COMP9315 19T2 DBMS Implementation

Prac Exercise 01

Setting up your PostgreSQL Server

Aims

This exercise aims to get you to:

- set up your directories on the `/srvr` file system
- install a PostgreSQL database server on `/srvr`

You ought to get it done before the end of Week 2.

Background

Notation: In the examples below, we have used the $ sign to represent the prompt from the Unix/Linux shell. The actual prompt may look quite different on your computer (e.g. it may contain the computer's hostname, or your username, or the current directory name). In the example interactions, all of the things that the computer displays are in `this font`. The commands that **you** are supposed to type are in **`this bold font`**. Comments in the examples are introduced by '...' and are written in `this grey font`; comments do not appear on the computer screen, they simply aim to explain what is happening. Whenever we use the word ***`edit`***, this means that you should use your favourite text editor (e.g. `vi`, `emacs`, `gedit`, etc.) Finally, all references to *YOU* should be replaced by your CSE username, (e.g. I would use `jas` everywhere that *YOU* is used).

PostgreSQL has three major components:

- the source code (and the compiled `*.o` files) (approx 150MB)
- the installed executables (like `pg_ctl` and `psql`) (approx 20MB)
- the data (including configuration files and databases) (at least 35MB)

You will not be able to fit the above components under your CSE *home* directory (insufficient disk quota), so what we have arranged is for you to have an additional directory (folder) called `/srvr/`*YOU* with enough space to hold all of the above. You can access this directory via the command:

```
$ cd /srvr/YOU
```

You *must* put your PostgreSQL source code and installed executables under `/srvr/`*YOU*. The data can be located either under `/srvr/`*YOU* or under the `/tmp` filesystem on the machine where you're working. You can edit, compile and execute PostgreSQL on any workstation within CSE. You can also use the server `grieg`; you *must not* use any of the other general-purpose servers (such as `wagner`, `weill`, etc.) for running PostgreSQL servers. You will need to configure things slightly differently depending on where you run PostgreSQL; how to do this is described below.

If you're doing all of this work on a laptop or PC at home, then you can configure things however you like. You will still need folders for the same three components (source code, executables, and data), but you can place them wherever you like. PostgreSQL doesn't require any special privileges to run (at least on Unix-based systems like Linux and Mac OS X), so you do *not* need to create a special privileged PostgreSQL user; you can run the server as yourself.

Setting Up your /srvr Directory (optional)

You should have a directory on `/srvr` already. If not, the way to create one is to run the following commands from any CSE workstation:

```
$ ssh grieg
... you are now logged into the computer called "grieg"
$ priv srvr
... create the directory /srvr/YOU
$ exit
... you are now logged off the computer called "grieg"
```

You should only need to do this once. Once your `/srvr/`*YOU* directory exists, repeating the above achieves nothing.

If you have a `/srvr/`*YOU* directory from a previous database course, you might want to clean out any `pgsql` directory before you continue with the steps below.

Setting up your PostgreSQL Server

Reminder: all of the commands related to compiling and running and using your PostgreSQL server run fastest on the computer called `grieg`. The times below are approximate; they could double or triple depending on which machine you use.

Quick summary (for experts only):

Non-experts should go straight to the detailed instructions below.

```
$ cd /srvr/YOU
$ tar xfj /web/cs9315/19T2/postgresql/postgresql-11.3.tar.bz2
... creates and populates a directory called postgresql-11.3 ...
$ cd postgresql-11.3
$ ./configure --prefix=/srvr/YOU/pgsql
... produces lots of output ...
$ edit src/backend/storage/ipc/latch.c
... and fix an annoying Grieg glitch ...
... search for "epoll_create1" ...
... on the line above "#if defined(WAIT_USE_EPOLL)" ...
... add "#undef EPOLL_CLOEXEC" ...
```

```
$ make
... produces lots of output; takes approx 3-5 minutes ...
$ make install
... produces lots of output ...
$ cp /web/cs9315/19T2/postgresql/env /srvr/YOU/env
$ source /srvr/YOU/env
$ which initdb
/srvr/YOU/pgsql/bin/initdb
$ initdb
... produces some output; takes approx 1 minute ...
$ ls $PGDATA
... gives a listing of newly-created PostgreSQL data directory ...
... including PG_VERSION, base, global ..., postgresql.conf ...
$ edit $PGDATA/postgresql.conf
... set listen_addresses = '' ...
... set max_connections = 8 ...
... set max_wal_senders = 4 ...
... set unix_socket_directories = 'name of PGDATA directory' ...
... if any of the above lines begins with '#', remove the '#'
$ which pg_ctl
/srvr/YOU/pgsql/bin/pg_ctl
$ pg_ctl start -l $PGDATA/log
waiting for server to start.... done
server started
$ psql -l
                          List of databases
   Name    | Owner | Encoding  | Collation | Ctype | Access privileges
-----------+-------+-----------+-----------+-------+-------------------
 postgres  | YOU   | LATIN1    | C         | C     |
 template0 | YOU   | LATIN1    | C         | C     | =c/YOU
                                                   : YOU=CTc/YOU
 template1 | YOU   | LATIN1    | C         | C     | =c/YOU
                                                   : YOU=CTc/YOU
(3 rows)
```

Installation Details (for non-experts):

```
$ pg_ctl stop
waiting for server to shut down.... done
server stopped
```

Note that the above times may be less on a home computer where you are accessing its local disk.

Installation Details (for non-experts):

Setting up directories

The first step is to make sure that the directory `/srvr/YOU` exists. You can check this via the command:

```
$ ls -l /srvr/YOU
```

If the above command says something like "No such file or directory", then you should create it using the instructions above.

Once you have a directory on the `/srvr` filesystem, the next step is to place a copy of the PostgreSQL source code under this directory. The following commands will do this:

```
$ cd /srvr/YOU
$ tar xf /web/cs9315/19T2/postgresql/src.tar.bz2
```

This creates a subdirectory called `postgresql-11.3` under your `/srvr/YOU` directory and unpacks all of the source code there. This produces no output and will take a few moments to complete. If you want to watch as `tar` unpacks the files, use `xvf` instead of `xf` as the first argument to `tar`.

Initial compilation

Once you've unpacked the source code, you should change into the newly created `postgresql-11.3` directory and configure the system so that it uses the directory `/srvr/YOU/pgsql` to hold the executables for your PostgreSQL server. (Note that `/srvr/YOU/pgsql` does not exist yet; it will be created in the `make install` step). The following commands will do the source code configuration:

```
$ cd /srvr/YOU/postgresql-11.3
$ ./configure --prefix=/srvr/YOU/pgsql
```

The `configure` command will print lots of messages about `checking` for various libraries/modules/etc. This process will take a minute, and should produce no errors.

Before you compile PostgreSQL, you need to fix one file which causes PostgreSQL not to work on Grieg. (You can skip this step on other machines). Go to the `postgresql-11.3` directory and edit the file:

```
edit src/backend/storage/ipc/latch.c
```

Search for the string `epoll_create1` and then make the code around there look like:

```
#undef EPOLL_CLOEXEC
#if defined(WAIT_USE_EPOLL)
#ifdef EPOLL_CLOEXEC
    set->epoll_fd = epoll_create1(EPOLL_CLOEXEC);
    if (set->epoll_fd < 0)
        elog(ERROR, "epoll_create1 failed: %m");
#else
```

Now you can continue with the compilation.

After configuring and fixing the source code, the next step is to build all of the programs. Stay in the `postgresql-11.3` directory and then run the command:

```
$ make
```

This compiles *all* of the PostgreSQL source code, and takes around 3-5 minutes (depending on the load on grieg). It will produce *lots* of output, but should compile everything OK, and end with the message:

```
All of PostgreSQL successfully made. Ready to install.
```

Installing executables

Once the PostgreSQL programs are compiled, you need to install them. The following command does this:

```
$ make install
```

This creates the directory `/srvr/YOU/pgsql`, and copies all of the executables (such as `pg_ctl` and `psql`) under that directory. It will take a minute to do this, and will produce quite a bit of output while it's doing it. Ultimately, it should end with the message:

```
PostgreSQL installation complete.
```

Data directories

You're not finished yet, however, since PostgreSQL has no directory in which to store all of its data. There are two possibilities in how to proceed at this stage:

- you could install the data directories under `/srvr/YOU/pgsql`, which has the advantage that you can leave them there permanently, but has the disadvantage that building databases will be slightly slower
- you could install the data directories under `/tmp`, which has the advantage that it's much faster to manipulate databases, but has the disadvantage that you'll: (a) need to re-create the data directories each time you want to use PostgreSQL, and (b) ensure that you stop the server, and remove the data before you log out.

We discuss both possibilities below.

Before doing anything with the database, however, you need to ensure that your Unix environment is set up correctly. We have written a small script called `env` that will do this. In this set up stage, you should copy this script to your `/srvr` directory:

```
$ cp /web/cs9315/19T2/postgresql/env /srvr/YOU
```

The `env` script contains the following:

```
PGHOME=/srvr/$USER/pgsql
export PGDATA=/tmp/pgsql.$USER
export PGDATA=$PGHOME/data
export PGHOST=$PGDATA
export PGPORT=5432
export LD_LIBRARY_PATH=$PGHOME/lib
export PATH=$PGHOME/bin:/home/cs9315/bin:$PATH
```

This script sets up a number of environment variables. The critical ones are:

PGDATA	which tells the PostgreSQL server where it's data directories are located
PGHOST	which tells PostgreSQL clients where are the socket files to connect to the server

Note that there are two definitions for PGDATA. The second one is the default and will use data directories under `/srvr/YOU/pgsql`. If you want to put the data directories under `/tmp` instead, simply swap the two `export PGDATA=...` lines.

What's the difference between the two ways of setting up the data directory? ...

If you use `/tmp` for the data, you will need to create the data directories and edit the PostgreSQL configuration file each time you have a session with PostgreSQL. It is also *essential* that you stop the server and remove the data directories at the end of your session in this case. If you use `/srvr` for the data, it will persist between your login sessions, so you have less setup each time *but* all of your interaction with the database will be slower.

Note that in the discussion below, we will use the string *YOUR_PGDATA* to refer to that value that you assigned to PGDATA in your `env` file and which has been set by `source`'ing the `env` file in your shell.

The precise combination of values in the `env` file depends on where you are running the server. Here are the suggested configurations:

Running server on grieg

You can put the data directories on either `/srvr` or `/tmp`. However, you may need to change the PGPORT value, since the port-space is shared and someone else might already be using port 5432. You will detect this when you try to run the server and it fails to start (check the

`/srvr/YOU/pgsql/log` file if your server will not start).

Running server on CSE lab workstation

You will need to put the data directories under `/tmp`. You may need to change the PGPORT value, if some anti-social COMP9315 student had left their PostgreSQL server running on your workstation and was using port 5432.

Initialising data directories and running server

Once you have a copy of the `env` script and have set the values appropriately, you need to invoke it in every shell window where you plan to interact with the database. You can do this by explicitly running the following command in each window:

```
$ source /srvr/YOU/env
```

If that gets tedious, you might consider adding the above command to your shell's startup script (e.g., `~/.bash_profile`).

Once you've set up the environment, check that it's ok via the following commands:

```
$ echo $PGHOME
/srvr/YOU/pgsql
$ echo $PGDATA
YOUR_PGDATA ... i.e. whatever value you set it to ...
$ which initdb
/srvr/YOU/pgsql/bin/initdb
$ which pg_ctl
/srvr/YOU/pgsql/bin/pg_ctl
```

If the system gives you different path names to the above, then your environment is not yet set up properly. Are you sure that you `source`'d your `env` file?

If all of the above went as expected, you are now ready to create the data directories and run the server. You can do this via the command:

```
$ initdb
... some output eventually finishing with something like ...
Success. You can now start the database server using:

    pg_ctl -D YOUR_PGDATA -l logfile start
```

If you look at your data directory now, you should see something like:

```
$ ls $PGDATA
base          pg_ident.conf  pg_serial     pg_tblspc    postgresql.auto.conf
global        pg_logical     pg_snapshots  pg_twophase  postgresql.conf
pg_commit_ts  pg_multixact   pg_stat        PG_VERSION
pg_dynshmem   pg_notify      pg_stat_tmp   pg_wal
pg_hba.conf   pg_replslot    pg_subtrans   pg_xact
```

You shouldn't start the server straight away, however, since there's one more bit of configuration needed. You need to edit the `postgresql.conf` file in the $PGDATA directory and change the values of the following:

- change the value of the `listen_addresses` parameter to `''`: this means that only Unix-domain sockets can be used to connect to the server (saving you fighting over TCP ports);
- reduce the value of `max_connections` from 100 to **8**: this reduces the resources tied up by the server to support those connections potentially occurring; and
- set the value of the `unix_socket_directories` parameter to the full path of your **`$PGDATA`** directory (e.g. `/srvr/YOU/pgsql/data`): this specifies where PostgreSQL keeps its connection sockets, and should be the same as your $PGDATA so `psql` and other clients can connect; and
- set the value of `max_wal_senders` to e.g. 4 (or any value less than whatever value you use for `max_connections`)

Once you're done, the "connections and authentications" part of your modified `postgresql.conf` file should look like (with the changes highlighted in red):

```
#------------------------------------------------------------------------------
# CONNECTIONS AND AUTHENTICATION
#------------------------------------------------------------------------------

# - Connection Settings -

listen_addresses = ''                   # what IP address(es) to listen on;
                                        # comma-separated list of addresses;
                                        # defaults to 'localhost'; use '*' for all
                                        # (change requires restart)
#port = 5432                            # (change requires restart)
max_connections = 8                     # (change requires restart)
#superuser_reserved_connections = 3     # (change requires restart)
unix_socket_directories = 'YOUR_PGDATA' # comma-separated list of directories
                                        # (change requires restart)
#unix_socket_group = ''                 # (change requires restart)
#unix_socket_permissions = 0777         # begin with 0 to use octal notation
                                        # (change requires restart)
....
max_wal_senders = 4
```

Note that it doesn't matter that the file says `port = 5432`: this value will be overridden by whatever you set your PGPORT environment variable to.

Note also that the 5432 also doesn't matter because the # at the start of the line means that it's a comment. In the case of the lines that you are supposed to change, **make sure that you remove the # from the start of those lines**.

Everything is now ready to start your PostgreSQL server, which you can do via the command:

```
$ pg_ctl start -l $PGDATA/log
```

Note that PostgreSQL says "server starting", whereas it should probably say "attempting to start server".

A quick way to check whether the server is working is to run the command:

```
$ psql -l
SET
                             List of databases
   Name    | Owner | Encoding | Collate |    Ctype    | Access privi
leges
-----------+-------+----------+---------+-------------+-------------
------
 postgres  | YOU   | UTF8     | C       | en_AU.UTF-8 |
 template0 | YOU   | UTF8     | C       | en_AU.UTF-8 | =c/YOU
     +
           |       |          |         |             | YOU=CTc/YOU
 template1 | YOU   | UTF8     | C       | en_AU.UTF-8 | =c/YOU
     +
           |       |          |         |             | YOU=CTc/YOU
(3 rows)
```

It is possible that the server may not start correctly. If the server does not appear to have started, you can check why by looking at the tail of the server log:

```
$ tail -20 $PGDATA/log
... information about what happened at server start-time ...
```

Note that you'll get error messages about not being able to run the statistics collector, and a warning that autovacuum was not started. These are not an issue at this stage.

If you're on grieg and see *many* errors like

```
epoll_create1 failed: Function not implemented
```

in your server log file, your PostgreSQL **will not work**. This means that you forgot to make the changes to the `latch.c` file that we mentioned above. Go back and fix it and then re-run `make install`.

which will give you a list of databases like the above if the server is running. If the server is not running, you'll get a message something like:

```
psql: could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "YOUR_PGDATA/.s.PGSQL.5432"?
```

If this happens, you should check the log file to find out what went wrong. (Other things to check in case of problems are described below).

Assuming that the server is running ok, you can now use it to create and manipulate databases (see the example below). Once you've finished your session using PostgreSQL, you need to stop the server.

```
$ pg_ctl stop
waiting for server to shut down.... done
```

If you still have a process that's using the database (e.g. a `psql` process in another window), then the server won't be able to shut down. You'll need to quit all of the processes that are accessing the database before the above command will work.

If you put your data under `/tmp`, you *must* also remove the data directories. You can do this via the command:

```
$ rm -r /tmp/pgsql.YOU
```

The `pgs` script

Since the above process is rather fiddly, we have provided a script that provides a single command to setup your data directory (if needed) and start your server. It still requires you to set the values in your `env` file appropriately, however. The script is called `pgs` and is located in the directory `/home/cs9315/bin`.

The `pgs` script is designed to help you manage your PostgreSQL servers and do a bit of error checking along the way to see if everything is ok. It has four possible arguments:

setup	create a new PGDATA directory (complains if one already exists)
cleanup	remove the PGDATA directory (make sure you backup anything important before doing this)
start	start your PostgreSQL server (waiting until it actually starts ok)
stop	stop your PostgreSQL server (waiting until it actually stops ok)

The `pgs` script is just a wrapper around two of the PostgreSQL commands mentioned above:

initdb	sets up the PGDATA directory
pg_ctl	controls the operation of the PostgreSQL server

As noted above, the `pgs` script has four modes of operation:

- **setting up the data directory:**

  If you leave your data under `/srvr/YOU/pgsql`, then you only need to do this once. If your data is on `/tmp`, you will need to do this each time you want to have a session using PostgreSQL.

  ```
  $ pgs setup
  Using PostgreSQL with data directory /your/PGDATA/directory
  The files belonging to this database system will be owned by user "YOU".
  This user must also own the server process.
  ```

  Running this command should eventually produce the output:

  ```
  Success. You can now start the database server using:

      pg_ctl -D YOUR_PGDATA -l logfile start
  ```

  After doing the above, your PostgreSQL server is ready to start and use.

- **starting the PostgreSQL server**:

  ```
  $ pgs start
  Using PostgreSQL with data directory YOUR_PGDATA
  waiting for server to start..... done
  server started
  Check whether the server started ok via the command 'psql -l'.
  If it's not working, check YOUR_PGDATA/log for details.
  ```

  If the "waiting for server to start" is followed by an ever-growing sequence of dots, it means that the server is not starting properly. You'll need to do some additional debugging (see below) for such cases.

- **stopping the PostgreSQL server:**

  The following command stops the PostgreSQL server:

  ```
  $ pgs stop
  Using PostgreSQL with data directory YOUR_PGDATA
  waiting for server to shut down.... done
  ```

  If you get an ever-growing sequence of dots, it means that the server cannot shut down. This is typically caused by some other process being connected to your PostgreSQL server (e.g. a `psql` process running in another window).

- **cleaning (removing) the data directory:**

  You only need to do this if you are not keeping your databases between sessions with PostgreSQL, i.e. because you have put the data directory under `/tmp`.

  ```
  $ pgs cleanup
  Using PostgreSQL with data directory YOUR_PGDATA
  This will remove all files under YOUR_PGDATA
  Do you want to continue? y
  ```

  If you decide that you really don't want to remove the data directories, typing anything other than `y` or `yes` will not do the cleanup. If you accidentally remove your data directory, it is easy enough to restore using `pgs setup`.

A Typical session with PostgreSQL

Once you've got your PostgreSQL server installed, this is what you'd normally do to use it:

```
$ source /srvr/YOU/env
$ pgs setup
... BUT ONLY if your PGDATA directory is on /tmp ...
$ pgs start
... hopefully concluding with the message ...
server started
$ psql -l
... hopefully giving a list of databases ...
$ createdb myNewDB
$ psql myNewDB
... do stuff with your database ...
$ pgs stop
... hopefully concluding with the message ...
server stopped
$ pgs cleanup
... BUT ONLY if your PGDATA directory is on /tmp ...
```

Reminder

You *must* shut down your server at the end of each session with PostgreSQL if you're working on the CSE workstations. Failure to do this means that the next student who uses that workstation may need to adjust their configuration (after first working out what the problem is) in order to start their server.

A Sample Database

Once your server is up-and-running, you ought to load up the small sample database (on beers) and try a few queries on its data. This is especially important if you haven't used PostgreSQL before; you need to get used to its interactive interface.

You can set up the beer database as follows:

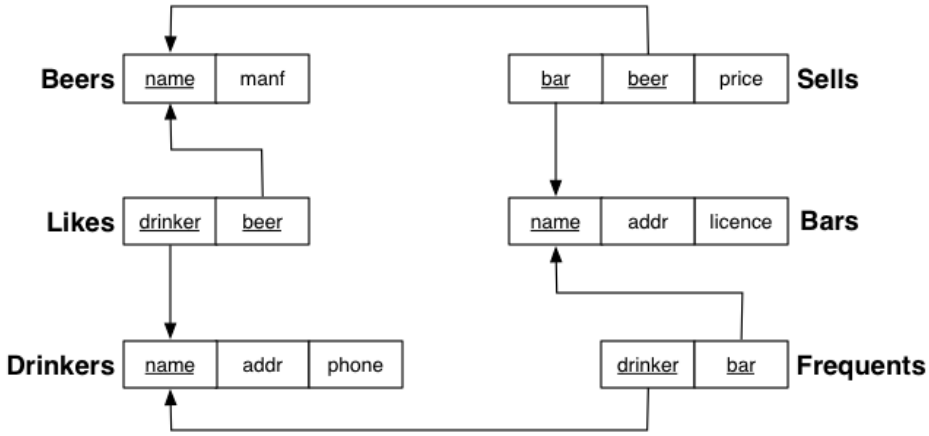
```
$ createdb beer
$ psql beer -f /web/cs9315/19T2/pracs/p01/beer.dump
... around 20 lines include SET, CREATE TABLE, ALTER TABLE...
$ psql beer
SET
psql (11.3)
Type "help" for help.

beer=# select count(*) from beers;
 count
-------
    24
(1 row)

beer=# \d
... gives a list of tables in the database ...
beer=#
... explore/manipulate the database ...
beer=# \q
$
```

For exploring the database with `psql`, there are a collection of `\d` commands. You can find out more about these via `psql`'s `\?` command or by reading the PostgreSQL manual chapter on `psql`.

To help with your explorations of the database, here is an diagram of the schema. Table/relation names are in bold; each box represents one attribute; primary keys are underlined. Note that all primary keys are symbolic (not numeric) in this database. You can look at the SQL schema from within `psql`.



Sorting out Problems

It is very difficult to diagnose problems with software over email, unless you give sufficient details about the problem. An email that's as vague as "My PostgreSQL server isn't working. What should I do?", is basically useless. Any email about problems with software should contain details of

- what you were attempting to do
- precisely what commands you used
- precisely what output you got

One way to achieve this is to copy-and-paste the last few commands and responses into your email.

But even with all of that information, there's a whole host of other environment information that's needed to be able to seriously work out why your server isn't running, that you can't put in an email. That's why it's better to come to a consultation, where we can work through the problem on a workstation (which is usually very quick).

Can't start server?

When you use `pgs start` to try to start your PostgreSQL server, you observe something like:

```
$ pgs start
Using PostgreSQL with data directory YOUR_PGDATA
waiting for server to start.....................................................pg_ctl: could not start server
Examine the log output.
Check whether the server started ok via the command 'psql -l'.
If it's not working, check /srvr/YOU/pgsql/log for details.
```

Take the advice given to you by the command and look at the end of the log file to see if there are any clues there. You can do this via the command:

```
$ tail -20 /srvr/YOU/pgsql/log
```

Sometimes you may need to look at more than the last 20 lines of the log file to find the relevant error message. Most of the error messages are self-explanatory, and you should learn what to do if any of them occurs. Some examples:

```
FATAL:  lock file "postmaster.pid" already exists
HINT:  Is another postmaster (PID 31265) running in data directory "YOUR_PGDATA"?

# You may already have another PostgreSQL server running
# Or, the previous server may have quit without cleaning up the postmaster.pid file
# Note that the server process may be running on another machine if you run your
#  server on the local machine rather than grieg
# If the server is running on another machine, log in there and run "pgs stop"

LOG:  could not bind IPv4 socket: Address already in use
HINT:  Is another postmaster already running on port 5432? If not, wait a few seconds and retry.
WARNING:  could not create listen socket for "localhost"
FATAL:  could not create any TCP/IP sockets

# Another user is running a PostgreSQL server on this machine
# Change the PGPORT value in /srvr/YOU/env
#  and then reset your environment and try starting the server again

FATAL:  could not open relation mapping file "global/pg_filenode.map": No such file or directory
FATAL:  could not open relation mapping file "global/pg_filenode.map": No such file or directory
FATAL:  could not open relation mapping file "global/pg_filenode.map": No such file or directory
FATAL:  could not open relation mapping file "global/pg_filenode.map": No such file or directory

# This means that there is another PostgreSQL server of yours still running
# You'll need to find it e.g. using the command "pgs status"
# Note that the process could be running on any CSE machine where you ever
#  ran a PostgreSQL server, so you may need to check on multiple machines
# Once you've found it, stop the server using the Unix kill command
# and then reset your environment and try starting the server again
```

Sometimes the `pg_ctl` command will give a message that the server has failed to start but you'll get no error messages at the end of the log file, which will look something like:

```
LOG:  database system was shut down at 2011-08-03 11:38:26 EST
LOG:  database system is ready to accept connections
```

One cause of this is having different directories for PGHOST in the `/srvr/YOU/env` file and for `unix_socket_directory` in the *YOUR_PGDATA*`/postgresql.conf` file. It is critical that these two both refer to the same directory. You can check this by running the command:

```
$ psql -l
psql: could not connect to server: No such file or directory
        Is the server running locally and accepting
        connections on Unix domain socket "/srvr/YOU/pgsql/.s.PGSQL.5432"?
```

You should then check the *YOUR_PGDATA*`/postgresql.conf` file to see whether `unix_socket_directories` has been set to `/srvr/YOU/pgsql`. Note that the directory name may not be exactly the same as this; the critical thing is that the directory be the same in both places.

Can't shut server down?

When you use `pgs stop` to try to shut down your PostgreSQL server, you observe something like:

```
$ pgs stop
Using PostgreSQL with data directory YOUR_PGDATA
waiting for server to shut down.....................
```

and no `done` ever appears.

This is typically because you have an `psql` session running in some other window (the PostgreSQL server won't shut down until all clients have disconnected from the server). The way to fix this is to find the `psql` session and end it. If you can find the window where it's running, simply use `\q` to quit from `psql`. If you can't find the window, or it's running from a different machine (e.g. you're in the lab and find that you left a `psql` running at home), then use `ps` to find the process id of the `psql` session and stop it using the Linux `kill` command.

Can't restart server?

Occasionally, you'll find that your PostgreSQL server was not shut down cleanly the last time you used it and you cannot re-start it next time you try to use it. The symptoms are:

```
Using PostgreSQL with data directory YOUR_PGDATA
pg_ctl: another server might be running; trying to start server anyway
pg_ctl: could not start server
Examine the log output.
Check whether the server started ok via the command 'psql -l'.
If it's not working, check /srvr/YOU/pgsql/log for details.
```

If you actually go and check the log file, you'll probably find, right at the end, something like:

```
$ tail -2 /srvr/YOU/pgsql/log
FATAL:  lock file "postmaster.pid" already exists
HINT:  Is another postmaster (PID NNNN) running in data directory "YOUR_PGDATA"?
```

where *NNNN* is a process number.

There are two possible causes for this: the server is already running or the server did not terminate properly after the last time you used it. You can check whether the server is currently running by the command `psql -l`. If that gives you a list of your databases, then you simply forgot to shut the server down last time you used it and it's ready for you to use again. If `psql -l` tells you that there's no server running, then you'll need to do some cleaning up before you can restart the server ...

When the PostgreSQL server is run, it keeps a record of the Unix process that it's running as in a file called:

```
YOUR_PGDATA/postmaster.pid
```

Normally when your PostgreSQL server process terminates (e.g. via `pgs stop`), this file will be removed. If your PostgreSQL server stops, and this file persists, then `pgs` becomes confused and thinks that there is still a PostgreSQL server running even though there isn't.

The first step in cleaning up is to remove this file:

```
$ rm YOUR_PGDATA/postmaster.pid
```

You should also clean up the socket files used by the PostgreSQL server. You can do this via the command:

```
$ rm YOUR_PGDATA/.s.PGSQL.*
```

Once you've cleaned all of this up, then the `pgs` command ought to allow you to start your PostgreSQL server ok.