Slony-I User Tutorial

Outline / Table of Contents

This is a working outline which contains reminders of what sections should cover. Each element in the outline has a deliverable marked by ==>.

This outline is off the top of my head with the current information I am using to set up clients. Obviously there is more to it. Comments are welcome.

- Non-Keyed Tables (if you *must*)
- ==> 3 table lists ready to paste into slony_tools.conf

Setting up Perltools

- Edit slony_tools.conf
- Generate anticipated scripts
- ==> Tool Kit of prepared scripts

Preparing replica databases

- dropdb, createdb
- Add users if necessary
- load from pg_dump -s of master
- ==> Replica ready for replication

Slony User Tutorial: Prepare Perl Tools and Scripts
Overview of perltools and slony_tools.conf

- Edit slony_tools.conf with input from Prerequisites and Definition of DB cluster
- Generate all scripts anticipated (and some which are not)
- ==> slonik scripts for all expected tasks for your cluster

Initializing Slony

- Executing Slonik Scripts: Read the scripts before you run them.
- Initialize, start slons on each machine, create set, subscribe set
- ON ERROR: oopsie; what to do
- Discuss slow copy issues and workarounds
- Viewing Log files
- ==> Slony cluster up and replicating

Removing a node for maintenance

- Replica
- ==> Slony cluster up as it was and replicating

Promote Replica to be Master

- Do stuff
- Switch apps to point to new master

- ==> Former Replica is now master and former master is now replica

Master Failover

- human decision, scripted action
- promotion of replica to master
- Switch apps to point to new master
- Fix the problem and
- drop fromer master node
- Prepare former master database as replica
- add master back and subscribe
- reset master as master

Replica Failure

- Drop replica node
- Re-prepare replica database
- Subscribe
- ==> Slony cluster up and replicating as it was

Running DDL

- Maintaing DDL changes over time
- ==> Execution of DDL on all nodes in sync

Adding a Table to Replication

- Maintaining slony_tools.conf dilemma
- create set
- merge set or not
- ==> New table defined and replicated

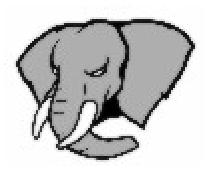
Slony-I Version Upgrades

- ==> upgraded slony on each machine in slony cluster.

Slony User Tutorial Introduction

Introducing Slony-I

Slony is the Russian plural for elephant. It is also the name of the replication project developed initially by Jan Weick. The mascot for Slony, Slon, is a good variation of the usual Postgres elephant mascot, created by Jan.
Slon, the Slony mascot.



Slony-I, the first iteration of the project, is an asynchronous replicator of a single master database to multiple replicas, which in turn may have *cascaded* replicas. It will include all features required to replicate large databases with a reasonable number of replicas. Jan has targeted Slony-I toward data centers and backup sites, implying that all nodes in the network are always available.

The master is the primary database with which the applications interact. Replicas are replications, or copies of the primary database. Since the master database is always changing, data replication is the system that enables the updates of secondary, or replica, databases as the master database updates. In synchronous replication systems, the master and the replica are consistent exact copies. The client does not receive a commit until all replicas have the transaction in question. Asynchronous replication loosens that binding and allows the replica to copy transactions from the master, rolling forward, at its own pace. The server issues a commit to the master client based on the state of the master database transaction.

Cascading replicas over a WAN minimizes bandwidth, enabling better scalability and also enables read-only (for example, reporting) applications to take advantage of replicas.

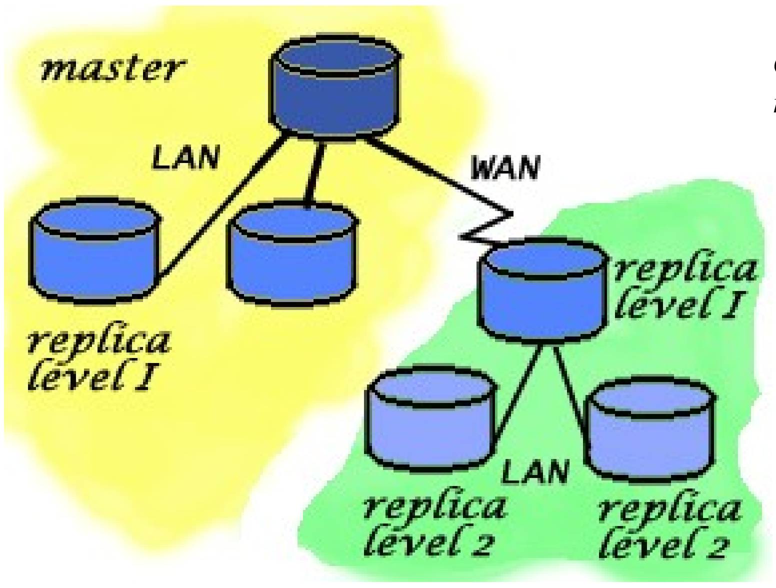


Figure 2. Cascading replicas

Assume you have a primary site, with a database server and a replica as backup server. Then you create a remote backup center with its own main server and its backup replica. The remote primary server is a direct replica, replicating from the master over the WAN, while the remote secondary server is a cascaded replica, replicating from the primary server via the LAN. This avoids transferring all of the transactions twice over the WAN. More importantly, this configuration enables you to have a remote backup with its own local failover already in place for cases such as a data center failure.

Slonys design goals differentiate it from other replication systems. The initial plan was to enable a few very important key features as a basis for implementing these design goals. An underlying theme to the design is to update only that which changes, enabling scalable replication for a reliable failover strategy.
The design goals for Slony are:

- The ability to install, configure, and create a replica and let it join and catch up with a running database.

  This allows the replacement of both masters and replicas. This idea also enables cascading replicas, which in turn adds scalability, limitation of bandwidth, and proper handling of failover situations.

- Allowing any node to take over for any other node that fails.

  In the case of a failure of a replica that provides data to other replicas, the other replicas can continue to replicate from another replica or directly from the master.

- Hot PostgreSQL installation and configuration.
  For failover, it must be possible to put a new master into place and reconfigure

the system to allow the reassignment of any replica to the master or to cascade from another replica. All of this must be possible without taking down the system.
This means that it must be possible to add and synchronize a new replica without disrupting the master. When the new replica is in place, the master switch can happen.
This is particularly useful when the new replica is a different PostgreSQL version than the previous one. If you create an 8.0 replica from your 7.4 master, it now is possible to promote the 8.0 to master as a hot upgrade to the new version.

- Schema changes.
  Schema changes require special consideration. The bundling of the replication transactions must be able to join all of the pertinent schema changes together, whether or not they took place in the same transaction. Identifying these change sets is very difficult.
  In order to address this issue, Slony-I has a way to execute SQL scripts in a controlled fashion. This means that it is even more important to bundle and save your schema changes in scripts. Tracking your schema changes in scripts is a key DBA procedure for keeping your system in order and your database recreatable.

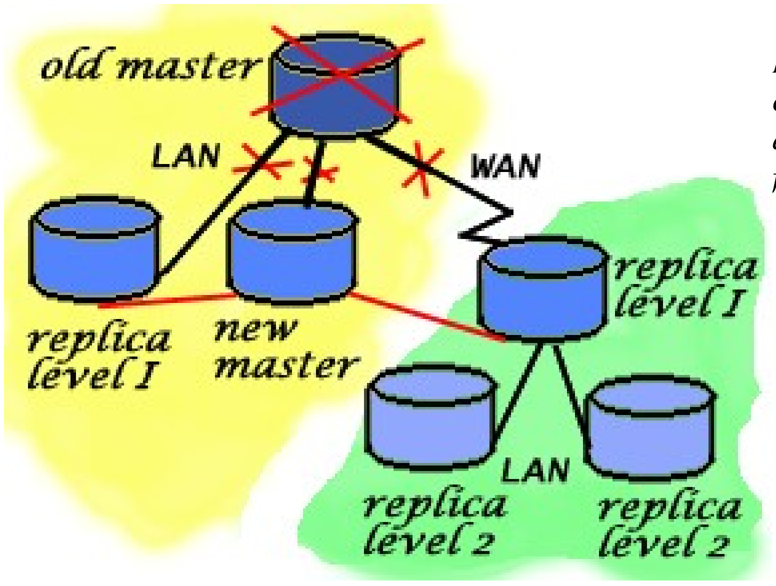


Figure 3. Replication continues after a failure

In the case where a master node fails, a replica can receive a promotion to become a master. Any other replicas can then replicate from the new master. Because Slony-I is asynchronous, the different replicas may be ahead of or behind each other. When a replica becomes a master, it synchronizes itself with the state of the most recent other replica.

In other replication solutions, this roll forward of the new master is not possible. In those solutions, when promoting a replica to master, any other replicas that exist must rebuild from scratch in order to synchronize with the new master correctly. A failover of a 1TB database leaves the new master with no failover of its own for quite a while. The Slony design handles the case where multiple replicas may be at different synchronization times with the master and are able to resynchronize when a new master arises. For example, different replicas could logically be in the future, compared to the new master. There is a way to detect and correct this. If there were not, you would have to dump and restore the other replicas from the new master to synchronize again.

Its possible to roll forward the new master, if necessary, from other replicas because of the packaging and saving of the replication transactions. Replication data is packaged into blocks of transactions and sent to each replica. Each replica knows what blocks it has consumed. Each replica can also pass those blocks along to other servers--this is the mechanism of cascading replicas. A new master may be on transaction block 17 relative to the old master, when another replica is on transaction block 20 relative to the old master. Switching to the new master causes the other replicas to send blocks 18, 19, and 20 to the new master.
Jan, said, "This feature took me a while to develop, even in theory."

The first part of Slony-I also does not address any of the user interface features required to set up and configure the system. After the core engine of Slony-I becomes available, development of the configuration and maintenance interface can begin. There may be multiple interfaces available, depending on who develops the user interface and how.

Jan points out that "replication will never be something where you type SETUP and all of a sudden your existing enterprise system will nicely replicate in a disaster recovery scenario." Designing how to set up your replication is a complex problem. The user interface(s) will be important to clarify and simplify the configuration and maintenance of your replication system. Some of the issues to address include the configuration of which tables to replicate, the requirement of primary keys, and the handling of sequence and trigger coordination.

The Slony-I release does not address the issues of multi-master, synchronous replication or sporadically synchronizable nodes (the "sales person on the road" scenario). However, Jan is considering these issues in the architecture of the system so that future Slony releases may implement some of them. It is critical to design future features into the system; analysis of existing replication systems has shown that it is next to impossible to add fundamental features to an existing replication system.

The primary question to ask regarding the requirements for a failover system is how much down time can you afford. Is five minutes acceptable? Is one hour? Must the failover be read/write, or is it acceptable to have a read-only temporary failover? The second question you must ask is whether you are willing to invest in the hardware required to support multiple copies of your database. A clear cost/benefit analysis is necessary, especially for large databases.

Slony User Tutorial Build and Install

You must build slony on each machine participating in Slony replication. There will be a slon daemon running for each replica on each machine so the program must be on each machine.

If you have trouble copying installations from machine to machine, just build and install it on each machine separately. This will be the case when the hardware is different between machines.

It is assumed that there is at least one PostgreSQL installation on each machine participating in replication.

Download

Download Slony from http://gborg.postgresql.org/project/slony1/projdisplay.php It may also be available in your favorite flavor of RPM, however, it is simple and clean to build from source.

You can configure Slony-I to be *built* anywhere on your machine. The configuration directives will tell it where to *install* it. For example, use /usr/local/src/ as a base to untar downloaded file.

Configure

In order to avoid some older installation bugs, it is preferable to be explicit in exactly where you want Slony to be installed. Slony is an add-on to PostgreSQL and can be installed within the PostgreSQL installation tree or separately.

You may want to install Slony in /usr/local/Slony or something similar. If you do this, you must ensure that the new locaion is on the default PATH for your Postgres environment. Alternatively, you can install Slony along side of PostgreSQL.

In the following example of configuration arguments, slony is being installed along with an instance of Postgres. Slony will use Postgres's bin, include, lib, package and share directories. This configuration helps ensure your PATH, which should be set to include your postgres bin will automatically be set to access your slony programs. The PGINSTALLATION is just a place holder for your actual PostgreSQL installation directory, for example /usr/local/pgsql.

The **include**, **share** and **lib** files should be your PostgreSQL's directories. The slony executables, *slon* and *slonik* should be in the same **bin** as *psql*. The perltool scripts will be in a separate place if you specify perltools or in the same **bin** as *slon* and *slonik*. The pgconfigdir should point to the bin directory where pg_config is located.

```
--prefix=PGINSTALLATION
--with-pgbindir=PGINSTALLATION/bin
--with-pgconfigdir=PGINSTALLATION/bin
--with-pgincludedir=PGINSTALLATION/include
--with-pgincludeserverdir=PGINSTALLATION/include/server
--with-pglibdir=PGINSTALLATION/lib
--with-pgpkglibdir=PGINSTALLATION/lib
--with-pgsharedir=PGINSTALLATION/share
--with-perltools=PGINSTALLATION/bin
```

Reconfigure these elements to match your own installation locations. This can be tricky and then you may have to move slony files into the proper places by hand until you get it correct. For example, **slony_funcs.so**, **xxid.so** and **slon_tools.pm** should all be in the $PGLIB directory along with **plpgsql.so**.

Including the perltools is required for most of the remainder of the tutorial so do not exclude them.

Build and Install

Change to the directory where you untar'd Slony and build with the usual anthem:

```
make
sudo make install
```

There should be no known build errors. So if you encounter a build error, send email to the Slony mailing list or ask on #slony on freenode.net.

Preparing directories for using Slony PerlTools

A workspace is necessary to use the perltools. You can put it anywhere. An etc and a Script directory will be required. The /etc directory will be created by the install where ever you installed Slony. The Scripts file should be its sibling you will need to create that by hand.

```
PGINSTALLATION/slony/etc
PGINSTALLATION/slony/Scripts
```

The etc directory has a template file called slon_tools.conf-sample. This is an important file. Copy the slon_tools.conf-sample to slon_tools.conf. We will collect information in the next several sections in order to fill out this configuration file.

Prepare your log directory

Create the directory where you want Slony to log. That directory must be writable by the postgres super user (usually postgres). The most common place for slony to store its log files is /var/log/slony/. Under that directory, once slony is running you will see a subdirectory called slony1 and a subdirectory for each node of the cluster on that machine. The actual log files will be in the node directories. *Always remember where your log files are.*

Directory Summary (from recommendations)

- **source**: /usr/local/src/Slony<version>
- **installation**: PGINSTALLATION, e.g. /usr/local/pgsql
- **logfile**: /var/log/pgsql/Slony1/nodexx
- **slony etc**: PGINSTALLATION/etc/
- **slony scripts**: PGINSTALLATION/Scripts/
- **PG Library:** PGLIB: PGINSTALLATION/lib/
- **PG Include:** PGINSTALLATION/include
- **PG Include Server:** PGINSTALLATION/include/server

Slony User Tutorial Define Clusters

A Slony node is a unique (database instance + a postgresql installation). Each node will have a slon process associated with it.

Choose the Master Node and Replica Nodes

The master will be the node that is read and write. Specify it by a number, say 10 as well as the machine name, postgresql instance and database name. Choose one or more replica nodes, denoting them in the same way. The replicas will be readonly. Slony nodes are numbered 10, 20, 30, or some other scheme that suits you. 1,2,3,etc. are not used due to an historical special meaning of node 1.

Usually the master feeds the other nodes. But replicas can also feed replicas. The latter is useful when the two replicas are across a WAN from the master.
Decide which nodes feed data to other nodes. For example in a 3 node setup, the Master might feed data to both replicas 20 and 30 or the first replica, 20, might send its data to the third replica, 30. When in doubt, have the master feed all replicas.

Draw a picture

No matter how simple it seems to you now. And put it on your wall. You will use this information to configure the slony perltools and to administer your cluster.

Make sure you you note your database names and node numbers.

Below are two simple ascii diagrams. The first diagram below shows a master feeding two replicas and the second shows a master feeding one replica which in turn feeds the second replica. Circles and lines are even better.

```
M(10)+--> R(20)
     |
     +--> R(30)

M(10) --> R(20) --> R(30)
```

Network Connectivity

Ensure that there is bidirectional network connectivity between each node.
There must be network connectivity from every machine participating in replication to every other machine participating in replication. The exception to this rule is only for slony networks across a WAN where you have configured a primary replica which

feeds other replicas. Connectivity must exists for every machine talking possible. You can and should test this before you begin to avoid any surprises. This can be done by invoking psql *from each node* to each of its target hosts. We're using a default database of template1 because ours are not yet set up.

```
master: psql -h repl20 -c "select 1;" template1
master: psql -h repl30 -c "select 1;" template1
repl20: psql -h master -c "select 1;" template1
repl20: psql -h repl30 -c "select 1;" template1
repl30: psql -h repl20 -c "select 1;" template1
repl30: psql -h master -c "select 1;" template1 # not needed for M-
>R->R
```

Defining Nodes in slon_tools.conf

The following are entries in the slon_tools.conf file. You should be able to read and understand the relatively simple perl structures. The comments are good. Look at your cluster diagram now. (You still have your diagram, don't you?)

```
$CLUSTER_NAME = 'replication';
```

- Decide on a name for your cluster. The name should follow the rules for other identifiers. You can use, for example, the name of the master database. The name of your cluster, preceded by an underscore will be the name of the schema in your data base where all slony functionality is stored.

```
$LOGDIR = '/var/log/slony';
```

- If your log directory is something other that /var/log/slony then change this value to reflect the proper name.

```
$MASTERNODE = 1;
```

- Change this to the proper node number for the master node.

```
add_node(node     => 1,
         host     => 'server1',
         dbname   => 'database',
         port     => 5432,
         user     => 'postgres',
         password => '');
```

Modify the add_node code *for each node in the cluster*, including the master node. Set the proper node number. Set the proper host, database and port information. Set the user and password information. It is not recommended that you include the passwords here. It is better to control access to postgres via the pg_hba.conf file.

If you are implementing a cluster like M(10)-> R(20) -> R(30) where R(20) has no access to M(10), then you will want to add one more attribute to the add_node section for R(30): parent = 20 where 20 is the node number of R(20). The down side of this is if R(20) goes down, then R(30) is stranded.

At this point, only the table lists remain to be specified in the slon_tools.conf.

Slony User Tutorial Table Preparation

Slony Requirements

Slony requires that each table participating in replication must have either a primary key or a unique index. If you have tables that do not have primary keys or unique indexes you must fix those cases or use an ill-advised workaround that is not covered here.

Viewing and Listing Tables

The goal is to create three lists:

1. tables with primary keys
2. tables with unique indexes (and no primary keys)
3. sequences

The slony_list_tables.sql (Appendix A) file contains four select statements. The first shows you all of your tables and their primary keys and indexes to give you an overview of the general situation. The last three will output, respectively, a list of primary keyed tables, a list of uniquely indexed tables with the index name, and finally a list of sequences. The output format of these last three scripts should allow you to cut and paste the results into slon_tools.conf at the proper time.

Use these select statements to evaluate your primary key/unique index situation. Double check that the list of primary keyed tables does not overlap with the list of uniquely indexed tables. Choose the primary key over the unique index. Eliminate tables that are not participating in replication. *Each table participating in replication must be in one and only one of these lists.*

The end result will be the three lists: primary keyed tables, uniquely indexed tables and sequences. Save these lists. They will be needed to cut and paste into the slon_tools.conf file.

NOTE: These scripts work on 8.1 and later. They do not run on 7.4 and were not tested against 8.0. In the case of 7.4, use \d for table lists and extract the primary keys and unique values from the description of each table.

Fixing Tables without Primary Keys

After you fix any tables without primary keys, be sure to regenerate the lists from above, including any edits you made to make the lists distinct. The lists may have changed to add the corrected table(s).

If you need to add a primary key, first you should try to choose a unique key for the table. You must determine what column or columns in the bad table actually define a unique row. Then you can alter the table to add the primary key on that column or columns.

```
        ALTER TABLE foo ADD PRIMARY KEY (column(s));
```

If you cannot find a set of unique columns, you must use alter the table to add a serial column *which will be a PRIMARY KEY* to the table.

```
        ALTER TABLE foo add column pkey SERIAL;
        ALTER TABLE foo ADD PRIMARY KEY (pkey);
```

Seek help on the pgsql-general@postgresql.org mailing list or IRC at postgresql on FreeNode.net if you are having trouble fixing your tables to have primary keys. This is a general problem that many people can help you with.

Updating Lists in slon_tools.conf

The set name can remain as set1 and the set_id can remain as 1. In the future you may want to create set2 in the slon_tools.conf file, but that will not be covered right here and now. See [Adding Sets.]

The table_id and sequence_id can also remain the same. These ids assigned to tables and sequences will be important later when adding second sets.

In the following section the primary keyed table list you generated previously will be substituted for the 'table1','table2',. Keep the values quoted and comma separated between the square brackets. Monkey see monkey do.

```
        “pkeyedtables” = [
           'table1',
           'table2',
        ];
```

In the next section you will include the list of tables and unique index names for those tables with unique indexes and without primary keys defined. The left side should be the table name; the right is the index name. Values should be quoted and entries should be comma separated.

```
        "keyedtables" => {
             'table3' => 'index_on_table3',
             'table4' => 'index_on_table4',
        },
```

In the last section you will ignore the serialtables (ill-advised route) and fill out the names of sequences from the list you created earlier.

Setting Up PerlTools

The perltools scripts are located in the directory designated by the **–with-perltools=PGINSTALLATION/bin** option on the configuration of Slony.

The *scripts generate scripts* to do what is described below each one:

`slonik_build_env`
generates slon_tools.conf; unused in this tutorial

`slonik_create_set [--config file] set`
creates set specified based on slon_tools.conf

`slonik_drop_node [--config file] node#`
drops node specified

`slonik_drop_set [--config file] set#`
drops set sepecified based on slon_tools.conf

`slonik_execute_script set# [--node=node#] -c SCRIPT`
execute ddl SCRIPT via slony, optionally run on specified node for specified set

`slonik_failover dead_node backup_node`
Abandons dead_node, making backup_node the origin for all sets on dead_node. move_set should be used if dead_node is still available, so that transactions are not lost.

`slonik_init_cluster`
Generates the slonik commands necessary to create a cluster and prepare the nodes for use.

`slonik_merge_sets [--config file] node# set# set#`
Merges the contents of the second set into the first one. The node specified is the origin of the two sets.

`slonik_move_set [--config file] set# from_node# to_node#`
Change a set's origin.

`slonik_restart_node [--config file] [--all] [node# ...]`
Restart slon on node specified. Documented to be obsolete.

`slonik_store_node [--config file] node#`
Generates the slonik commands necessary to add a node to a cluster.

`slonik_subscribe_set [--config file] set# node#`
Begins replicating a set *to* the specified node.

`slonik_uninstall_nodes [--config file]`

Removes Slony configuration from all nodes in a cluster.

`slonik_unsubscribe_set  [--config file] set# node#`
Stops replicating a set on the specified node.

`slonik_update_nodes [--config file]`
Updates the functions on all nodes.

These slony perltools read your **slon_tools.conf** file and the appropriate parameters and generate *slonik* scripts.

The perltools do not act on their own. They just generate scripts. For example to generate the init cluster slonik script use:

```
$ slonik_init_cluster > init_cluster.slnk
```

The resulting slonik scripts are run with slonik, e.g.

```
$ slonik init_cluster.slnk;
```

Generating Slonik Scripts

We have a prototype script **create_all_slonik.sh** as shown in **APPENDIX C** to be put into the **Scripts** directory. This script runs most of the perltools above to give you a toolkit of scripts which you can use as is or as prototypes for executing slonik commands to intialize and manage your slony cluster.

create_all_slonik.sh creates scripts necessary for a three node cluster. You can and should modify this script for different cluster configurations. If you use node names other than 10, 20 and 30 or sets other than 1, 2... then you want to reflect the accurate names in the parameter lists for the various functions. Note also that we set a variable SLONCONF. Make sure this variable points to the **etc** directory where your **slon_tools.conf** lives.

Note that the execute script function is only a prototype. You will need to modify a copy of this script to do the actual execution because at this time the DDL script file is not known.

Use **create_all_slonik.sh** to generate your tool kit **on each slony installation**. You will have your toolkit available on each machine. This is important so that if a machine really crashes you can still control the replication for failover.
These scripts can be run from any node, but are usually issued by custom from the master node. The next sections will describe how you should use these tools.

Preparing Replica Databases

Each replica in your cluster must start out as an empty database with the same schema as your master database.

Drop the database if necessary; recreate it and populated it with a schema dump of the master database. In this example we are naming the replica databases replicadb. and are calling the master database master10. You may want to use the same database name for all databases in your cluster. master_host is the host name of the master database. Since you already set up and tested your network connectivity the pg_dump piped to psql should work fine. We are using the user *postgres* for all of the slony operations.

On each replica machine at the shell:

```
$ dropdb -U postgres replicadb; -- if necessary
$ createdb -U postgres replicadb;
$ pg_dump -U postgres -s -h master_host master10
| psql -U postgres replicaxx
```

These commands give you a clean, empty replica database populated directly by the schema dump from the master database.

At this time you will also want to add users to the replica databases if necessary. If your applications are connecting to the replicas (in read only mode) with a user name other than your database super user.

If at anytime you need to restart the entire replication process on a database, you will need to have an empty database, populated with the schema to replicate to. For example, if too many errors cause you to throw up your hands, restart from here.

Initializing Slony

- Executing Slonik Scripts: Read the scripts before you run them.
- Initialize, start slons on each machine, create set, subscribe set
- ON ERROR: oopsie; what to do
- Discuss slow copy issues and workarounds
- Viewing Log files
- ==> Slony cluster up and replicating

Before you run any of the generated slonik scripts you must read them and understand what they will do. See the documentation of the slonik commands if necessary.

To initialize the slonik cluster, we must have in place the master database node and the replica node(s). The replica nodes must be prepared—they must be empty and contain the exact schema of the master database. The slonik tool kit must have been prepared and created using create_all_scripts.slnk. These examples will deal with a master node 10 replicated to both replica node 20 and replica node 30.

Now we will initialize slony, start the slon daemons and subscribe the replica nodes to the master's table set 1.

On the master node:

```
slonik slonik_init_cluster
```

The init_cluster will create a schema in each participating database named _CLUSTER where CLUSTER is what you named your replication cluster in you slon_tools.conf file.

On each machine, for each node that you are running slony, you will need to start the slon daemon. This means logging into each machine and issuing the command for each node, including the master and the replicas. Where node# is the node number of the node you are starting:

```
slon_start –config $SLONYCONF –watchdog –sleep
60 node#
```

Back on the master node, subscribe set 1 to nodes 20 and 30. Set 1 originates on node 10.

```
subscribe_set_1_20.slnk

subscribe_set_1_30.slnk
```

Now you should have slony running. You can check your replica databases to see if they now have data in them. Sometimes the subscribe is slow, so be patient. It is also very often blindingly fast.

To monitor the progress of slony, including the initial copying of the sets review each logfile for each machine. Remember the log file was set up in your slon_tools.conf and is usually something like /var/log/pgsql/Slony1/nodexx where xx is your node number. Each node has its own log file.

You will also want to look at the view _CLUSTER.sl_status.

Appendix A: Slony List Tables SQL

Notes:

These scripts work on 8.1 and later. They do not run on 7.4 and were not tested against 8.0. In the case of 7.4, use \d for table lists and extract the primary keys and unique values from the description of each table.

```
------------------------------------------------
-- View your primary keys and unique indexes. --
------------------------------------------------
SELECT n.nspname || '.' || c.relname AS table_name,
   con.conname AS constraint_name,
   con.contype AS contype,
   (SELECT ARRAY(SELECT aa.attname
    FROM pg_attribute aa
    JOIN generate_series
         (1,current_setting('max_index_keys')::int,1) s(i)
    ON (aa.attnum = conkey[i])
    WHERE aa.attrelid=a.attrelid ORDER BY i)
   ) AS columns
FROM pg_constraint con
JOIN pg_namespace n ON n.oid = con.connamespace
JOIN pg_class c ON c.oid = con.conrelid
JOIN pg_attribute a ON (a.attrelid = c.oid)
WHERE con.conrelid<>0 AND con.contype in ('p','u')
          AND c.relkind = 'r' and a.attnum >0
          AND a.attnum = ANY (con.conkey)
          AND nspname<>'pg_catalog'
ORDER BY con.contype;

-------------------------------
-- List Primary Keyed Tables --
-------------------------------
SELECT '''' || n.nspname || '.' || c.relname || ''',' as table_name
FROM pg_constraint con
JOIN pg_namespace n ON n.oid = con.connamespace
JOIN pg_class c ON c.oid = con.conrelid
WHERE con.conrelid<>0 AND con.contype in ('p')
          AND c.relkind = 'r'
          AND nspname<>'pg_catalog'
ORDER BY con.contype;

----------------------------------------
-- List Tables and Unique Index Names --
----------------------------------------
SELECT '''' || n.nspname || '.' || c.relname || ''' => '''
|| con.conname || ''',' as table_idx_name
FROM pg_constraint con
```

```
JOIN pg_namespace n ON n.oid = con.connamespace
JOIN pg_class c ON c.oid = con.conrelid
JOIN pg_attribute a ON (a.attrelid = c.oid)
WHERE con.conrelid<>0 AND con.contype in ('p','u')
          AND c.relkind = 'r' and a.attnum >0
          AND a.attnum = ANY (con.conkey)
          AND nspname<>'pg_catalog'
ORDER BY con.contype;

--------------------
-- List Sequences --
--------------------
SELECT '''' || n.nspname || '.' || c.relname || ''',' as table_name
FROM
pg_namespace n
JOIN pg_class c ON n.oid = c.relnamespace
WHERE
c.relkind = 'S' AND nspname<>'pg_catalog';
```

Appendix B: Resources

Slony-I References/Help

Web:
    http://gborg.postgresql.org/project/slony1/projdisplay.php

IRC:
    #slony on freenode.net

Email:
    slony1-general@gborg.postgresql.org

Documentation:
    http://gborg.postgresql.org/project/slony1/genpage.php?howto_idx

PostgreSQL Help

Email:
    pgsql-general@postgresql.org

IRC:
    #postgresql on freenode.net

Web:
    www.postgresql.org

Documentation:
    www.postgresql.org/docs/

Appendix C: create_all_slonik.sh

```
# Create slonik scripts based on slon_tools.conf in SLONYCONF
# Nodes are 10, 20, 30 where 10-->20 AND 10 --> 30
# Edit this script to suit your needs.
export SLONYCONF=../etc/slon_tools.conf

# Setting up Slony
slonik_init_cluster   --config $SLONYCONF > init_cluster.slnk
slonik_subscribe_set  --config $SLONYCONF 1 10 > subscribe_set_1_10.slnk
slonik_subscribe_set  --config $SLONYCONF 1 20 > subscribe_set_1_20.slnk
slonik_subscribe_set  --config $SLONYCONF 1 30 > subscribe_set_1_30.slnk

# Modify DDL
slonik_execute_script  --config $SLONYCONF -C xxx 1 > execute_script_xxx.slnk

# Adding/Deleting/Unistalling Nodes
slonik_store_node      --config $SLONYCONF 10 > store_node10.slnk
slonik_store_node      --config $SLONYCONF 20 > store_node20.slnk
slonik_store_node      --config $SLONYCONF 30 > store_node30.slnk
slonik_drop_node       --config $SLONYCONF 10 > drop_node10.slnk
slonik_drop_node       --config $SLONYCONF 20 > drop_node20.slnk
slonik_drop_node       --config $SLONYCONF 30 > drop_node30.slnk
slonik_uninstall_nodes --config $SLONYCONF > uninstall_nodes.slnk

# Creating Subscribing and Manipulating Sets
slonik_create_set      --config $SLONYCONF 1 > create_set_1.slnk
slonik_subscribe_set   --config $SLONYCONF 1 10 > subscribe_set_1_10.slnk
slonik_subscribe_set   --config $SLONYCONF 1 20 > subscribe_set_1_20.slnk
slonik_subscribe_set   --config $SLONYCONF 1 30 > subscribe_set_1_30.slnk
slonik_unsubscribe_set --config $SLONYCONF 1 20 > unsubscribe_set_1_20.slnk
slonik_unsubscribe_set --config $SLONYCONF 1 30 > unsubscribe_set_1_30.slnk
slonik_unsubscribe_set --config $SLONYCONF 1 10 > unsubscribe_set_1_10.slnk
slonik_merge_sets      --config $SLONYCONF 10 1 2 > merge_sets_10_1_2.slnk
slonik_move_set        --config $SLONYCONF 1 10 20 > move_set_1_10_20.slnk
slonik_move_set        --config $SLONYCONF 1 20 10 > move_set_1_20_10.slnk
slonik_drop_set        --config $SLONYCONF 1 > drop_set.slnk

# Failover
slonik_failover        --config $SLONYCONF 10 20 > failover_10_20.slnk
slonik_failover        --config $SLONYCONF 20 30 > failover_20_30.slnk

# Slony UPdate
slonik_update_nodes    --config $SLONYCONF > update_nodes.slnk
```