POSTGRES INSTALLATION INSTRUCTIONS

C-Only Release 3.0

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 it obtain an educational license to the derived work. 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 3/xx family of processors with SunOS 3.4, or 3.5, and 4.0, and Sparc architecture machines (Sparcstation and Sun 4) running SunOS 4.0 and higher. Postgres is also supported on DECstations 3100's and 5000's running Ultrix 4.0 and higher. Tested but unsupported ports for DECstation Ultrix lower than 4.0 and Sequent Symmetry machines running Dynix are included. These ports are unsupported for the following reasons: the old Ultrix dynamic loader is quite buggy, and the standard *make* on the Sequent Symmetry cannot compile Postgres. 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 DECstation version requires a kernel which allows 4 megabytes of shared memory. To compile the Sequent Symmetry port, you will have to obtain a System V Release 3 compatible *make.* Consult your Sequent representative for details.

2.2. Software

This implementation of POSTGRES is completely in C. The distribution contains no Lisp or C++ code.

2.3. Distribution tape

These instructions assume you have a POSTGRES Version 3 distribution tape (in either 9 track, SCSI cartridge, or TK50 cartridge format) or a POSTGRES tar file.

2.4. Expertise

Once a site is properly configured and POSTGRES is properly 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 the various system administration procedures. Also note that various steps require superuser authority 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.

One exception to this rule is that we use Sun's SysV-compatible *make* to build the system. This is the version of *make* that is installed in both the BSD and SysV environments on Suns, so this should pose no problems on these platforms. We have no problems on DECstations either. However, we have had a great deal of difficulty on Sequent Symmetry machines, so we advise that you obtain a Sun-compatible make for these platforms.

2.5.2. Disk Partition

POSTGRES requires 45 megabytes of disk space, preferably on a single partition. If you don't have enough space, it is still possible to compile and run POSTGRES but you will have to modify the installation scripts.

2.5.3. Kernel

POSTGRES makes use of the optional System V shared memory operations provided by SunOS, DEC Ultrix, and Dynix, 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

POSTGRES installation consists of the following steps:

- Preparation
- Loading
- Configuration
- Compilation
- Testing

Each of these steps is described below. It is advised that you read over each of these steps carefully before beginning the installation.

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 authority. 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 45 megabytes of free space available for POSTGRES. If you haven't any single partition with 25 megabytes free, you might have to spread apart the POSTGRES directories across several partitions, and glue them together with symbolic links.

3.1.2. Creating the POSTGRES directory

Once you have located a partition with enough space, create a directory called ‘‘postgres’’ someplace on this partition. Then **cd** to this directory and type **pwd**. This is the full path of the directory you will install postgres in. Write it down in preparation for the next step. 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%     /public
/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
            /public looks like a good place (it has 60 megs free) so we decide
```

to create the postgres directory there...

```
# cd /public
# mkdir postgres
# cd postgres
# pwd
/public/postgres
#
```

3.2. Creating /usr/postgres

POSTGRES expects to be *logically* installed in a directory called ''/usr/postgres'', so you must create a symbolic link from /usr/postgres to whatever directory you created in the previous step. In our example, we would now type:

```
# ln -s  /public/postgres  /usr/postgres
```

3.3. Creating the ''postgres'' user

Finally, we need to create a user called ''postgres'' whose shell is /bin/csh and whose home directory is /usr/postgres. This can be done using the "adduser" procedures particular to your platform and site. See your system administration manual for details.

Note:

Due to a bug in this release, the "postgres" user must be user 6 (six). Otherwise, you may encounter problems with backends hanging, etc. See the **Release Notes** (described in Section 6.2 of this document) for instructions on how to get around this problem if it causes problems at your site. If it is not convienent for you to make the "postgres" user userid 6, complete the below instructions on **Loading** POSTGRES, but read the **Release Notes** notes on how to get around this problem **before** continuing on to the **Configuration** section.

3.4. Step 2 - Loading POSTGRES

After completing step 1 (Preparation), you should be 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".

3.4.1. Loading POSTGRES from a Tape

Login as postgres.

3. Run "tar" with the "extract, verbose, file" 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-v3r0.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.

Please proceed to the section "Loading POSTGRES from a Tar File".

3.5. Loading POSTGRES from a Tar File

If you are not logged in as POSTGRES already, log in as POSTGRES. Make sure your current working directory is the POSTGRES home directory, and that the POSTGRES tar file is there. For the purpose of this discussion, the POSTGRES tar file will be called

postgres-v3r0.tar.Z

Uncompress the tar file.

```
% uncompress postgres-v3r0.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-v3r0.tar
```

Extract POSTGRES from the tar file, using the "extract, verbose, file" options:

```
% tar xvf postgres-v3r0.tar
```

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

Now do an "ls":

The output of the ls should look something like:

```
COPYRIGHT    bench/   demo/   newconf/            src/
README       doc/     ref/    test/     sample/   video/
```

At this point you have loaded the POSTGRES files. Other directories will be created by the installation process.

3.6. 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 authority 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.

3.6.1. Kernel reconfiguration for Suns and 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.

3.6.2. 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.

3.6.3. Kernel reconfiguration for Sequent Symmetry machines

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

The following line should be added to /sys/conf/KERNEL:

```
options         SVSEMA
```

After adding this line, run config over the config file, install the new kernel, and reboot.

3.7. Configuring POSTGRES

This release of POSTGRES may require some configuration. For performance reasons, Postgres is by default compiled with the optimizer enabled and internal debugging assertions disabled. If you plan to modify Postgres, you may want to enable debugging (note that this will take Postgres up to about 50 megs from about 45 megs otherwise), and enable internal debugging assertions.

To enable compiler directives, read the file ./newconf/CONFIG/README for instructions on what to change. Now to edit the configuration file,

```
% cd /usr/postgres/newconf/CONFIG

% vi config.mk.<port>
```

where *<port>*is

dec – DS3100 running Ultrix LOWER than 4.0
ultrix4 – DS3100, 5000, 5500, etc. running Ultrix 4.0 or higher
sun – Sun 3 running SunOS 3.4 or 3.5
sunos4 – Sun 3 running SunOS 4.0 or higher
seq – Sequent Symmetry running Dynix
sparc – Sparcstation or Sun 4

The *only* thing we recommend changing is the GCFLAGS variable. Remember the **port name** used here as it is necessary for Step 4.

3.8. Step 4 - Compiling and Installing POSTGRES Now you are ready to install Postgres. To do so, simply execute the following commands:

```
% cd ˜postgres/newconf
% setenv POSTGRESHOME ˜postgres
% ./Make install
```

Make install will ask you for the port you wish to use. Use the port name that you used in **Configuring** POSTGRES. You will also be asked for the name of the object tree directory; the default is ˜postgres/obj.<port>. (throughout the rest of this document obj.<port> refers to the object tree directory). This step will take from about 40 minutes on Sparc II or DEC 5000 class machines to several hours on Sun 3's. The POSTGRESHOME environment variable is the home directory of the Postgres user. In the course of the installation process, the Postgres **bin** and **data** directories will be created and populated.

Make is a C shell script that runs *make* with Makefiles that are constructed on the fly. If you have problems at this point, it is possible that your *.cshrc* file does strange things – changes directories, sets or unsets environment variables, and so on.

You should see no errors during this phase, except possibly for warnings (which can be ignored) when compiling the output of *yacc* and *lex.*

3.8.1. 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. To do this, type

```
% setenv POSTGRESHOME ~postgres
% ~postgres/bin/postmaster &
% ~postgres/bin/createdb postgres
% kill %~postgres/bin/postmaster
```

This will create the bootstrap template database, from which the database ‘‘postgres’’ will be generated. The *postmaster* program will be discussed later - however, you must have it running in order to run *createdb*. If several users wish to use POSTGRES, we advise you to create additional databases, one for each user. This can be done by running *createdb* with the username as the first argument. For example, to create a database for the user ‘‘bill’’, type

```
% ~postgres/bin/createdb bill
```

3.9. Step 4 - Testing

3.9.1. Testing POSTGRES

After compiling the POSTGRES backend and support programs and creating the initial database, you should test your compilation with the following. Commands you should type appear in **boldface**.

```
% ~postgres/bin/postgres
      ---debug info---
      Quiet =        f
      Noversion =    f
      override  =    f
      DatabaseName = [postgres]
      ----------------

      **** Transaction System Active ****
      InitPostgres()..

POSTGRES backend interactive interface
$Revision: 1.22 $ $Date: 91/12/02 04:15:58 $
      StartTransactionCommand() at Thu Nov  2 15:43:35 1989
> retrieve (pg_user.all)
 now in make_Var
relation = pg_user, attr = usecatupd
vnum = 1
      ...
      lots of debugging output...

----  parser outputs :
((1 retrieve nil (("pg_user" 86 0 nil nil ))0 nil )((#S(resdom :resno 1
:restype 19 :reslen 16 :resname "usename" :reskey 0 :reskeyop 0)#S(var
      ...
      lots more debugging output...

      ProcessQuery() at Thu Nov  2 15:43:50 1989

blank
       1: usename (typeid = 19, len = 16, byval = f)
       2: usesysid      (typeid = 21, len = 2, byval = t)
       3: usecreatedb   (typeid = 16, len = 1, byval = t)
       4: usetrace      (typeid = 16, len = 1, byval = t)
       5: usesuper      (typeid = 16, len = 1, byval = t)
       6: usecatupd     (typeid = 16, len = 1, byval = t)
      ----
       1: usename = "postgres"      (typeid = 19, len = 16, byval = f)
       2: usesysid = "6"      (typeid = 21, len = 2, byval = t)
```

```
 3: usecreatedb = "t"   (typeid = 16, len = 1, byval = t)
 4: usetrace = "t"      (typeid = 16, len = 1, byval = t)
 5: usesuper = "t"      (typeid = 16, len = 1, byval = t)
 6: usecatupd = "t"     (typeid = 16, len = 1, byval = t)
----
 1: usename = "goh"     (typeid = 19, len = 16, byval = f)
 2: usesysid = "234"    (typeid = 21, len = 2, byval = t)
 3: usecreatedb = "t"   (typeid = 16, len = 1, byval = t)
 4: usetrace = "t"      (typeid = 16, len = 1, byval = t)
 5: usesuper = "t"      (typeid = 16, len = 1, byval = t)
 6: usecatupd = "t"     (typeid = 16, len = 1, byval = t)
----
 ...

CommitTransactionCommand() at Thu Nov  2 15:43:51 1989

StartTransactionCommand() at Thu Nov  2 15:43:51 1989
```

It works!

The above response is an example of the raw output
generated by the backend. Your actual output may be slightly
different. Normally, you would use a
terminal monitor to talk to the backend instead.
To leave the backend, type <ctrl-D>:

```
> ^D
%
```

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. The terminal monitor sends commands to the postmaster which forwards commands to a backend. If you just completed step 3, then you have already been introduced to the POSTGRES backend, so we'll talk about the other two processes now.

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. To start the postmaster, type:

```
% cd ~postgres/bin
% setenv POSTGRESHOME ~postgres
% postmaster &
```

Here we are using the default parameters for the postmaster. For more details, consult the Reference.

4.2. The POSTGRES Terminal Monitor

The POSTGRES terminal monitor is a front-end user interface to the POSTGRES backend. To start a terminal monitor, type

```
% monitor <database>
```

Database is the name of the database you want to use. Now we will run the monitor:

```
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         |
 -----------------------------
 | 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/bin/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 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
createdb	– creates new postgres databases
createdb.sh	– creates new postgres databases - old version
destroydb	– destroys postgres databases
ipcclean	– frees up garbage shared memory from failed backends
pg_version	– make version numbers for createdb
pg_id	– gets user id’s for createdb

5. Optional Installation

5.1. Installing LIBPQ, the POSTGRES frontend library

The file ˜postgres/obj.<port>/libpq.a is created when you compile 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/lib so that the C compiler can reference it with -lpq. To do this, type:

```
# cp ˜postgres/obj.<port>/libpq.a   /usr/lib
```

5.2. Demo Database

In ˜postgres/demo are files to be included by the terminal monitor to set up a demo database. Additional files demonstrate inheritance, historical queries, abstract data types, and various other features of POSTGRES. A description of the demo database can be found in ˜postgres/demo/DEMO-README.

5.3. Video Demo Database

In ˜postgres/video are files that were used in the 1991 Postgres SIGMOD video. These files demonstrate both the instance level and query rewrite rule systems, views, versions, and spatial queries using R-Tree indices. A description of the video database can be found in ˜postgres/video/VIDEO-README.

5.4. Wisconsin Benchmark Database

In ˜postgres/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. Instructions for running the benchmark are in ˜postgres/bench/WISC-README.

5.5. Minimal Installation

The directories (in ˜postgres) necessary for a minimal running system are:

bin/	the binary programs comprising POSTGRES
data/	support files and user created databases
files/	database initialization scripts

When compiled using the "default" compilation options as shipped, (ie optimization and no debugging), these directories will take up about 5 Mbytes. The following directories are necessary if Postgres is to ever be recompiled.

newconf/	the POSTGRES configuration directory
obj.<port>/	compiled POSTGRES object files
src/	POSTGRES source files

When compiled using the "defaults", these directories will use about 16 Mbytes. Additional Postgres directories are as follows:

demo/	demo database scripts
video/	video demo database scripts
bench/	Wisconsin benchmark database scripts
sample/	Sample LIBPQ application
doc/	postgres technical reports and the POSTGRES Manual
ref/	POSTGRES Reference source

These directories take up about 2 Mbytes, and can be reduced to about 200 Kbytes if the Postscript files in doc and ref are deleted.

We do not recommend deleting these unless absolutely necessary.

6. Documentation

6.1. Printing the Manual

The POSTGRES manual is now in the file

~postgres/doc/manual.me

This manual replaces the old tutorials, which are no longer distributed. It is recommended that you read this manual before making extensive use of POSTGRES. A Postscript version of this manual is in

~postgres/doc/psdump/manual.ps

6.2. Printing the Reference

The Reference is the document which details the exact syntax used by POSTGRES commands, and provides interface definitions for LIBPQ and large objects. It is intended as a reference and should not be read cover to cover.

If you have a Postscript printer, you can print the Reference by changing directory to the

~postgres/ref

directory, where you will find a Postscript file called **ref.t**. This file can be simply sent to the printer in whatever fashion your site uses to print Postscript files.

If you do not have a Postscript printer, or you have font problems, etc., the **ref** directory contains a /bin/sh script called "genref". Edit this script and set the appropriate parameters for printing at your site in this script, and then execute it by typing

```
% genref
```

This script will not actually try to send the job to the printer - rather it will create the **ref.t** output file. You can then print the manual by sending this file directly to the printer with the regular printing commands used by your site.

If **genref** fails, you may not have the **grn** preprocessor. This preprocessor for troff allows pictures drawn with the **gremlin** graphics editor to be printed using a "troff" command. A script called **eatgrn** is provided, which will cause **genref** to ignore **grn** directives and print anyway - this will result in the reference being printed without illustrations. There is only one illustration, so this should not be too much of a problem.

6.2.1. If you do not have a Postscript printer

If you do not have a Postscript printer, change the **psroff** command in the **genref** script to the text formatting command you use at your site, typically **nroff**, **troff**, or **ditroff**. As stated above, use the **eatgrn** script if you do not have **grn**. Output suitable for a line-printer can be created using **nroff**.

6.3. Printing the Technical Reports and Tutorials

Postscript versions of the Postgres technical reports, tutorials, and release notes are in the directory **~postgres/doc/psdump**. Some files are not included in Postscript form and are simply regular files - read the file **~postgres/doc/README** for details.

The /bin/sh script **~postgres/doc/print** contains site-specific printing parameters, and understands the file extension protocol used in the **doc** directory. Once you set the site-specific parameters for printing in this script, you should be able to print all the files in the **doc** easily, by executing

```
% print <filename>
```

from the **~postgres/doc** directory.

Unlike the **genref** script described above, this script will send the job directly to the printer. If you do not have the **grn** utility described above, you should use the **eatgrn** script here as well. For technical reports which require **make**, continue to the following section.

6.3.1. If the directory has a makefile

A couple of the technical reports use makefiles to generate their printable versions rather than the **print** script. If the subdirectory has a makefile, you will have to change the site-specific parameters in the makefile, run

```
% make
```

and then it will either print or create a printable file. Note that if the makefile uses **grn** and you do not have access to this utility, you can use the **eatgrn** script here as well.

6.4. The 3.0 Release Notes

The Postgres 3.0 Release Notes are in the file **~postgres/doc/release3.0.me**. Before working extensively with Postgres, you should read this file to find new features, known bugs, and other useful information about this release.

As described above, you can print this file by typing

```
% print ~postgres/doc/release3.0.me
```

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. Known Bugs List

A Known Bugs List with suggested workarounds is maintained on the machine **postgres.berkeley.edu** in the file **pub/postgres-v3r0.bugs**. The Internet address of this machine is **128.32.149.1**, and if you cannot access Postgres, this file can be sent to you via e-mail.

7.3. 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 (415) 642-7520, Monday through Friday, between 1 and 4 PM Pacific Time.

7.4. 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.