Automation Recipes

NOTE - out of date, to be updated

Commands are a bit simpler now with the jenkins changes (see https://jenkins.modelmetricssoftware.com/view/AntSFDC_Demo/job/AntSFDC_Dev_QA_Auto/ for an example)

Setup Git Submodule

To get started, include the 'sfdcantproject' to a git submodule.

A git module is a reference to another project and allows for easy upgrading as needed.

```
Please note, that if someone doesn't have access to the submodule/project, the
folder will appear empty and may provide an error message. Giving them access
will clear the message and ensure that they can proceed as expected.
```

To add a submodule to a git project, navigate to the base directory and run the following shellscript.

```
git submodule add git@tig2.modelmetricssoftware.com:proth/sfdcantproject.git
sfdcantproject
git submodule init
git submodule update

# schedule the submodule for the next commit
git add sfdcantproject
# commit
git commit sfdcantproject
# push the change out so we can use it in jenkins
git push
```

This will have created a new folder/module of your project called 'sfdcantproject'.

The rest of this document assumes the following structure, but can of course be configured as needed.

```
<base project>
*   /sfdcantproject
*   /src
        /objects
        /labels
        /pages
        /...
*   /resources
        /<staticresource_zip expanded folder>
        /<staticresource_zip expanded folder>
        /<staticresource_zip expanded folder>
        /<staticresource expanded file>
        /<staticresource expanded file>
        /...
    /documentation
    /mockups
    /...
```

For more information about this structure, please see the `Readme` document

Initialization

Once the submodule is included, we can then setup the project (and credentials) for it to run.

This can then be done either by executing the following shellscript, or creating a new Jenkins ShellScript build step with the following.

This will need to be run first before any of the following Automation Scripts can be run

```
cd sfdcantproject

srcPath=[ ex: ../src ]
resourcesPath=[ ex: ../resources ]
retrieveEnvironment=[ ex: dev ]
retrieveUser=[ your sfdc username ]
retrievePass=[ your sfdc password with token: ex yourpasswordYOURTOKEN ]
retrieveHost=
deployEnvironment=
deployUser=
deployPass=
deployHost=[ login.salesforce OR test.salesforce ]
isTestDeploy=false
commitFrom=[ EX: 40518c88e60f2af2076c096477b1445124dff7b4 ]

##  #   #   #   #   #   #   #   #
## The following is the actual code and does not need to be changed.
## (It references the variables set above)
##  #   #   #   #   #   #   #   #

forceEnvironment=$retrieveEnvironment
ant -Drun=setup -DsetupContinueA= -DsetupOption=1 -DlocalSrcPath=$srcPath -
DlocalResourcePath=$resourcesPath -DsetupCredentials=n
ant -Dignore=credentials setupEnvironment -Denvironment.in=$deployEnvironment -
Dretrieve.username.in=$deployUser -Dretrieve.password.in=$deployPass -
Dretrieve.host=$deployHost
ant -Dignore=credentials setupRetrieveCredentials -
Dretrieve.environment.in=$retrieveEnvironment -
Dretrieve.username.in=$retrieveUser -Dretrieve.password.in=$retrievePass -
Dretrieve.host=$retrieveHost
ant testDeploy -DisTestDeployInput=$isTestDeploy
ant lastAutoDeploy -DlatestCommit=$commitFrom
#test
ant prop
```

Again, once this has been run, the script is no longer needed (but should be kept for records) So the script (or build step) can be immediately removed / replaced with the following Automation Scripts.

The following is an Extended version

The following is a longer form (with comments).

```
##  #   #   #   #   #   #   #   #
## Fill in the following information
##  #   #   #   #   #   #   #   #

cd sfdcantproject

#location of src path (absolute or relative) - must be named 'src'
#(If using the recommendation of sfdcantproject as a submodule, use '../src')
srcPath=[ ex: ../src ]

#location of where you want extracted resources
#(If using the recommendation of sfdcantproject as a submodule, use
'../resources')
resourcesPath=[ ex: ../resources ]

#name of the environment to use
#(This is a human readable name - without spaces - of where you are retriving
from)
#(and can also be set with Jenkins build parameters if setup correctly)
retrieveEnvironment=[ ex: dev ]

retrieveUser=[ your sfdc username ]
retrievePass=[ your sfdc password with token: ex yourpasswordYOURTOKEN ]

# either login.salesforce or test.salesforce
#(note, host must currently be be test.salesforce or login.salesforce)
#(docs will need to be updated to describe custom URLs/myDomains)
retrieveHost=

## Deploy credentials are set in the same manner

deployEnvironment=
deployUser=
deployPass=
deployHost=[ login.salesforce OR test.salesforce ]

#whether the deploys should be tested (true) or actually deploy (false)
isTestDeploy=false

#SHA of commit to start autodeploying from (non inclusive)
commitFrom=[ EX: 40518c88e60f2af2076c096477b1445124dff7b4 ]




##  #   #   #   #   #   #   #   #
```

```
## The following is the actual code and does not need to be changed.
## (It references the variables set above)
##  #   #   #   #   #   #   #   #

forceEnvironment=$retrieveEnvironment


#(Please note, the following ant arguments are on different lines for
legibility)
#(followed by a line ant understands, but is less legible)

#run setup
#ant
#   -Drun=setup
#   -DsetupContinueA=
#   -DsetupOption=1
#   -DlocalSrcPath=$srcPath
#   -DlocalResourcePath=$resourcesPath
#   -DsetupCredentials=n

#- line ant understands
ant -Drun=setup -DsetupContinueA= -DsetupOption=1 -DlocalSrcPath=$srcPath -
DlocalResourcePath=$resourcesPath -DsetupCredentials=n

#specify the environment currently on, or the deployment environment
#ant
#   -Dignore=credentials setupEnvironment
#   -Denvironment.in=$deployEnvironment
#   -Dretrieve.username.in=$deployUser
#   -Dretrieve.password.in=$deployPass
#   -Dretrieve.host=$deployHost

#- line ant understands
ant -Dignore=credentials setupEnvironment -Denvironment.in=$deployEnvironment -
Dretrieve.username.in=$deployUser -Dretrieve.password.in=$deployPass -
Dretrieve.host=$deployHost

#if the refresh environment is different
#ant
#   -Dignore=credentials setupRetrieveCredentials
#   -Dretrieve.environment.in=$retrieveEnvironment
#   -Dretrieve.username.in=$retrieveUser
#   -Dretrieve.password.in=$retrievePass
#   -Dretrieve.host=$retrieveHost

#- line ant understands
ant -Dignore=credentials setupRetrieveCredentials -
```

```
Dretrieve.environment.in=$retrieveEnvironment -
Dretrieve.username.in=$retrieveUser -Dretrieve.password.in=$retrievePass -
Dretrieve.host=$retrieveHost

#whether to test autodeploys or not
ant testDeploy -DisTestDeployInput=$isTestDeploy

#SHA of commit to start autodeploying from (non-inclusive)
ant lastAutoDeploy -DlatestCommit=$commitFrom

#once all setup is done
ant prop
```

Once this has been run, the script is no longer needed (but should be kept for records) So the script (or build step) can be immediately removed / replaced with the following Automation Scripts.

Automation Scripts

Once the project has been setup as a git module and initialized, any of the following can be run.

Run Automated Documentation

Automated Documentation can either run to to show all information that is currently in VersionControl or is currently in SalesForce (and not necessarily in Version Control)

Automated Documentation from Version Control

```
cd sfdcantproject

ant docAllAuto
#note - depending on your environment, you may need to run the following instead
#to ensure the correct library is run
#ant docAllAuto -lib antlib
```

Automated Documentation from SalesForce

Again, as this is not what is in version control, then this means that all information will need to be re-retrieved each time this is run.

To accomplish this, we temporarily remove all items from version control, and re-retrieve everything based on the 'package.xml' file from the src/metadata directory.

[[TODO: paul to add an ant task to reset the src directory]]

```
#!/bin/bash
#name of the environment to use
#ex: forceEnvironment=swsdev
forceEnvironment=prod

# remove the existing src directory
cd sfdcantproject
ant resetRefresh -DoverrideContinue=true

ant docAllAuto -lib antlib
```

Change your credentials

You will need to change your credentials occasionally.

Simply set the targetEnvironment to the environment to work with and fill in the rest.
(Regardless of whether it is used for retreival or not)

```
cd sfdcantproject

targetEnvironment=
targetUser=
targetPass=
#either test.salesforce or login.salesforce
targetHost=
ant resetCredentials -Dtarget.environment=$targetEnvironment \
    -Dretrieve.username.in=$targetUser \
    -Dretrieve.password.in=$targetPass \
    -Dretrieve.host=$targetHost
```

Update TestDeploy or Actual Deploy

Occassionally, you need to test the credentials (especially when those credentials change)

```
cd sfdcantproject

#whether the deploys should be tested (true) or actually deploy (false)
isTestDeploy=false

#whether to test autodeploys or not
ant testDeploy -DisTestDeployInput=$isTestDeploy
```

Automated Deployment Script

This script performs autoDeployment from one org (retrieval org) to the next (deployment org).

```
cd sfdcantproject

ant autoDeploy
#note - depending on your environment, you may need to run the following instead
#to ensure the correct library is run
#ant autoDeploy -lib antlib
```

Considerations

- When doing autodeployment, only items that are in VersionControl are deployed. If things are not in version control, they will not be deployed.
- To capture metadata changes (and other changes not in version control), it is recommended that a regular cadence is setup to perform the command `ant refresh` (possibly by a TA / SR Dev) to review changes that were made in SalesForce and are not in VersionControl. This should include changes to profiles/page layouts/objects/etc.
- While it is very easy to setup automated commits of config related areas (such as objects/layouts/profiles/etc.) - this is strongly recommended against. There are often changes that are made in development environments that are not necessarily intended for production use and should be discussed rather than included whole-cloth.
- **An early decision was that the Automated Deployment would not automatically destroy metadata.**

  This proved too dangerous (especially considering certain open considerations from SalesForce - such as profiles only reflect items included, but certain aspects - like page layout assignments - are considered deleted if not included)
- One of the greater benefits of using GIT is that individual ranges within a file (not

necessarily the whole file) can be committed to version control. If there are specific changes, such as certain fields, profile changes, etc that are desired.

While they can be individually committed using git http://stackoverflow.com/questions/1085162/how-can-i-commit-only-part-of-a-file-in-git.

It is much easier to do this with an application (such as Tower - http://www.git-tower.com/ or SourceTree - http://www.sourcetreeapp.com/)

- **Prior to doing merges from one Branch to Another (such as from Dev to QA), it is recommended that you do a quick refresh of the target org before the migration.** This ensures that differences in the target org cannot be overwritten/lost.

  If the refresh is done after the migration, there is a risk of changes to be deployed getting lost as that refresh would be considered a removal/rollback of those changes.

Update SHA of Base Commit

Currently, the autoDeploy traditionally deploys from the commitFrom / build.autodeploy SHA to HEAD. (Again this is configurable)

Considering how most projects have used the commit (especially since metadata changes are not often committed on a timely basis), the build.autodeploy (starting point) does not automatically move for each deploy.

Instead, this has traditionally been a starting point (such as at the beginning of a sprint or when synchronizing to SalesForce).

```
cd sfdcantproject

#SHA of commit to start autodeploying from (non inclusive)
commitFrom=

#SHA of commit to start autodeploying from (non-inclusive)
ant lastAutoDeploy -DlatestCommit=$commitFrom
```

It is recommended that you explicitly set the end commit, but again, you can always get the head commit SHA through the following:

```
commitFrom=`git log --pretty=format:'%H' -n 1`
```