Experiments Done to Maintain AOSP Source Code Hosted On Google's External Server on BTC's Internal Git Server (Stash)

Author: Nishit Dabi

Date: 22 April 2020

Prerequisite

For downloading the AOSP code Google uses the `repo` tool. This tool is helpful for downloading, syncing, maintaining the directory structure of the AOSP's source code, and has many additional features as well. For more information about Repo, see the Repo Command Reference and Repo README.

The `repo` tool has two parts, one is the launcher script (based on python3, so make sure that your system supports python3) and the other being the full `repo` tool which is automatically downloaded when we do a source checkout.

To download the `repo launcher script` do the following:

1. Go to (or create) the desired location where you want to download the `repo` tool.

```
cd /desired/location
```

2. Add the desired location in the `PATH` shell variable permanently, by modifying your `~/.bashrc` (or `~/.zshrc` if you use ZSH) as follows:

```
vi ~/.bashrc
# Now add the following line at the bottom of the file
export PATH=$PATH:/desired/location
```

3. Download the `repo launcher` at the `/desired/location` and make it an executable

```
curl https://storage.googleapis.com/git-repo-downloads/repo >
~/desired/location/repo
chmod a+x ~/desired/location/repo # This line will make the repo an
executable
```

4. Open a new terminal and make sure the `repo launcher` is installed by checking its version

```
repo --version
```

```
# Sample output as follows
repo version v2.5
       (from https://gerrit.googlesource.com/git-repo)
repo launcher version 2.5
       (from /mnt/work/bin/repo)
       (currently at 2.5)
repo User-Agent git-repo/2.5 (Linux) git/2.17.1 Python/3.6.9
git 2.17.1
git User-Agent git/2.17.1 (Linux) git-repo/2.5
Python 3.6.9 (default, Nov  7 2019, 10:44:02)
[GCC 8.3.0]
OS Linux 4.15.0-96-generic (#97-Ubuntu SMP Wed Apr 1 03:25:46 UTC 2020)
CPU x86_64 (x86_64)
```

The `repo launcher script` is now install on the system and we are ready to proceed with downloading the AOSP source code.

Method 1 - Downloading the multiple repositories of the AOSP and squashing them in a single repo

In this method the AOSP source code was downloaded on a Linux-PC using the Google's `repo` tool and all source code in multiple repositories of the AOSP was pushed to a single stash repo.

Steps Followed

1. Create a folder called `aosp` at the location where the AOSP code will reside.

   ```
   cd /path/to/desired/location
   mkdir aosp
   ```

2. Run `repo init` to get the latest version of Repo with its most recent bug fixes. You must specify a URL for the manifest (`manifest.xml`), which specifies where the various repositories included in the Android source are placed within your working directory.

   ```
   repo init -u https://android.googlesource.com/platform/manifest
   ```

   To check out a branch other than *master*, specify it with `-b`. For a list of branches, see Source code tags and builds.

   ```
   repo init -u https://android.googlesource.com/platform/manifest -b
   android-4.0.1_r1 # this will initialize the repo to download android-
   4.0.1_r1 source code
   ```

   A successful initialization ends with a message stating that Repo is initialized in your working directory. Your client directory should now contain a `.repo` directory where files such as the manifest are kept.

3. *Optional step:* The `default.xml` file which is included in the `manifest.xml` can be modified to download only a few repos for experimentation. The path for `default.xml` will be `aosp/manifests/default.xml`. In my experiment the `default.xml` was as follows (initial experimentation):

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <manifest>
3
4    <remote  name="aosp"
5             fetch=".."
6             review="https://android-review.googlesource.com/" />
7    <default revision="master"
8             remote="aosp"
9             sync-j="4" />
10
11   <manifest-server url="http://android-
   smartsync.corp.google.com/android.googlesource.com/manifestserver" />
12
13   <project path="build/make" name="platform/build" groups="pdk" >
14     <copyfile src="core/root.mk" dest="Makefile" />
15     <linkfile src="CleanSpec.mk" dest="build/CleanSpec.mk" />
16     <linkfile src="buildspec.mk.default"
   dest="build/buildspec.mk.default" />
17     <linkfile src="core" dest="build/core" />
18     <linkfile src="envsetup.sh" dest="build/envsetup.sh" />
19     <linkfile src="target" dest="build/target" />
20     <linkfile src="tools" dest="build/tools" />
21   </project>
22   <project path="build/blueprint" name="platform/build/blueprint"
   groups="pdk,tradefed" />
23   <project path="build/soong" name="platform/build/soong"
   groups="pdk,tradefed" >
24     <linkfile src="root.bp" dest="Android.bp" />
25     <linkfile src="bootstrap.bash" dest="bootstrap.bash" />
26   </project>
27
28 </manifest>
```

   The above `default.xml` will let the `repo` know that we only want to download three project repos, viz., `make`, `blueprint`, and `soong`; with their destination path being `aosp/build/make`, `aosp/build/blueprint`, and `aosp/build/soong` respectively. The `revision` at line number 7 in the above example can be changed to any of the tags so as to make the sync happen at that tag specifically.

4. To download the Android source tree to your working directory from the repositories as specified in the default manifest, run:

```
1  repo sync
```

   This will take a couple of hours depending on your internet connection speed and the number of projects in the `default.xml` file.

5. Now, to squash all the repos of AOSP into a single stash (internal) repo, we will go ahead and create a repo in stash first (**Administrator prermissions required for the corresponding project**). Log-in to your stash account and then click on *Create repository*, go ahead and fill all the required details and copy the repo URL which will look

something like https://intstl-stash01.dtc.dish.corp/scm/aosp/your-newly-created-repo-name.git

6. Go to the AOSP folder and do a `git init` there to initialize a new git repo.

```
cd /path/to/desired/location/aosp
git init
```

7. Now, we will link this newly created local git repo with our internal-stash-repo created in the above step 5.

```
 git remote add origin https://intstl-stash01.dtc.dish.corp/scm/aosp/your-newly-created-repo-name.git
```

8. Now, find all directories which contain a `.git` folder. The presence of a `.git` folder implies that the folder is in-fact a repository. So if we directly do a `git add --all` all the files in sub-folders which contain a `.git` folder will not be added since they should be added as a `git submodule`. But our requirement is to squash all the sub-repos into a single repo. So, we will find out all the folders containing a `.git` folder and them to our local-repo (which we just initalized) one-by-one.

```
find -iname '.git'
# The sample output for the above manifest looks as follows:
./.git
./.repo/repo/.git
./.repo/manifests/.git
./build/blueprint/.git
./build/soong/.git
./build/make/.git
```

In the above example output, the `./.git` corresponds to the local-repo that we initialized in step 6. `./.repo/repo/.git` corresponds to the `repo tool source` with which the `repo launcher script` interacts with. `./.repo/manifests/.git` corresponds to the repo which contains the manifest information, and the rest three corresponds to the project source code that we downloaded by `repo sync`. It is not required to add the `repo` and `manifests` repos to the internal-stash-repo as these are only required by the local-machine, which will be handling the business of syncing. Finally add all the remaining code

9. To, add the project repositories to the internal-stash-repo we need to do the following for each AOSP repo that got listed in step 7.

```
git add build/blueprint/
git add build/soong/
git add build/make/
```

NOTE: Make sure to use the `/` (forward-slash) after the directory name which contains a `.git` folder. Otherwise it will try to get add as a sub-module but fail.

10. After adding all the repos of AOSP to our local-repo, we need to commit the changes, link the local-repo to the internal-stash-repo (created in step 5), and finally push the code to stash.

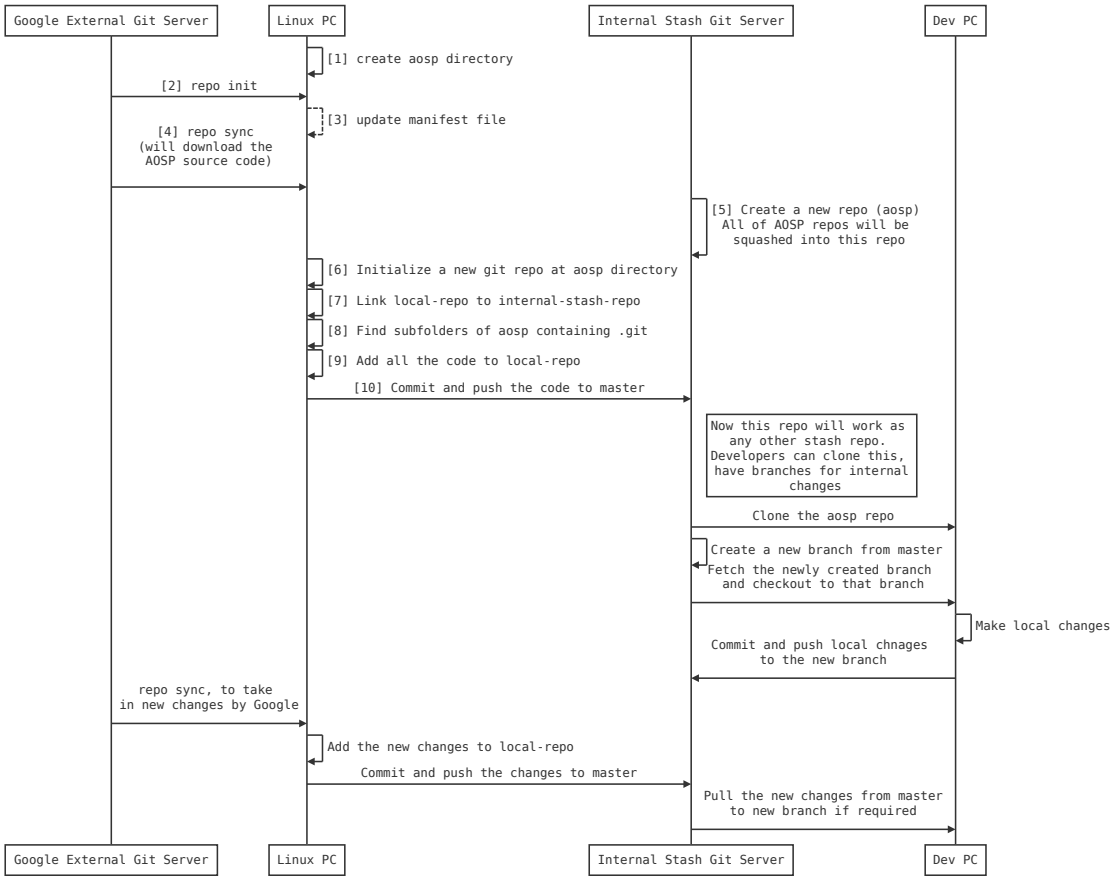
```
git commit -m 'Added blueprint, soong and make repos'
git push -u origin master
```

11. Now a user of the internal-stash-repo can go ahead and clone it, make branches, push changes to those branches, and perform all the other operations just like any other repository on stash. All the local-modifications will be saved and we can take in new changes from Google as well. To get an understanding of this please see the commits and branches of the following repo https://stash.dtc.dish.corp/projects/EV2/repos/android-experiment/browse

---

Sequence Diagram

NOTE: The numbering in the diagrams to follow matches to the steps provided for that method. For links without a numbering, it is assumed that the reader of this document has an understanding of *git* to perform basic operations



Method 2 - Partial Mirroring

In this method the AOSP source code was downloaded using the `repo` tool and the repositories of AOSP project were pushed to separate internal-stash-server repos by changing the destination of `git push`

Steps Followed

NOTE: Steps 1-4 here are same as that of Method-1

1. Create a folder called `aosp` at the location where the AOSP code will reside.

```
cd /path/to/desired/location
mkdir aosp
```

2. Run `repo init` to get the latest version of Repo with its most recent bug fixes. You must specify a URL for the manifest (`manifest.xml`), which specifies where the various repositories included in the Android source are placed within your working directory.

```
repo init -u https://android.googlesource.com/platform/manifest
```

   To check out a branch other than *master*, specify it with `-b`. For a list of branches, see Source code tags and builds.

```
repo init -u https://android.googlesource.com/platform/manifest -b
android-4.0.1_r1 # this will initialize the repo to download android-
4.0.1_r1 source code
```

   A successful initialization ends with a message stating that Repo is initialized in your working directory. Your client directory should now contain a `.repo` directory where files such as the manifest are kept.

3. *Optional step:* The `default.xml` file which is included in the `manifest.xml` can be modified to download only a few repos for experimentation. The path for `default.xml` will be `aosp/manifests/default.xml`. In my experiment the `default.xml` was as follows (initial experimentation):

```
 <?xml version="1.0" encoding="UTF-8"?>
 <manifest>

   <remote  name="aosp"
            fetch=".."
            review="https://android-review.googlesource.com/" />
   <default revision="master"
            remote="aosp"
            sync-j="4" />

   <manifest-server url="http://android-
smartsync.corp.google.com/android.googlesource.com/manifestserver" />

   <project path="build/make" name="platform/build" groups="pdk" >
     <copyfile src="core/root.mk" dest="Makefile" />
     <linkfile src="CleanSpec.mk" dest="build/CleanSpec.mk" />
     <linkfile src="buildspec.mk.default"
dest="build/buildspec.mk.default" />
     <linkfile src="core" dest="build/core" />
     <linkfile src="envsetup.sh" dest="build/envsetup.sh" />
     <linkfile src="target" dest="build/target" />
     <linkfile src="tools" dest="build/tools" />
   </project>
```

```
22     <project path="build/blueprint" name="platform/build/blueprint"
    groups="pdk,tradefed" />
23     <project path="build/soong" name="platform/build/soong"
    groups="pdk,tradefed" >
24       <linkfile src="root.bp" dest="Android.bp" />
25       <linkfile src="bootstrap.bash" dest="bootstrap.bash" />
26     </project>
27
28   </manifest>
```

The above `default.xml` will let the `repo` know that we only want to download three project repos, viz., `make`, `blueprint`, and `soong`; with their destination path being `aosp/build/make`, `aosp/build/blueprint`, and `aosp/build/soong` respectively. The `revision` at line number 7 in the above example can be changed to any of the tags so as to make the sync happen at that tag specifically.

4. To download the Android source tree to your working directory from the repositories as specified in the default manifest, run:

```
1  repo sync
```

This will take a couple of hours depending on your internet connection speed and the number of projects in the `default.xml` file. In our example this will download (clone) `make`, `blueprint`, and `soong` projects (repositories).

5. Now, we will go ahead and create the corresponding repositories in internal-stash-server. We require 3 repos for out testing purpose `build_make`, `build_blueprint`, and `build_soong`. Note that the repo names are prefixed with the actual path were the repo will finally reside. **NOTE:** This step can be automated, but will require additional permissions on the internal-stash-server for REST-APIs at least once.

6. To push the `make`, `blueprint`, and `soong` repos to the internal-stash-server, we will have to change the destination URL for `git push`. This can be achived as follows:

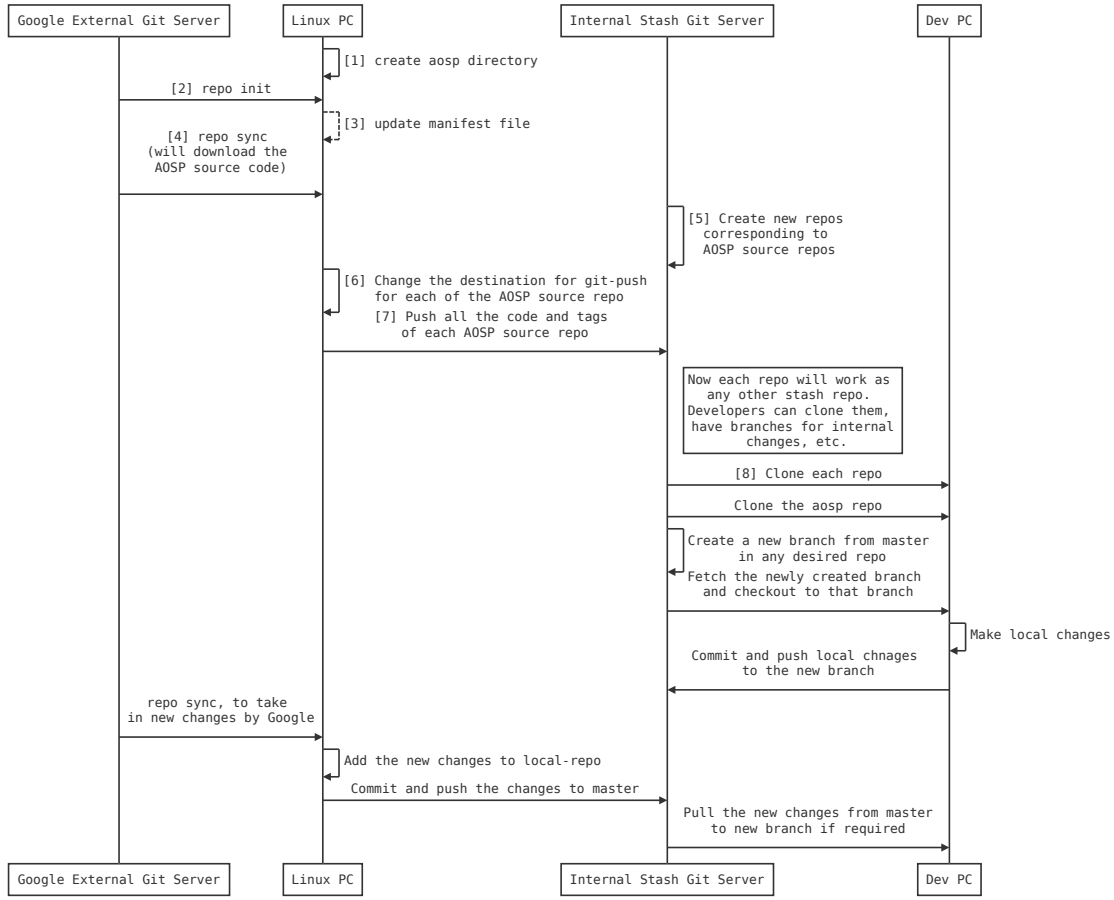
```
1  cd build/make
2  git remote set-url --push aosp https://intstl-
   stash01.dtc.dish.corp/scm/aosp/build_make.git
3  cd build/blueprint
4  git remote set-url --push aosp https://intstl-
   stash01.dtc.dish.corp/scm/aosp/build_blueprint.git
5  cd build/soong
6  git remote set-url --push aosp https://intstl-
   stash01.dtc.dish.corp/scm/aosp/build_soong.git
```

7. Once, the git push destinations are set, we can push all of the code with refs to internal-stash-server repos as follows:

```
1  cd build/make
2  git push aosp master refs/remotes/*:refs/remotes/*
   refs/tags/*:refs/tags/*
```

8. Now, developers can clone the repos from internal-stash-server to their development machines and use these repos like any other stash repo.
   NOTE: Active development should not happen on `master`, the `master` should only be used for pushing in new changes from Google.
   NOTE: The cloning from internal-stash-server will require a script that can clone the repos exactly as Google maintains their source-tree (or directory structure).

Sequence Diagram



Method 3 - Complete Mirroring

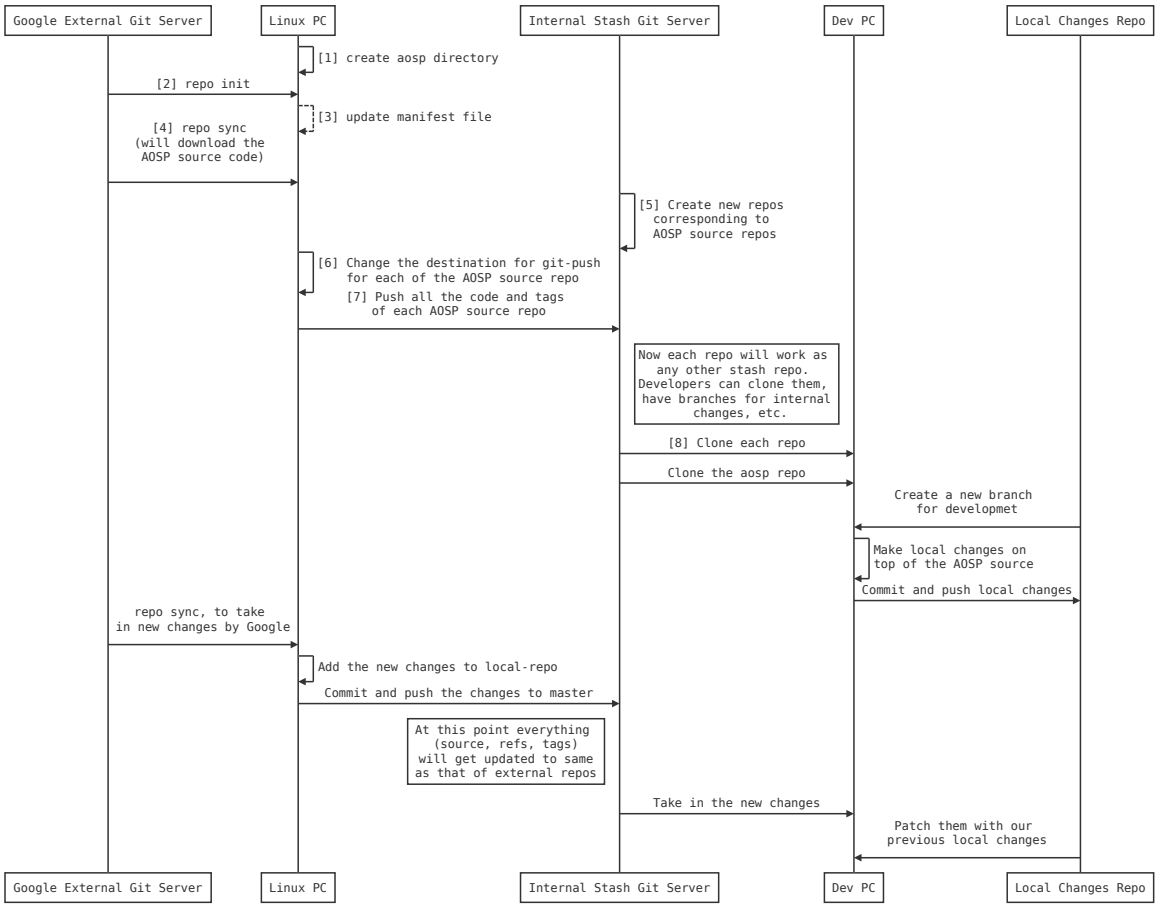
In this method the references of each of the repos in the internal-stash-server will be maintained completely in sync with that of Google's external git server. This way, if any change is made on the Google's external git server it will be reflected on out internal-stash-server and all the local modifications will be deleted.

For this method, everything remains same as Method 2, except Step 7, where for pushing the code to internal-stash-server the following command is used for each of the repos

```
git push --prune aosp master +refs/remotes/*:refs/remotes/*
+refs/tags/*:refs/tags/*
```

For maintaining local changes in internal-stash-server we will require to have a separate repo. Once the sync is done, we can call a script that will patch the local changes to the newly synced AOSP code.

Sequence Diagram



Final Approach Taken Into Consideration

The approach of Method 1 was taken into consideration. The complete AOSP code was synced from Google's external repo to the intermediate Linux-PC and pushed to internal-stash server. For this, the step 3 from the method needs to be skipped.

A script was written which will add all the repos (manually taken from the `manifest.xml` file), commit and push the changes. If we use `http` instead of `ssh` for pushing we will have to do multiple smaller (less than 4 GB) commits and pushes.

Note: For cloning the internal-stash-repo to Dev-PC the developer might need to update the Git's post-buffer size by running the following command

Following are the details on the time taken and network usage for different steps:

Operation	Time taken	Network Usage
Repo sync (from Google external to Linux-PC	~1.5 hours	Downloads a source checkout of ~117GB after decompression
First time adding all the repos for commit	~10 minutes	N/A
Committing the first time (Linux-PC)	~1 hour	N/A
Pushing the first time to internal-stash-repo	~2.5 hours	This will upload ~16.5 GB of compressed Git objects
Cloning internal-stash-repo to Dev-PC	~45 minutes	This will download ~16.5 GB of compressed Git Objects, Decompressed to a ~68 GB source checkout
Iterative repo sync, commit and push	Depends on new changes made by Google	

Further Suggestions

- We can have the script setup to sync with Google external repo once a week, mostly on weekend.
- The script should be able to send e-mail to evolve2.dev (and other people linked with this) on a success or failure to sync the Google external server to internal-stash-server.
