POSTGRES INSTALLATION INSTRUCTIONS

Release 4.2

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 assistance in fixing problems. This is *unsupported* software.

2. Site Requirements

2.1. Hardware

POSTGRES currently has been tested by the POSTGRES development team on the following systems:

- Digital Equipment DECstation 3000 series (Alpha AXP architecture) machines running Alpha OSF/1 1.3.
- Digital Equipment DECstation 3100 and 5000 series (MIPS architecture) machines running Ultrix 4.2A and 4.3A.
- Hewlett-Packard H-P 9000 Series 700 and 800 (PA-RISC architecture) machines running HP-UX 9.00 and 9.01.
- International Business Machines RS/6000 (POWER architecture) machines running AIX 3.2.5.
- Sun Microsystems (SPARC architecture) machines running SunOS 4.1.2 (Solaris 1.0.1) and 4.1.3. (POSTGRES will **not** run on machines running SunOS 5.x (Solaris 2.x)).

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 patch number CLD-CXO09447. (This patch has been included in Ultrix 4.3a.)

2.2. Software

2.3. Distribution Tape

These instructions assume you have a POSTGRES Version 4.2 distribution tape (in either SCSI cartridge or DEC TK50 cartridge format) or a POSTGRES tar file. The previous version of POSTGRES came with pre-compiled binaries for the SPARC and Ultrix platforms, however this release is a source distribution only. The source code is available in tar format over the Internet via anonymous ftp from the site s2k-ftp.CS.Berkeley.EDU. Look in the directory `pub/postgres/postgres-v4r2`. The compressed tar file containing the complete source distribution is named `postgres-v4r2.tar.Z`.

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 recommend 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 System V 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. 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.

2.5.3. Kernel

POSTGRES makes use of the System V inter-process communication (shared memory and semaphore) operations provided by the operating system. SunOS and Ultrix 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

To install the system you must load the sources into your filesystem and build the system from scratch. To run the compiled programs your kernel may need to be configured to support the shared memory and semaphore operations required by the code. Assuming you have run POSTGRES in the past, your machine may already be properly configured.

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

First you should locate a disk partition with at least 50 megabytes of free space available for POSTGRES. 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...

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:

\# **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. You can make this work without creating a "postgres" login but this is highly discouraged — all POSTGRES processes run with this user id, and making yourself the POSTGRES user essentially grants all of your database users the ability to become *you* without a password.

Note that the restriction about having a "postgres" login with UID 6 was removed in POSTGRES Version 4.0.1.

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. 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.

Run `tar` with the following options

% **tar xvfp** *<tape-device>*

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

The file `postgres-v4r2.tar.Z` will appear in your POSTGRES home directory. You may need to rewind 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.

Uncompress the tar file.

% **uncompress postgres-v4r2.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-v4r2.tar`.

Extract POSTGRES from the tar file using the following command:

% **tar xvfp postgres-v4r2.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 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-v4r2.tar**

3.5. Step 3 - Kernel 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 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 manual pages for `shmget(3)`, `shmop(3)`, `shmctl(3)`, etc. Now proceed to the appropriate section for your machine.

3.5.1. Kernel reconfiguration for Sparcs

We previously suggested that you reconfigure your kernel under SunOS. However, the parameters in the generic kernel of current releases of SunOS ought to suffice. (We never reconfigure our kernels and have not noticed any problems while running several `postmasters` at once.)

3.5.2. Kernel reconfiguration for DECs

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

Remember that Ultrix 4.3 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.

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 (saving a copy of the old kernel first!) and reboot.

3.6. Compiling and Installing POSTGRES

The sources for POSTGRES are all under the directory `src/`.

3.6.1. Build Step 1: Customization

`Cd` into the `src/` directory and edit the file `Makefile.global`. You may change the various configuration options here, such as where the POSTGRES executable files are installed and where it looks for the database directory. The configuration switches are fairly self-explanatory, but we will go over some of the more commonly-changed options.

The PORTNAME option **must** be set correctly. By default the PORTNAME is `ultrix4`, but

aix
alpha
hpux
sparc
ultrix4

are all valid choices.

The top-level directory where POSTGRES binaries, documentation, header files, libraries, and databases are installed is controlled by the variable POSTGRESDIR. This variable defaults to `/usr/local/postgres`. Whether you change this variable or not, make sure that the directory to which POSTGRESDIR refers exists before proceeding.

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 that these directories exist before proceeding.

Standards notwithstanding, every system has an `install` program with slightly different options and behavior. Our Makefiles assume that `install` accepts BSD (as opposed to System V) options. For example, the `install` that comes with SunOS and Ultrix accept BSD options. However, the default OSF/1, HP-UX and AIX installation programs do not. You can set the INSTALL variable to control which program is called. On OSF/1, set INSTALL to `/bin/installbsd`. On AIX, use `/usr/ucb/install` instead of `/bin/install`. On HP-UX, you will have to find a BSD-compatible installation program and set INSTALL to the location of this program. `bsdinst`, which comes with the MIT X Window System distribution (in `mit/util/scripts`), is widely available (we include it in the `src/tools` directory) and works acceptably. The GNU `install` program does not.

If you are not installing POSTGRES using superuser privileges, or you are installing POSTGRES without creating a "postgres" user, you may have to set the values of several variables that control the system ownership. For example, "janeuser" who is a member of group "users" may have to set:

```
BINGRP=        users
BINOWN=        janeuser
LIBGRP=        users
LIBOWN=        janeuser
MANGRP=        users
MANOWN=        janeuser
```

to prevent the `install` program from complaining.

If you absolutely insist on having a POSTGRES user other than "postgres", set the variable POSTGRESLOGIN to the name of that user.

3.6.2. On HP-UX only: installing bsdinst

We told you about this above, but in case you missed it, you will have to install `bsdinst` from `src/tools/bsdinst` if you don't have it installed already. You can just copy the shell script `bsdinst.sh` into some convenient place (probably `/usr/local/bin`) as `bsdinst`.

3.6.3. Build Step 2: Bootstrapping "bmake"

Because the `bmake` program is normally installed into `/usr/local/{bin,lib}` and that directory is usually only writable by root, it will be necessary to temporarily become root to install `bmake`. If you don't have root privilege you can change where `bmake` is installed by changing the value of the TOOLSBINDIR and TOOLSLIBDIR options in `src/Makefile.global`, otherwise you will need to `su`(1) to root. Next make sure that the `make` program in your PATH environment variable is the system-provided `make` (usually `/bin/make`) and **NOT** GNU `make` (many users have reported problems getting old versions of GNU `make` to work).

`Cd` into `src/tools/bmake` and type

% **./Bootstrap** *portname*

where *portname* is the value of PORTNAME (ultrix4, sparc, etc). 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`.

3.6.4. On Ultrix only: building and installing libdl

This version of POSTGRES uses the `dlopen(3)` interface to implement dynamic function loading. Since there is no such vendor-supplied interface on the Ultrix platform you must build and install a version that we provide.

The same previous discussion about being root also holds true here. However, if you cannot become root to install this library, then you will have to change the backend Makefile to look for this library where you have decided to put it using the -L loader flag.

To compile and install the library simply `cd` into `src/tools/libdl` and type

% **bmake all install**

This will build and install the `libdl.a` library into TOOLSLIBDIR (this is `/usr/local/lib` by default). If your linker/loader doesn't search here by default (it usually does), you may have to `ranlib(1)` on it.

3.6.5. On AIX only: installing mkldexport

`Cd` into `src/tools/mkldexport` and type

% **bmake all install**

3.6.6. Build Step 3: Compile and Install

`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 into 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`, you could type

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

and if you run `ksh` or `sh`, 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 doughnuts and coffee.

3.7. 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 the POSTGRES programs were loaded into `/usr/local/postgres/bin`). If you run `csh`, you would put

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

in the `.login` for the POSTGRES user. 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.

3.8. Step 4 - Testing

We suggest you run the regression tests to make sure the release was installed successfully. Also, you need to have the `postmaster` running to run the test, so type the following:

% **TZ=GMT0**
% **export TZ**
% **postmaster &**

if you use `sh` or `ksh`. If you use `csh`, type

% **setenv TZ GMT0**
% **postmaster &**

instead. The bit about "TZ" is just to make sure that your regression output will be comparable to ours; you don't have to do this every time you start the `postmaster`. Next, create a new user using the `createuser` command (see the Reference Manual). **Do not** run the regression test script as user "postgres"!

Change directories to `src/regress/regress`.

% **cd src/regress/regress**

Make sure that the `obj` subdirectory is writable by the "postgres" user, as some of the tests involve the "postgres" user copying data into that directory. (We ship it that way, but if you left the "p" flag off of your `tar` command, it will be set according to your umask.) Type the following command to run the test:

% **bmake clean all 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:

% **sh checkdiff**

It may take a little while to run as the regression results are quite large. There will be many differences corresponding to the timestamp lines and occasional lines where object id numbers (OIDs) are different. The checkdiff program attempts to weed out these innocuous differences. If you see any differences not corresponding to timestamps 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,
- a front-end program (often the terminal monitor), and
- the backend server.

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 a front-end program such as the terminal monitor and a POSTGRES backend. Without a running `postmaster`, the front-end program 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 &**

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 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
%
```

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 it 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 short: never do this if there is a `postmaster` running. In addition, when using the terminal monitor, returned tuples are displayed more usefully and input is buffered better. The backend should only be used interactively 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 database 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
reindexdb	– rebuild system catalog indices after disaster
shmemdoc	– shared memory buffer pool doctor
vacuum	– database vacuum cleaner
icopy	– inversion filesystem file management utility
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. 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/local/lib` or `/usr/lib` so that the C compiler can reference it with `-lpq`. If you do not, you will have to use the `-L` directive to the `cc` and `ld` commands so that they can find `libpq.a`.

5.2. Postgres header files

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

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

Occassionally a front-end program source might reference header files from the postgres source that were not copied into the include directory. If your front-end program complains 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 `#defines` in your source to set the PORTNAME — add them prior to any `#includes`:

`#define PORTNAME_`*PORTNAME*

For example, if you are running on the ultrix4 port, you would put

```
#define PORTNAME_ultrix4
```

in your program, or if you're running the sparc port put

```
#define PORTNAME_sparc
```

in your program source.

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, `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. Sometimes however, as stated earlier, 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. Should you really wish to remove the source to reclaim space you could always copy all the header files from the backend into the include directory (preserving the directory structure of course).

6. Documentation

Plain text and PostScript versions of the manual pages 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 `Makefile` to use other facilities if you have them. If you change directories to `man/` and `doc/` and type:

% **bmake**
% **bmake install**

in each, it will format and install the documents into the corresponding destination directories. In general, recreating documents from their source is very difficult due to differences in macro packages and formatting programs (in short, good luck!).

7. Miscellaneous

7.1. Bug reports

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

```
bug-postgres@postgres.Berkeley.EDU
```

or

```
uunet!ucbvax!postgres!bug-postgres
```

describing as precisely as possible the command that caused the problem, concise instructions on how to repeat the bug, and a script showing the bug. If possible, a stack trace (generated using a debugger such as `dbx` or `gdb`) should also be provided. (The backend program will leave its core dumps in the directory PGDATA`/base/your_database`, where "your_database" is the name of your database.)

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 provide support in a research environment.

If you do want to talk directly to the POSTGRES group, electronic mail is the only method. We can be reached via the Internet as

```
post_questions@postgres.Berkeley.EDU
```

or

```
uunet!ucbvax!postgres!post_questions
```

Please be aware that this was the last release of POSTGRES from the University of California, and for all practical purposes there is no longer a POSTGRES group. However some of us may still be reading mail for a while and it is likely that some correspondence will occur.

7.3. Postgres Mailing List

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
```

or

```
uunet!ucbvax!postgres!postgres-request
```

with `ADD` as the subject. Note that mail sent to this address is processed **electronically**. (Deletion requests are handled by sending mail to the same address with subject `DEL`.)

The mailing list itself is called

```
postgres@postgres.Berkeley.EDU
```

or

```
uunet!ucbvax!postgres!postgres
```

and all mail sent to this address will be will be routed to the mailing list membership. As of the time of this release, this mailing list was being distributed to over 600 sites around the world, so please do **NOT** send administrative requests to this address. If you have any problems with the mailing list, send mail to `post_questions`.