Setting up PostgreSQL

1 Introduction to PostgreSQL

PostgreSQL is an object-relational database management system based on POSTGRES, which was developed at the University of California at Berkeley. PostgreSQL is an open-source descendant of this original Berkeley code.

2 Installation of PostgreSQL

This section describes how to download, install and use PostgreSQL version 8.1.4. This is the version that will be used for all assignments in this course.

A compressed tar archive containing the PostgreSQL version 8.1.4 source code can be found in the CS448 course account as `/u/cs448/public/postgresql-8.1.4.tgz`. This same archive can be downloaded via the Web though the link that can be found on the CS448 course web page. For a more complete installation guide, refer to the INSTALL file, which is included with the PostgreSQL source code.

You can install PostgreSQL on your own machine, or in your account in the CS student computing environment.

2.1 Installing PostgreSQL in the CS Student Computing Environment

First, log in to a Linux server in the student.cs computing environment. The simplest way to do this is to log in to `linux.student.cs.uwaterloo.ca`. This is a load balancing alias which will automatically choose a specific Linux server to log you into. Alternatively, you can choose a specific Linux server and log in directly to that server. A list of available Linux servers in the student computing environment can be found at `http://www.cs.uwaterloo.ca/cscf/student/hosts.shtml`. Make sure you choose a **Linux** server.

Next, from your home directory, extract a copy of the PostgreSQL source from the course account:

```
tar xzf /u/cs448/public/postgresql-8.1.4.tgz
```

This may take a few minutes. It should create a new directory called `postgresql-8.1.4` containing a complete copy of the PostgreSQL source code.

PostgreSQL is large. You will need about 70MB of space to hold the unpacked source code. By the time you have built and installed PostgreSQL , you will be consuming about 200MB of space. If you are registered for CS448/648, you should have plenty of disk quota to allow you to work with PostgreSQL , provided that you did not filled your quota with unrelated stuff! You can check your disk quota and disk usage using the `diskquota` command.

In your home directory, do

```
mkdir pgbuild
```

This creates a `pgbuild` directory in which the PostgreSQL binaries and libraries will be placed once they have been built. Next, `cd` into the `postgresql-8.1.4` directory, which was created when you unpacked the PostgreSQL distribution, and configure PostgreSQL like this:

```
./configure --prefix=$HOME/pgbuild CFLAGS='-g -O0' --enable-debug --enable-cassert --with-maxbackends=3
```

The two configuration arguments, `--enable-debug` and `--enable-cassert`, are used to enable debugging of PostgreSQL code and assertions, respectively. The CFLAGS setting also simplifies debugging by turning off compiler optimizations. Note that in the `-O0` in the CFLAGS setting, the first character is an upper-case letter `O`, and the second is the digit `0` (zero).

Then, build PostgreSQL by running

```
make
```

Again, this make take a few minutes. You may see warnings during the build, but it should complete successfully.

Next, install PostgreSQL into your pgbuild directory by running:

```
make install
```

At this point, PostgreSQL has been built and installed in your `$HOME/pgbuild` directory. Before running PostgreSQL commands, you will need to set your `PATH` and `LD_LIBRARY_PATH` environment variables so that the PostgreSQL binaries and libraries can be found. `csh` and `tcsh` users should do this in their `.cshrc` file. Look for a line like this:

```
setenv PATH '/bin/showpath $HOME/bin standard'
```

and add `$HOME/pgbuild/bin` to the list, similar to this:

```
setenv PATH '/bin/showpath $HOME/bin $HOME/pgbuild/bin standard'
```

Assuming `LD_LIBRARY_PATH` is not already being defined somewhere in the file, you should also define that variable using a line like this:

```
setenv LD_LIBRARY_PATH $HOME/pgbuild/lib
```

Users of `sh` or `bash` should make similar changes in their `.profile` or `.bashrc` file. The syntax is slightly different, but should be self-explanatory. You may need to log out and log back in again to get these environment variable settings to take effect.

The next step is to create a directory in which to host the database and related server state:

```
initdb --locale=C -D $HOME/pgdb
```

This will initialize the database in a `pgdb` directory under your home directory. If you wish, you can choose a different directory name. Don't forget the `--locale` option; it is important for Assignment 1.

Last but not least, you should ensure that your copy of the PostgreSQL code is not visible to others. In your home directory, issue the following command:

```
chmod 700 postgresql-8.1.4
```

You should now be able to start the database server. The most convenient way to run PostgreSQL is to use two separate shell command windows. In one window, you will launch the server. In the other window, you will run programs that issue commands to the server. To use this method, you must be logged in to the **same Linux machine** in both windows. You can check which machine you are logged into in a particular window by running the `hostname` command.

Once you have two command windows on the same machine, launch the PostgreSQL server in one window using the following command:

```
postmaster -p <port-number> -D $HOME/pgdb
```

Note that because you are running the PostgreSQL server on a shared host, you have to use a port number other than the default one to avoid conflicts with other instances of PostgreSQL initiated by other students. This is the purpose of the `-p` flag. Legitimate port numbers are greater than 1024 and less than or equal to 65536. To minimize the likelihood of conflicts, choose a random port number with 5 digits (i.e., greater than 10000).

In the other window, you can now run PostgreSQL client programs and utilities that issue commands to the PostgreSQL server that you just launched. You tell these client progams how to find the server by specifying, as a command line parameter to the client, the port number that is being used by your server.

For example, you can use the `createdb` utility to create a new database. To create a new database named `mytest`, you would use the command

```
createdb -p <port-number> mytest
```

Here, the port number that you specify must match the one with which you launched your server.

One client that you will need to use is `psql`, which gives you a simple, text-based command interface that you can use to issue SQL commands to the database server, and view the results of those commands. If you created a test database called `mytest`, you can launch `psql` on your test database like this:

```
psql -p <port-number> mytest
```

You can now use the `psql` client to interactively create tables, insert data, and issue queries. A sample script that creates two tables and performs a number of queries can be found at
`postgresql-8.1.4/src/tutorial/basics.source`

To quit the interactive client, use the `psql` command $\backslash q$. `psql` also has on-line help available, which you can access using the \? `psql` command. Further documentation for `psql`, as well as all of the other PostgreSQL clients, can be found in Section VI-II of the on-line PostgreSQL documentation, which is linked to by the course web page.

When you are finished running PostgreSQL client programs, please **shut down your PostgreSQL server**, so that unused servers do not clog up the student.cs Linux machines.

You can shut down your PostgreSQL server by simply typing control-C in the server's window. The server should output some log information indicating that it is shutting down, like this

```
 ^CLOG:  received fast shutdown request
LOG:  shutting down
LOG:  database system is shut down
```

That's all that's necessary. You can double-check that you have no PostgreSQL servers running by using the `ps f` command, which can show all of the processes that you have running on a machine. This should give you a list of processes in a format that illustrates their parent-child relationships. The output of the `ps f` command may vary a bit from machine to machine, but it should look something like this:

```
@linux032[125]% ps f
  PID TTY      STAT   TIME COMMAND
56096 pts/5    Ss     0:00 -tcsh
62695 pts/5    R+     0:00  _ ps f
50503 pts/3    Ss     0:00 -tcsh
50541 pts/3    S+     0:00  _ postmaster -p 34543 -D /u5/kmsalem/pgdb
50546 pts/3    S+     0:00      _ postmaster -p 34543 -D /u5/kmsalem/pgdb
50547 pts/3    S+     0:00      _ postmaster -p 34543 -D /u5/kmsalem/pgdb
50548 pts/3    S+     0:00          _ postmaster -p 34543 -D /u5/kmsalem/pgdb
32475 pts/2    Ss+    0:00 -tcsh
```

or perhaps like this:

```
zonker2: ps f
  PID TTY      STAT   TIME COMMAND
18002 pts/4    Ss     0:00 bash
18031 pts/4    R+     0:00  _ ps f
17998 pts/3    Ss     0:00 bash
18017 pts/3    S+     0:00  _ psql -p 34543 mytest
17994 pts/2    Ss     0:00 bash
18010 pts/2    S+     0:00  _ postmaster -p 34543 -D ./pgdb
18012 pts/2    S+     0:00      _ postgres: writer process
18013 pts/2    S+     0:00      _ postgres: stats buffer process
18014 pts/2    S+     0:00      |   _ postgres: stats collector process
18018 pts/2    S+     0:00      _ postgres: kmsalem mytest [local] idle
```

If you see any `postmaster` or `postgres` processes (as in the examples), you have a PostgreSQL server running. You can kill any such servers using the `kill` command, specifying the process ID (PID) of the parent `postmaster` process. In the first example above, the PID of the parent `postmaster` is 50541, so you can kill it with:

```
kill -INT 50541
```

In the second example, the parent `postmaster` has PID 18010. Once you have issued the `kill` command, you can use `ps f` again to make sure that PostgreSQL is really dead.

2.2 Installing PostgreSQL on Your Own Linux Machine

You should be able to install CS448's version of PostgreSQL on your own Linux machine by following the instructions for installation in the student.cs computing environment. You will not be able to extract the PostgreSQL code directly from the course account. Instead, you can download a compressed tar archive of the PostgreSQL source code using the link on the course web page, and then unpack your copy of the archive using `tar`:

```
tar xzf postgresql-8.1.4.tgz
```

This will create the `postgresql-8.1.4` directory, in which you can configure and build PostgreSQL .

If PostgreSQL fails to configure or make, it may because you are missing software packages required by PostgreSQL . One common problem is missing `readline` packages. On Ubuntu, make sure you have both `libreadline` and `libreadline-dev` installed. Other package-based Linux distributions should have similar packages. Support for zlib may also be missing. On Ubuntu, make sure that `zlib1g-dev` is installed if the PostgreSQL configuration complains about lack of zlib support.

3 Modifying PostgreSQL Source Code

Course assignments will require that you add or modify PostgreSQL source files. Before modifying PostgreSQL files, make sure that you have a backup copy of the original file so that you can always undo your modifications. Note that after making changes to PostgreSQL files, you should clean the built version using `make clean` before rebuilding from the modified source code. This is particularly important if you have modified header files.

Before each assignment, you should start with a fresh copy of the PostgreSQL source code. Each assignment is standalone, i.e. the assignments are not incremental.

4 Debugging PostgreSQL

PostgreSQL is a client/server system, meaning that a user runs a client process, like the `psql` command interpreter, which talks to a PostgreSQL server process. The main (parent) PostgreSQL server, called `postmaster`, spawns a separate server process (also called `postmaster`) for each client connection.

There are two main methods that can be used for debugging. The first method is to print out debugging information (e.g. variables' values) from within the server process. The second method is to use a debugging facility to insert breakpoints at interesting locations and inspect the variables' values and the flow of control.

4.1 Printing Server Debugging Information

To insert debugging statements into PostgreSQL server code, use the `elog()` function, with the first argument being `DEBUG1`. Note that `elog()` takes a message string as its main argument; to construct such a string you may want to use the `sprintf()` routine. You'll find examples of the use of `elog()` in the `nbtinsert.c` file, which you will be working on for Assignment 1.

To get `elog()` messages to be displayed by the server, you should use the `-d 1` flag when you launch the `postmaster`, e.g.,

```
postmaster -d 1 -p <port-number> -D $HOME/pgdb
```

The server will display your debug messages in its log, which is normally sent to the server process's stderr output.

4.2 Using a Debugger

You may use any available debugger, such as gdb, to debug PostgreSQL server code. To start debugging a PostgreSQL server process on a local machine, you first need to startup the server (i.e., `postmaster`), and the client (i.e., `psql`). Then, you must attach the debugger to the PostgreSQL server process that is serving your `psql` client. To do this using gdb, open another shell window **on the same host on which your PostgreSQL server is running**. This should leave you with three separate windows: one for the server, one for the client program (e.g., `psql`), and one for the debugger. In the debugger window, enter the command:

```
ps f
```

This will give you a list of your PostgreSQL server processes. Assuming you have launched the PostgreSQL server and one `psql` client, your process list should look something like this.

```
@linux032[126]% ps f
  PID TTY      STAT   TIME COMMAND
56096 pts/5    Ss     0:00 -tcsh
62802 pts/5    R+     0:00  _ ps f
50503 pts/3    Ss     0:00 -tcsh
50541 pts/3    S+     0:00  _ postmaster -p 34543 -D /u5/kmsalem/pgdb
50546 pts/3    S+     0:00      _ postmaster -p 34543 -D /u5/kmsalem/pgdb
50547 pts/3    S+     0:00      _ postmaster -p 34543 -D /u5/kmsalem/pgdb
50548 pts/3    S+     0:00      |   _ postmaster -p 34543 -D /u5/kmsalem/pgdb
62783 pts/3    S+     0:00      _ postmaster -p 34543 -D /u5/kmsalem/pgdb
32475 pts/2    Ss     0:00 -tcsh
62782 pts/2    S+     0:00  _ psql -p 34543 A1test
```

In this example, the PostgreSQL server process that is serving the `psql` client is the `postmaster` process with PID 62783. You can tell which `postmaster` process is the one serving your `psql` client in several ways:

- Use the `ps f` command before and after starting the `psql` command and compare the output. A new `postmaster` process will have been started to serve the `psql` client.
- Look for the `postmaster` process with a PID that is larger than the PID of the `psql` client.
- Usually, the correct `postmaster` process will be the last one in the list.

The new PostgreSQL server process that is serving your `psql` client is the process that you will need to connect the debugger to. In this example, its process ID (PID) is 62783.

Once you have identified the correct process id, launch the debugger:

```
gdb postgres
```

At the gdb command prompt, enter

```
attach <process-id>
```

where `<process-id>` is the PostgreSQL server process id that you just identified: 62783 in the example above. Attaching `gdb` to the PostgreSQL server process will cause the server process to pause, so that you can use the debugger to inspect code and variables, set breakpoints, and so on. Issue `gdb`'s `continue` command when you are ready to let the server process continue running. If you wish to exit `gdb` without killing the PostgreSQL server process, you can issue a `detach` command to `gdb`.

5 Documentation

The main source for PostgreSQL information is the official documentation, to which there is a link from the course web page. In the source code, you will find `README` files within each component directory (e.g. parser, executor and optimizer components). Comments found in the PostgreSQL code are particularly helpful in understanding how PostgreSQL functions are implemented.