Salesforce Development Methodology

May 15th, 2018

William Steele
Program Architect

Agenda

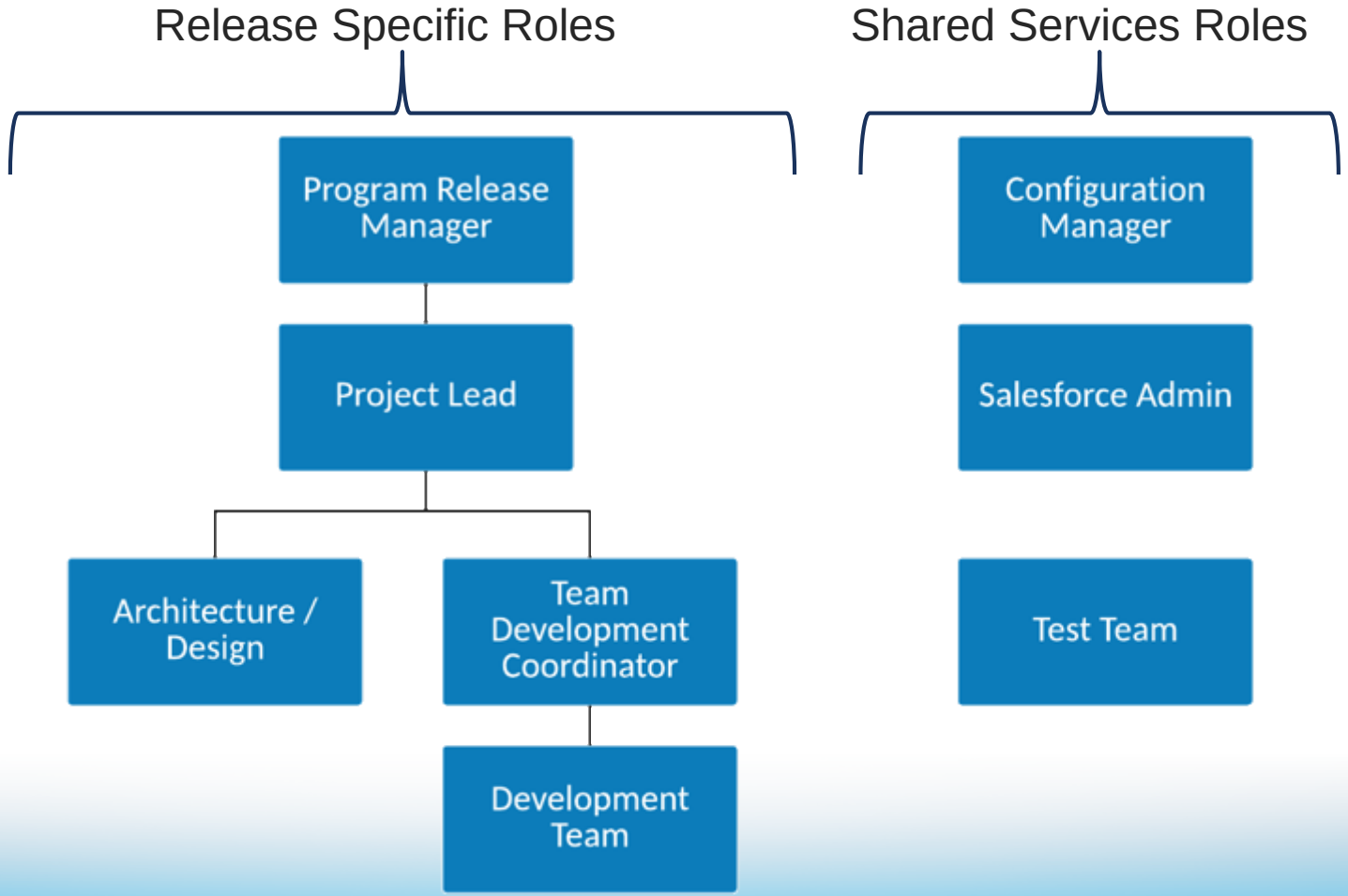
- Overview
  - Guiding Principles
  - Roles
  - Governance
  - Environment Overview
- Trunks, Branches, and Releases
- Development Tools and Process
- Training

Overview

Guiding Principles

- New build, enhancements, and changes must be tracked and approved in Application Enhancement Requests and DTC Exception Requests
- Major changes (workflows, validation rules) **shall not be done** directly in Production (test in lower env, then migrate for traceability and regression testing)
- Developers work in the same environment as much as possible (exceptions are POC's)
- All code is stored in a centralized repository
- Developers and admins have a standardized developer workstation, and follow the same processes to update code to the centralized repository
- Code is only moved from environment to environment from the centralized repository
- Code migration is automated and repeatable
- Admin privileges in higher environments should be restricted to prevent developers from making changes directly
- Manual tasks that cannot be automated should be documented in Excel that is checked in/out of GitHub.com

Roles (1 of 3)


Release Specific Roles
Shared Services Roles
Program Release Manager
Project Lead
Architecture / Design
Team Development Coordinator
Development Team
Configuration Manager
Salesforce Admin
Test Team

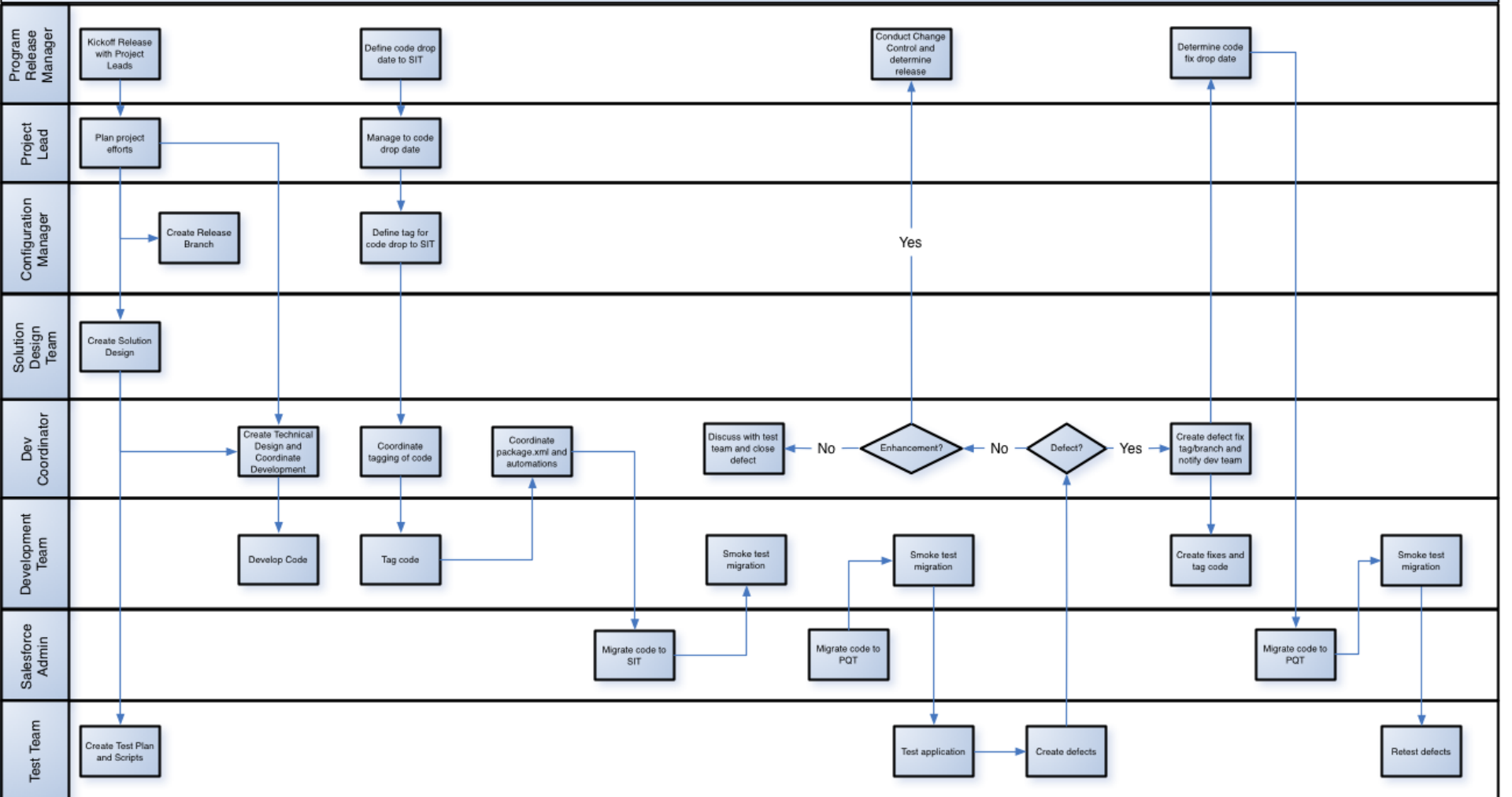
Roles (2 of 3)

Role	Task
Program Release Manager	➢ Coordinate and plan the development projects (work streams) for code drops and merges to SIT, SIT2, UAT, and Production ➢ Coordinate code migrations, smoke tests, data migration, testing, etc. for the Release ➢ Conduct Change Control Board ➢ Create and set release dates for all features and enhancements
Project Lead	➢ Coordinate and plan the overall efforts for the project – Requirements, Design, Build, Test, and Deploy ➢ Maintain detailed tasks, milestones and dependencies ➢ Manage weekly status reports ➢ Coordinate Architectural Design activities ➢ Plan various development activities with each Team Development Coordinator ➢ Ensure adherence to Salesforce methodology policies
Architecture / Design	➢ Support the creation of the solution design: ➢ Business Requirements, Use Cases, Mock-up UIs, Data Model, Security considerations, Data Integration and Migration , Service Assurance, Reports & Dashboard
Team Development Coordinator	➢ Work hand in hand Architecture / Design to be able to create/deliver/review the Technical Design documents ➢ Coordinate development work and packaging of work products (tagging) for code drops ➢ Ensure adherence to development standards, guidelines, and methodology ➢ Enforce entry/exit criteria for SDLC development and testing phases ➢ Coordinate and ensure quality checks (code/peer reviews) ➢ Coordinate branch creation for defect fixing, code drops, etc with the developers ➢ Create package.xml with respective metadata per the tag to provide to Salesforce Admin for each code drop ➢ Coordinate Script management / automation for Salesforce Admin (i.e. Custom Settings data migration) ➢ Create release notes for any manual tasks, known issues, functionality delivered, etc

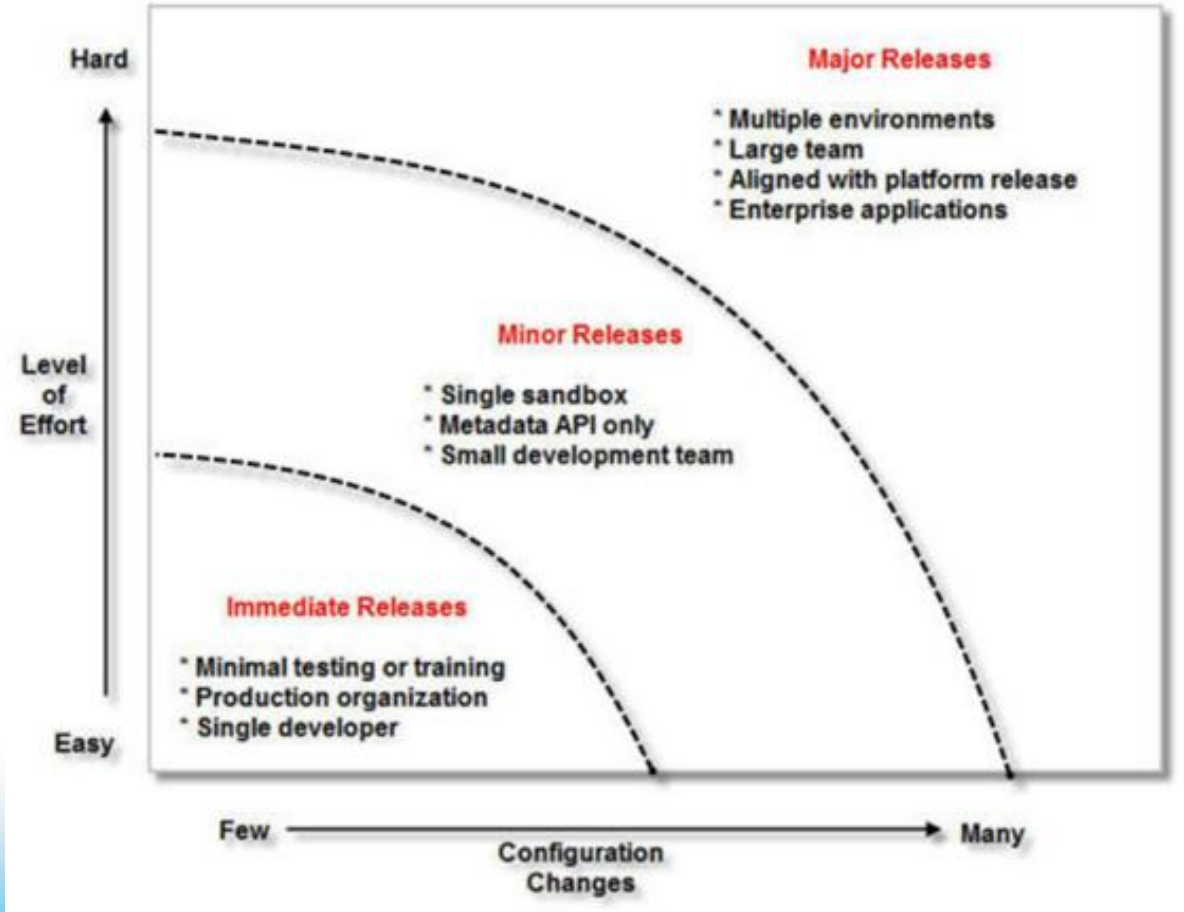
Roles (3 of 3)

Role	Task
Developers	➢ Development of code in the development environment. Should have own GitHub.com and Salesforce logins ➢ Associate development files to appropriate tag/label specified by Configuration Manager ➢ Smoke tests after code migrations to the various environments to ensure functionality is working as expected based on the code 'tagged' ➢ Perform Unit testing and maintain test results ➢ Ensure code coverage of > 95% ➢ Adhere to established development standards, guidelines and methodologies ➢ Responsible for merging and conflict resolution for code merges between branches
Configuration Manager	➢ Ensure adherence to configuration management policies ➢ Provide GitHub.com access ➢ Responsible for Trunk/branch creation for new releases and service packs ➢ Responsible for providing tag/label for release and code drops ➢ Ensure code drop tag is not modified after specified point in time (need to lock)
Salesforce Admin	➢ Provide Project/Issue access ➢ Sandbox creation and refreshes ➢ Code migration to SIT, UAT, Training, and Prod environments
Test Team	➢ Create test plan and test scripts applicable for the release ➢ Perform SIT2 on developed product ➢ Report defects

Sample Release Management Process



Governance Model for Changes (1 of 2)



Governance Model for Changes (2 of 2)

Release Type	Activities	Examples	Level of Effort	Group
L1 Immediate	➢ Small changes, rapidly applied directly to Production ➢ Min impact to users ➢ Outside scope of Change process	➢ Dashboards ➢ Reports ➢ List Views ➢ Data Loads ➢ Knowledge Articles	LOW ➢ No training required ➢ No integration ➢ Implemented by DTC	Business
L2 Minor (Monthly)	➢ Minor impact to users ➢ Config, test and deploy w/ minor impact to small # of users	➢ New Fields ➢ Picklist Values ➢ Standard page layouts updates ➢ New standard page layouts	MEDIUM ➢ < 1 day of training ➢ < 1 week of config ➢ IT involvement	DTC
L3 Major (Quarterly)	➢ Significant change to business users ➢ Major UI change, data migration / integration ➢ Apex/VF Code	➢ AppExchange apps ➢ Process impacting changes (workflows, validation rules) ➢ Visualforce page layouts ➢ Data migration ➢ Integration changes	HIGH ➢ 1+ days training ➢ 1+ weeks config ➢ 1+ weeks of integration development ➢ IT lead	DTC

Work Instructions: Application/Enhancement Requests

Who:

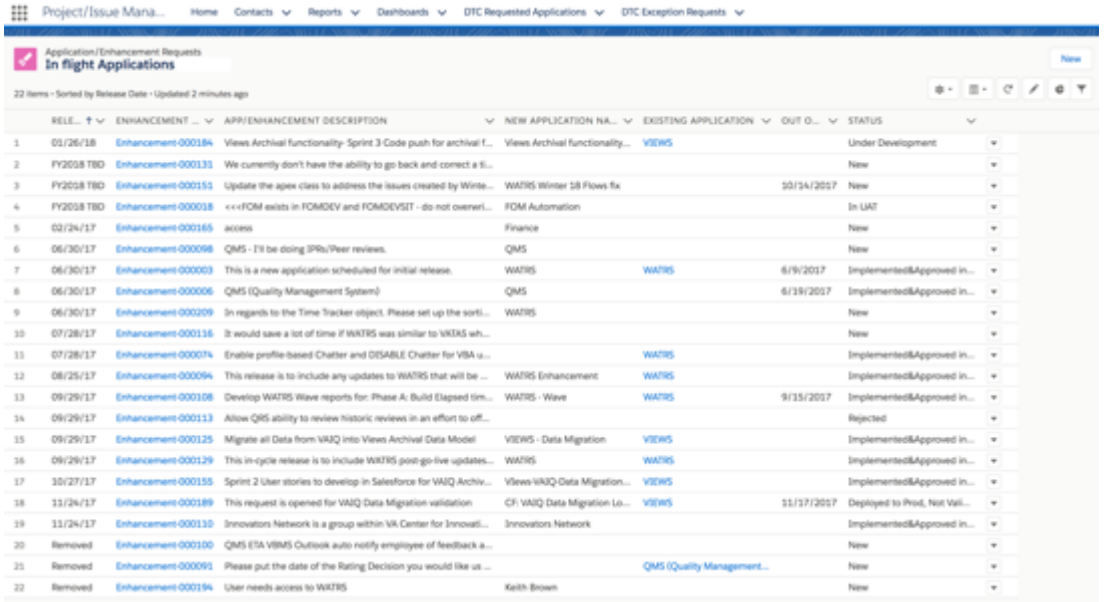
- Project Manager or Business Owner

When:

- At beginning of each release, the project team creates a request for a new/enhanced application. The project management reviews and reviews the application for work once the requirements have been met.

Steps:

- Log into VA Production Org
- Access the Application/Enhancement tab
- Create/edit an application enhancement request
- Provide all required details to facilitate more efficient approval



Work Instructions: DTC Exception Requests

Who:

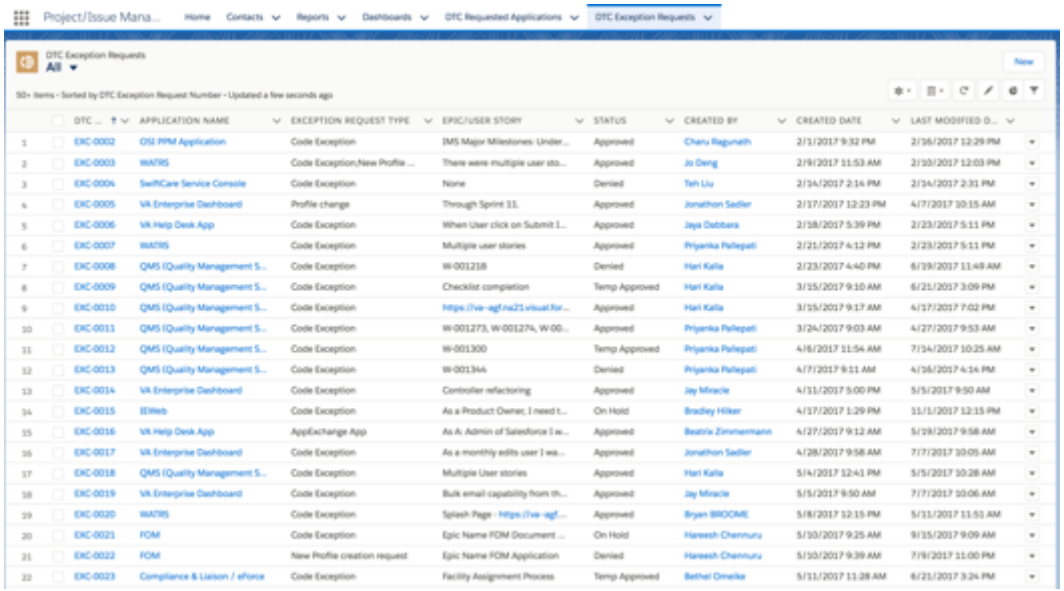
- Development teams

When:

- When a requirement requires additional fields or capabilities beyond configuration, a DTC Exception request needs to be filed to require agreement from the team
- Developer meeting attendees should review the list weekly and determine whether to accept/reject
- If accepted, the development team should prepare to move forward in the development cycle

Steps:

- Log into the VA production account
- Access the DTC Exception Requests tab
- Create/edit a DTC exception request
- Provide all required details to facilitate more efficient approval



Release/Build Notes (1 of 2)

New for the DTC process!

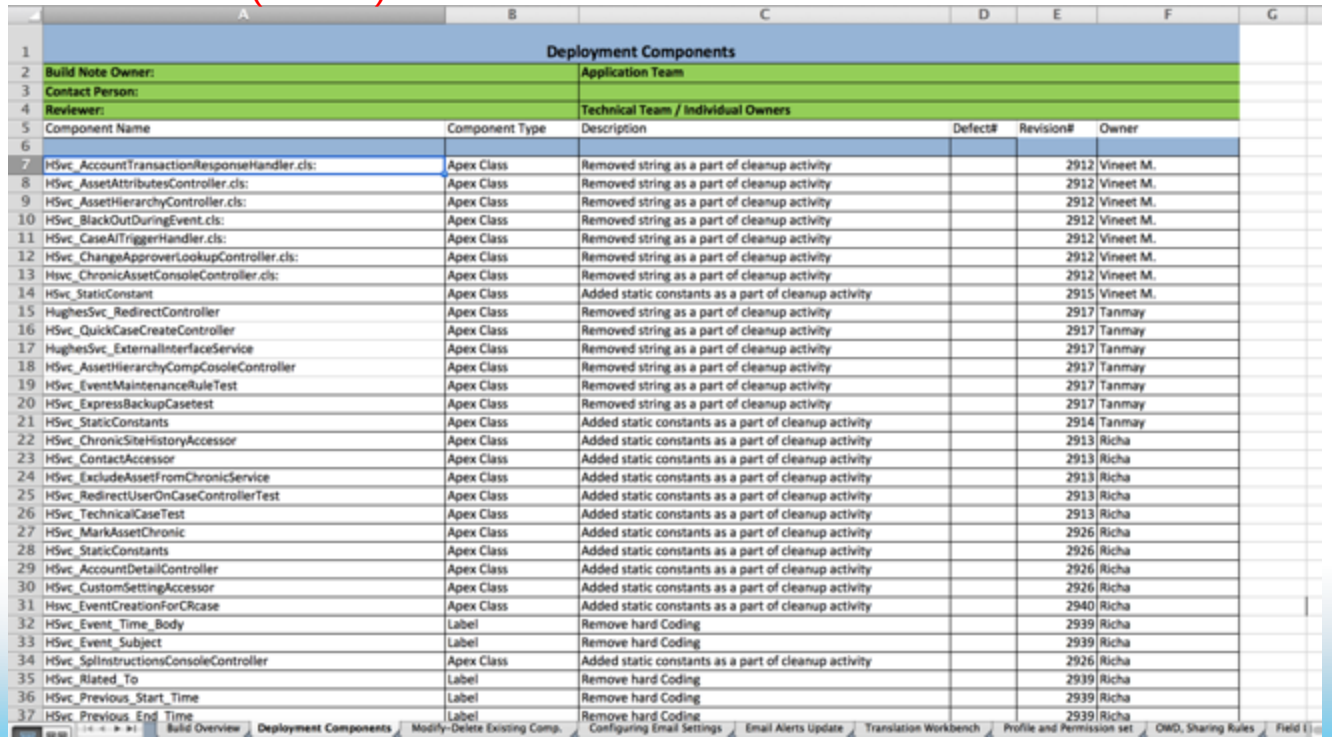
All teams shall document their changes in the following xls template and upload the xls to a ‘Deployments’ sub-branch/folder on the development branch. Manual steps shall be tracked here. Copies of these will be placed in VA Production under Application or Exception requests so those without production access (testers) can make use of.

Putting together format and instructions

Release/Build Notes (2 of 2)

New for the DTC process!

All teams shall document their changes in the following xls template and upload the xls to a ‘Deployments’ sub-branch/folder on the development branch. Manual steps shall be tracked here. Copies of these will be placed in VA Production under Application or Exception requests so those without production access (testers) can make use of.

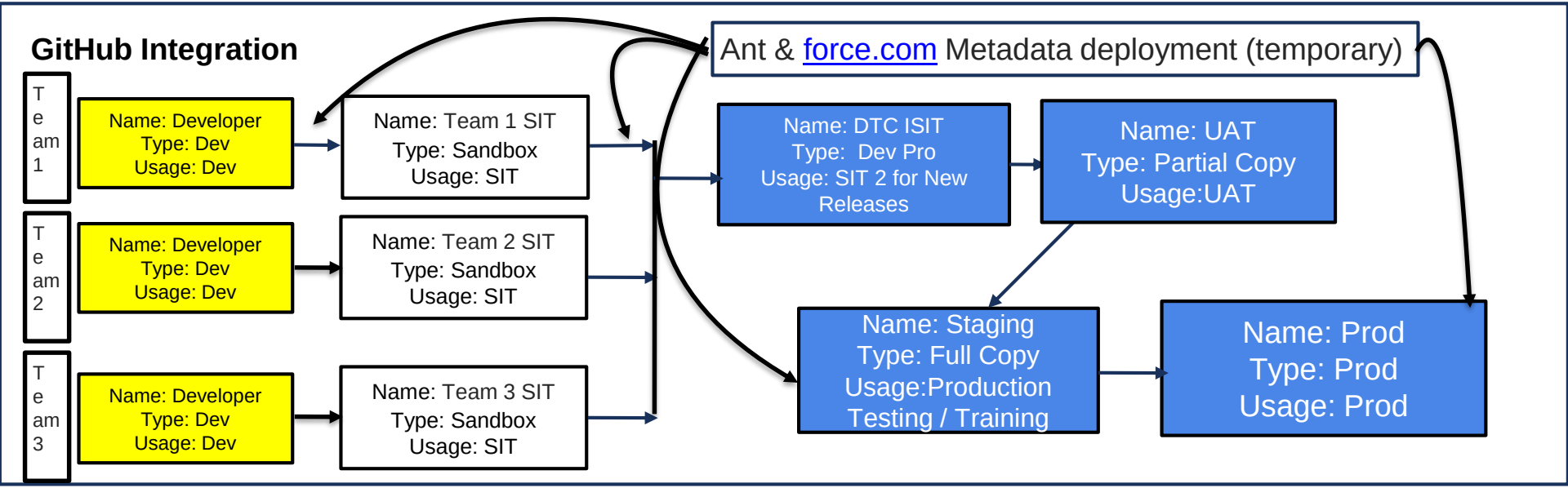


	A	B	C	D	E	F	G
1	**Deployment Components**						
2	**Build Note Owner:**		**Application Team**				
3	**Contact Person:**						
4	**Reviewer:**		**Technical Team / Individual Owners**				
5	Component Name	Component Type	Description	Defect#	Revision#	Owner	
6							
7	HSvc_AccountTransactionResponseHandler.cls:	Apex Class	Removed string as a part of cleanup activity		2912	Vineet M.	
8	HSvc_AssetAttributesController.cls:	Apex Class	Removed string as a part of cleanup activity		2912	Vineet M.	
9	HSvc_AssetHierarchyController.cls:	Apex Class	Removed string as a part of cleanup activity		2912	Vineet M.	
10	HSvc_BlackOutDuringEvent.cls:	Apex Class	Removed string as a part of cleanup activity		2912	Vineet M.	
11	HSvc_CaseAITriggerHandler.cls:	Apex Class	Removed string as a part of cleanup activity		2912	Vineet M.	
12	HSvc_ChangeApproverLookupController.cls:	Apex Class	Removed string as a part of cleanup activity		2912	Vineet M.	
13	Hsvc_ChronicAssetConsoleController.cls:	Apex Class	Removed string as a part of cleanup activity		2912	Vineet M.	
14	HSvc_StaticConstant	Apex Class	Added static constants as a part of cleanup activity		2915	Vineet M.	
15	HughesSvc_RedirectController	Apex Class	Removed string as a part of cleanup activity		2917	Tanmay	
16	HSvc_QuickCaseCreateController	Apex Class	Removed string as a part of cleanup activity		2917	Tanmay	
17	HughesSvc_ExternalInterfaceService	Apex Class	Removed string as a part of cleanup activity		2917	Tanmay	
18	HSvc_AssetHierarchyCompCosoleController	Apex Class	Removed string as a part of cleanup activity		2917	Tanmay	
19	HSvc_EventMaintenanceRuleTest	Apex Class	Removed string as a part of cleanup activity		2917	Tanmay	
20	HSvc_ExpressBackupCasetest	Apex Class	Removed string as a part of cleanup activity		2917	Tanmay	
21	HSvc_StaticConstants	Apex Class	Added static constants as a part of cleanup activity		2914	Tanmay	
22	HSvc_ChronicSiteHistoryAccessor	Apex Class	Added static constants as a part of cleanup activity		2913	Richa	
23	HSvc_ContactAccessor	Apex Class	Added static constants as a part of cleanup activity		2913	Richa	
24	HSvc_ExcludeAssetFromChronicService	Apex Class	Added static constants as a part of cleanup activity		2913	Richa	
25	HSvc_RedirectUserOnCaseControllerTest	Apex Class	Added static constants as a part of cleanup activity		2913	Richa	
26	HSvc_TechnicalCaseTest	Apex Class	Added static constants as a part of cleanup activity		2913	Richa	
27	HSvc_MarkAssetChronic	Apex Class	Added static constants as a part of cleanup activity		2926	Richa	
28	HSvc_StaticConstants	Apex Class	Added static constants as a part of cleanup activity		2926	Richa	
29	HSvc_AccountDetailController	Apex Class	Added static constants as a part of cleanup activity		2926	Richa	
30	HSvc_CustomSettingAccessor	Apex Class	Added static constants as a part of cleanup activity		2926	Richa	
31	Hsvc_EventCreationForCRcase	Apex Class	Added static constants as a part of cleanup activity		2940	Richa	
32	HSvc_Event_Time_Body	Label	Remove hard Coding		2939	Richa	
33	HSvc_Event_Subject	Label	Remove hard Coding		2939	Richa	
34	HSvc_SplInstructionsConsoleController	Apex Class	Added static constants as a part of cleanup activity		2926	Richa	
35	HSvc_Rlated_To	Label	Remove hard Coding		2939	Richa	
36	HSvc_Previous_Start_Time	Label	Remove hard Coding		2939	Richa	
37	HSvc_Previous_End_Time	Label	Remove hard Coding		2939	Richa	

Build Overview | Deployment Components | Modify-Delete Existing Comp. | Configuring Email Settings | Email Alerts Update | Translation Workbench | Profile and Permission set | OWD, Sharing Rules | Field (...

Flow of Code through the VA DTC Environments

Code must move through SDLC of Dev -> SIT -> SIT 2 -> UAT -> Prod



- Code is only pushed from GitHub.com
- Ant and shell scripts will be used to automate the deployments and pull from GitHub.com

Legend: 1 - Emergency fixes to Prod Code Base

Individual Developer | Prod Code Base

Trunks, Branches, and Releases

Subversion vs GIT: Which is better?

Philosophical Question. Both have their advantages and disadvantages. Given the various developers/SI's that come in/out of VA, GIT makes the best sense in providing reliability, flexibility and scalability.

Subversion

- Centralized Repository
- Must be connected to Repository
- Simpler to learn, and simplified checkout/lock
- Simpler to standardize processes

GIT

- Decentralized / Distributed
- Can work on local repository (does not need to be connected)
- Harder to learn. Added complexities around commands locally vs when connected, and managing changes/merges among developers.
- Flexible, but means there's more than one way to do something
- Faster than Subversion due to decentralization
- Branching is easier

Decision:

Go with GitHub.com for now as we build a foundation moving us towards a true CI (continuous integration) environment.

Trunks, Branches, and Releases: Git

Main Branches

Three main branches with an infinite lifetime:

- Master (Trunk)
- Release (for all teams within a specific release)
- Develop (for each team to perform development during a sprint)

Master to be the main branch where the source code of HEAD always reflects a production-ready state

Release to be the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the current release

Develop to be the main branch where the source code of HEAD always reflects a state with the latest delivered development changes for the each team

When the source code in the develop branch reaches a stable point and is ready to be deployed to environments above dev, all of the changes should be merged back into the release branch and then tagged with a release number.

The release branch will remain viable until it has passed UAT and is ready to be deployed to production. At that point, the release branch will be merged into the master branch.

Each time when changes are merged back into master, this is a new production release by definition. Ultimate Goal is to automatically build and roll-out software to our production servers everytime there was a commit on master.

Support Branches

The different types of branches we will use are:

- Feature branches
- Release branches
- Hotfix branches

Each of these branches have a specific purpose and are bound to strict rules as to which branches may be their originating branch and which branches must be their merge targets.

By no means are these branches “special” from a technical perspective. The branch types are categorized by how we use them. They are of course plain old branches.

Feature Branches (Prototypes)

May branch off from:

- Develop

Must merge back into:

- Develop

Branch naming convention:

- anything except master, develop-<team>-<application>*, release-<release #>, or hotfix-<team>-<application>*

Feature branches (or sometimes called topic branches) are used to develop new features for the upcoming or a distant future release. When starting development of a feature, the target release in which this feature will be incorporated may well be unknown at that point. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).

Release Branches

May branch off from:

- master

Must merge back into:

- master

Branch naming convention:

- release-<release #>

Release branch supports preparation of a new production release a.k.a TSIT/UAT/Staging. By doing all of this work on a release branch, the develop branch is cleared to receive features for the next big release.

The key moment to branch off a new release branch from master is at the start of a new release. Once the release branch is created for all teams, a develop branch will be created to support individual team development.

It is exactly at the start of a release branch that the upcoming release is moved into Test Phase. Up until that moment, the develop branch reflects changes for the upcoming release.

When the state of the release branch is ready to become a real release, some actions need to be carried out

- First, the release branch is merged into master (since every commit on master is a new release by definition)
- Next, that commit on master must be tagged
- Finally, the changes made on the release branch need to be merged back into develop, so that future releases also contain these bug fixes.
- Once merged into production this release branch is CLOSED

A New Release Branch is Created for Each New Major Release

Hotfix Branches

May branch off from:

- Master

Must merge back into:

- Master, Release, (Develop)

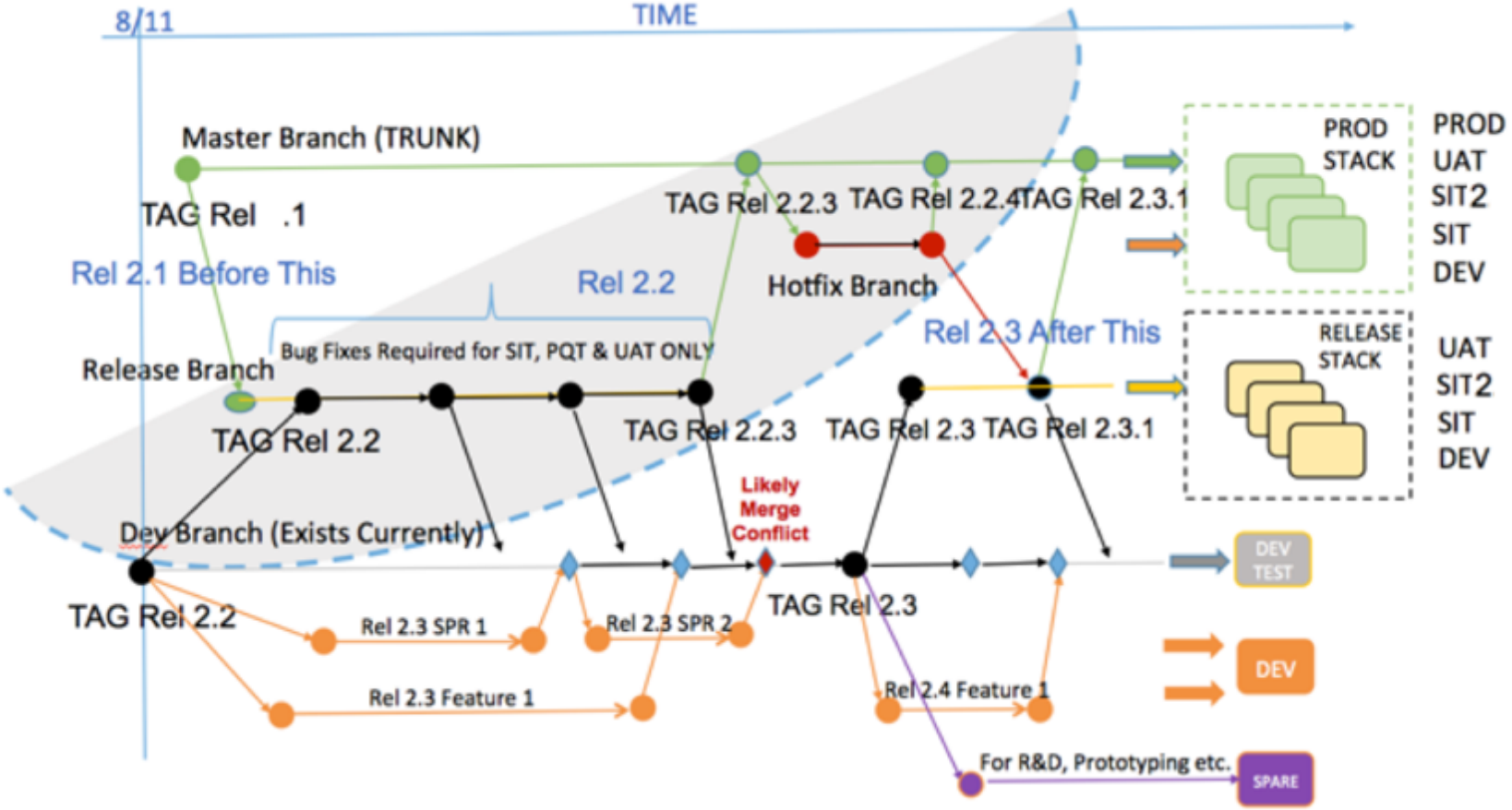
Branch naming convention:

- hotfix-<team>-<application>*

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version. The essence is that work of team members (on the develop branch) can continue, while another person is preparing a quick production fix.

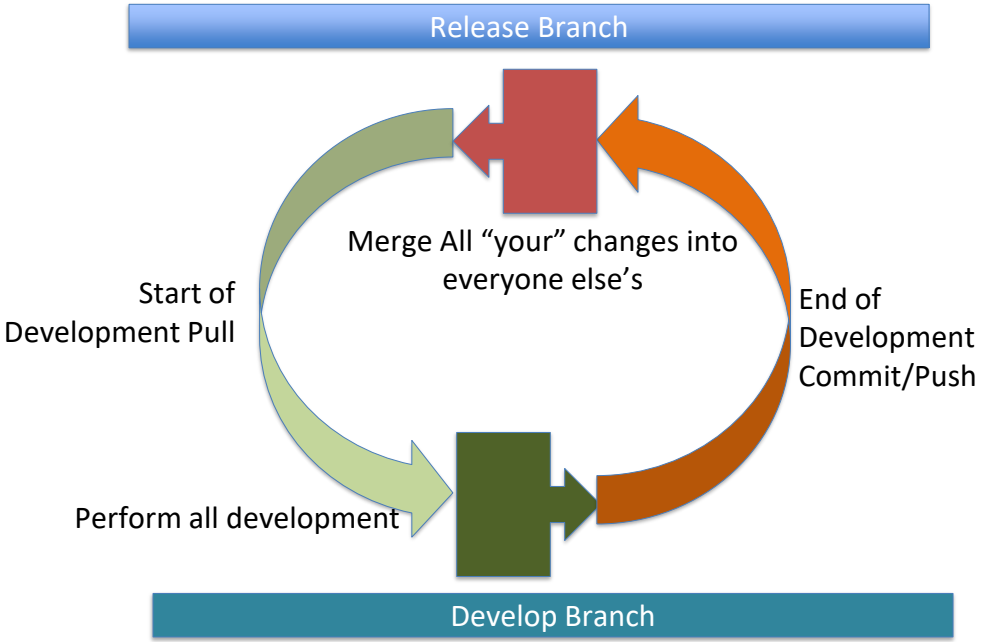
When finished, the bugfix needs to be merged back into master, but also needs to be merged back into release branch, in order to safeguard that the bugfix is included in the next release as well. Back-merging the bugfix into the release branch will eventually result in the bugfix being merged into develop too, when the release branch is CLOSED. If a release branch currently does NOT exists, the hotfix changes need to be merged into the develop branch. (If work in develop immediately requires this bugfix and cannot wait for the release branch to be finished, you may safely merge the bugfix into develop now already as well.)

Trunks and Branches

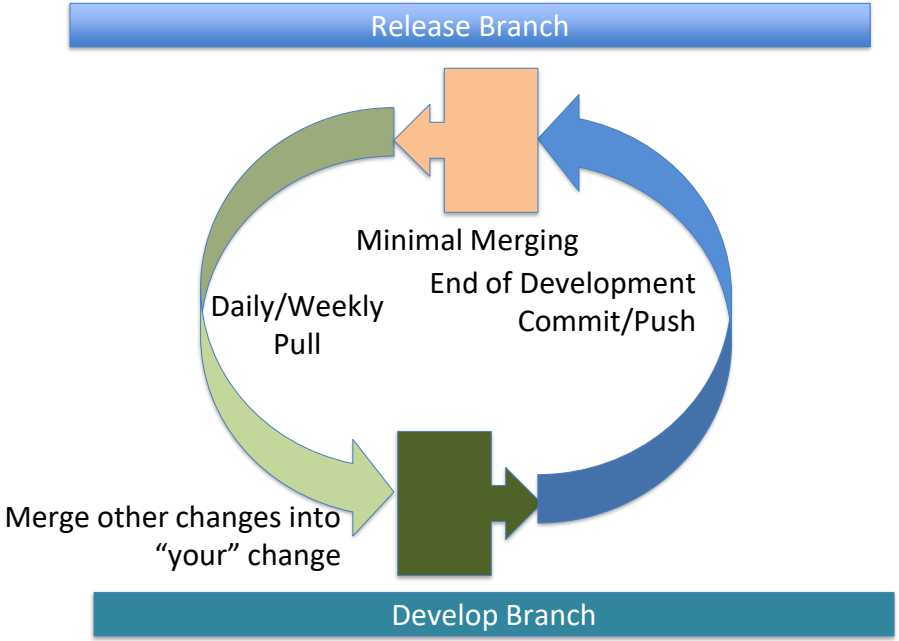


Usual Pull/Push/Merge Process

Might appear the most logical

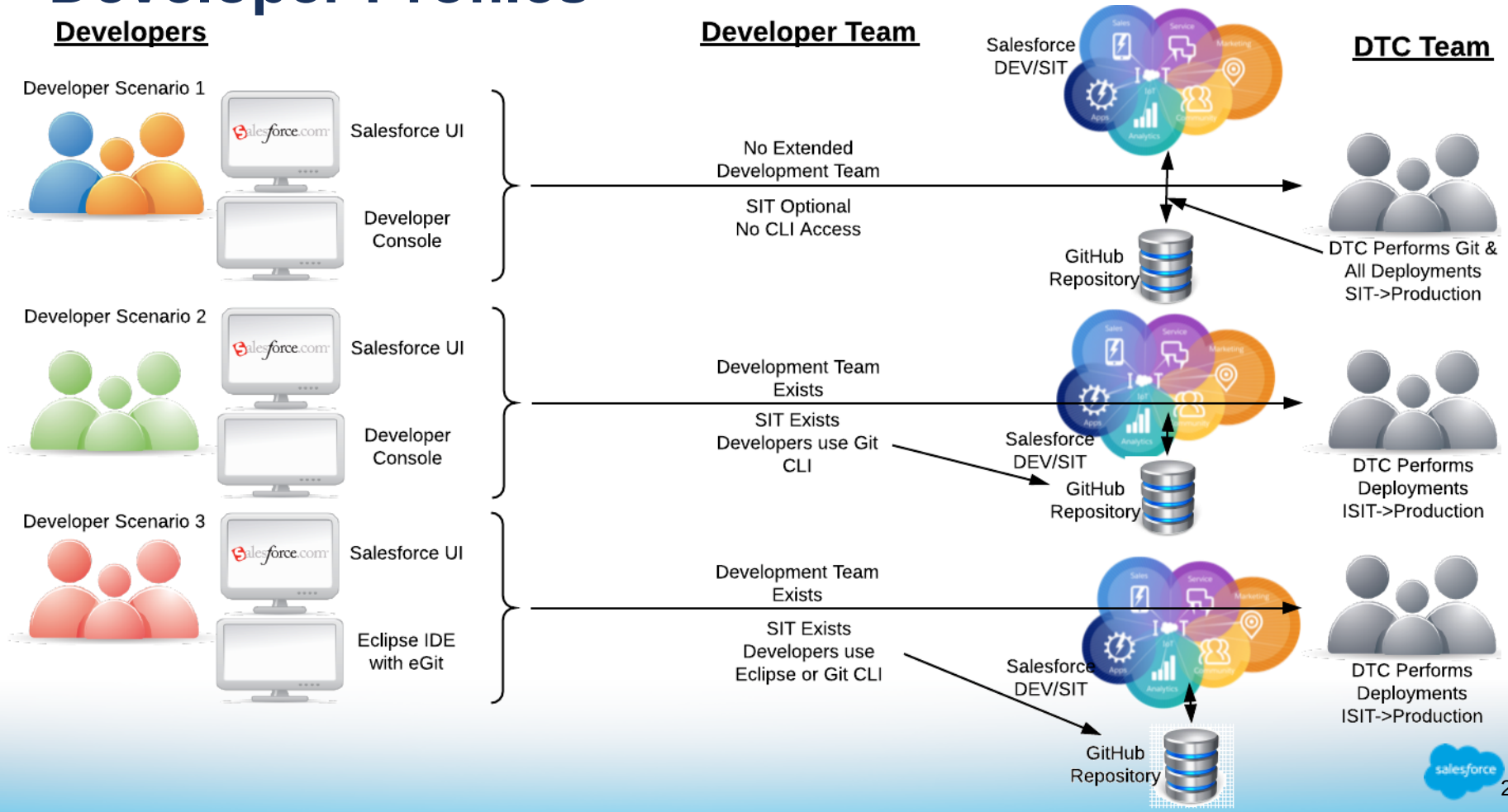


Best Pull/Push/Merge Process



What Type of Developer are YOU?

Developer Profiles



Developer Profile for Developer Scenario 1

Individual developer(s), no extended team, no SIT for testing, only uses SF UI & developer console

Developer

Setup

- No additional setup

Activities

- Uses SF UI and/or developer console
- Does not have a SIT box

Development Team

No extended development team – no activities here

DTC Deployment team

Setup

(see next section on Development Tools)

- Install and configure GIT CLI
- Install and configure Ant
- Install and configure Force.com Migration Tool
- Configure build.xml, build.properties, & package.xml for each team/sandbox

Activities

- DTC script to perform Git pull and Merge
- DTC script to export changes from SF
- DTC script to push changes to GitHub.com
- Perform GitHub.com merging with developer(s)
- For SIT/pull GitHub.com repo and deploy

Developer Profile for Developer Scenario 2

Individual developer(s), has an extended team, uses SIT for testing, uses SF UI & developer console & Git CLI

Developer

Setup

(see next section on Development Tools)

- Install and configure GIT CLI
- Install and configure Ant
- Install and configure Force.com Migration Tool
- Configure build.xml, build.properties, & package.xml for each team/sandbox

Activities

- Uses SF UI and/or developer console
- Uses DTC deployment scripts for extract, Git pull, Git merge, Git push

Development Team

If there isn't a Team Development Coordinator-no activities here

Setup

(see next section on Development Tools)

- Install and configure GIT CLI
- Install and configure Ant
- Install and configure Force.com Migration Tool
- Configure build.xml, build.properties, & package.xml for each sandbox

Activities

- DTC script to perform Git pull and Merge
- DTC script to export changes from SF
- DTC script to push changes to GitHub.com
- Perform GitHu.comb merging with developer(s)
- For SIT/pull GitHub.com repo and deploy

DTC Deployment team

Setup

(see next section on Development Tools)

- Install and configure GIT CLI
- Install and configure Ant
- Install and configure Force.com Migration Tool
- Configure build.xml, build.properties, & package.xml for each team/sandbox

Activities

- DTC script to perform Git pull and Merge
- DTC script to export changes from SF
- DTC script to push changes to GitHub.com
- Perform GitHub.com merging with developer(s)

Developer Profile for Developer Scenario 3

Individual developer(s), has an extended team, uses SIT for testing, uses SF UI & Eclipse IDE & eGit for development

Developer Setup

(see next section on Development Tools)

- Install and configure Eclipse & SF Plugin
- Install and configure eGit

Activities

- Uses SF UI
- Uses Eclipse SF IDE Plugin
- Uses eGit Plugin for all GitHub.com integration

Development Team

If there isn't a Team Development Coordinator-no activities here

Setup

(see next section on Development Tools)

- Install and configure GIT CLI
- Install and configure Ant
- Install and configure Force.com Migration Tool
- Configure build.xml, build.properties, & package.xml for each sandbox

Activities

- DTC script to perform Git pull and Merge
- DTC script to export changes from SF
- DTC script to push changes to GitHub.com
- Perform GitHub.com merging with developer(s)
- For SIT/pull GitHub.com repo and deploy

DTC Deployment team Setup

(see next section on Development Tools)

- Install and configure GIT CLI
- Install and configure Ant
- Install and configure Force.com Migration Tool
- Configure build.xml, build.properties, & package.xml for each team/sandbox

Activities

- DTC script to perform Git pull and Merge
- DTC script to export changes from SF
- DTC script to push changes to GitHub.com
- Perform GitHub.com merging with developer(s)

Development Tools

Problem Statement: Development Tools

Currently, there isn't a published standard for development tools. Which ones are recommended and which versions are known to work the best? Also, there isn't an official procedure for automating the build and deploy processes via Any or some other tool.

Impacts:

- Without the use of GitHub.com plug-in, code is not being checked-out. This can lead to multiple developers updating the same file, with no source of truth
- Without the use of Ant, code migrations are not being done in a consistent format.

Proposed Resolution:

- All developers should use Eclipse with the Force.com IDE and GitHub.com plug-in (eGit)
- Only those responsible for migration need to have Ant installed on their machine
    - Long-term, a Migration Server with Ant installed or Jenkins should be used for all migrations

Developer Workstation

All developers and admins shall have the following developer workstation setup:

- A supported operating system
  - Windows 7, 8, or 10
  - macOS 10.7, 10.8, 10.9, 10.10, or 10.11
  - Ubuntu 12.04 LTS or 14.04 LTS
- Java SE Development Kit (JDK), Runtime Environment 8, or later
- Eclipse Mars or Neon
  - The Eclipse IDE for Java Developers distribution is recommended
- Eclipse plug-in for GIT (eGit)

* Sublime (Text Editor) and MavensMate (Open Source for Force.com IDE) is another popular option, but is not the official standard in the Salesforce link above. Sublime costs $50-70/license, and Sublime GIT plugin cost $16/user

Tool Versions (1 of 2)

All developers and admins shall have the following developer workstation setup:

Tool	Version	Who	Install Location
Force.com IDE	Eclipse46 Neon	Developer	Within Eclipse, point to the following URL(links to download Java and Eclipse Neon 3): https://developer.salesforce.com/docs/atlas.en-us.eclipse.meta/eclipse/ide_install.htm
eGit	4.9.x	Developer	Within Eclipse, point to the following URL: http://download.eclipse.org/egit/updates
Git CLI	v41	Developer	Download and install from git: https://git-scm.com/downloads
Ant & Force.com Ant Migration Tool	v41	Admin	Any Salesforce Org: Setup -> Build -> Develop -> Tools v41: —OR— https://developer.salesforce.com/docs/atlas.en-us.daas.meta/daas/forcemigrationtool_install.htm
SOAP UI	5.3.x	Integration Developer	https://www.soapui.org/downloads/soapui.html

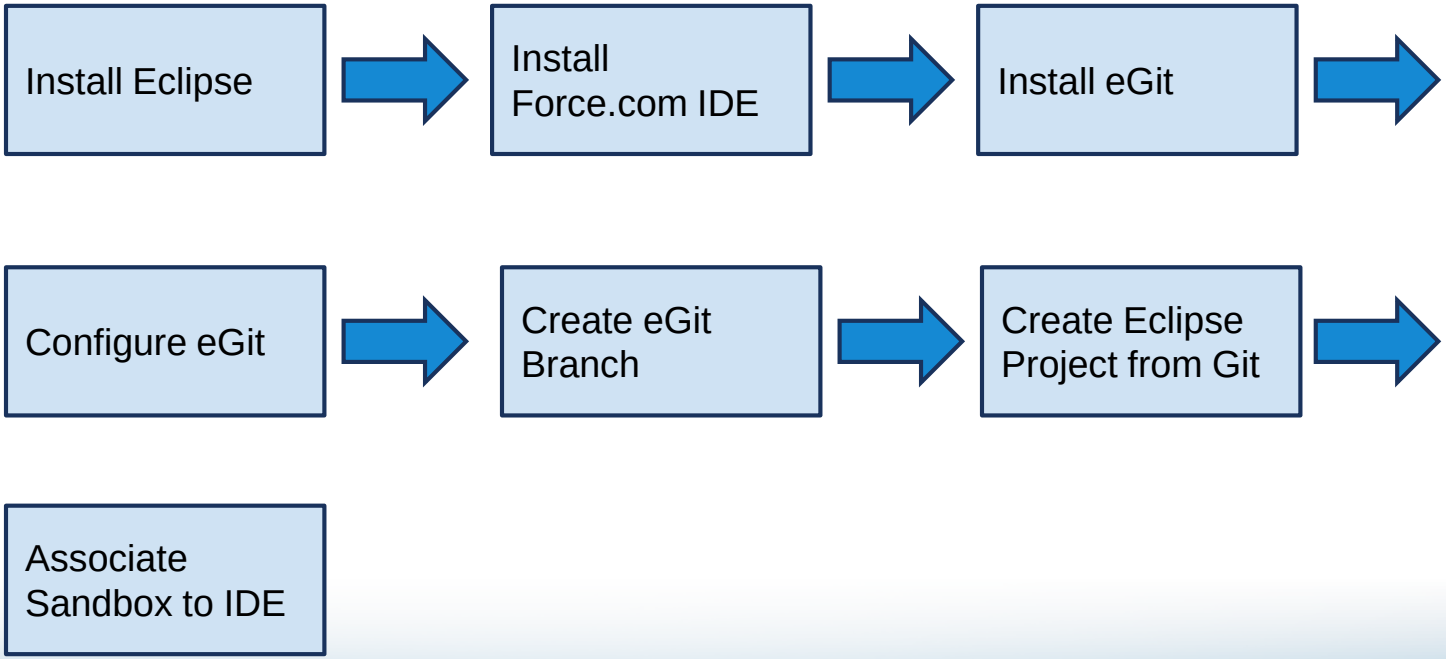
Tool Versions (2 of 2)

All developers and admins shall have the following developer workstation setup:

Tool	Version	Who	Install Location
Apex Data Loader	V36	Integration Developer	Any Salesforce Org: Setup -> Administration Setup -> Data Management -> Data Loader v34: https://developer.salesforce.com/page/Data_Loader
Force.com Explorer	0.58A Beta	Integration Developer	Any Salesforce Org: Setup -> Build -> Develop -> Tools 0.58A Beta: https://developer.salesforce.com/page/ForceExplorer

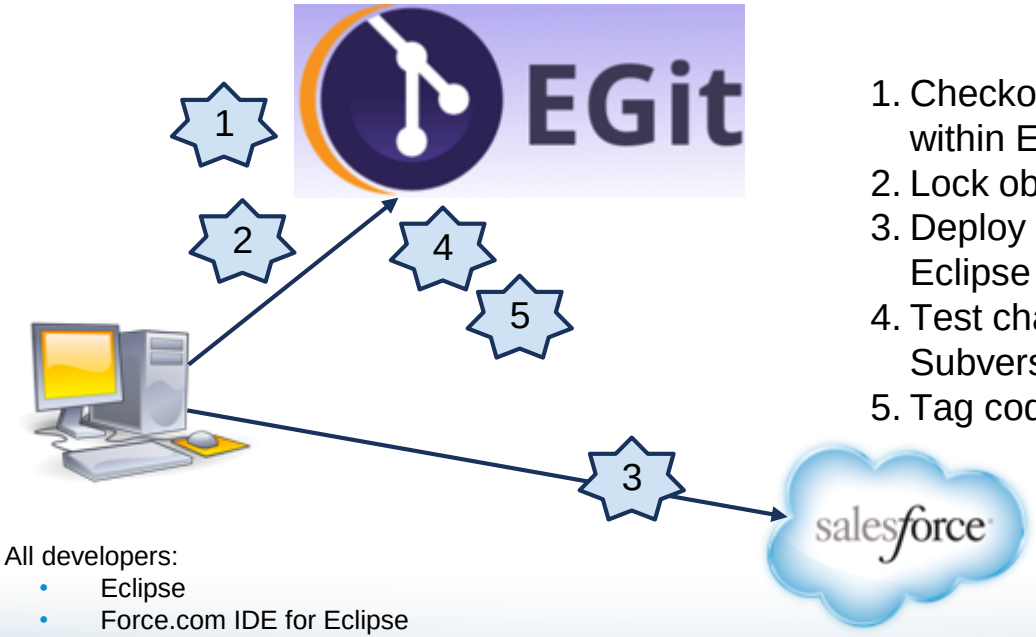
Development Process: Force.com (Eclipse) IDE & Git

Development IDE Install Steps



Development Process

All developers shall checkout code from eGit, develop locally, then save back to eGit



1. Checkout project/branch from Git using eGit plug-in within Eclipse
2. Lock objects/files to be modified
3. Deploy code to Dev Sandbox via Force.com IDE for Eclipse
4. Test changes, and commit/unlock changes back to Subversion
5. Tag code to applicable code drop or Release

All developers:

- Eclipse
- Force.com IDE for Eclipse
- GitHub Plug-in (eGit)

Work Instructions: Install Eclipse

Who:

- Developers

Steps:

- Download Eclipse Neon 4.6
- Follow and accept all the prompts
- Full details can be found at https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/neon3

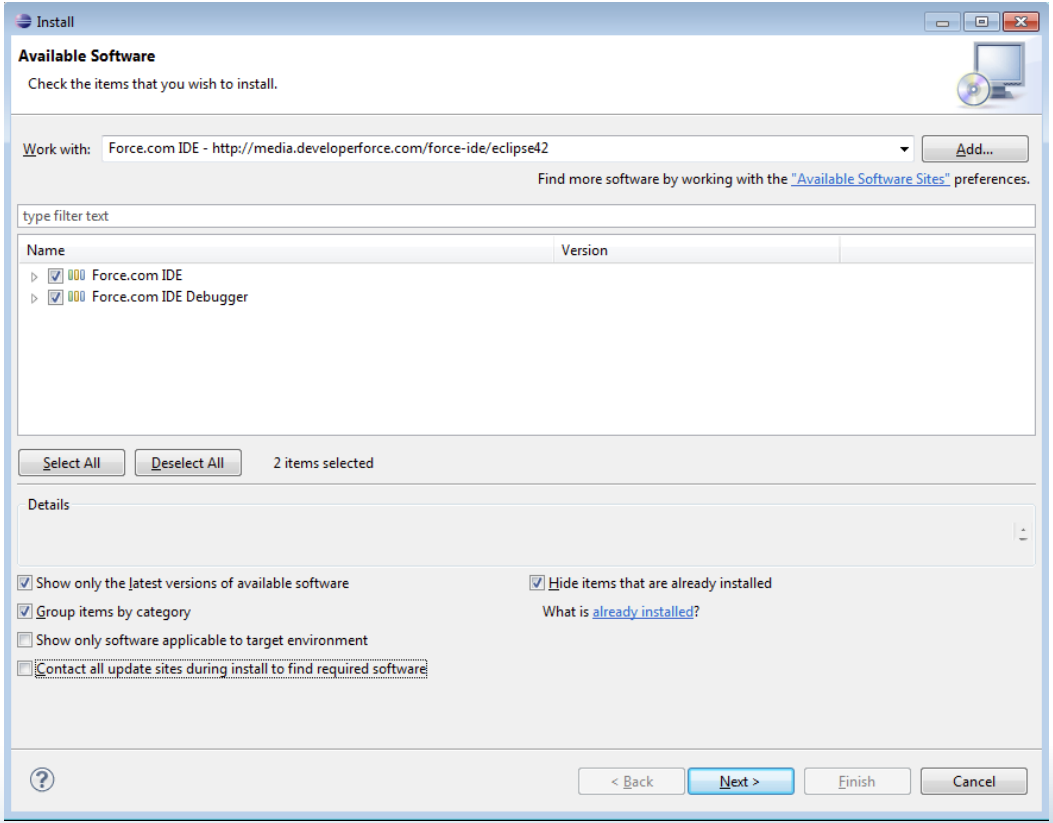
Work Instructions: Install Force.com IDE

Who:

- Developers

Steps:

- In Eclipse, go to Help -> Install New Software
- Click Add and put in the URL: https://developer.salesforce.com/media/force-ide/eclipse45
- Check both Force.com IDE and Force.com IDE Debugger
- Follow and accept all the prompts
- Full details can be found at https://developer.salesforce.com/page/Force.com_IDE_Installation



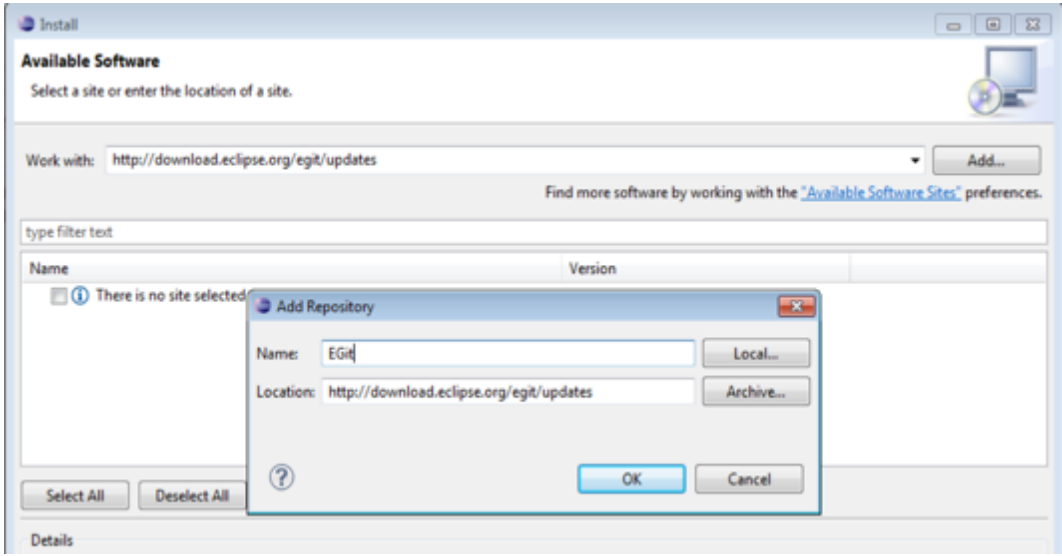
Work Instructions: Install eGit

Who:

- Developers

Steps:

- In Eclipse, go to Help -> Install New Software
- Click Add and put in the URL: http://download.eclipse.org/egit/updates
- Check all the items – Eclipse Git Team Provider, Eclipse Git Team Provider – experimental features, and eGit
- Follow and accept all the prompts
- Full details can be found at http://www.eclipse.org/egit/download/



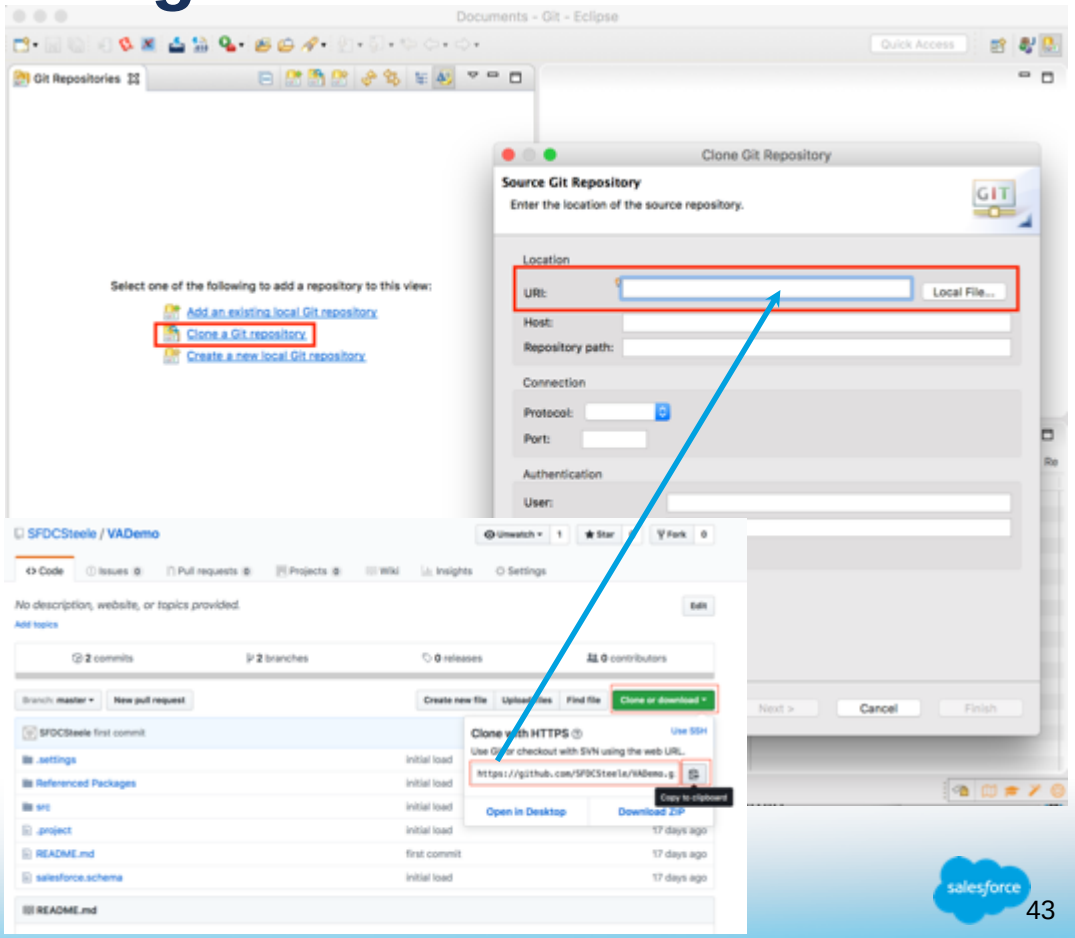
Work Instructions: Configure eGit

Who:

- Developers

Steps:

- In Eclipse, go to Window -> Open Perspective -> Other
- Select ‘Git’ and click OK
- Accept all other defaults and click Finish
- Click the ‘Clone Git Repository’ link and put in the URI provided by your admin
- Place URI from GitHub.com session ***once provided***
- Accept all other defaults and click Finish



Work Instructions: Create Git Branch to work on

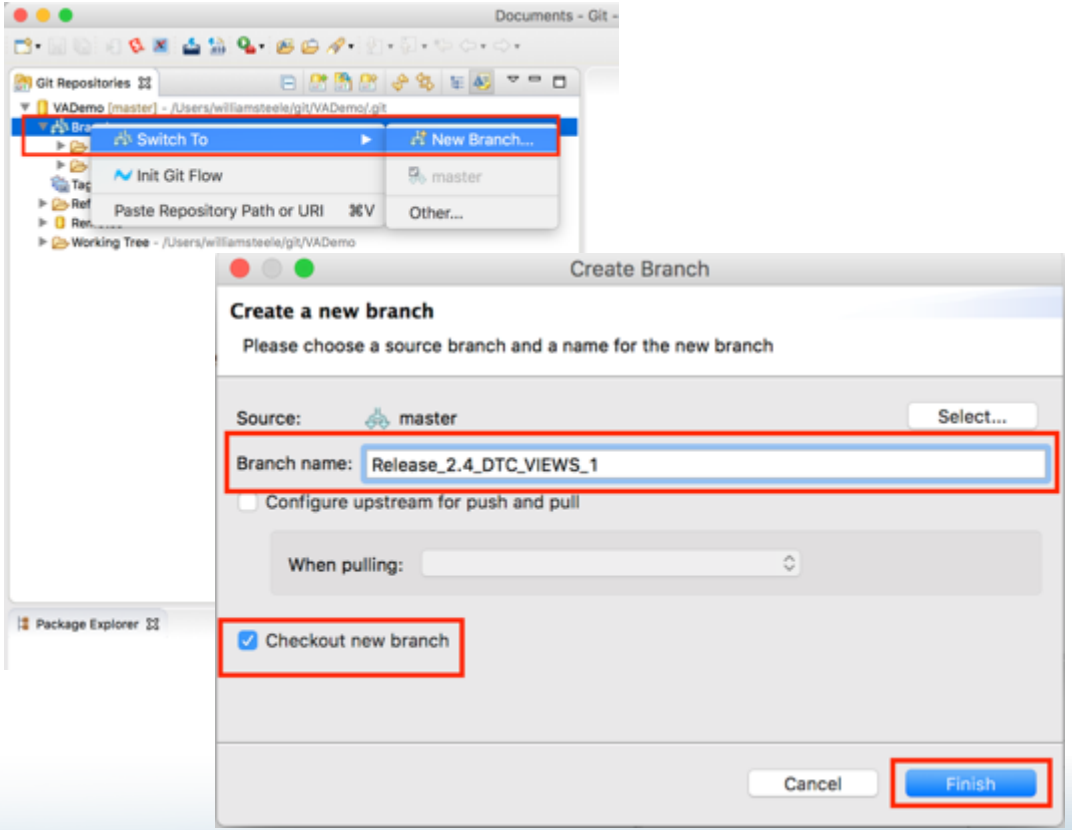
Who:

- Developers

Steps:

- Right click the project, and go to Switch To -> New Branch
- Give the Branch a name, i.e. “Release_2.4_DTC_VIEWS_1”
- Leave the ‘Checkout new branch’ checked
- Click Finish

*****Branch name standards will be provided later**



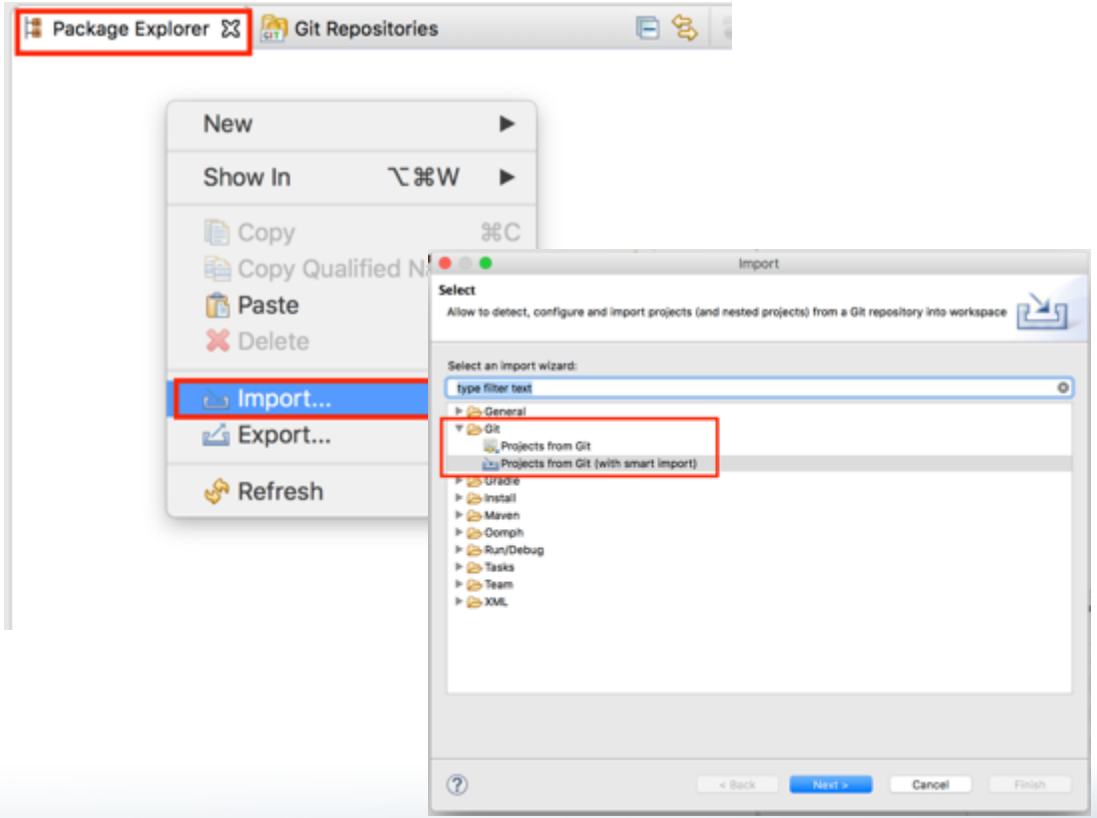
Work Instructions: Create Eclipse Project from GIT (1 of 3)

Who:

- Developers

Steps:

- In Package Explorer, right click and click on ‘Import’
- Select ‘Projects from Git (either option)



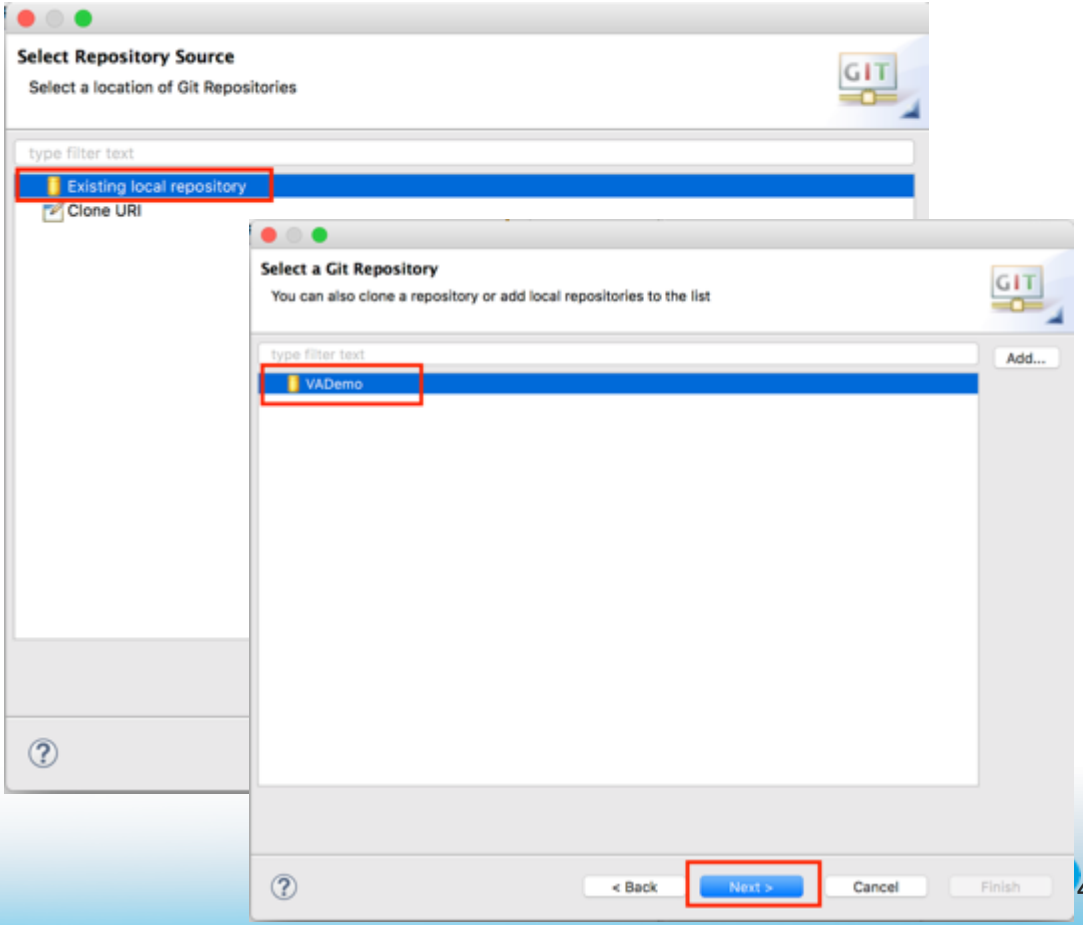
Work Instructions: Create Eclipse Project from GIT (2 of 3)

Who:

- Developers

Steps:

- Select ‘Existing local repository’
- Select the GIT repository that was added earlier



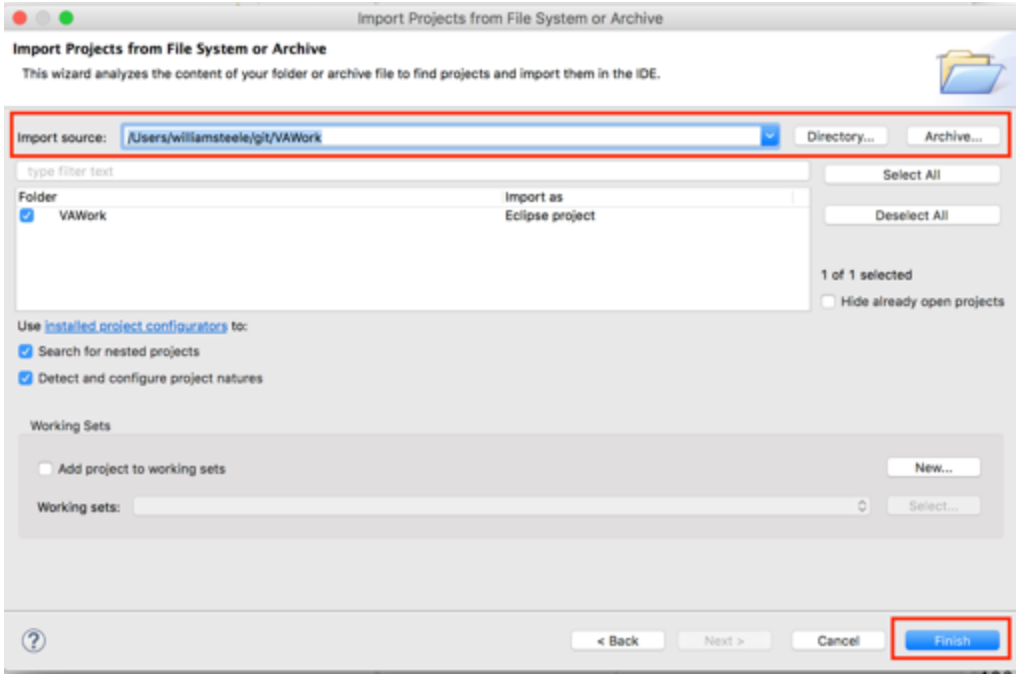
Work Instructions: Create Eclipse Project from GIT (3 of 3)

Who:

- Developers

Steps:

- Ensure “Import Source” is correct
- Validate other options and click Finish



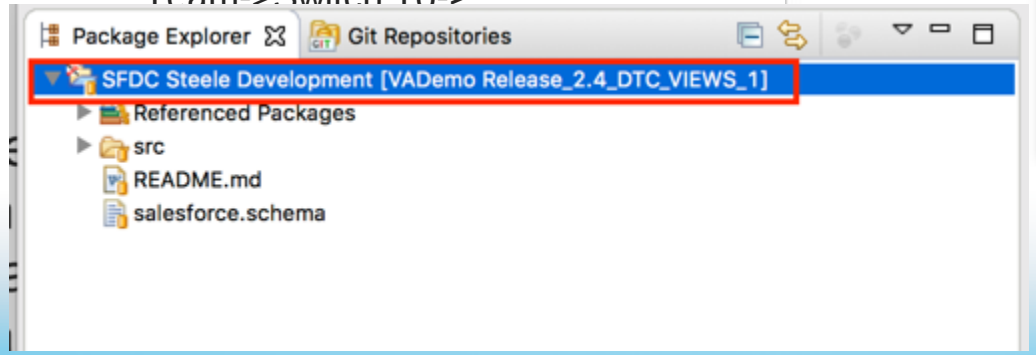
Work Instructions: Switch to Specific Branch for Development

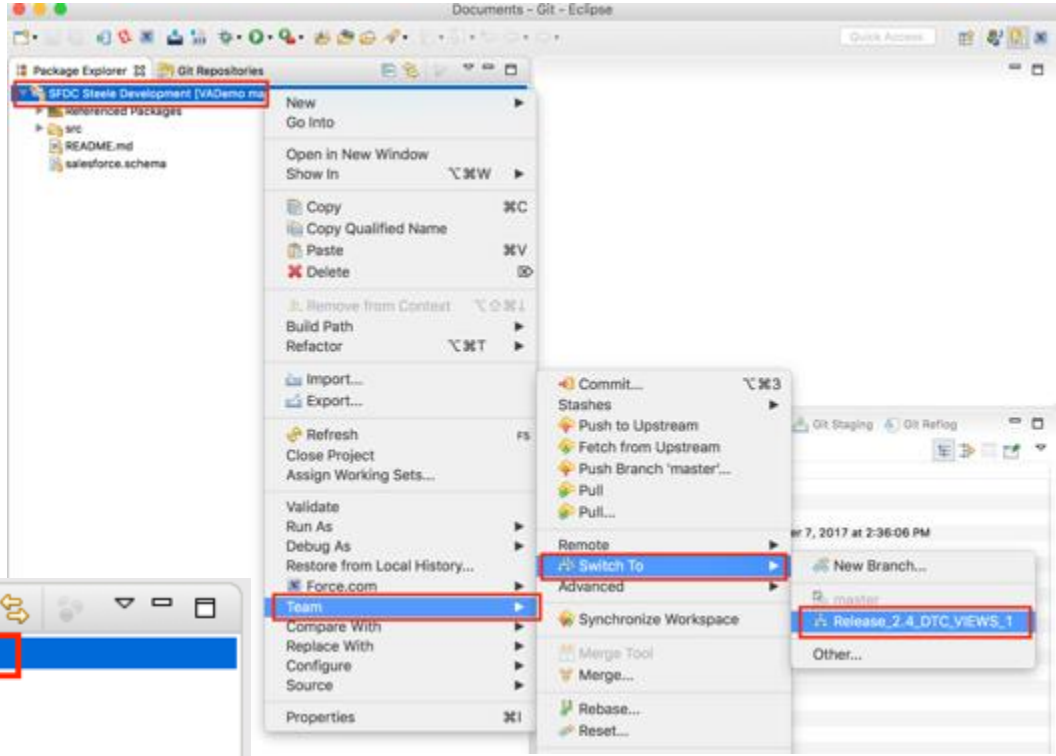
Who:

- Developers

Steps:

- View in the brackets whether you're on the master branch (VADemo master) or a child branch created earlier (VADemo Relase_2.4_DTC_VIEWS_1)
- To switch to child branch to develop, right click the project, and navigate to Team->Switch To->





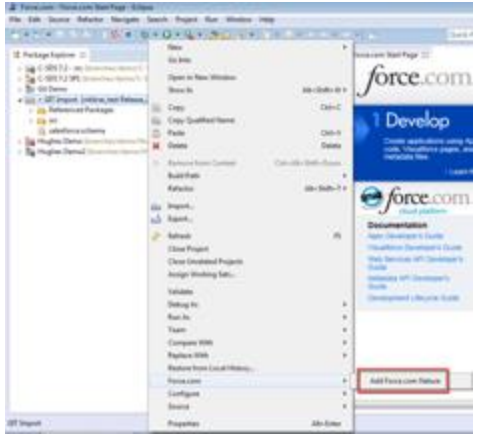
Work Instructions: Associate Sandbox to IDE (1 of 2)

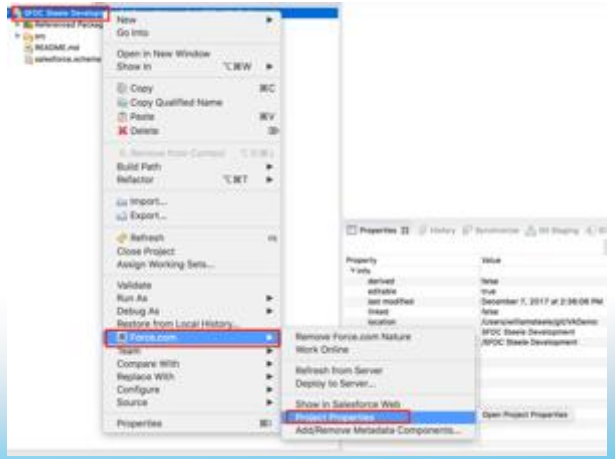
Who:

- Developers

Steps:

- Right click project, and navigate to Force.com->Add Force.com Nature (if not already added)
- Right click project, and navigate to Force.com-> Project Properties





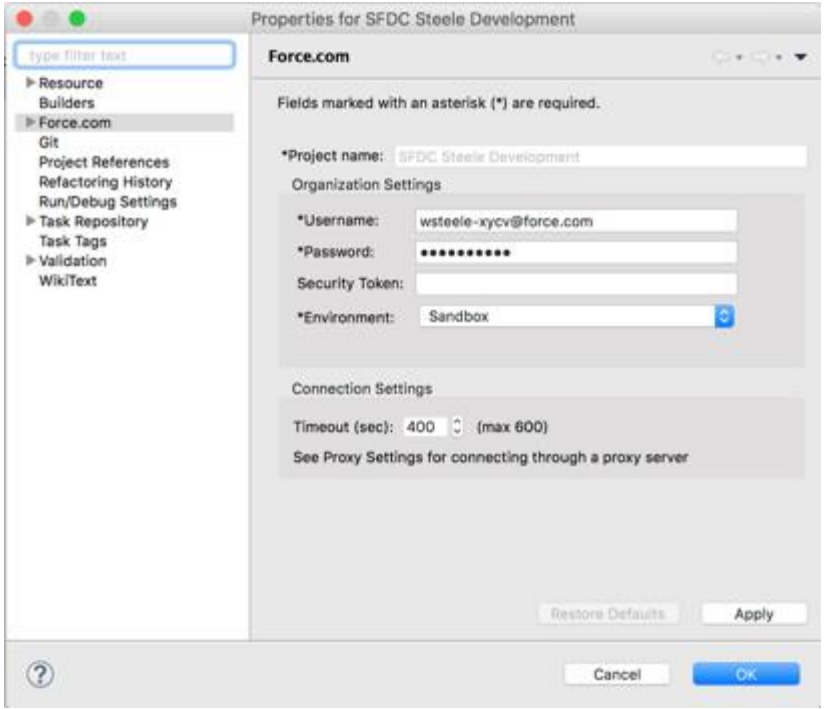
Work Instructions: Associate Sandbox to IDE (2 of 2)

Who:

- Developers

Steps:

- Enter the appropriate username, password, and environment and click OK
- If prompted, do not overwrite



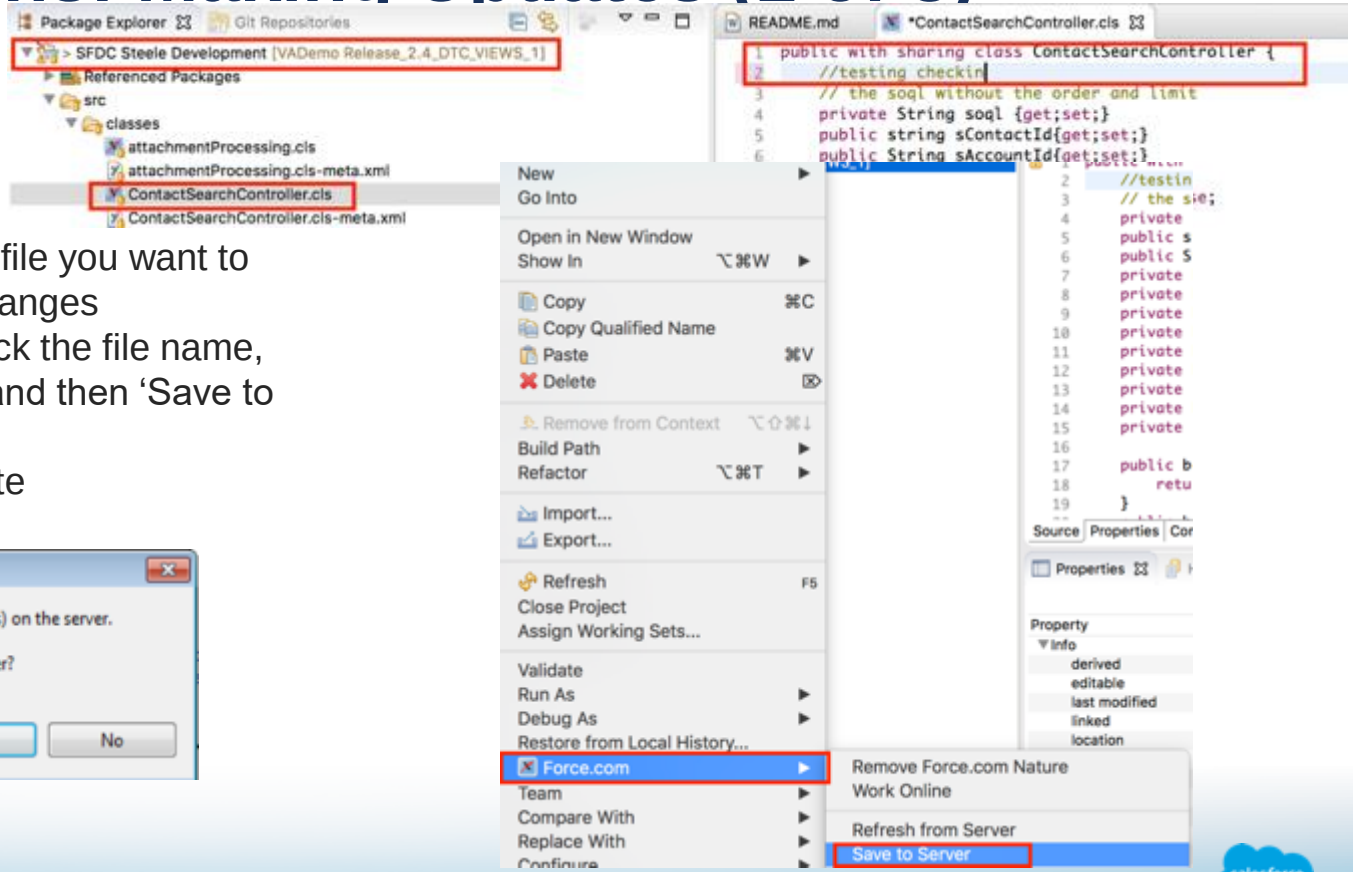
Work Instructions: Making Updates (1 of 3)

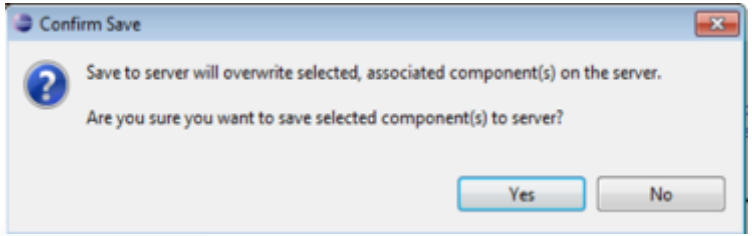
Who:

- Developers

Steps:

- In Eclipse, open the file you want to modify and make changes
- When done, right click the file name, select ‘Force.com’, and then ‘Save to Server’
- Confirm the Overwrite

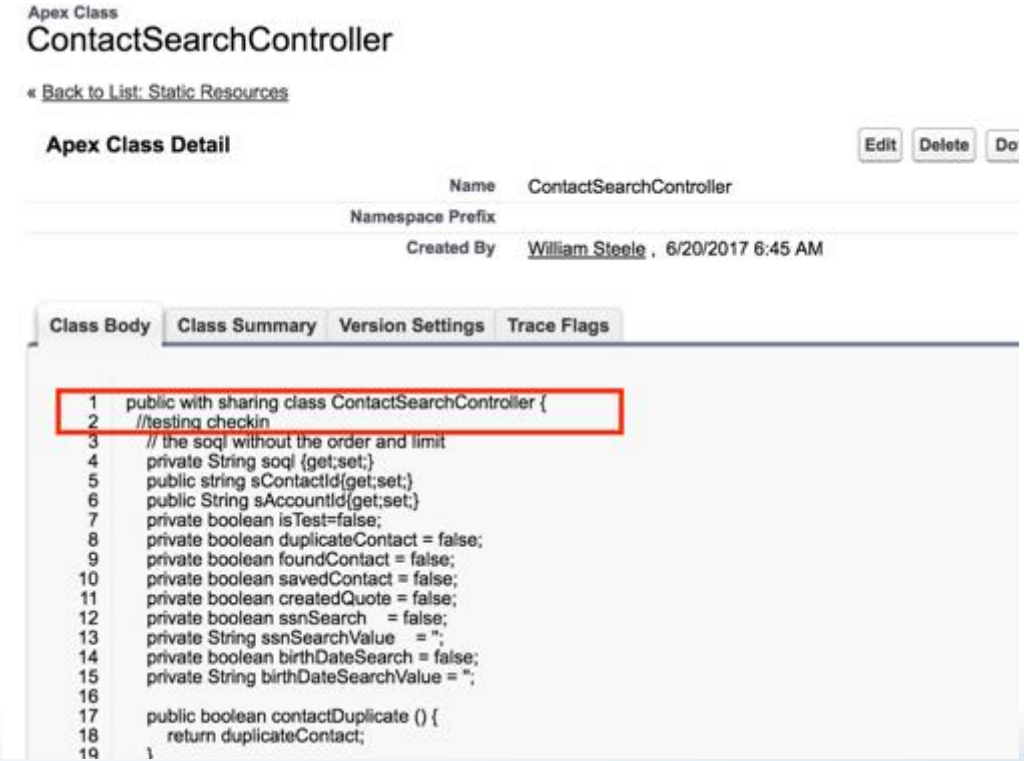




Work Instructions: Making Updates (2 of 3)

Steps:

- Validate that the changes have been sync'ed to the Sandbox Env
- Perform testing as needed
- Create/update code coverage scripts



Work Instructions: Making Updates (3 of 3)

Steps:

- Alternatively, some items are easier to update in Salesforce GUI than in Eclipse, such as Workflows, Validation Rules. In this instance:
  - Lock the respective object via Eclipse (i.e. lock the Account object to make Account workflow or validation rule changes)
  - Make changes in Salesforce GUI
  - Once done, go to Eclipse, right click the object, and go to Force.com->'Refresh from Server'
  - Right click file, and go to Team->Commit

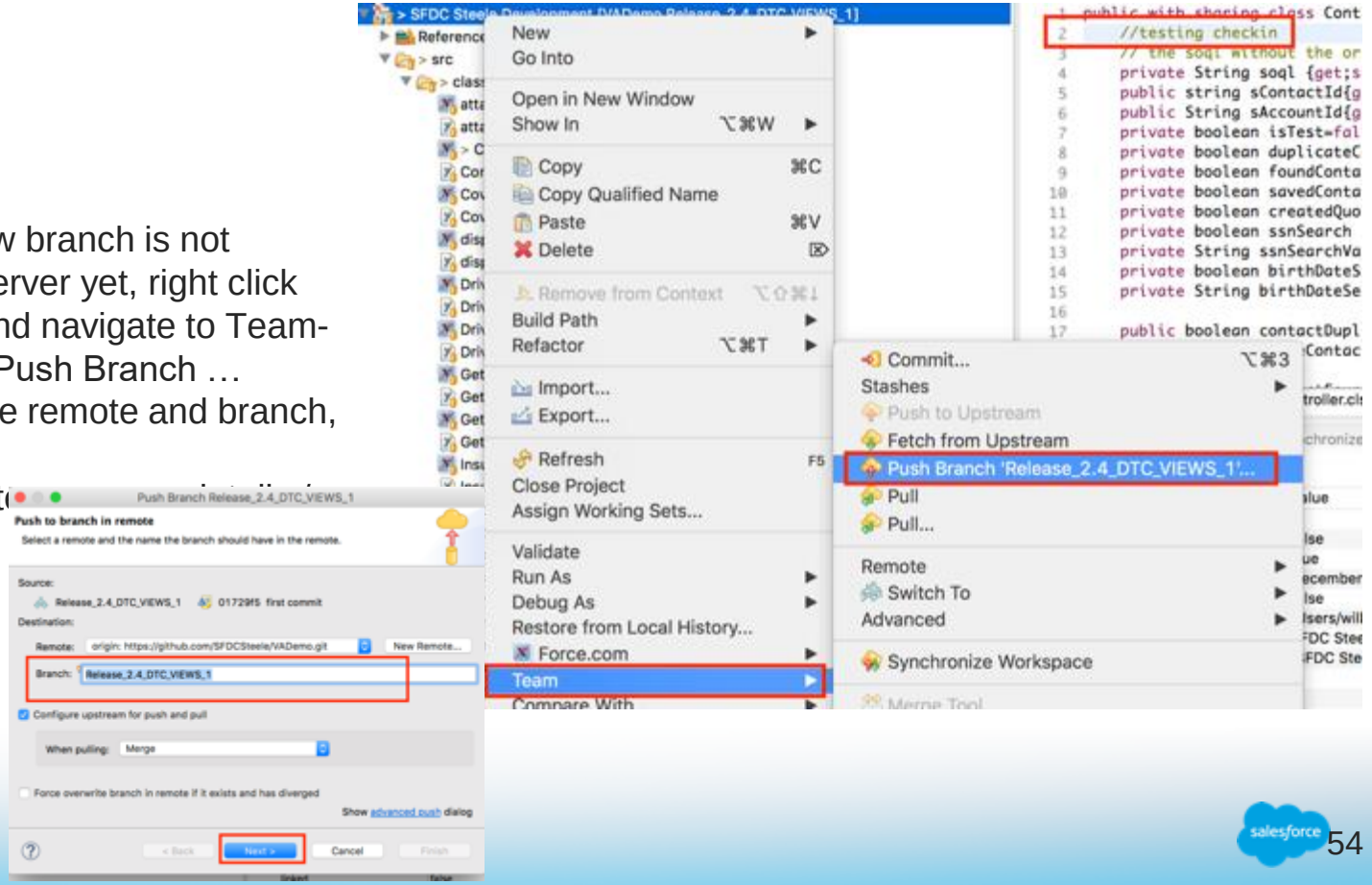
Work Instructions: Commit and Push to Server – Branch Update Only (1 of 3)

Who:

- Developers

Steps:

- In Eclipse, if new branch is not synced to Git Server yet, right click the file name, and navigate to Team-> Repository-> Push Branch …
- Select applicable remote and branch, etc
- Enter appropriat comments
- Click Finish



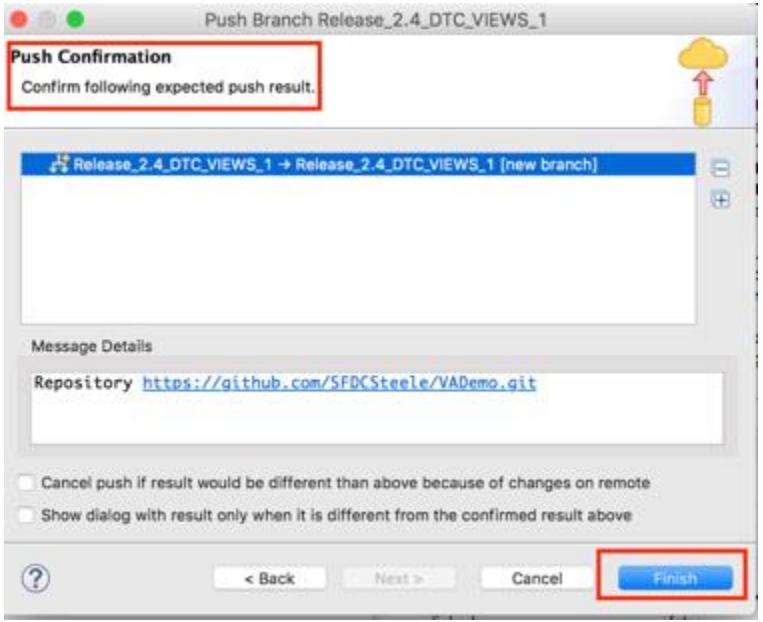
Work Instructions: Commit and Push to Server - Branch Update Only (2 of 3)

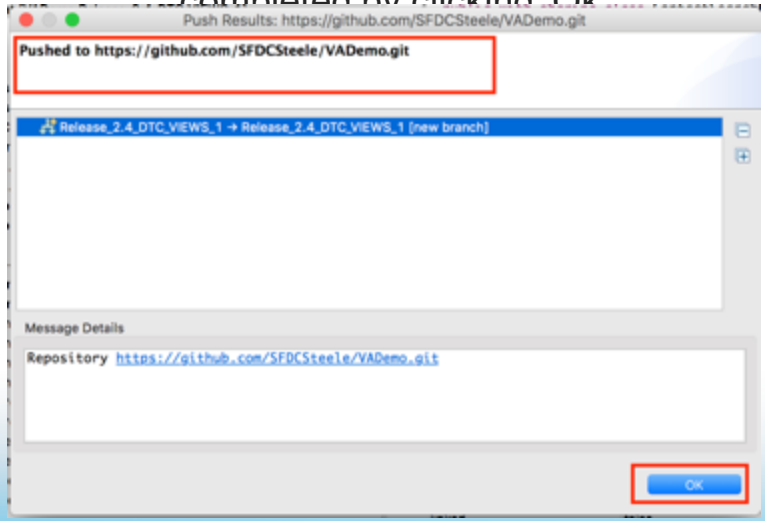
Who:

- Developers

Steps:

- Finalize/confirm the push by clicking ‘Finish’
- Acknowledge the push has completed by clicking ‘Ok’





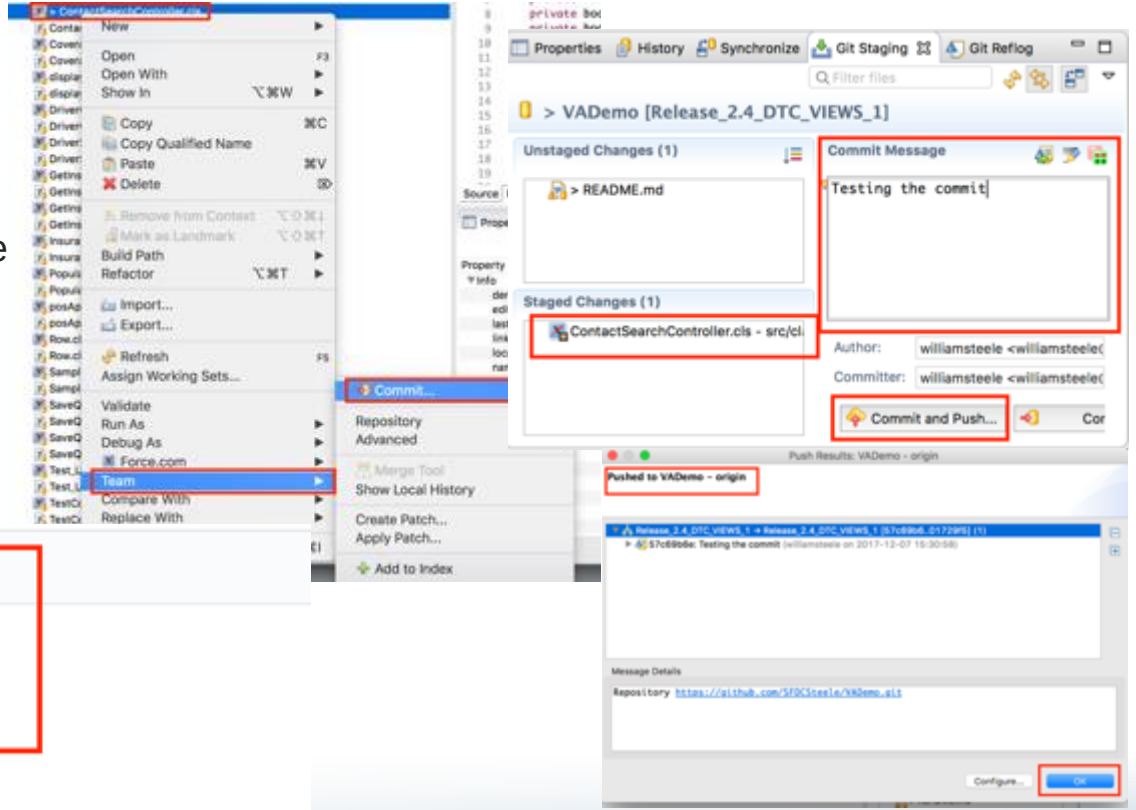
Work Instructions: Commit and Push to Server - Branch Update Only (3 of 3)

Who:

- Developers

Steps:

- In Eclipse, right click the file name and navigate to Team-> Commit
- Add Commit Comments
- Click on ‘Commit and Push’
- Verify that update has been uploaded




```
298 lines (269 sloc)  11.1 KB
1   public with sharing class ContactSearchController {
2           //testing checkin
3           // the soql without the order and limit
4           private String soql {get;set;}
5       public string sContactId{get;set;}
6       public String sAccountId{get;set;}
7       private boolean isTest=false;
8       private boolean duplicateContact = false;
9       private boolean foundContact = false;
```

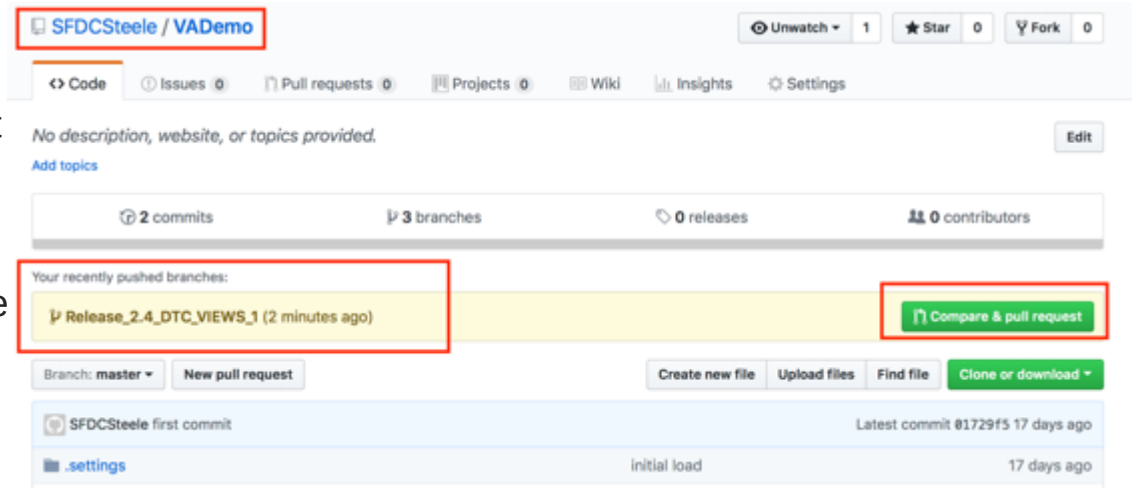
Work Instructions: Sync to Main Branch - Pull Request (1 of 3)

Who:

- Developers or Team Development Coordinator

Steps:

- On browser, navigate to applicable GitHub.com URL
- Click on ‘Compare & pull request’ button



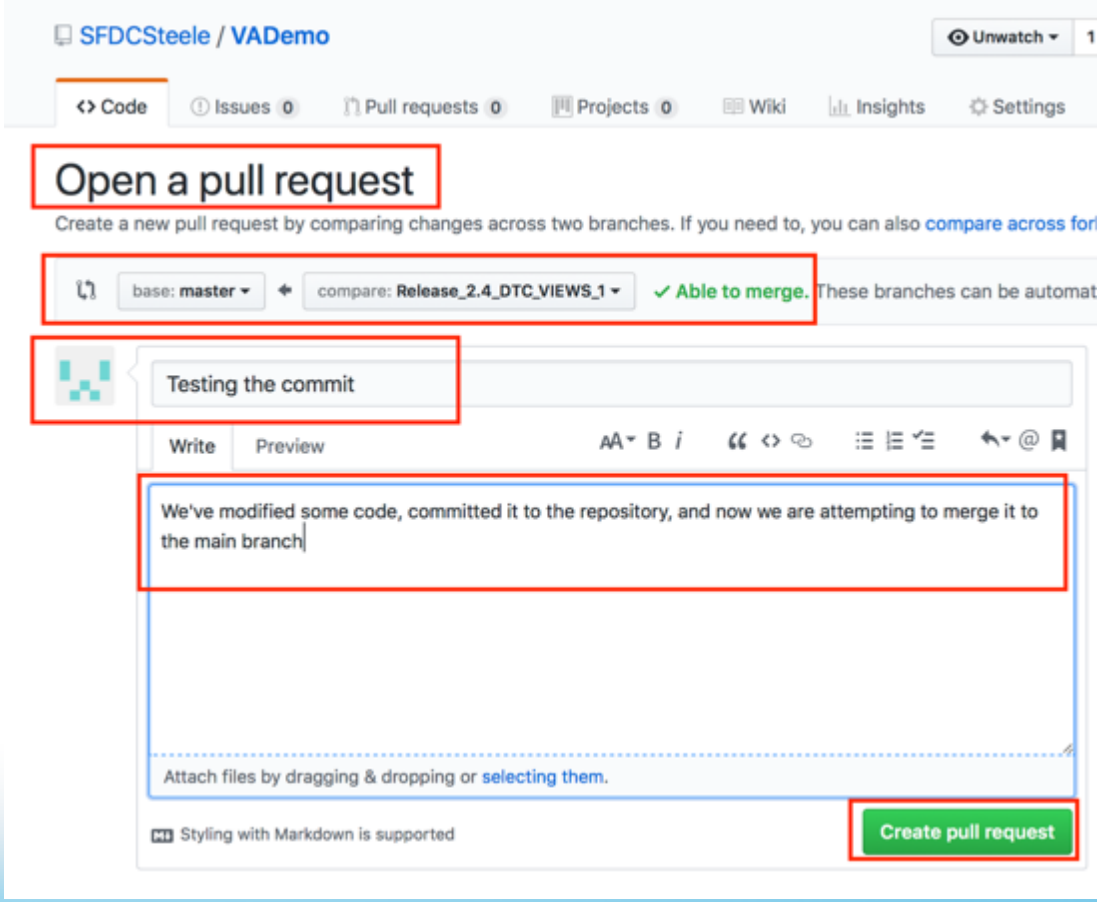
Work Instructions: Sync to Main Branch - Pull Request (2 of 3)

Who:

- Developers or Team Development Coordinator

Steps:

- Click on ‘Create a pull request’
- Set the source (i.e. feature branch) and the target (i.e. main dev branch)
- Put in comments for reviewer
- Select specific reviewer if applicable, else leave blank
- Click ‘Create pull request’
- Wait or seek out for someone to approve



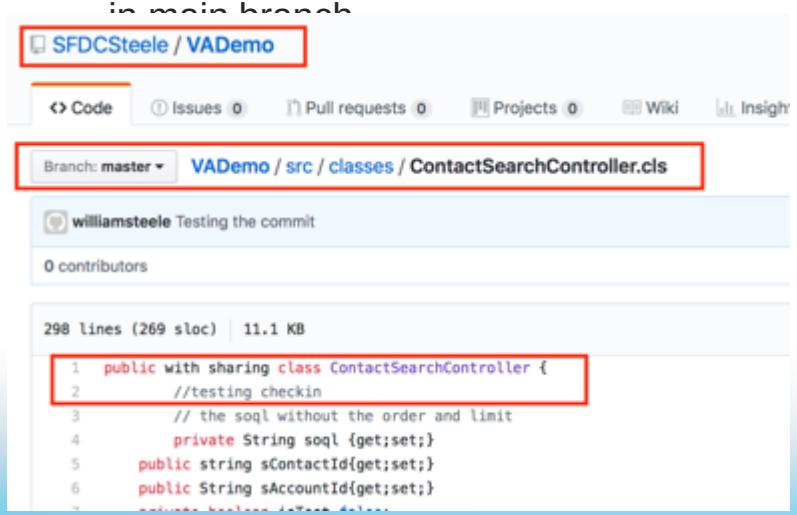
Work Instructions: Sync to Main Branch - Pull Request (3 of 3)

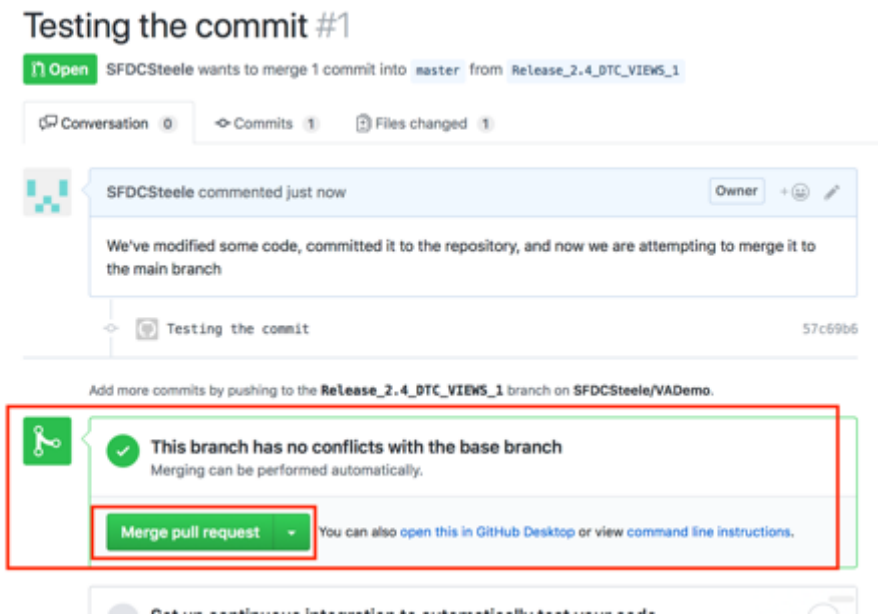
Who:

- Developers

Steps:

- If approved, approver or original developer can click on ‘Merge’ to merge code to main branch
- Verify that change has been updated in main branch





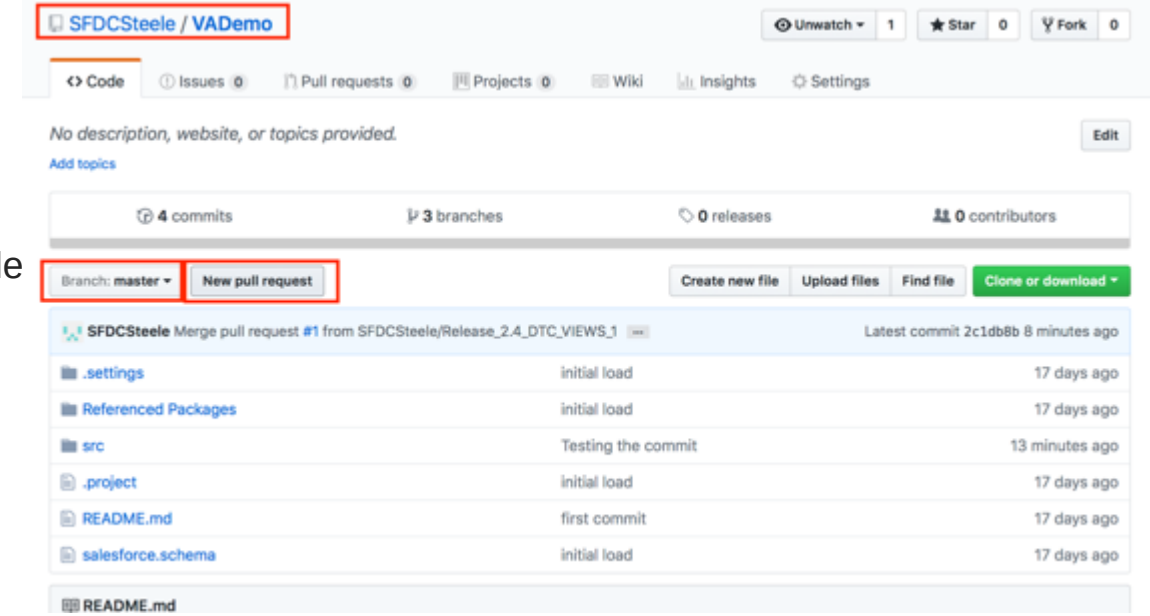
Work Instructions: Approve a Pull Request (1 of 2)

Who:

- Developers (Peers or leads)

Steps:

- On browser, navigate to applicable GitHub.com URL
- Click on ‘Pull Request’ button
- Drill down into pull request



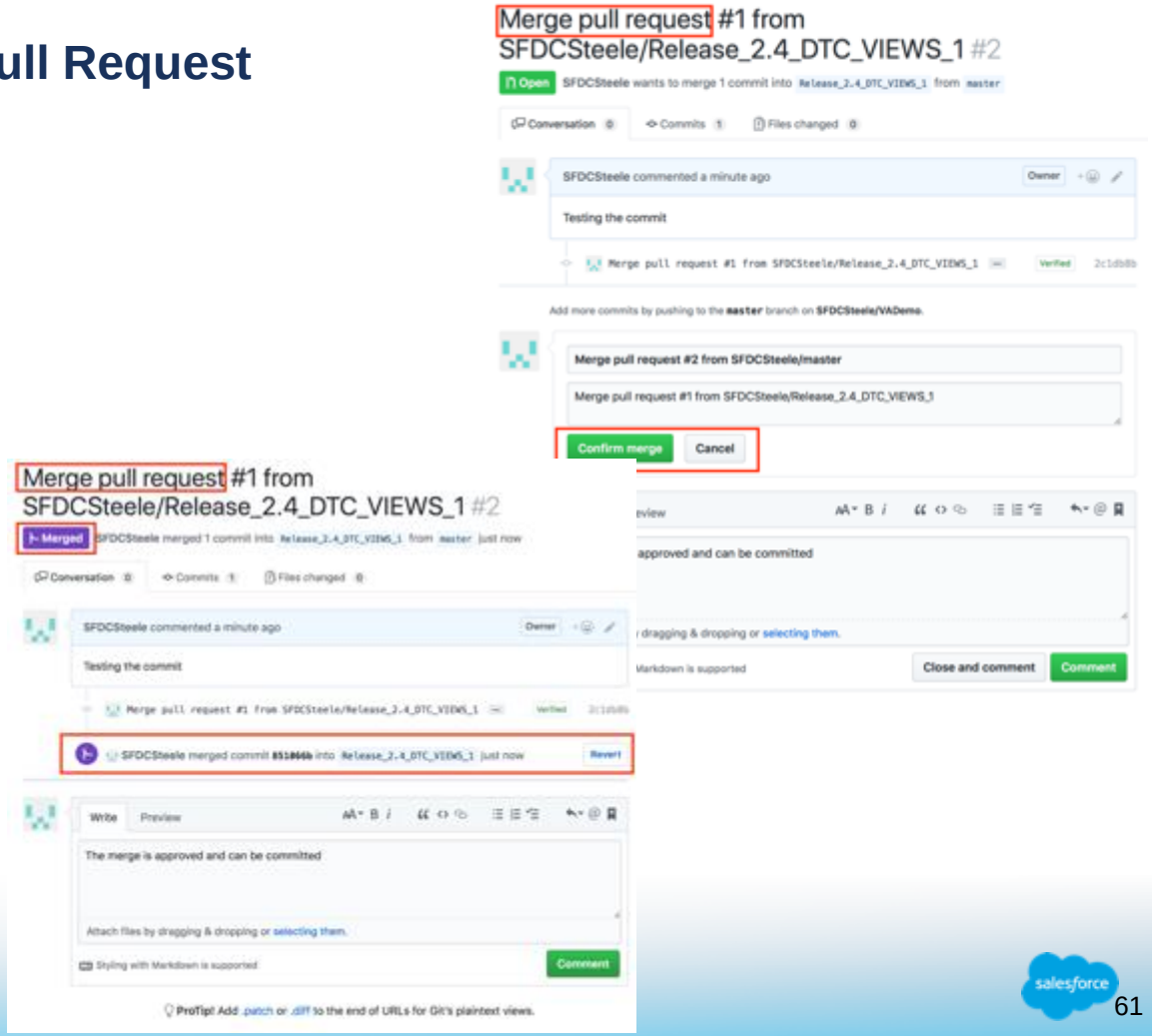
Work Instructions: Approve a Pull Request (2 of 2)

Who:

- Developers

Steps:

- Review details of change
- Click on ‘Approve’ or ‘Decline’
- If approved, approver or original developer can click on ‘Merge’



Repository Management

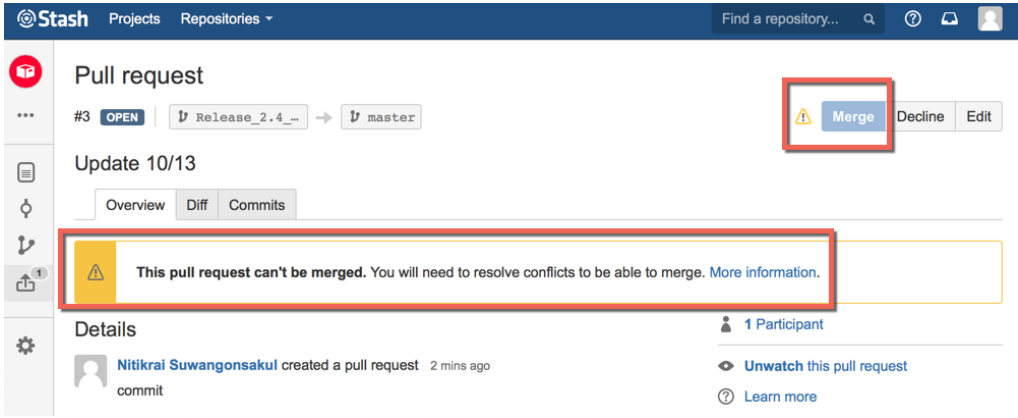
Work Instructions: Merge Conflicts (1 of 4)

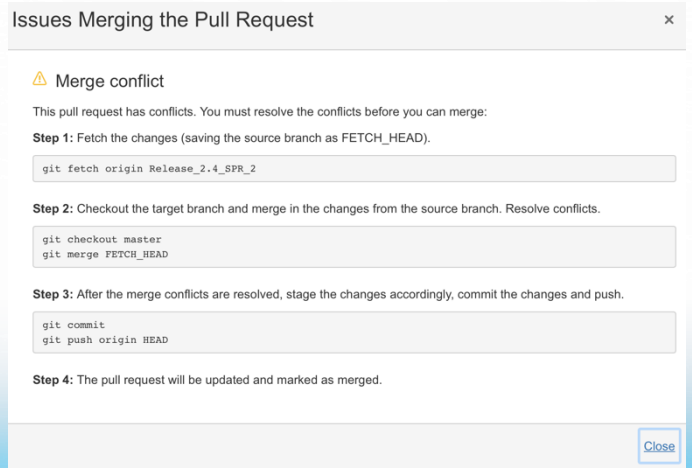
Who:

- Developers

Steps:

- When two developers try to “Pull”/Update the same file on the main branch, the first will succeed while the 2nd will result in an error
- Click on ‘More Information’ to see steps needed





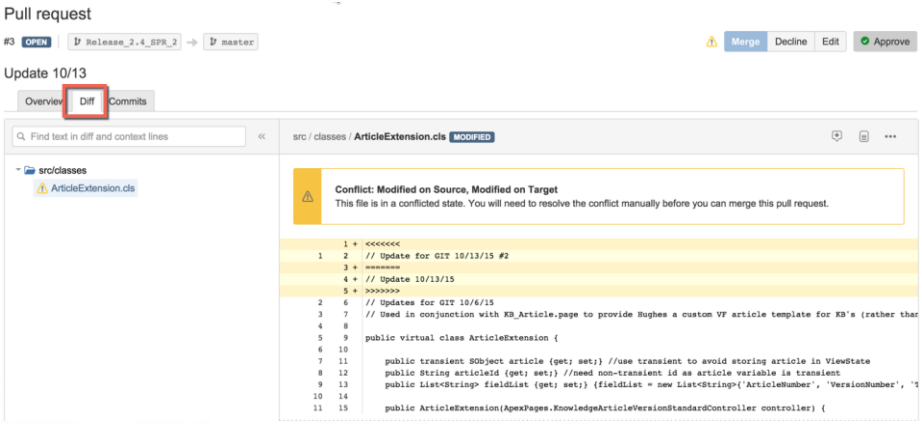
Work Instructions: Merge Conflicts (2 of 4)

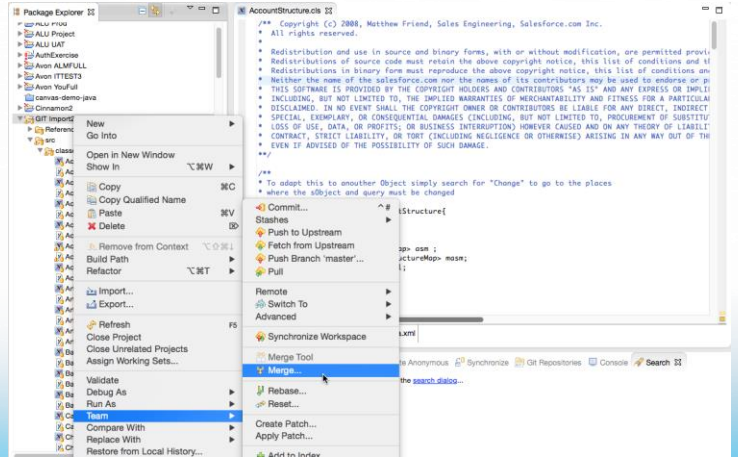
Who:

- Developers

Steps:

- View the conflict by clicking on the ‘Diff’ tab
- Within Eclipse, switch to the master branch by right clicking the project/workspace and going to Team->Switch To->master
- Go to Team->Merge
- Select the Local Branch to merge
- Click Merge





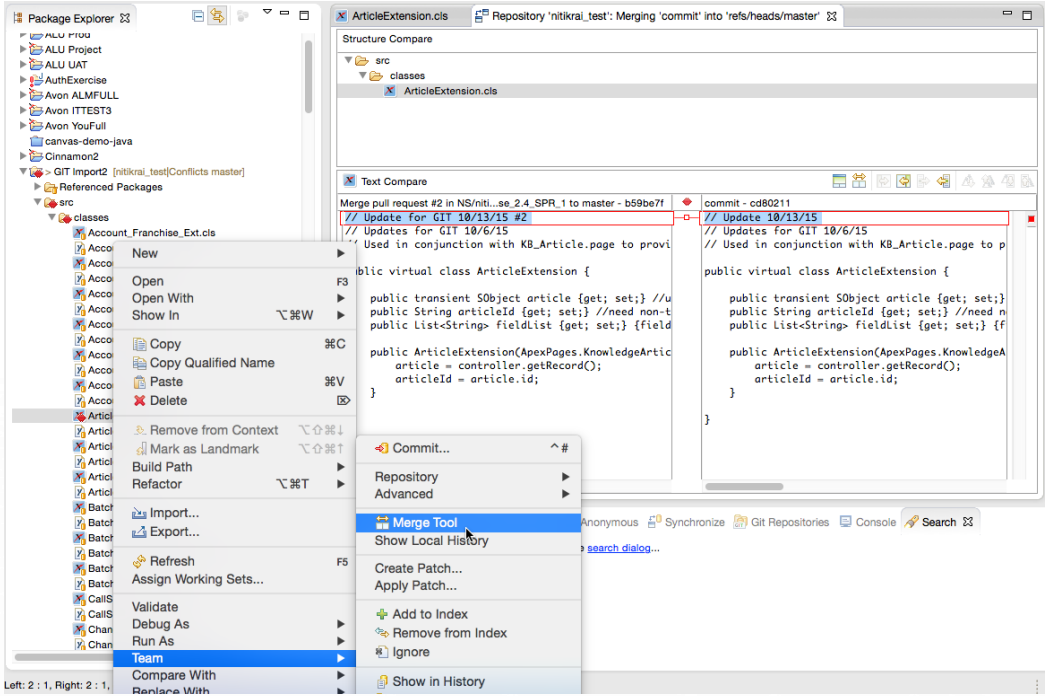
Work Instructions: Merge Conflicts (3 of 4)

Who:

- Developers

Steps:

- Go to the file with the conflict (denoted by red dots), right click, and go to Team->Merge Tool
- Make the changes needed



See full details in:
http://wiki.eclipse.org/EGit/User_Guide#Resolving_a_merge_conflict

Work Instructions: Merge Conflicts (4 of 4)

Who:

- Developers

Steps:

- Save the file, then right click file Team->Add to Index
- Select Commit and Push
- View in Git/Stash that Pull Request has been completed

See full details in:
http://wiki.eclipse.org/EGit/User_Guide#Resolving_a_merge_conflict

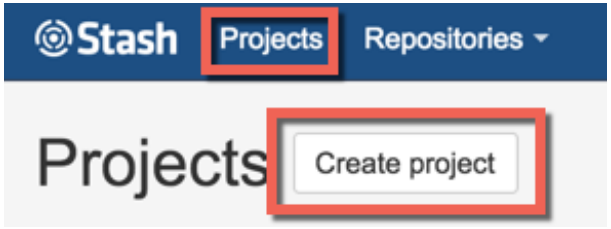
Work Instructions: Create Project

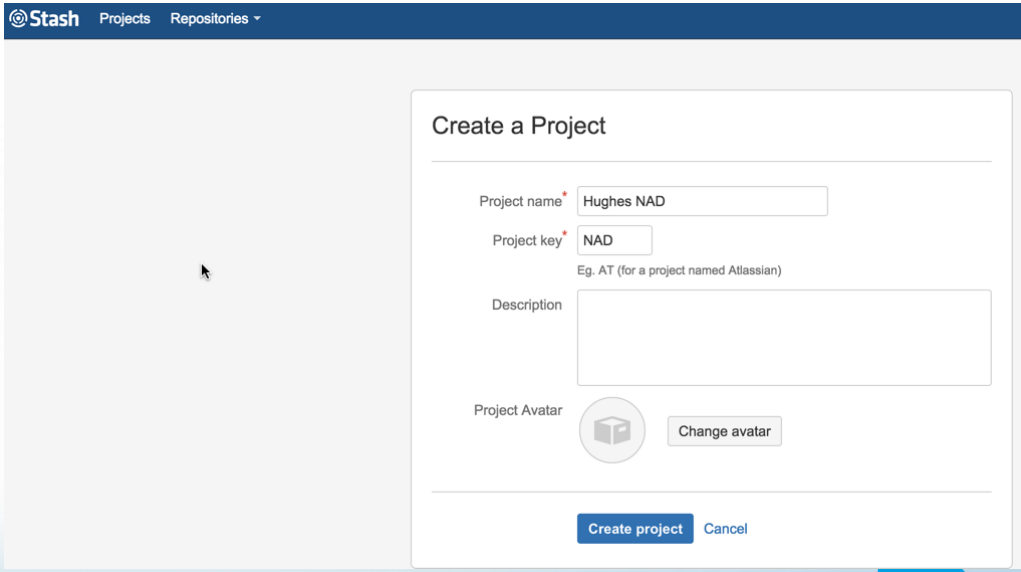
Who:

- Admin

Steps:

- Create new projects for each Trunk / Org (NAD, Brazil, Yahsat, or Sage)
- Click on the Projects tab and click on ‘Create Project’
- Give it an appropriate name and key
- Click the ‘Create Project’ button





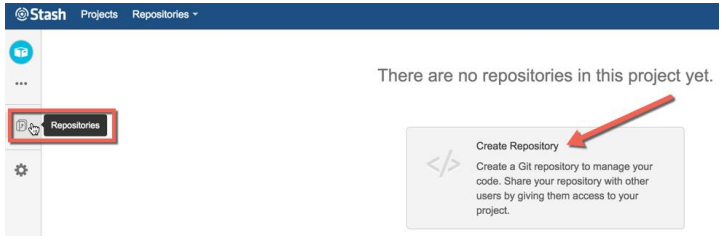
Work Instructions: Create Repository (Trunk or Branch)

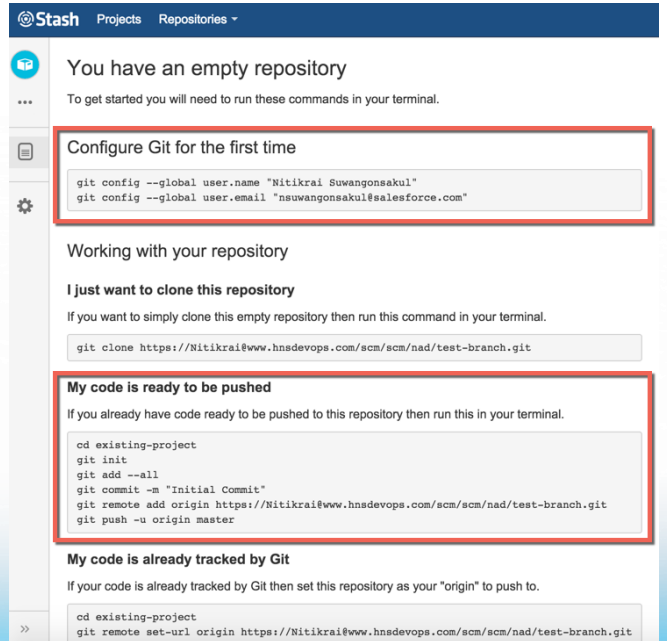
Who:

- Admin

Steps:

- Export metadata from org using Ant retrieve (see Ant slides)
- Click on the ‘Repository’ icon within the Project, and click on the ‘Create Repository’ box
- Give the ‘Repository’ a name, and follow the steps provided to upload metadata code from your local (pulled from Ant)





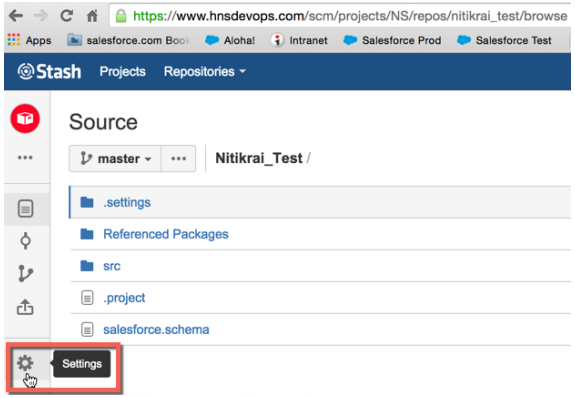
Work Instructions: Set Up Approvers - Pull Request (1 of 2)

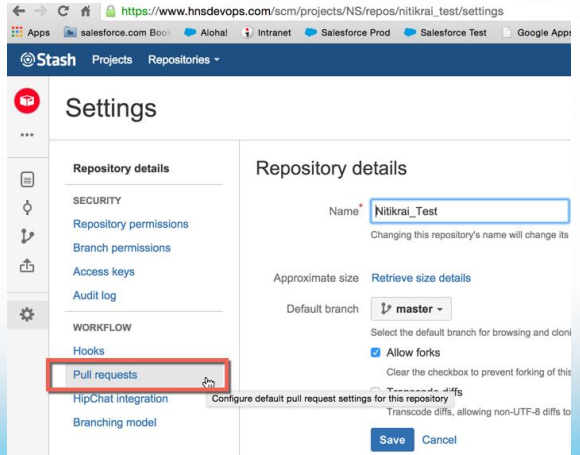
Who:

- Admin

Steps:

- On browser, navigate to applicable Git/Stash URL
- Click on ‘Settings’ button
- Click on ‘Pull requests’





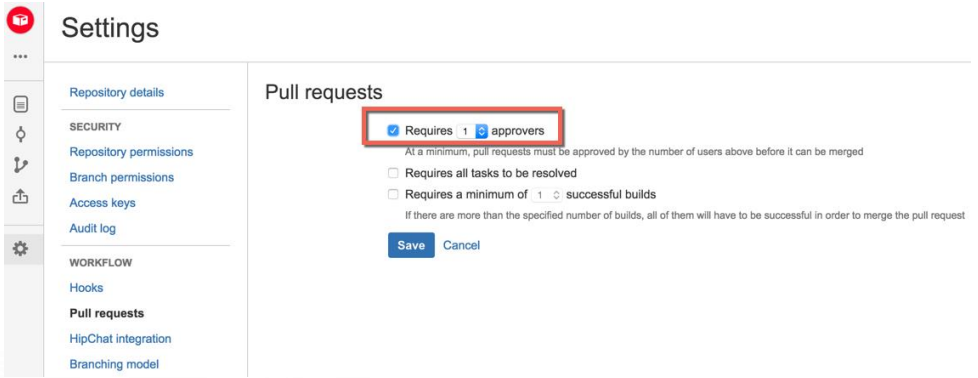
Work Instructions: Set Up Approvers - Pull Request (2 of 2)

Who:

- Admin

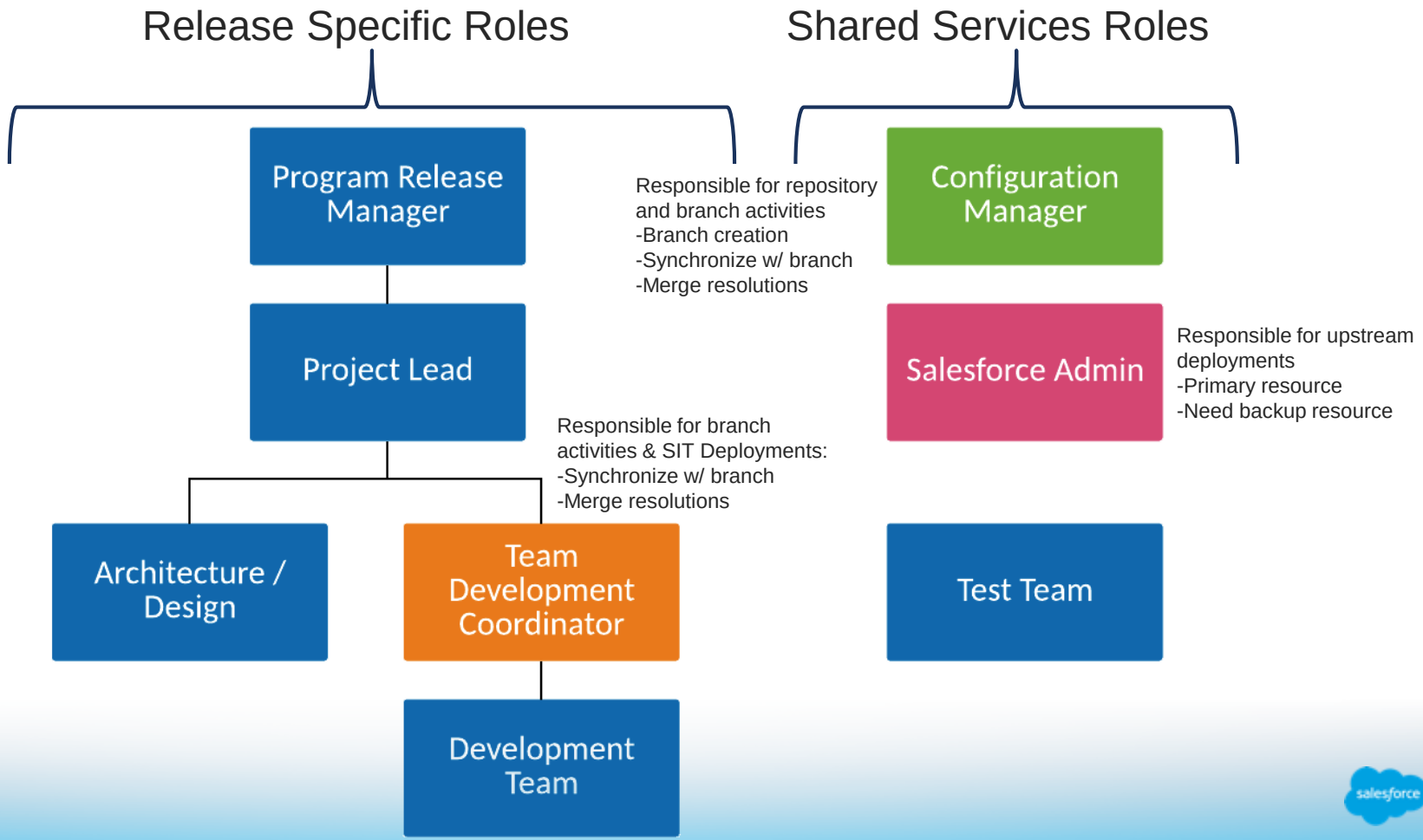
Steps:

- Check the ‘Requires approvers’ and set the number of approvers needed
- Click Save



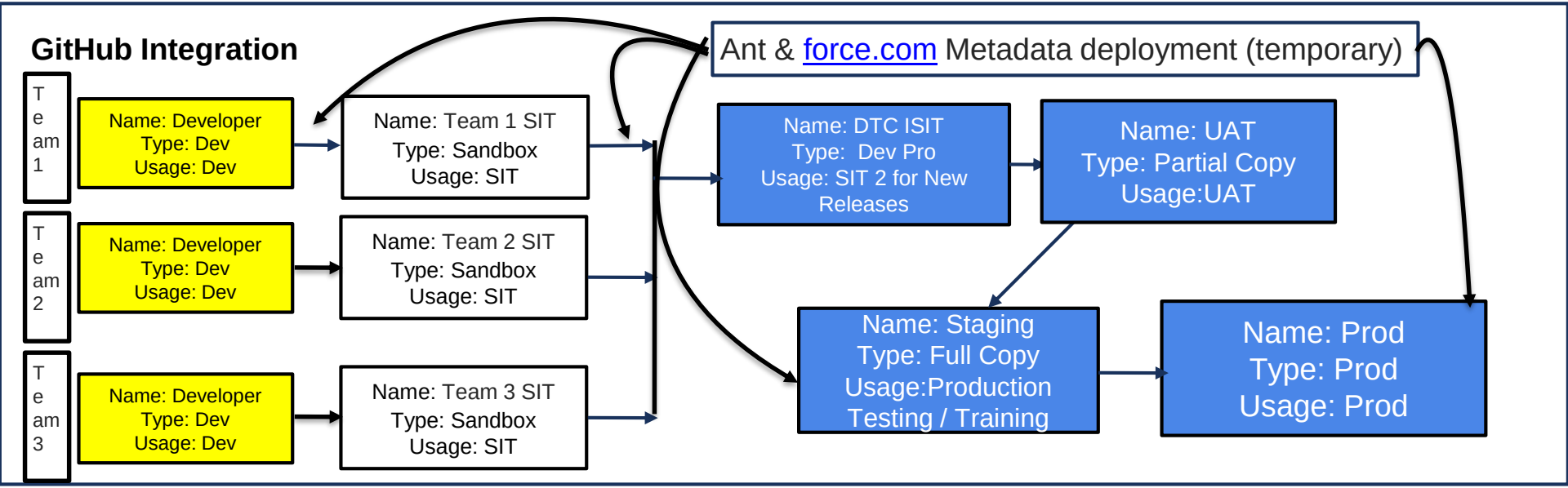
Scripted Deployment

Roles


Release Specific Roles
Shared Services Roles
Program Release Manager
Project Lead
Architecture / Design
Team Development Coordinator
Development Team
Configuration Manager
Salesforce Admin
Test Team
Responsible for repository and branch activities
-Branch creation
-Synchronize w/ branch
-Merge resolutions
Responsible for branch activities & SIT Deployments:
-Synchronize w/ branch
-Merge resolutions
Responsible for upstream deployments
-Primary resource
-Need backup resource

Flow of Code through the VA DTC Environments

Code must move through SDLC of Dev -> SIT -> SIT 2 -> UAT -> Prod

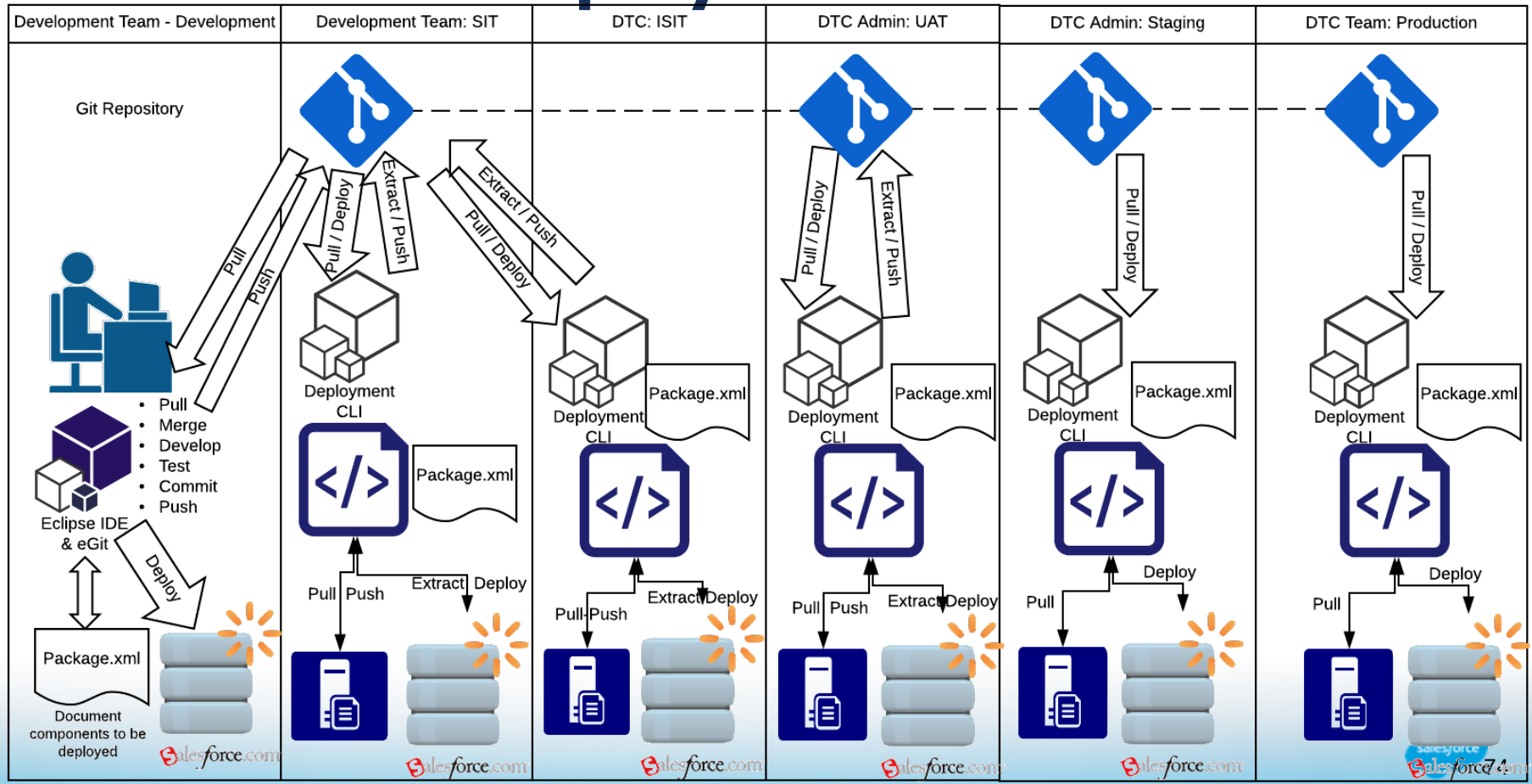


- Code is only pushed from GitHub.com
- Ant and shell scripts will be used to automate the deployments and pull from GitHub.com

Legend: 1 - Emergency fixes to Prod Code Base

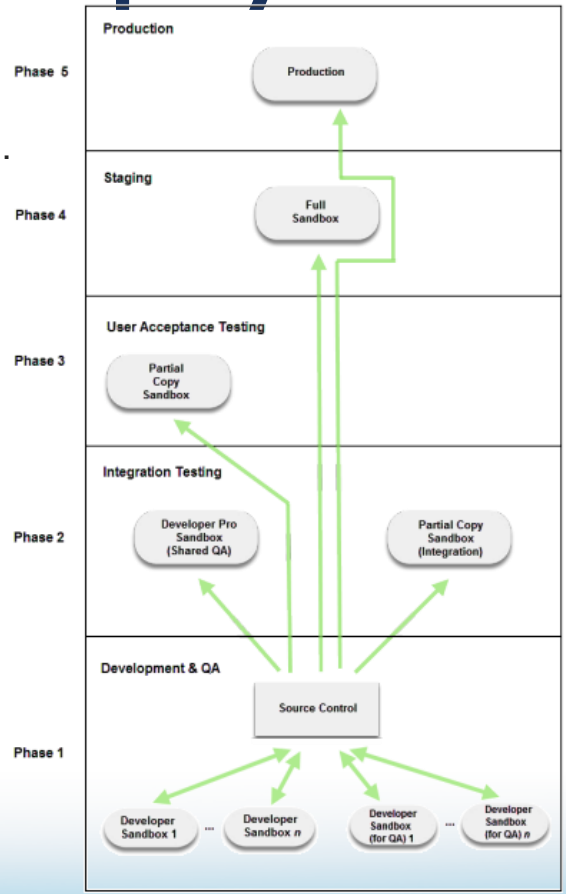
Individual Developer | Prod Code Base

Environment Deployments

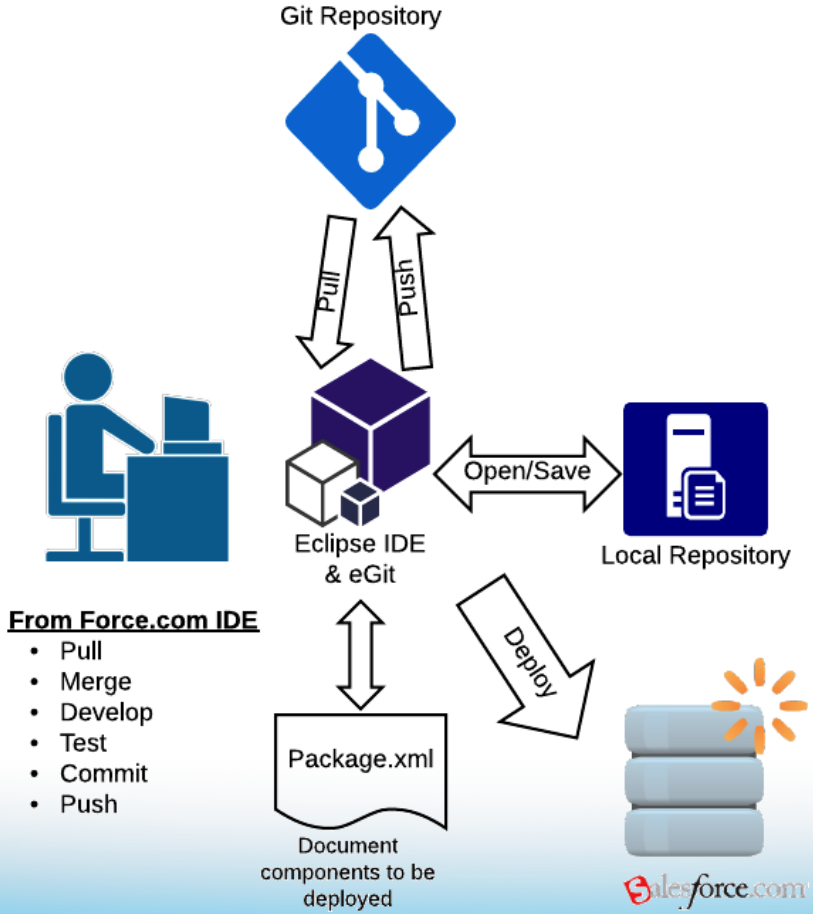


Environment Deployments

A view from a different angle…

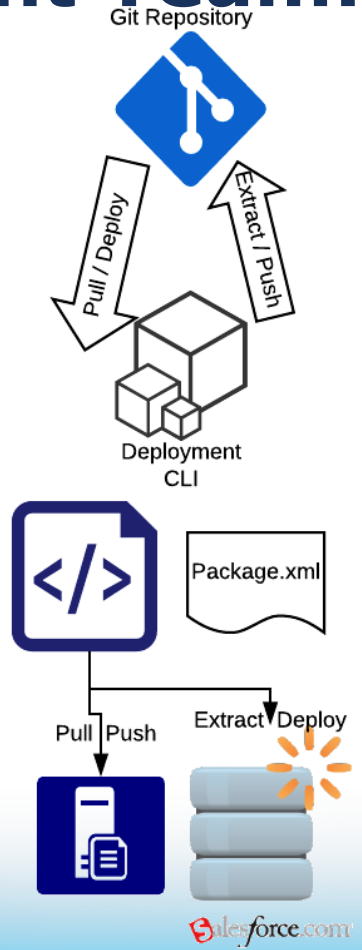


Development Team: Dev Deployment



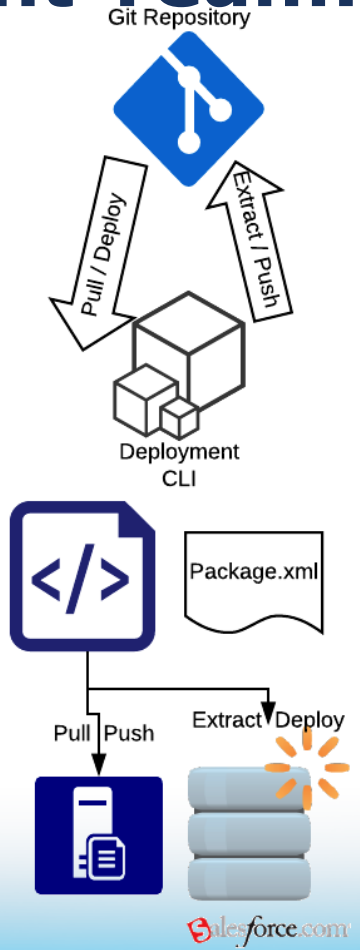
CLI Deployment Action	Deployment Action Description
Senario	New Dev instance
dev-git-clone	Create directory structure, clone the remote repository and the pull that latest source
Senario	Update of the Dev instance
dev-git-pull	Update the current local repository from the remote repository
Senario	Changes made in ORG, need to be checked in (from changesets too)
dev-extract	Extract all configuration from the org based on package.xml to local repository
dev-push	Push changes in the local repository to the remote repository
dev-extract-push	Extract all configuration from the org based on package.xml to local repository and then push to the remote repository
Senario	Deploy local repository to SF org
dev-deploy	Deploy to the SF Org based on the package.xml
dev-deploy-check	Deploy to the SF Org based on the package.xml as a validate only deployment
dev-quick-deploy	Quick deploy the validated deployment
dev-pull-deploy	Pull the latest updates from the remote repository and deploy to the SF Org based on package.xml

Development Team: SIT Deployment



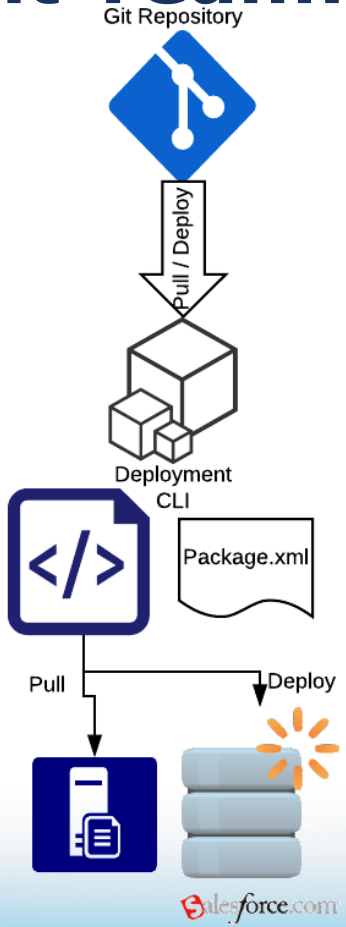
CLI Deployment Action	Deployment Action Description
Senario	New SIT instance
sit-git-clone	Create directory structure, clone the remote repository and the pull that latest source
Senario	Update of the SIT instance
sit-git-pull	Update the current local repository from the remote repository
Senario	Changes made in ORG, need to be checked in (from changesets too)
sit-extract	Extract all configuration from the org based on package.xml to local repository
sit-push	Push changes in the local repository to the remote repository
sit-extract-push	Extract all configuration from the org based on package.xml to local repository and then push to the remote repository
Senario	Deploy local repository to SF org
sit-deploy	Deploy to the SF Org based on the package.xml
sit-deploy-check	Deploy to the SF Org based on the package.xml as a validate only deployment
sit-quick-deploy	Quick deploy the validated deployment
sit-pull-deploy	Pull the latest updates from the remote repository and deploy to the SF Org based on package.xml

Development Team: UAT Deployment



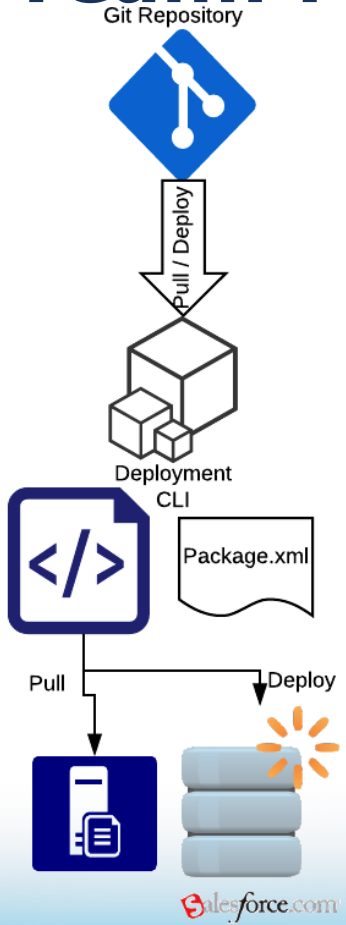
CLI Deployment Action	Deployment Action Description
Senario	New UAT instance
uat-git-clone	Create directory structure, clone the remote repository and the pull that latest source
Senario	Update of the UAT instance
uat-git-pull	Update the current local repository from the remote repository
Senario	Changes made in ORG, need to be checked in (from changesets too)
uat-extract	Extract all configuration from the org based on package.xml to local repository
uat-push	Push changes in the local repository to the remote repository
uat-extract-push	Extract all configuration from the org based on package.xml to local repository and then push to the remote repository
Senario	Deploy local repository to SF org
uat-deploy	Deploy to the SF Org based on the package.xml
uat-deploy-check	Deploy to the SF Org based on the package.xml as a validate only deployment
uat-quick-deploy	Quick deploy the validated deployment
uat-pull-deploy	Pull the latest updates from the remote repository and deploy to the SF Org based on package.xml

Development Team: Staging Deployment



CLI Deployment Action	Deployment Action Description
Senario	New Staging instance
staging-git-clone	Create directory structure, clone the remote repository and the pull that latest source
Senario	Update of the Staging instance
staging-git-pull	Update the current local repository from the remote repository
Senario	Changes made in ORG, need to be checked in (from changesets too)
staging-extract	Extract all configuration from the org based on package.xml to local repository
staging-push	Push changes in the local repository to the remote repository
staging-extract-push	Extract all configuration from the org based on package.xml to local repository and then push to the remote repository
Senario	Deploy local repository to SF org
staging-deploy	Deploy to the SF Org based on the package.xml
staging-deploy-check	Deploy to the SF Org based on the package.xml as a validate only deployment
staging-quick-deploy	Quick deploy the validated deployment
staging-pull-deploy	Pull the latest updates from the remote repository and deploy to the SF Org based on package.xml

Development Team: Production Deployment



CLI Deployment Action	Deployment Action Description
Senario	New Production instance
prod-git-clone	Create directory structure, clone the remote repository and the pull that latest source
Senario	Update of the Production instance
prod-git-pull	Update the current local repository from the remote repository
Senario	Changes made in ORG, need to be checked in (from changesets too)
prod-extract	Extract all configuration from the org based on package.xml to local repository
prod-push	Push changes in the local repository to the remote repository
prod-extract-push	Extract all configuration from the org based on package.xml to local repository and then push to the remote repository
Senario	Deploy local repository to SF org
prod-deploy	Deploy to the SF Org based on the package.xml
prod-deploy-check	Deploy to the SF Org based on the package.xml as a validate only deployment
prod-quick-deploy	Quick deploy the validated deployment
prod-pull-deploy	Pull the latest updates from the remote repository and deploy to the SF Org based on package.xml

Developer Profiles and Activities

1. Developer: no/minimal team – performs the tasks above stand-alone (support from DTC)
2. Developer: part of a team – Team Development Coordinator performs tasks above
3. Developer: Full IDE – performs all functions within the Force.com IDE (specified previously)

Developer Deployment Setup

For a developer or a Team Development Coordinator to perform the tasks above, the setup below much be completed:

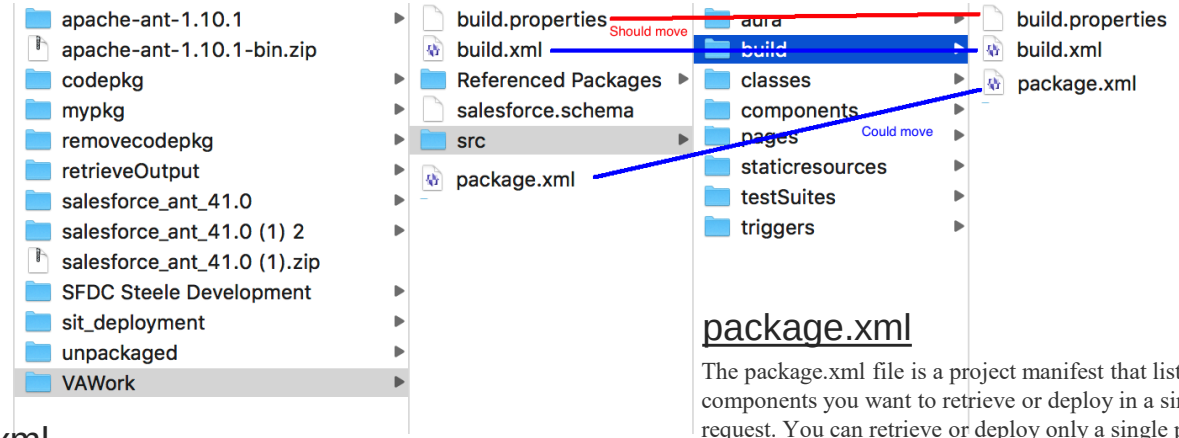
- Download and install the GIT Command Line Interface (CLI) from https://git-scm.com/downloads
- Install Force.com Migration tool (and ANT) from https://developer.salesforce.com/docs/atlas.en-us.daas.meta/daas/forcemigrationtool_install.htm
- Set environment variables to meet your environment-see the example below:

export ANT_HOME=/Users/<user>/Documents/workspace/apache-ant-1.10.1/bin
export PATH=/Users/<user>/Documents/workspace/apache-ant-1.10.1:/Users/<user>/Documents/workspace/salesforce_ant_41.0/ant-salesforce.jar:/Users/<user>/Documents/workspace/apache-ant-1.10.1/bin:/Users/<user>/Documents/workspace/apache-ant-1.10.1/lib:$

Force.com Migration Tool Artifacts

build.properties

The build.properties file contains all the properties used in the build.xml. Defaults can be replace by environment specific values, removing the need to enter a list of parameters each time an ant command is issued. This file can be copied to the root directory of your Eclipse workspace and modified, so that there is not worries of it being overwritten or being flagged to need to be checked-in.



build.xml

The build.xml file specifies a series of commands to be executed by Ant. Within the build.xml file are named targets that process a series of commands when you run Ant with a target name. The following parameters can be set for each `<sf:retrieve>` target:

There are a number of VA specific commands that have been created to support the deployment process at the VA.

This file can also be moved to your workspace root if it needs to be customized. Leaving it in the src/build directory allows you to receive any new or updated targets that will be provided.

package.xml

The package.xml file is a project manifest that lists all the components you want to retrieve or deploy in a single request. You can retrieve or deploy only a single package at a time.

Using this file instead of changesets, teams will be able to communicate all the metadata components they wish to be deployed, thus providing the ability to alway perform a “delta” deployment.

These files can be merged into one, deployed individually, or all packages can be deployed with one command.

Force.com Migration Tool - build.properties

```
# build.properties
#

#git repository & branch used for this effort
sf.repository = <insert your git repository URI here>
sf.branch = <Insert the specific branch name here>

# Specify the login credentials for the desired Salesforce organization
dev.username = <Insert your Salesforce username here>
dev.password = <Insert your Salesforce password here>
#dev.sessionId = <Insert your Salesforce session id here.  Use this or username/password above.  Cannot use both>
dev.serverurl = https://login.salesforce.com

sit.username = <Insert your Salesforce username here>
sit.password = <Insert your Salesforce password here>
#sit.sessionId = <Insert your Salesforce session id here.  Use this or username/password above.  Cannot use both>
sit.location = <set directory>
sit.serverurl = https://test.salesforce.com

isit.username = <Insert your Salesforce username here>
isit.password = <Insert your Salesforce password here>
#isit.sessionId = <Insert your Salesforce session id here.  Use this or username/password above.  Cannot use both>
isit.location = <set directory>
isit.serverurl = https://test.salesforce.com

uat.username = <Insert your Salesforce username here>
uat.password = <Insert your Salesforce password here>
#uat.sessionId = <Insert your Salesforce session id here.  Use this or username/password above.  Cannot use both>
uat.location = <set directory>
uat.serverurl = https://test.salesforce.com

staging.username = <Insert your Salesforce username here>
staging.password = <Insert your Salesforce password here>
#staging.sessionId = <Insert your Salesforce session id here.  Use this or username/password above.  Cannot use both>
staging.location = <set directory>
staging.serverurl = https://test.salesforce.com

prod.username = <Insert your Salesforce username here>
prod.password = <Insert your Salesforce password here>
#prod.sessionId = <Insert your Salesforce session id here.  Use this or username/password above.  Cannot use both>
prod.location = <set directory>
prod.serverurl = https://login.salesforce.com


#sf.pkgName = <Insert comma separated package names to be retrieved>
#sf.zipFile = <Insert path of the zipfile to be retrieved>
#sf.metadataType = <Insert metadata type name for which listMetadata or bulkRetrieve operations are to be performed>

# Use 'https://login.salesforce.com' for production or developer edition (the default if not specified).
# Use 'https://test.salesforce.com for sandbox.
sf.serverurl = https://login.salesforce.com

sf.maxPoll = 20
# If your network requires an HTTP proxy, see http://ant.apache.org/manual/proxy.html for configuration.
#
```

Force.com Migration Tool - build.xml

```
<project name="VA usage of Salesforce Ant tasks" default="test" basedir="." xmlns:sf="antlib:com.salesforce">

    <property file="./build.properties"/>
    <property environment="env"/>

    <!-- Setting default value for username, password and session id properties to empty string
         so unset values are treated as empty. Without this, ant expressions such as ${sf.username}
         will be treated literally.
    -->

    <!-- dev instance variables -->
    <condition property="dev.username" value=""> <not> <isset property="dev.username"/> </not> </condition>
    <condition property="dev.password" value=""> <not> <isset property="dev.password"/> </not> </condition>
    <condition property="dev.sessionId" value=""> <not> <isset property="dev.sessionId"/> </not> </condition>

    <!-- sit instance variables -->
    <condition property="sit.username" value=""> <not> <isset property="sit.username"/> </not> </condition>
    <condition property="sit.password" value=""> <not> <isset property="sit.password"/> </not> </condition>
    <condition property="sit.sessionId" value=""> <not> <isset property="sit.sessionId"/> </not> </condition>

    <!-- isit instance variables -->
    <condition property="isit.username" value=""> <not> <isset property="isit.username"/> </not> </condition>
    <condition property="isit.password" value=""> <not> <isset property="isit.password"/> </not> </condition>
    <condition property="isit.sessionId" value=""> <not> <isset property="isit.sessionId"/> </not> </condition>

    <!-- uat instance variables -->
    <condition property="uat.username" value=""> <not> <isset property="uat.username"/> </not> </condition>
    <condition property="uat.password" value=""> <not> <isset property="uat.password"/> </not> </condition>
    <condition property="uat.sessionId" value=""> <not> <isset property="uat.sessionId"/> </not> </condition>

    <!-- staging instance variables -->
    <condition property="staging.username" value=""> <not> <isset property="staging.username"/> </not> </condition>
    <condition property="staging.password" value=""> <not> <isset property="staging.password"/> </not> </condition>
    <condition property="staging.sessionId" value=""> <not> <isset property="staging.sessionId"/> </not> </condition>

    <!-- prod instance variables -->
    <condition property="prod.username" value=""> <not> <isset property="prod.username"/> </not> </condition>
    <condition property="prod.password" value=""> <not> <isset property="prod.password"/> </not> </condition>
    <condition property="prod.sessionId" value=""> <not> <isset property="prod.sessionId"/> </not> </condition>

    <taskdef resource="com/salesforce/antlib.xml" uri="antlib:com.salesforce">
        <classpath>
            <pathelement location="../salesforce_ant_41.0/ant-salesforce.jar" />
        </classpath>
    </taskdef>

    <macrodef name = "git">
        <attribute name = "command" />
        <attribute name="options" default="" />
        <attribute name="dir" default="" />
        <attribute name="failerror" default="false" />
        <element name = "args" optional = "true" />
        <sequential>
            <echo message="git dir @{dir}" />
            <echo message="git @{command}" />
            <exec executable="git" dir="@{dir}" failonerror="@{failerror}">
                <arg line="@{command} @{options}" />
                <args />
```

```
    <!-- dev Git and Migration tool targets -->
    <target name="dev-git-clone" unless="file.found" description="Create directory structure, clone the remote repository and the pull that latest source">
        <echo message="Cloning the repository ${sf.repository} to ${dev.location}" />
        <delete dir="${dev.location}"/>
        <git command="clone">
            <args>
                <arg value="${sf.repository}" />
                <arg value="${dev.location}" />
            </args>
        </git>
        <git command="status" />
    </target>

    <target name="dev-git-pull" unless="file.found" description="Update the current local repository from the remote repository">
        <echo message="Pulling updates to the repository ${sf.repository} to ${dev.location}" />
        <git command="pull">
            <args>
                <arg value="${sf.repository}" />
                <arg value="${dev.location}" />
            </args>
        </git>
        <git command="status" />
    </target>

    <target name="dev-extract" description="Extract all configuration from the org based on package.xml to local repository">
      <!-- mkdir dir="retrieveUnpackaged"/ -->
      <!-- Retrieve the contents into another directory -->
      <echo message="Extracting all unpackaged from ${dev.serverurl} to ${dev.location}" />
      <sf:retrieve username="${dev.username}" password="${dev.password}" serverurl="${dev.serverurl}" maxPoll="${sf.maxPoll}" retrieveTarget="retrieveUnpackaged" unpackaged="unpackaged/package.xml"/>
    </target>

    <target name="dev-push" description="Push changes in the local repository to the remote repository">
        <input message="Commit message" addproperty="commit-message" />
        <echo message="Commiting all changes with message ${commit-message}" />
        <git command="add">
            <args>
                <arg value="." />
            </args>
        </git>
        <git command="commit">
            <args>
                <arg value="-am ${commit-message}" />
            </args>
        </git>
        <git command="push" />
    </target>
```

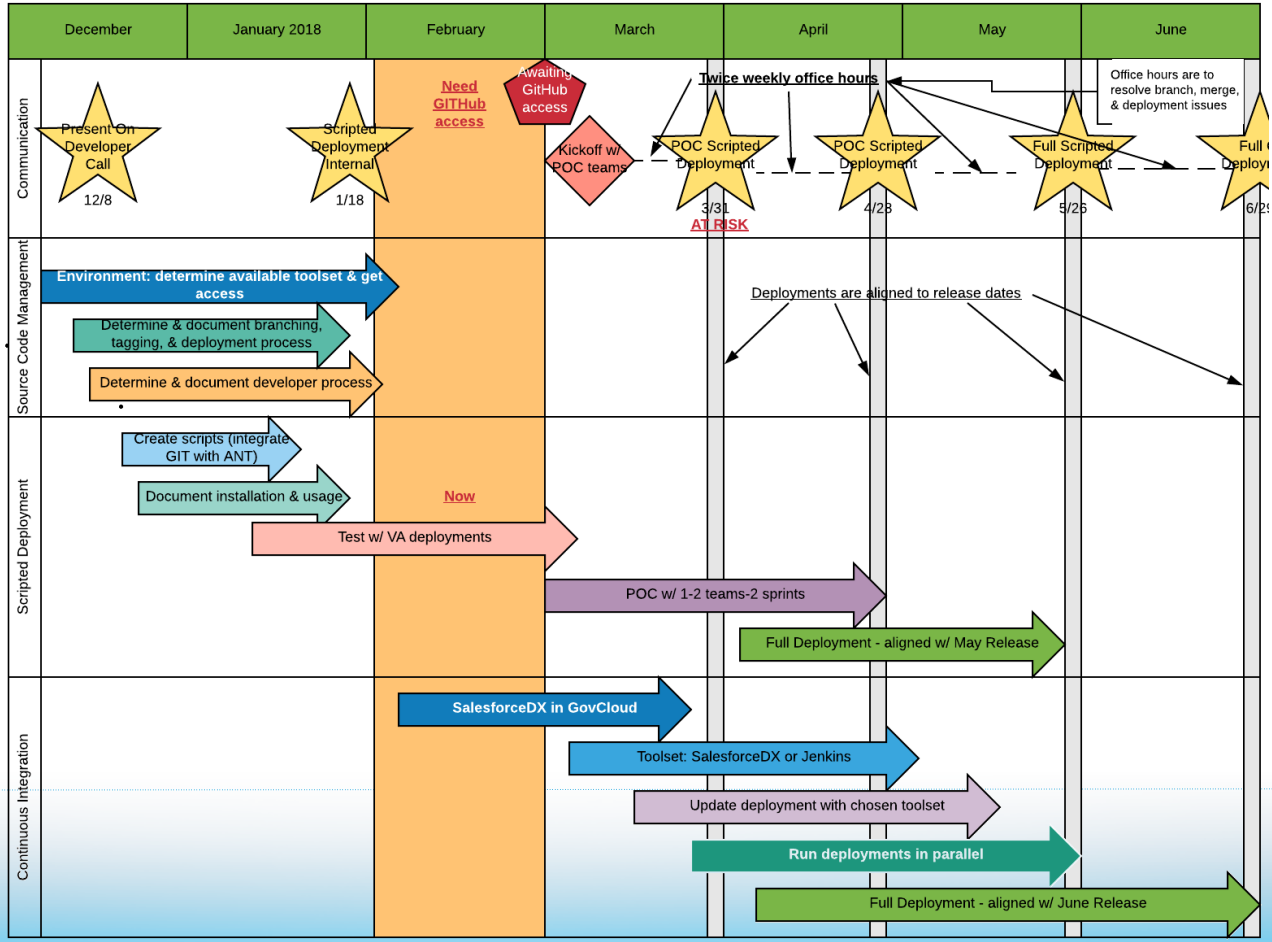
Force.com Migration Tool - package.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Package xmlns="http://soap.sforce.com/2006/04/metadata">
    <types>
        <members>*</members>
        <name>ApexClass</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexComponent</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexPage</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexTestSuite</name>
    </types>
    <types>
        <members>*</members>
        <name>ApexTrigger</name>
    </types>
    <types>
        <members>*</members>
        <name>AuraDefinitionBundle</name>
    </types>
    <types>
        <members>*</members>
        <name>StaticResource</name>
    </types>
    <version>38.0</version>
</Package>
```

Detailed instruction regarding the contents of the package.xml file can be found at:
https://developer.salesforce.com/docs/atlas.en-us.daas.meta/daas/daas_package.htm

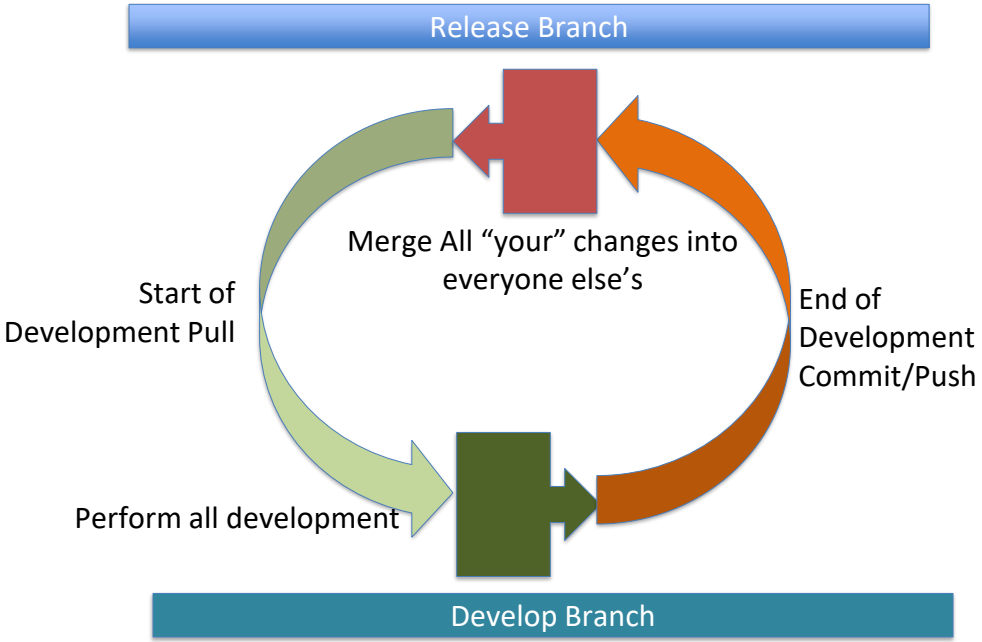
Full list of all the available metadata types can be found at:
https://developer.salesforce.com/docs/atlas.en-us.210.0.api_meta.meta/api_meta/meta_types_list.htm

Deployment Timeline

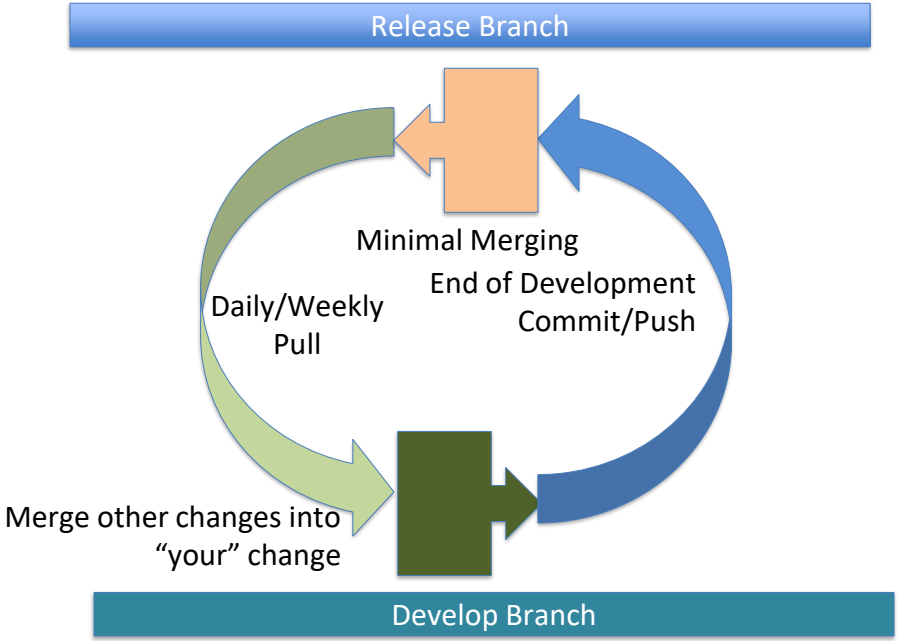


Usual Pull/Push/Merge Process

Might appear the most logical



Best Pull/Push/Merge Process



Developer Training

Developer Training

Provider	Product	Category	Location/URL
Salesforce	Trailhead	Git and GitHub.com Basics	https://trailhead.salesforce.com/modules/git-and-git-hub-basics/units/learn-why-version-control-is-important-for-team-based-development (6 modules)
GitHub.com	On-demand Training	GitHub's On Demand Training	https://services.github.com/on-demand/
GitHub.com	On-demand Training	GitHub's Recommended Learning Path	https://services.github.com/on-demand/resources/learning-path/
Git	Web training	Getting Started - Git Basics	https://git-scm.com/book/en/v2/Getting-Started-Git-Basics
RogerDudler	Web training	git - the simple guide	http://rogerdudler.github.io/git-guide/

Next Steps

Next Steps - Phase I

- Administration
  - **Waiting for GitHub.com account from VA**
  - Complete testing on all targets using recent deployment requests - create/update targets as necessary
  - Setup GitHub.com account and access for all
  - Create GitHub.com teams
  - Create approvers
  - Import production SF repository to GitHub.com
  - Create release branch
  - Associate teams with release branch

- Developers/all—phase I
  - Create GitHub.com account
  - Complete specified training
  - Install developer work station
  - Install eGit

- Developers/all—phase II **after you receive your GitHub.com invitation to the VA repository
  - Create Eclipse Project from GIT
  - Switch to branch for development
  - Associate Sandbox to IDE

- Developers - start developing!
  - Make changes
  - Making Updates to repository
  - Commit and Push to Server – Branch Update Only
  - Sync to Main Branch **only when development cycle is complete**

- Team Lead / Project Lead
  - Approve a Pull Request

thank you