DX PROJECT SHELL SCRIPTS

Shells scripts to assist the DX process

Abstract

There is a lot of information contained (and tangled) in Salesforce Orgs. In understanding those dependencies and relationships one can better slice components into manageable chunks to maintain and service.

bill anderson

Bill.anderson@salesforce.com

Table of Contents

DX Project Shell Scripts

This document outlines the shells scripts defined to support the DX Project initiative. These scripts (bash) provide a means to accelerate and encapsulate common behavior. It should be noted, these scripts are very much dependent upon SFDX command line (CLI) behavior and will likely change over the evolution of SFDX CLI.

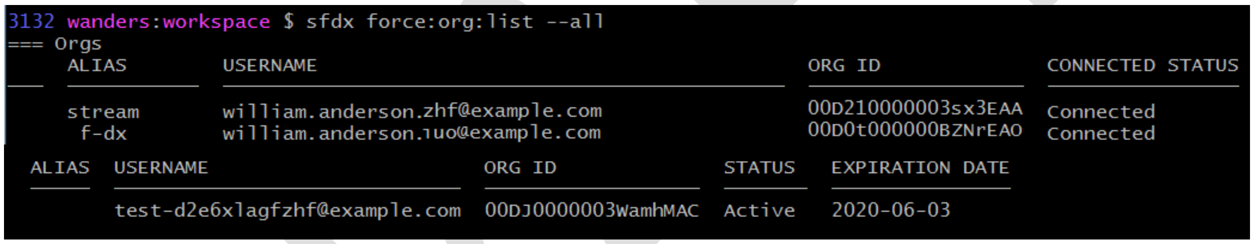
Assumptions

Shell Commands

It is assumed the users whom maintain these scripts have a general understanding of shell scripts and shell tools (*awk*, *grep*,*sed*, *cut*, etc.). All main scripts have the ability to turn on debug mode (via the command line parameter ***-d***).

SFDX CLI

Understanding general functionality of SFDX CLI is needed. This comes into play when authorizing access to a non-ScratchOrg or ScratchOrg. Many of the commands expect a username/target-name (***-u***) to reference the Org. For example, a common command to see what Orgs (and Scratch Orgs) one has available, ***sfdx force:org:list –all***:



```
3132 wanders:workspace $ sfdx force:org:list --all
=== Orgs
     ALIAS        USERNAME                                        ORG ID              CONNECTED STATUS
───  ───────────  ──────────────────────────────────────────────  ──────────────────  ────────────────
     stream       william.anderson.zhf@example.com                00D210000003sx3EAA  Connected
      f-dx        william.anderson.ıuo@example.com                00D0t000000BZNrEAO  Connected
  ALIAS  USERNAME                         ORG ID              STATUS  EXPIRATION DATE
  ─────  ───────────────────────────────  ──────────────────  ──────  ───────────────
         test-d2e6xlagfzhf@example.com    00DJ0000003WamhMAC  Active  2020-06-03
```

Manifest

Below are a list of the shell scripts defined for the DX Project.

Script name	Description
analyzeOrg.sh	Utility analyzes a Salesforce Org
buildScratchOrg.sh	Utility builds a Scratch Org with needed packages (i.e. FSC)
createProject.sh	Utility creates a VS Code Project; seeded with an unmanaged Package from Sandbox
dotGen.sh	Utility to pull metadata dependencies and create a visualization (Graphviz)
md_used.sh	Utility to pull down the metadata used in an Org
sfdxUrlEncrpt.sh	Utility to encrypt a SFDX URL stored token
packConfig.sh	Utility to set environment variables
setScratchOrgJSON.sh	Utility to create a project definition file for a scratch Org
csvSheets.sh	Utility to create CSV files [**awk**]
filterPackages.sh	Utility to get installed package names and corresponding package Ids [**awk**]

Note, the scripts marked in **red**, indicate a stand-alone script. Scripts marked in **black**, are utility scripts.

Tools Used

In order to limit the dependencies of these shell scripts, all shell scripts (bash) utilize the following (tools natively found in Unix environments) aspects:

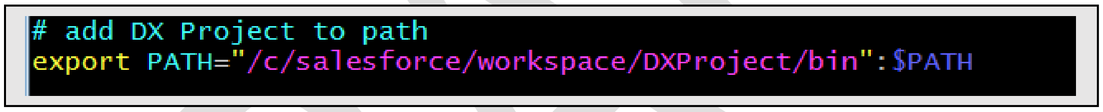
- awk,
- grep
- sed,
- cut
- bash (shell script)

The two non-standard tools utilized are :

- SFDX CLI
- Graphviz (dot)

Configuration

Each user is required to add the DX *bin* path to their PATH environment variable. At the time of this writing it is not certain where these scripts will reside. However, for example, if the DX project resides in the following directory (and assuming a git-bash shell installation on Windows), */c/salesforce/workspaceDXProject*. One would add the following to their resource file (either *~/.bashrc* or *~/.bash_profile*):

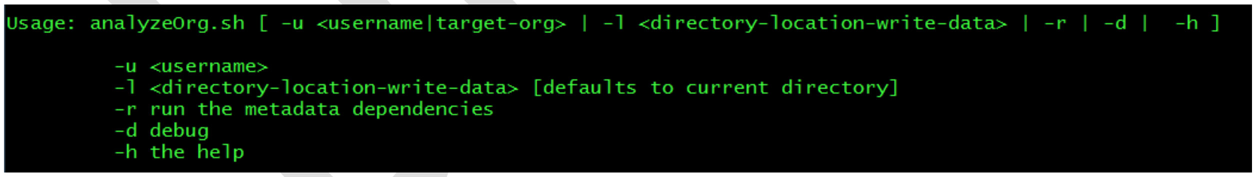


```
# add DX Project to path
export PATH="/c/salesforce/workspace/DXProject/bin":$PATH
```

Standalone Shell Scripts

Below are the list of standalone scripts and their respective functionality. All standalone scripts provide general help (**-h**) and debug (**-d**).

analyzeOrg.sh



```
Usage: analyzeOrg.sh [ -u <username|target-org> | -l <directory-location-write-data> | -r | -d |  -h ]

         -u <username>
         -l <directory-location-write-data> [defaults to current directory]
         -r run the metadata dependencies
         -d debug
         -h the help
```

This script analyzes a **non ScratchOrg** placing the results into either the current working directory or user specified directory (**-l**) into a directory named ***orgAnalysis***. The script performs the following tasks:

- Counts the Standard and Custom Objects (placing results in user specified directory)
- With the **-r** option, [**md_used.sh**] pulls metadata dependency information and [**dotGen.sh**] creates Graphviz (dot) files and PNG file (for visualized dependencies) .

Command-line Arguments

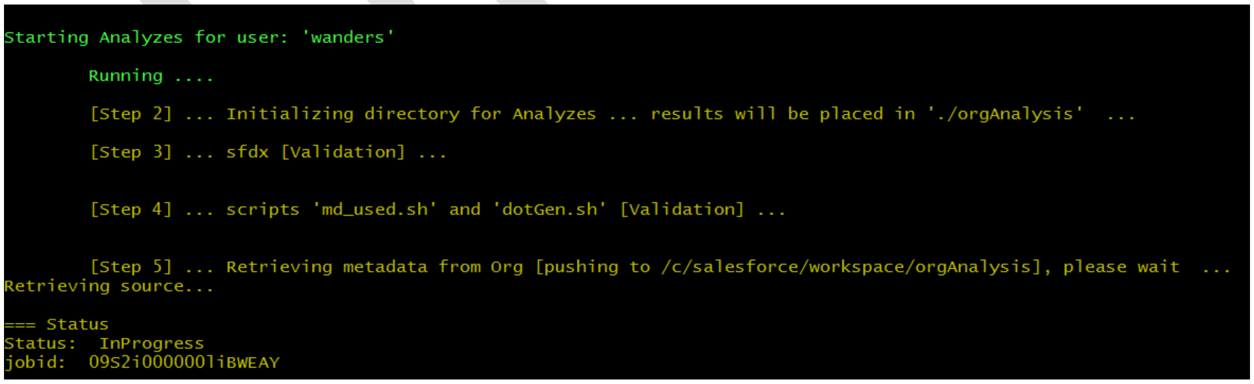
Name	Comments
-u <target-org>	The target-org name (or alias). This is the Org which you are analyzing. You can get a target-org (or alias) by using the **sfdx force:auth** command (***sfdx force:auth*** –help for more information)
-l <directory>	The option specifies where to create the directory ***orgAnalysis***. By default, it creates the directory (***orgAnalysis)*** in your current directory. For example, if you specify, **-l myCRV,** it will create **myCRV** in your current directory, and then create ***orgAnalysis,*** underneath (**./myCRV/orgAnalysis**)
-r	Run the metadata dependency to gather metrics. This invokes the md_used.sh and dotGen.sh scripts placing metadata dependency information in **orgAnalysis/data**, and Graphviz files in ***orgAnalysis/png*** and ***orgAnalysis/parsed.*** The png directory creates **SVG** files open in your browser (especially, Chrome browsers).
-d	Runs the script using **set -xv**
-h	Help

General Comments

- The script uses a large package.xml to pull most metadata from the target Org.
- The script **does not create a project** (but has to include *sfdx-project.json* in order to run sfdx cli).
- Uses the converted source in order to gather statistics and create CSV file(s).
- Because there is a large amount of metadata to pull from the target Org, this script will take 10-30 minutes to run (though, the times will vary based on network, system, etc.)
- When the script runs, it will create a directory, **orgAnalysis**, in the user defined location. Results from the analysis are placed there.

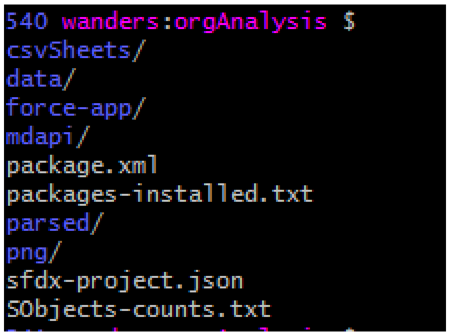
Output

The output of this script are as follows (assuming **-l** *orgAnalysis* as the current directory) [assuming **-r** option as well] (***analyzeOrg.sh -u wf-dx -l . -r***):



```
Starting Analyzes for user: 'wanders'

        Running ....

        [Step 2] ... Initializing directory for Analyzes ... results will be placed in './orgAnalysis'  ...

        [Step 3] ... sfdx [Validation] ...


        [Step 4] ... scripts 'md_used.sh' and 'dotGen.sh' [Validation] ...


        [Step 5] ... Retrieving metadata from Org [pushing to /c/salesforce/workspace/orgAnalysis], please wait  ...
Retrieving source...

=== Status
Status:  InProgress
jobid:  09S2i000000liBWEAY
```

After completion, the above files and directories are created


```
540 wanders:orgAnalysis $
csvSheets/
data/
force-app/
mdapi/
package.xml
packages-installed.txt
parsed/
png/
sfdx-project.json
SObjects-counts.txt
```

Created files and directories

Name	Type	Comments
data/	D	Metadata Dependencies files are placed here
force-app/	D	Common force-app directory to hold converted metadata
csvSheets/	D	All created CSV files
parsed/	D	Data parsed from the metadata (used to assist in debug, if needed)
png/	D	SVG or PNG files and DOT (Graphviz) files
mdapi/	D	Metadata source from Source Org
package.xml	F	Metadata resources to pull from Source Org
sfdx-project.json	F	Used to allow SFDX CLI (if ***sfdx*** sees this file it assumes project)
SObjects-counts.txt	F	Custom and Standard Object Count
packages-installed.txt	F	List of packages installed in Source Org
csvSheets/MetadataCounts.txt	F	Holds counts per metadata

Example File Output for SObjects-counts.txt

```
Number of Standard SObjects: 878
Number of Custom SObjects  : 164
Number of ALL SObjects     : 1042
```

Example File Output for packages-installed.txt

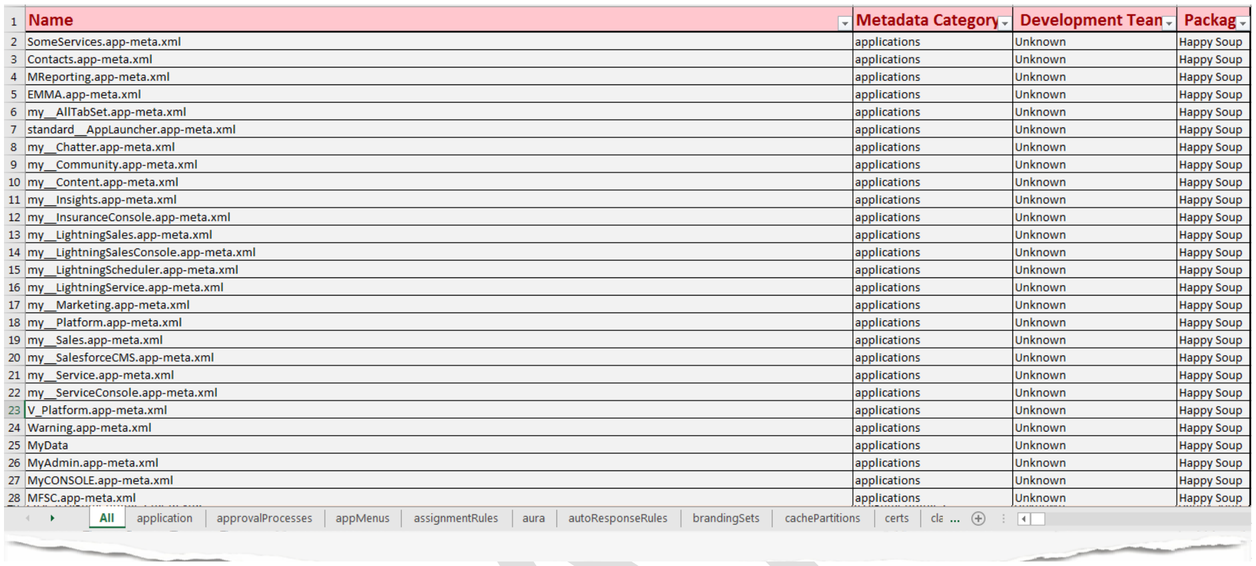
```
=== Installed Package Versions [8]
ID                  Package ID           Package Name                         Namespace    Package Version ID  Version Name      Version
__________________  ___________________  ___________________________________  ___________  __________________  ________________
__________

0A36A000100HgA8SAK  03330003210O0ndAAC  Knowledge Base Dashboards & Reports  SK          04t30000670bqOuAAI  Summer'10       1.23.0.3
0A36A000400PQlzSAG  03330000980wDAbAAM  Salesforce Connected Apps           sf_com_apps  04t300GH001DUvrAAG  Winter '16      1.7.0.1
0A36A000800H5J3SAK  0334100060000Q2XAAU  Salesforce Adoption Dashboards                 04t41000AZ09jsfAAA  1.2            1.2.0.1
0A36A000900HbEUSA0  0334B0000400DZ1UQAW  ACT                                            04t4B000MISDxh1QAC  ACT Integration   1.1.0.1
0A36A000500Hg9ySAC  03380005000AqsRAAS  Financial Services Referral Ext                 04t8123MNB01AWrfAAG  FSC Referral Ext  216.0.0.1
0A36A000300Hg9tSAC  03380001000FSQMAA4  Financial Services Ext                          04t88760Q01AWVhAAO  FSC ext          216.0.0.1
0A36A000900Hg9oSAC  03380032000U1CPAA0  Financial Services Cloud            FinServ     04t1KAJX001AXfpQAG  r226.2.0         226.2.0.1
```

Force-app and mdapi Directory

These directories are created to pull statistics and cull information for analysis. These directories can be deleted if necessary.

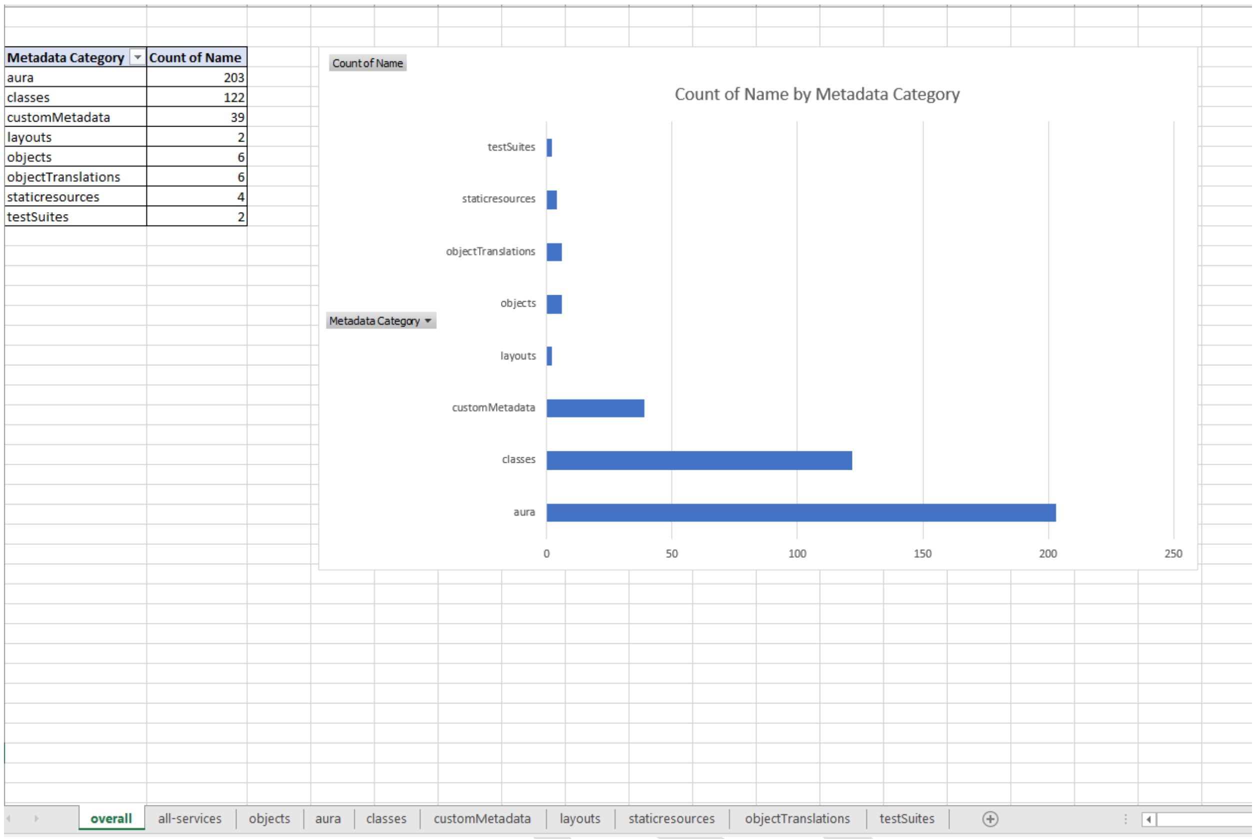
CsvSheets Directory

This directory holds the CSV files created from the Force-app directory. There is a main CSV file, ***allCRVmetadata.csv***, which holds all the metadata. Each Metadata Category, i.e. *applications*, *appMenus*, etc. is created in a separate CSV file. You can bring these separate CVS files into the ***allCRVmetadata.csv*** individually or via a script.



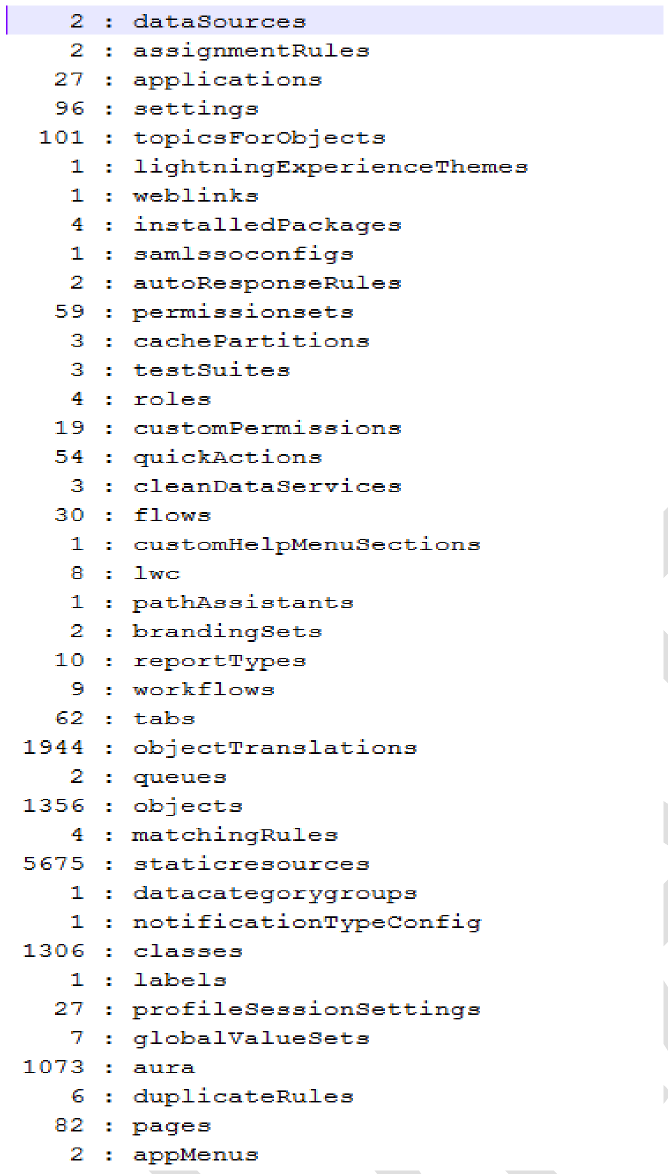
1	Name	Metadata Category	Development Tean	Packag
2	SomeServices.app-meta.xml	applications	Unknown	Happy Soup
3	Contacts.app-meta.xml	applications	Unknown	Happy Soup
4	MReporting.app-meta.xml	applications	Unknown	Happy Soup
5	EMMA.app-meta.xml	applications	Unknown	Happy Soup
6	my__AllTabSet.app-meta.xml	applications	Unknown	Happy Soup
7	standard__AppLauncher.app-meta.xml	applications	Unknown	Happy Soup
8	my__Chatter.app-meta.xml	applications	Unknown	Happy Soup
9	my__Community.app-meta.xml	applications	Unknown	Happy Soup
10	my__Content.app-meta.xml	applications	Unknown	Happy Soup
11	my__Insights.app-meta.xml	applications	Unknown	Happy Soup
12	my__InsuranceConsole.app-meta.xml	applications	Unknown	Happy Soup
13	my__LightningSales.app-meta.xml	applications	Unknown	Happy Soup
14	my__LightningSalesConsole.app-meta.xml	applications	Unknown	Happy Soup
15	my__LightningScheduler.app-meta.xml	applications	Unknown	Happy Soup
16	my__LightningService.app-meta.xml	applications	Unknown	Happy Soup
17	my__Marketing.app-meta.xml	applications	Unknown	Happy Soup
18	my__Platform.app-meta.xml	applications	Unknown	Happy Soup
19	my__Sales.app-meta.xml	applications	Unknown	Happy Soup
20	my__SalesforceCMS.app-meta.xml	applications	Unknown	Happy Soup
21	my__Service.app-meta.xml	applications	Unknown	Happy Soup
22	my__ServiceConsole.app-meta.xml	applications	Unknown	Happy Soup
23	V_Platform.app-meta.xml	applications	Unknown	Happy Soup
24	Warning.app-meta.xml	applications	Unknown	Happy Soup
25	MyData	applications	Unknown	Happy Soup
26	MyAdmin.app-meta.xml	applications	Unknown	Happy Soup
27	MyCONSOLE.app-meta.xml	applications	Unknown	Happy Soup
28	MFSC.app-meta.xml	applications	Unknown	Happy Soup

As another example, looking at services group, it was possible to bring in the related CSV files and gain further analysis.



Metadata Category	Count of Name
aura	203
classes	122
customMetadata	39
layouts	2
objects	6
objectTranslations	6
staticresources	4
testSuites	2

Finally, this directory contains a file, ***MetadataCounts.txt***, which holds the number of metadata components.



```
   2 : dataSources
   2 : assignmentRules
  27 : applications
  96 : settings
 101 : topicsForObjects
   1 : lightningExperienceThemes
   1 : weblinks
   4 : installedPackages
   1 : samlssoconfigs
   2 : autoResponseRules
  59 : permissionsets
   3 : cachePartitions
   3 : testSuites
   4 : roles
  19 : customPermissions
  54 : quickActions
   3 : cleanDataServices
  30 : flows
   1 : customHelpMenuSections
   8 : lwc
   1 : pathAssistants
   2 : brandingSets
  10 : reportTypes
   9 : workflows
  62 : tabs
1944 : objectTranslations
   2 : queues
1356 : objects
   4 : matchingRules
5675 : staticresources
   1 : datacategorygroups
   1 : notificationTypeConfig
1306 : classes
   1 : labels
  27 : profileSessionSettings
   7 : globalValueSets
1073 : aura
   6 : duplicateRules
  82 : pages
   2 : appMenus
```

Figure 1MetadataCounts.txt

Data Directory

This directory holds metadata dependencies. These files are used to map the dependencies of related components and used to provide a visualization (Graphviz). The data can be further analyzed after the script is done. This information utilizes the SFDX query mechanism as outlined by Andy Fawcett, here.

Parsed Directory

This directory holds parsing information and used to allow for debugging, if needed.

PNG Directory

This directory contains all the converted dot files into SVG. The dot files (i.e. *apex_dot.dot*) are left in for further manipulation (as you can also generate PNG files, PDF, files etc.).

SVG files are easier to work with as you can show the visualization via a browser (i.e. Chrome). For example, a partial display of visual-force SVG file (within a Chrome Browser):

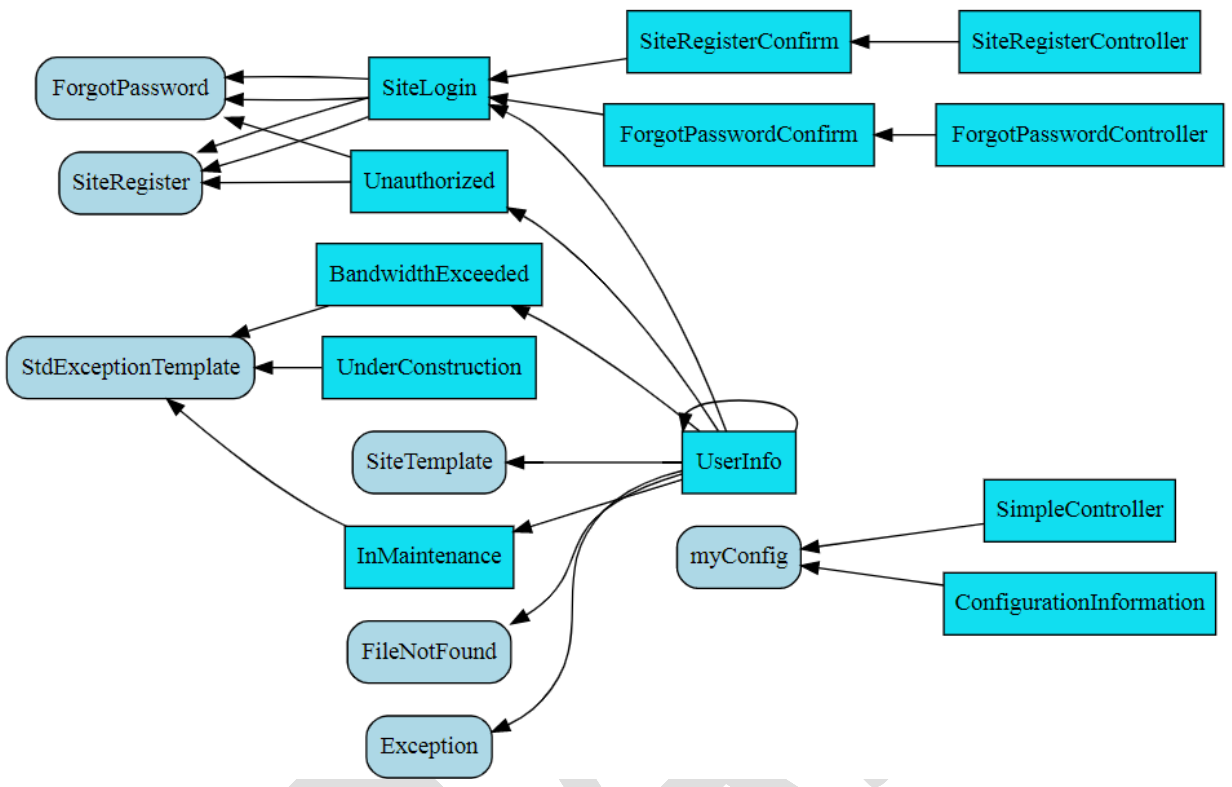


Figure 2 Partial Display of Static Resource SVG

When to Use?

Use this script to get an overview of a Source Org. The spreadsheets and visualization will help assist in breaking the metadata into appropriate areas of the Capability Squads.

buildScratchOrg.sh

```
Usage: buildScratchOrg.sh [ -u <source|username> | -s <scratch-org-user>| -f | -d | -h ]

        -s <scratch-org-user>
        -u <source|username> [Source of a NON-Scratch-Org, if any]
        -f [ensures the FSC package is installed]
        -d debug
        -h the help
```

This script builds a scratch org based on installed source Org. A Scratch Org can contain many features and required packages (i.e. FSC). Instead of guessing how and what to do, this script builds a scratch org for the user. At the time of this writing, *data is NOT loaded into the Scratch Org – **TBD***.

The script takes a scratch-org user (***-s <test*>***), if known, otherwise, it creates a scratch org and applies the features and installs required packages into the scratch Org.

Command-line Arguments

Name	Comments
-s <test*>	If an scratch org has already been created, you can pass that user-id (test-flfhb460d6df@example.com) and it will install into that scratch org. However, any features not enabled may cause a conflict. The scratch Org will expire after 2 days.
-u <source-org>	If there is a source Org (non-scratch Org) which contains installed packages you want in your scratch Org; include it via the **-u** option. The script will read all the installed packages are attempt to replicate in your scratch org. Note, it ignores the package ID for the Financial Service Cloud to used ensure you are using the latest.
-f	Ensures the latest Financial Service Cloud (FSC) package is installed. You would ONLY need this flag if: 1) Your source org did not contain the FSC package, or 2) The common install package file did not include the FSC package name
-d	Runs the script using **set -xv**
-h	Help

General Comments

- The script pulls from a list of packages names (*res/installPackages.txt*). Note, there has to be an empty (new-line) at the end of this file. If the package id is known (04t*) it would appear first (followed by semi-colon [**:**]). If there is a source org specified, the script will use the org as a reference and determine the package id, if not present. For example, the *installPackage.txt* (below) does not specify the *Financial Services Cloud,* as it pulls the latest.

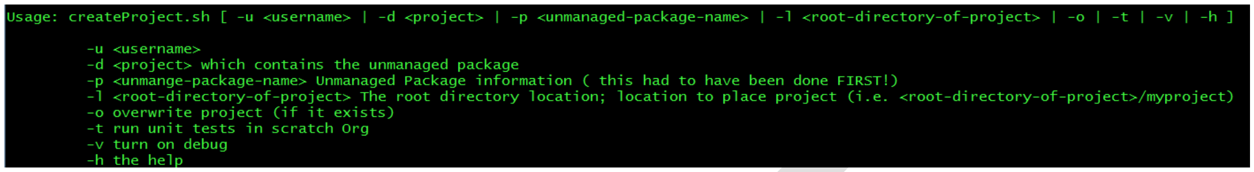
```
04t30000000bqOUYAI:Knowledge Base Dashboards & Reports
04t390100001DUvrAAG:Salesforce Connected Apps
04t410000009jsLKAA:Salesforce Adoption Dashboards
:Financial Services Cloud
:Financial Services Ext
```

- The script will check for the latest version of FSC (*Financial Services Cloud*) and install it. This is done because an FSC extension or feature may be deprecated. Thus, does not depend on the package id of the non-Scratch Org.
- The script does not require a *VS Code Project Folder* nor *sfdx-project.json* file as is the case for many of the **SFDX** CLI commands.
- The script uses a non-scratch org instance (if included **-u <source-org>**) to determine what packages to be included. It uses the **SFDX** command (*force:package:installed:list*) along with the file, *installPackages.txt,* to determine what packages are required
- Depending on the number of packages to install, the script may take well over 20 minutes to install. For example, the list found in the example above, the installation took ~20 minutes.

When to Use?

Use this script to initialize a Scratch Org for development. Some development teams require features and packages to be installed in order to test and validate. For example, many financial companies use Financial Service Cloud (FSC) which must be installed before installing dependent components.

createProject.sh



```
Usage: createProject.sh [ -u <username> | -d <project> | -p <unmanaged-package-name> | -l <root-directory-of-project> | -o | -t | -v | -h ]

        -u <username>
        -d <project> which contains the unmanaged package
        -p <unmange-package-name> Unmanaged Package information ( this had to have been done FIRST!)
        -l <root-directory-of-project> The root directory location; location to place project (i.e. <root-directory-of-project>/myproject)
        -o overwrite project (if it exists)
        -t run unit tests in scratch Org
        -v turn on debug
        -h the help
```

This script will build a VS Code Project. It provides the ability to bring down an unmanaged package as a starting point. However, if there is no pre-defined package just enter return when prompted.

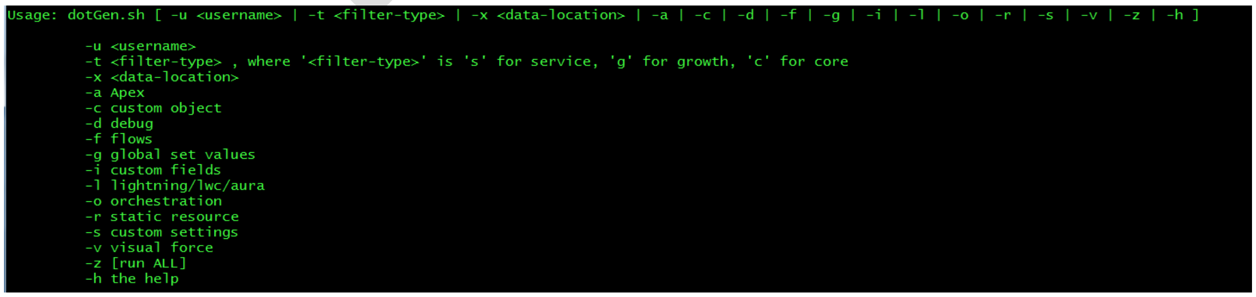
Command-line Arguments

Name	Comments
-u <source-org>	The source Org (non-scratch Org) which contains the unmanaged package, if any. This Org is the non-scratch Org you VS Code project will authenticate/attach to.
-d <project>	The project name which will contain your Visual Studio Code project.
-p <unmanaged-package-name>	If a unmanaged package was created in the Source Org the script pulls down the package into your Visual Studio Code project
-l <root-directory>	Root directory location for the Visual Studio Code project and related information
-o	Overwrite information in the project directory
-t	Run Unit Tests when the metadata is pulled down.
-v	Runs the script using **set -xv**
-h	Help

When to Use?

Use this script to initialize a Visual Studio Code project from a unmanaged package in a non-scratch Org.

dotGen.sh



```
Usage: dotGen.sh [ -u <username> | -t <filter-type> | -x <data-location> | -a | -c | -d | -f | -g | -i | -l | -o | -r | -s | -v | -z | -h ]

        -u <username>
        -t <filter-type> , where '<filter-type>' is 's' for service, 'g' for growth, 'c' for core
        -x <data-location>
        -a Apex
        -c custom object
        -d debug
        -f flows
        -g global set values
        -i custom fields
        -l lightning/lwc/aura
        -o orchestration
        -r static resource
        -s custom settings
        -v visual force
        -z [run ALL]
        -h the help
```

This script parses the metadata dependency files (generated by ***md_used.sh***, and called from ***analyzeOrg.sh***) to generate a visual dependency SVG graph (from a dot file). This script will be used to support filtering of metadata (i.e. search for Services related metadata, ***CR_*** prefix). The SVG files are placed in data location (***-x <directory>***).

This script will create two directories:

- **png** – holds the SVG files and DOT related file
- **parsed** – holds the parsed data used to generate the DOT file

This script will read from a **data** directory (either in the current directory or the **-x** <root-dir>).

It is expected that either **md_used.sh** or **analyzeOrg.sh** are run prior to this script. Why? The script looks for the metadata dependency information in the **data** directory. For example, if you run this command from your current directory, and there is a ./**data** directory (i.e. **./data**) which contains information previously created by either **md_used.sh** or **analyzeOrg.sh.**

Command-line Arguments

Name	Comments
-u <source-org>	The source Org (non-scratch Org) which contains the Org you want to run analysis on.
-t [c|s]	Filters the data pulled from the Org. At the time of this writing it supports : • **c** for Core components • **s** for Service components
-x <root-directory>	The root directory where it will find the metadata dependency information directory, **data**. For example, if you run the script from a directory (/c/myHome), it will look for **/c/myHome/data** to hold the metadata dependency information. [***-x /c/myHome***].
-a	Process Apex related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-c	Process Objects related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-g	Process Global Set Values related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-i	Process Custom Fields related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-l	Process Lightning related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-o	Process orchestration related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.

Name	Comments
-r	Process Static Resources related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-s	Process Custom Settings related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-v	Process VisualForce related data (generate SVG files placed in **png** directory). Parsed data from the metadata files are placed in **parsed** directory.
-z	Process ALL metadata information
-d	Runs the script using **set -xv**
-h	Help

General Comments

- The script uses Graphviz (dot) to generate SVG files. Note, from a dot file (i.e. apex.dot) you can also generate other output (PNG, PDF, etc.). More information of what you can do with DOT files can be found here.

  # To generate a SVG file from a dot file (i.e. apex.dot)

  - **dot** -Tsvg -o apex.svg apex.dot

  # To generate a PNG file from a dot file (i.e. apex.dot)

  - **dot** -Tpng -o apex.png apex.dot

- The script reads generated metadata dependencies (see **md_used.sh** script) found in **data** directory.
- The script visual output can be modified as needed. One could got to the **png** directory and change the data in the dot file (i.e. *apex.dot*).

When to Use?

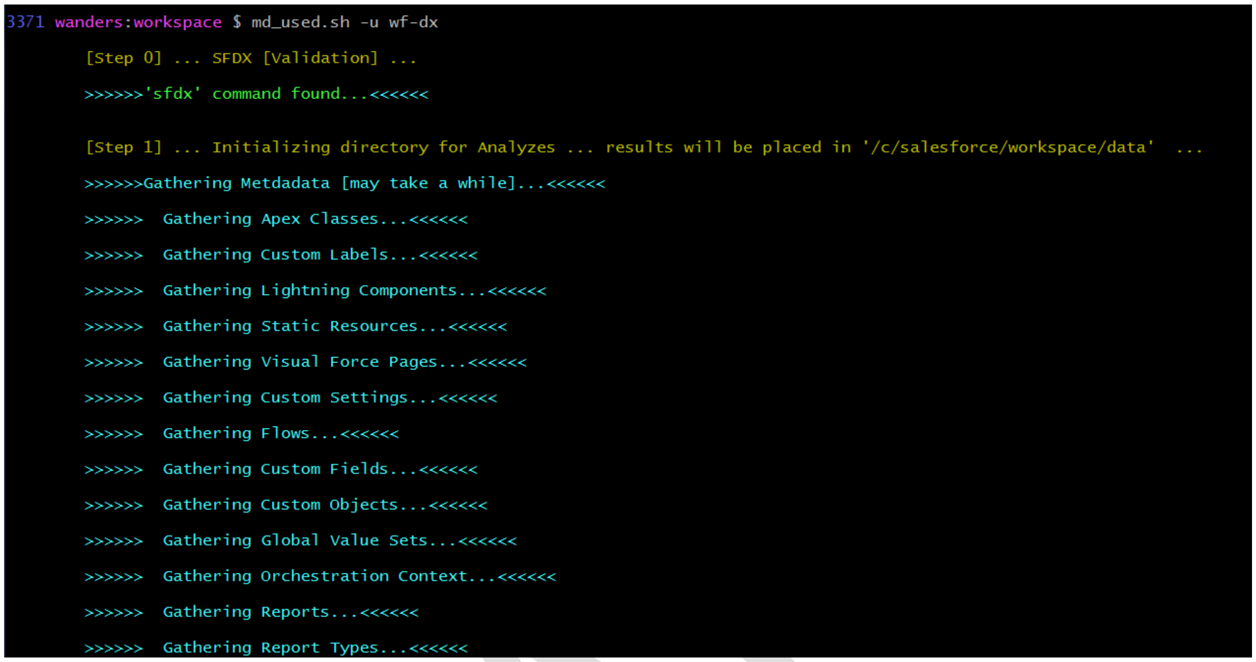
Use this script when you want to create graph dependency visualizations from metadata dependency files from a Salesforce Org. Note, it looks for specific names to know how to process. Use this script after you ran **md_used.sh** or invoked via **analyzeOrg**.sh

md_used.sh

```
Usage: md_used.sh [ -u <username|target-org> | -l <data-location> | -d |  -h ]

        -u <username|target-org>
        -l <data-location> [will create, if not present]
        -d debug
        -h the help
```

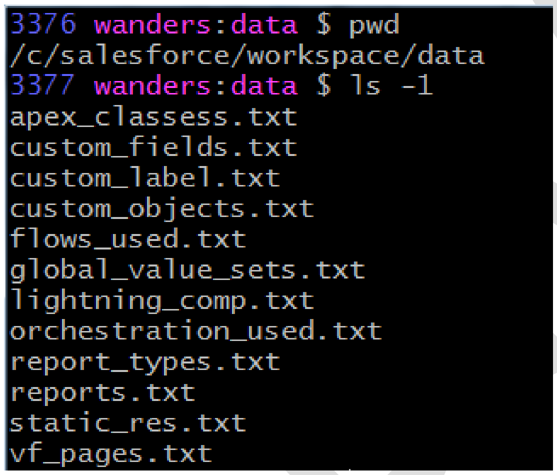
This script pulls the metadata dependency information from the specified Org. By default, the script with create a data directory (**data**) in the current working directory (**./data**).

For example, (assuming I have already authenticated to the an Org with an alias of ***wf-dx*** and my current working directory is **/c/salesforce/workspace**:



```
3371 wanders:workspace $ md_used.sh -u wf-dx
        [Step 0] ... SFDX [Validation] ...
        >>>>>>'sfdx' command found...<<<<<<

        [Step 1] ... Initializing directory for Analyzes ... results will be placed in '/c/salesforce/workspace/data'  ...
        >>>>>>Gathering Metdadata [may take a while]...<<<<<<
        >>>>>>  Gathering Apex Classes...<<<<<<
        >>>>>>  Gathering Custom Labels...<<<<<<
        >>>>>>  Gathering Lightning Components...<<<<<<
        >>>>>>  Gathering Static Resources...<<<<<<
        >>>>>>  Gathering Visual Force Pages...<<<<<<
        >>>>>>  Gathering Custom Settings...<<<<<<
        >>>>>>  Gathering Flows...<<<<<<
        >>>>>>  Gathering Custom Fields...<<<<<<
        >>>>>>  Gathering Custom Objects...<<<<<<
        >>>>>>  Gathering Global Value Sets...<<<<<<
        >>>>>>  Gathering Orchestration Context...<<<<<<
        >>>>>>  Gathering Reports...<<<<<<
        >>>>>>  Gathering Report Types...<<<<<<
```

After the script finishes, there will be a number of files **(.txt)** placed in **./data** directory, as shown below:



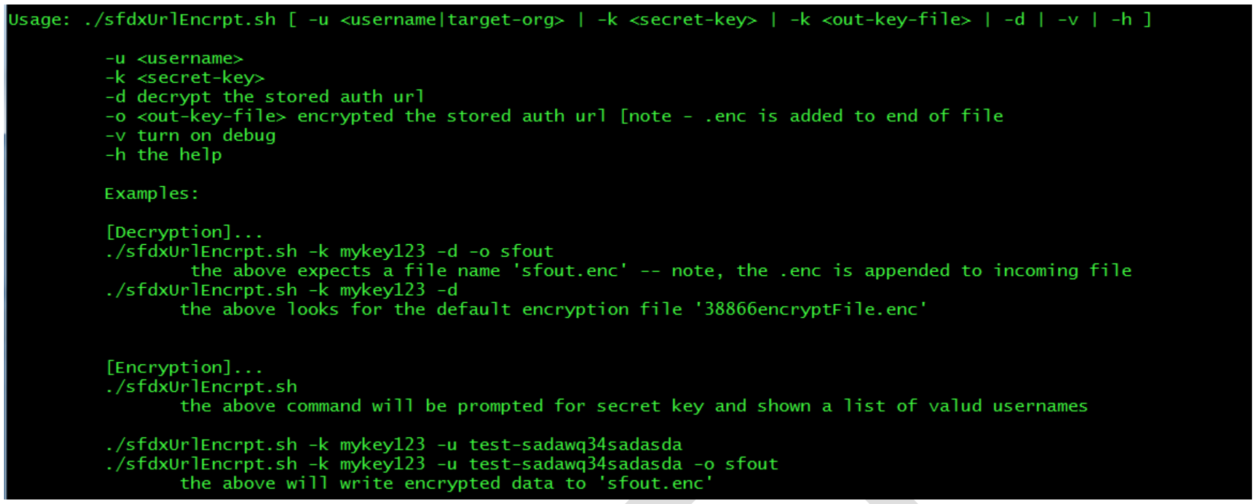
```
3376 wanders:data $ pwd
/c/salesforce/workspace/data
3377 wanders:data $ ls -1
apex_classess.txt
custom_fields.txt
custom_label.txt
custom_objects.txt
flows_used.txt
global_value_sets.txt
lightning_comp.txt
orchestration_used.txt
report_types.txt
reports.txt
static_res.txt
vf_pages.txt
```

The script, **dotGen.sh**, will read this information and create a dependency graph.

When to Use?

Use this script to generate metadata dependency files of a Salesforce Org. Note, it generates specific files that **dotGen.sh** looks for.

sfdxUrlEncrpt.sh



```
Usage: ./sfdxUrlEncrpt.sh [ -u <username|target-org> | -k <secret-key> | -k <out-key-file> | -d | -v | -h ]

         -u <username>
         -k <secret-key>
         -d decrypt the stored auth url
         -o <out-key-file> encrypted the stored auth url [note - .enc is added to end of file
         -v turn on debug
         -h the help

         Examples:

         [Decryption]...
         ./sfdxUrlEncrpt.sh -k mykey123 -d -o sfout
                 the above expects a file name 'sfout.enc' -- note, the .enc is appended to incoming file
         ./sfdxUrlEncrpt.sh -k mykey123 -d
                the above looks for the default encryption file '38866encryptFile.enc'


         [Encryption]...
         ./sfdxUrlEncrpt.sh
                the above command will be prompted for secret key and shown a list of valud usernames

         ./sfdxUrlEncrpt.sh -k mykey123 -u test-sadawq34sadasda
         ./sfdxUrlEncrpt.sh -k mykey123 -u test-sadawq34sadasda -o sfout
                the above will write encrypted data to 'sfout.enc'
```

This script encrypts/decrypts the **sfdx** stored-url (ascii file). The data contained in this file allows a tokenized access to an Org.

When to Use?

Use this script to generate a protected/encrypted stored-url file.

Appendix: Comments

Below are some general comments:

- The scripts will evolve over time. For example, **md_used.sh** will be updated to pull more metadata.
- The script **dotGen.sh** does not create a dependency graph for ALL metadata. This is a work in progress [WIP]
- SFDX CLI is evolving and some scripts may need to be adjusted
- The scripts could be refactored; placing commonly functionality into a separately source shell script.