Bill Anderson

Bill.anderson@salesforce.com

Abstract

Simple steps using SFDX to Package multiple components

Sample DX Packaging

Multiple Packages within a Project

# Multiple Packages

This is a simple document that outlines steps taken to create multiple 2GP unlocked packages with the same project file. The steps are outlined along with the completed result. We package only two of the components (**Core** and **Growth-1**) and leave the final one, *Common*, to the user as an exercise.

Note, **Core** represents a base utilized by the other packages. **Growth-1** represents a Development Team and their related work. Not ALL functionality needs to be package. You could have another folder (i.e. ***crm/sales, crm/service, crm/marketing, …***) which represents all other (non-package) source of your project. The point is to organize your project[[1]](#footnote-1) that best reflects the Teams understanding. If you are NOT responsible for the other content, do not include it!

# General Comment

Packaging process can be broken down in steps. As SFDX is evolving, some steps may change over time. Thus, this document will need updating.

In addition, some of the larger questions need to be answered (and not covered here):

* Who owns the content? Or, is it a shared resource?
  + Should Governance, ARB, or some Committee decide ownership or shared resource?
* Does the content require sign-off and/or governance (i.e. Data Architect for compliance, etc.)?
* Naming Conventions (package names, version numbers, etc.)

For example, should all custom and standard objects be owned and controlled by Central/Core group in order to ensure proper governance is followed as prescribed by the Data Architect and/or Enterprise Architect?

These are general comments require a larger, and higher-level, discussion.

# Knowns

This sample project has already gone through the process of **creating a DX project**:

sfdx force:project:create --projectname sample-mpd-project --defaultpackagedir package --manifest

If you view the ***MyWork*** sub-folder, its (salient) contents are as followed:



| Name | Description |
| --- | --- |
| config | Holds the Scratch Org Configuration Information |
| manifest | Holds package.xml (for MDAPI retrieval) |
| package | Our Content |
| part2 | Holds the second part of the sample content (**growth-1**) |
| part3 | Holds the third part of the sample content (**commons**) |

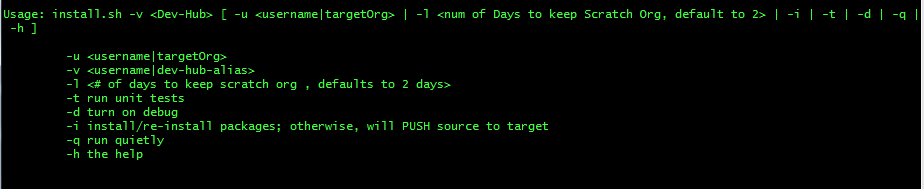
# Initial Steps

The first step assumes you are using the "**MyWork**" sub-folder to create a Scratch Org. Because I often do this task, I have written scripts to expedite the process.

## Step 1 – Create Scratch Org

For Creating a Scratch and initial data, I use **./scripts/init/install.sh**. For all the command line options pass in "-h":

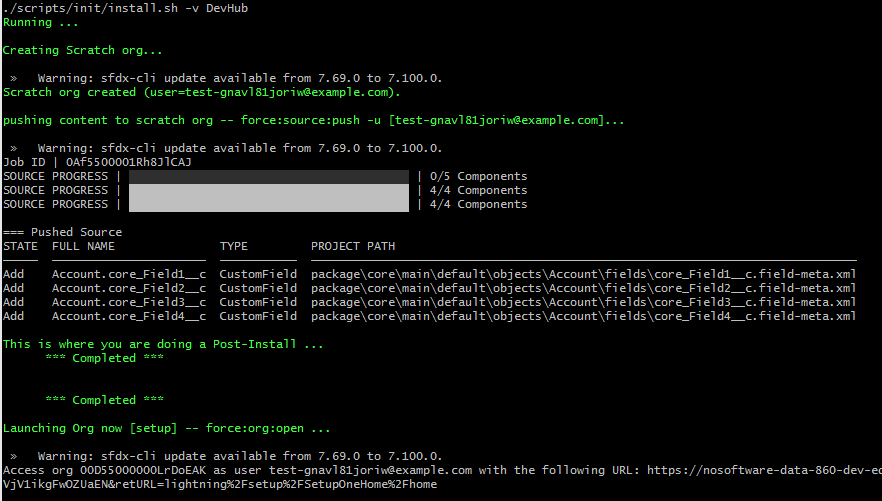
*./scripts/init/install.sh -h*



From the top-level directory (**MyWork**), run the following command (which creates the Scratch Org and pushes all content:

**./scripts/init/install.sh -v *<Dev-Hub-Name>***

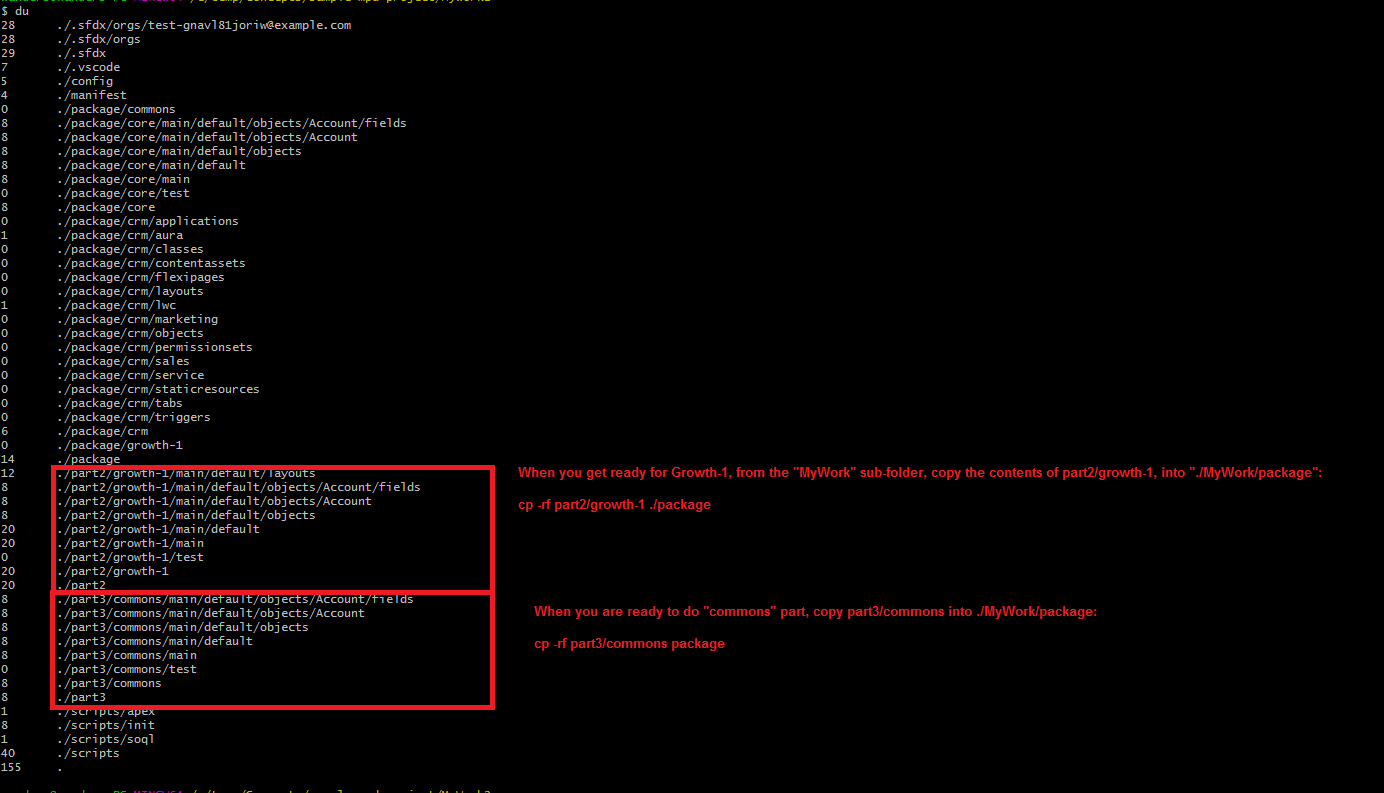
(Please note, my *<Dev-Hub-Name>*, is alias to **DevHub.**



(Note, if you ALREADY created a Scratch Org, you can pass that value into the command *./scripts/init/install.sh -u <scratch-org-name> -v <Dev-Hub-Name>)*

The above script, (1) Created a Scratch Org, (2) pushed the contents of **core** into the Scratch Org.

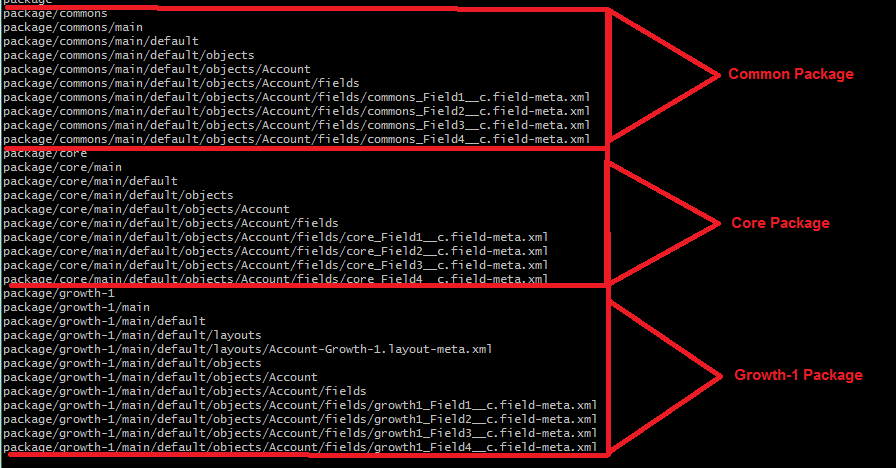
Your initial contents of **MyWork**, is reflected below:



# Sample

In the folder, **sample-mpd-project**, we have three components which we will package 2 of them with steps in this document:

* **Core** (to be Packaged) [*Part-1*]
* *Common* (will not Package in this scenario, but can) [*Part-3*]
* **Growth-1** (to be Packaged) [*Part-2*]



The above represents the final structure of the source. This will evolve as we progress in the DX packaging. The sections which follow go through the process of creating two packages. All these two packages just create new Fields on the Account Object. One could go back later and create classes and triggers, etc. But we leave that as an exercise for the user.

The user will also note, there is no iteration(s) of unit testing to do other than validate the Account fields pushed into the Scratch Org. If there are Flow, Process Builder, Workflows, Apex, etc. related actions, you will need to write, test and ensure proper coverage!

This document does not go into getting access to a Dev Hub nor the steps in creating a Scratch Org. The Scratch Org will be working with has an alias name of **mpd**.

Below is the basic flow in creating our packages:

# SFDX-Project.JSON

At the beginning we have a basic structure for our *sfdx-project.json*. This will evolve as we create packages.

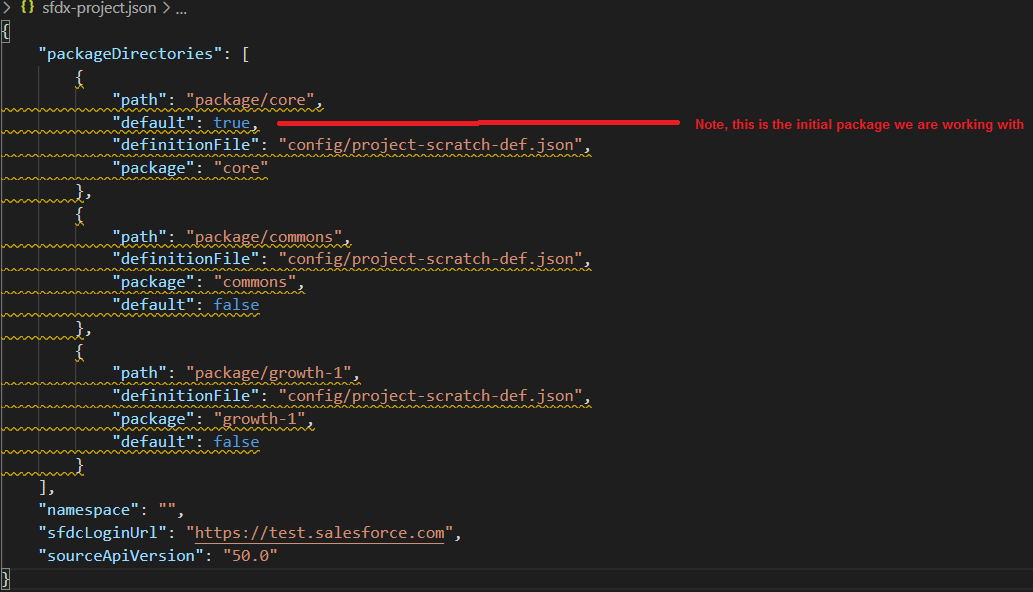


Figure Beginning Project File (**sfdx-project.json**)

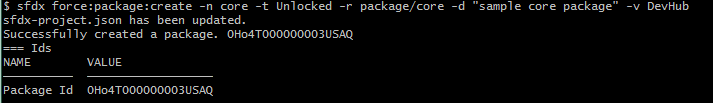
Please note, there as we progress with creating and promoting packages our **sfdx-project.json** will be updated.

# Package CORE

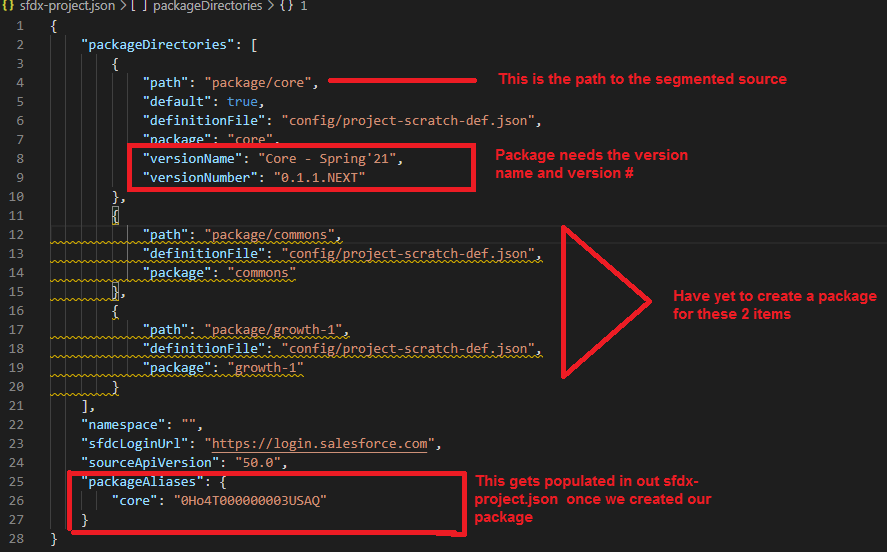
Please note, initially, in **MyWork** sub-folder **core** is already created with Account Fields. We will just create the package, package version, validate and the promote.

## Create a Package

**Create a package named core.**



After we have created our package, ***sfdx*** command updates the *sfdx-project.json* file.



Please note, before we created the package, we added line 6-9:

* Line-6 – definition of the Org Shape
* Line-7 – package name
* Line-8 – package version name
* Line-9 – package version number

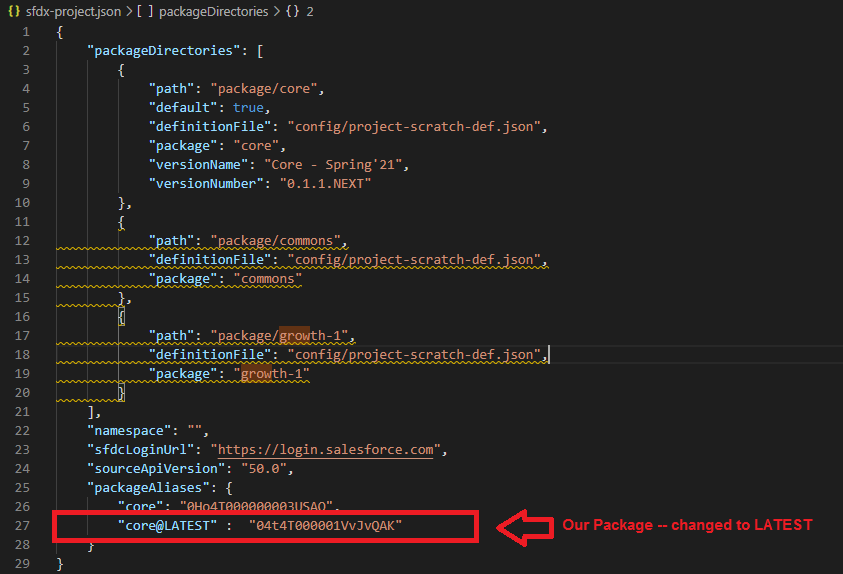
After the created package command was done, ***sfdx*** added lines 25-27.

## Create a Package Version

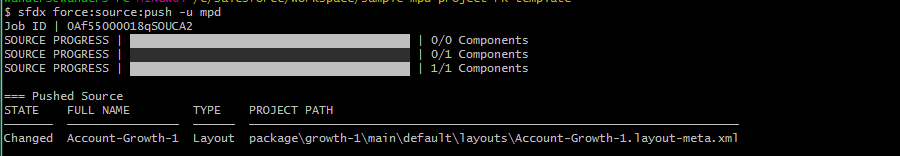
Next, we create the package version of **core**.



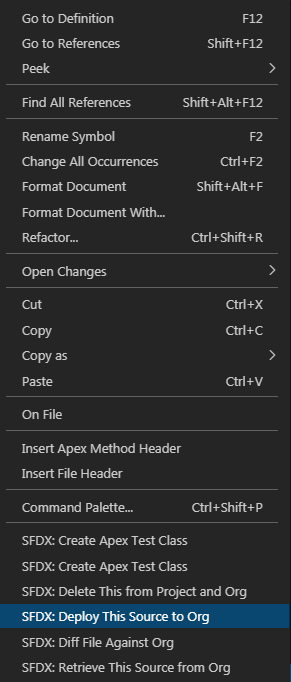
After we created the package version, ***sfdx*** updates the *sfdx-project.json* file as follows:



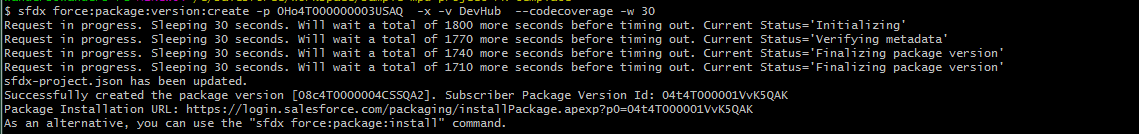
Next, I made some changes to the Account layout in my project and pushed to the Scratch Org (for Growth-1). This will not affect this package.



Developers will go back and forth pull/push scenario with a Scratch Org. If using a non-Scratch Org, you can use the Retrieve/Deploy options in Visual Studio Code.

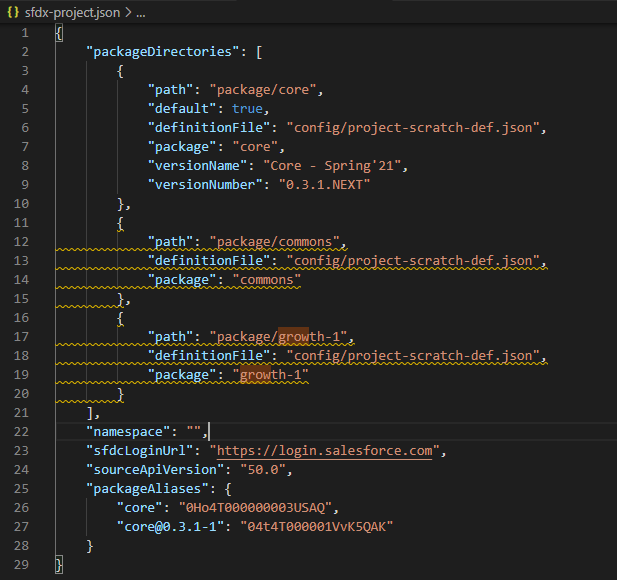


Once we are done, unit tested with code-coverage >= 75%, we can promote. Please note, initially when we created our package version, **we DID NOT** include code coverage. Why? We were in the process of updating our Beta version and I had not validated nor tested in my Scratch Org. Once we have tested and validated, we are ready to create a version to promote. But first one **MUST** create a version with code-coverage option!





The *sfdx-project.json*, after promotion of core, looks like:



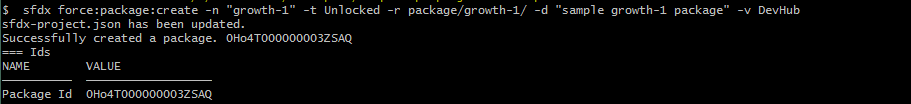
# Package Growth-1

We have packaged **Core**, now, let us package **Growth-1**. **Growth-1** adds new fields and a layout to the *Account*.

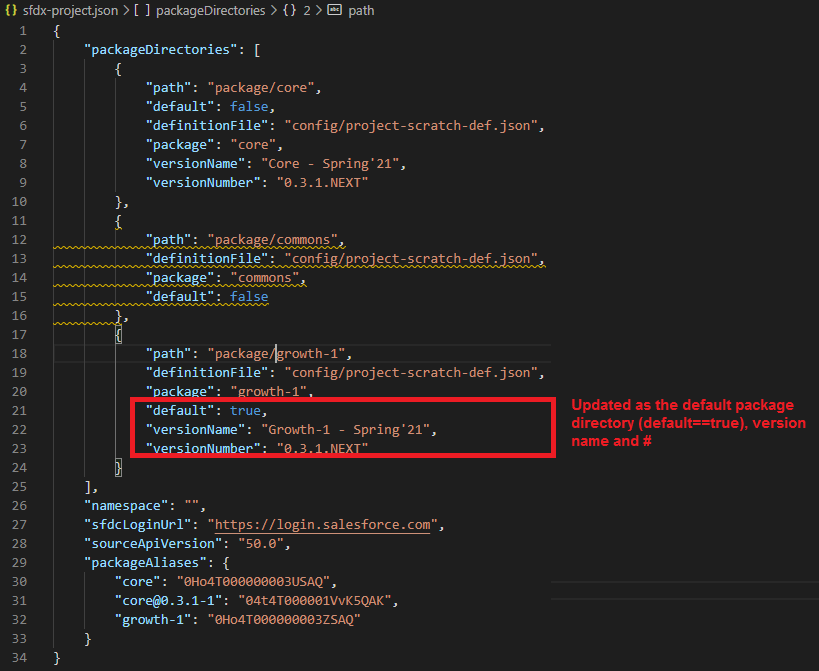
## Create a Package

Here we create a package named **growth-1**. Note, we specified **codecoverage** flag.

sfdx force:package:version:create -p 0Ho4T000000003ZSAQ -x -v DevHub --codecoverage -w 30

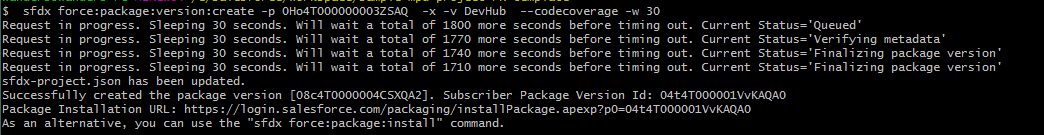


After we have created our package, ***sfdx*** command updates the *sfdx-project.json* file.



## Create a Package Version

Next, we create the package version of **growth-1**. Please note, for this version we are adding the ***codecoverage*** switch.

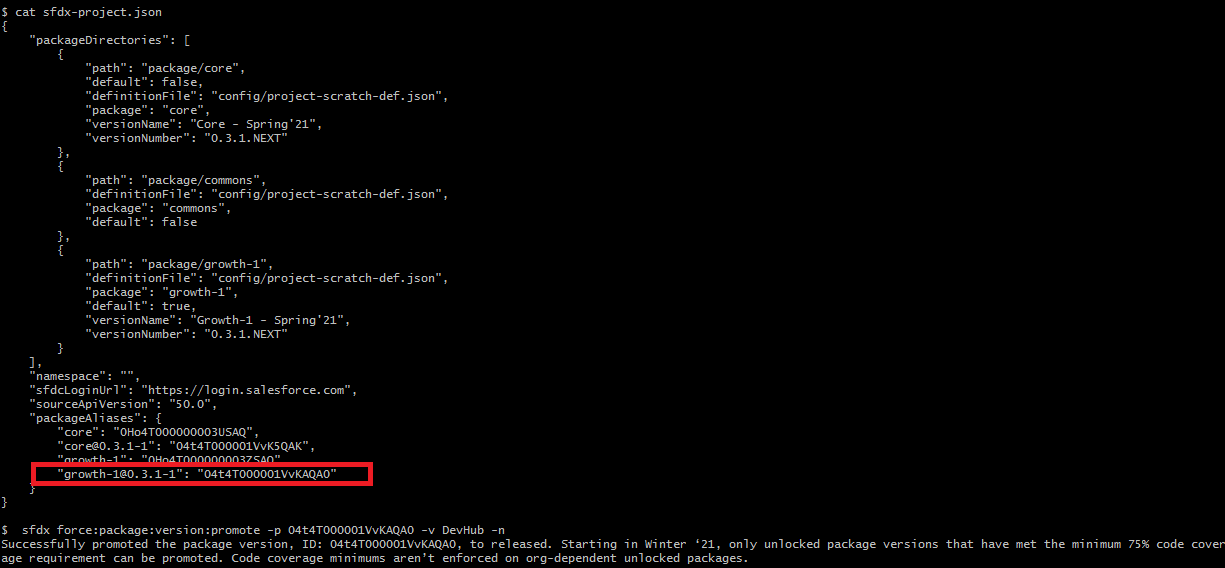


After we created the version, sfdx updates the *sfdx-project.json* file as follows:



Because we used the ***–codecoverage*** switch, we can now, promote our growth-1 package,

sfdx force:package:version:promote -p **04t4T000001VvKAQA0** -v DevHub -n



# Appendix: Zipped file of Project

Contents of the project.



1. You can always refactor the folders as needed. [↑](#footnote-ref-1)