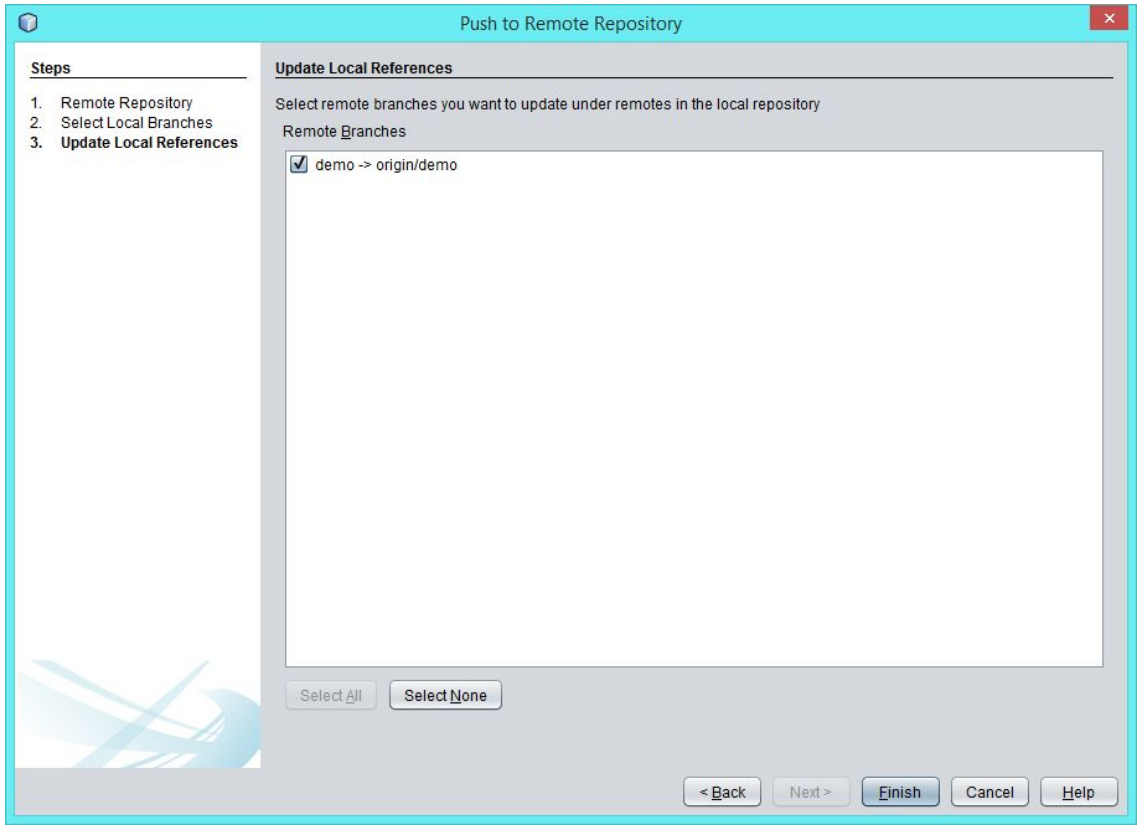


Seleccionar la rama que se desea modificar

Next



Finish



Recomendaciones

- Si se está trabajando en equipo con otras personas en la misma funcionalidad, se recomienda no hacer push de código con errores, a menos que se informe a los compañeros de trabajo o que sea por necesidad como el caso en que se necesite que otra persona revise el codigo que esta generando el error.
- En caso de estar trabajando con otras personas en la misma funcionalidad, se recomienda hacer pull con cierta frecuencia [una vez al dia como minimo].
- El comentario que se pone al hacer commit debe ser los suficientemente detallado.