Building gretl on MS Windows

Allin Cottrell

Revised, August 2023

Please note: This revision comes several years after the original version of "Building gretl on MS Windows" and in that time there have been substantial changes in MSYS2. If you have an old MSYS2 installation in place and want to try building according to the current guide, we recommend that you uninstall MSYS2 (after copying out any personal files you want to keep), then install the latest version.

1 Introduction

From the 2017d release onward, building gretl on MS Windows is relatively straightforward. This depends on MSYS2 and Mingw-w64; the gretl developers have neither the time nor the relevant expertise to support a gretl build using Microsoft's own compiler/toolchain. The basic idea is that with the help of the specified (free) tools one can do on Windows the same sort of "`./configure ; make ; make install`" routine that one does on Linux or similar OS.

This document explains how to install MSYS2, Mingw-w64 and related tools. It then presents a shell script which can be used to perform the required set-up for building gretl, starting from a basic install of MSYS2, followed by discussion of the various steps involved. We recommend reading the whole of this document before diving in.

2 MSYS2 and Mingw-w64

MSYS2 provides a unix-type shell; the Mingw-w64 compiler and related tools can then be installed from within that shell, using the package manager pacman (ported from Arch Linux) which ships as a base component of MSYS2. There's excellent documentation for pacman readily available on the web. It's easy to use, but you'll want to take a short while to acquaint yourself with basic usage; see https://wiki.archlinux.org/index.php/pacman and/or the brief "cheat sheet" at https://gretl.sourceforge.net/winbuild/pacman.txt (or see Appendix B below).

MSYS2 can be found at https://www.msys2.org/. Download and installation are pretty straightforward, but we recommend reading the long-form guide at

https://www.msys2.org/wiki/MSYS2-installation/

MSYS2 offers several software-building "environments," which differ by the compiler used (gcc or clang) and the Microsoft C library employed (the traditional MSVCRT or the newer UCRT). The choices are set out and clearly explained at

https://www.msys2.org/docs/environments/

and of course you're free to experiment with any of them. But if you want to follow this guide exactly, please use the MINGW64 environment (which employs gcc and MSVCRT).[1]

[1] URCT may be the way to go in future, but not yet: I gather it may be problematic to mix elements from the two systems, and our build process relies on some pieces cross-compiled on Linux against MSVCRT.

With current MSYS2 you'll find that the default shell now accesses the UCRT64 environment; that name will appear in the command prompt on first starting the program after installation. To get to MINGW64, close that shell window and instead open `c:/msys64/mingw64.exe`, to which you'll probably want to create a Desktop shortcut. Or you can open a MINGW64 window via the menu item MSYS2 MINGW64 in the MSYS2 group under the Windows Start menu.

A few more words on understanding MSYS2. In the shell, you'll find yourself in your "home" directory under the MSYS tree, which is by default "rooted" at `c:\msys64` in the Windows filesystem. You refer to paths unix-style (with forward slashes). If you want to refer to paths outside of the MSYS tree, use this sort of syntax:

```
/c/Users/yourname
```

where "`/c`" takes you outside, to the root of the `C:` drive. The cygpath utility can be used to translate between unix-style paths and native Windows ones.

3 Setting up for a gretl build

Step zero, particularly if you're starting from scratch with a fresh install of MSYS2, is to ensure that the system is up to date (since things may have moved on in MSYS-world since the installer package was created). And since we use wget as a downloading tool below, let's go ahead and install it right away. Here's the combined command:

```
pacman -Syu
pacman -S wget --needed
```

(The `--needed` flag tells pacman not to bother downloading a package is it's already up to date locally.)

Listing 1 shows a shell script, `setup.sh`, that can be used to perform all the set-up steps for building gretl from the git sources. Assuming you have a basic installation of MSYS2 you could download the script and run it from the MSYS2 shell:

```
wget -N https://gretl.sourceforge.net/winbuild/setup.sh
bash setup.sh
```

In the remainder of this section we offer some comments to facilitate understanding of what's going on in `setup.sh` and why, and to enable you to do the job step-by-step yourself if you prefer.

3.1 Adding required MSYS2 packages

The first substantive step in `setup.sh` is to install a bunch of packages, namely these:

```
make
git
unzip
diffutils
intltool
mingw-w64-x86_64-gcc
mingw-w64-x86_64-gcc-fortran
mingw-w64-x86_64-gtk2
mingw-w64-x86_64-fftw
mingw-w64-x86_64-json_glib
mingw-w64-x86_64-libgsf
mingw-w64-x86_64-dlfcn
mingw-w64-x86_64-libxslt
mingw-w64-x86_64-pkg-config
```

Listing 1: setup.sh: script to automate gretl build setup

This script can be downloaded from https://gretl.sourceforge.net/winbuild/. Prerequisites: a basic MSYS2 installation plus wget.

```
## start preliminaries
if [ "$MSYSTEM_CARCH" != "x86_64" ] ; then
  echo "Architecture is $MSYSTEM_CARCH but this script is designed"
  echo "for an x86_64 build (MING64)"
  exit 1
fi

TEXDIR=$(cygpath $HOMEDRIVE\\w32tex)
SOURCE='https://gretl.sourceforge.net/winbuild'
PKGLIST='pkglist.txt'
DEPS='gretl-x86_64-deps'
## end preliminaries

## Get MSYS2 Packages
cd # get into "home"
wget -N ${SOURCE}/${PKGLIST} || exit 1
for i in $(cat ${PKGLIST}); do pacman -S $i --needed --noconfirm ; done

## Make source + build arena
mkdir -p src
mkdir -p src/gretl-git
mkdir -p src/build
# and get gretl git sources
cd ~/src && git clone git://git.code.sf.net/p/gretl/git gretl-git

# Add gretl repo and install gretl 'deps' package
cd
wget -N ${SOURCE}/add-repo.sh && bash add-repo.sh && \
pacman -Sy ${DEPS} --needed --noconfirm

# Do we seem to have a TeX installation?
PDFL=`which pdflatex`
if [ "x$PDFL" != "x" ] ; then
  echo "It looks like TeX is already installed. If you reckon that's"
  echo "not right, you can call texinst.sh from the command line"
else
  # install w32tex
  cd
  wget -N ${SOURCE}/texinst.sh && bash texinst.sh $TEXDIR
  export PATH=$PATH:${TEXDIR}/bin
fi

cd ~/src/build && wget -N ${SOURCE}/conf.sh && bash conf.sh
```

If the script above runs successfully, you should be ready to do `make`, followed by `make install`, in `~/src/build`.

To install these packages manually, start by executing the command

```
pacman -Sy
```

to ensure your package database is up to date. Then go through the list doing

```
pacman -S <pkgname>
```

replacing <pkgname> with each of the names shown above. Note that the packages with prefix mingw-w64-x86_64 are specific to the MINGW64 environment while the others are generic MSYS2 utilities.

A semi-automatic alternative is to download pkglist.txt and then process it:

```
wget -N https://gretl.sourceforge.net/winbuild/pkglist.txt
pacman -Sy
pacman -S - < pkglist.txt --needed --noconfirm
```

In addition to the required packages, if you want a high-powered text editor we recommend installing emacs:

```
pacman -S mingw-w64-x86_64-emacs
```

Or if you prefer a more Windows-y editor, Notepad++ is a good choice—see https://notepad-plus-plus.org/.

3.2 Creating source and build trees

The next order of business is to create an arena in which to store the gretl sources and build the software.

We strongly recommend building outside of the directory that contains the gretl source files. That way you won't get your own files mixed up with the sources, and if something goes wrong it's easy to blow away your build attempt and start over. Here's how you might create a suitable setup manually, under your home directory.

```
cd            # ensure you're in "home"
mkdir src     # create top-level directory for handling sources
cd src
mkdir gretl-git
# download the gretl git sources into ~/src/gretl-git
git clone git://git.code.sf.net/p/gretl/git gretl-git
mkdir build   # create a build directory parallel to gretl-git
```

If you have write permission for the gretl repo you can substitute

```
git clone ssh://USERNAME@git.code.sf.net/p/gretl/git gretl-git
```

(giving your SF username, of course) for the git clone invocation above.

3.3 Additional gretl-specific package

Besides the regular MINGW64 packages and the gretl source code itself, you will need an additional pacman package containing various bits and pieces (DLLs, import libraries, headers) which support features present in the release and snapshot packages that we prepare for MS Windows. Installation of this package is handled by setup.sh, but if you prefer to do things manually the commands are as follows:

```
wget https://gretl.sourceforge.net/winbuild/add-repo.sh
# add gretl-specific package repo to /etc/pacman.conf
bash add-repo.sh
pacman -Sy gretl-x86\_64-deps
```

The deps package supplies a build of OpenBLAS parallelized via OpenMP, an import library and header for Microsoft's MPI implementation, headers to support direct access by gretl to the R shared library, support for the GTK "Clearlooks" theme, and a build of libcurl that is suitable for use with gretl. In principle you could build these things for yourself—or omit some of the optional components—but to keep things as simple as possible for now, please just install the package.

3.4 TEXDIR and related matters

Building the gretl documentation requires that you have a working TEX installation; we recommend using w32tex, and setup.sh takes care of installing it. If you want to take charge of that yourself you can download https://gretl.sourceforge.net/winbuild/texinst.sh and modify it if necessary. If you already have a reasonably complete TEX installation that might work; experiment if you wish.

For the gretl build you need to ensure that your TEX binary directory is in your MSYS2 PATH. This can be done by adding the line

```
export PATH=$PATH:/c/w32tex/bin # or whatever
```

to the file .bash_profile in your home directory. That will take effect when you next open an MSYS2 terminal window; to make it happen immediately, in addition execute the line above at the command prompt.

3.5 Configuration

The first step towards building gretl is running the configure script. A plain-vanilla invocation would look like this (if you follow the directory layout suggested above):

```
cd ~/src/build
../gretl-git/configure
```

But wait! You'll want to pass some options to the configure script. Exactly which options depends on what sort of gretl installation you want: Is this just for your own use, or do you wish to make a package that you can distribute to others (and that will work OK without any MSYS2 installation)? The "own use only" case is pretty simple and we'll discuss it here. The case of producing a distributable package is handled in section 6 below.

We show below an example invocation of gretl's configure script:

```
RLIB_CFLAGS="-I${MSYSTEM_PREFIX}/lib/R/include" \
MPICC=gcc MPILINK="-lmsmpi" \
LAPACK_LIBS="-lopenblas -lgfortran" \
../gretl-git/configure --prefix=/opt/gretl \
 --disable-gnuplot-checks \
 --enable-build-doc \
 --enable-build-addons \
 --enable-quiet-build
```

This can be downloaded from

https://gretl.sourceforge.net/winbuild/conf.sh

You could place this file in your build directory and call it via the command

```
bash conf.sh
```

Note the choice of installation prefix: `/opt/gretl`. The `/opt` directory is a recognized location for optional software and by putting gretl into its own directory you ensure that it won't get tangled up with any packages installed via pacman. In addition it will be easy to remove the entire gretl installation if need be, via `rm -rf /opt/gretl`.

Also note that the flag `--enable-quiet-build` is purely optional; you can omit it to see a verbose account of the build process.

4 Build and install

Having completed all of the above, actually building gretl should be straightforward. In your build directory, first do

```
make
```

and if that goes OK, then `make install`.

To build and install the documentation, follow up with

```
make pdfdocs
make install-doc
```

Note that your TEX `bin` directory must be in your path for the doc build to work—see Section 3.4.

4.1 Gnuplot

Gretl calls gnuplot to make plots, so for normal functionality you'll need to install it. You could build the program yourself, but for an easier option you can download a build of gnuplot that's suitable for use with gretl.[2] Our gnuplot package is available in two forms. If you followed the recommendation of installing gretl in `/opt/gretl` you can just do

```
pacman -Sy gretl-x86_64-gnuplot
```

This assumes that pacman is aware of the gretl-specific package repository at sourceforge, as described in section 3.3.

If you installed gretl in some other location, you will have to download and extract a gnuplot "tarball" manually, in the `bin` subdirectory of your gretl installation, as in

```
cd
wget https://gretl.sourceforge.net/winbuild/gp544w64.tar.gz
cd $PREFIX/bin # substitute your gretl installation prefix
tar xvf ~/gp544w64.tar.gz && rm ~/gp544w64.tar.gz
```

5 Running your gretl build

The simplest way to run gretl as built according to this guide is to open a MINGW64 terminal window and invoke gretl as

```
/opt/gretl/bin/gretl &
```

[2] The build prerequisites for gretl also suffice for building gnuplot, but a few tweaks are needed to get things working nicely with gretl.

(where the ampersand has the effect of giving you the command prompt back). If that fails, something has gone wrong! If it succeeds, there are a couple of points to note.

First, if graphing doesn't work you should go to the Tools, Preferences, General menu and select the Programs tab: look for the item titled "Command to launch gnuplot" and ensure that the path is correct. It should look like this:

```
c:\msys64\opt\gretl\bin\wgnuplot.exe
```

Second, you may wish to set up a Desktop shortcut to gretl (hence avoiding the need to go via an MSYS2 terminal window). This can be done using the auto-generated script `mkshortcut.sh`, which will be found in the `win32` subdirectory of your build directory after configuring the gretl source. That is, do

```
bash win32/mkshortcut.sh
```

after installing gretl.

A shortcut can be created manually, but it's tricky since running `gretl.exe` requires that the MSYS2-supplied DLLs under `/bin` and `/mingw64/bin` are in your PATH. That's ensured automatically when you're working from the MSYS2 shell, but not when coming off a Windows shortcut. The shortcut created by `mkshortcut.sh` addresses this issue by targetting a batch file named `gretlrun.cmd`, which sets the environment correctly before calling `gretl.exe`. (The batch file gets installed along with the binaries.)

5.1 Coexistence with an installed gretl package

It's possible for a standard gretl release or snapshot package and gretl built under MSYS2 to coexist on the same machine, but you should note some potential points of contention.

The regular gretl installer for Windows creates several entries in the Windows registry (a) to associate specific icons with gretl-related files such as `.gdt` data files and `.inp` scripts, and (b) to map these file-types onto the installed `gretl.exe`, such that (for example) double-clicking on a `.gdt` file opens it using the `gretl.exe` installed under `Program Files`.

That has two implications. First, if there isn't any standard installation of gretl on the system, these associations will be absent: no special icons for gretl-related files, and no automatic action for double-clicking on them. Second, if there *is* a standard gretl installation in place it will "grab" the double-click action, regardless of whether you have your own gretl build in place under MSYS2.

What this means for you depends on whether (1) you wish to have your own gretl build coexist with an installed package, or (2) you're OK with using with your own build exclusively, and deleting any standard package you might have installed previously.

(1) **Coexistent**: With a standard gretl package installed, the gretl file icons should show up OK. You just have to remember that double-clicking on a gretl-related file will open it in the packaged `gretl.exe`, *not* your own build. To open a file in your own build, use the `open` command in a script or via the console, use the File menu, or use drag-and-drop.

(2) **Exclusive**: You don't have a standard gretl package installed or you're willing to remove such a package in favor of your own build. (In the latter case you should run the `uninstall` program under the gretl entry in the Programs menu to ensure that the registry entries created by the installer are removed.) But then, presumably, you'd like to add registry entries geared to your gretl build. After running gretl's `configure` script you should find in the `win32` subdirectory of your build directory a file name `gretlmime.reg`. Starting from your build directory you can do the following:

```
cd win32
```

```
bash ~/src/gretl-git/win32/dosify.sh gretlmime.reg
cp gretlmime.reg $USERPROFILE\\desktop
```

Then double-click on `gretlmime.reg` on your desktop to make the required changes to the Windows registry, after which you can delete the copy. Explanation: running `dosify.sh` on the `reg` file ensures that it uses Windows' CR/LF line termination; and copying it to the desktop for double-clicking ensures that Windows' `regedit` will escalate your privilege level as needed to make changes to the registry.

5.2 MPI support

Gretl built according to the instructions in this document will support MPI (parallelization via Message Passing Interface) *in potentia*; it will include the MPI executable, `gretlmpi.exe`. However, to activate this support you will have to install Microsoft's MPI apparatus. This is a free download, referenced at

https://learn.microsoft.com/en-us/message-passing-interface/

And if you wish to run gretl scripts that use MPI from an MSYS2 terminal window you will also have to add the appropriate directory to your MSYS2 PATH:

```
export PATH=$PATH:$MSMPI_BIN
```

6 Making a distributable package

This section addresses the creation of a gretl-for-Windows package that is usable by others who do not have MSYS2 installed. Here's an overview of what has to be done.

1. You'll need to install Inno Setup to create the self-installer package. This is very straightforward: download from https://www.jrsoftware.org/isdl.php and do as directed.
2. Your gretl build has to be configured specially; the sample configuration script shown in section 3.5 above will *not* do the job.
3. After `make` and `make install`, you should clean up a bunch of files that aren't needed for a runtime package.
4. You'll need to copy all the required "run-time" files (primarily DLLs but also various other things) from your MSYS2 tree into the package directory. This is a basically a one-time task.
5. You then need to create an installer script for Inno Setup and run the command-line package compiler, `ISCC.exe`.

Listing 2 shows a script, `pkgbuild.sh`, which automates steps 2 to 5 above.

We now proceed to expand on the actions performed by `pkgbuild.sh`, as well as filling in some steps which have to be performed manually.

6.1 Configuring for a distributable package

When configuring for a distributable package you should specify a target installation directory (`prefix`) that's outside of your MSYS2 tree. This will make it easier to verify that the package you're building is truly independent of MSYS2 as intended. In addition, the installation directory *must* be named `gretl`, and it should lie within another dedicated directory that will be used as workspace for building the package. So, for example, you might give

```
--prefix=/c/users/yourname/gbuild/gretl
```

Listing 2: Shell script to automate gretl package build

This script can be downloaded from https://gretl.sourceforge.net/winbuild/pkgbuild.sh. Prerequisites: everything that's needed for an "own use" build of gretl, plus Inno Setup must be installed.

```
# installation prefix outside of MSYS2 tree
if [ "x$PREFIX" = "x" ] ; then
  winpath="$HOMEDRIVE$HOMEPATH\\gbuild\\gretl"
  export PREFIX="$(cygpath $winpath)"
fi

SOURCE='https://gretl.sourceforge.net/winbuild'

cd ~/src
wget -N ${SOURCE}/pkgconf.sh
mkdir -p pkgbuild
cd pkgbuild
bash ../pkgconf.sh
make
make install
make install-doc # build and install PDF docs
bash win32/windist/post-install.sh # delete redundant files

cd $PREFIX/..
wget ${SOURCE}/mkdist.sh

# install-runtime won't have to be repeated often: only
# if there's a change in the dependency DLLs
cd ~/src/pkgbuild/win32/windist && \
 bash install-runtime.sh && cd $PREFIX/..

# run to create installer
bash mkdist.sh
```

Listing 3: Configuring for a package build

```
if [ "x$PREFIX" = "x" ] ; then
  winpath="$HOMEDRIVE$HOMEPATH\\gbuild\\gretl"
  export PREFIX="$(cygpath $winpath)"
fi

export CFLAGS='-O2'
export MPICC=gcc
export MPILINK='-lmsmpi'
export RLIB_CFLAGS='-I${MSYSTEM_PREFIX}/lib/R/include'
export CPPFLAGS=$CFLAGS
export LAPACK_LIBS='-lopenblas -lgfortran'
#
../gretl-git/configure --prefix=$PREFIX \
  --bindir=$PREFIX \
  --libdir=$PREFIX \
  --localedir=$PREFIX/locale \
  --datarootdir=$PREFIX \
  --disable-gnuplot-checks \
  --enable-pkgbuild \
  --enable-quiet-build \
  --enable-build-doc \
  --enable-build-addons \
  --disable-xdg \
  --disable-avx
```

where `/c/users/yourname/gbuild` is the dedicated workspace directory. In `pkgbuild.sh` the lines

```
winpath="$HOMEDRIVE$HOMEPATH\\gbuild\\gretl"
prefix="$(cygpath $winpath)"
```

will produce a `prefix` of this form, substituting the correct drive letter and Windows username. But note that if you specify `PREFIX` in the environment when calling this script that will override the automatic version.

In addition you *must* pass the flag `--enable-pkgbuild` to gretl's `configure` script, plus several path options (referencing `bindir`, `libdir`, `localedir` and `datarootdir`) so that the installation mimics the relatively "flat" directory structure of our gretl packages for Windows.

Listing 3 shows `pkgconf.sh`, as downloaded and called by `pkgbuild.sh`. You may wish to customize this.

The flag `--disable-avx` is optional, but here's the point of it. If your machine supports AVX (Intel/AMD's "Advanced Vector Extensions") then by default gretl will be compiled to take advantage of that facility. That's good in itself, but it means that your gretl build *will not run* on any machine that doesn't support AVX. So disabling AVX support is the safe option if you plan to distribute your package to others.

6.2 Clean-up and first test

Having successfully configured the build, you then do

```
make && make install && make install-doc
```

as usual. In a further step, you should delete from the installation several files that are not required in a runtime package (import libraries, headers and so on). This can done by invoking the auto-generated script named `post-install.sh`:

```
bash win32/windist/post-install.sh
```

At this point you should have an installation that works under MSYS2. To verify this you can execute the following:

```
GRETL_HOME=$PREFIX $PREFIX/gretl.exe
```

(with PREFIX defined appropriately, of course).

6.3 Copying run-time files and second test

For a build that is independent of MSYS2 it's necessary to copy all the required DLLs (and other runtime files) into the `gretl` directory. This can be accomplished using the auto-generated script `install-runtime.sh` which can be found in the `win32/windist` subdirectory of your gretl build tree. You may wish to take a look at this script before calling it to check that it meets your case OK.

Please note that among other things, this script attempts to download, and unpack into the gretl package directory, a suitable build of gnuplot for Windows. If you would prefer to handle this yourself, edit the line in `install-runtime.sh` that says:

```
INSTALL_GNUPLOT="yes"
```

Either way, however, you must install a working `wgnuplot.exe` into your package directory or the package will not be complete.

After installing the run-time files you should check that gretl can be run independently of MSYS2. To do this, open a `cmd.exe` window and execute something like the following (depending on your installation path):

```
set GRETL_HOME=$USERPROFILE\gbuild\gretl
$GRETL_HOME\gretl.exe
```

(The $USERPROFILE reference corresponds to the recommended installation prefix; change it if you didn't go with the recommendation.)

It's possible that Windows may complain about certain DLLs being missing. If so, the script `install-runtime.sh` has not been entirely successful in detecting the dependencies of your gretl build; this is a bug and should be reported as such. However, you can proceed by copying in the missing DLLs from MSYS2 until Windows stops complaining and `gretl.exe` runs.

6.4 Create and run Inno Setup script

The final step — once gretl is verified as running independently of MSYS2 — is to create and run a suitable Inno Setup script. This is automated by the script `mkdist.sh`, which can be found at https://gretl.sourceforge.net/winbuild/mkdist.sh. This should be executed in an MSYS2 terminal window, in the directory immediately above the gretl "pkgbuild" installation, for example

```
/c/users/yourname/gbuild
```

If `mkdist.sh` runs successfully, the installer `exe` should be found in the `Output` subdirectory of the directory in which the script was run. In case of failure, there should be some relevant information in a file named `errlog`.

6.5 Updating a package build

Not all of the steps described above have to be repeated if you are updating a previously successful build. Here's a typical update routine:

1. Run `git pull` in your gretl sources directory.
2. If you wish to make any local changes to the code, go ahead and do so.
3. Rerun gretl's `configure` script.
4. To be on the safe side, run `make clean` in your gretl build directory.
5. Do `make && make install && make install-doc`.
6. Rerun `win32/winbuild/post-install.sh` for clean-up.
7. Change directory to the parent of your gretl installation and rerun `mkdist.sh`.

You won't have to rerun `install-runtime.sh` unless there has been a change in gretl's dependencies since your last build—but in that case update the runtime files before running `mkdist.sh`.

Appendices

A Shell scripts

Here we give a summary of the location, purpose and context of the various shell scripts referenced above.

The following scripts can be found on the server at https://gretl.sourceforge.net/winbuild/:

(a) General purpose

`setup.sh`	All-in-one script for initial gretl-build setup
`texinst.sh`	Installs `w32tex`; called by `setup.sh` but may be used independently
`add-repo.sh`	Adds gretl repository to `pacman` configuration file; called by `setup.sh` but may be used independently

(b) Specific to "own use" build (to be run under MSYS2)

`conf.sh`	Configuration driver

(c) Specific to distributable "package build"

`pkgbuild.sh`	All-in-one script for package build (initial setup is assumed)
`pkgconf.sh`	Configuration driver for package build; called by `pkgbuild.sh` but may be used independently
`mkdist.sh`	Creates installer `exe` (assumes Inno Setup installed)

The following are auto-generated scripts, which can be found under your `build` directory after running `configure`. The first one may be used to create a desktop shortcut to an "own use" build of gretl under MSYS2. The other three pertain specifically to making a package build; they are called by `pkgbuild.sh` but may also be used independently.

(a) Specific to "own use" build (in `win32`)

`mkshortcut.sh`	make desktop shortcut to gretl

(b) Specific to package build (in `win32/windist`)

`post-install.sh`	Clean up files that are not needed in package
`install-runtime.sh`	Copy run-time files into gretl directory for packaging
`mkiss.sh`	Create `gretl.iss` file for use by Inno Setup

B pacman cheat sheet

Here are some commonly used pacman idioms.

- Refresh local cache of repo information:
  `pacman -Sy`
- List installed packages with their versions:
  `pacman -Q`
- List installed packages without their version info:
  `pacman -Qq`
- Sync (install, reinstall or upgrade) a package from repository:
  `pacman -S <pkg>`
- Install (upgrade) package using local <pkgfile>:
  `pacman -U <pkgfile>`
- Update all installed packages, as needed:
  `pacman -Syu`
- Remove an installed package:
  `pacman -R <pkg>`
- Get a listing of files owned by a package:
  `pacman -Ql <pkg>`
- List “orphaned” packages (unused dependencies):
  `pacman -Qtdq`
- Remove all “orphaned” packages:
  `pacman -Rns $(pacman -Qtdq)`
- Query repository for packages providing a specific content:
  `pacman -Ss <content>`