Wine User Guide

Wine User Guide

Table of Contents

Chapter 1. Introduction

Overview / About

Purpose of this document and intended audience

This document, called the Wine User Guide, is both an easy installation guide and an extensive reference guide. This guide is for both the new Wine user and the experienced Wine user, offering full step-by-step installation and configuration instructions, as well as featuring extensive reference material by documenting all configuration features and support areas.

Further questions and comments

If, after examining this guide, the FAQ (http://wiki.winehq.org/FAQ), and other relevant documentation there is still something you cannot figure out, we would love to hear from you. The mailing lists (http://www.winehq.org/site/forums) section contains several mailing lists and an IRC channel, all of which are great places to seek help and offer suggestions. If you are particularly savvy, and believe that something can be explained better, you can file a bug report (http://bugs.winehq.org/) or post a patch (http://wiki.winehq.org/SubmittingPatches) on Wine documentation itself.

Content overview / Steps to take

In order to be able to use Wine, you must first have a working installation. This guide will help you to move your system from an empty, Wineless void to one boasting a fresh, up to date Wine install. The first step, Getting Wine, illustrates the various methods of getting Wine's files onto your computer. The second step, Configuring Wine, shows how to customize a Wine installation depending on your individual needs. The final step, Running Wine, covers the specific steps you can take to get a particular application to run better under Wine, and provides useful links in case you need further help.

Quick start

The process of installing and running Wine can be summarised as follows:

- Get a distribution as indicated in Getting Wine and see the Wine Downloads (http://www.winehq.org/site/download) page. For the casual or new user the simplest is to get a packaged version for your distribution.
- Optionally configure Wine using the **winecfg** command. Wine should work without any additional configuration options.
- To test your installation run Wine notepad clone using the **`wine notepad`** command.
- Check the Wine AppDB (http://appdb.winehq.org) for specific instructions or steps required to install or run your application.
- Run Wine using the **`wine`** ***`path/to/appname.exe`*** command.

  The first command you will run will be to install an application. Typically something like **`wine /media/cdrom/setup.exe`** or the equivalent path might be used to install an application from CD.

What is Wine?

Windows and Linux

Different software programs are designed for different operating systems, and most won't work on systems that they weren't designed for. Windows programs, for example, won't run in Linux because they contain instructions that the system can't understand until they're translated by the Windows environment. Linux programs, likewise, won't run under the Windows operating system because Windows is unable to interpret all of their instructions.

This situation presents a fundamental problem for anyone who wants to run software for both Windows and Linux. A common solution to this problem is to install both operating systems on the same computer, known as "dual booting". When a Windows program is needed, the user boots the machine into Windows to run it; when a Linux program is then needed, the user then reboots the machine into Linux. This option presents great difficulty: not only must the user endure the frustration of frequent rebooting, but programs for both platforms can't be run simultaneously. Having Windows on a system also creates an added burden: the software is expensive, requires a separate disk partition, and is unable to read most filesystem formats, making the sharing of data between operating systems difficult.

What is Wine, and how can it help me?

Wine makes it possible to run Windows programs alongside any Unix-like operating system, particularly Linux. At its heart, Wine is an implementation of the Windows Application Programing Interface (API) library, acting as a bridge between the Windows program and Linux. Think of Wine as a compatibility layer, when a Windows program tries to perform a function that Linux doesn't normally understand, Wine will translate that program's instruction into one supported by the system. For example, if a program asks the system to create a Windows pushbutton or text-edit field, Wine will convert that instruction into its Linux equivalent in the form of a command to the window manager using the standard X11 protocol.

If you have access to the Windows program source code, Wine can also be used to recompile a program into a format that Linux can understand more easily. Wine is still needed to launch the program in its recompiled form, however there are many advantages to compiling a Windows program natively within Linux. For more information, see the Winelib User Guide.

Wine features

Throughout the course of its development, Wine has continually grown in the features it carries and the programs it can run. A partial list of these features follows:

- Support for running Win64, Win32 (Win 95/98, NT/2000/XP/2003/Vista/2008/7/2012/10), Win16 (Win 3.1) and DOS programs
- Optional use of external vendor DLL files (such as those included with Windows)
- X11-based graphics display, allowing remote display to any X terminal, as well as a text mode console
- Desktop-in-a-box or mixable windows
- DirectX support for games
- Good support for various sound drivers including OSS and ALSA

- Support for alternative input devices such as graphics tablets.
- Printing: PostScript interface driver (`psdrv`) to standard Unix PostScript print services
- Modem, serial device support
- Winsock TCP/IP networking support
- ASPI interface (SCSI) support for scanners, CD writers, and other devices
- Advanced Unicode and foreign language support
- Full-featured Wine debugger and configurable trace logging messages for easier troubleshooting

Versions of Wine

Wine from WineHQ

Wine is an open source project, and there are accordingly many different versions of Wine for you to choose from. The standard version of Wine comes in intermittent releases (roughly twice a month), and can be downloaded over the Internet in both prepackaged binary form and ready to compile source code form. Alternatively, you can install a development version of Wine by using the latest available source code from the Git repository. See the next chapter, Getting Wine, for further details.

Other Versions of Wine

There are a number of programs that are derived from the standard Wine codebase in some way or another. Some of these are commercial products from companies that actively contribute to the Wine project.

These products try to stand out or distinguish themselves from the standard version of Wine by offering greater compatibility, easier configuration, and commercial support. If you require such things, it is a good idea to consider purchasing these products.

Table 1-1. Various Wine offerings

Product	Description	Distribution Form
CodeWeavers CrossOver Office (http://www.codeweavers.com/products/office)	CrossOver Office allows you to install your favorite Windows productivity applications in Linux, without needing a Microsoft Operating System license. CrossOver includes an easy to use, single click interface, which makes installing a Windows application simple and fast.	Commercial; 30-day fully-functional demo available.

Alternatives to Wine you might want to consider

There are many ways to run software other than through Wine. If you are considering using Wine to run an application you might want to think about the viability of these approaches if you encounter difficulty.

Native Applications

Instead of running a particular Windows application with Wine, one frequently viable alternative is to simply run a different application. Many Windows applications, particularly more commonly used ones such as media players, instant messengers, and filesharing programs have very good open source equivalents. Furthermore, a sizable number of Windows programs have been ported to Linux directly, eliminating the need for Wine (or Windows) entirely. These alternatives should be found through your system package management facilities.

Another Operating System

Probably the most obvious method of getting a Windows application to run is to simply run it on Windows. However, security, license cost, backward-compatibility, and machine efficiency issues can make this a difficult proposition, which is why Wine is so useful in the first place.

Another alternative is to use ReactOS (http://www.reactos.com), which is a fully open source alternative to Windows. ReactOS shares code heavily with the Wine project, but rather than running Windows applications on top of Linux they are instead run on top of the ReactOS kernel. ReactOS also offers compatibility with Windows driver files, allowing the use of hardware without functional Linux drivers.

Virtual Machines

Rather than installing an entirely new operating system on your machine, you can instead run a virtual machine at the software level and install a different operating system on it. Thus, you could run a Linux system and at the same time run Windows along with your application in a virtual machine simultaneously on the same hardware. Virtual machines allow you to install and run not only different versions of Windows on the same hardware, but also other operating systems, including ReactOS.

There are several different virtual machine offerings out there, and some are also able to emulate x86 hardware on different platforms. The open source Bochs (http://bochs.sourceforge.net/), VirtualBox (http://www.virtualbox.org/) and QEMU (http://www.qemu.org/) can run both Windows and ReactOS virtually. Other, commercial virtual machine offerings include VMware (http://www.vmware.com/) and Microsoft's VirtualPC (http://www.microsoft.com/windows/virtualpc/).

There are significant drawbacks to using virtual machines, however. Unlike Wine, such programs *are* emulators, so there is an inevitable speed decrease which can be quite substantial. Furthermore, running an application inside a virtual machine prevents fully integrating the application within the current environment. You won't, for example, be able to have windows system tray icons or program shortcuts sitting alongside your desktop Linux ones, since instead the Windows applications must reside completely within the virtual machine.

Chapter 2. Getting Wine

Wine Installation Methods

Once you've decided that Wine is right for your needs, the next step is to decide how you want to install it. There are three methods for installing Wine from WineHQ, each with their own advantages and disadvantages.

Installation from a package

By far the easiest method for installing Wine is to use a prepackaged version of Wine. These packages contain ready-to-run Wine binary files specifically compiled for your distribution, and they are tested regularly by the packagers for both functionality and completeness.

Packages are the recommended method for installing Wine. We make them easily available at the WineHQ downloads page (http://www.winehq.org/site/download), and these are always the latest packages available. Being popular, Wine packages can also be found elsewhere in official distribution repositories. These can, however, sometimes be out of date, depending on the distribution. Packages are easily upgradable as well, and many distributions can upgrade Wine seamlessly with a few clicks. Building your own installable binary package from a source package is also possible, although it is beyond the scope of this guide.

Installation from a source archive

Sometimes the Wine packages don't fit your needs exactly. Perhaps they're not available for your architecture or distribution, or perhaps you want to build Wine using your own compiler optimizations or with some options disabled, or perhaps you need to modify a specific part of the source code before compilation. Being an open source project, you are free to do all of these things with Wine source code, which is provided with every Wine release. This method of installation can be done by downloading a Wine source archive and compiling from the command line. If you are comfortable with such things and have special needs, this option may be for you.

Getting Wine source archives is simple. Every release, we put a source package in compressed `tar.bz2` format at the WineHQ downloads page (http://www.winehq.org/site/download). Compiling and installing Wine from source is slightly more difficult than using a package, however we will cover it in depth and attempt to hold your hand along the way.

Installation from a Git tree

If you wish to try out the bleeding edge of Wine development, or would even like to help develop Wine yourself, you can download the very latest source code from our Git repository. Instructions for downloading from the Wine Git repository are available at http://www.winehq.org/site/git.

Please take note that the usual warnings for using a developmental version still apply. The source code on the Git repository is largely untested and may not even compile properly. It is, however, the best way to test out how Wine will work in the next release, and if you're modifying source code it's best to get the latest copy. The Git repository is also useful for application maintainers interested in testing if an application will still work right for the next release, or if a recent patch actually improves things. If you're interested in helping us to get an application working in Wine, see the HowTo (http://www.winehq.org/site/howto).

Installing Wine from a package

Installing a fresh package

Installing a package on a fresh system is remarkably straightforward. Simply download and install the package using whatever utility your distribution provides. There is usually no need to explicitly remove old packages before installing, as modern Linux distributions should upgrade and replace them automatically. If you installed Wine from source code, however, you should remove it before installing a Wine package. See the section on uninstalling Wine from source for proper instructions.

Different Distributions

Wine works on a huge amount of different Linux distributions, as well other Unix-like systems such as Solaris and FreeBSD, each with their own specific way of installing and managing packages. Fortunately, however, the same general ideas apply to all of them, and installing Wine should be no more difficult than installing any other software, no matter what distribution you use. Uninstalling Wine packages is simple as well, and in modern Linux distributions is usually done through the same easy interface as package installation.

We won't cover the specifics of installing or uninstalling Wine packages among the various systems' methods of packaging and package management in this guide, however, up to date installation notes for particular distributions can be found at the WineHQ website in the HowTo (http://www.winehq.org/site/howto). If you need further help figuring out how to simply install a Wine package, we suggest consulting your distribution's documentation, support forums, or IRC channels.

Installing Wine from source

Before installing Wine from source, make sure you uninstall any Wine binary packages you may have on your system. Installing from source requires use of the terminal window as well as a full copy of the Wine source code. Once having downloaded the source from Git or extracted it from an archive, navigate to it using the terminal and then follow the remaining steps.

Getting the Build Dependencies

Wine makes use of many open source libraries during its operation. While Wine is not strictly dependent on these libraries and will compile without most of them, much of Wine functionality is improved by having them available at compile time. In the past, many user problems were caused by people not having the necessary development libraries when they built Wine from source; because of this reason and others, we highly recommend installing via binary packages or by building source packages which can automatically satisfy their build dependencies.

If you wish to install build dependencies by hand, there are several ways to see if you're missing some useful development libraries. The most straightforward approach is to watch the output of **configure** before you compile Wine and see if anything important is missing; if it is, simply install what's missing and rerun **configure** before compiling. You can also check the file **configure** generates (`include/config.h`) and see what files **configure** is looking for but not finding.

Compiling Wine

Once you've installed the build dependencies you need, you're ready to compile the package. In the terminal window, after having navigated to the Wine source tree, run the following commands:

```
$ ./configure
```

```
$ make depend
$ make
# make install
```

The last command requires root privileges. Although you should never run Wine as root, you will need to install it this way.

Uninstalling Wine from Source

To uninstall Wine from source, once again navigate to the same source folder that you used to install Wine using the terminal. Then, run the following command:

```
# make uninstall
```

This command will require root privileges, and should remove all of the Wine files from your system. It will not, however, remove your Wine configuration and applications located in your user home directory, so you are free to install another version of Wine or delete that configuration by hand.

Chapter 3. Running Wine

This chapter will describe all aspects of running Wine, like e.g. basic Wine invocation, command line parameters of various Wine support programs etc.

Basic usage: applications and control panel applets

Applications are installed under Wine the same way you would in Windows: by running the installer. You can just accept the defaults for where to install, most installers will default to `C:\Program Files`, which is fine. If the application installer requests it, you may find that Wine creates icons on your desktop and in your app menu. If that happens, you can start the app by clicking on them.

The standard way to uninstall things is for the application to provide an uninstaller, usually registered with the “Add/Remove Programs” control panel applet. To access the Wine equivalent, run the **uninstaller** program (it is located in the `programs/uninstaller/` directory in a Wine source directory) in a *terminal*:

```
$ wine uninstaller
```

Some programs install associated control panel applets, examples of this would be Internet Explorer and QuickTime. You can access the Wine control panel by running in a *terminal*:

```
$ wine control
```

which will open a window with the installed control panel applets in it, as in Windows.

If the application doesn't install menu or desktop items, you'll need to run the app from the command line. Remembering where you installed to, something like:

```
$ wine "C:\Program Files\appname\appname.exe"
```

will probably do the trick. The path isn't case sensitive, but remember to include the double quotes. Some programs don't always use obvious naming for their directories and executable files, so you might have to look inside the `Program Files` directory to see what was put where.

How to run Wine

You can simply invoke the **wine** command to get a small help message:

```
Usage: wine PROGRAM [ARGUMENTS...]   Run the specified program
       wine --help                   Display this help and exit
       wine --version                Output version information and exit
```

The first argument should be the name of the file you want **wine** to execute. If the executable is in the Path environment variable, you can simply give the executable file name. However, if the executable is not in Path, you must give the full path to the executable. For example, given a Path of the following:

```
Path="c:\windows;c:\windows\system;e:\;e:\test;f:\"
```

You could run the file `c:\windows\system\foo.exe` with:

```
$ wine foo.exe
```

However, you would have to run the file `c:\myapps\foo.exe` with this command:

```
$ wine c:\\myapps\\foo.exe
```

(note the backslash-escaped '\') or:

```
$ wine ~/.wine/drive_c/myapps/foo.exe
```

For details on running text mode (CUI) executables, read the section below.

Explorer-like graphical Wine environments

If you prefer using a graphical interface to manage your files you might want to consider using **winefile**. This Winelib application comes with Wine and can be found with the other Wine programs. It is a useful way to view your drive configuration and locate files, plus you can execute programs directly from **winefile**. Please note that many functions are not yet implemented.

Wine Command Line Options

`--help`

Shows a small command line help page.

`--version`

Shows the Wine version string. Useful to verify your installation.

Environment variables

WINEDEBUG=*`channels`*

Wine isn't perfect, and many Windows applications still don't run without bugs under Wine (but then, a lot of programs don't run without bugs under native Windows either!). To make it easier for people to track down the causes behind each bug, Wine provides a number of *debug channels* that you can tap into.

Each debug channel, when activated, will trigger logging messages to be displayed to the console where you invoked **wine**. From there you can redirect the messages to a file and examine it at your leisure. But be forewarned! Some debug channels can generate incredible volumes of log messages. Among the most prolific offenders are `relay` which spits out a log message every time a win32 function is called, `win` which tracks windows message passing, and of course `all` which is an alias for every single debug channel that exists. For a complex application, your debug logs can easily top 1 MB and higher. A `relay` trace can often generate more than 10 MB of log messages, depending on how long you run the application. (You'll want to check out `RelayExclude` registry key to modify what the `relay` trace reports). Logging does slow down Wine quite a bit, so don't use WINEDEBUG unless you really do want log files.

Within each debug channel, you can further specify a *message class*, to filter out the different severities of errors. The four message classes are: `trace`, `fixme`, `warn`, `err`.

To turn on a debug channel, use the form *`class+channel`*. To turn it off, use *`class-channel`*. To list more than one channel in the same WINEDEBUG option, separate them with commas. For example, to request `warn` class messages in the `heap` debug channel, you could invoke **wine** like this:

```
$ WINEDEBUG=warn+heap wine program_name
```

If you leave off the message class, Wine will display messages from all four classes for that channel:

```
$ WINEDEBUG=heap wine program_name
```

If you wanted to see log messages for everything except the `relay` channel, you might do something like this:

```
$ WINEDEBUG=+all,-relay wine program_name
```

You can find a list of the debug channels and classes at DebugChannels (http://wiki.winehq.org/DebugChannels). More channels will be added to (or subtracted from) later versions.

For more details about debug channels, check out the The Wine Developer's Guide (http://www.winehq.org/docs/winedev-guide/).

WINEDLLOVERRIDES=*DLL Overrides*

It's not always possible to run an application on builtin DLLs. Sometimes native DLLs simply work better. Although these DLL overrides can be set using **winecfg** you might want to use the WINEDLLOVERRIDES environment variable to set them.

Examples:

```
$ WINEDLLOVERRIDES="comdlg32,shell32=n,b" wine program_name
```

Try to load `comdlg32` and `shell32` as native Windows DLLs first and try the builtin version if the native load fails.

```
$ WINEDLLOVERRIDES="comdlg32,shell32=n;c:\\foo\\bar\\baz=b" wine program_name
```

Try to load the `comdlg32` and `shell32` libraries as native windows DLLs. Furthermore, if an application requests to load `c:\foo\bar\baz.dll` load the builtin library `baz`.

```
$ WINEDLLOVERRIDES="comdlg32=b,n;shell32=b;comctl32=n;oleaut32=" wine program_name
```

Try to load `comdlg32` as builtin first and try the native version if the builtin load fails; load `shell32` always as builtin and `comctl32` always as native. `oleaut32` will be disabled.

For more information about DLL overrides, please refer to the DLL overrides section of this guide.

WINEARCH

Specifies the Windows architecture to support. It can be set either to `win32` (support only 32-bit applications), or to `win64` (support both 64-bit applications and 32-bit ones in WoW64 mode). The architecture supported by a given Wine prefix is set at prefix creation time and cannot be changed afterwards. When running with an existing prefix, Wine will refuse to start if WINEARCH doesn't match the prefix architecture.

OSS Audio Driver Settings

If you are using the OSS audio driver and you have multiple devices, (i.e. `/dev/dsp*`, `/dev/mixer*`) you can specify which one you want to use with the following environment variables:

- **AUDIODEV=*audio device***
- **MIXERDEV=*mixer device***
- **MIDIDEV=*MIDI device***

As an example:

```
$ AUDIODEV=/dev/dsp4 MIXERDEV=/dev/mixer1 MIDIDEV=/dev/midi3 wine program_name
```

wineserver Command Line Options

wineserver usually gets started automatically by Wine whenever the first Wine process gets started. However, **wineserver** has some useful command line options that you can add if you start it up manually, e.g. via a user login script or so.

-d *n*

Sets the debug level for debug output in the terminal that **wineserver** got started in at level *n*. In other words: everything greater than 0 will enable **wineserver** specific debugging output.

-h

Display **wineserver** command line options help message.

-k [*n*]

Kill the current **wineserver**, optionally with signal *n*.

-p [*n*]

This parameter makes **wineserver** persistent, optionally for *n* seconds. It will prevent **wineserver** from shutting down immediately.

Usually, **wineserver** quits almost immediately after the last Wine process using this **wineserver** terminated. However, since **wineserver** loads a lot of things on startup (such as the whole Windows registry data), its startup might be so slow that it's very useful to keep it from exiting after the end of all Wine sessions, by making it persistent.

-w

This parameter makes a newly started **wineserver** wait until the currently active instance terminates.

Setting Windows/DOS environment variables

Your program might require some environment variable to be set properly in order to run successfully. In this case you need to set this environment variable in the Linux shell, since Wine will pass on the entire shell environment variable settings to the Windows environment variable space. Example for the **bash** shell (other shells may have a different syntax!):

```
export MYENVIRONMENTVAR=myenvironmentvarsetting
```

This will make sure your Windows program can access the MYENVIRONMENTVAR environment variable once you start your program using Wine. If you want to have MYENVIRONMENTVAR set permanently, then you can place the setting into `/etc/profile`, or also `~/.bashrc` when using **bash**.

Note however that there are some exceptions to the rule: If you want to change the PATH, SYSTEM or TEMP variables, then of course you can't modify it that way, since this will alter the Unix environment settings. Instead, you should set them into the registry. To set them you should launch **`wine regedit`** and then go to the

```
HKEY_CURRENT_USER/Environment
```

key. Now you can create or modify the values of the variables you need

```
"System" = "c:\\windows\\system"
```

This sets up where the Windows system files are. The Windows system directory should reside below the directory used for the `Windows` setting. Thus when using `/usr/local/wine_c_windows` as Windows path, the system directory would be `/usr/local/wine_c/windows/system`. It must be set with no trailing slash, and you must be sure that you have write access to it.

```
"Temp" = "c:\\temp"
```

This should be the directory you want your temp files stored in, `/usr/local/wine_c/temp` in our previous example. Again, no trailing slash, and *write access*!!!

```
"Path" = "c:\\windows;c:\\windows\\system;c:\\blanco"
```

Behaves like the PATH setting on UNIX boxes. When Wine is run like **`wine sol.exe`**, if `sol.exe` resides in a directory specified in the `Path` setting, Wine will run it (of course, if `sol.exe` resides in the current directory, Wine will run that one). Make sure it always has your `windows` directory and system directory (for this setup, it must contain `"c:\\windows;c:\\windows\\system"`).

Text mode programs (CUI: Console User Interface)

Text mode programs are program which output is only made out of text (surprise!). In Windows terminology, they are called CUI (Console User Interface) executables, by opposition to GUI (Graphical User Interface) executables. Win32 API provide a complete set of APIs to handle this situation, which goes from basic features like text printing, up to high level functionalities (like full screen editing, color support, cursor motion, mouse support), going through features like line editing or raw/cooked input stream support

Given the wide scope of features above, and the current usage in Un*x world, Wine comes out with three different ways for running a console program (aka a CUI executable):

- bare streams
- **wineconsole** with `user` backend
- **wineconsole** with `curses` backend

The names here are a bit obscure. “bare streams” means that no extra support of Wine is provided to map between the Unix console access and Windows console access. The two other ways require the use of a specific Wine program (**wineconsole**) which provide extended facilities. The following table describes what you can do (and cannot do) with those three ways.

Table 3-1. Basic differences in consoles

Function	Bare streams	Wineconsole with `user` backend	Wineconsole with `curses` backend
How to run (assuming executable is called `foo.exe`)	`$ wine foo.exe`	`$ wineconsole -- --backend=user foo.exe`	`$ wineconsole foo.exe` You can also use *`--backend=curses`* as an option
Good support for line oriented CUI applications (which print information line after line)	Yes	Yes	Yes

Function	Bare streams	Wineconsole with `user` backend	Wineconsole with `curses` backend
Good support for full screen CUI applications (including but not limited to color support, mouse support...)	No	Yes	Yes
Can be run even if X11 is not running	Yes	No	Yes
Implementation	Maps the standard Windows streams to the standard Unix streams (`stdin`/`stdout`/`stderr`)	**wineconsole** will create a new window (hence requiring the `user32` DLL is available) where all information will be displayed	**wineconsole** will use existing Unix console (from which the program is run) and with the help of the `curses`/`ncurses` library take control of all the terminal surface for interacting with the user
Known limitations			Will produce strange behavior if two (or more) Windows consoles are used on the same Un*x terminal.

Configuration of CUI executables

When **wineconsole** is used, several configuration options are available. Wine (as Windowses do) stores, on a per application basis, several options in the registry. This let a user, for example, define the default screen-buffer size he would like to have for a given application.

As of today, only the `user` backend allows you to edit those options (we don't recommend editing by hand the registry contents). This edition is fired when a user right clicks in the console (this popups a menu), where you can either choose from:

- Default: this will edit the settings shared by all applications which haven't been configured yet. So, when an application is first run (on your machine, under your account) in **wineconsole**, it will inherit these default settings. Afterwards, the application will have its own settings, that you'll be able to modify at your will.

  Properties: this will edit the application's settings. When you're done, with the edition, you'll be prompted whether you want to:

  1. Keep these modified settings only for this session (next time you run the application, you will not see the modification you've just made).
  2. Use the settings for this session and save them as well, so that next you run your application, you'll use these new settings again.

Here's the list of the items you can configure, and their meanings:

Table 3-2. Wineconsole configuration options

Configuration option	Meaning
Cursor size	Defines the size of the cursor. Three options are available: small (33% of character height), medium (66%) and large (100%)
Popup menu	It's been said earlier that **wineconsole** configuration popup was triggered using a right click in the console's window. However, this can be an issue when the application you run inside **wineconsole** expects the right click events to be sent to it. By ticking Control or Shift you select additional modifiers on the right click for opening the popup. For example, ticking Shift will send events to the application when you right click the window without **Shift** being hold down, and open the window when you right-click while **Shift** being hold down.
Quick edit	This tick box lets you decide whether left-click mouse events shall be interpreted as events to be sent to the underlying application (tick off) or as a selection of rectangular part of the screen to be later on copied onto the clipboard (tick on).
History	This lets you pick up how many commands you want the console to recall. You can also drive whether you want, when entering several times the same command - potentially intertwined with others - whether you want to store all of them (tick off) or only the last one (tick on).
Police	The Police property sheet allows you to pick the default font for the console (font file, size, background and foreground color).
Screenbuffer & window size	The console as you see it is made of two different parts. On one hand there's the screenbuffer which contains all the information your application puts on the screen, and the window which displays a given area of this screen buffer. Note that the window is always smaller or of the same size than the screen buffer. Having a strictly smaller window size will put on scrollbars on the window so that you can see the whole screenbuffer content.
Close on exit	If it's ticked, then **wineconsole** will exit when the application within terminates. Otherwise, it'll remain opened until the user manually closes it: this allows seeing the latest information of a program after it has terminated.

Configuration option	Meaning
Edition mode	When the user enter commands, he or she can choose between several edition modes: • `Emacs`: the same keybindings as under **emacs** are available. For example, **Ctrl-A** will bring the cursor to the beginning of the edition line. See your **emacs** manual for the details of the commands. • `Win32`: these are the standard Windows console key-bindings (mainly using arrows).

Chapter 4. Configuring Wine

Most of the most common configuration changes can be done with the **winecfg** tool. We'll go through an easy, step-by-step introduction to **winecfg** and outline the options available. In the next section we'll go over more advanced changes you can make using **regedit** as well as provide a complete reference to all Wine configuration settings. Finally, some things you might want to configure fall out of the scope of **winecfg** and **regedit**, and we'll go over those.

Using Winecfg

In the past, Wine used a special configuration file that could be found in `~/.wine/config`. If you are still using a version of Wine that references this file (older than June, 2005) you should upgrade before doing anything else. All settings are now stored directly in the registry and accessed by Wine when it starts.

Winecfg should have been installed on your computer along with the rest of the Wine programs. If you can't figure out how to start it, try running **winecfg**.

When the program starts you'll notice there are tabs along the top of the window for:

- Applications
- Libraries
- Graphics
- Desktop Integration
- Drives
- Audio
- About

Changing settings in the Applications and Libraries tabs will have the most impact on getting an application to run. The other settings focus on getting Wine itself to behave the way you want it to.

Note: The Applications, Libraries, and Graphics tabs are linked together! If you have "Default Settings" selected under Applications, all of the changes made within the Libraries and Graphics tabs will be changed for all applications. If you've configured a specific application under the Applications tab and have it selected, then any changes made in Libraries or Graphics will affect only that application. This allows for custom settings for specific applications.

Application Settings

Wine has the ability to mimic the behavior of different versions of Windows. In general, the biggest difference is whether Wine behaves as a Win9x version or an NT version. Some applications require a specific behavior in order to function and changing this setting may cause a buggy app to work. Wine default Windows version is Windows XP. Some applications may perform better if you choose Windows 98.

Within the tab you'll notice there is a *Default Settings* entry. If you select that you'll see the current default *Windows Version* for all applications. A troublesome application is best configured separately from the Default Settings. To do that:

1. Click on the Add application button.
2. Browse until you locate the executable.

3. After it's been added you can choose the specific Windows version Wine will emulate for that application.

Libraries Settings

Likewise, some applications require specific libraries in order to run. Wine reproduces the Windows system libraries (so-called native DLLs) with completely custom versions designed to function exactly the same way but without requiring licenses from Microsoft. Wine has many known deficiencies in its built-in versions, but in many instances the functionality is sufficient. Using only builtin DLLs ensures that your system is Microsoft-free. However, Wine has the ability to load native Windows DLLs.

DLL Overrides

It's not always possible to run an application on builtin DLLs, so sometimes native versions will be recommended as a workaround for a specific problem. Some may be directly copied to the directory configured as `c:\windows\system32` (more on that in the drives section) while others may require an installer, see the next section on **winetricks**. Native versions of these DLLs do not work: `kernel32.dll`, `gdi32.dll`, `user32.dll`, and `ntdll.dll`. These libraries require low-level Windows kernel access that simply doesn't exist within Wine.

With that in mind, once you've copied the DLL you just need to tell Wine to try to use it. You can configure Wine to choose between native and builtin DLLs at two different levels. If you have *Default Settings* selected in the Applications tab, the changes you make will affect all applications. Or, you can override the global settings on a per-application level by adding and selecting an application in the Applications tab.

To add an override for `FOO.DLL`, enter **`FOO`** into the box labeled New override for library: and click on the Add button. To change how the DLL behaves, select it within the Existing overrides: box and choose Edit. By default the new load order will be native Windows libraries before Wine builtin ones (*Native then Builtin*). You can also choose native only, builtin only, or disable it altogether.

DLLs usually get loaded in the following order:

1. The directory the program was started from.
2. The current directory.
3. The Windows system directory.
4. The Windows directory.
5. The PATH variable directories.

Notes About System or Missing DLLs

There are of course DLLs that Wine does not currently implement very well (or at all).

In case Wine complains about a missing DLL, you should check whether this file is a publicly available DLL or a custom DLL belonging to your program. In the latter case, check that you have installed your program correctly.

Most often applications will assume that a required redistributable package has already been installed and subsequently fail to run when the required dependencies are not met. For example:

```
err:module:import_dll Library MFC42.DLL (which is needed by L"C:\\Program Files\\Foo\\Bar.dll") no
```

Redistributable packages which install the necessary runtimes can be obtained through the use of winetricks (http://wiki.winehq.org/winetricks). Note these components are subject to their own license and are not part of the Wine project. You should refer to the application's AppDB (http://appdb.winehq.org) entry for advice on what is required.

Graphics Settings

There are basically five different graphics settings you can configure. For most people the defaults are fine.

The first setting primarily affect games and is somewhat self-explanatory. You can prevent the mouse from leaving the window of a full-screen program (e.g. a game) and the default is to not have that box checked. That is mostly needed when using a virtual desktop.

You may find it helpful to tick Emulate a virtual desktop. In this case, all programs will run in a separate window. You may find this useful as a way to test buggy games that change (possibly unsuccessfully) the screen resolution. Confining them to a window can allow for more control over them at the possible expense of decreased usability. Sizes you might want to try are 800x600 (the default) or 1024x768.

Drive Settings

Windows requires a fairly rigid drive configuration that Wine imitates. Most people are familiar with the standard notation of the `A:` drive representing the floppy disk, the `C:` drive representing the primary system disk, etc. Wine uses the same concept and maps those drives to the underlying native filesystem.

Wine drive configuration is relatively simple. In **winecfg** under the Drives tab you'll see buttons to add and remove available drives. When you choose to add a drive, a new entry will be made and a default drive mapping will appear. You can change where this drive points to by changing what's in the Path: box. If you're unsure of the exact path you can choose Browse to search for it. Removing a drive is as easy as selecting the drive and clicking Remove.

winecfg has the ability to automatically detect the drives available on your system. It's recommended you try this before attempting to configure drives manually. Simply click on the Autodetect button to have Wine search for drives on your system.

You may be interested in configuring your drive settings outside of **winecfg**, in which case you're in luck because it's quite easy. All of the drive settings reside in a special directory: `~/.wine/dosdevices`. Each "drive" is simply a link to where it actually resides. Wine automatically sets up two drives the first time you run Wine:

```
$ ls -la ~/.wine/dosdevices/
lrwxrwxrwx  1 wineuser wineuser   10 Jul 23 15:12 c: -> ../drive_c
lrwxrwxrwx  1 wineuser wineuser    1 Jul 23 15:12 z: -> /
```

To add another drive, for example your CD-ROM, just create a new link pointing to it:

```
$ ln -s /mnt/cdrom ~/.wine/dosdevices/d:
```

Take note of the DOS-style naming convention used for links - the format is a letter followed by a colon, such as `a:`. So, if the link to your `c:` drive points to `~/.wine/drive_c`, you can interpret references to `c:\windows\system32` to mean `~/.wine/drive_c/windows/system32`.

Audio Settings

Wine can work with quite a few different audio subsystems. You can see the selected driver that Wine figures out for you under the Audio tab.

You can manually select which device will be used for Output, Input, Voice output and Voice input. For example you can choose the digital output of your sound device instead of the analog one.

Desktop Integration

Wine can load Windows themes if you have them available. While this certainly isn't necessary in order to use Wine or applications, it does allow you to customize the look and feel of a program. Wine supports the newer MSStyles type of themes. Unlike the older Microsoft Plus! style themes, the `uxtheme` engine supports special `.msstyles` files that can retheme all of the Windows controls. This is more or less the same kind of theming that modern Linux desktops have supported for years. If you'd like to try this out:

1. Download a Windows XP theme. Be sure it contains a `.msstyles` file.
2. Use the Desktop Integration tab of **winecfg** to install and select the new theme.

Using the Registry and Regedit

All of the settings you change in **winecfg**, with exception of the drive settings, are ultimately stored in the registry. In Windows, this is a central repository for the configuration of applications and the operating system. Likewise, Wine implements a registry and some settings not found in Winecfg can be changed within it (there's actually more of a chance you'll need to dip into the registry to change the settings of an application than Wine itself).

Now, the fact that Wine itself uses the registry to store settings has been controversial. Some people argue that it's too much like Windows. To counter this there are several things to consider. First, it's impossible to avoid implementing a registry simply because applications expect to be able to store their settings there. In order for Wine to store and access settings in a separate configuration file would require a separate set of code to basically do the same thing as the Win32 APIs Wine already implements.

Registry Structure

Okay... with that out of the way, let's dig into the registry a bit to see how it's laid out. The Windows registry is an elaborate tree structure, and not even most Windows programmers are fully aware of how the registry is laid out, with its different "hives" and numerous links between them; a full coverage is out of the scope of this document. But here are the basic registry keys you might need to know about for now:

`HKEY_LOCAL_MACHINE`

This fundamental root key (in win9x it's stored in the hidden file `system.dat`) contains everything pertaining to the current Windows installation. This is often abbreviated `HKLM`.

`HKEY_USERS`

This fundamental root key (in win9x it's stored in the hidden file `user.dat`) contains configuration data for every user of the installation.

`HKEY_CLASSES_ROOT`

This is a link to `HKEY_LOCAL_MACHINE\Software\Classes`. It contains data describing things like file associations, OLE document handlers, and COM classes.

`HKEY_CURRENT_USER`

This is a link to `HKEY_USERS\`*`your_username`*, i.e., your personal configuration.

Registry Files

Now, what you're probably wondering is how that translates into Wine structure. The registry layout described above actually lives in three different files within each user's `~/.wine` directory:

`system.reg`

This file contains `HKEY_LOCAL_MACHINE`.

`user.reg`

This file contains `HKEY_CURRENT_USER`.

`userdef.reg`

This file contains `HKEY_USERS\.Default` (i.e. the default user settings).

These files are automatically created the first time you use Wine. A set of global settings is stored in the `wine.inf` file and is processed by the **rundll32.exe** program. The first time you run Wine the `wine.inf` file gets processed to populate the initial registry. The registry is also updated automatically if `wine.inf` changes, for instance when upgrading to a newer Wine version.

Note: Older Wine versions (before 1.0) required you to run the **wineprefixcreate** command manually to upgrade your settings. This is no longer necessary.

It is not advisable to edit these files to modify the registry as they are managed by Wine internally. Use **regedit.exe**, **reg.exe** or any program which uses the standard registry functions.

Using Regedit

An easy way to access and change the registry is with the **regedit** tool. Similar to the Windows program it replaces, **regedit** serves to provide a system level view of the registry containing all of the keys. When you start it, you'll immediately notice that the cryptic keys displayed in the text file are organized in a hierarchical fashion.

To navigate through the registry, click on the keys on the left to drill down deeper. To delete a key, click on it and choose Delete from the Edit menu. To add a key or value, locate where you want to put it and choose New from the Edit menu. Likewise, you modify an existing key by highlighting it in the right-hand window pane and choosing Modify from the Edit menu. Another way to perform these same actions is to right-click on the key or value.

Of particular interest to Wine users are the settings stored in `HKEY_CURRENT_USER\Software\Wine`. Most of the settings you change within **winecfg** are written to this area of the registry.

System Administration Tips

With the above file structure, it is possible for a system administrator to configure the system so that a system Wine installation (and applications) can be shared by all the users, and still let the users all have their own personalized configuration. An administrator can, after having installed Wine and any Windows application software he wants the users to have access to, copy the resulting `system.reg` and over to the global registry files (which we assume will reside in `/usr/local/etc` here), with:

```
cd ~root/.wine
cp system.reg /usr/local/etc/wine.systemreg
```

and perhaps even symlink these back to the administrator's account, to make it easier to install apps system-wide later:

```
ln -sf /usr/local/etc/wine.systemreg system.reg
```

You might be tempted to do the same for `user.reg` as well, however that file contains user specific settings. Every user should have their own copy of that file along with the permissions to modify it.

You'll want to pay attention to drive mappings. If you're sharing the `system.reg` file you'll want to make sure the registry settings are compatible with the drive mappings in `~/.wine/dosdevices` of each individual user. As a general rule of thumb, the closer you keep your drive mappings to the default configuration, the easier this will be to manage. You may or may not be able to share some or all of the actual `c:` drive you originally installed the application to. Some applications require the ability to write specific settings to the drive, especially those designed for Windows 95/98/ME.

Note that the **tools/wineinstall** script used to do some of this if you installed Wine source as root, however it no longer does.

A final word of caution: be careful with what you do with the administrator account - if you do copy or link the administrator's registry to the global registry, any user might be able to read the administrator's preferences, which might not be good if sensitive information (passwords, personal information, etc) is stored there. Only use the administrator account to install software, not for daily work; use an ordinary user account for that.

Complete List of Registry Keys

You'll find an up-to-date list of useful registry keys and values (http://wiki.winehq.org/UsefulRegistryKeys) in the wiki.

Other Things to Configure

This section is meant to cover the rest of the things you can configure. It also serves as a collection of tips and tricks to get the most out of using Wine.

Serial and Parallel Ports

Serial and parallel port configuration is very similar to drive configuration - simply create a symbolic link in `~/.wine/dosdevices` with the name of the device. Windows serial ports follow a naming convention of the word `com` followed by a number, such as `com1`, `com2`, etc. Similarly, parallel ports use `lpt` followed by a number, such as `lpt1`. You should link these directly to the corresponding Unix devices, such as `/dev/ttyS0` and `/dev/lp0`. Make sure you have the needed access rights to that device. For example, to configure one serial port and one parallel port, run the following commands:

```
ln -s /dev/ttyS0 com1
ln -s /dev/lp0 lpt1
```

Network Shares

Windows shares can be mapped into the `unc/` directory so anything trying to access `\\myserver\some\file` will look in `~/.wine/dosdevices/unc/myserver/some/file`. For example, if you used Samba to mount `\\myserver\some` on `/mnt/smb/myserver/some` then you can do

```
ln -s /mnt/smb/myserver/some unc/myserver/some
```

to make it available in Wine (don't forget to create the `unc` directory if it doesn't already exist).

Fonts

Font configuration, once a nasty problem, is now much simpler. If you have a collection of TrueType fonts in Windows it's simply a matter of copying the `.ttf` files into `c:\windows\fonts`.

Printers

Wine can interact directly with the local CUPS printing system to find the printers available on your system. Configuring printers with Wine is as simple as making sure your CUPS configuration works. Wine still needs the **lpr** command (from CUPS), when printing a document.

If you do not use CUPS, the old BSD-Printing system is used:

- All Printers from `/etc/printcap` are installed automatically in Wine.
- Wine needs a PPD file for every Printer (`generic.ppd` comes with Wine).
- The **lpr** command is called when printing a document

Scanners

In Windows, scanners use the TWAIN API to access the underlying hardware. Wine builtin TWAIN DLL simply forwards those requests to the Linux SANE libraries. So, to utilize your scanner under Wine you'll first need to make sure you can access it using SANE. After that you'll need to make sure you have **xscanimage** available for use. Currently it is shipped with the `sane-frontends` package but it may not be installed with your distribution. Scanner access is currently known to have problems. If you find it works for you, please consider updating this section of the user guide to provide details on using SANE with Wine.

ODBC Databases

The ODBC system within Wine, as with the printing system, is designed to hook across to the Unix system at a high level. Rather than ensuring that all the Windows code works under Wine it uses a suitable Unix ODBC provider, such as `unixODBC`. Thus if you configure Wine to use the built-in `odbc32.dll`, that Wine DLL will interface to your Unix ODBC package and let that do the work, whereas if you configure Wine to use the native `odbc32.dll` it will try to use the native ODBC32 drivers etc.

Configuring ODBC on Unix

The first step in using a Unix ODBC system with Wine is, of course, to get the Unix ODBC system working itself. This may involve downloading code or binary packages etc. There are several Unix ODBC systems available such as `unixODBC` or an ODBC-ODBC bridge that can be used to access a Microsoft Access database. Typically such systems will include a tool, such as **isql**, which will allow you to access the data from the command line so that you can check that the system is working.

The next step is to hook the Unix ODBC library to the Wine built-in `odbc32` DLL. The built-in `odbc32` (currently) looks to the environment variable LIB_ODBC_DRIVER_MANAGER for the name of the ODBC library. For example:

```
LIB_ODBC_DRIVER_MANAGER=/usr/lib/libodbc.so.1.0.0
```

If that environment variable is not set then it looks for a library called `libodbc.so` and so you can add a symbolic link to equate that to your own library. For example as root you could run the commands:

```
# ln -s libodbc.so.1.0.0 /usr/lib/libodbc.so
# /sbin/ldconfig
```

The last step in configuring this is to ensure that Wine is set up to run the built-in version of `odbc32.dll`, by modifying the DLL configuration. This built-in DLL merely acts as a stub between the calling code and the Unix ODBC library.

If you have any problems then you can use **`WINEDEBUG=+odbc32`** command before running Wine to trace what is happening. One word of warning: some programs actually cheat a little and bypass the ODBC library. For example the Crystal Reports engine goes to the registry to check on the DSN. The fix for this is documented at `unixODBC` site where there is a section on using `unixODBC` with Wine.

Using Windows ODBC drivers

Native ODBC drivers have been reported to work for many types of databases including MSSQL and Oracle. In fact, some like MSSQL can only be accessed on Linux through a Winelib app. Rather than just copying DLL files, most ODBC drivers require a Windows-based installer to run to properly configure things such as registry keys.

In order to set up MSSQL support you will first need to download and run the **mdac_typ.exe** installer from www.microsoft.com. In order to configure your ODBC connections you must then run **CLICONFG.EXE** and **ODBCAD32.EXE** under Wine. You can find them in the `windows\system` directory after mdac_typ runs. Compare the output of these programs with the output on a native Windows machine. Some things, such as protocols, may be missing because they rely on being installed along with the operating system. If so, you may be able to copy missing functionality from an existing Windows installation as well as any registry values required. A native Windows installation configured to be used by Wine should work the same way it did when run natively.

Types successfully tested under Wine:

DB Type	Usefulness
MS SQL	100%

Please report any other successes to the wine-devel (mailto:wine-devel@winehq.org) mailing list.

Chapter 5. Troubleshooting / Reporting bugs

What to do if some program still doesn't work?

There are times when you've been trying everything, you even killed a cat at full moon and ate it with rotten garlic and foul fish while doing the Devil's Dance, yet nothing helped to make some damn program work on some Wine version. Don't despair, we're here to help you... (in other words: how much do you want to pay?)

Verify your Wine configuration

Look at the output from **`wine --version`** to make sure you're running a recent version of Wine. Launch **winecfg** and look over the settings to make sure you have settings that look normal. Look in `~/.wine/dosdevices` to make sure your `c:` points to where you think it should.

Use different Windows version settings

In several cases using different Windows version settings can help.

Use different startup paths

The following sometimes helps, too:

- `wine x:\\full\\path\\to\\prg.exe`
- `wine ~/.wine/drive_c/full/path/to/prg.exe`
- `cd ~/.wine/drive_c/full/path/to/ && wine prg.exe`

Fiddle with DLL configuration

Run with **`WINEDEBUG=+loaddll`** to figure out which DLLs are being used, and whether they're being loaded as native or built-in. Then make sure you have proper native DLL files in your configured `C:\windows\system32` directory and fiddle with DLL load order settings at command line or with **winecfg**.

Check your system environment!

Just an idea: could it be that your Wine build/execution environment is broken? Make sure that there are no problems whatsoever with the packages that Wine depends on (**gcc**, glibc, X libraries, OpenGL (!), ...)

Use different GUI (Window Manager) modes

Instruct Wine via **winecfg** to use either desktop mode, or normal managed mode. That can make a lot of difference, too.

Check your app!

Maybe your app is using some kind of copy protection? Many copy protections currently don't work on Wine. Some might work in the future, though. (the CD-ROM layer isn't really full-featured yet).

Reconfigure Wine

Sometimes Wine installation process changes and new versions of Wine account on these changes. This is especially true if your setup was created long time ago. Rename your existing `~/.wine` directory for backup purposes. Use the setup process that's recommended for your Wine distribution to create new configuration. Use information in old `~/.wine` directory as a reference. Later you can remove the new `~/.wine` directory and rename your old one back.

Check out further information

There is a really good chance that someone has already tried to do the same thing as you. You may find the following resources helpful:

- Search WineHQ Application Database (http://appdb.winehq.org) to check for any tips relating to the program. If your specific version of the program isn't listed you may find a different one contains enough information to help you out.
- Google (http://www.google.com) can be useful depending on how you use it. You may find it helpful to search Google Groups (http://groups.google.com), in particular the `comp.emulators.ms-windows.wine` group.
- Freenode.net (http://www.freenode.net) hosts an IRC channel for Wine. You can access it by using any IRC client such as Xchat. The settings you'll need are:

server	irc.freenode.net
port	6667
channel	#winehq

- If you have a program that requires a redistributable runtime to be installed, e.g. for `mfc42.dll`, Visual Basic and so on, **winetricks** can be used to supply this. Note, these components are subject to their own license and are not part of the Wine project. For further information on **winetricks** see the winetricks wiki entry (http://wiki.winehq.org/winetricks).
- Wine forums (http://forums.winehq.org/) and mailing lists (http://www.winehq.org/site/forums) may also help, especially `wine-users`. The `wine-devel` list may be appropriate depending on the type of problem you are experiencing. If you post to `wine-devel` you should be prepared to do a little work to help diagnose the problem. Read the section below to find out how to debug the source of your problem.
- If all else fails, you may wish to investigate commercial versions of Wine to see if your application is supported.

Debug it!

Finding the source of your problem is the next step to take. There is a wide spectrum of possible problems ranging from simple configurations issues to completely unimplemented functionality in Wine. The next section will describe how to file a bug report and how to begin debugging a crash. For more information on using Wine debugging facilities be sure to read the Wine Developers Guide.

How To Report A Bug

Please find all information about reporting a bug on the main page of Wine Bugzilla (http://bugs.winehq.org/).

Glossary

Binary

A file which is in machine executable, compiled form: hex data (as opposed to a source code file).

Distribution

A distribution is usually the way in which some "vendor" ships operating system CDs (usually mentioned in the context of Linux). A Linux environment can be shipped in lots of different configurations: e.g. distributions could be built to be suitable for games, scientific applications, server operation, desktop systems, etc.

DLL

A DLL (Dynamic Link Library) is a file that can be loaded and executed by programs dynamically. Basically it's an external code repository for programs. Since usually several different programs reuse the same DLL instead of having that code in their own file, this dramatically reduces required storage space. A synonym for a DLL would be "library".

Editor

An editor is usually a program to create or modify text files. There are various graphical and text mode editors available on Linux.

Examples of graphical editors are: **nedit**, **gedit**, **kedit**, **xemacs**, **gxedit**.

Examples of text mode editors are: **joe**, **ae**, **emacs**, **vim**, **vi**. In a *terminal*, simply run them via:

```
$ editorname filename
```

Environment variable

Environment variables are text definitions used in a *Shell* to store important system settings. In a **bash** shell (the most commonly used one in Linux), you can view all environment variables by executing:

```
set
```

If you want to change an environment variable, you could run:

```
export MYVARIABLE=mycontent
```

For deleting an environment variable, use:

```
unset MYVARIABLE
```

Git

Git is a fast version control system, originally written for use with large repositories, such as the Linux Kernel source. See the Git chapter in the Wine Developers Guide for detailed usage information.

Package

A package is a compressed file in a *distribution* specific format. It contains the files for a particular program you want to install. Packages are usually installed via the **dpkg** or **rpm** package managers.

root

`root` is the account name of the system administrator. In order to run programs as root, simply open a *Terminal* window, then run:

```
$ su -
```

This will prompt you for the password of the `root` user of your system, and after that you will be able to system administration tasks that require special root privileges. The `root` account is indicated by the

```
#
```

prompt, whereas '$' indicates a normal user account.

Shell

A shell is a tool to enable users to interact with the system. Usually shells are text based and command line oriented. Examples of popular shells include **bash**, **tcsh** and **ksh**. Wine assumes that for Wine installation tasks, you use **bash**, since this is the most popular shell on Linux. Shells are usually run in a *Terminal* window.

Source code

Source code is the code that a program consists of before the program is being compiled, i.e. it's the original building instructions of a program that tell a compiler what the program should look like once it's been compiled to a *Binary*.

Terminal

A terminal window is usually a graphical window that one uses to execute a *Shell*. If Wine asks you to open a terminal, then you usually need to click on an icon on your desktop that usually shows a big black window. Wine assumes you're using the **bash** shell, so if your terminal happens to use a different shell program, simply type **`bash`** in the terminal window.