POSTGRES INSTALLATION INSTRUCTIONS

Release 4.1

Document Overview

1. Introduction

This document gives installation instructions for the POSTGRES database system under development at the University of California, Berkeley. POSTGRES is distributed in source code format and is the property of the Regents of the University of California. However, the University will grant unlimited commercialization rights for any derived work on the condition that an educational license to the derived work is obtained. For further information, consult the Berkeley Campus Software Office, 295 Evans Hall, University of California, Berkeley, CA 94720.

The University and the POSTGRES development group provide no warranty as to the fitness of the code for any purpose whatsoever, and cannot guarantee to assist in fixing problems. This is *unsupported* software.

2. Site Requirements

2.1. Hardware

POSTGRES currently has been tested by the POSTGRES development team on Sun Microsystems Sparc architecture machines running SunOS 4.1 and higher. POSTGRES is also supported on DECstations 3100's and 5000's running Ultrix 4.1 and higher. In order to use POSTGRES, your machine should have at least 8 megabytes of memory and you will require at least 45 megabytes of disk space to hold source, binaries, and user databases. If you choose to compile POSTGRES for source-level debugging, you will need roughly twice as much disk space. See the section on compilation for details.

The Ultrix version requires a kernel which allows 4 megabytes of shared memory. Also, the original release of Ultrix 4.3 has a kernel bug that causes the operating system to hang when running POSTGRES. The bug is in the shared memory module in the kernel, and a patch is available from DEC which fixes the bug. Contact your DEC representative for the patch.

2.2. Software

2.3. Distribution Tape

These instructions assume you have a POSTGRES Version 4.1 distribution tape (in either 9 track, SCSI cartridge, or TK50 cartridge format) or a POSTGRES tar file. This release comes with a pre-installed version of POSTGRES ready to run for the two platforms we officially support. This is a test to see if our users can run the pre-installed system to avoid compiling the sources from scratch. Should any problems arise with the pre-installed system, it is recommended that the sources be rebuilt from scratch to rule out any compatability problems between the system we compiled on here and your own. There are two potential releases you may want (same source, different binaries). The tar version of the ultrix release is `postgres-v4r1.ultrix.tar.Z` and the sparc release is `postgres-v4r1.sparc.tar.Z` available via anonymous ftp from postgres.cs.berkeley.edu.

2.4. Expertise

Once a site is properly configured and POSTGRES is installed, very little UNIX expertise is required to maintain things. However, initially setting things up for your site to run POSTGRES may be difficult and we advise that the person installing POSTGRES be familiar with system administration procedures. Also note that various steps require superuser privilege on the system, so we advise that your site's system administrator read this document also.

2.5. Configuration

This section briefly describes the configuration you need to run POSTGRES. Read this to familiarize yourself with the procedure. Detailed instructions for making appropriate modifications to your system are given later in this document.

2.5.1. Operating System

POSTGRES expects things to be configured for BSD by default. If the default on your site is to use the SunOS SysV compiler and libraries then you may have to make some changes to this procedure before compiling POSTGRES.

To compile this release from sources requires a new version of make which comes from the latest BSD release. A bootstrapping version of the make sources is included with this release, as is a precompiled binary with its configuration files. The make program is installed using the name `bmake` to avoid any conflict with your native make.

2.5.2. Disk Requirements

POSTGRES requires 50 megabytes of disk space, preferably on a single partition, to compile and install from sources. If you run the pre-installed system, you only need 25 megabytes to hold the executables and sources.

2.5.3. Kernel

POSTGRES makes use of the optional System V shared memory operations provided by SunOS and DEC Ultrix which require a properly configured kernel which is in general different than the factory-shipped generic kernel. See the section on kernel configuration for details.

3. Installing POSTGRES

There are potentially two routes you can go to install this system. Either you can choose to run the pre-installed system shipped with the release, or you can compile and install the sources from scratch. In either case your kernel must be configured to support the shared memory and semaphore operations required by the code. Assuming you have run POSTGRES in the past, you may already be properly configured to run. Also, should you opt to run the pre-installed system, let us reiterate that if you run into any complications, it would be wise to recompile the sources from scratch before reporting any problems to our mailing list. We are, however, interested in any feedback you may have on the pre-installed system. In both cases you need to decide where to load the system.

3.1. Step 1 – Preparation

Some of the tasks involved in this step normally fall in the domain of the site's system administrator and may require superuser privilege. If possible, we advise you to have your system administrator perform these steps.

3.1.1. Find a good place for POSTGRES

You should locate a disk partition with at least 50 megabytes of free space available for POSTGRES. If you plan to run the pre-installed system, it would be best if you loaded the release into the directory `/usr/local/postgres`, as that is what the release was configured for. It is not mandatory, though, as the only relevent dependency is that the programs are setup to look for the database directory in `/usr/local/postgres/data`, but that can be changed by setting the environment variable PGDATA somwhere else before starting the postmaster. Also note that you can install the system anywhere, and then make a symbolic link from `/usr/local/postgres` to the place you really loaded the system. If you are compiling the system from scratch, it doesn't matter where you load the system at all (as long as there is space).

3.1.2. Creating the POSTGRES directory

Once you have decided, create the directory to hold the release if it doesn't already exist. Then **cd** to this directory and type **pwd**. This is the full path of the directory you will install POSTGRES in. For example:

df

Filesystem	kbytes	used	avail	capacity	Mounted on
/dev/xy0a	8421	6703	875	88%	/
/dev/xy0f	10829	6743	3003	69%	/pub.MC68020
/dev/xy2h	110811	81181	18548	81%	/usr3
/dev/xy2g	221279	167405	31746	84%	/b
/dev/xy1g	221279	138365	60786	69%	/usr/local
/dev/xy1a	8179	944	6417	13%	/tmp
/dev/xy0h	119999	101623	6376	94%	/usr.MC68020
/dev/xy0g	156033	135499	4930	96%	/usr2
/dev/rf0d	539421	465026	20452	96%	/a

/usr/local looks like a good place (it has 60 megs free)
so we decide to create the POSTGRES directory there...

\# **cd /usr/local**
\# **mkdir postgres**
\# **cd postgres**
\# **pwd**
/usr/local/postgres
\#

3.2. Creating the POSTGRES user

Finally, we need to create a user called POSTGRES whose shell is `/bin/csh`. Though not necessary, it is convenient to make the home directory of the POSTGRES user the pathname where we install the system. This can be done using the "adduser" procedures particular to your platform and site. See your system administration manual for details.

The reason we make a postgres login is so that when we install the system and create the database directory, everything is owned by this postgres user. Then, we start the postmaster when logged in as this postgres user, and all the backends are also started by the user postgres and are able to access the databases. Although you can probably make this work without creating a postgres login (note that the restriction about having a postgres login with uid 6 was removed a while ago), we provide no support for doing so (you are, as they say, "on your own").

3.3. Step 2 - Loading POSTGRES

Now you are ready to load the POSTGRES files onto your system. To do this, you will need either a distribution tape or a POSTGRES tar file (available via anonymous ftp from postgres.cs.berkeley.edu).

If you are loading POSTGRES from a tape, follow these instructions; if you are loading from a tar file obtained via FTP, skip to the section "Loading POSTGRES from a Tar File".

3.3.1. Loading POSTGRES from a Tape

Login as postgres and change directories to the postgres directory that was created.

3. Run "tar" with the following options

% tar xvf <tape-device>

where <tape-device> is the name for your tape device, i.e., /dev/rmt0, /dev/rst8, etc.

The file "postgres-v4r1.XXX.tar.Z" will appear in your POSTGRES home directory, where XXX is the name of the platform. You may need to re-wind your tape to get it out of your tape drive - see your system administrator for instructions.

Proceed to the next section "Loading POSTGRES from a Tar File".

3.4. Loading POSTGRES from a Tar File

If you are not logged in as `postgres` already, do so now. For the purpose of this discussion, the POSTGRES tar file will be called `postgres-v4r1.XXX.tar.Z`

Uncompress the tar file.

    % uncompress postgres-v4r1.XXX.tar.Z

A larger file should now be in the POSTGRES home directory, and the '.Z' ending should be gone, so it is now named postgres-v4r1.XXX.tar

Extract POSTGRES from the tar file using the following command:

    % tar xvf postgres-v4r1.XXX.tar

Lots of file names and such should appear on the screen. This step may take several minutes.

Now do an "ls":

    % ls

The output of the ls should look something like:

```
COPYRIGHT      README      doc       local     src
INITIAL_SETUP  bin         include   man
Makefile       data        lib       obj
```

At this point you have loaded the POSTGRES files. Remove the tar file to get back the space.

    % rm postgres-v4r1.XXX.tar

3.5. Step 3 - Configuration

This step requires familiarity with configuring a UNIX kernel. If you are unfamiliar with this procedure, we advise you to read the section on configuring a kernel in the SunOS or DEC system administration manual carefully. This task requires superuser privilege and should probably not be done without the assistance of your system administrator. We assume that whoever undergoes this procedure has an understanding of the process and procedures involved.

POSTGRES uses shared memory segments which must be compiled into the kernel of the host which will act as the POSTGRES server. If you try to run a POSTGRES backend process on a machine without enough shared memory, the backend will abort with an error message.

This is by far the most complicated part of the installation so these steps should be performed by someone with system administration experience. Again, we advise you to consult the system administration section of your manual before doing this step.

For a brief discussion of shared memory, you may want to consult the Man pages for *shmget()*, *shmop()*, *shmctl()*, etc. Now proceed to the appropriate section for your machine.

3.5.1. Kernel reconfiguration for Sparcs

In order to reconfigure Sun or Sparc kernel, you will have to become root and add some lines to /usr/sys/conf (your kernel config file) and /usr/sys/conf/param.c (your kernel parameters file). We *strongly* advise you to make a spare copy of your system's original config and parameter files before you make any changes.

The following lines should be added to /usr/sys/conf/KERNEL:

```
options     IPCMESSAGE      # SystemV IPC Message Facility
options     IPCSEMAPHORE    # SystemV IPC Semaphore Facility
options     IPCSHMEM        # SystemV IPC Shared-Memory Facility
```

```
options         EMOREIPCS     # more semaphores and shared memory (for 8M)
```

At Berkeley, we substitute the line:

```
options         EMOREIPCS     # more semaphores and shared memory (for 8M)
```

with the line:

```
options         TTMOREIPCS    # more semaphores and shared memory (for 32M)
```

to allocate more shared memory so that we can run more POSTGRES backends at the same time. Either of the lines will result in a kernel that has enough shared memory allocated.

Also add the following lines to the *top* of /usr/sys/conf/param.c:

```
/*
 * LOCAL DEFINITIONS START
 */

#ifdef    EMORESEMS
#define EMOREIPCS
#endif    /* defined(EMORESEMS) */

#ifdef    TTMORESEMS
#define TTMOREIPCS
#endif    /* defined(TTMORESEMS) */

#ifdef    EMOREIPCS
#define SEMMNI         30       /* # of semaphore identifiers */
#define SEMMNS                  180      /* # of semaphores in system */
#define SEMUME                  10       /* max # of undo entries per process */
#define SEMMNU                  30       /* # of undo structures in system */

#define SHMPOOL        1536              /* max total shared memory system wide (in Kbytes) */
#define SHMSEG                  6        /* max attached shared memory segments per process */
#define SHMMNI                  100      /* # of shared memory identifiers */
#endif    /* defined(EMOREIPCS) */

#ifdef    TTMOREIPCS
#define SEMMNI         60       /* # of semaphore identifiers */
#define SEMMNS                  384      /* # of semaphores in system */
#define SEMUME                  10       /* max # of undo entries per process */
#define SEMMNU                  30       /* # of undo structures in system */

#define SHMPOOL        8192              /* max total shared memory system wide (in Kbytes) */
#define SHMSEG                  6        /* max attached shared memory segments per process */
#define SHMMNI                  100      /* # of shared memory identifiers */
#endif                          /* defined(TTMOREIPCS) */

/*
 * LOCAL DEFINITIONS END
 */
```

After adding these lines, run config over the config file, install the new kernel, and reboot.

3.5.2. Kernel reconfiguration for DECs

In order to reconfigure your DECstation 3100 or 5000 Ultrix kernel, you will have to become root and add some lines to /usr/sys/conf (your kernel config file).

The following lines should be added to /usr/sys/conf/KERNEL:

```
smmax    256
smseg    12
smbrk    1024
```

After adding these lines, run *config* over the configuration file, install the new kernel, and reboot.

Also, remember that Ultrix 4.2 has a kernel bug that causes the operating system to hang when running POSTGRES. You should apply the DEC supplied patch at this point if you have not already.

3.6. Running the pre-installed POSTGRES system

Assuming you have loaded the system into /usr/local/postgres, there is very little else to do. Add /usr/local/postgres/bin to your shell path. If you run csh, you would put

```
set path=(/usr/local/postgres/bin $path)
```

in the .login for the postgres user. Or if you run sh or ksh, put

```
PATH=/usr/local/postgres/bin:$PATH
export PATH
```

in your .profile. Then log back in so the change takes effect. Now run the following command

```
% newbki
```

This will reset the userid of the special POSTGRES user of the database to have the proper value. Even though newbki allows you to enter a login name, you should select the default value of `postgres`. It allows a different name because at some point we plan to remove the restriction that a special user called `postgres` even exists -- but not yet...

After doing this, you may wish to install the precompiled version of bmake into /usr/local/bin. This will allow you to recompile POSTGRES right away without bootstrapping bmake. To do this, cd into the directory called "local". You will see two directories and a README. Become the superuser and do the following:

```
# cp bin/bmake /usr/local/bin
# mkdir /usr/local/lib /usr/local/lib/mk
(you may get an error if this directory alreadt exists - ignore it)
# cp lib/mk/* /usr/local/lib/mk
```

If you installed POSTGRES somewhere other than /usr/local/postgres, everything can still work right if you set the environment variable PGDATA for the special POSTGRES user. Lets assume you installed POSTGRES in the directory "/private/packages/postgres". If you run csh, put the following in the .login for the POSTGRES user

```
setenv PGDATA /private/packages/postgres/data
```

or if you run sh or ksh, put the following in .profile

```
PGDATA=/private/packages/postgres/data
export PGDATA
```

This will instruct the postmaster and backends to look in that directory for the databases.

You can now skip to the setion called "Creating the initial database".

3.7. Compiling and Installing POSTGRES

The sources for POSTGRES are all under the directory src/. The first step is to install the new bmake program. By the way, you can look at the previous section entitled "Running the pre-installed system" in the paragraph about bmake, and just copy the pre-compiled bmake to /usr/local/{bin,lib} if you want and skip step 2) in this section. What follows is a step by step list to compile and install this release from sources:

- build step 1: customization -

Cd into the src/ directory and edit the file "Makefile.global". By default the PORTNAME is "ultrix4". Change this to "sparc" if you are running on the sparcstation. Also, if you do not want "bmake" installed into /usr/local/bin and /usr/local/lib, change the values of TOOLSBINDIR and TOOLSLIBDIR and make sure these directories exist.

At this point you can also change where the POSTGRES executable files are installed and where it looks for the database directory. In general the rest of the configuration switches are self-explanatory. The only one which must be set is PORTNAME.

- build step 2: bootstrapping the new bmake program -

Cd into src/tools/bmake and type

```
% make -f Makefile.boot
```

You may get warning messages during this bootstapping process about "illegal combination of pointer and integer" -- just ignore them. This should compile and install the "bmake" program and its supporting files, including the POSTGRES related makefile templates. If all went well you will now be able to use the new make program by typing "bmake", assuming you have /usr/local/bin in your shell path. You may have to type "rehash" if you run csh(1), since it's too stupid to know new executables have been placed in its path.

- build step 3: compile and install the world -

Cd back to the src/ directory (i.e. cd ../..) and type

```
% bmake all install
```

This builds and installs the entire system. The Makefiles contain directives for running all the underlying Makefiles in all the directories, so the whole thing should unfold and compile beautifully and install to the target directory. Should this not be the case, it would be a good idea to save the results of the compile in a file. If you run csh(1), you could type

```
% bmake all install >& mk.log
```

and if you run ksh(1) or sh(1), type

```
% bmake all install > mk.log 2>&1
```

This will save the results in the file "mk.log" so you can inspect it later. This would be an ideal opportunity to get some donuts and coffee.

3.8. Creating the initial database

POSTGRES databases are stored in the directory .../postgres/data. After you have compiled POSTGRES, you will need to create the initial database. If you haven't already put the path to the POSTGRES executable programs in the shell path, do the following (assuming POSTGRES was loaded into /usr/local/postgres). If you run csh, you would put

```
set path=(/usr/local/postgres/bin $path)
```

in the .login for the POSTGRES user. Or if you run sh or ksh, put

```
PATH=/usr/local/postgres/bin:$PATH
export PATH
```

in your .profile. Then log back in so the change takes effect.

Now you can create the initial database by running the following command.

    % initdb

If that completes successfully, congratulations.

Now, to make the system operational you must run the postmaster. The section after "Testing" discusses this.

3.9. Step 4 - Testing

We suggest you run the regression tests to make sure the release was installed successfully. You must have installed the bmake program to run the test. Also, you need to have the postmaste running to run the test, so type the following:

    % postmaster &

Change directories to src/regress/regress and type the following bmake command:

    % cd src/regress/regress
    % bmake runtest

This will run a whole slew of regression tests and might take a long time to run. When it's done, the output of the test is in the file "obj/regress.out". You can compare this to a sample run that we supply in the file "sample.regress.out" with the following command:

    % diff sample.regress.out obj/regress.out

Unfortunatly you will see many differences corresponding to the timestamp lines and occasional lines where object id numbers (OID's) are different. We regret that we do not have a more sophisticated script for weeding out these innocuous differences at this time. If you see any differences not corresponding to time stamps or OIDS, there may be a problem. Have your local POSTGRES expert take a look at it.

4. Running POSTGRES

POSTGRES is designed to be a multiuser system. In practice, POSTGRES consists of three (or more) processes:

- the postmaster,
- the terminal monitor, and
- the backend.

Users are expected to use the terminal monitor for direct access to the database. The terminal monitor sends commands to the postmaster which forwards commands to a backend.

4.1. The POSTGRES Postmaster

The postmaster is a process which manages communication between the user's terminal monitor and a POSTGRES backend. Without a running postmaster, the terminal monitor will not be able to connect to a backend. In general, the postmaster must be running for you (or others) to run any of the normal POSTGRES commands. Always start the postmaster when logged in as the special "postgres" user, otherwise the system will not be able to access the database files. To start it, type

    % postmaster &

Now you can leave the postmaster running if you wish to keep the database system accessable. In general, when the postmaster is running, the system is running. When it is not running, none of the frontend programs will work.

4.2. The POSTGRES Terminal Monitor

The POSTGRES terminal monitor is a front-end user interface to the POSTGRES backend.

Lets assume *database* is the name of the database you want to use. It is an error if *database* does not exist, so to create the database, you would type

> createdb *database*

Now we will run the monitor:

> % monitor *database*

```
Welcome to the C POSTGRES terminal monitor

Go
*
```

> *The "*" is the terminal monitor prompt. We are now talking to the backend, so let's send a simple test query: list the names and user ids of the POSTGRES users. We terminate the query with a \g — the "go" command to the terminal monitor.*

***retrieve (u.usename, u.usesysid) from u in pg_user**
\g

```
Query sent to backend is "retrieve (u.usename, u.usesysid) from u in pg_user"
```

usename	usesysid
postgres	6
mike	799
sp	1511
jhingran	943
cimarron	2359
goh	1994
ong	2802
hong	2469
mao	1806
marc	2435
margo	2697
sullivan	1517

```
| kemnitz      | 3491        |
-----------------------------
| choi         | 3898        |
-----------------------------
| mer          | 3665        |
-----------------------------

Go
```

Okay, this worked, too. Now we'll quit.

***\q** `I live to serve you.  %`

4.3. The POSTGRES Backend

The POSTGRES backend is the process which does all the "real" work. This process is started by the postmaster when the postmaster receives a connection from a terminal monitor, so you should not normally need to start up the backend yourself. Should you wish to start the backend and talk to it directly (without a terminal monitor) you can do this by typing:

% **postgres** *database*

where *database* is the name of the database you wish to use. If you run a backend in this manner, you will be talking to the backend parser directly. We recommend using the terminal monitor; if you are using Postgres as a multiuser system, running the backend can result in locking failures and corrupt databases, as the Postmaster handles shared resources such as semaphores and shared memory. In general, never do this if there is a postmaster running. In addition, returned tuples are displayed more usefully and input is buffered better. The backend is used interactively primarily during debugging.

4.4. POSTGRES Support Programs

Included in POSTGRES are a handful of support programs. Most of these are used internally by the system but here is a list of them for your information.

initdb – creates the initial template database
cleardbdir – totally destroys the data/ directory, allowing a new initdb
createdb – creates new POSTGRES databases
createuser – add a new user to the POSTGRES system
destroydb – destroys POSTGRES databases
destroyuser – delete a user from the datasbe system
ipcclean – frees up garbage shared memory from failed backends
newbki – adjust userid of "postgres" in the database
pg_version – make version numbers for createdb
pg_id – gets user id's - used by various commands
pagedoc – disk page doctor
shmemdoc – shared memory buffer pool doctor
vacuum – database vacuum cleaner
pg_copytree – copy te source tree (probably shouldn't be shipped)
pcat – inversion filesystem cat command
pcd – inversion filesystem cd command
pls – inversion filesystem ls command
pmkdir – inversion filesystem mkdir command
pmv – inversion filesystem mv command
ppwd – inversion filesystem pwd command
prm – inversion filesystem rm command
prmdir – inversion filesystem rmdir command

5. Optional Installation

5.1. Installing LIBPQ, the POSTGRES frontend library

The file .../lib/libpq.a is created when you install the system (or on the pre-installed system). This library contains various routines intended for use by frontend programs. You use this library if you want to execute POSTGRES queries from a C program. If you plan on doing software development, you may wish to copy this file to /usr/lib so that the C compiler can reference it with -lpq.

5.2. Postgres header files

The directory .../include contains copies of all the header files that front-end applications might need. You can compile a frontend program with the -I cc directive as illustrated in the following example:

```
cc -I/usr/local/postgres/include -o foo foo.c -lpq
```

This feature of copying just the necessary header files into .../include was incorporated right before we produced the release and has not had sufficient testing. It your frontend programs complain about not being able to find header files, either add the missing header files to the include/ directory and notify us of the problem, or just point the -I compiler directive directly into the source as was done in the past. E.g.

```
cc -I/usr/local/postgres/src/backend -o foo foo.c -lpq
```

If you do this, you may need to add the following #define's in your source to set the PORTNAME - add them prior to any #includes:

```
#define PORTNAME ultrix4
#define PORTNAME_ultrix4
```

for ultrix, or

```
#define PORTNAME sparc
#define PORTNAME_sparc
```

We're sorry for any inconvenience this may cause, and fully intend that the include/ directory be more rigorously tested in the next release.

5.3. Wisconsin Benchmark Database

In .../postgres/src/regress/bench are files which are the queries used in the Postgres version of the Wisconsin benchmark. The Wisconsin benchmark illustrates "basic" relational performance using B-Tree indices on nontrivial amounts of data. To run the benchmark, type cd to that directory and type

```
bmake runtest
```

5.4. Minimal Installation

It is the intention that everything below the src/ directory can be removed, however we didn't have time to test this. Specifically, the include/ directory may not have all the necessary files to compile all the frontend programs, so you may might get burned if you remove the source. But if you'd like to try it out, we'd love to know if it works for you.... :)

6. Documentation Clear text and postscript versions of the manpages and documents are available in the directories man/ and doc/ at the top level. To recreate these documents, there are corresponding directories in src/{man,doc}. They are currently configured to require groff and friends, but you may be able to change the makefiles to use other facilities if you have them (good luck!). If you change directories to man/ and doc/ and type a

```
% bmake
% bmake install
```

in each, it will format and install the documents into the corresponding destination directories.

7. Miscellaneous

7.1. Bug reports

If you find a bug with POSTGRES, please send mail to

bug-postgres@postgres.Berkeley.EDU

or

(ucbvax!postgres!bug-postgres)

describing as precisely as possible the command that caused the problem, instructions on how to repeat the bug, and a script showing the bug.

7.2. Consulting

This software is unsupported, public domain software. Although we are interested in feedback, it is impossible for us to make any commitment to support in a research environment.

If you do want to talk directly to the Postgres group, electronic mail is strongly preferred. We can be reached via the Internet as

post_questions@postgres.Berkeley.EDU

or

(ucbvax!postgres!post_questions)

We can also be reached at (510) 642-7520, Monday through Friday, between 1 and 4 PM Pacific Time.

7.3. Postgres BBS A mailing list for Postgres announcements and discussion is available for anyone who is interested. If you wish to subscribe to this mailing list, send mail to

postgres-request@postgres.Berkeley.EDU

with "Add" as the subject. Note that mail sent to this address is processed **electronically.**

The mailing list itself is called

postgres@postgres.Berkeley.EDU

and all mail sent to this address will be will be routed to the mailing list membership.