R Installation and Administration

R Development Core Team

Table of Contents

1 Obtaining R

Sources, binaries and documentation for R can be obtained via CRAN, the "Comprehensive R Archive Network". See the file 'RESOURCES' for information on CRAN.

1.1 Getting and unpacking the sources

The simplest way is to download the most recent 'R-$x.y.z$.tgz' file, and unpack it with

```
tar xvfz R-x.y.z.tgz
```

on systems that have GNU `tar` installed. On other systems you need at least to have the `gzip` program installed. Then you can use

```
gzip -dc R-x.y.z.tgz | tar xvf -
```

If you need to transport the sources on floppy disks, you can download the 'R-$x.y.z$.tgz-split.*' files and paste them together at the destination with (Unix)

```
cat R-x.y.z-split.* > R-x.y.z.tgz
```

and proceed as above. If you want the build to be usable by a group of users, set `umask` before unpacking so that the files will be readable by the target group (e.g., `umask 022` to be usable by all users).

Finally, for minor-minor releases ('$x.y.z$' with $z \mathrel{!}= 0$), a patch against the preceding release is made available in 'R-$x.y.\{z\text{-}1\}$-$x.y.z$.diff.gz' (e.g., 'R-1.2.2-1.2.3.diff.gz'), which is generally a much smaller file that the '.tgz' files. Such a file can be applied to the sources of the previous version by changing to the top directory of it and

```
gzip -dc /path/to/it/R-x.y.{z-1}-x.y.z.diff.gz | patch -E -p1
```

Beware that this does not necessarily work if the older sources have been modified (e.g., by building in their directories).

1.2 Using rsync

Sources are also available via anonymous rsync. Use

```
rsync -rC rsync.r-project.org::module R
```

to create a copy of the source tree specified by *module* in the subdirectory 'R' of the current directory, where *module* specifies one of the four existing flavors of the R sources, and can be one of 'r-release' (current released version), 'r-patched' (patched released version), and 'r-devel' (development version, less stable), and 'r-ng' (next generation, unstable). The rsync trees are created directly from the master CVS archive and are updated hourly. The '-C' option in the `rsync` command is to cause it to skip the CVS directories. Further information on `rsync` is available at http://rsync.samba.org/rsync/.

2 Installing R under Unix

R will configure and build from source under a number of common Unix-like platforms, including 'i386-freebsd', 'i386-linux-gnu', 'i386-sun-solaris', 'powerpc-linux-gnu', 'powerpc-apple-darwin', 'mips-sgi-irix', 'alpha-linux-gnu', 'alpha-dec-osf4', 'rs6000-ibm-aix', 'hppa-hp-hpux', 'sparc-linux-gnu' and 'sparc-sun-solaris'.

In addition, binary distributions are available for most common Linux distributions, Compaq Alpha systems running OSF/Tru64, and for MacOS X (Darwin) with X11. See the FAQ for current details. These are installed in platform-specific ways. So for the rest of this chapter we consider only building from the sources.

2.1 Simple compilation

First review the essential and useful tools and libraries in Appendix A [Essential and useful other programs], page 9, and install those you want or need.

Choose a place to install the R tree (R is not just a binary, but has additional data sets, help files, font metrics etc). Let us call this place 'R_HOME'. Untar the source code. This should create directories 'src', 'doc', and several more. Issue the following commands:

```
./configure
make
```

(See Section B.3 [Using make], page 12 if your make is not called 'make'.)

Then check the built system works correctly, by

```
make check
```

Failures are not necessarily problems as they might be caused by missing functionality, but you should look carefully at any reported discrepancies. To re-run the tests you would need

```
make check FORCE=FORCE
```

If these commands execute successfully, the R binary will be copied to the '${R_HOME}/bin' directory. In addition, a shell-script front-end called 'R' will be created and copied to the same directory. You can copy this script to a place where users can invoke it, for example to '/usr/local/bin/R'. You could also copy the man page 'R.1' to a place where your man reader finds it, such as '/usr/local/man/man1'. If you want to install the complete R tree to, e.g., '/usr/local/lib/R', see Section 2.3 [Installation], page 3. Note: you do not *need* to install R: you can run it from where it was built.

You do not necessarily have to build R in the top-level source directory (say, '*TOP_SRCDIR*'). To build in '*BUILDDIR*', run

```
cd BUILDDIR
TOP_SRCDIR/configure
make
```

and so on, as described further below. This has the advantage of always keeping your source tree "clean". (You may need GNU make to allow this.)

Make will also build plain text help pages as well as HTML and LaTeX versions of the R object documentation (the three kinds can also be generated separately using make help, make html and make latex). Note that you need Perl version 5: if this is not available on your system, you can obtain PDF versions of the documentation files via CRAN.

Now `rehash` if necessary, type `R`, and read the R manuals and the R FAQ (files 'FAQ' or 'doc/html/faq.html', or http://www.ci.tuwien.ac.at/~hornik/R/R-FAQ.html which always has the latest version).

2.2 Making the manuals

There is a set of manuals that can be built from the sources,

`refman`   Printed versions of all the help pages.

`R-FAQ`   R FAQ (which is already built for you).

`R-intro`   "An Introduction to R".

`R-data`   "R Data Import/Export".

`R-admin`   "R Installation and Administration", this manual.

`R-exts`   "Writing R Extensions".

`R-lang`   "The R Language Definition".

To make these, use

```
make dvi        to create DVI versions
make pdf        to create PDF versions
make info       to create info files (not refman).
```

You will not be able to build the info files unless you have `makeinfo` version 4 or later installed (and some Linux distributions have 3.12).

The DVI versions can be previewed and printed using standard programs such as `xdvi` and `dvips`. The PDF versions can be viewed using Acrobat Reader or (recent versions of) ghostscript: they have hyperlinks that can be followed in Acrobat Reader. The info files are suitable for reading online with Emacs or the standalone GNU Info.

2.3 Installation

After

```
./configure
make
make check
```

have been completed successfully, you can install the complete R tree to your system by typing

```
make install
```

This will install to the following directories:

'${prefix}/bin'
            the front-end shell script

'${prefix}/man/man1'
            the man page

'${prefix}/lib/R'
            all the rest (libraries, on-line help system, . . .)

where `prefix` is determined during configuration (typically '`/usr/local`') and can be set by running `configure` with the option '`--prefix`', as in

```
./configure --prefix=/where/you/want/R/to/go
```

This causes `make install` to install the R executable to '`/where/you/want/R/to/go/bin`', and so on. The prefix of the installation directories can be seen in the status message that is displayed at the end of `configure`. You can install into another directory by using

```
make prefix=/path/to/here install
```

To install DVI, info and PDF versions of the manuals, use one or more of

```
make install-dvi
make install-info
make install-pdf
```

To ensure that the installed tree is usable by the right group of users, set `umask` appropriately (perhaps to `022`) before unpacking the sources and throughout the build process.

3 Installing R under Windows

The 'bin/windows' directory of a CRAN site contains binaries for a base distribution and a large number of add-on packages from CRAN to run on Windows 95, 98, NT4, 2000 and ME (at least) on Intel and clones (but not on other platforms).

You do need one of those Windows versions: Windows 3.11+win32s will not work.

Your file system must allow long file names (as is likely except perhaps for some network-mounted systems).

The simplest way is to use 'SetupR.exe' or 'miniR.exe'. Just double-click on the icon and follow the instructions. If you installed R this way you can uninstall it from the Control Panel or Start Menu (unless you suppressed making a group for R).

See the R Windows FAQ for more details.

3.1 Building from source

If you want to build this port from the sources, see the file 'src/gnuwin32/INSTALL' in the source distribution. You will need to collect, install and test an extensive set of tools: see http://www.stats.ox.ac.uk/pub/Rtools/ for the current locations.

You may need to compile under a case-honouring file system: we found that a samba-mounted file system (which maps all file names to lower case) did not work. Open a commands window at a directory whose path does not contain spaces, and run something like

```
tar zxvf R-1.3.0.tgz
cd R-1.3.0\src\gnuwin32
make
```

sit back and wait (for about 5 minutes on 1GHz PIII with a fast local disc).

For further details, including how to make the documentation and how to cross-compile, see 'src/gnuwin32/INSTALL'.

4 Installing R on Classic MacOS

The 'bin/macos' directory of a CRAN site contains bin-hexed ('hqx') and stuffit ('sit')
archives for a base distribution and a large number of add-on packages to run under MacOS
8.6 to MacOS 9.1 or MacOS X natively. Just extract one of these archives in a suitable
folder using standard utilities like Aladdin Stuffit Expander (tm).

There is also a port to MacOS X which is considered to be a Unix variant in this
document. You can find it in the 'bin/macosx' directory at of a CRAN site.

5 Add-on packages

This chapter applies to Unix-like and Windows versions of R, but not to the Classic MacOS port.

It is helpful to use the correct terminology. A *package* is loaded from a *library* by the function `library`. Thus a library is a directory containing installed packages; the main library is '`R_HOME/library`', but others can be used, for example by setting the environment variable `R_LIBS` or using the R function `.libPaths()`.

5.1 Installing packages

Note that you need to specify implicitly or explicitly the library to which the package is to be installed. This is only an issue if you have more than one library, of course.

To install packages from source on Unix use

```
R CMD INSTALL -l /path/to/library pkg1 pkg2 ...
```

The part '`-l /path/to/library`' can be omitted, when the first library in `R_LIBS` is used if set, otherwise the main library '`R_HOME/library`'.

The Windows equivalent is[1]

```
Rcmd INSTALL -l /path/to/library pkg1 pkg2 ...
```

Alternatively, packages can be downloaded and installed from within R. First set the option `CRAN` to your nearest CRAN mirror, for example

```
> options(CRAN = "http://cran.us.r-project.org/")
```

Then download and install package **foo** by

```
> install.packages("foo")
```

Unless the library is specified (argument `lib`) the first library in the library search path is used.

What this does is different on Unix and Windows. On Unix it consults the list of available source packages on CRAN, downloads the latest version of the **foo** sources, and installs it (via `R CMD INSTALL`). On Windows it looks at the list of *binary* versions of packages and downloads the latest version (if any).

On Windows `install.packages` can also install a binary package from a local '`zip`' file by setting argument `CRAN` to `NULL`. `RGui.exe` has a menu `Packages` with a GUI interface to `install.packages`, `update.packages` and `library`.

5.2 Updating packages

The command `update.packages()` is the simplest way to ensure that all the packages on your system are up to date. Set the `CRAN` option as in the previous section. The `update.packages()` downloads the list of available packages and their current versions, compares it with those installed and offers to fetch and install any that have later versions on CRAN.

---

[1] if you have Perl and the source-code package files installed

An alternative way of keeping packages up-to-date is provided by the command `packageStatus()`, which returns an object with information on all installed packages and packages available at multiple repositories (CRAN, local archive, ...). The `print` and `summary` methods give an overview of installed and available packages, the `upgrade` method offers to fetch and install outdated packages. This allows R to fetch packages from several repositories and keep in sync with all of them, instead of only one CRAN mirror, and is intended to become the default package manager for future versions of R.

5.3 Removing packages

Packages can be removed in a number of ways. From a command prompt they can be removed by

```
R CMD REMOVE -l /path/to/library pkg1 pkg2 ...
```
(Unix) or
```
Rcmd REMOVE -l /path/to/library pkg1 pkg2 ...
```
(Windows).

From a running R process they can be removed by

```
> remove.packages(c("pkg1", "pkg2"),
                   lib = file.path("path", "to", "library"))
```

Finally, in most installations one can just remove the package directory from the library.

Appendix A Essential and useful other programs

This appendix gives details of programs you will need to build R on Unix-like platforms, or which will be used by R if found by `./configure`.

A.1 Essential programs

You need a means of compiling C and FORTRAN (see Section B.4 [Using FORTRAN], page 12). Some add-on packages also need a C++ compiler.

You will need Perl version 5, available via http://www.perl.com/CPAN/, to build any of the on-line documentation.

You will not be able to build the info files unless you have `makeinfo` version 4 or later installed (and some Linux distributions have 3.12).

The typeset documentation needs `tex` and `latex`, or `pdftex` and `pdflatex`.

A.2 Useful libraries and programs

The command-line editing depends on the `readline` library available from any GNU mirror: you will need a fairly recent version.

Use of `gzfile` connections needs `zlib` (version 1.1.3 or later).

The bitmapped graphics devices `jpeg()` and `png()` need the appropriate libraries installed: `jpeg` (version 6b or later) or `libpng` (version 1.0.5 or later) and `zlib` (version 1.1.3 or later) respectively.

The `bitmap` and `dev2bitmap` devices make use of ghostscript (http://www.cs.wisc.edu/~ghost).

A.2.1 Tcl/Tk

The **tcltk** package needs Tcl/Tk installed: the sources are available at http://dev.scriptics.com/. To specify the locations of the Tcl/Tk files you may need the configuration options

'`--with-tcltk`'
  use Tcl/Tk, or specify its library directory

'`--with-tcl-config=`*TCL_CONFIG*'
  specify location of '`tclConfig.sh`'

'`--with-tk-config=`*TK_CONFIG*'
  specify location of '`tkConfig.sh`'

A.2.2 Linear algebra

The linear algebra routines in R can make use of enhanced BLAS (Basic Linear Algebra Subprograms, http://www.netlib.org/blas/faq.html) routines. Some are compiler-system-specific (`libsunperf` on Sun Sparc[1], `libessl` on IBM) but ATLAS

---

[1] Using the SunPro `cc` and `f95` compilers

(http://www.netlib.org/atlas/) is a 'tuned' BLAS that runs on a wide range of Unix-alike platforms. If no more specific library is found, a `libblas` library in the library path will be used. You can specify a specific BLAS library by the configuration option '`--with-blas`' and not to use an external BLAS library by '`--without-blas`'.

Note that the BLAS library will be used for several add-on packages as well as for R itself. This means that it is better to use a shared BLAS library, as most of a static library will be compiled into the R executable and each BLAS-using package.

You will need double-precision and double-complex versions of the BLAS, but not single-precision nor complex routines.

Optimized versions of LAPACK are available, but no provision is made for using them with R as the likely performance gains are thought to be small.

As with all libraries, you need to ensure that they and R were compiled with compatible compilers and flags. For example, this means that on Sun Sparc using the native compilers the flag '`-dalign`' is needed so `libsunperf` can be used.

An ATLAS 'tuned' BLAS can also be used on Windows: see '`src/gnuwin32/INSTALL`' for how to enable this.

Note that if R is compiled against a non-default BLAS, then all BLAS-using packages must also be. So if R is re-built after ATLAS is installed, then packages such as **quantreg** will need to be re-installed.

Appendix B Configuration on Unix

B.1 Configuration options

`configure` has many options: running

```
./configure --help
```

will give a list. Probably the most important ones not covered elsewhere are (defaults in brackets)

'`--with-x`'

  use the X Window System

'`--x-includes=`*DIR*'

  X include files are in *DIR*

'`--x-libraries=`*DIR*'

  X library files are in *DIR*

'`--with-readline`'

  use readline library (if available) [yes]

'`--enable-R-profiling`'

  attempt to compile support for `Rprof()` [yes]

'`--enable-R-shlib`'

  build R as a shared library [no]

You can use '`--without-foo`' or '`--disable-foo`' for the negatives.

You will want to use '`--disable-R-profiling`' if you are building a profiled executable of R (e.g. with '`-pg)`'.

Flag '`--enable-R-shlib`' causes the make process to build R as a shared library, typically called '`libR.so`', and to take considerably longer, so you probably only want this if you will be using an application which embeds R.

B.2 Configuration variables

If you need or want to set certain configure variables to something other than their default, you can do that by either editing the file '`config.site`' (which documents all the variables you might want to set) or on the command line as

```
VAR="..." ./configure               # Bourne shell compatibles
(setenv VAR "..."; ./configure)     # C shell
```

One common variable to change is `R_PAPERSIZE`, which defaults to '`a4`', not '`letter`'.

If you have libraries and header files, e.g., for GNU readline, in non-system directories, use the variables `LIBS` (for libraries) and `CPPFLAGS` (for header files), respectively, to specify these locations. These default to '`/usr/local/lib`' and '`/usr/local/include`' to catch the most common cases. If libraries are still not found, then maybe your compiler/linker does not support re-ordering of '`-L`' and '`-l`' flags (this has been reported to be a problem on HP-UX with the native `cc`). In this case, use a different compiler (or a front end shell script which does the re-ordering).

If you find you need to alter configure variables, it is worth noting that some settings are cached in the file 'config.cache', and it is a good idea to remove that file before re-configuring.

B.3 Using make

To compile R, you will most likely find it easiest to use GNU make. On Solaris 2.6/7/8 in particular, you need a version of GNU make different from 3.77; 3.79 works fine, as does the Sun make. The native make is reported to fail on SGI Irix 6.5.

To build in a separate directory you need a make that uses the VPATH variable, for example GNU make, or Sun make on Solaris 2.7/8 (but not earlier).

If you want to use a make by another name, for example if your GNU make is called gmake, you need to set the environment variable MAKE at configure time, for example

```
MAKE=gmake ./configure        (sh, bash)
env MAKE=gmake ./configure    (csh)
```

B.4 Using FORTRAN

To compile R, you need a FORTRAN compiler or f2c, the FORTRAN-to-C converter (http://www.netlib.org/f2c). The default is to search for g77, f77, xlf, cf77, cft77, pgf77, fl32, af77, fort77, f90, xlf90, pgf90, epcf90, f95, xlf95, lf95, g95, and fc (in that order)[1], and then for f2c, and use whichever is found first; if none is found, R cannot be compiled. The search mechanism can be changed using the '--with-g77', '--with-f77', and '--with-f2c' command line options to configure. If your FORTRAN compiler is in a non-standard location, you should set the environment variable PATH accordingly before running configure.

If your FORTRAN libraries are in slightly peculiar places, you should also look at LD_LIBRARY_PATH or your system's equivalent to make sure that all libraries are on this path.

You must set whatever compilation flags (if any) are needed to ensure that FORTRAN integer is equivalent to a C int pointer and FORTRAN double precision is equivalent to a C double pointer. This is checked during the configuration process.

Some of the FORTRAN code makes use of COMPLEX*16 variables, which is a FORTRAN 90 extension. This is checked for at configure time[2], but you may need to avoid compiler flags[3] asserting FORTRAN 77 compliance.

For performance reasons[4] you may want to choose a FORTRAN 90/95 compiler.

If you use f2c you may need to ensure that the FORTRAN type integer is translated to the C type int. Normally 'f2c.h' contains typedef long int integer;, which will work on a 32-bit platform but not on a 64-bit platform.

---

[1] On HPUX fort77 is the POSIX compliant FORTRAN compiler, and comes second in the search list.

[2] as well as its equivalence to the Rcomplex structure defined in 'R_ext/Complex.h'.

[3] In particular, avoid g77's '-pedantic', which gives confusing error messages.

[4] e.g., to use an optimized BLAS on Sun/Sparc

B.5 Compile and load flags

A wide range of flags can be set in the file 'config.site' or via environment variables. We have already mentioned

CPPFLAGS   extra include flags

LIBS   libraries and '-L/lib/path' flags

and others include

CFLAGS   debugging and optimization flags, C

MAIN_CFLAGS
            ditto, for compiling the main program

SHLIB_CFLAGS
            for shared libraries

FFLAGS   debugging and optimization flags, FORTRAN

MAIN_FFLAGS
            ditto, for compiling the main program

SHLIB_FFLAGS
            for shared libraries

MAIN_LDFLAGS
            additional flags for the main link

SHLIB_LDFLAGS
            additional flags for linking the shared libraries

Library paths specified as '-L/lib/path' in LIBS are collected together and prepended to LD_LIBRARY_PATH (or your system's equivalent), so there should be no need for '-R' or '-rpath' flags.

To compile a profiling version of R, one might for example want to use 'MAIN_CFLAGS=-pg', 'MAIN_FFLAGS=-pg', 'MAIN_LDFLAGS=-pg' on platforms where '-pg' cannot be used with position-independent code.

Beware: it may be necessary to set CFLAGS and FFLAGS in ways compatible with the libraries to be used: one possible issue is the alignment of doubles, another is the way structures are passed.

B.6 Building the GNOME interface

This interface is experimental and incomplete. It provides a console and two graphics devices named gtk() and gnome(). The console offers a basic command line editing and history mechanism, along with tool and button bars that give a point-and-click interface to some R commands. Many of the features of the console are currently stubs. The gtk() graphics device is a port of the x11() device to GDK (the GIMP Drawing Kit). The gnome() device uses the GNOME canvas.

Due to its experimental nature, the GNOME interface for R will not be built automatically. You must specify it by running configure with the '--with-gnome' option. For example, you might run

```
    ./configure --with-gnome
```

but please check you have all the requirements first. The GNOME interface for R is currently being developed under GNOME 1.4, and is not guaranteed to work with previous versions. You therefore need at least the following libraries installed

```
    audiofile-0.2.1
    esound-0.2.23
    glib-1.2.10
    gtk+-1.2.10
    imlib-1.9.10
    ORBit-0.5.12
    gnome-libs-1.4.1.2
    libxml-1.8.16
    libglade-0.17
```

It is preferable to have a complete installation of the GNOME desktop environment. If you use Linux, then this should be provided with your distribution. In addition, packaged binary distributions of GNOME are available from http://www.ximian.com for the most popular Linux distributions and for Solaris.

Remember that some package management systems (such as RPM and deb) make a distinction between the user version of a package and the developer version. The latter usually has the same name but with the extension "-devel". If you use a pre-packaged version of GNOME then you must have the developer versions of the above packages in order to compile the R-GNOME interface.

The full list of GNOME options to configure is

'`--with-gnome`'
>   use GNOME, or specify its prefix [no]

'`--with-gnome-includes=`*DIR*'
>   specify location of GNOME headers

'`--with-gnome-libs=`*DIR*'
>   specify location of GNOME libs

'`--with-libglade-config=`*LIBGLADE_CONFIG*'
>   specify location of `libglade-config`

B.7 Platform notes

This section provides some notes on building R on different Unix-like platforms. These notes are based on tests run on one or two systems in each case with particular sets of compilers and support libraries. Success in building R depends on the proper installation and functioning of support software; your results may differ if you have other versions of compilers and support libraries.

B.7.1 MacOS X

You can build R as a Unix application on MacOS X. You will need the DevTools, `f2c` or `g77`, and the `dlcompat` library. You will also need to install an X sub-system or configure with '`--without-x`'.

f2c, g77, the `dlcompat` library, and X server and support libraries are available from the Fink project (http://fink.sourceforge.net). At the time of writing `f2c` and `g77` were not available as part of the Fink binary distribution and needed to be installed directly; for example for g77 use

```
fink install g77
```

Fink prefers to install in '`/sw`' so `CPPFLAGS`, `LDFLAGS` need to be set accordingly. If you want to use Tcl/Tk, also available from Fink, then you will also need to use the `configure` options to specify the Tcl and Tk config locations.

B.7.2 Solaris on Sparc

R has been built successfully on Solaris 2.7 using `gcc/g77` and the SunPro WorkShop 6 compilers, and less regularly on 2.5.1, 2.6 and 8. GNU `make` is needed prior to 2.7 for building other than in the source tree, and perhaps even then.

When using the SunPro compilers do *not* specify '`-fast`', as this disables IEEE arithmetic and `make check` will fail. The maximal set of optimization options known to work is

```
-xlibmil -xO5 -dalign
```

We have found little performance difference between `gcc` and `cc` but considerable benefit from using a SunPro Fortran compiler: the `gcc/f77` combination works well.

To compile for a 64-bit target on Solaris (which needs an UltraSparc chip and for support to be enabled in the OS) we used

```
CC="cc -xarch=v9"
CFLAGS="-xO5 -xlibmil -dalign"
FC="f95 -xarch=v9"
FFLAGS="-xO5 -xlibmil -dalign"
CXX=CC
CXXFLAGS="-xO5 -xlibmil -dalign -xarch=v9"
```

in '`config.site`'. Note that using `f95` allows the Sun performance library to be selected: it will not work with `f77`, nor with `g77`.

Some case is needed to ensure that libraries found by `configure` are compatible with the R executable and modules, as the testing process will not detect many of the possible problems. For 32-bit builds under `cc` the flag '`-dalign`' is needed for some of the Sun libraries: fortunately the equivalent flag for `gcc`, '`-munaligned-doubles`', is the default. In theory, libraries such as `libpng`, `libjpeg`, `zlib` and the ATLAS libraries need to be built with a `pic` or `PIC` flag, which could be a problem if static libraries are used. In practice this seems to give little problem for 32-bit builds, but seems essential for 64-bit builds.

For a 64-bit build, 64-bit libraries must be used. As the configuration process starts with `LIBS=/usr/local/lib` you may need to reset it to avoid finding 32-bit addons.

B.7.3 HPUX

R has been built successfully on HPUX 10.2 and HPUX 11.0 using both native compilers and `gcc`. By default, R is configured to use `gcc` and `g77` on HPUX (if available). Some installations of `g77` only install a static version of the `g2c` library that cannot be linked into a shared library since its files have not been compiled with the appropriate flag for

producing position independent code (PIC). This will result in `make` failing with a linker error complaining that code needs to be compiled with a '+z' or '+Z' flag, the native `cc` PIC flag. If this is the case you either need to modify your `g77` installation or configure with the

```
--with-f77=fort77
```

flag to specify use of the native POSIX-compliant FORTRAN 77 compiler.

You may find that `configure` detects other libraries, such as BLAS, that R needs to use as shared libraries but are only available as static libraries. If you cannot install shared versions you will need to tell `configure` not to use these libraries.

Some versions of `gcc` may contain what appears to be a bug at the '-O2' optimization level that causes

```
> 2 %/% 2
[1] 1
> 1:2 %/% 2
[1] 0 0      # wrong!!
```

which will cause `make check` to fail. If this is the case, you should use `CFLAGS` to specify '-O' as the optimization level to use.

Some systems running HPUX 11.0 may have a `gcc` that was installed under HPUX 10.2. Between versions 10.2 and 11.0 HPUX changed its support functions for IEEE arithmetic from the recommended functions of the IEEE standard to the ones specified in the C9x draft standard. In particular, this means that `finite` has been replaced by `isfinite`. A `gcc` configured for HPUX 10.2 run on 11.0 will not find `isfinite`, and as a result `configure` does not recognize the machine as fully supporting IEEE arithmetic and does not define `IEEE_754` when compiling C code. This results in a failure in `make check`. The best solution is to install a properly configured `gcc`. An alternative work-around is to add '-DIEEE_754' to the `CFLAGS` variable.

You can configure R to use both the native `cc` and `fort77` with (assuming you are using a `csh` shell)

```
env CC=cc ./configure --with-f77=fort77
```

B.7.4 IRIX

R has been built successfully on IRIX64 6.5 using `gcc/f77` or `cc/f77` for 32-bit executables and the native compilers for a 64-bit executable. The command

```
env CC="cc -64" ./configure --with-f77="f77 -64" --with-tcltk=no
```

was used to create the 64-bit executable. It was necessary to explicitly omit Tcl/Tk because `configure` would find the 32-bit version but not detect that is was incompatible with a 64-bit build.

A 32-bit build using `gcc/g77` passed `make check` but failed `make test-all-extras` in the complex LAPACK tests.

B.7.5 Alpha/OSF1

R has been built successfully on an Alpha running OSF1 V4.0 using `gcc/g77` and `cc/f77`. Mixing `cc` and `g77` fails to configure. The `configure` option '--without-blas' was used since the native blas seems not to have been built with the flags needed to suppress `SIGFPE`'s.

Currently R does not set a signal handler for `SIGFPE` on platforms that support IEEE arithmetic, so these are fatal.

`make check` passes with no problems. `make test-all-extras` in the 'tests' directory fails at the end of `reg-tests-1.R` since this system does not have `vsnprintf`. As currently written, the test at the end of `reg-tests-1.R` causes a segfault in this case. You can complete the tests by executing

    touch reg-tests-1.Rout

B.7.6 Alpha/FreeBSD

Attempts to build R on an Alpha with FreeBSD 4.3 have been only partly successful. Configuring with '`-mieee`' added to both `CFLAGS` and `FFLAGS` builds successfully, but tests fail with `SIGFPE`'s. It would appear that '`-mieee`' only defers these rather than suppressing them entirely. Advice on how to complete this port would be greatly appreciated.

Appendix C New platforms

There are a number of sources of problems when installing R on a new hardware/OS platform. These include

Floating Point Arithmetic: R supports the POSIX, SVID and IEEE models for floating point arithmetic. The POSIX and SVID models provide no problems. The IEEE model however can be a pain. The problem is that there is no agreement on how to set the signalling behaviour; Sun/Sparc, SGI/IRIX and ix86 Linux require no special action, FreeBSD requires a call to (the macro) `fpsetmask(0)` and OSF1 requires that computation be done with a '`-ieee_with_inexact`' flag etc. On a new platform you must find out the magic recipe and add some code to make it work. This can often be done via the file '`config.site`' which resides in the top level directory.

Beware of using high levels of optimization, at least initially. On many compilers these reduce the degree of compliance to the IEEE model. For example, using '`-fast`' on the Solaris SunPro compilers causes R's `NaN` to be set incorrectly.

Shared Libraries: There seems to be very little agreement across platforms on what needs to be done to build shared libraries. there are many different combinations of flags for the compilers and loaders. GNU libtool cannot be used (yet), as it currently does not fully support FORTRAN (and will most likely never support `f2c`: one would need a shell wrapper for this). The technique we use is to first interrogate the X window system about what it does (using `xmkmf`), and then override this in situations where we know better (for tools from the GNU Compiler Collection and/or platforms we know about). This typically works, but you may have to manually override the results. Scanning the `cc(1)` and `ld(1)` manual entries usually reveals the correct incantation. Once you know the recipe you can modify the file '`config.site`' (following the instructions therein) so that the build will use these options.

If you do manage to get R running on a new platform please let us know about it so we can modify the configuration procedures to include that platform.

If you are having trouble getting R to work on your platform please feel free to get in touch to ask questions. We've had a fair amount of practice at porting R to new platforms . . . .

Function and variable index

Concept index

F

H

I

L

M

O

P

S