UnixShellScriptProgrammingGuide

Introduction

With HADOOP-9902, the shell script code base has been refactored, with common functions and utilities put into a shell library (hadoop-functions.sh). Here are some tips and tricks to get the most out of using this functionality:

The Skeleton

All properly built shell scripts contain the following sections:

1. `hadoop_usage` function that contains an alphabetized list of subcommands and their description. This is used when the user directly asks for help, a command line syntax error, etc.
2. `HADOOP_LIBEXEC_DIR` configured. This should be the location of where `hadoop-functions.sh`, `hadoop-config.sh`, etc, are located.
3. `HADOOP_NEW_CONFIG=true`. This tells the rest of the system that the code being executed is aware that it is using the new shell API and it will call the routines it needs to call on its own. If this isn't set, then several default actions that were done in Hadoop 2.x and earlier are executed and several key parts of the functionality are lost.
4. `$HADOOP_LIBEXEC_DIR/abc-config.sh` is executed, where abc is the subproject. HDFS scripts should call `hdfs-config.sh`. MAPRED scripts should call `mapred-config.sh` YARN scripts should call `yarn-config.sh`. Everything else should call `hadoop-config.sh`. This does a lot of standard initialization, processes standard options, etc. This is also what provides override capabilities for subproject specific environment variables. For example, the system will normally ignore `yarn-env.sh`, but `yarn-config.sh` will activate those settings.
5. At this point, this is where the majority of your code goes. Programs should process the rest of the arguments and doing whatever their script is supposed to do.
6. Before executing a Java program (preferably via hadoop_java_exec) or giving user output, call `hadoop_finalize`. This finishes up the configuration details: adds the user class path, fixes up any missing Java properties, configures library paths, etc.
7. Either an `exit` or an `exec`. This should return 0 for success and 1 or higher for failure.

Adding a Subcommand to an Existing Script (NOT hadoop-tools-based)

In order to add a new subcommand, there are two things that need to be done:

1. Add a line to that script's `hadoop_usage` function that lists the name of the subcommand and what it does. This should be alphabetized.
2. Add an additional entry in the case conditional. Depending upon what is being added, several things may need to be done:
    a. Set the `HADOOP_CLASSNAME` to the Java method. b. Add $HADOOP_CLIENT_OPTS to $HADOOP_OPTS (or, for YARN apps, $YARN_CLIENT_OPTS to $YARN_OPTS) if this is an interactive application or for some other reason should have the user client settings applied. c. For methods that can also be daemons, set `HADOOP_SUBCMD_SUPPORTDAEMONIZATION=true`. This will allow for the `--daemon` option to work. See more below. d. If it supports security, set `HADOOP_SUBCMD_SECURESERVICE=true` and `HADOOP_SUBCMD_SECUREUSER` equal to the user that should run the daemon.
3. If a new subcommand needs one or more extra environment variables:
    a. Add documentation and a **commented** out example that shows the default setting. b. Add the default(s) to that subprojects' hadoop_subproject_init or hadoop_basic_init for common, using the current shell vars as a guide. (Specifically, it should allow overriding!)

Adding a Subcommand to an Existing Script (hadoop-tools-based)

As of HADOOP-12930, subcommands that come from hadoop-tools utilizing the Dynamic Subcommands functionality. This allows for end-users to replace /override these utilities with their own versions as well as prevent the classpath from exploding with extra dependencies.

1. Create a src/main/shellprofile.d directory
2. Inside there, create a hadoop-name.sh file that contains the bash functions necessary to create a Dynamic Subcommand. Note that versions that ship with hadoop need to verify that the function doesn't already exist. (See, for example, hadoop-archives/src/main/shellprofile.d)
3. Modify the hadoop-tools assembly to copy this shellprofile.d in the correct place.
4. To get hadoop_add_to_classpath_tools functionality to work for your command, add the following to your pom.xml. This uses the Maven dependency plug-in to create a file that the build system will use to create the file needed by that function.

```
      <plugin>
       <groupId>org.apache.maven.plugins</groupId>
       <artifactId>maven-dependency-plugin</artifactId>
       <executions>
         <execution>
           <id>deplist</id>
           <phase>compile</phase>
           <goals>
             <goal>list</goal>
           </goals>
           <configuration>
             <!-- referenced by a built-in command -->
             <outputFile>${project.basedir}/target/hadoop-tools-deps/${project.artifactId}.tools-builtin.txt<
/outputFile>
           </configuration>
         </execution>
       </executions>
     </plugin>
```

Better Practices

- Avoid adding more globals or project specific globals and/or entries in *-env.sh and/or a comment at the bottom here. In a lot of cases, there is pre-existing functionality that already does what you might need to do. Additionally, every configuration option makes it that much harder for end users. If you do need to add a new global variable for additional functionality, start it with HADOOP_ for common, HDFS_ for HDFS, YARN_ for YARN, and MAPRED_ for MapReduce. It should be documented in either *-env.sh (for user overridable parts) or hadoop-functions.sh (for internal-only globals). This helps prevents our variables from clobbering other people.
- Remember that abc_xyz_OPTS can and should act as a catch-all for Java daemon options. Custom heap environment variables and other custom daemon variables add unnecessary complexity for both the user and us. They should be avoided. In almost every case, it is better to have a global and apply it to all daemons to have a universal default. Users can/will override that variables as necessary in their init scripts. This also helps cover the case when functionality starts in one chunk of Hadoop but ends up in multiple places.
- If you absolutely need to have a per-daemon environment variable, try to generalize the solution and use the `HADOOP_${command}_yourthing`-type of layout. See the hadoop_verify_user function for an example.
- Avoid mutli-level `if`'s where the comparisons are static strings. Use case statements instead, as they are easier to read.
- BSDisms, GNUisms, or SysVisms. Double check your command, parameters, and/or reading of OS files on multiple operating systems. (Usually a quick Google search will pull up man pages for other OSes.) In particular, check out Linux, OS X, FreeBSD, Solaris, and AIX. It's reasonable to expect code to work for approximately three-five years. Also take note of hadoop_os_tricks, where OS-specific start up stuff can go, so long as a user would never want to change it...
- Output to the screen, especially for daemons, should be avoided. No one wants to see a multitude of messages during startup. Errors should go to STDERR instead of STDOUT. Use the `hadoop_error` function to make it clear in the code.
- This isn't Java. CamelCase is weird and you will be endlessly mocked if you use it for variable names. Instead, all capital letters for globals, lowercase for locals.
- If you need to add a new function, it should start with hadoop_ and declare any local variables with the `declare` or `local` tag. This allows for 3rd parties to use our function libraries without worry around conflicts.
- The Bash Hackers website and Google have great general advice for style guidlines in bash. Additionally, Paul Lutus's Beautify Bash does a tremendously good job reformatting bash.
- A decent shell lint is available at http://www.shellcheck.net . Mac users can `brew install shellcheck` to install it locally. Like lint, however, be aware that it will sometimes flag things that are legitimate. These can be marked using a 'shellcheck disable' comment. (Usually, the flag for $HADOOP_OPTS being called without quotes is our biggest offense that shellcheck flags. Our usage without quotes is correct for the current code base. It is, however, a bad practice and shellcheck is correct for telling us about it.)

Standard Environment Variables

In addition to all of the variables documented in `*-env.sh` and `hadoop-layout.sh`, there are a handful of special env vars:

- `HADOOP_HEAP_MAX` - This is the Xmx parameter to be passed to Java. (e.g., `"-Xmx1g"`). This is present for backward compatibility, however it should be added to `HADOOP_OPTS` via `hadoop_add_param HADOOP_OPTS Xmx "${JAVA_HEAP_MAX}"` prior to calling `hadoop_finalize`.
- `HADOOP_DAEMON_MODE` - This will be set to `start` or `stop` based upon what `hadoop-config.sh` has determined from the command line options.
- `HADOOP_LOGFILE` - This sets hadoop.log.file, which in turn will go to log4j. In many cases, you'll want to modify this based upon the daemon.
- `HADOOP_SECURE_USER` - This should be set to the user that should be running your secure daemon. This **MUST** be set prior to calling the `hadoop_secure_*` functions.
- `HADOOP_SLAVES` - This is the file name the user passed via `--hosts`.
- `HADOOP_SLAVE_NAMES` - This is the list of hosts the user passed via `--hostnames`.

About Daemonization

In branch-2 and previous, "daemons" were handled via wrapping "standard" command lines. If we concentrate on the functionality (vs. the code rot...) this has some interesting (and inconsistent) results, especially around logging and pid files. If you run the `*-daemon` version, you got a pid file and `hadoop.root.logger` is set to be `INFO,(something)`. When a daemon is run in non-daemon mode (e.g., straight up: `hdfs namenode`), no pid file is generated and `hadoop.root.logger` is kept as `INFO,console`. With no pid file generated, it is possible to run, e.g. hdfs namenode, both in *-daemon.sh mode and in straight up mode again. It also means that one needs to pull apart the process list to safely determine the status of the daemon since pid files aren't always created. This made building custom init scripts fraught with danger. This inconsistency has been a point of frustration for many operations teams.

Post-HADOOP-9902, there is a slight change in the above functionality and one of the key reasons why this is an incompatible change. Sub-commands that were intended to run as daemons (either fully, e.g., namenode or partially, e.g. balancer) have all of this handling consolidated, helping to eliminate code rot as well as providing a consistent user experience across projects. daemon=true, which is a per-script local, but is consistent across the hadoop sub-projects, tells the latter parts of the shell code that this sub-command needs to have some extra-handling enabled beyond the normal commands. In particular, `supportdaemonization=true` sub-commands will always get pid and out files. They will prevent two being run on the same machine by the same user simultaneously (see footnote 1, however). They get some extra options on the java command line. Etc, etc.

So where does `--daemon` come in? The value of that is stored in a global called `HADOOP_DAEMON_MODE`. If the user doesn't set it specifically, it defaults to 'default'. This was done to allow the code to mostly replicate the behavior of branch-2 and previous when the `*-daemon.sh` code was NOT used. In other words, `--daemon` default (or no value provided), let's commands like `hdfs namenode` still run in the foreground, just now with pid and out files. `--daemon start` does the `disown` (previously a `nohup`), change the logging output from `HADOOP_ROOT_LOGGER` to `HADOOP_DAEMON_ROOT_LOGGER`, add some extra command line options, etc, etc similar to the `*-daemon.sh` commands.

What happens if daemon mode is set for all commands? The big thing is the pid and out file creation and the checks around it. A user would only ever be able to execute one `hadoop fs` command at a time because of the pid file! Less than ideal.

To summarize, `supportdaemonization=true` tells the code that `--daemon` actually means something to the sub-command. Otherwise, `--daemon` is ignored.

1-... unless `HADOOP_IDENT_STRING` is modified appropriately. This means that post-HADOOP-9902, it is now possible to run two secure datanodes on the same machine as the same user, since all of the logs, pids, and outs, take that into consideration! QA folks should be very happy.

A New Subproject or Subproject-like Structure

The following files should be the basis of the new bits:

- libexec/(project)-config.sh

This contains the new, common configuration/startup bits. At a minimum, it should contain the bootstrap stanza for hadoop-config.sh and a function called hadoop_subproject_init that does the actual, extra work that needs to be done. Variables should be HADOOP_(project)*(whatever) and should be initialized based off of the standard HADOOP** variables.

- bin/(project) or sbin/(project)

User-facing commands, those should be in bin. Administrator commands should be in sbin. See the skeleton example up above on how to build these types of scripts.

- conf/*-env.sh

Ideally, this should follow the pattern already established by the other *-env.sh files. Entries in it can also be in hadoop-env.sh to provide for a consistent and much easier operational experience.