Installation Guide for the UNIX Versions

1. Required tools.

Compiling PARI requires a `C` or `C++` compiler supporting C99 or higher. 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

`https://www.gnu.org/prep/ftp.html`

(On macOS, this is also provided in the `Xcode` tool suite; or the lightweight "Command-Line Tools (CLT) for `Xcode`".) You can compile PARI with a different compiler, but the PARI kernel takes advantage of optimizations provided by `gcc`. Compiling with GCC results in at least 20% speedup on most architectures.

Optional libraries and programs. 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 functionality they provide. All of libraries and programs are free. The download page on our website

`https://pari.math.u-bordeaux.fr/download.html`

contains pointers on how to get them.

• GNU `MP` (GMP) library. GMP provides an alternative multiprecision kernel, which is faster than PARI's native one. When using GMP the binaries become incompatible, so the resulting PARI library `SONAME` is libpari-gmp.

• GNU `readline` library. Readline provides line editing under `gp`, an automatic context-dependent completion, and an editable history of commands.

• GNU `emacs` and the `PariEmacs` package. The `gp` calculator 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's GP-mode.

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

• `perl` provides extended online help (full text from the manual) about functions and concepts. The script handling this online help can be used under `gp` or independently.

2. Compiling the library and the gp calculator.

2.1. Basic configuration. 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 a 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
```

The command

```
./Configure --builddir=DIR
```

configures the build for a particular system. It extracts some files and creates the *build directory* *DIR*, where the object files and executables will be built. Thus you can build PARI/GP for several different machines from the same source tree. When building for multiple machines the builds are independent and can be performed simultaneously.

If the `--builddir=`*DIR* argument is omitted, the build directory is named `O`*osname-arch*, where the *osname* and *arch* components depends on your architecture and operating system. A debug build (`Configure -g`) or a profiling build (`Configure -pg`) bring in an extra `.dbg` (resp. `.prf`) suffix.

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. Look especially for the `gmp`, `readline` and `X11` libraries, and the `perl` and `gunzip` (or `zcat`) binaries. If anything should have been found and was not, consider that `Configure` failed and follow the instructions in section 3.

The `Configure` script creates a file `config.log` in the build directory, which contains debugging information — in particular, all messages from compilers — that may help diagnose problems. This file is erased and recreated from scratch each time `Configure` is run.

2.2. Advanced configuration. `Configure` accepts many other flags, and you may use any number of them to build quite a complicated configuration command. See `Configure --help` for a complete list. In particular, GMP is configured using the set of flags `--with-gmp*`, and Readline is configured using the set of flags `--with-readline*`.

`--with-gmp`: enables a particular installation of GMP. If the GMP library was installed in `/an/exotic/directory`, then you can use `--with-gmp=/an/exotic/directory` to find the headers and libraries.

If you are using GMP, tune it first, then tune PARI. Make sure you tune PARI on the machine that will actually run your computations. Do not use a heavily loaded machine for tunings.

`--with-readline`: enables a particular installation of Readline. If Readline was installed in `/an/exotic/directory`, then you can use `--with-readline=/an/exotic/directory` to find the headers and libraries.

`make -jN`: You may speed up the compilation by using a parallel make. The command below tells make to execute 4 recipes simultaneously.

```
env MAKE="make -j4" Configure
```

Next, we focus on some of the nonobvious options used by `Configure`:

`--tune`: fine tunes the library for the host used for compilation. Tuning the package adjusts thresholds by running a large number of comparative tests and creates a file `tune.h` in the build directory, that will be used from now on, overriding the ones in `src/kernel/none/` and `src/kernel/gmp/`. It will take a while: about 30 minutes. Expect a small performance boost, perhaps a 10% speed increase compared to default settings. Note that this is dependent on the `--mt` option, so make sure to include the relevant value, as in `Configure --tune --mt=pthread`.

`--graphic=`*lib*: enables a particular graphic library. The default is `X11` on most platforms, but PARI can use `fltk` or `win32` (GDI), or even dump a `ps` or `svg` file and open it using an external viewer.

`--time=`*function*: chooses a timing function. The default usually works fine, however you can use a different one that better fits your needs. PARI can use `getrusage`, `clock_gettime`, `times` or `ftime` as timing functions. (Not all timing functions are available on all platforms.) The three first functions give timings in terms of CPU usage of the current task, approximating the complexity of the algorithm. The last one, `ftime` is deprecated and gives timings in terms of absolute (wall-clock) time. Moreover, the `clock_gettime` function is more precise, but much slower (at the time of this writing), than `getrusage` or `times`.

`--with-runtime-perl=`*perl*: absolute path to the runtime `perl` binary to be used by the `gphelp` and `tex2mail` scripts. Defaults to the path found by `Configure` on the build host (usually `/usr/bin/perl`). For cross-compiling builds, when the target and build hosts have mismatched configurations; suggested values are

`/usr/bin/env perl`: the first `perl` executable found in user's `PATH`,

`/usr/bin/perl`: `perl`'s standard location.

The remaining options are specific to parallel programming. We provide an *Introduction to parallel GP programming* in the file `doc/parallel.dvi`, and to multi-threaded `libpari` programs in Appendix D. Beware that these options change the library ABI, in particular you will need to `make clean` between successive builds using different values of this option:

`--mt=`*engine*: specify the engine used for parallel computations. Supported value are

• single: (default) no parallellism.

• pthread: use POSIX threads. This is well-suited for multi-core systems. Setting this option also set `--enable-tls`, see below. This option requires the pthread library.

• mpi: use the MPI interface to parallelism. This allows PARI to take advantage of clusters using MPI. This option requires a MPI library. It is usually necessary to set the environment variable `CC` to `mpicc`.

Note that running `Configure --tune` implicitly compiles and tunes for `--mt=single`; if you intend to install a binary compiled with another option, say `pthread`, you should use `Configure --tune --mt=pthread` instead.

`--enable-tls`: build the thread-safe version of the library. Implied by `--mt=pthread`. This tends to slow down the *shared* library `libpari.so` by about 25%, so you probably want to use the static library `libpari.a` instead.

2.3. Compilation. To compile the `gp` binary and build the documentation, type

```
make all
```

from the build directory. Recall the build directory is (by default) of the form `O`*osname*`-`*arch* or passed to `Configure` using the option `--build=`*DIR*. To only compile the `gp` binary, type

```
make gp
```

in the toplevel directory (this will correctly determine the `O`*osname*`-`*arch* default directory) or `make -C` *DIR* `gp` (it is also possible to `cd` *DIR* first).

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

2.4. Basic tests.

To test the binary, type `make bench`. This runs a quick series of tests, for a few seconds on modern machines.

In many cases, this will also build a different binary (named `gp-sta` or `gp-dyn`) linked in a slightly different way and run the tests with both. (In exotic configurations, one may pass all the tests while the other fails and we want to check for this.) To test only the default binary, use `make dobench` which starts the bench immediately.

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.

2.5. Cross-compiling.

When cross-compiling, you can set the environment variable `RUNTEST` to a program that is able to run the target binaries, e.g. an emulator. It will be used for both the `Configure` tests and `make bench`.

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 about 30 of them, and default answers are provided. If you accept all default answers, `Configure` 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 install location of programs and libraries is controlled by the `--prefix` configure option. The default location is `/usr/local`.

The precise destinations of programs and libraries 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 `$prefix/include/pari`. Other system-dependent data go to `$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 `$share_prefix/man`, and other system-independent data under `$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:

`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. Target plateform ABI.: By design, Configure will always build PARI for the default target ABI of the compiler. To build for a different ABI, add the relevant compiler flag to CFLAGS. Some examples (with gcc):

`CFLAGS=-m32` build 32-bit binaries (on a 64-bit system)

`CFLAGS=-mavx2` allows uses of AVX2 instructions on x86_64

`CFLAGS=-march=native` use the largest ABI supported by the CPU PARI is compiled on. Binaries will be faster but not portable to older CPU.

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

`Configure -g`

creates a directory `O`*xxx*`.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 `O`*xxx*`.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.

3.5. Multiprecision kernel. The kernel can be specified via the

`--kernel=`*fully_qualified_kernel_name*

switch. The PARI kernel consists of two levels: Level 0 (operation on words) and Level 1 (operation on multi-precision integers and reals), which can take the following values.

Level 0: `auto` (as detected), `none` (portable C) or one of the assembler micro-kernels

```
alpha
hppa hppa64
ia64
ix86 x86_64
m68k
ppc ppc64
sparcv7 sparcv8_micro sparcv8_super
```

Level 1: `auto` (as detected), `none` (native code only), or `gmp`

• A fully qualified kernel name is of the form *Level0*`-`*Level1*, the default value being `auto-auto`.

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

3.6. 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. You can 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`

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 macOS: these systems 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

`https://pari.math.u-bordeaux.fr/`.

3.7. Testing.

3.7.1. Known problems. if `BUG` shows up in `make bench`

• 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.

3.7.2. 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).

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

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

`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. Thorough, but slow. Some of the tests require extra packages (`elldata`, `galdata`, etc.) to be available. If you want to test such an extra package *before* `make install` (which would install it to its final location, where `gp` expects to find it), run

```
env GP_DATA_DIR=$PWD/data make test-all
```

from the PARI toplevel directory, otherwise the test will fail.

`make test-io` tests writing to and reading from files. It requires a working `system()` command (fails on Windows + MingW).

`make test-time` tests absolute and relative timers. This test has a tendency to fail when the machine is heavily loaded or if the granularity of the chosen system timer is bigger than 2ms. Try it a few times before reporting a problem.

`make test-install` tests the GP function `install`. This may not be available on your platform, triggering an error message ("not yet available for this architecture"). The implementation may be broken on your platform triggering an error or a crash when an install'ed function is used.

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 beginners: use `sudo make install`.) In this case, it is advised to type `make all` first to avoid running unnecessary commands as `root`.

Caveat. Install directories are created honouring your `umask` settings: if your umask is too restrictive, e.g. `077`, the installed files will not be world-readable. (Beware that running `sudo` may change your user umask.)

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.

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.

4.1. Static binaries and libraries. By default, if a dynamic library `libpari.so` can be built, the `gp` binary we install is `gp-dyn`, pointing to `libpari.so`. On the other hand, we can build a `gp` binary into which the `libpari` is statically linked (the library code is copied into the binary); that binary is not independent of the machine it was compiled on, and may still refer to other dynamic libraries than `libpari`.

You may want to compile your own programs in the same way, using the static `libpari.a` instead of `libpari.so`. By default this static library `libpari.a` is not created. If you want it as well, 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, possibly up to 20% when using `pthreads`), 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.

4.2. Extra packages. The following optional packages endow PARI with some extra capabilities:

• `elldata`: This package contains the elliptic curves in John Cremona's database. It is needed by the functions `ellidentify`, `ellsearch`, `forell` 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).

• `seadata`: This package contains the database of modular polynomials extracted from the ECHIDNA databases and computed by David R. Kohel. It is used to speed up the functions `ellap`, `ellcard` and `ellgroup` for primes larger than 10^{20}.

• `galpol`: This package contains the GALPOL database of polynomials defining Galois extensions of the rationals, accessed by `galoisgetpol`.

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.

4.3. 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 favorite 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 `pdftex` 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. The PARI Community. PARI's home page at the address

`https://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

`https://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 a number of mailing lists devoted to PARI/GP, and most feedback should be directed there. Instructions and archives can be consulted at

`https://pari.math.u-bordeaux.fr/lists-index.html`

The most important are:

• `pari-announce` (*read-only*): 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 or patch submissions. Bug reports can be discussed here, but as a rule it is better to submit them directly to the BTS.

• `pari-users`: for everything else.

You may send an email to the last two without being subscribed. To subscribe, send an message respectively to

`pari-announce-request@pari.math.u-bordeaux.fr`
`pari-users-request@pari.math.u-bordeaux.fr`

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

with the word `subscribe` in the `Subject:`. 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):

```
@preamble{\usepackage{url}}
@manual{PARI2,
    organization = "{The PARI~Group}",
    title        = "{PARI/GP version 2.19.1}",
    year         = 2026,
    address      = "Bordeaux",
    note         = "available from \url{https://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!