Appendix A:
Installation Guide for the UNIX Versions

1. Required tools.

Compiling PARI requires an `ANSI C` or a `C++` compiler. If you do not have one, we suggest that you obtain the `gcc/g++` compiler. As for all GNU software mentioned afterwards, you can find the most convenient site to fetch `gcc` at the address

    http://www.gnu.org/order/ftp.html

(On Mac OS X, this is also provided in the `Xcode` tool suite.) You can certainly compile PARI with a different compiler, but the PARI kernel takes advantage of optimizations provided by `gcc`. This results in at least 20% speedup on most architectures.

Optional packages. The following programs and libraries are useful in conjunction with `gp`, but not mandatory. In any case, get them before proceeding if you want the functionalities they provide. All of them are free.

• GNU `MP` library. This provides an alternative multiprecision kernel, which is faster than PARI's native one, but unfortunately binary incompatible. To enable detection of GMP, use `Configure --with-gmp`. You should really do this if you only intend to use GP, and probably also if you will use libpari unless you have backwards compatibility requirements.

• GNU `readline` library. This provides line editing under GP, an automatic context-dependent completion, and an editable history of commands. Note that it is incompatible with SUN commandtools (yet another reason to dump Suntools for X Windows).

• GNU `gzip/gunzip/gzcat` package enables GP to read compressed data.

• GNU `emacs`. GP can be run in an Emacs buffer, with all the obvious advantages if you are familiar with this editor. Note that `readline` is still useful in this case since it provides a better automatic completion than is provided by Emacs GP-mode.

• `perl` provides extended online help (full text from this manual) about functions and concepts, which can be used under GP or independently (`http://www.perl.com` will direct you to the nearest `CPAN` archive site).

• A colour-capable `xterm`, which enables GP to use different (user configurable) colours for its output. All `xterm` programs which come with current X11 distributions satisfy this requirement.

2. Compiling the library and the GP calculator.

2.1. Basic configuration: First, have a look at the `MACHINES` file to see if anything funny applies to your architecture or operating system. Then, type

    ./Configure

in the toplevel directory. This attempts to configure PARI/GP without outside help. Note that if you want to install the end product in some nonstandard place, you can use the `--prefix` option, as in

    ./Configure --prefix=/an/exotic/directory

(the default prefix is `/usr/local`). For example, to build a package for a Linux distribution, you may want to use

    ./Configure --prefix=/usr

This phase extracts some files and creates a directory $Oxxx$ where the object files and executables will be built. The xxx part depends on your architecture and operating system, thus you can build GP for several different machines from the same source tree (the builds are independent and can be done simultaneously).

Technical note: `Configure` accepts many other flags besides `--prefix`. See `Configure --help` for a complete list. In particular, there are sets of flags related to GNU MP (`--with-gmp*`) and GNU readline library (`--with-readline*`). Note that autodetection of GMP is *disabled* by default, you probably want to enable it.

Decide whether you agree with what `Configure` printed on your screen, in particular the architecture, compiler and optimization flags. Look for messages prepended by `###`, which report genuine problems. If anything should have been found and was not, consider that `Configure` failed and follow the instructions in the next section. Look especially for the `gmp`, `readline` and `X11` libraries, and the `perl` and `gunzip` (or `zcat`) binaries.

2.2. Compilation: To compile the GP binary and build the documentation, type

    make all

To only compile the GP binary, type

    make gp

in the toplevel directory. If your `make` program supports parallel make, you can speed up the process by going to the $Oxxx$ directory that `Configure` created and doing a parallel make here, for instance `make -j4` with GNU make. It should even work from the toplevel directory.

2.3. Basic tests:

To test the binary, type `make bench`. This will build a static executable (the default, built by `make gp` is probably dynamic) and run a series of comparative tests on those two. To test only the default binary, use `make dobench` which starts the bench immediately.

The static binary should be slightly faster. In any case, this should not take more than a few seconds on modern machines. See the file `MACHINES` to get an idea of how much time comparable systems need. We would appreciate a short note in the same format in case your system is not listed and you nevertheless have a working GP executable.

If a `[BUG]` message shows up, something went wrong. The testing utility directs you to files containing the differences between the test output and the expected results. Have a look and decide for yourself if something is amiss. If it looks like a bug in the Pari system, we would appreciate a report (see the last section).

3. Troubleshooting and fine tuning.

In case the default `Configure` run fails miserably, try

```
./Configure -a
```

(interactive mode) and answer all the questions (there are not that many). Of course, `Configure` still provides defaults for each answer but if you accept them all, it will fail just the same, so be wary. In any case, we would appreciate a bug report (see the last section).

3.1. Installation directories: The precise default destinations are as follows: the `gp` binary, the scripts `gphelp` and `tex2mail` go to `$prefix/bin`. The pari library goes to `$prefix/lib` and include files to `$prefix/include/pari`. Other system-dependant data go to `$prefix/lib/pari`.

Architecture independent files go to various subdirectories of `$share_prefix`, which defaults to `$prefix/share`, and can be specified via the `--share-prefix` argument. Man pages go into `$share_prefix/man`, Emacs files into `$share_prefix/emacs/site-lisp/pari`, and other system-independant data to various subdirectories of `$share_prefix/pari`: documentation, sample GP scripts and C code, extra packages like `elldata` or `galdata`.

You can also set directly `--bindir` (executables), `--libdir` (library), `--includedir` (include files), `--mandir` (manual pages), `--datadir` (other architecture-independent data), and finally `--sysdatadir` (other architecture-dependent data).

3.2. Environment variables: `Configure` lets the following environment variable override the defaults if set:

`AS`: Assembler.

`CC`: C compiler.

`DLLD`: Dynamic library linker.

`LD`: Static linker.

For instance, `Configure` may avoid `/bin/cc` on some architectures due to various problems which may have been fixed in your version of the compiler. You can try

```
env CC=cc Configure
```

and compare the benches. Also, if you insist on using a `C++` compiler and run into trouble with a fussy g++, try to use `g++ -fpermissive`.

The contents of the following variables are *appended* to the values computed by `Configure`:

CFLAGS: Flags for `CC`.

CPPFLAGS: Flags for `CC` (preprocessor).

LDFLAGS: Flags for `LD`.

The contents of the following variables are *prepended* to the values computed by `Configure`:

C_INCLUDE_PATH is prepended to the list of directories searched for include files. Note that adding `-I` flags to CFLAGS is not enough since `Configure` sometimes relies on finding the include files and parsing them, and it does not parse CFLAGS at this time.

LIBRARY_PATH is prepended to the list of directories searched for libraries.

You may disable inlining by adding `-DDISABLE_INLINE` to CFLAGS, and prevent the use of the `volatile` keyword with `-DDISABLE_VOLATILE`.

3.3. Debugging/profiling: If you also want to debug the PARI library,

    Configure -g

creates a directory $0xxx$.dbg containing a special `Makefile` ensuring that the GP and PARI library built there is suitable for debugging. If you want to profile GP or the library, using `gprof` for instance,

    Configure -pg

will create an $0xxx$.prf directory where a suitable version of PARI can be built.

The GP binary built above with `make all` or `make gp` is optimized. If you have run `Configure` `-g` or `-pg` and want to build a special purpose binary, you can `cd` to the `.dbg` or `.prf` directory and type `make gp` there. You can also invoke `make gp.dbg` or `make gp.prf` directly from the toplevel.

3.4. Multiprecision kernel: The kernel can be fully specified via the `--kernel=`*fqkn* switch. The PARI kernel is build from two kernels, called level 0 (L0, operation on words) and level 1 (L1, operation on multi-precision integer and real).

Available kernels:

L0: `auto`, `none` and

    alpha hppa hppa64 ia64 ix86 x86_64 m68k ppc sparcv7
    sparcv8_micro sparcv8_super

L1: `auto`, `none` and `gmp`

`auto` means to use the auto-detected value. L0=`none` means to use the portable C kernel (no assembler), L1=`none` means to use the PARI L1 kernel.

• A fully qualified kernel name *fqkn* is of the form L_0-L_1.

• A *name* not containing a dash '-' is an alias. An alias stands for *name*-`none`, but `gmp` stand for `auto-gmp`.

• The default kernel is `auto-none`.

4

3.5. Problems related to readline: `Configure` does not try very hard to find the `readline` library and include files. If they are not in a standard place, it will not find them. Nonetheless, it first searches the distribution toplevel for a `readline` directory. Thus, if you just want to give `readline` a try, as you probably should, you can get the source and compile it there (you do not need to install it). You can also use this feature together with a symbolic link, named `readline`, in the PARI toplevel directory if you have compiled the readline library somewhere else, without installing it to one of its standard locations.

You can also invoke `Configure` with one of the following arguments:

`--with-readline[=`*prefix to* `lib/libreadline.`*xx and* `include/readline.h]`

`--with-readline-lib=`*path to* `libreadline.`*xx*

`--with-readline-include=`*path to* `readline.h`

Technical note: Configure can build GP on different architectures simultaneously from the same toplevel sources. Instead of the `readline` link alluded above, you can create `readline-`*osname-arch*, using the same naming conventions as for the $Oxxx$ directory, e.g `readline-linux-i686`.

Known problems:

• on Linux: Linux distributions have separate `readline` and `readline-devel` packages. You need both of them installed to compile gp with readline support. If only `readline` is installed, `Configure` will complain. `Configure` may also complain about a missing libncurses.so, in which case, you have to install the `ncurses-devel` package (some distributions let you install `readline-devel` without `ncurses-devel`, which is a bug in their package dependency handling).

• on OS X.4: Tiger comes equipped with a fake `readline`, which is not sufficient for our purpose. As a result, gp is built without readline support. Since `readline` is not trivial to install in this environment, a step by step solution can be found in the PARI FAQ, see

> `http://pari.math.u-bordeaux.fr/`

3.6. Testing

3.6.1. Known problems: if `BUG` shows up in `make bench`

• `program`: the GP function `install` may not be available on your platform, triggering an error message ("not yet available for this architecture"). Have a look at the `MACHINES` files to check if your system is known not to support it, or has never been tested yet.

• If when running `gp-dyn`, you get a message of the form

`ld.so: warning: libpari.so.`*xxx* `has older revision than expected` *xxx*

(possibly followed by more errors), you already have a dynamic PARI library installed *and* a broken local configuration. Either remove the old library or unset the `LD_LIBRARY_PATH` environment variable. Try to disable this variable in any case if anything *very* wrong occurs with the `gp-dyn` binary, like an Illegal Instruction on startup. It does not affect `gp-sta`.

• Some implementations of the `diff` utility (on HPUX for instance) output `No differences encountered` or some similar message instead of the expected empty input. Thus producing a spurious `[BUG]` message.

3.6.2. Some more testing [*Optional*]

You can test GP in compatibility mode with `make test-compat`. If you want to test the graphic routines, use `make test-ploth`. You will have to click on the mouse button after seeing each image. There will be eight of them, probably shown twice (try to resize at least one of them as a further test). More generaly, typing `make` without argument will print the list of available extra tests among all available targets.

The `make bench` and `make test-compat` runs produce a Postscript file `pari.ps` in $0xxx$ which you can send to a Postscript printer. The output should bear some similarity to the screen images.

3.6.3. Heavy-duty testing [*Optional*] There are a few extra tests which should be useful only for developpers.

`make test-kernel` checks whether the low-level kernel seems to work, and provides simple diagnostics if it does not. Only useful if `make bench` fails horribly, e.g. things like `1+1` do not work.

`make test-all` runs all available test suites. Slow.

4. Installation.

When everything looks fine, type

`make install`

You may have to do this with superuser privileges, depending on the target directories. (Tip for MacOS X beginners: use `sudo make install`.) In this case, it is advised to type `make all` first to avoid running unnecessary commands as `root`.

Beware that, if you chose the same installation directory as before in the `Configure` process, this will wipe out any files from version 1.39.15 and below that might already be there. Libraries and executable files from newer versions (starting with version 1.900) are not removed since they are only links to files bearing the version number (beware of that as well: if you are an avid `gp` fan, do not forget to delete the old pari libraries once in a while).

This installs in the directories chosen at `Configure` time the default GP executable (probably `gp-dyn`) under the name `gp`, the default PARI library (probably `libpari.so`), the necessary include files, the manual pages, the documentation and help scripts and emacs macros.

To save on disk space, you can manually `gzip` some of the documentation files if you wish: `usersch*.tex` and all `dvi` files (assuming your `xdvi` knows how to deal with compressed files); the online-help system can handle it.

By default, if a dynamic library `libpari.so` could be built, the static library `libpari.a` will not be created. If you want it as well, you can use the target `make install-lib-sta`. You can install a statically linked `gp` with the target `make install-bin-sta`. As a rule, programs linked statically (with `libpari.a`) may be slightly faster (about 5% gain), but use more disk space and take more time to compile. They are also harder to upgrade: you will have to recompile them all instead of just installing the new dynamic library. On the other hand, there is no risk of breaking them by installing a new pari library.

4.1. Extra packages: The following optional packages endow PARI with some extra capabilities (only two packages for now!).

● `elldata`: This package contains the elliptic curves in John Cremona's database. It is needed by the functions `ellidentify`, `ellsearch` and can be used by `ellinit` to initialize a curve given by its standard code.

● `galdata`: The default `polgalois` function can only compute Galois groups of polynomials of degree less or equal to 7. Install this package if you want to handle polynomials of degree bigger than 7 (and less than 11).

To install package *pack*, you need to fetch the separate archive: *pack*`.tgz` which you can download from the `pari` server. Copy the archive in the PARI toplevel directory, then extract its contents; these will go to `data/`*pack*`/`. Typing `make install` installs all such packages.

4.2. The `GPRC` file: Copy the file `misc/gprc.dft` (or `gprc.dos` if you are using `GP.EXE`) to `$HOME/.gprc`. Modify it to your liking. For instance, if you are not using an ANSI terminal, remove control characters from the `prompt` variable. You can also enable colors.

If desired, read `$datadir/misc/gpalias` from the `gprc` file, which provides some common shortcuts to lengthy names; fix the path in gprc first. (Unless you tampered with this via Configure, `datadir` is `$prefix/share/pari`.) If you have superuser privileges and want to provide system-wide defaults, copy your customized `.gprc` file to `/etc/gprc`.

In older versions, `gphelp` was hidden in pari lib directory and was not meant to be used from the shell prompt, but not anymore. If gp complains it cannot find `gphelp`, check whether your `.gprc` (or the system-wide `gprc`) does contain explicit paths. If so, correct them according to the current `misc/gprc.dft`.

5. Getting Started.

5.1. Printable Documentation: Building gp with `make all` also builds its documentation. You can also type directly `make doc`. In any case, you need a working (plain) TEX installation.

After that, the `doc` directory contains various `dvi` files: `libpari.dvi` (manual for the PARI library), `users.dvi` (manual for the `gp` calculator), `tutorial.dvi` (a tutorial), and `refcard.dvi` (a reference card for GP). You can send these files to your favourite printer in the usual way, probably via `dvips`. The reference card is also provided as a `PostScript` document, which may be easier to print than its `dvi` equivalent (it is in Landscape orientation and assumes A4 paper size).

If the `pdftex` package is part of your TEX setup, you can produce these documents in PDF format, which may be more convenient for online browsing (the manual is complete with hyperlinks); type

```
make docpdf
```

All these documents are available online from PARI home page (see the last section).

5.2. C programming: Once all libraries and include files are installed, you can link your C programs to the PARI library. A sample makefile `examples/Makefile` is provided to illustrate the use of the various libraries. Type `make all` in the `examples` directory to see how they perform on the `extgcd.c` program, which is commented in the manual.

This should produce a statically linked binary `extgcd-sta` (standalone), a dynamically linked binary `extgcd-dyn` (loads libpari at runtime) and a shared library `libextgcd`, which can be used from `gp` to `install` your new `extgcd` command.

The standalone binary should be bulletproof, but the other two may fail for various reasons. If when running `extgcd-dyn`, you get a message of the form "DLL not found", then stick to statically linked binaries or look at your system documentation to see how to indicate at linking time where the required DLLs may be found! (E.g. on Windows, you will need to move `libpari.dll` somewhere in your `PATH`.)

5.3. GP scripts: Several complete sample GP programs are also given in the `examples` directory, for example Shanks's SQUFOF factoring method, the Pollard rho factoring method, the Lucas-Lehmer primality test for Mersenne numbers and a simple general class group and fundamental unit algorithm. See the file `examples/EXPLAIN` for some explanations.

5.4. EMACS: If you want to use `gp` under GNU Emacs, read the file `emacs/pariemacs.txt`. If you are familiar with Emacs, we suggest that you do so.

5.5. The PARI Community: PARI's home page at the address

> `http://pari.math.u-bordeaux.fr/`

maintains an archive of mailing lists dedicated to PARI, documentation (including Frequently Asked Questions), a download area and our Bug Tracking System (BTS). Bug reports should be submitted online to the BTS, which may be accessed from the navigation bar on the home page or directly at

> `http://pari.math.u-bordeaux.fr/Bugs`

Further information can be found at that address but, to report a configuration problem, make sure to include the relevant `*.dif` files in the `O`xxx directory and the file `pari.cfg`.

There are three mailing lists devoted to PARI/GP (run courtesy of Dan Bernstein), and most feedback should be directed to those. They are:

• `pari-announce`: to announce major version changes. You cannot write to this one, but you should probably subscribe.

• `pari-dev`: for everything related to the development of PARI, including suggestions, technical questions, bug reports or patch submissions. (The BTS forwards the mail it receives to this list.)

• `pari-users`: for everything else.

You may send an email to the last two without being subscribed. (You will have to confirm that your message is not unsollicited bulk email, aka *Spam*.) To subscribe, send empty messages respectively to

> `pari-announce-subscribe@list.cr.yp.to`
> `pari-users-subscribe@list.cr.yp.to`

```
pari-dev-subscribe@list.cr.yp.to
```

You can also write to us at the address

```
pari@math.u-bordeaux.fr
```

but we cannot promise you will get an individual answer.

If you have used PARI in the preparation of a paper, please cite it in the following form (BibTeX format):

```
@manual{PARI2,
    organization = "{The PARI~Group}",
    title        = "{PARI/GP, Version 2.3.5}",
    year         = 2006,
    address      = "Bordeaux",
    note         = "available from {\tt http://pari.math.u-bordeaux.fr/}"
}
```

In any case, if you like this software, we would be indebted if you could send us an email message giving us some information about yourself and what you use PARI for.

Good luck and enjoy!