Windows under Linux

FULL BODIED AND MATURING NICELY

PETER GANTEN

Imagine if Windows programs could be used on any Linux system. Imagine too, that this could be achieved without the need for a cumbersome emulator or a Windows operating system software license. Well, with Wine, it's possible.

Thanks to the Wine project it is possible to run Windows programs under Linux without a need to pay money to Microsoft. The snag is, so far not all Windows programs will run. But the developers of Wine have made enormous progress and the number of Windows programs that can be used under Linux with Wine is increasing all the time. In this article you'll find out what Wine really is, how it works and how you can install Wine on your system in order to use Windows programs.

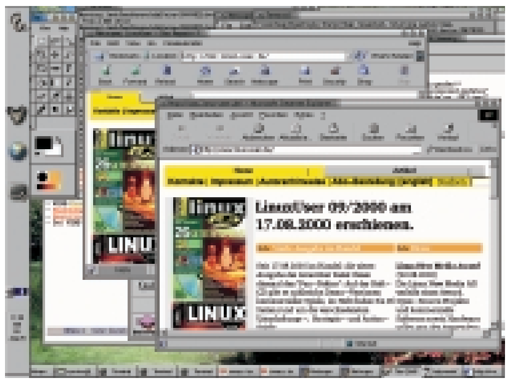
EXE files?

The fact that any **i386**-compatible processor can execute Windows programs doesn't mean that such programs will automatically run under Linux on Intel. Something more is needed so that Windows programs can be loaded from the hard drive into working memory and then executed. This task is performed by the *program loader*. When you start Windows programs under Windows (perhaps by selecting a program in the Start menu) this function is carried out by the Windows operating system.

Linux has similar functionality by which native Linux programs can be loaded when, for example, you call up an application in the KDE menu, from the shell or using the GNOME panel. If a user tries to start a program the operating system first checks whether the corresponding program file is on the hard disk and in the correct file format. Just as Netscape cannot show StarOffice files, Linux cannot simply load Windows programs. If it appears that the file to be executed has an unknown format, the operating system interrupts and emits an error message. Therefore, in order to start Windows programs under Linux, a special Windows program loader is required.

API differences

Then there is another problem: each operating system makes available certain functions to be used by programs that run under it in order to open files,



LinuxUser 09/2000 am 17.08.2000 erschienen.

Fig. 1: Handy for the development of web pages. The two rivals, Netscape and Internet Explorer side by side displaying the homepage of one of the sister of Linux Magazine. Wine makes it possible

i386: *Designation of the processor architecture developed by the chip manufacturer Intel. Among Intel 386-compatible processors are Intel's 80386, 80486, 80586 (Pentium), 80686 (Pentium II) processors, but also processors from other CPU manufacturers such as AMD's Athlon or VIA's Cyrix III. The i386 architecture dominates the domain of the desktop computer. When i386-compatible computers are mentioned, this often also means PCs or IBM-compatible computers.*

API: *Application Programmer's Interface. This refers to the interfaces of an operating system, a system component or a program library which can be used by other programs.*

Program library: *This is a file containing program code which can be executed by the processor. It is not in itself a complete program. In general, program libraries lack the so-called main() function, which is called up by the operating system in order to start a program. The code in program libraries can however be used by executable programs. A program library can for example provide functions to display windows on the screen. Programs that use this library do not need to contain the corresponding code themselves, which saves memory space. This also ensures a uniform appearance for all windows of programs which use the library. Program libraries under Windows often have the filename suffix .DLL (Dynamic Link Library). The equivalent under Unix/Linux have names that end in .so (shared object).*

display things on the screen, receive data from the Internet and so on. These interfaces are referred to as the **API** of the operating system. APIs, and the way in which they are used, differ considerably between Windows and Linux. The way APIs are used under Windows can best be explained by means of an example.

In order to open or create a file the API *CreateFile()* is used. This is a function located in a **program library**. A Windows program that uses this function has to load the corresponding program library (in this case, the library *KERNEL32.DLL*). In this way the function call of the program is linked with the function in the library. When the function *CreateFile()* is called up, control is handed over to the library. Depending on which version of Windows (NT or 95/98) is involved, some very different functions can be used by the library in order to execute the required operation (i.e. the opening or creation of a file).

If a Windows program was loaded into memory under Linux and then executed, it would almost certainly fail. This would be because functions such as *CreateFile()* are not available. The Linux kernel provides a similar function, in this case one known as *open()*, but it is called up in a completely different way. In the realm of computer programs, similar is not good enough.

How can this problem be solved? You may have already guessed. The necessary APIs have to be reproduced under Linux. They can then be linked with the program to be executed, just as occurs under Windows. If, for example, the program running under Linux invokes the Windows *CreateFile()* API, the library is called, and calls in turn the corresponding Linux system calls. Any result returned by the Linux call is transformed if necessary into the form expected by the Windows program.

This perhaps seems complicated, but in practice it doesn't have any disadvantages in comparison with the "real" Windows. To stay with the example, under DOS-based Windows versions such as Windows 95/98, *CreateFile()* under certain conditions calls DOS routines in order to actually open a file. Under Windows NT or Windows 2000 the corresponding NT API (in this instance *NtCreateFile()*) is called up from *CreateFile()*. Even under the "real" Windows, more and more layers have to be run through. Exactly the same happens under Linux. In fact, where Linux performs a function more efficiently than Windows, a Windows program running under Linux may, despite the overhead of Wine, still execute more efficiently than under Windows.

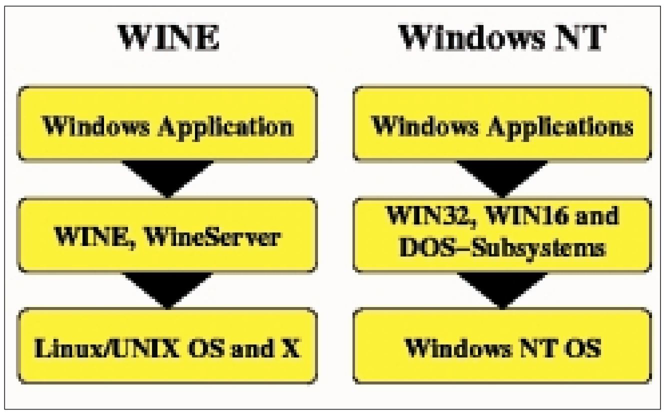


Fig. 2: Windows programs under Wine and Windows NT in comparison

Safer than the original

So what precisely does Wine contain? For a start, it has a program loader for Windows programs. With this, 32-bit and 16-bit Windows programs (and also DOS programs) can be loaded into the working memory and executed. This is only a small (if very important) part of its functionality, however. Most of Wine consists of the program code which makes available the APIs that DOS and Windows programs expect to find. These are located, as under Windows, in special libraries, which are linked by the loader or at run time with the Windows program to be executed.

Wine is an ordinary Linux program from the point of view of the Linux kernel. It doesn't even require special rights to be executed. Windows programs can thus be more safely executed under Wine than for example under Windows 98, where each program has full access rights to the whole computer and all files.

The acronym Wine, by the way, stands for "Wine Is Not an Emulator". This is intended to point out that Wine doesn't emulate a Windows computer. Instead, Wine executes Windows programs directly, precisely as happens under Windows. Nevertheless Wine does also mean WINdows Emulator, because the Windows APIs it provides do not contain the same code as the "real" APIs written by Microsoft. Wine could be said to emulate Windows APIs. However, the word *re-implementation* would perhaps be more appropriate.

Before getting started on the installation and configuration of Wine, one more word about the current state of its development. The project is at present still in the alpha stage, i.e. in the middle of development. Many Windows programs do in fact already run very stably with Wine. Unfortunately, many others still don't run at all. Because Wine is still being worked on very intensively, it could also happen that a program which functioned well with one version of Wine no longer runs with a more recent version. In such cases it is a good idea to send a bug report to the Wine newsgroup on the Internet and wait for the next version of the program package.

Installation

Almost all Linux distributions now include Wine packages which can be installed via the distribution package management program. Due to the rapid development of Wine, however, these packages are already obsolete by the time the distribution comes out. So it can pay to download a current Wine package from the Internet and install this.

If you do this, you have the option of either installing a binary package in RPM or Debian format, or of using the Wine source code and compiling this on your own system. This sounds more complicated than it really is. Compilation on your own computer has the advantage that the Wine program thus created is precisely tailored to the software of your own computer. When this happens a check is made first as to which components are present in the system. Wine can then include properties that use these components. So for example the OpenGL components of Wine only function when an OpenGL library with specific properties is on the system. If you are using a binary package containing a version of Wine which has been compiled to support a specific **OpenGL** version, but the requisite OpenGL library has not been installed on your system, the result may be that Wine will not run. Conversely, you cannot use the OpenGL functionality unless your version of Wine has been compiled for use by OpenGL, even if an OpenGL library is present in your system. In order to use the latest version of Wine it is recommended that you take a source code package and compile it yourself.

Software requirements

In order to compile Wine and then use it certain programs and files must be installed on the system. These files are all an integral part of modern distributions, but this doesn't necessarily mean they are currently installed. To prevent any problems arising during compilation you should check whether the following components are already on your system:

- Linux kernel version 2.2.x. Wine also functions with older kernels (version 2.0.x.) However, there can be problems with these kernels when 32-bit programs with several threads are executed.
- The GNU C run time library. The recommended version here is 2.1. You can also use Wine with the older version 2.0. It is not advisable to use the now-obsolete C library *libc5*. Apart from the actual library, which is installed on every Linux system, you also need the development files for it. These files are included under Debian 2.2 in the package *libc6-dev* and under Suse 7.0 in the package *libc* (Series d). Also, under Suse the header files of the kernel have to be installed as well, and these are in the *linclude* package, Series d.
- Wine normally uses the X Window system to display windows on the screen, so X has to be installed. You will also need the X developer files (Debian: *xlib6g-dev* package, Suse: *xdevel* package, Series x)
- The X Pixmap library (*libxpm*) will also be required. Under Debian you will have to install the packages *xpm4g* and *xpm4g-dev*. Under Suse the corresponding packages are included in the packages *shlibs* and *xdevel*.
- In order for Wine to be compiled, you will of course need a compiler. The minimum requirement in this case is the GNU C Compiler, version 2.7.2. The latest version, 2.95 is recommended. You can find these compilers both under Debian, Suse and doubtless most other distributions in the *gcc* package.
- Plus, you'll need a few ancillary tools such as *make*, *bison* and *flex*, which are found under Debian and Suse in packages with the corresponding names.
- Wine can optionally use a few other libraries. These primarily include the *ncurses* library (Debian: *libncurses5* and *libncurses5-dev* packages. Suse: *ncurses*, Series a) and OpenGL libraries. You'll find developer files for OpenGL under Debian in the package *mesag-dev* and under Suse in the package *mesadev* (Series x3d). Please note that you will also need either an OpenGL graphics card and an X-Server with OpenGL support for your card or the (slow) OpenGL software implementation Mesa (Debian: *mesag3*. Suse: *mesa* and *mesasoft*, Series x3d).

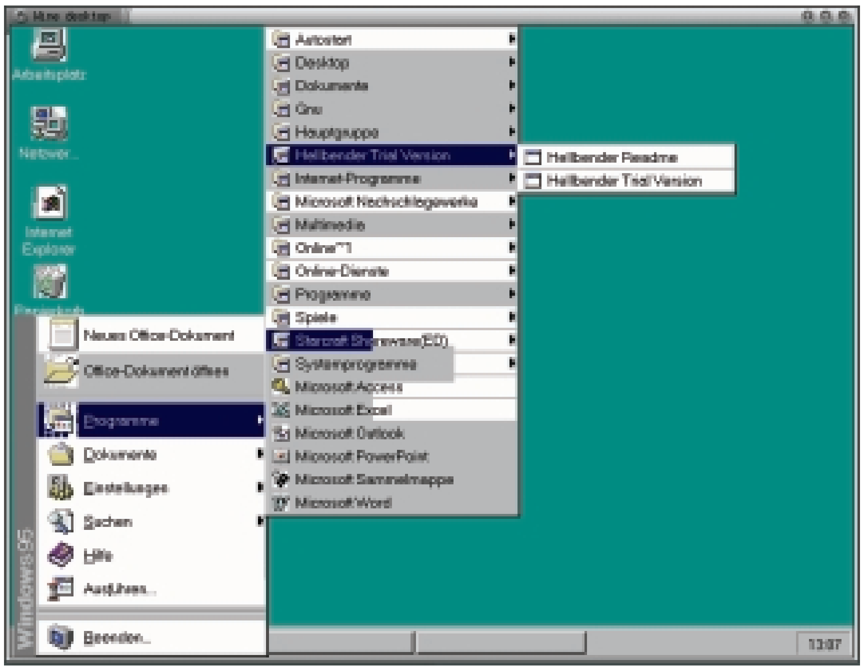
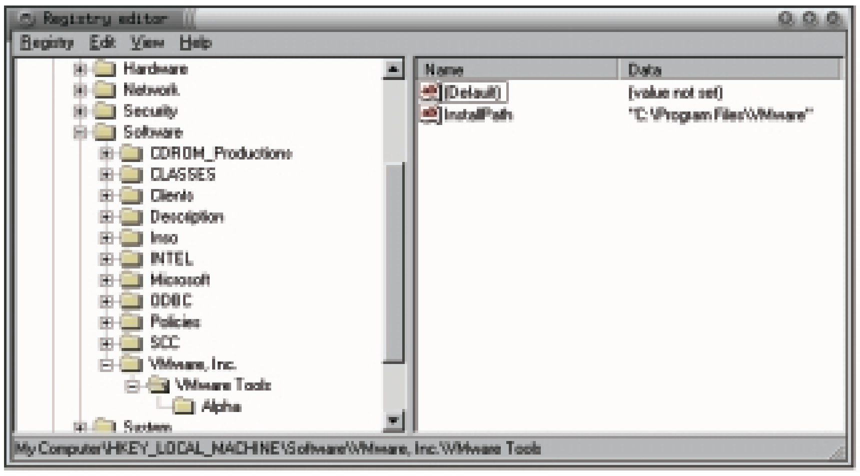


Fig. 3: Still room for improvement: Explorer from the German version of Windows 95 in the window

OpenGL: *OpenGL is a collection of elementary graphics functions which, unlike other graphics libraries (e.g. Direct3D under Microsoft Windows) has the enormous advantage of being not only hardware-independent, but also suitable for any platform (and also for any operating system!) You can find out more at http://www.opengl. org/.*

■



(Default) (value not set)
InstallPath "C:\Program Files\VMware"

My Computer\HKEY_LOCAL_MACHINE\Software\VMware, Inc.\VMware Tools

Fig. 5: Wine provides a Windows registry under Linux, which can of course be edited using Windows tools

Debug: *The process of debugging: removing bugs (not unwanted insects, but faults) from the program code.*

Creating and installing the source code

The current Wine source code can be downloaded from the Internet from one of the following addresses:

- *ftp://metalab.unc.edu/pub/Linux/ALPHA/wine/*
- *ftp://ftp.infomagic.com/pub/mirrors/linux/sunsite /ALPHA/wine/development/*
- *http://metalab.unc.edu/pub/Linux/ALPHA/wine/ development/*.

You will find compressed *tar* archives in directories with names consisting of the designation *Wine-*, the date on which the respective version was published and the ending *.tar.gz*. Thus the file *Wine-20000821.tar.gz* for example contains the Wine source code for the version of 21 August 2000. Normally you should use the latest applicable version.

Once you have downloaded the tar archive and stored it in your home directory you can unpack it with the following command:

```
$ tar -xvzf Wine-20000821.tar.gz
```

You must obviously adjust the designation of the file name (in this case, *Wine-20000821.tar.gz*), if you are using another version.

Compilation and installation of Wine

As soon as the archive has been unpacked, the source code can be compiled and Wine can then be installed. To do this first change to the directory in which Wine was unpacked. The name of this directory is made up from the character string *wine-* and the date of the Wine version, so for example it could be called *wine-20000821*.

```
$ cd wine-20000821
```

Then enter this command in order to configure the source code for your system:

```
$ ./configure
```

All being well, after a series of messages, the following text will appear:

```
Configure finished.  Do 'make depend && mak↵
e' to compile Wine.
```

Should this not be the case it is probably due the fact that certain files have not been found. A corresponding error message should have been issued. You will have to reinstall the missing package and try again. When all is well you can continue following the instructions and enter this command:

```
$ make depend && make
```

Tip: if you have little space free on your hard disk and wish to use Wine but not to **debug** it, you can omit the debug information in the binary files created during compilation. In that case you should use this command to compile the code:

```
$ make depend && make CFLAGS=-O2
```

Now the program can be installed so that it can be used by all users of the system. This must be carried out with the rights of the administrator, *root*. If necessary acquire these rights by typing:

```
$ su
```

and entering the *root* password. Then install the newly-compiled Wine using:

```
# make install
```

If you don't wish to make Wine available for the whole system you can bypass this step.

Configuration

Configuring Wine is relatively time-consuming in comparison with other software packages. This is mainly due to the fact that Windows programs expect a certain infrastructure, which of course is already in place on a Windows system. Under Linux, however, it must first be created.

For example, configuration data and other information is stored under Windows in a system file called the *Registry*, which of course has to be made available by Wine. There is also a considerable difference between Unix/Linux and Windows in that under Windows, drive letters are used to designate different storage devices. If a Windows program wants to open, for example, the file *C:\My Documents\Letter.doc*, Wine has to convert this filename into a valid Linux filename and open the appropriate file. To do this, a configuration file is used in which, among other things, Windows drive letters are assigned to directories under Linux. So for example the *C:* drive could be assigned to a directory called */c* so that the file */c/My Documents/Letter.doc* will be opened by accessing *C:\My Documents\Letter.doc*.

There are basically two different ways of configuring Wine: with and without an existing Windows installation. If Windows is installed on your computer (for example in a dual-boot configuration, where the Windows partition can be mounted under

Linux), Wine can share this installation. Then all settings and files from this existing Windows installation are taken over. This has the advantage that programs installed under Windows can be executed under Linux with Wine without having to be reinstalled. Wine can then make use of a whole range of libraries in the Windows installation. Since at present the number of Windows libraries emulated (or re-implemented) under Wine is still very much a subset of those commonly used, many Windows programs run better under Wine if the "proper" libraries are available.

There is no risk of Wine changing Windows configuration files. Changes to the *registry* or to *INI* files which are made by the programs running under Wine are stored separately in the home directory of the user concerned and not written back to the Windows installation. This can lead to inconsistencies if you install programs with Wine and files in the Windows installation are overwritten.

Of course, the partition containing the Windows installation has to be available under Linux so that Wine can access it. If this is not yet the case on your system you should make a corresponding entry in the file */etc/fstab*. Assuming that your Windows installation is located on the first primary partition of your first IDE hard disk and it is to be mounted on the */c* directory, the */c* directory should be created first, using:

```
# mkdir /c
```

Then the following line should be entered in the file */etc/fstab* :

```
/dev/hda1  /c  vfat  defaults
```

If you wish to exclude completely the possibility that Wine might change anything on your Windows installation you can also mount the partition as read-only:

```
/dev/hda1  /c  vfat  defaults,ro
```

Now the partition still has to be actually mounted. This can be done with this command:

```
# mount /c
```

You can, of course, also use Wine without any existing Windows installation. You must then define at least one directory, which should be assigned to a drive letter, and create this directory if it doesn't yet exist. Also, a few sub-directories must be created in the directory – these are already in place in a Windows installation – together with a Registry. The simplest way to perform this step, together with the creation of a Wine configuration file, is using the script *wineinstall*, which can be found in the *tools* subdirectory of the source code directory.

For all seasons: *wineinstall*

Wineinstall can be used to configure Wine for use with an existing Windows installation. The script can tell whether one exists by checking whether a Windows partition is mounted in the file */etc/fstab*. So if you have mounted a Windows installation but don't wish it to be used by Wine it is advisable to comment out the entry before you run *wineinstall*. You can then later remove the comment.

Depending on whether you run *wineinstall* as *root* (system administrator) or as an ordinary user, the script will create a configuration which is valid for the whole system or one that is merely valid for the user who is running it. In the case of a configuration which is valid for the whole system the configuration data is stored in the */usr/local/etc* directory. The most important file in this directory is the *wine.conf* file. This is the central configuration file for Wine. In the case of a user configuration the configuration file is written in the home directory (~) of the user who is running it, where it bears the name *.winerc* (The dot before the file name, by the way, means that this is a "hidden" file which is only displayed by *ls* when the additional parameter *-a* is provided to it.) If both files exist, Wine will use the file in the home directory of the user who is running it. So you should enter the following command to configure Wine, either as user or as administrator:

```
# tools/wineinstall
```

A check will then be made as to whether Wine has been compiled and installed. If not, the required steps will be taken. After that there will be a check as to whether an existing Windows installation is in place. If so, the required configuration file will be created. If not, you will be asked which directory should correspond to the *C:* drive. This directory will be created if it doesn't yet exist. The registry will also be created. After that a configuration file will be written.

The big moment

Now you can try to execute a simple Windows program in order to test Wine. If you are using an existing Windows installation try starting the program *notepad.exe*. If not, you can start with a test by going straight into the installation of a program, as described further on.

To start Windows programs with Wine from the command line, the name of the Windows program to be run must be specified as an argument to the *wine* program. So to run *notepad.exe* the following command should be entered:

```
$ wine notepad.exe
```

You can also provide a full path to the program to be run. This is necessary if it cannot be found in the search path for Windows programs, which is specified in the configuration file. When you do this you can specify either the Unix path name or the Windows path name, depending on the assignments in the configuration file. So if the */c* directory is assigned to the *C:* drive, both of the following commands would have the same effect:

```
$ wine /c/windows/notepad.exe
$ wine c:\\windows\\notepad.exe
```

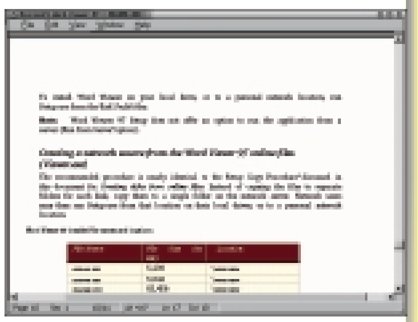


Fig. 5:
Handy for viewing Word documents under Linux: the free Word Viewer from Microsoft.

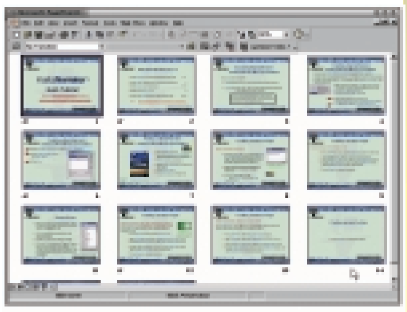
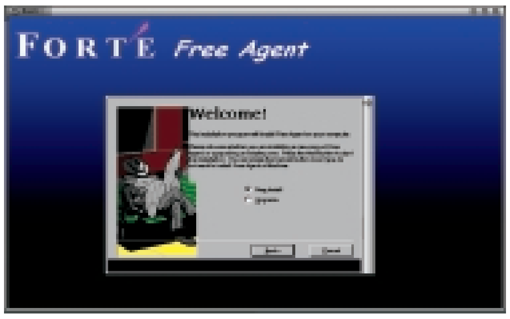
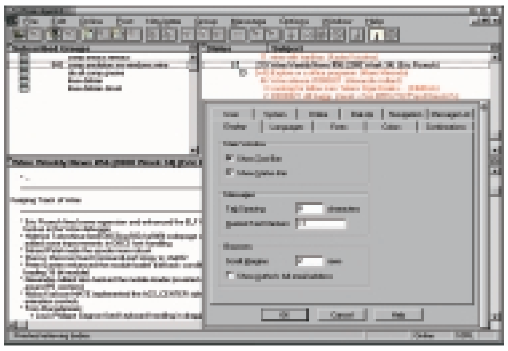


Fig. 6:
Another possibility: creating a presentation via Wine with PowerPoint



[top] Fig. 7: Installing a Windows program under Linux with Wine. Does this look familiar to you?



[above] Fig. 8: Eh voila! Wine repays us for all this effort with a ”managed” newsreader

Note that the backslash was specified twice in each case in the second command because it has a special significance for the Linux shell. No matter how you specify the file name of the program to be executed, one thing must always be noted: each Windows program to be executed has to be in a directory to which a path can be formed starting from one of the directories assigned to a drive letter. These assignments are made in the configuration file, and have the following format:

```
[Drive C]
Path=/c
Type=hd
Label=MS-DOS
Filesystem=win95
```

What matters here in particular is the designation in the square brackets, (by which the name of the drive is defined) and the value following the *Path* designator which defines the Unix directory the drive corresponds to. The *Filesystem* designator should in most cases be followed by *win95* , regardless of which file system is actually being used on that drive.

If you would like to have the whole file system of your computer available to programs running under Wine, simply define a drive corresponding to the root directory of your system:

```
[Drive R]
Path=/
Type=hd
Label=ROOT
Filesystem=win95
```

If Wine is used with an existing Windows installation, the drive assignments under Wine should be kept the same as under Windows. Otherwise there can be problems. If a program, when running under Windows, searches on the C: drive for a file but under Wine the file is found on the *D:* drive, for example, you are likely to receive an error message. You can of course define additional drives under Wine, perhaps to be able to address the root directory of your system or your own home directory.

Command line options

If you have already tested Wine as described you will probably have noticed that windows controlled by Wine look like Windows windows rather than native Linux windows and work independently of the Linux window manager. On the whole it is better for Windows programs running under Wine to appear as normal Linux applications and to be controlled like all other windows via the window manager. The command line option *--managed* is used to specify this preference this. So if Notepad is to run ”managed”, Wine would be invoked as follows:

```
$ wine --managed notepad.exe
```

Wine can also be operated in ”desktop mode.” The windows of Windows programs are then, as under *VMware*, all shown together in one window. The command line option for this is *--desktop*. If desired the size of the desktop window to be displayed can be specified. In order to execute *notepad.exe* in a desktop window with a size of 640 x 480 dots, Wine would be invoked as follows:

```
$ wine --desktop 640x480 notepad.exe
```

Many Windows programs run under Windows 95 but not under NT, while others may have the opposite preference. Using the command line option *--winver* you can inform Wine as to which version of Windows it should pretend to be In order to make it appear to a Windows program that it is being executed under Windows NT 4.0, Wine would be run as follows:

```
$ wine --winver nt40 notepad.exe
```

To ”fake” Windows 3.1 the value *win31* would be used. Other possible arguments are *win95* and *win98*. If Wine is run using the command line option *--help* then, like many Linux programs it presents a list of all valid options.

Another important option is *--dll*. With this it is possible to specify which libraries Wine should use from an existing Windows installation and which it should provide itself. To do this, the option should be followed by the name(s) of the libraries (without the *.dll*) for which the default setting should be overridden; where several libraries are specified their names should be separated using commas. An equals sign can be used to specify whether the version provided by Wine (*b* for built-in) or the Windows version (*n* for native) is to be used. In order, for example, to use the Windows versions of the *shell* and *shell32* libraries, you could invoke Wine like this:

```
$ wine --dll shell,shell32=n notepad.exe
```

Info

The Wine Project
http://www.winehq.com/

The option *--dll* offers many more options, which are described in the Wine man pages. To alter the default settings and avoid the need to continually type lengthy command lines the configuration file can be modified.

Installing Windows programs under Wine

If you don't have a Windows installation available to you, Windows programs which you want to use under Linux must be installed under Wine. In principle that's no problem, because installation programs are of course nothing more than ordinary Windows programs. In practice, however, it often turns out that the installation programs do not work under Wine even though the actual programs themselves, once installed, are perfectly usable. If you cannot install a program under Wine you should either wait for a new version of Wine (as mentioned, development is making rapid progress) or read the Wine documentation and then write up a bug report.

To demonstrate the installation of Windows programs under Wine we will use the newsreader Free Agent (that's the freeware version from Forte Agent) which is popular among Windows users,. It can be obtained from *http://www.forteinc.com/getfa/download.htm*. Download the file *fa32-121.exe* from this website. This is the 32-bit version of the program. There is also a 16-bit version of the program available which can also be used.

Before attempting to install the program you must first ensure that Wine has write access to the Windows directory. If necessary, Wine should be executed by the administrator *root*. Then Wine can be run with the name of the downloaded file as an argument, like this:

```
# wine fa32-121.exe --managed
```

The installation program should now start. Don't worry about the minor graphical errors which can sometimes arise at this point. Respond to all requests from the program just as you would under Windows. The installation program creates among other things an entry in the Windows Start menu. There is currently no link between these entries and the start menus of GNOME or KDE. After installation you must therefore start Free Agent from the command line, although you can of course create a GNOME or KDE menu entry manually. If you have installed the program in the directory *C:\Program Files\Agent* and the drive letter *C:* corresponds to the Unix directory */c*, you could start the newsreader using the following command line:

```
$ wine /c/Program\ Files/Agent/agent.exe--ma➥
naged
```

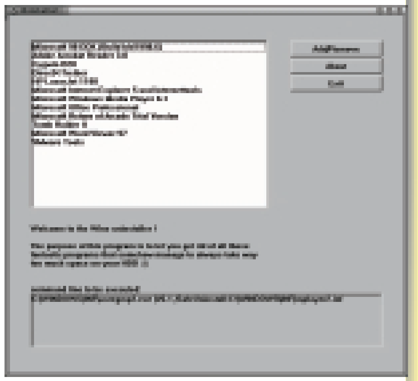
(The backslash after *Program* is necessary here to inform the shell that the space which follows it is part of the file name.) This command starts the program, which ought to be as usable as it is under Windows.

If you've made it this far, you can now try to execute the programs in your existing Windows installation with Wine – just start Internet Explorer! If you have no Windows installation on your computer, try installing some other programs. But bear in mind that Wine is still very much under development and consequently is not completely stable. If your program hangs, you can terminate it using the command:

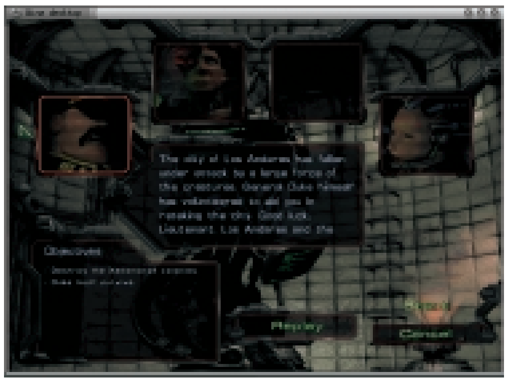
```
$ killall -9 wine
```

Further information

This article can only give a glimpse of the possibilities of Wine. You'll find additional information in the documentation directory of the Wine source code, where there is also a comprehensive set of installation instructions. We'd also recommend a visit to the Wine Project's home page. There you'll find links to the Wine FAQ and a Wine HOWTO. Online support can be found in the Wine newsgroup, which goes by the name of *comp.emulators.ms-windows.wine*. ■

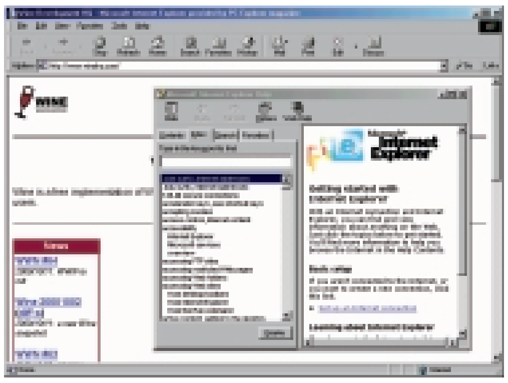


[right] Fig. 9:
The Wine uninstaller is found in the *programs* subdirectory of Wine The programs installed under Wine can be managed with this.



[left] Fig. 10:
The game StarCraft shows off the multimedia properties of Wine. Get the StarCraft demo from *http://www.blizzard.com/starcraft/scdemo.shtml*.

[top] Fig. 11:
Wing Commander is a popular game that can be played under Wine. Obtained a demo from *http://www.wingcommanderprophecy.com/demo.html*



[above] Fig. 12:
Can you tell the difference? Internet Explorer with the homepage of the Wine project and HTML help