VISCOSITY
NORTH AMERICA

On the Tenth Day of 18c/19c, Viscosity Gave to me...

AutoUpgrade Utility

December 21, 2020

The Oracle 19c database is replete with new capabilities and enhanced features; but, customers can only take advantage of these improvements by upgrading. Historically, upgrades and migrations were painful, sometimes lengthy efforts that few database administrators (or organizations) looked forward to. Database upgrades are infrequent events for most environments and each time steps that worked for prior versions, are unlikely to apply to the current database. Source and target versions combine to dictate a potentially lengthy and confusing series of actions for checking compatibility, verifying prerequisites, disabling deprecated and desupported features, and updating configurations.

Fortunately, among the improvements introduced alongside Oracle 18c and 19c, a tool that greatly streamlines and simplifies the process of upgrading Oracle databases: AutoUpgrade!

Why AutoUpgrade?

As AutoUpgrade is a new tool, database administrators may be reluctant to entertain it, instead deferring to more familiar methods they've used in the past. One objection to AutoUpgrade is that it's too easy. There's an expectation that upgrades must be difficult, and AutoUpgrade's simple setup and interface makes it easy to assume that it's not powerful or comprehensive enough for any except the most simple of environments. In fact, AutoUpgrade boasts an impressive set of checks and capabilities behind its unassuming interface.

Among these features:

- A robust set of pre-checks and reporting that identify (and in many cases, automatically correct) issues discovered in the environment. These range from checking for adequate resources and proper database settings, to identifying deprecated and desupported features in the existing database.
- An interactive command line interface that allows users to manage the full upgrade process, view status and progress, pause and resume upgrades, and even restore the database to its original state.
- AutoUpgrade is resumable. If it encounters a problem, AutoUpgrade reports the error and pauses the process. Once the problem is fixed, administrators restart the upgrade and the intelligent process picks up where it left off. If the issue can't be resolved, users can roll back the upgrade within the tool. In either case, there's no need to identify the operation that failed or build any special commands. AutoUpgrade tracks its progress and runs what is needed.

Technology Overview

Database: 18c / 19c

Company: Oracle Corporation

Topic: AutoUpgrade Utility

Viscosity can help with any of your Database Upgrade needs

Viscosity has performed numerous zero-downtime database migrations and upgrades over the years and has a proven track record with business critical and mission critical databases.

Viscosity's Database Migration & Upgrade Services can plan, upgrade, validate and migrate all database content - quickly and effectively with our automated approach and proven methodology.

Learn more about how you can maintain and maximize your investments at viscosityna.com or email us at hello@viscosityna.com.

www.viscosityna.com

- Extensive logging and feedback, with meaningful messages and status updates. AutoUpgrade compares very favorably to Database Upgrade Assistant in this category, particularly during more lengthy steps. Side-by-side, DBUA may appear to be hung on these steps, but AutoUpgrade continues to update its status, assuring users that it's still actively working.
- Upgrades Oracle databases from version 11.2.0.4 to target versions of 12.2.0.1, 18.5 and 19.3 or newer.

When running an upgrade, AutoUpgrade performs many operations DBAs might normally run manually. This includes creating a Guaranteed Restore Point, collecting statistics on fixed objects, compiling invalid objects, emptying the Recycle Bin, removing hidden (underscore) parameters, and updating the database time zone. Of course, users still have the ability to fine tune which steps AutoUpgrade runs, allowing it to be tailored to suit nearly any environment.

Invoking AutoUpgrade: Analyze, Fix, and Upgrade Databases

Invoke AutoUpgrade by referencing a configuration file and a mode:

```
$JAVA_PATH/java -jar $AUTOUPGRADE_PATH/autoupgrade.jar -config
$CONFIG_PATH/$CONFIG_FILE -mode [analyze|fixups|upgrade|deploy]
```

We'll cover the configuration file in a moment. First, let's discuss the four modes.

- Analyze: performs a read-only analysis and produces a list of passed and failed checks, identifies issues, and recommends fixes the user can act on.
- Fixups: runs the same analysis as analyze, but additionally fixes all of the exceptions that can be automatically performed. All other issues are reported for manual intervention.
- Upgrade: used when AutoUpgrade is run on a server different than the one hosting the source database. In this case, AutoUpgrade can't read certain components of the source database's Oracle Home directory and therefore can't automatically identify or resolve certain issues. It runs the upgrade and post-upgrade checks, but it doesn't create a Guaranteed Restore Point (GRP) or run any post upgrade fixes. Upgrade is typically used in conjunction with running fixups on the source, as part of a migration strategy.
- Deploy: runs all of the following as part of a complete upgrade:
    - setup
    - pre-upgrade checks
    - creates a Guaranteed Restore Point
    - runs pre-upgrade fixes
    - drains load from the database
    - upgrades the database
    - post-upgrade checks
    - post-upgrade fixes

The AutoUpgrade Configuration File

The flexibility and utility of AutoUpgrade is driven through its configuration file. These files can range from minimal settings, for upgrading a single database with defaults, to detailed instructions for

controlling (and even scheduling) multiple databases across an enterprise. While a single configuration file can manage complex, conditional upgrades for dozens or even hundreds of databases, the syntax is easy to read and understand.

Configuration files are built from name-value pairs that AutoUpgrade reads at run time. Each entry consists of a prefix, a parameter, and a value. For example:

```
global.autoupg_log_dir=/opt/oracle/autoupgrade
db1.target_home=/opt/oracle/product/19c/dbhome_1
```

Here, global and db1 are prefixes, and autoupg_log_dir and target_home are parameters separated from their values by the equal sign.

A Simple Configuration

Putting this to practice, a minimal configuration for AutoUpgrade might look like this:

```
# Global parameters
global.autoupg_log_dir=/home/oracle/autoupgrade

# Database parameters
db1.sid=<DB_NAME>
db1.run_utlrp=yes
db1.start_time=now
db1.timezone_upg=yes
db1.upgrade_node=<MY_DB_HOST>
db1.target_version=19.8
db1.target_home=/u01/app/oracle/product/19c/dbhome_1
db1.source_home=/u01/app/oracle/product/12.1.0.2/dbhome_1
```

AutoUpgrade can generate a sample configuration template to build from using the -create_sample_file flag:

```
$JAVA_PATH/java -jar $AUTOUPGRADE_PATH/autoupgrade.jar -create_sample_file
$CONFIG_PATH/$CONFIG_FILE
```

Prefix, Global, and Local Settings

Every entry in the configuration file includes a global prefix (to denote parameters applied to all operations) or a user-defined prefix that groups settings for individual instances. Global settings are used to set values that apply to multiple upgrades, and every configuration file will include at least one global parameter, the AutoUpgrade log directory. A more realistic example of a global configuration might include the target home path and version:

```
global.autoupg_log_dir=/home/oracle/autoupgrade
global.target_home=/u01/app/oracle/product/19c/dbhome_1
```

```
global.target_version=19.8
```

Local settings are prefixed by a user-defined value and links parameters to a specific upgrade; in the example above, db1. Local settings include the host and database SID. AutoUpgrade matches local parameters to a database by reading values for host, SID, and source and/or target database home. A single configuration file can independently manage upgrades for multiple databases in a single home, multiple homes, and multiple hosts.

Some parameters may be set both globally and locally. In this case, the local setting for a database overrides the global value in the configuration. This is useful when an environment has many databases that mostly follow a convention, with a few outliers. The conventional values can be set as a global parameter and the exceptions managed through a local setting:

```
# Global parameters
global.autoupg_log_dir=/home/oracle/autoupgrade
global.target_home=/u01/app/oracle/product/19c/dbhome_1
global.target_version=19.8

# Database 1
db1.upgrade_node=node1
...
# Database 10
db10.upgrade_node=node10
db10.target_home=/u01/app/oracle/product/19c/dbhome_2
```

Control Database Configuration Settings

AutoUpgrade's detailed control of upgrades extends beyond just managing operations and directories. It can add and remove database parameters during and after the upgrade:

```
db1.add_during_upgrade_pfile=/home/oracle/initdb1.add.during.ora
db1.del_during_upgrade_pfile=/home/oracle/initdb1.del.during.ora
db1.add_after_upgrade_pfile=/home/oracle/initdb1.add.after.ora
db1.del_after_upgrade_pfile=/home/oracle/initdb1.del.after.ora
```

The files for adding parameters are formatted like a regular database pfile, with each parameter=value pair to be added on a separate line. Those for deleting parameters, is a list of parameters to be deleted, one per line, without an equal sign or value.

AutoUpgrade can also remove hidden (underscore) parameters:

```
global.remove_underscore_parameters=yes
```

This is a global parameter only without a local counterpart. For environments where most hidden parameters need to be deleted, this option can be combined with add_after_upgrade_pfile to control the environment.

Run User-Defined Scripts

Some environments may have requirements to call custom scripts before or after the upgrade. AutoUpgrade can call these scripts as part of the process. This localizes control of the entire upgrade process within AutoUpgrade, and users can leverage the interface and reporting features to include these custom needs. In Linux these are shell scripts with a .sh suffix. In Windows, they are batch scripts with either a .cmd or .bat suffix, or PowerShell scripts with a .ps1 suffix:

```
global.before_action=/home/oracle/pre_upgrade.ALL.sh
db1.before_action=/home/oracle/pre_upgrade.DB1.sh
db1.after_action=/home/oracle/post_upgrade.DB1.sh
```

Managing Jobs with the AutoUpgrade Console

By default, running AutoUpgrade takes the user to an interactive upg> console prompt:

```
$ java -jar autoupgrade.jar -config config.txt -mode analyze
AutoUpgrade tool launched with default options
Processing config file ...
+--------------------------------+
| Starting AutoUpgrade execution |
+--------------------------------+
1 databases will be analyzed
Type 'help' to list console commands
upg>
```

This may catch some first-time users off guard, as the prompt isn't an intuitive or expected part of an automatic utility. When the AutoUpgrade job completes it will report its status and return to the shell:

```
upg> Job 100 completed
------------------- Final Summary --------------------
Number of databases            [ 1 ]

Jobs finished successfully     [1]
Jobs failed                    [0]
Jobs pending                   [0]
------------- JOBS FINISHED SUCCESSFULLY -------------
Job 100 for VNA
$
```

Shorter operation line analyze won't normally have much need for interaction, but upgrade and deploy will take time. The console affords users the opportunity to monitor and manage the progress of these jobs. The full set of commands is displayed by running help at the prompt:

```
exit                          // To close and exit
help                          // Displays help
lsj [(-r|-f|-p|-e) | -n <number>]  // list jobs by status up to n elements.
       -f Filter by finished jobs.
```

```
       -r Filter by running jobs.
       -e Filter by jobs with errors.
       -p Filter by jobs being prepared.
       -n <number> Display up to n jobs.
lsr                          // Displays the restoration queue
lsa                          // Displays the abort queue
tasks                        // Displays the tasks running
clear                        // Clears the terminal
resume -job <number>         // Restarts a previous job that was running
status [-job <number> [-long]]// Lists all the jobs or a specific job
restore -job <number>        // Restores the database to its state
                                prior to the upgrade
restore all_failed           // Restores all failed jobs to their
                                previous states prior to the upgrade
logs                         // Displays all the log locations
abort -job <number>          // Aborts the specified job
h[ist]                       // Displays the command line history
/[<number>]                  // Executes the command specified from the
                                history. The default is the last command
```

Helpful commands include lsj, logs, and status.

- lsj lists jobs and a summary of their status.
- Logs identifies the location of log files for active jobs.
- Status offers detailed information on the current state of a job, including the current stage and how long various steps have taken.

Jobs are managed by the abort, resume, and restore commands. Abort may be misleading; aborting a job is often thought of as an unrecoverable or non-resumable operation, but in AutoUpgrade it's a mechanism for stopping or pausing a job. After aborting a job, users can resume the upgrade, or restore the database to its pre-upgrade state.
If a job fails for any reason, users have the option of fixing the problem and resuming the job, or restoring the database.

AutoUpgrade can be called with the -noconsole option to suppress the console for silent, background operation. No special environment configurations or settings are necessary to run AutoUpgrade.

Reporting and Visibility

Upgrades capture the interest of the enterprise, as stakeholders and database administrators have competing interests. DBAs want to focus on the technical aspects of the upgrade, without interruptions from people asking for status, while the rest of the organization wants regular progress updates. AutoUpgrade includes a HTML reporting option that offers stakeholders accurate, up-to-date visibility into the upgrade progress.

A few extra steps are necessary to enable this, but they're well worth the effort. Start a simple web server on the upgrade host:

```
cd $ORACLE_BASE/autoupgrade/cfgtoollogs/upgrade/auto
```

```
python -m SimpleHTTPServer 8080
AutoUpgrade updates its state.html page every few seconds and users can view
it by navigating to the host/port:
http://upgradehost:8080/state.html
```

This may seem like a trivial thing, but this level of openness builds trust and confidence in the process by offering end users access to the same communication and information available to administrators.

Obtaining AutoUpgrade

AutoUpgrade is included in the Oracle version 19.3 database software download, and updated versions are part of each Release Update. The latest version of AutoUpgrade is always available from My Oracle Support under Note 2485457.1.

AutoUpgrade doesn't require any special licensing, and it can be used for Standard and Enterprise Edition databases. There are no plugins or agents to install. It's a roughly 4-megabyte JAR file that requires Java 8 or later, satisfied by the target Oracle home, if it's not already part of the environment.

AutoUpgrade Tips and Tricks

Below are some valuable, real-world, recommendations from our experience using AutoUpgrade in particular and upgrading to Oracle 18c and 19c specifically:

- Run AutoUpgrade in analyze mode and review the results before anything else. Then, manually address as many issues as possible weeks, days, and hours ahead of the upgrade.
- Check DB_RECOVERY_FILE_DEST_SIZE and DB_RECOVERY_FILE_DEST. An inadequate setting for the recovery destination size is probably the number one cause of a failed upgrade. Make sure this is set to the minimum value recommended by AutoUpgrade, during the analyze process. This is not something AutoUpgrade can or will fix during the fixup stage, since it can impact space use on the operating system.
- Manually run ultrp.sql and set run_utlrp to no. utlrp.sql as it is time consuming to run on most systems. Running it manually prior to the upgrade, removes the operation from the outage and minimizes the length of the upgrade.
- Collect dictionary and fixed object status ahead of time. Like utlrp.sql, these can take time to run and moving them before the upgrade reduces down time.
- Re-run analyze leading up to the upgrade to insure that no new issues are detected.
- Practice upgrading in a lab environment. Upgrades and migrations are infrequent events for most organizations. The tools and processes will be unfamiliar at best. There is value in practicing the process, particularly practicing failure. Troubleshooting issues in critical systems is easier if users are already comfortable navigating and reading diagnostics produced by tools.
- Increase AWR retention to 30, 60, or 90 days or more, and generate baselines for important activities, so that pre- and post-upgrade performance is quantified and recorded for comparison.
- Remove SQLPATH, glogin.sql, login.sql from environments prior to running AutoUpgrade. The login.sql and glogin.sql files can include settings that interfere with or break scripts during an upgrade.

- Check for obsolete features, including APEX and Streams, and remove any unused features and demo schemas.
- Backup wallets and convert Access Control Lists (ACL) to Access Control Entries (ACE). There are some situations where ACL entries are interpreted differently in 19c databases, leading to unexpected behavior.
- Take this opportunity to review and update internal documentation.
- Validate backup and recovery procedures. A backup alone does not guarantee recovery! Test the process and clearly document the steps.

Summary

AutoUpgrade is the preferred method for upgrading Oracle databases and advances the process for DBAs. It's a robust, powerful utility hiding behind an unassuming interface that makes the once difficult, sometimes painful, task of upgrading databases easier and more reliable.

One of the sayings we have at Viscosity is our customer's, "have four aces in their pocket". Over the next 2 days, the talented staff at Viscosity along with our Oracle ACEs will address more Oracle Database 18c and 19c new features. Continue to join us next year, as we continue our Oracle Database 19c hands-on-lab workshops.

Happy Holidays!